

结构之法 算法之道

面试、算法、机器学习在线课程：julyedu.com

目录视图

摘要视图

RSS 订阅

个人资料



v_JULY_v

访问：11239113次
积分：46749
等级：

BLOG > 8

排名：第47名

原创：151篇 转载：0篇
译文：5篇 评论：13271条

博主简介

July，于2010年10月11日开始在CSDN上写博（搜索：“结构之法”，进入本博客），博客专注面试、算法、机器学习。2015年正式创业，七月在线科技创始人兼CEO，公司官网：七月在线（<https://www.julyedu.com/>），微博@研究者July。新书《编程之法》15年10月14日起正式上市。JulyEdu c++/算法Q群：123531805。July，2016/5月。

我的微博

【CSDN会员专属福利】OpenStack Days China 大会门票，先到先得

【收藏】Html5 精品资源汇集

我们为什么选择Java

从K近邻算法、距离度量谈到KD树、SIFT+BBF算法

2012-11-20 16:31 129416人阅读 评论(129) 收藏 举报

分类：

24.data structures (11)

30.Machine L & Deep Learning (11)

目录(?)

[+]

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内的文章与你于别处所见的任何都不同。于是，等啊等，等一台电脑，只好等待..”。得益于田，借了我一台电脑（借他电脑的时候，我连表示感谢，他说“能找到工作全靠你的博客，这点儿小忙还说，不地道”，有的时候，稍许感受到受人信任也是一种压力，愿我不辜负大家对我的信任），于是今天开始[Top 10 Algorithms in Data Mining](#)系列第三篇文章，即本文「从K近邻算法谈到KD树、SIFT+BBF算法」的创作。

一个人坚持自己的兴趣是比较难的，因为太多的人太容易为外界所动了，而尤其当你无法从中得到多少实际性的回报时，所幸，我能一直坚持下来。毕达哥拉斯学派有句名言：“万物皆数”，最近读完「微积分概念发展史」后也感受到了这一点。同时，从算法到数据挖掘、[机器学习](#)，再到数学，其中每一个领域任何一个细节都值得探索终生，或许，这就是“终生为学”的意思。

本文各部分内容分布如下：

1. 第一部分讲K近邻算法，其中重点阐述了相关的距离度量表示法，
2. 第二部分着重讲K近邻算法的实现--KD树，和KD树的插入，删除，最近邻查找等操作，及KD树的一系列相关改进(包括BBF，M树等)；
3. 第三部分讲KD树的应用：SIFT+kd_BBF搜索算法。

同时，你将看到，K近邻算法同本系列的前两篇文章所讲的决策树分类贝叶斯分类，及支持向量机SVM一样，也是用于解决分类问题的算法，

http://blog.csdn.net/v_july_v/article/details/8203674

Page 1 of 47



研究者July

加关注

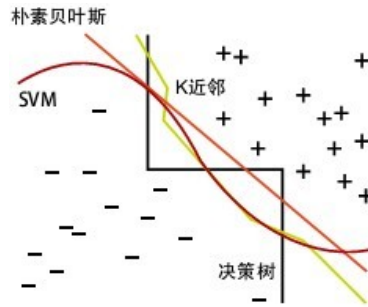
吼吼，不止讲师队伍在不断壮大，学员之间更是互帮互助、互相答疑、讨论，每个人都可以从其他人身上学到不少知识。所有人向所有人学习，共建在线教育。//@怒放的生命0119: 2年之内参加了3个班，切身体会到了七月在线实力水平的越来越强！祝越来越好

July新书《编程之法》上市

京东 当当 天猫 Amazon
异步社区 互动出版网

文章分类

- 03.Algorithms (实现) (9)
- 01.Algorithms (研究) (27)
- 02.Algorithms (后续) (22)
- 04.Algorithms (讨论) (1)
- 05.MS 100' original (7)
- 06.MS 100' answers (13)
- 07.MS 100' classify (4)
- 08.MS 100' one Keys (6)
- 09.MS 100' follow-up (4)
- 10.MS 100' comments (4)
- 11.TAOPP (编程艺术) (34)
- 12.TAOPP string (8)
- 13.TAOPP array (14)
- 14.TAOPP list (2)
- 15.stack/heap/queue (0)
- 16.TAOPP tree (2)
- 17.TAOPP c/c++ (2)
- 18.TAOPP function (2)
- 19.TAOPP algorithms (8)
- 20.number operations (1)
- 21.Essays (7)
- 22.Big Data Processing (5)



而本数据挖掘十大算法系列也会按照分类，聚类，关联分析，预测回归等问题依次展开阐述。

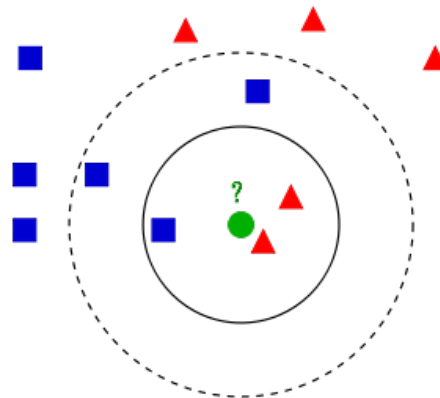
OK，行文仓促，本文若有任何漏洞，问题或者错误，欢迎朋友们随时不吝指正，各位的批评也是我继续写下去的动力之一。感谢。

第一部分、K近邻算法

1.1、什么是K近邻算法

何谓K近邻算法，即K-Nearest Neighbor algorithm，简称KNN算法，单从名字来猜想，可以简单粗暴的认为是：K个最近的邻居，当K=1时，算法便成了最近邻算法，即寻找最近的那个邻居。为何要找邻居？打个比方来说，假设你来到一个陌生的村庄，现在你要找到与你有着相似特征的人群融入他们，所谓入伙。

用官方的话来说，所谓K近邻算法，即是给定一个训练数据集，对新的输入实例，在训练数据集中找到与该实例最邻近的K个实例（也就是上面所说的K个邻居），这K个实例的多数属于某个类，就把该输入实例分类到这个类中。根据这个说法，咱们来看下引自维基百科上的一幅图：



如上图所示，有两类不同的样本数据，分别用蓝色的小正方形和红色的小三角形表示，而图正中间的那个绿色的圆所标示的数据则是待分类的数据。也就是说，现在，我们不知道中间那个绿色的数据是从属于哪一类（蓝色小正方形or红色小三角形），下面，我们就要解决这个问题：给这个绿色的圆分类。

我们常说，物以类聚，人以群分，判别一个人是一个什么样品质特征的人，常常可以从他/她身边的朋友入手，所谓观其友，而识其人。我们不是要判别上图中那个绿色的圆是属于哪一类数据么，好说，从它的邻居下手。但一次性看多少个邻居呢？从上图中，你还能看到：

- 如果K=3，绿色圆点的最近的3个邻居是2个红色小三角形和1个蓝色小正方形，少数从属于多数，基于统计的方法，判定绿色的这个待分类点属于红色的三角形一类。
- 如果K=5，绿色圆点的最近的5个邻居是2个红色三角形和3个蓝色的正方形，还是少数从属于多数，基于统计的方法，判定绿色的这个待分类点属于蓝色的正方形一类。

23.Redis/MongoDB (0)
 24.data structures (12)
 25.Red-black tree (7)
 26.Image Processing (3)
 27.Architecture design (4)
 28.Source analysis (3)
 29.Recommend&Search (4)
 30.Machine L & Deep Learning (12)

博客专栏



数据挖掘十大算法系列

文章：11篇
 阅读：1397209



微软面试100题系列

文章：18篇
 阅读：2871449



程序员编程艺术

文章：32篇
 阅读：2203810



经典算法研究

文章：32篇
 阅读：2831394

文章搜索

阅读排行

支持向量机通俗导论（理
(489464)
 程序员面试、算法研究、
(480124)
 教你如何迅速秒杀掉：95
(433255)
 从B树、B+树、B*树谈到
(349043)
 九月十月百度人搜，阿里
(268618)
 十道海量数据处理面试题
(266694)
 十一、从头到尾解析Has
(225882)
 从头到尾彻底理解KMP
(217841)
 横空出世，席卷互联网--
(211695)
 教你初步了解红黑树
(189989)

评论排行

程序员面试、算法研究、 (495)

于此我们看到，当无法判定当前待分类点是从属于已知分类中的哪一类时，我们可以依据统计学的理论看它所处的位置特征，衡量它周围邻居的权重，而把它归为(或分配)到权重更大的那一类。这就是K近邻算法的核心思想。

1.2、近邻的距离度量表示法

上文第一节，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。但有的读者可能就有疑问了，我是要找邻居，找相似性，怎么又跟距离扯上关系了？

这是因为特征空间中两个实例点的距离可以反应出两个实例点之间的相似性程度。K近邻模型的特征空间一般是n维实数向量空间，使用的距离可以使欧式距离，也可以是其它距离，既然扯到了距离，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。

- 1. 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 $x = (x_1, \dots, x_n)$ 和 $y = (y_1, \dots, y_n)$ 之间的距离为：

$$d(x, y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

(1)二维平面上两点 $a(x_1, y_1)$ 与 $b(x_2, y_2)$ 间的欧氏距离：

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

(2)三维空间两点 $a(x_1, y_1, z_1)$ 与 $b(x_2, y_2, z_2)$ 间的欧氏距离：

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

(3)两个n维向量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的欧氏距离：

$$d_{12} = \sqrt{\sum_{k=1}^n (x_{1k} - x_{2k})^2}$$

也可以用表示成向量运算的形式：

$$d_{12} = \sqrt{(a - b)(a - b)^T}$$

其上，二维平面上两点欧式距离，代码可以如下编写：

```
[cpp] view plain copy print ?
//unixfy: 计算欧氏距离
double euclideanDistance(const vector<double>& v1, const vector<double>& v2)
{
    assert(v1.size() == v2.size());
    double ret = 0.0;
    for (vector<double>::size_type i = 0; i != v1.size(); ++i)
    {
        ret += (v1[i] - v2[i]) * (v1[i] - v2[i]);
    }
    return sqrt(ret);
}
```

- 2. 曼哈顿距离，我们可以定义曼哈顿距离的正式意义为L1-距离或城市区块距离，也就是在欧几里得空间的固定直角坐标系上两点所形成的线段对轴产生的投影的距离总和。例如在平面上，坐标 (x_1, y_1) 的点P1与坐标 (x_2, y_2) 的点P2的曼哈顿距离为： $|x_1 - x_2| + |y_1 - y_2|$ ，要注意的是，曼哈顿距离依赖座标系统的转度，而非系统在座标轴上的平移或映射。

支持向量机通俗导论 (理 (487)
 九月十月百度人搜, 阿里 (390)
 从头到尾彻底理解KMP (357)
 九月腾讯, 创新工场, 淘 (351)
 从B树、B+树、B*树谈到 (351)
 当今世界最为经典的十大 (348)
 横空出世, 席卷互联网-- (327)
 教你如何迅速秒杀掉: 9s (292)
 程序员编程艺术: 第一章 (281)
 (256)

最新评论

CNN笔记: 通俗理解卷积神经网络
 Eyecools: 好文章, 膜拜一下

CNN笔记: 通俗理解卷积神经网络
 Eyecools: 好文章, 膜拜一下

红黑树从头至尾插入和删除结点!
 zourun: 请问删到全剩黑色节点的时候, 再去删除13这个节点的时候怎么删除的, 看你代码好像和这个过程不一样啊

B树的C实现
 阳光梦: 您好, 测试代码开源下吧。

CNN笔记: 通俗理解卷积神经网络
 v_JULY_v: @Mark_LQ: 你好, 非常感谢您的提醒, 之前的图确实有问题, 所以刚刚根据cs231n的卷积动图依次...

CNN笔记: 通俗理解卷积神经网络
 SunnyMarkLiu: 参考文献与延伸阅读还是值得一读的

CNN笔记: 通俗理解卷积神经网络
 SunnyMarkLiu: "cs231d的动态卷积图"是有问题的。。。? Input Volume输入层的数据是固定的, 怎么一直...

CNN笔记: 通俗理解卷积神经网络
 lm_蜡笔小新456: 偶然间看到的, 很感兴趣!

CNN笔记: 通俗理解卷积神经网络
 v_JULY_v: @zhoutengtengsonw: 如果我们讲师团队里的冯老师所说: "也可以全用relu, 看个人选择。..."

CNN笔记: 通俗理解卷积神经网络
 zhoutengtengsonw: "常用的非线性激活函数有sigmoid、tanh、relu等等, 前两者sigmoid/tanh比较常..."

Google or baidu?

Google搜--"结构之法" (My BLOG)

baidu 搜--"结构之法" (My BLOG)

我的驻点

00、我的新浪微博

01、我的Github主页

通俗来讲, 想象你在曼哈顿要从一个十字路口开车到另外一个十字路口, 驾驶距离是两点间的直线距离吗? 显然不是, 除非你能穿越大楼。而实际驾驶距离就是这个“曼哈顿距离”, 此即曼哈顿距离名称的来源, 同时, 曼哈顿距离也称为城市街区距离(City Block distance)。

(1)二维平面两点 $a(x_1, y_1)$ 与 $b(x_2, y_2)$ 间的曼哈顿距离

$$d_{12} = |x_1 - x_2| + |y_1 - y_2|$$

(2)两个n维向量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的曼哈顿距离

$$d_{12} = \sum_{k=1}^n |x_{1k} - x_{2k}|$$

• 3. 切比雪夫距离, 若二个向量或二个点 p 、and q , 其坐标分别为 P_i 及 Q_i , 则两者之间的切比雪夫距离定义如下: $D_{\text{Chebyshev}}(p, q) := \max_i (|p_i - q_i|)$.

这也等于以下 L_p 度量的极值: $\lim_{k \rightarrow \infty} \left(\sum_{i=1}^n |p_i - q_i|^k \right)^{1/k}$, 因此切比雪夫距离也称为 L^∞ 度量。

以数学的观点来看, 切比雪夫距离是由一致范数 (uniform norm) (或称为上确界范数) 所衍生的度量, 也是超凸度量 (injective metric space) 的一种。

在平面几何中, 若二点 p 及 q 的直角坐标系坐标为 (x_1, y_1) 及 (x_2, y_2) , 则切比雪夫距离为: $D_{\text{Chess}} = \max(|x_2 - x_1|, |y_2 - y_1|)$ 。

玩过国际象棋的朋友或许知道, 国王走一步能够移动到相邻的8个方格中的任意一个。那么国王从格子 (x_1, y_1) 走到格子 (x_2, y_2) 最少需要多少步?。你会发现最少步数总是 $\max(|x_2 - x_1|, |y_2 - y_1|)$ 步。有一种类似的一种距离度量方法叫切比雪夫距离。

(1)二维平面两点 $a(x_1, y_1)$ 与 $b(x_2, y_2)$ 间的切比雪夫距离

$$d_{12} = \max(|x_1 - x_2|, |y_1 - y_2|)$$

(2)两个n维向量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的切比雪夫距离

$$d_{12} = \max_i (|x_{1i} - x_{2i}|)$$

这个公式的另一种等价形式是

$$d_{12} = \lim_{k \rightarrow \infty} \left(\sum_{i=1}^n |x_{1i} - x_{2i}|^k \right)^{1/k}$$

• 4. 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。

(1) 闵氏距离的定义

两个n维变量 $a(x_{11}, x_{12}, \dots, x_{1n})$ 与 $b(x_{21}, x_{22}, \dots, x_{2n})$ 间的闵可夫斯基距离定义为:

$$d_{12} = \sqrt[p]{\sum_{k=1}^n |x_{1k} - x_{2k}|^p}$$

02、七月在线
 03、寒
 04、joycewyj的机器学习笔记
 05、Harry
 06、NoSQLFan
 07、酷勤网
 08、52nlp
 09、IT面试论坛
 10、北大朋友的挖掘乐园
 11、跟Sophia_qing一起读硕士
 12、caopengcs
 13、51nod
 14、韩寒
 15、曾经的叛逆与年少
 16、code4app:iOS代码示例
 17、斯坦福机器学习公开课
 18、Memory Model与并发编程
 19、淘宝搜索技术博客
 20、interviewstreet
 21、LeetCode
 22、Team_Algorithms

文章存档

2016年07月 (1)
 2015年10月 (1)
 2015年08月 (1)
 2014年11月 (4)
 2014年10月 (1)

展开

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。

- **5. 标准化欧氏距离 (Standardized Euclidean distance)**，标准化欧氏距离是针对简单欧氏距离的缺点而作的一种改进方案。标准欧氏距离的思路：既然数据各维分量的分布不一样，那先将各个分量都“标准化”到均值、方差相等。至于均值和方差标准化到多少，先复习点统计学知识。

假设样本集X的数学期望或均值(mean)为m，标准差(standard deviation，方差开根)为s，那么X的“标准化变量” X^* 表示为： $(X-m)/s$ ，而且标准化变量的数学期望为0，方差为1。

即，样本集的标准化过程(standardization)用公式描述就是：

$$X^* = \frac{X - m}{s}$$

标准化后的值 = (标准化前的值 - 分量的均值) / 分量的标准差

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：

$$d_{12} = \sqrt{\sum_{k=1}^n \left(\frac{x_{1k} - x_{2k}}{s_k} \right)^2}$$

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)。

- **6. 马氏距离(Mahalanobis Distance)**

(1) 马氏距离定义

有M个样本向量 $X_1 \sim X_m$ ，协方差矩阵记为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：

$$D(X) = \sqrt{(X - \mu)^T S^{-1} (X - \mu)}$$

(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 X_i 与 X_j 之间的马氏距离定义为：

$$D(X_i, X_j) = \sqrt{(X_i - X_j)^T S^{-1} (X_i - X_j)}$$

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：

$$D(X_i, X_j) = \sqrt{(X_i - X_j)^T (X_i - X_j)}$$

也就是欧氏距离了。

若协方差矩阵是对角矩阵，公式变成了标准化欧氏距离。

(2)马氏距离的优缺点：量纲无关，排除变量之间的相关性的干扰。

「微博上的seafood高清版点评道：原来马氏距离是根据协方差矩阵演变，一直被老师误导了，怪不得看Killian在05年NIPS发表的LMNN论文时候老是看到协方差矩阵和半正定，原来是这回事」

- **7、巴氏距离 (Bhattacharyya Distance)**，在统计中，Bhattacharyya距离测量两个离散或连续概率分布的相似性。它与衡量两个统计样品或种群之间的重叠量的Bhattacharyya系数密切相关。Bhattacharyya距离和Bhattacharyya系数以20世纪30年代曾在印度统计研究所工作的一个统计学家A. Bhattacharya命名。同时，Bhattacharyya系数可以被用来确定两个样本被认为相对接近的，它是用来测量中的类分类的可分离性。

(1) 巴氏距离的定义

对于离散概率分布 p和q在同一域 X，它被定义为：

$$D_B(p, q) = -\ln(BC(p, q))$$

其中:

$$BC(p, q) = \sum_{x \in X} \sqrt{p(x)q(x)}$$

是Bhattacharyya系数。

对于连续概率分布, Bhattacharyya系数被定义为:

$$BC(p, q) = \int \sqrt{p(x)q(x)} dx$$

在 $0 \leq BC \leq 1$ and $0 \leq D_B \leq \infty$ 这两种情况下, 巴氏距离 D_B 并没有服从三角不等式。(值得一提的是,

Hellinger距离不服从三角不等式 $0 \leq D_B \leq \infty D_B \sqrt{1 - BC}$)。

对于多变量的高斯分布 $p_i = N(m_i, P_i)$,

$$D_B = \frac{1}{8}(m_1 - m_2)^T P^{-1}(m_1 - m_2) + \frac{1}{2} \ln \left(\frac{\det P}{\sqrt{\det P_1 \det P_2}} \right),$$

$$P = \frac{P_1 + P_2}{2}$$

和是手段和协方差的分布

需要注意的是, 在这种情况下, 第一项中的Bhattacharyya距离与马氏距离有关联。

(2) Bhattacharyya系数

Bhattacharyya系数是两个统计样本之间的重叠量的近似测量, 可以被用于确定被考虑的两个样本的相对接近。

计算Bhattacharyya系数涉及集成的基本形式的两个样本的重叠的时间间隔的值的两个样本被分裂成一个选定的分区数, 并且在每个分区中的每个样品的成员的数量, 在下面的公式中使用

$$\text{Bhattacharyya} = \sum_{i=1}^n \sqrt{(\Sigma \mathbf{a}_i \cdot \Sigma \mathbf{b}_i)}$$

考虑样品a和b, n是分区数, 并且 $\Sigma \mathbf{a}_i$, $\Sigma \mathbf{b}_i$ 被一个和b i的日分区中的样本数量的成员。更多介绍请参看: http://en.wikipedia.org/wiki/Bhattacharyya_coefficient。

- 8. 汉明距离(Hamming distance), 两个等长字符串s1与s2之间的汉明距离定义为将其中一个变为另外一个所需要作的最小替换次数。例如字符串“1111”与“1001”之间的汉明距离为2。应用: 信息编码(为了增强容错性, 应使得编码间的最小汉明距离尽可能大)。

或许, 你还没明白我再说什么, 不急, 看下[上篇blog](#)中第78题的第3小题整理的一道面试题, 便一目了然了。

如下图所示:

3. 给定一个源串和目标串, 能够对源串进行如下操作:

1. 在给定位置上插入一个字符
2. 替换任意字符
3. 删除任意字符

写一个程序, 返回最小操作数, 使得对源串进行这些操作后等于目标串, 源串和目标串的长度都小于2000。

点评:

1. 此题反复出现, 如上文第38题第4小题9月26日百度一二面试题, 10月9日腾讯面试题第1小题, 及上面第69题10月13日百度2013校招北京站笔试题第二大道题第3小题, 同时, 还可以看下这个链

[cpp] view plain copy print ?

//动态规划:

//f[i,j]表示s[0...i]与t[0...j]的最小编辑距离。

f[i,j] = min { f[i-1,j]+1, f[i,j-1]+1, f[i-1,j-1]+(s[i]==t[j]?0:1) }

//分别表示: 添加1个, 删除1个, 替换1个(相同就不用替换)。

与此同时, 面试官还可以继续问下去: 那么, 请问, 如何设计一个比较两篇文章相似性的算法? (这个问

题的讨论可以看看这里：<http://t.cn/zl82CAH>，及这里关于simhash算法的介

绍：<http://www.cnblogs.com/linecong/archive/2010/08/28/simhash.html>），接下来，便引出了下文关于夹角余弦的讨论。

（上篇blog中第78题的第3小题给出了多种方法，读者可以参看之。同时，程序员编程艺术系列第二十八章将详细阐述这个问题）

- **9. 夹角余弦(Cosine)**，几何中夹角余弦可用来衡量两个向量方向的差异，机器学习中借用这一概念来衡量样本向量之间的差异。

(1)在二维空间中向量A(x1,y1)与向量B(x2,y2)的夹角余弦公式：

$$\cos\theta = \frac{x_1x_2 + y_1y_2}{\sqrt{x_1^2 + y_1^2} \sqrt{x_2^2 + y_2^2}}$$

(2) 两个n维样本点a(x11,x12,...,x1n)和b(x21,x22,...,x2n)的夹角余弦

$$\cos(\theta) = \frac{a \cdot b}{|a| |b|}$$

类似的，对于两个n维样本点a(x11,x12,...,x1n)和b(x21,x22,...,x2n)，可以使用类似于夹角余弦的概念来衡量它们间的相似程度，即：

$$\cos(\theta) = \frac{\sum_{k=1}^n x_{1k} x_{2k}}{\sqrt{\sum_{k=1}^n x_{1k}^2} \sqrt{\sum_{k=1}^n x_{2k}^2}}$$

夹角余弦取值范围为[-1,1]。夹角余弦越大表示两个向量的夹角越小，夹角余弦越小表示两向量的夹角越大。当两个向量的方向重合时夹角余弦取最大值1，当两个向量的方向完全相反夹角余弦取最小值-1。

- **10. 杰卡德相似系数(Jaccard similarity coefficient)**

(1) 杰卡德相似系数

两个集合A和B的交集元素在A，B的并集中所占的比例，称为两个集合的杰卡德相似系数，用符号J(A,B)表示。

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

杰卡德相似系数是衡量两个集合的相似度一种指标。

(2) 杰卡德距离

与杰卡德相似系数相反的概念是杰卡德距离(Jaccard distance)。

杰卡德距离可用如下公式表示：

$$J_d(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

杰卡德距离用两个集合中不同元素占所有元素的比例来衡量两个集合的区分度。

(3) 杰卡德相似系数与杰卡德距离的应用

可将杰卡德相似系数用在衡量样本的相似度上。

举例：样本A与样本B是两个n维向量，而且所有维度的取值都是0或1，例如：A(0111)和B(1011)。我们将样本看成是一个集合，1表示集合包含该元素，0表示集合不包含该元素。

M11：样本A与B都是1的维度的个数

M01：样本A是0，样本B是1的维度的个数

M10：样本A是1，样本B是0 的维度的个数

M_{00} : 样本A与B都是0的维度的个数

依据上文给的杰卡德相似系数及杰卡德距离的相关定义, 样本A与B的杰卡德相似系数J可以表示为:

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}.$$

这里 $M_{11}+M_{01}+M_{10}$ 可理解为A与B的并集的元素个数, 而 M_{11} 是A与B的交集的元素个数。而样本A与B的杰卡德距离表示为J':

$$J' = \frac{M_{01} + M_{10}}{M_{01} + M_{10} + M_{11}}.$$

• 11.皮尔逊系数(Pearson Correlation Coefficient)

在具体阐述皮尔逊相关系数之前, 有必要解释下什么是相关系数 (Correlation coefficient)与相关距离(Correlation distance)。

相关系数 (Correlation coefficient)的定义是:

$$\rho_{XY} = \frac{\text{Cov}(X,Y)}{\sqrt{D(X)}\sqrt{D(Y)}} = \frac{E[(X-EX)(Y-EY)]}{\sqrt{D(X)}\sqrt{D(Y)}}$$

(其中, E为数学期望或均值, D为方差, D开根号为标准差, $E\{[X-E(X)][Y-E(Y)]\}$ 称为随机变量X与Y的协方差, 记为 $\text{Cov}(X,Y)$, 即 $\text{Cov}(X,Y) = E\{[X-E(X)][Y-E(Y)]\}$, 而两个变量之间的协方差和标准差的商则称为随机变量X与Y的相关系数, 记为 ρ_{XY})

相关系数衡量随机变量X与Y相关程度的一种方法, 相关系数的取值范围是[-1,1]。相关系数的绝对值越大, 则表明X与Y相关度越高。当X与Y线性相关时, 相关系数取值为1 (正线性相关) 或-1 (负线性相关)。

具体的, 如果有两个变量: X、Y, 最终计算出的相关系数的含义可以有如下理解:

1. 当相关系数为0时, X和Y两变量无关系。
2. 当X的值增大 (减小), Y值增大 (减小), 两个变量为正相关, 相关系数在0.00与1.00之间。
3. 当X的值增大 (减小), Y值减小 (增大), 两个变量为负相关, 相关系数在-1.00与0.00之间。

相关距离的定义是:

$$D_{xy} = 1 - \rho_{XY}$$

OK, 接下来, 咱们来重点了解下皮尔逊相关系数。

在统计学中, 皮尔逊积矩相关系数 (英语: Pearson product-moment correlation coefficient, 又称作PPMCC或PCCs, 用r表示) 用于度量两个变量X和Y之间的相关 (线性相关), 其值介于-1与1之间。

通常情况下通过以下取值范围判断变量的相关强度:

相关系数	0.8-1.0	极强相关
	0.6-0.8	强相关
	0.4-0.6	中等程度相关
	0.2-0.4	弱相关
	0.0-0.2	极弱相关或无相关

在自然科学领域中, 该系数广泛用于度量两个变量之间的相关程度。它是由卡尔·皮尔逊从弗朗西斯·高尔顿在19世纪80年代提出的一个相似却又稍有不同的想法演变而来的。这个相关系数也称作“皮尔森相关系数r”。

(1)皮尔逊系数的定义:

两个变量之间的皮尔逊相关系数定义为两个变量之间的协方差和标准差的商:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X\sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X\sigma_Y},$$

以上方程定义了总体相关系数, 一般表示成希腊字母 ρ (rho)。基于样本对协方差和方差进行估计, 可以得到样

本标准差,一般表示成r:

$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

一种等价表达式的是表示成标准分的均值。基于(Xi, Yi)的样本点, 样本皮尔逊系数是

$$r = \frac{1}{n-1} \sum_{i=1}^n \left(\frac{X_i - \bar{X}}{s_X} \right) \left(\frac{Y_i - \bar{Y}}{s_Y} \right)$$

$$\frac{X_i - \bar{X}}{s_X}$$

其中 $\frac{X_i - \bar{X}}{s_X}$ 、 \bar{X} 及 s_X , 分别是标准分、样本平均值和样本标准差。

或许上面的讲解令你头脑混乱不堪, 没关系, 我换一种方式讲解, 如下:

假设有两个变量X、Y, 那么两变量间的皮尔逊相关系数可通过以下公式计算:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E((X - \mu_X)(Y - \mu_Y))}{\sigma_X \sigma_Y} = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E^2(X)} \sqrt{E(Y^2) - E^2(Y)}}$$

• 公式一:

注: 勿忘了上面说过, “皮尔逊相关系数定义为两个变量之间的协方差和标准差的商”, 其中标准差的计算公式为:

随机变量的标准差计算公式

一随机变量 X 的标准差定义为:

$$\sigma = \sqrt{E((X - E(X))^2)} = \sqrt{E(X^2) - (E(X))^2}$$

$$\rho_{X,Y} = \frac{N \sum XY - \sum X \sum Y}{\sqrt{N \sum X^2 - (\sum X)^2} \sqrt{N \sum Y^2 - (\sum Y)^2}}$$

• 公式二:

$$\rho_{X,Y} = \frac{\sum (X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum (X - \bar{X})^2} \sqrt{\sum (Y - \bar{Y})^2}}$$

• 公式三:

$$\rho_{X,Y} = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sqrt{(\sum X^2 - \frac{(\sum X)^2}{N})(\sum Y^2 - \frac{(\sum Y)^2}{N})}}$$

• 公式四:

以上列出的四个公式等价, 其中E是[数学期望](#), cov表示[协方差](#), N表示变量取值的个数。

(2)皮尔逊相关系数的适用范围

当两个变量的标准差都不为零时, 相关系数才有定义, 皮尔逊相关系数适用于:

1. 两个变量之间是线性关系, 都是连续数据。
2. 两个变量的总体是正态分布, 或接近正态的单峰分布。
3. 两个变量的观测值是成对的, 每对观测值之间相互独立。

(3)如何理解皮尔逊相关系数

rubyist: 皮尔逊相关系数理解有两个角度

其一, 按照高中数学水平来理解, 它很简单, 可以看做将两组数据首先做Z分数处理之后, 然后两组数据的乘积和除以样本数, Z分数一般代表正态分布中, 数据偏离中心点的距离. 等于变量减掉平均数再除以标准差.(就是

高考的标准分类似的处理)

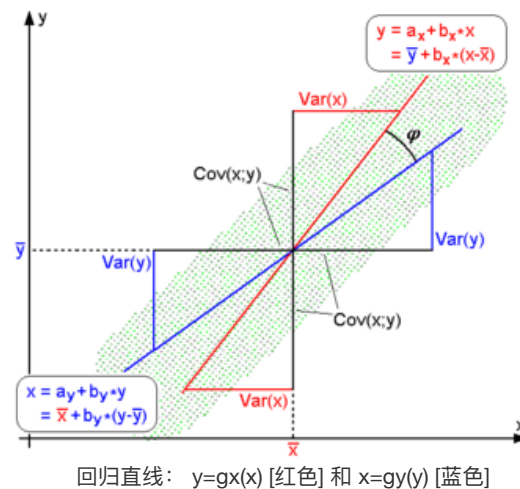
样本标准差则等于变量减掉平均数的平方和，再除以样本数，最后再开方，也就是说，方差开方即为标准差，样本标准差计算公式为：

$$\sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2}$$

所以，根据这个最朴素的理解，我们可以将公式依次精简为：

$$\begin{aligned} r_{xy} &= \frac{\sum Z_x Z_y}{N} \\ &= \frac{\sum \left(\frac{X - \bar{X}}{S_x} \right) \left(\frac{Y - \bar{Y}}{S_y} \right)}{N} \\ &= \frac{\sum (X - \bar{X})(Y - \bar{Y})}{N \cdot S_x S_y} \\ &= \frac{\sum (X - \bar{X})(Y - \bar{Y})}{N \cdot \left(\sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2} \right) \left(\sqrt{\frac{1}{n} \sum_{i=1}^n (Y_i - \bar{Y})^2} \right)} \\ &= \frac{\sum (X - \bar{X})(Y - \bar{Y})}{\left(\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \right) \left(\sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2} \right)} \end{aligned}$$

其二，按照大学的线性数学水平来理解，它比较复杂一点，可以看做是两组数据的向量夹角的余弦。下面是关于此皮尔逊系数的几何学的解释，先来看一幅图，如下所示：



如上图，对于没有中心化的数据，相关系数与两条可能的回归线 $y=g_x(x)$ 和 $x=g_y(y)$ 夹角的余弦值一致。对于没有中心化的数据 (也就是说，数据移动一个样本平均值以使其均值为0)，相关系数也可以被视作由两个随机变量 向量 夹角的 余弦值 (见下方)。

举个例子，例如，有5个国家的国民生产总值分别为 10, 20, 30, 50 和 80 亿美元。假设这5个国家 (顺序相同) 的贫困百分比分别为 11%, 12%, 13%, 15%, and 18%。令 x 和 y 分别为包含上述5个数据的向量: $x = (1, 2, 3, 5, 8)$ 和 $y = (0.11, 0.12, 0.13, 0.15, 0.18)$ 。

利用通常的方法计算两个向量之间的夹角 (参见 数量积), 未中心化 的相关系数是:

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{2.93}{\sqrt{103}\sqrt{0.0983}} = 0.920814711.$$

我们发现以上的数据特意选定为完全相关: $y = 0.10 + 0.01x$ 。于是, 皮尔逊相关系数应该等于1。将数据中心化 (通过 $E(x) = 3.8$ 移动 x 和通过 $E(y) = 0.138$ 移动 y) 得到 $x = (-2.8, -1.8, -0.8, 1.2, 4.2)$ 和 $y = (-0.028, -0.018, -0.008, 0.012, 0.042)$, 从中

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{0.308}{\sqrt{30.8}\sqrt{0.00308}} = 1 = \rho_{xy},$$

(4)皮尔逊相关的约束条件

从以上解释, 也可以理解皮尔逊相关的约束条件:

- 1 两个变量间有线性关系
- 2 变量是连续变量
- 3 变量均符合正态分布,且二元分布也符合正态分布
- 4 两变量独立

在实践统计中,一般只输出两个系数,一个是相关系数,也就是计算出来的相关系数大小,在-1到1之间;另一个是独立样本检验系数,用来检验样本一致性。

简单说来, 各种“距离”的应用场景简单概括为, 空间: 欧氏距离, 路径: 曼哈顿距离, 国际象棋国王: 切比雪夫距离, 以上三种的统一形式: 闵可夫斯基距离, 加权: 标准化欧氏距离, 排除量纲和依存: 马氏距离, 向量差距: 夹角余弦, 编码差别: 汉明距离, 集合近似度: 杰卡德类似系数与距离, 相关: 相关系数与相关距离。

1.3、K值的选择

除了上述1.2节如何定义邻居的问题之外, 还有一个选择多少个邻居, 即K值定义为多大的问题。不要小看了这个K值选择问题, 因为它对K近邻算法的结果会产生重大影响。如李航博士的一书「统计学习方法」上所说:

1. 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;
2. 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。
3. $K=N$, 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。

在实际应用中, K值一般取一个比较小的数值, 例如采用交叉验证法 (简单来说, 就是一部分样本做训练集, 一部分做测试集) 来选择最优的K值。

第二部分、K近邻算法的实现: KD树

2.0、背景

之前blog内曾经介绍过SIFT特征匹配算法, 特征点匹配和数据库查、图像检索本质上是同一个问题, 都可以归结为一个通过距离函数在高维矢量之间进行相似性检索的问题, 如何快速而准确地找到查询点的近邻, 不少人提出了很多高维空间索引结构和近似查询的算法。

一般说来, 索引结构中相似性查询有两种基本的方式:

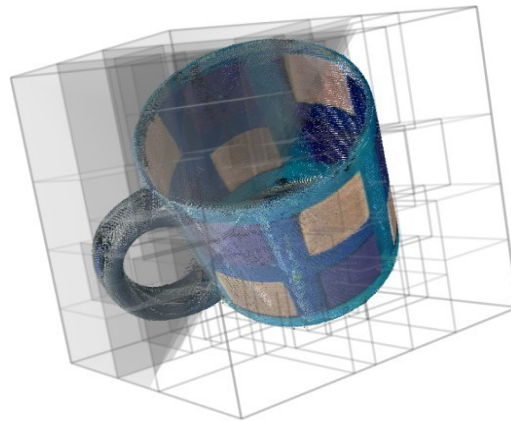
1. 一种是范围查询, 范围查询时给定查询点和查询距离阈值, 从数据集中查找所有与查询点距离小于阈值的数
据
2. 另一种是K近邻查询, 就是给定查询点及正整数K, 从数据集中找到距离查询点最近的K个数据, 当K=1时,
它就是最近邻查询。

同样, 针对特征点匹配也有两种方法:

- 最容易的办法就是线性扫描, 也就是我们常说的穷举搜索, 依次计算样本集E中每个样本到输入实例点的距
离, 然后抽取计算出最小距离的点即为最近邻点。此种办法简单直白, 但当样本集或训练集很大时,
它的缺点就立马暴露出来了, 举个例子, 在物体识别的问题中, 可能有数千个甚至数万个SIFT特征点, 而去
一一计算这成千上万的特征点与输入实例点的距离, 明显是不足取的。
- 另外一种, 就是构建数据索引, 因为实际数据一般都会呈现簇状的聚类形态, 因此我们想到建立数据索引,
然后再进行快速匹配。索引树是一种树结构索引方法, 其基本思想是对搜索空间进行层次划分。根据划分的
空间是否有混叠可以分为Clipping和Overlapping两种。前者划分空间没有重叠, 其代表就是k-d树; 后者划分
空间相互有交叠, 其代表为R树。

而关于R树本blog内之前已有介绍(同时, 关于基于R树的最近邻查找, 还可以看下这篇文
章: http://blog.sina.com.cn/s/blog_72e1c7550101dsc3.html), 本文着重介绍k-d树。

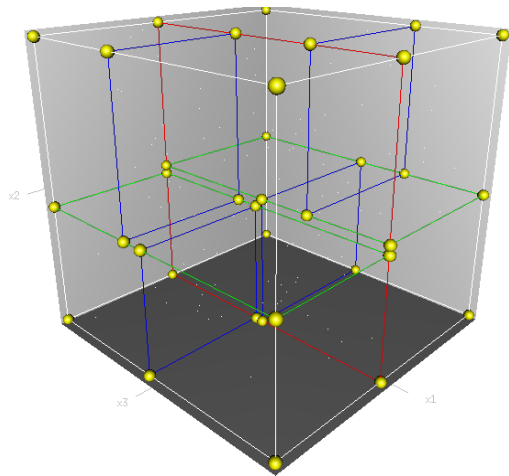
1975年, 来自斯坦福大学的Jon Louis Bentley在ACM杂志上发表的一篇论文: Multidimensional Binary Search
Trees Used for Associative Searching 中正式提出和阐述的如下图形式的把空间划分为多个部分的k-d树。



2.1、什么是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:

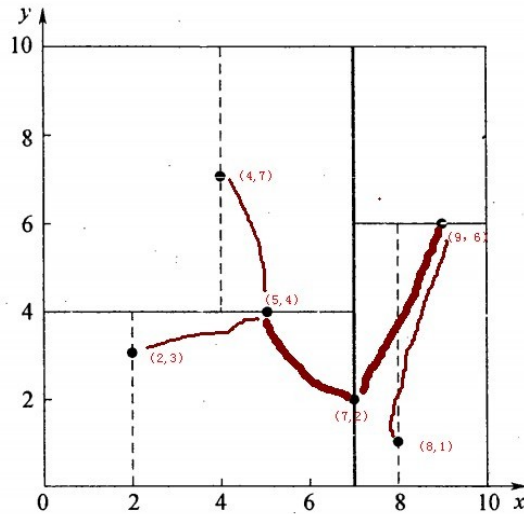


2.2、KD树的构建

kd树构建的伪代码如下所示：

算法：构建 Kd-树
输入：数据点集 Data-set 和 其所在的空间：Range
输出：Kd，类型为 Kd-tree
1. If Data-set 是空的，则返回空的 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），统计它们在每个维上的数据方差。以 SURF 为例，描述子为 64 维，可计算出 64 个方差。挑选出方差中的最大值，对应的维就是:split 域的值。数据方差最大表明沿该坐标轴方向上数据点分散得比较开，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 按其第 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 ∈ Data-set' && d[:split] ≤ Node-data[:split]} Left_Range={Range && dataleft} dataright = {d ∈ Data-set' && d[:split] > Node-data[:split]} Right_Range={Range && dataright}
4. :left= 由(dataleft,Left_Range)建立的 Kd-tree 设置:left 的 parent 域（父节点）为 Kd :right=由 (dataright,Right_Range)建立的 Kd-tree 设置:right 的 parent 域（父节点）为 Kd

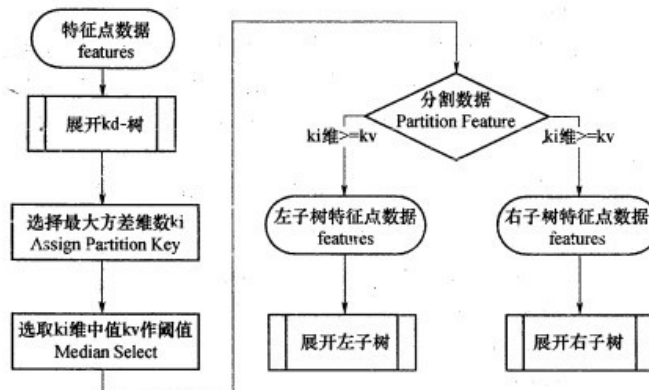
再举一个简单直观的实例来介绍k-d树构建算法。假设有6个二维数据点{(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)}, 数据点位于二维空间内，如下图所示。为了能有效的找到最近邻，k-d树采用分而治之的思想，即将整个空间划分为几个小部分，首先，粗黑线将空间一分为二，然后在两个子空间中，细黑直线又将整个空间划分为四部分，最后虚黑直线将这四部分进一步划分。



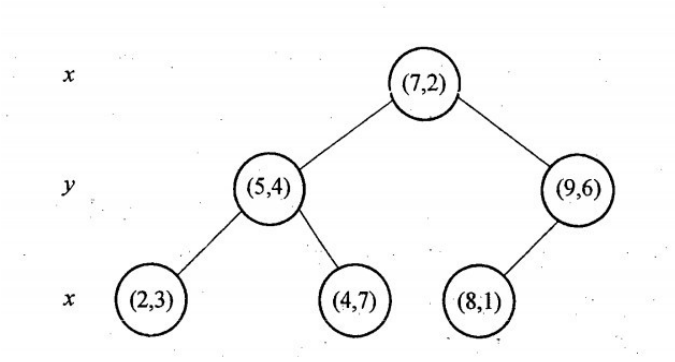
6个二维数据点{(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)}构建kd树的具体步骤为：

1. 确定：split域=x。具体是：6个数据点在x, y维度上的数据方差分别为39, 28.63, 所以在x轴上**方差**更大, 故split域值为x；
2. 确定：Node-data = (7,2)。具体是：根据x维上的值将数据排序, 6个数据的中值(所谓中值, 即中间大小的值)为7, 所以Node-data域位数据点 (7,2)。这样, 该节点的分割超平面就是通过 (7,2) 并垂直于：split=x轴的直线x=7；
3. 确定：左子空间和右子空间。具体是：分割超平面x=7将整个空间分为两部分：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)}；

如上算法所述, kd树的构建是一个递归过程, 我们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一级子节点 (5,4) 和 (9,6), 同时将空间和数据集进一步细分, 如此往复直到空间中只包含一个数据点。



与此同时, 经过对上面所示的空间划分之后, 我们可以看出, 点(7,2)可以为根结点, 从根结点出发的两条红粗斜线指向的(5,4)和(9,6)则为根结点的左右子结点, 而(2,3), (4,7)则为(5,4)的左右孩子(通过两条细红斜线相连), 最后, (8,1)为(9,6)的左孩子(通过细红斜线相连)。如此, 便形成了下面这样一棵k-d树：



k-d树的数据结构

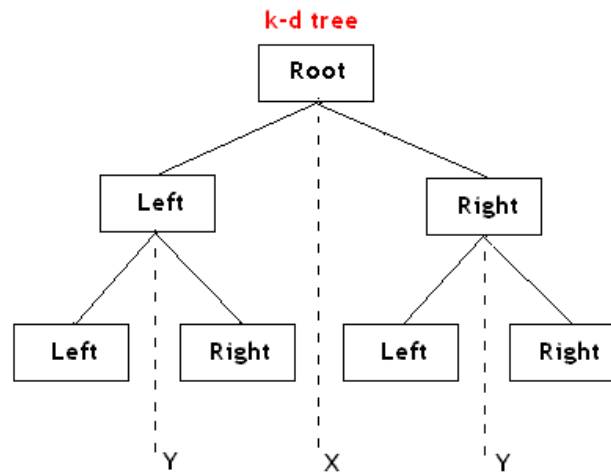
域名	类型	描述
dom_elt	kd维的向量	kd维空间中的一个样本点
split	整数	分裂维的序号，也是垂直于分割超面的方向轴序号
left	kd-tree	由位于该结点分割超面左子空间内所有数据点构成的kd-tree
right	kd-tree	由位于该结点分割超面右子空间内所有数据点构成的kd-tree

针对上表给出的kd树的数据结构，转化成具体代码如下所示(注，本文以下代码分析基于Rob Hess维护的sift库)：

```
[cpp] view plain copy print ?
/** a node in a k-d tree */
struct kd_node
{
    int ki;                /**< partition key index *///关键点直方图方差最大向量系列位置
    double kv;             /**< partition key value *///直方图方差最大向量系列中最中间模值
    int leaf;              /**< 1 if node is a leaf, 0 otherwise */
    struct feature* features; /**< features at this node */
    int n;                 /**< number of features */
    struct kd_node* kd_left; /**< left child */
    struct kd_node* kd_right; /**< right child */
};
```

也就是说，如之前所述，kd树中，kd代表k-dimension，每个节点即为一个k维的点。每个非叶节点可以想象为一个分割超平面，用垂直于坐标轴的超平面将空间分为两个部分，这样递归的从根节点不停的划分，直到没有实例为止。经典的构造k-d tree的规则如下：

1. 随着树的深度增加，循环的选取坐标轴，作为分割超平面的法向量。对于3-d tree来说，根节点选取x轴，根节点的孩子选取y轴，根节点的孙子选取z轴，根节点的曾孙子选取x轴，这样循环下去。
2. 每次均为所有对应实例的中位数的实例作为切分点，切分点作为父节点，左右两侧为划分的作为左右两子树。



对于n个实例的k维数据来说，建立kd-tree的时间复杂度为 $O(k*n*\log n)$ 。

以下是构建k-d树的代码：

```

[cpp] view plain copy print ?
struct kd_node* kdtree_build( struct feature* features, int n )
{
    struct kd_node* kd_root;

    if( ! features || n <= 0 )
    {
        fprintf( stderr, "Warning: kdtree_build(): no features, %s, line %d\n",
            __FILE__, __LINE__ );
        return NULL;
    }

    //初始化
    kd_root = kd_node_init( features, n ); //n--number of features,inititalize root of tree.
    expand_kd_node_subtree( kd_root ); //kd tree expand

    return kd_root;
}

```

上面的涉及初始化操作的两个函数kd_node_init，及expand_kd_node_subtree代码分别如下所示：

```

[cpp] view plain copy print ?
static struct kd_node* kd_node_init( struct feature* features, int n )
{
    struct kd_node* kd_node;

    kd_node = (struct kd_node*)(malloc( sizeof( struct kd_node ) ));
    memset( kd_node, 0, sizeof( struct kd_node ) ); //0填充
    kd_node->ki = -1; //???????
    kd_node->features = features;
    kd_node->n = n;

    return kd_node;
}

```

```

[cpp] view plain copy print ?
static void expand_kd_node_subtree( struct kd_node* kd_node )
{
    /* base case: leaf node */
    if( kd_node->n == 1 || kd_node->n == 0 )
    {
        //叶节点 //伪叶节点
        kd_node->leaf = 1;
        return;
    }

    assign_part_key( kd_node ); //get ki,kv
    partition_features( kd_node ); //creat left and right children,特征点ki位置左树比右树模值小,kv作为分界
    //kd_node中关键点已经排序
}

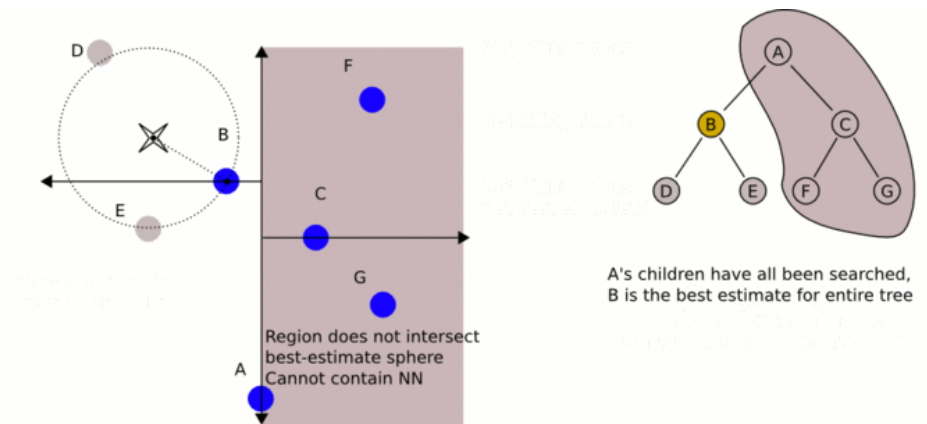
```

```

if( kd_node->kd_left )
    expand_kd_node_subtree( kd_node->kd_left );
if( kd_node->kd_right )
    expand_kd_node_subtree( kd_node->kd_right );
}

```

构建完kd树之后，如今进行最近邻搜索呢？从下面的动态gif图中，你是否能看出些许端倪呢？

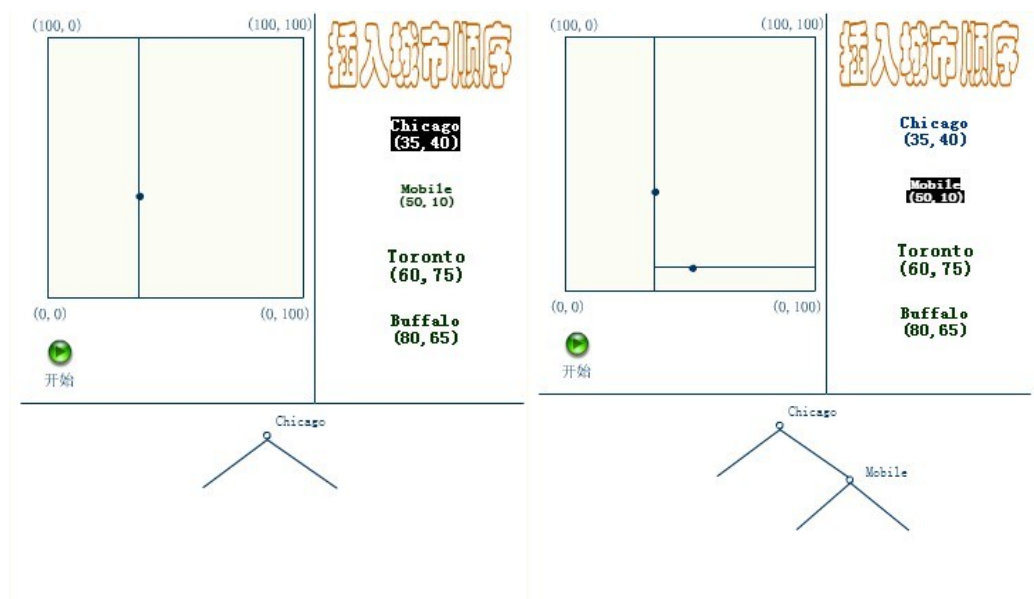


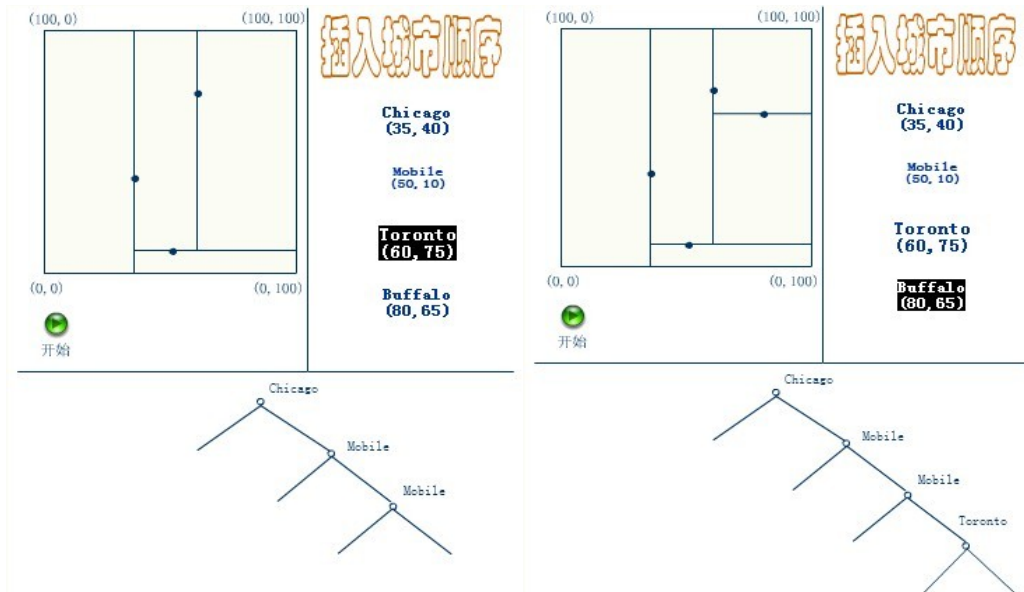
k-d树算法可以分为两大部分，除了上部分有关k-d树本身这种数据结构建立的算法，另一部分是在建立的k-d树上各种诸如插入，删除，查找(最邻近查找)等操作涉及的算法。下面，咱们依次来看kd树的插入、删除、查找操作。

2.3、KD树的插入

元素插入到一个K-D树的方法和二叉检索树类似。本质上，在偶数层比较x坐标值，而在奇数层比较y坐标值。当我们到达了树的底部，（也就是当一个空指针出现），我们也找到了结点将要插入的位置。生成的K-D树的形状依赖于结点插入时的顺序。给定N个点，其中一个结点插入和检索的平均代价是 $O(\log_2 N)$ 。

下面4副图(来源：中国地质大学电子课件)说明了插入顺序为(a) Chicago, (b) Mobile, (c) Toronto, and (d) Buffalo，建立空间K-D树的示例：



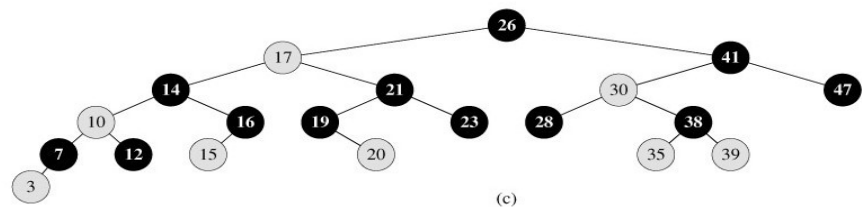


应该清楚，这里描述的插入过程中，每个结点将其所在的平面分割成两部分。因比，Chicago 将平面上所有结点分成两部分，一部分所有的结点x坐标值小于35，另一部分结点的x坐标值大于或等于35。同样Mobile将所有x坐标值大于35的结点以分成两部分，一部分结点的Y坐标值是小于10，另一部分结点的Y坐标值大于或等于10。后面的Toronto、Buffalo也按照一分为二的规则继续划分。

2.4、KD树的删除

KD树的删除可以用递归程序来实现。我们假设希望从K-D树中删除结点 (a,b)。如果 (a,b) 的两个子树都为空，则用空树来代替 (a,b)。否则，在 (a,b) 的子树中寻找一个合适的结点来代替它，譬如(c,d)，则递归地从K-D树中删除 (c,d)。一旦(c,d)已经被删除，则用 (c,d) 代替 (a,b)。假设(a,b)是一个X识别器，那么，它得替代节点要么是 (a,b) 左子树中的X坐标最大值的结点，要么是 (a,b) 右子树中x坐标最小值的结点。

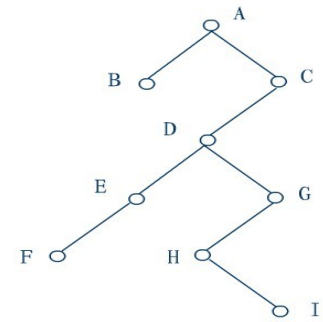
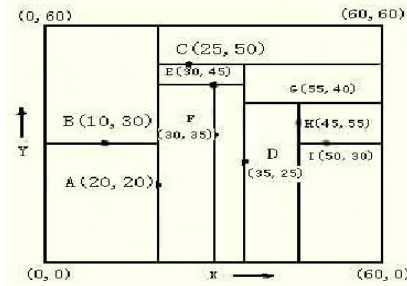
也就是说，跟普通二叉树(包括如下图所示的**红黑树**)结点的删除是同样的思想：用被删除节点A的左子树的最右节点或者A的右子树的最左节点作为替代A的节点(比如，下图红黑树中，若要删除根结点26，第一步便是用23或28取代根结点26)。



当(a,b)的右子树为空时，找到 (a,b) 左子树中具有x坐标最大的结点，譬如 (c,d)，将(a,b)的左子树放到(c,d)的右子树中，且在树中从它的上一层递归地应用删除过程（也就是 (a,b) 的左子树）。

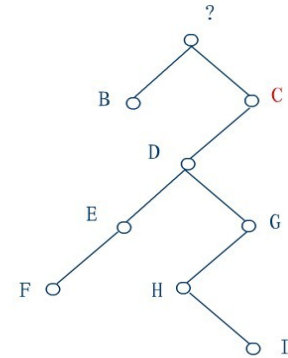
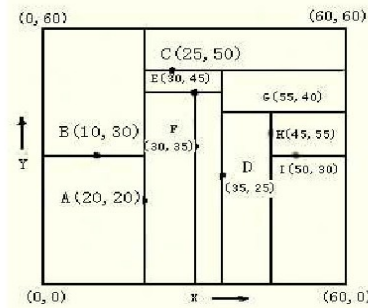
下面来举一个实际的例子(来源：中国地质大学电子课件，原课件错误已经在下文订正)，如下图所示，原始图像及对应的kd树，现在要删除图中的A结点，请看一系列删除步骤：

原始图像及对应k-d树



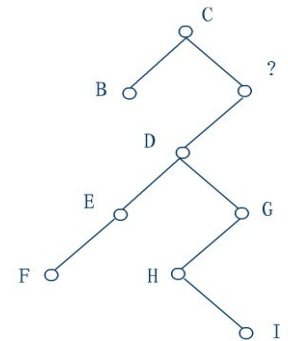
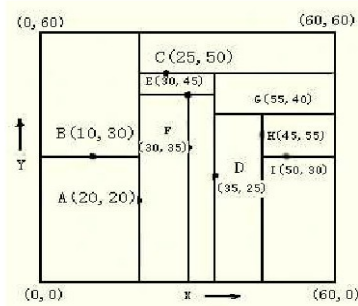
要删除上图中结点A，选择结点A的右子树中X坐标值最小的结点，这里是C，C成为根，如下图：

原始图像及对应k-d树



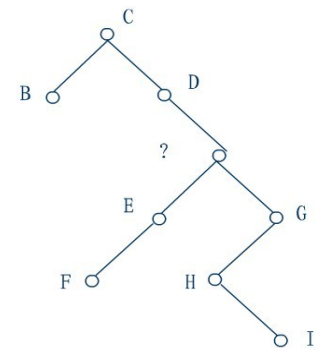
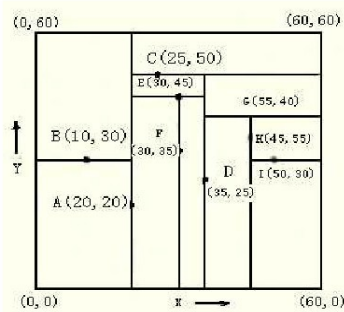
从C的右子树中找出一个结点代替先前C的位置，

原始图像及对应k-d树



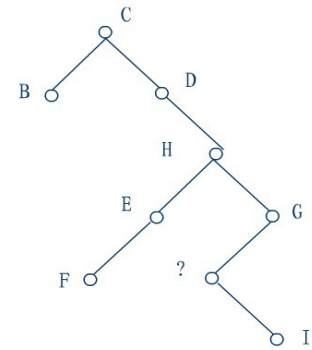
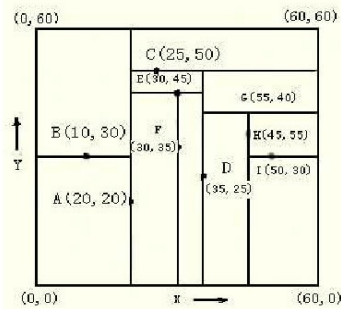
这里是D，并将D的左子树转为它的右子树，D代替先前C的位置，如下图：

原始图像及对应k-d树



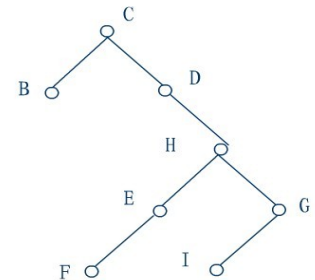
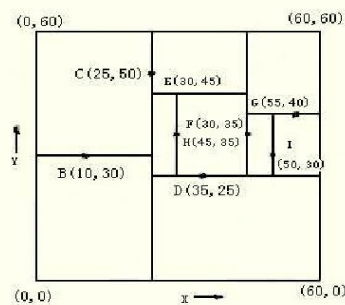
在D的新右子树中，找X坐标最小的结点，这里为H，H代替D的位置，

原始图像及对应k-d树



在D的右子树中找到一个Y坐标最小的值，这里是I，将I代替原先H的位置，从而A结点从图中顺利删除，如下图所示：

生成的新图像及k-d树



从一个K-D树中删除结点(a,b)的问题变成了在(a,b)的子树中寻找x坐标为最小的结点。不幸的是寻找最小x坐标值的结点比二叉检索树中解决类似的问题要复杂得多。特别是虽然最小x坐标值的结点一定在x识别器的左子树中，但它同样可在y识别器的两个子树中。因此关系到检索，且必须注意检索坐标，以使在每个奇数层仅检索2个子树中的一个。

从K-D树中删除一个结点是代价很高的，很清楚删除子树的根受到子树中结点个数的限制。用TPL (T) 表示树T总的长度。可看出树中子树大小的总和为TPL (T) + N。以随机方式插入N个点形成树的TPL是O(N*log2N),这就意味着从一个随机形成的K-D树中删除一个随机选取的结点平均代价的上界是O(log2N)。

2.5、KD树的最近邻搜索算法

现实生活中有许多问题需要在多维数据的快速分析和快速搜索，对于这个问题最常用的方法是所谓的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。在一个N维的笛卡儿空间在两个点之间的距离是由下述公式确定：

$$d = \sqrt{(x_1^1 - x_2^1)^2 + (x_1^2 - x_2^2)^2 + \dots + (x_1^N - x_2^N)^2}$$

2.5.1、k-d树查询算法的伪代码

k-d树查询算法的伪代码如下所示：

[cpp] view plain copy print ?

```

算法：k-d树最近邻查找
输入：Kd, //k-d tree类型
      target //查询数据点
输出：nearest, //最近邻数据点
      dist //最近邻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
        Min_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;
        End while
3. //回溯查找
while (search_path != NULL)
    back_point = 从search_path取出一个节点指针; //从search_path堆栈弹栈
    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
    End while

```

读者来信点评@yhxyhxyhx, 在“将Kd_point压入search_path堆栈;”这行代码后, 应该是调到步骤2再往下走二分搜索的逻辑一直到叶结点, 我写了一个递归版本的二维kd tree的搜索函数你对比的看看:

```

[cpp] view plain copy print ?
void innerGetClosest(NODE* pNode, PT point, PT& res, int& nMinDis)
{
    if (NULL == pNode)
        return;
    int nCurDis = abs(point.x - pNode->pt.x) + abs(point.y - pNode->pt.y);
    if (nMinDis < 0 || nCurDis < nMinDis)
    {
        nMinDis = nCurDis;
        res = pNode->pt;
    }
    if (pNode->splitX && point.x <= pNode->pt.x || !pNode->splitX && point.y <= pNode->pt.y)
        innerGetClosest(pNode->pLft, point, res, nMinDis);
    else
        innerGetClosest(pNode->pRgt, point, res, nMinDis);
    int rang = pNode->splitX ? abs(point.x - pNode->pt.x) : abs(point.y - pNode->pt.y);
    if (rang > nMinDis)
        return;
    NODE* pGoInto = pNode->pLft;
    if (pNode->splitX && point.x > pNode->pt.x || !pNode->splitX && point.y > pNode->pt.y)
        pGoInto = pNode->pRgt;
    innerGetClosest(pGoInto, point, res, nMinDis);
}

```

下面, 以两个简单的实例(例子来自图像局部不变特性特征与描述一书)来描述最邻近查找的基本思路。

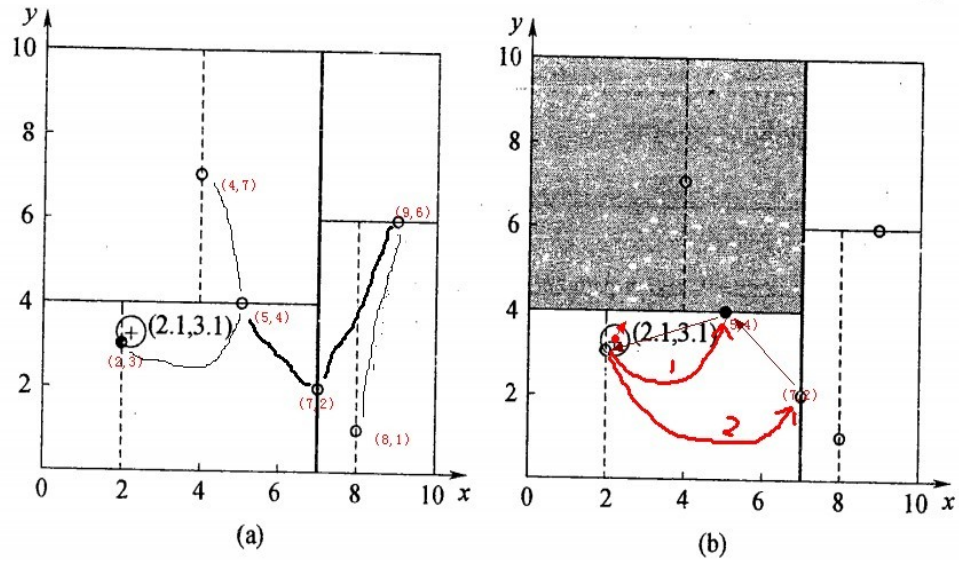
2.5.2、举例：查询点 (2.1,3.1)

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, 还需要进行相关的‘回溯’操作。也就是说, 算法首先沿搜索路径反向查找是否有距离查询点更近的数据点。

以查询 (2.1,3.1) 为例:

1. 二叉树搜索: 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<(7,2), (5,4), (2,3)>, 首先以 (2,3) 作为当前最近邻点, 计算其到查询点 (2.1,3.1) 的距离为 0.1414,
2. 回溯查找: 在得到 (2,3) 为查询点的最近点之后, 回溯到其父节点 (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(图中灰色区域)去搜索;
3. 最后, 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

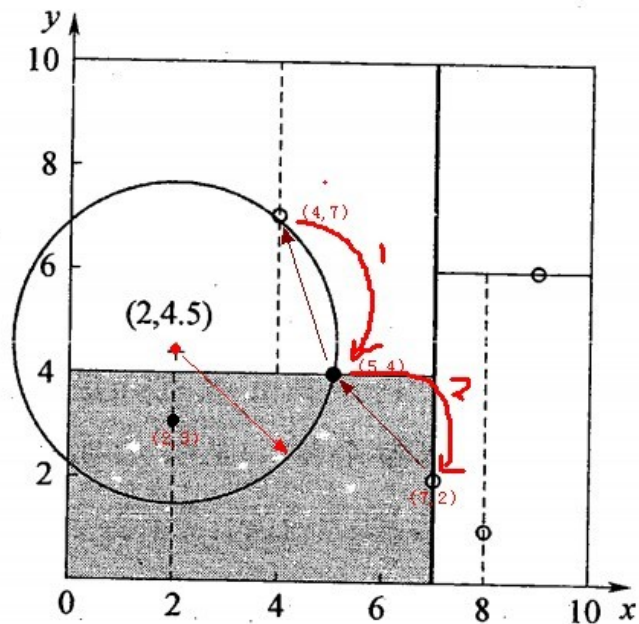
(2,3) , 最近距离为0.1414。



2.5.3、举例：查询点 (2, 4.5)

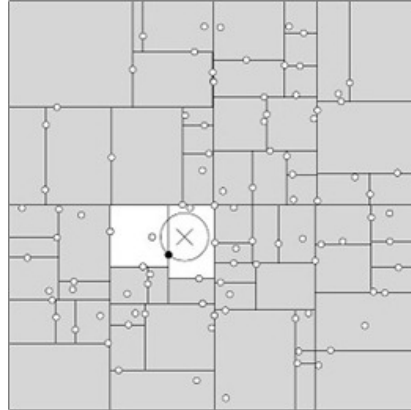
一个复杂点的例子如查找点为 (2, 4.5) , 具体步骤依次如下：

1. 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 $\langle (7,2), (5,4), (4,7) \rangle$ ，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 $\langle (7,2), (2,3) \rangle$ ；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；
3. 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。

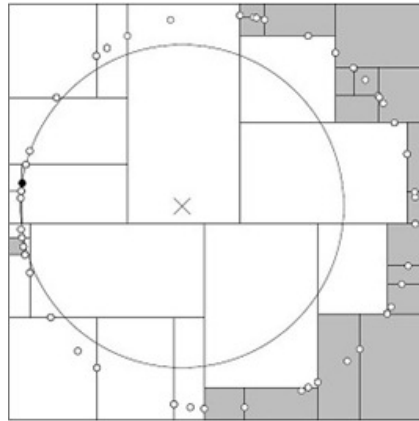


上述两次实例表明，当查询点的邻域与分割超平面两侧空间交割时，需要查找另一侧子空间，导致检索过程复杂，效率下降。

一般来讲，最临近搜索只需要检测几个叶子结点即可，如下图所示：



但是，如果当实例点的分布比较糟糕时，几乎要遍历所有的结点，如下图所示：



研究表明N个节点的K维k-d树搜索过程时间复杂度为： $t_{\text{worst}} = O(kN^{1-1/k})$ 。

同时，以上为了介绍方便，讨论的是二维或三维情形。但在实际的应用中，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（维数不超过20）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 $N \gg 2^D$ ，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(下文2.6节kd树近邻搜索算法的改进：BBF算法，与2.7节球树、M树、VP树、MVP树)。

2.6、kd树近邻搜索算法的改进：BBF算法

咱们顺着上一节的思路，参考《统计学习方法》一书上的内容，再来总结下kd树的最近邻搜索算法：

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树种找出包含目标点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为球心，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。

如果实例点是随机分布的，那么kd树搜索的平均计算复杂度是 $O(\log N)$ ，这里的N是训练实例数。所以说，kd树更适用于训练实例数远大于空间维数时的k近邻搜索，当空间维数接近训练实例数时，它的效率会迅速下降，一降降到“解放前”：线性扫描的速度。

也正因为上述k最近邻搜索算法的第4个步骤中的所述：“回退到根结点时，搜索结束”，每个最近邻点的查询比较完成过程最终都要回退到根结点而结束，而导致了許多不必要回溯访问和比较到的结点，这些多余的损耗在高维度数据查找的时候，搜索效率将变得相当之低下，那有什么办法可以改进这个原始的kd树最近邻搜索算法呢？

从上述标准的kd树查询过程可以看出其搜索过程中的“回溯”是由“查询路径”决定的，并没有考虑查询路径上一些数据点本身的一些性质。一个简单的改进思路就是将“查询路径”上的结点进行排序，如按各自分割超平面（也称bin）与查询点的距离排序，也就是说，回溯检查总是从优先级最高（Best Bin）的树结点开始。

针对此BBF机制，读者Feng&书童点评道：

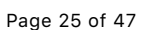
1. 在某一层，分割面是第 k_i 维，分割值是 k_v ，那么 $abs(q[k_i]-k_v)$ 就是没有选择的那个分支的优先级，也就是计算的是那一维上的距离；
2. 同时，从优先队列里面取节点只在某次搜索到叶节点后才发生，计算过距离的节点不会出现在队列的，比如1~10这10个节点，你第一次搜索到叶节点的路径是1-5-7，那么1，5，7是不会出现在优先队列的。换句话说，优先队列里面存的都是查询路径上节点对应的相反子节点，比如：搜索左子树，就把对应这一层的右节点存进队列。

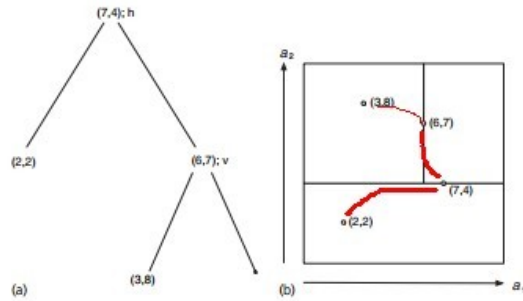
如此，就引出了本节要讨论的kd树最近邻搜索算法的改进：BBF（Best-Bin-First）查询算法，它是由发明sift算法的David Lowe在1997的一篇文章中针对高维数据提出的一种近似算法，此算法能确保优先检索包含最近邻点可能性较高的空间，此外，BBF机制还设置了一个运行超时限定。采用了BBF查询机制后，kd树便可以有效的扩展到高维数据集上。

伪代码如下图所示（图取自图像局部不变特性特征与描述一书）：

还是以上面的查询 (2,4.5) 为例，搜索的算法流程为：

- 如你在下图所见到的那样（话说，用鼠标在图片上写字着实不好写）：

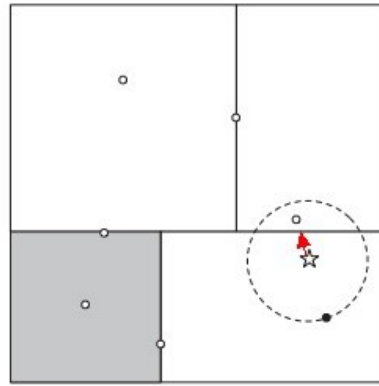




现要找它的最近邻。

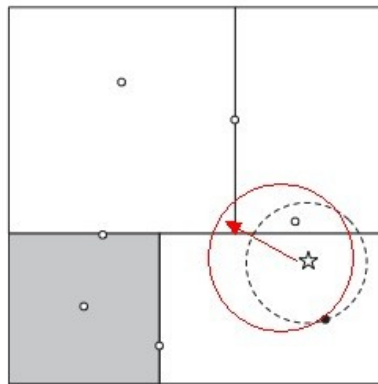
通过上文2.5节，总结来说，我们已经知道：

1、为了找到一个给定目标点的最近邻，需要从树的根结点开始向下沿树找出目标点所在的区域，如下图所示，给定目标点，用星号标示，我们似乎一眼看出，有一个点离目标点最近，因为它落在以目标点为圆心以较小长度为半径的虚线圆内，但为了确定是否可能还村庄一个最近的近邻，我们会先检查叶节点的同胞结点，然叶节点的同胞结点在图中所示的阴影部分，虚线圆并不与之相交，所以确定同胞叶结点不可能包含更近的近邻。



2、于是我们回溯到父节点，并检查父节点的同胞结点，父节点的同胞结点覆盖了图中所有横线X轴上的区域。因为虚线圆与右上方的矩形(KD树把二维平面划分成一个一个矩形)相交...

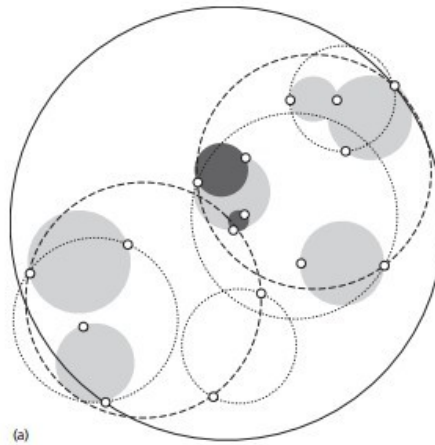
如上，我们看到，KD树是可用于有效寻找最近邻的一个树结构，但这个树结构其实并不完美，当处理不均匀分布的数据集时便会呈现出一个基本冲突：既邀请树有完美的平衡结构，又要求待查找的区域近似方形，但不管是近似方形，还是矩形，甚至正方形，都不是最好的使用形状，因为他们都有角。



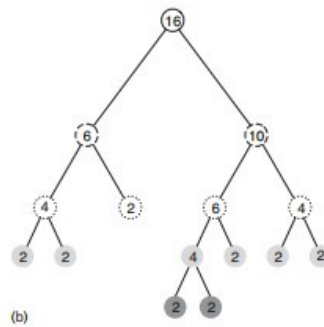
什么意思呢？就是说，在上图中，如果黑色的实例点离目标点星点再远一点，那么势必那个虚线圆会如红线所示

那样扩大，以致与左上方矩形的右下角相交，既然相交了，那么势必又必须检查这个左上方矩形，而实际上，最近的点离星点的距离很近，检查左上方矩形区域已是多余。于此我们看见，KD树把二维平面划分成一个个矩形，但矩形区域的角却是个难以处理的问题。

解决的方案就是使用如下图所示的球树：

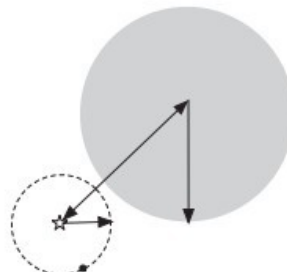


先从球中选择一个离球的中心最远的点，然后选择第二个点离第一个点最远，将球中所有的点分配到离这两个聚类中心最近的一个上，然后计算每个聚类的中心，以及聚类能够包含它所有数据点所需的最小半径。这种方法的优点是分裂一个包含 n 个特殊点的球的成本只是随 n 呈线性增加。



使用球树找出给定目标点的最近邻方法是，首先自上而下贯穿整棵树找出包含目标点所在的叶子，并在这个球里找出与目标点最靠近的点，这将确定出目标点距离它的最近邻点的一个上限值，然后跟KD树查找一样，检查同胞结点，如果目标点到同胞结点中心的距离超过同胞节点的半径与当前的上限值之和，那么同胞结点里不可能存在一个更近的点；否则的话，必须进一步检查位于同胞结点以下的子树。

如下图，目标点还是用一个星表示，黑色点是当前已知的目标点的最近邻，灰色球里的所有内容将被排除，因为灰色球的中心点离得太远，所以它不可能包含一个更近的点，像这样，递归的向树的根结点进行回溯处理，检查所有可能包含一个更近于当前上限值的点的球。



球树是自上而下的建立，和KD树一样，根本问题就是要找到一个好的方法将包含数据点集的球分裂成两个，在实践中，不必等到叶子结点只有两个数据点时才停止，可以采用和KD树一样的方法，一旦结点上的数据点打到预先设置的最小数量时，便可提前停止建树过程。

也就是上面所述，先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，将球中所有的点分配到离这两个聚类中心最近的一个上，然后计算每个聚类的中心，以及聚类能够包含它所有数据点所需的最小半径。这种方法的优点是分裂一个包含n个数据点的球的成本只是随n呈线性增加(注：本小节内容主要来自参考条目19：数据挖掘实用机器学习技术，[新西兰]Ian H.Witten 著，第4章4.7节)。

2.7.2、VP树与MVP树简介

高维特征向量的距离索引问题是基于内容的图像检索的一项关键技术，目前经常采用的解决办法是首先对高维特征空间做降维处理，然后采用包括四叉树、kd树、R树族等在内的主流多维索引结构，这种方法的出发点是：目前的主流多维索引结构在处理维数较低的情况时具有比较好的效率，但对于维数很高的情况则显得力不从心(即所谓的维数危机)。

实验结果表明当特征空间的维数超过20的时候，效率明显降低，而可视化特征往往采用高维向量描述，一般情况下可以达到 10^2 的量级，甚至更高。在表示图像可视化特征的高维向量中各维信息的重要程度是不同的，通过降维技术去除属于次要信息的特征向量以及相关性较强的特征向量，从而降低特征空间的维数，这种方法已经得到了一些实际应用。

然而这种方法存在不足之处采用降维技术可能会导致有效信息的损失，尤其不适合于处理特征空间中的特征向量相关性很小的情况。另外主流的多维索引结构大都针对欧氏空间，设计需要利用到欧氏空间的几何性质，而图像的相似性计算很可能不限于基于欧氏距离。这种情况下人们越来越关注基于距离的度量空间高维索引结构可以直接应用于高维向量相似性查询问题。

度量空间中对象之间的距离度量只能利用三角不等式性质，而不能利用其他几何性质。向量空间可以看作由实数坐标串组成的特殊度量空间，目前针对度量空间的高维索引问题提出的索引结构有很多种大致可以作如下分类，如下图所示：

- (1)BK-tree(Burkhard and Keller,1973)及其变种 FQ-tree (Baeza-Yates, 1994)、FHQ-tree (Baeza-Yates, 1997)、FQA (Chavez, 1999)；
- (2)VP-tree(Uhlmann, 1991)及其变种 MVP-tree(Bozkaya, 1997)、VPF(Yianlios, 1999)；
- (3)BS-tree(Kalantari, 1983)及其变种 GH-tree(Uhlmann, 1991)、GNA-tree(Brin, 1995)、VT(Dehne, 1987)；
- (4)M-tree(Ciaccia, 1997)；
- (5)SA-tree(Navarro, 1999)；
- (6)AESA(Ruiz, 1986)及其变种 LAESA(María, 1994)。

其中，VP树和MVP树中特征向量的举例表示为：

特征向量之间的距离值采用公式(1)计算得到：

$$Dist(X, Y) = \sqrt[p]{\sum_{i=1}^N (|X_i - Y_i|)^p} \quad (1)$$

读者点评：

1. UESTC_HN_AY_GUOBO：现在主要是在kdtree的基础上有了mtree或者mvptree，其实关键还是pivot的选择，以及度量空间中算法怎么减少距离计算；
2. mandycool：mvp-tree，是利用三角形不等式来缩小搜索区域的，不过mvp-tree的目标稍有不同，查询的是到query点的距离小于某个值r的点；另外作者test的数据集只有20维，不知道上百维以后效果如何，而减少距

离计算的一个思路是做embedding，通过不等式排除掉一部分点。

更多内容请参见论文1：DIST ANCE-BASED INDEXING FOR HIGH-DIMENSIONAL METRIC SP ACES，作者：Tolga Bozkaya & Meral Ozsoyoglu，及论文2：基于度量空间高维索引结构VP-tree及MVP-tree的图像检索，王志强，甘国辉，程起敏。

当然，如果你觉得上述论文还不够满足你胃口的话，这里有一大堆nearest neighbor algorithms相关的论文可供你看：http://scholar.google.com.hk/scholar?q=nearest+neighbor+algorithms&btnG=&hl=zh-CN&as_sdt=0&as_vis=1（其中，这篇可以看下：Spill-Trees, An investigation of practical approximate nearest neighbor algorithms）。

第三部分、KD树的应用：SIFT+KD_BBf搜索算法

3.1、SIFT特征匹配算法

之前本blog内阐述过图像特征匹配SIFT算法，写过五篇文章，这五篇文章分别为：

- 九、图像特征提取与匹配之SIFT算法 （sift算法系列五篇文章）
- 九（续）、sift算法的编译与实现
- 九（再续）、教你一步一步用c语言实现sift算法、上
- 九（再续）、教你一步一步用c语言实现sift算法、下
- 九（三续）：SIFT算法的应用—目标识别之Bag-of-words模型

不熟悉SIFT算法相关概念的可以看上述几篇文章，这里不再做赘述。与此同时，本文此部分也作为十五个经典算法研究系列里SIFT算法的九之四续。

OK，我们知道，在sift算法中，给定两幅图片图片，若要做特征匹配，一般会先提取出图片中的下列相关属性作为特征点：

```
[cpp] view plain copy print ?
/**
Structure to represent an affine invariant image feature. The fields
x, y, a, b, c represent the affine region around the feature:


$$a(x-u)(x-u) + 2b(x-u)(y-v) + c(y-v)(y-v) = 1$$

*/
struct feature
{
    double x;           /**< x coord */
    double y;           /**< y coord */
    double a;           /**< Oxford-type affine region parameter */
    double b;           /**< Oxford-type affine region parameter */
    double c;           /**< Oxford-type affine region parameter */
    double scl;         /**< scale of a Lowe-style feature */
    double ori;         /**< orientation of a Lowe-style feature */
    int d;              /**< descriptor length */
    double descr[FEATURE_MAX_D]; /**< descriptor */
    int type;           /**< feature type, OXFD or LOWE */
    int category;       /**< all-purpose feature category */
    struct feature* fwd_match; /**< matching feature from forward image */
    struct feature* bck_match; /**< matching feature from backward image */
    struct feature* mdl_match; /**< matching feature from model */
    CvPoint2D64f img_pt; /**< location in image */
    CvPoint2D64f mdl_pt; /**< location in model */
    void* feature_data; /**< user-definable data */
    char dense;         /*表征特征点所处稠密程度*/
};
```

而后在sift.h文件中定义两个关键函数，这里，我们把它们称之为函数一，和函数二，如下所示：

函数一的声明：

```
[cpp] view plain copy print ?
extern int sift_features( IplImage* img, struct feature** feat );
```

函数一的实现：

```
[cpp] view plain copy print ?
int sift_features( IplImage* img, struct feature** feat )
{
    return _sift_features( img, feat, SIFT_INTVLS, SIFT_SIGMA, SIFT_CONTR_THR,
                           SIFT_CURV_THR, SIFT_IMG_DBL, SIFT_DESCR_WIDTH,
                           SIFT_DESCR_HIST_BINS );
}
```

从上述函数一的实现中，我们可以看到，它内部实际调用的是这个函数：_sift_features(..)，也就是下面马上要分析的函数二。

函数二的声明：

```
[cpp] view plain copy print ?
extern int _sift_features( IplImage* img, struct feature** feat, int intvls,
                          double sigma, double contr_thr, int curv_thr,
                          int img_dbl, int descr_width, int descr_hist_bins );
```

函数二的实现：

```
[cpp] view plain copy print ?
int _sift_features( IplImage* img, struct feature** feat, int intvls,
                  double sigma, double contr_thr, int curv_thr,
                  int img_dbl, int descr_width, int descr_hist_bins )
{
    IplImage* init_img;
    IplImage*** gauss_pyr, *** dog_pyr;
    CvMemStorage* storage;
    CvSeq* features;
    int octvs, i, n = 0, n0 = 0, n1 = 0, n2 = 0, n3 = 0, n4 = 0;
    int start;

    /* check arguments */
    if( ! img )
        fatal_error( "NULL pointer error, %s, line %d", __FILE__, __LINE__ );

    if( ! feat )
        fatal_error( "NULL pointer error, %s, line %d", __FILE__, __LINE__ );

    /* build scale space pyramid; smallest dimension of top level is ~4 pixels */
    start=GetTickCount();
    init_img = create_init_img( img, img_dbl, sigma );
    octvs = log( (float)(MIN( init_img->width, init_img->height )) ) / log((float)(2.0)) -5;
    gauss_pyr = build_gauss_pyr( init_img, octvs, intvls, sigma );
    dog_pyr = build_dog_pyr( gauss_pyr, octvs, intvls );
    fprintf( stderr, " creat the pyramid use %d\n", GetTickCount()-start);

    storage = cvCreateMemStorage( 0 ); //创建存储内存, 0为默认64k
    start=GetTickCount();
    features = scale_space_extrema( dog_pyr, octvs, intvls, contr_thr,
                                   curv_thr, storage ); //在DOG空间寻找极值点, 确定关键点位置
    fprintf( stderr, " find the extrum points in DOG use %d\n", GetTickCount()-start);

    calc_feature_scales( features, sigma, intvls ); //计算关键点的尺度

    if( img_dbl )
        adjust_for_img_dbl( features ); //如果原始空间图扩大, 特征点坐标就缩小
    start=GetTickCount();
    calc_feature_oris( features, gauss_pyr ); //在gaussian空间计算关键点的主方向和幅值
    fprintf( stderr, " get the main oritation use %d\n", GetTickCount()-start);

    start=GetTickCount();
    compute_descriptors( features, gauss_pyr, descr_width, descr_hist_bins ); //建立关键点描述器
    fprintf( stderr, " compute the descriptors use %d\n", GetTickCount()-start);

    /* sort features by decreasing scale and move from CvSeq to array */
    //start=GetTickCount();
    //cvSeqSort( features, (CvCmpFunc)feature_cmp, NULL ); //?????
    n = features->total;
    *feat = (feature*)(calloc( n, sizeof(struct feature) ));
    *feat = (feature*)(cvCvtSeqToArray( features, *feat, CV_WHOLE_SEQ )); //整条链表放在feat指向的内存

    for( i = 0; i < n; i++ )
    {
        free( (*feat)[i].feature_data );
        (*feat)[i].feature_data = NULL; //释放ddata(r,c,octv,intvl,xi,scl_octv)
    }
}
```

```

        if((*feat)[i].dense == 4) ++n4;
        else if((*feat)[i].dense == 3) ++n3;
        else if((*feat)[i].dense == 2) ++n2;
        else if((*feat)[i].dense == 1) ++n1;
        else ++n0;
    }

    //fprintf( stderr, " move features from sequece to array use %d\n",GetTickCount()-start);
    //start=GetTickCount();
    fprintf( stderr, "In the total feature points the extent4 points is %d\n",n4);
    fprintf( stderr, "In the total feature points the extent3 points is %d\n",n3);
    fprintf( stderr, "In the total feature points the extent2 points is %d\n",n2);
    fprintf( stderr, "In the total feature points the extent1 points is %d\n",n1);
    fprintf( stderr, "In the total feature points the extent0 points is %d\n",n0);
    cvReleaseMemStorage( &storage );
    cvReleaseImage( &init_img );
    release_pyr( &gauss_pyr, octvs, intvls + 3 );
    release_pyr( &dog_pyr, octvs, intvls + 2 );
    //fprintf( stderr, " free the pyramid use %d\n",GetTickCount()-start);
    return n;
}

```

说明：上面的函数二，包含了SIFT算法中几乎所有函数，是SIFT算法的核心。本文不打算一一分析上面所有函数，只会抽取其中涉及到BBF查询机制相关的函数。

3.2、K个最小近邻的查找：大顶堆优先级队列

上文中一直在讲最近邻问题，也就是说只找最近的那唯一一个邻居，但如果现实中需要找到k个最近的邻居。该如何做呢？对的，之前blog内曾相近阐述过[寻找最小的k个数](#)的问题，显然，寻找k个最近邻与寻找最小的k个数的问题如出一辙。

回忆下寻找k个最小的数中关于构造大顶堆的解决方案：

“寻找最小的k个数，更好的办法是维护k个元素的最大堆，即用容量为k的最大堆存储最先遍历到的k个数，并假设它们即是最小的k个数，建堆费时 $O(k)$ 后，有 $k_1 < k_2 < \dots < k_{max}$ （ k_{max} 设为大顶堆中最大元素）。继续遍历数列，每次遍历一个元素x，与堆顶元素比较， $x < k_{max}$ ，更新堆（用时 $\log k$ ），否则不更新堆。这样下来，总费时 $O(k + (n-k) * \log k) = O(n * \log k)$ 。此方法得益于在堆中，查找等各项操作时间复杂度均为 $\log k$ 。”

根据上述方法，咱们来写大顶堆最小优先级队列相关代码实现：

```

[cpp] view plain copy print ?
void* minpq_extract_min( struct min_pq* min_pq )
{
    void* data;

    if( min_pq->n < 1 )
    {
        fprintf( stderr, "Warning: PQ empty, %s line %d\n", __FILE__, __LINE__ );
        return NULL;
    }
    data = min_pq->pq_array[0].data; //root of tree
    min_pq->n--; //0
    min_pq->pq_array[0] = min_pq->pq_array[min_pq->n];
    restore_minpq_order( min_pq->pq_array, 0, min_pq->n );
    //0
    return data;
}

```

上述函数中，restore_minpq_order的实现如下：

```

[cpp] view plain copy print ?
static void restore_minpq_order( struct pq_node* pq_array, int i, int n )
{
    struct pq_node tmp;
    int l, r, min = i;

    l = left( i ); //2*i+1,????????????
    r = right( i ); //2*i+2,????????????
    if( l < n )
        if( pq_array[l].key < pq_array[i].key )
            min = l;
}

```

```

if( r < n )
    if( pq_array[r].key < pq_array[min].key )
        min = r;

if( min != i )
{
    tmp = pq_array[min];
    pq_array[min] = pq_array[i];
    pq_array[i] = tmp;
    restore_minpq_order( pq_array, min, n );
}
}

```

3.3、KD树近邻搜索改进之BBF算法

原理在上文第二部分已经阐述明白，结合大顶堆找最近的K个近邻思想，相关主体代码如下：

```

[cpp] view plain copy print ?
//KD树近邻搜索改进之BBF算法
int kdtree_bbf_knn( struct kd_node* kd_root, struct feature* feat, int k,
                   struct feature*** nbrs, int max_nn_chks ) //2
{
    //200
    struct kd_node* expl;
    struct min_pq* min_pq;
    struct feature* tree_feat, ** _nbrs;
    struct bbf_data* bbf_data;
    int i, t = 0, n = 0;

    if( ! nbrs || ! feat || ! kd_root )
    {
        fprintf( stderr, "Warning: NULL pointer error, %s, line %d\n",
                 __FILE__, __LINE__ );
        return -1;
    }

    _nbrs = (feature**)(calloc( k, sizeof( struct feature* ) )); //2
    min_pq = minpq_init();
    minpq_insert( min_pq, kd_root, 0 ); //把根节点加入搜索序列中

    //队列有东西就继续搜，同时控制在t<200步内
    while( min_pq->n > 0 && t < max_nn_chks )
    {
        //刚进来时，从kd树根节点搜索，expl是根节点
        //后进来时，expl是min_pq差值最小的未搜索节点入口
        //同时按min_pq中父，子顺序依次检验，保证父节点的差值比子节点小。这样减少返回搜索时间
        expl = (struct kd_node*)minpq_extract_min( min_pq );
        if( ! expl )
        {
            fprintf( stderr, "Warning: PQ unexpectedly empty, %s line %d\n",
                     __FILE__, __LINE__ );
            goto fail;
        }

        //从根节点(或差值最小节点)搜索，根据目标点与节点模值的差值(小)
        //确定在kd树的搜索路径，同时存储各个节点另一入口地址\同级搜索路径差值。
        //存储时比较父节点的差值，如果子节点差值比父节点差值小，交换两者存储位置，
        //使未搜索节点差值小的存储在min_pq的前面，减小返回搜索的时间。
        expl = explore_to_leaf( expl, feat, min_pq );
        if( ! expl )
        {
            fprintf( stderr, "Warning: PQ unexpectedly empty, %s line %d\n",
                     __FILE__, __LINE__ );
            goto fail;
        }

        for( i = 0; i < expl->n; i++ )
        {
            //使用expl->n原因:如果是叶节点,expl->n=1,如果是伪叶节点,expl->n=0.
            tree_feat = &expl->features[i];
            bbf_data = (struct bbf_data*)(malloc( sizeof( struct bbf_data ) ));
            if( ! bbf_data )
            {
                fprintf( stderr, "Warning: unable to allocate memory,"
                             " %s line %d\n", __FILE__, __LINE__ );
                goto fail;
            }
            bbf_data->old_data = tree_feat->feature_data;
            bbf_data->d = descr_dist_sq(feat, tree_feat); //计算两个关键点描述器差平方和
            tree_feat->feature_data = bbf_data;

            //取前k个
            n += insert_into_nbr_array( tree_feat, _nbrs, n, k );//
        }
        t++;
    }
}

```



```

minpq_release( &min_pq );
for( i = 0; i < n; i++ ) //bbf_data为何搞个old_data?
{
    bbf_data = (struct bbf_data*)(_nbrs[i]->feature_data);
    _nbrs[i]->feature_data = bbf_data->old_data;
    free( bbf_data );
}
*nbrs = _nbrs;
return n;

fail:
minpq_release( &min_pq );
for( i = 0; i < n; i++ )
{
    bbf_data = (struct bbf_data*)(_nbrs[i]->feature_data);
    _nbrs[i]->feature_data = bbf_data->old_data;
    free( bbf_data );
}
free( _nbrs );
*nbrs = NULL;
return -1;
}

```

依据上述函数kdtree_bbf_knn从上而下来，注意几点：

1、上述函数kdtree_bbf_knn中，explore_to_leaf的代码如下：

```

[cpp] view plain copy print ?
static struct kd_node* explore_to_leaf( struct kd_node* kd_node, struct feature* feat,
                                       struct min_pq* min_pq ) //kd tree //the before
{
    struct kd_node* unexpl, * expl = kd_node;
    double kv;
    int ki;

    while( expl && !expl->leaf )
    {
        ki = expl->ki;
        kv = expl->kv;

        if( ki >= feat->d )
        {
            fprintf( stderr, "Warning: comparing incompatible descriptors, %s" \
                    " line %d\n", __FILE__, __LINE__ );
            return NULL;
        }
        if( feat->descr[ki] <= kv ) //由目标点描述器确定搜索向kd tree的左边或右边
        { //如果目标点模值比节点小，搜索向tree的左边进行
            unexpl = expl->kd_right;
            expl = expl->kd_left;
        }
        else
        {
            unexpl = expl->kd_left; //else
            expl = expl->kd_right;
        }

        //把未搜索数分支入口，差值存储在min_pq,
        if( minpq_insert( min_pq, unexpl, ABS( kv - feat->descr[ki] ) ) )
        {
            fprintf( stderr, "Warning: unable to insert into PQ, %s, line %d\n",
                    __FILE__, __LINE__ );
            return NULL;
        }
    }
    return expl;
}

```

2、上述查找函数kdtree_bbf_knn中的参数k可调，代表的是要查找近邻的个数，即number of neighbors to find，在sift特征匹配中，k一般取2

```

[cpp] view plain copy print ?
k = kdtree_bbf_knn( kd_root_0, feat, 2, &nbrs, KDTREE_BBF_MAX_NN_CHKS_0 ); //点匹配函数(核心)
if( k == 2 ) //只有进行2次以上匹配过程,才算是正常匹配过程

```

3、上述函数kdtree_bbf_knn中“bbf_data->d = descr_dist_sq(feat, tree_feat); //计算两个关键点描述器差平方和”，使用的计

Page 34 of 47

16. 皮尔逊相关系数维基百科页面, <http://t.cn/zjy6Gpg>;
17. 皮尔逊相关系数的一个应用: <http://www.sobuhu.com/archives/567>;
18. <http://blog.csdn.net/wsywl/article/details/5727327>;
19. 标准差, <http://zh.wikipedia.org/wiki/%E6%A0%87%E5%87%86%E5%B7%AE>;
20. 协方差与相关性, <http://t.cn/zjyXFRB> ;
21. 电子科大kd树电子课件: <http://t.cn/zjbpXna>;
22. 编程艺术之寻找最小的k个数: http://blog.csdn.net/v_JULY_v/article/details/6403777;
23. 机器学习那些事儿, http://vdisk.weibo.com/s/ix_9F;
24. 大嘴巴漫谈数据挖掘, <http://vdisk.weibo.com/s/bUbzJ>;
25. <http://www.codeproject.com/Articles/18113/KD-Tree-Searching-in-N-dimensions-Part-I>;
26. 一个库: http://docs.pointclouds.org/trunk/group__kdtree.html;
27. 3D上使用kd树: <http://pointclouds.org/>;
28. 编辑数学公式: http://webdemo.visionobjects.com/equation.html?locale=zh_CN;
29. 基于R树的最近邻查找: http://blog.sina.com.cn/s/blog_72e1c7550101dsc3.html;
30. 包含一个demo: <http://www.leexiang.com/kd-tree>;
31. 机器学习相关降维算法, <http://www.cnblogs.com/xbinworld/category/337861.html>;
32. Machine Learning相关topic, <http://www.cnblogs.com/jerrylead/tag/Machine%20Learning/>;
33. 机器学习中的数学, <http://www.cnblogs.com/LeftNotEasy/category/273623.html>;
34. 一堆概念性wikipedia页面;
35. 基于度量空间高维索引结构VP-tree及MVP-tree的图像检索, 王志强, 甘国辉, 程起敏;
36. Spill-Trees, An investigation of practical approximate nearest neighbor algorithms;
37. DIST ANCE-BASED INDEXING FOR HIGH-DIMENSIONAL METRIC SP ACES, 作者: Tolga Bozkaya & Meral Ozsoyoglu;
38. "Multidimensional Binary Search Trees Used for Associative Searching", Jon Louis Bentley.

后记

从当天下午4点多一直写, 一直写, 直接写到了隔日凌晨零点左右, 或参考或直接间接引用了很多人的作品及一堆wikipedia页面(当然, 已经注明在上面参考文献及推荐阅读中前半部分条目, 后半部分条目则为行文之后看到的一些好的资料文章, 可以课外读读), 但本文还是有诸多地方有待修补完善, 也欢迎广大读者不吝赐教 & 指正, 感谢大家。完。July、二零一二年十一月二十一日零点五分。

顶 踩
119 5

上一篇 九月十月百度人搜, 阿里巴巴, 腾讯华为笔试面试八十题(第331-410题)

下一篇 数据挖掘中所需的概率论与数理统计知识

我的同类文章

24.data structures (11)

30.Machine L & Deep Learning (11)

• 从LSM-Tree、COLA-Tree谈...	2012-05-01	阅读 73448	• 从Trie树（字典树）谈到后...	2011-10-22	阅读 122804
• B树的C实现	2011-08-31	阅读 27230	• 从2-3-4树谈到Red-Black Tr...	2011-06-08	阅读 36010
• 从B树、B+树、B*树谈到R 树	2011-06-07	阅读 349144	• 红黑树的C++完整实现源码	2011-03-29	阅读 36239
• 红黑树从头至尾插入和删除...	2011-03-28	阅读 49869	• 大刀阔斧，抽丝剥茧：评红...	2011-01-30	阅读 8948
• 一步一图一代码，一定要让...	2011-01-09	阅读 60228	• 红黑树的C实现完整源码	2011-01-03	阅读 37049
更多文章					

参考知识库



机器学习知识库
5314 关注 | 308 收录









MySQL知识库
8640 关注 | 1396 收录

猜你在找

营销型网站建设	从K近邻算法距离度量谈到KD树SIFT+BBF算法
数据结构和算法	从K近邻算法距离度量谈到KD树SIFT+BBF算法
iOS高级开发学习之 - UI多视图	从K近邻算法距离度量谈到KD树SIFT+BBF算法
深入浅出Unity3D——第一篇	从K近邻算法距离度量谈到KD树SIFT+BBF算法
从此不求人:自主研发一套PHP前端开发框架	从K近邻算法距离度量谈到KD树SIFT+BBF算法

查看评论

- 79楼 [biscate_ccc3](#) 2016-04-11 21:29发表
 感谢卤煮，比图像局部特征不变性的书详细很多，学习了
- 78楼 [zdczdccccc](#) 2016-04-06 14:25发表
 kdtree创建时，lz描述的分割面是标准差最小的那维，但是对应的图却是按tree深度来按序轮流当超平面。初看会摸不着头脑
- 77楼 [celia01](#) 2015-11-25 20:30发表
 “如果实例点是随机分布的，那么kd树搜索的平均计算复杂度是O（NlogN），这里的N是训练实例树。”
复杂度是O(logN)才对，这么丰富的总结，细节别错了，伪代码什么的最好也仔细check下
Re: [v_JULY_v](#) 2015-11-25 21:33发表
 回复celia01：多谢指正。的确应该是：“如果实例点是随机分布的，那么kd树搜索的平均计算复杂度是O（logN），这里的N是训练实例树。”
如果还发现其他问题，敬请随时指正，thanks。
- 76楼 [Dounm](#) 2015-11-16 18:41发表
 博主，你的kd树搜索最近邻的伪代码是错误的，就不要放上去了吧。
读者来信的思路是正确的，但是贴出的代码里也有笔误，最后几个语句中，pGoInfo = pNode->PLft才对吧。
希望修正一下，毕竟这个博客的浏览率挺高的~
- 75楼 [萌之痴吃](#) 2015-09-05 20:40发表
 这一句“6个数据点在x，y维度上的数据方差分别为39，28.63”
博主：请问在k-d树的构建中，这里在x，y维度上的数据方差是什么意思？
是不是指所有的x坐标值的方差和y坐标值的方差？怎么计算结果不等于39和28.63。

```
var([2,5,9,4,8,7])=6.9667  
var([3,4,6,7,1,2])=5.3667
```

Re: [RacobitRatr](#) 2016-04-06 14:41发表



回复tutudezhenshen: 我也在同问这个问题? 你有答案了吗?

74楼 [Philipxushen](#) 2015-08-14 21:02发表



博主, 我觉得您的2.6 BBF算法的那个伪代码是错的。。

因为如果按照那个伪代码去执行, 事实上, 要遍历整棵树? $O(N)$, 我觉得, 那个伪代码应该在出堆的下面加一个判断是否需要进入左/右子树的代码, 像普通算法那样。

如果不需要进入, 那就直接Delete_Min下一个节点

73楼 [ai543064193](#) 2015-06-29 20:48发表



您好~我想问一下这个是什么意思?

```
struct feature* features; /**< features at this node  
为什么调试的时候凡是带feature的都是过不去
```

72楼 [LHY_045](#) 2015-05-24 20:32发表



感谢分享!

我目前想将sift检测到的点, 用暴力的方法匹配, 我看别人的代码中有这么两句,
BFMatcher matcher;
matcher.knnMatch(testDescriptor, matches, 2);
testDescriptor是描述符。
这里的knnMatches是什么意思呢?

71楼 [原来未知](#) 2015-05-13 17:31发表



疑问: 为什么我将博主的sift算法和作者文章的sift算法以及博主之前自己实现的sift算法的参数改为一致时, 虽然图片相同, 但是结果确实差距很大呢, 本文算法找到的点更多, 但是却丢失了更多的点, 貌似本文中点的正确性更低了

70楼 [墨清扬](#) 2015-04-06 11:07发表



为什么在回溯查找时, 只需要进入一侧的子空间呢? 如(2, 4.5)那个例子, 当回溯到(2, 3)时, 如果(2, 3)的两棵子树不为空, 那么需要两棵子树都搜索下去才对

69楼 [baidu_25682391](#) 2015-01-29 10:34发表



july兄, kd-tree的构建在Range Searching using Kd Tree中是向量按维度顺序逐层构建树结构的。使用方差的方法是哪里介绍的? 效果有没有差异?

68楼 [Finlay](#) 2015-01-22 12:13发表



KNN写的很通俗易懂。

67楼 [north_damao](#) 2015-01-21 11:05发表



都是眼泪..感谢!!

66楼 [yyp158](#) 2014-12-23 22:04发表



统计学习方法一书中说KD树是一棵二叉平衡树, 但是看插入和删除的时候, 你那里画的好像不再是一棵平衡树。是否需要去维护这个特性呢?

65楼 [David_Yang](#) 2014-11-30 11:18发表



引用“v_JULY_v”的评论:

预告

本文之后, 待写的几篇文章罗列如下:

1、数据挖掘中所需的概率论与数理统计知识, 估计年底...

神经网络入门导论，明年1月内完成；还没有出来撒，期待ing

64楼 [jacksuncss](#) 2014-11-26 02:50发表



博主，请问如何用map/reduce，
去实现kd index(kd tree)?

63楼 [ndschoo](#) 2014-11-20 17:47发表



你好，插入例子“ Chicago, (b) Mobile, (c) Toronto, and (d) Buffalo，建立空间K-D树”的示例得到的kd树是不平衡的，之前不是说kd树是平衡的吗

62楼 [尘埃1206](#) 2014-11-16 17:16发表



博主写文章显然是花了很多心思，非常感谢！我前面已经进行了SIFT特征提取，今天刚好搜到博主的这篇文章，由于对这些相关知识还不太了解，而又有网友说博文中存在一些错误，那么，作为过来人的博主能不能给点建议，作为初学者，该怎么来利用这篇博文呢？谢谢了

61楼 [绿柚子](#) 2014-11-15 16:13发表



博主，不好意思，我觉得你这一节在某些概念上理解是错误的，这一篇博客有些地方是错误的。既然是CSDN的专家是不是应该抱着对大家负责的态度来写blog呢，所以我建议您认真读一下，把错误的地方改正过来。

Re: [v_JULY_v](#) 2014-11-15 21:42发表



回复Grace_0642：你好，本博客中的几乎每一篇文章我都花费了很大的精力去写，尤其是这个机器学习&数据挖掘的系列，然时间跨度比较长（比如上面这篇写于2年前），可能因当时的理解有限，难免有疏漏的地方，如果各位发现问题或错误，欢迎随时不吝指出，thanks。

Re: [尘埃1206](#) 2014-11-16 17:14发表



回复v_JULY_v：博主写文章显然是花了很多心思，非常感谢！我前面已经进行了SIFT特征提取，今天刚好搜到博主的这篇文章，由于对这些相关知识还不太了解，而又有网友说博文中存在一些错误，那么，作为过来人的博主能不能给点建议，作为初学者，该怎么来利用这篇博文呢？谢谢了

60楼 [绿柚子](#) 2014-11-15 15:59发表



还有，博主，你的2.3小节错误太多了，建议您检查一下在放上来。

59楼 [绿柚子](#) 2014-11-15 15:38发表



博主，您好，你写的这个“元素插入到一个K-D树的方法和二叉检索树类似。本质上，在偶数层比较x坐标值，而在奇数层比较y坐标值。”那要是大于等于三维怎么办？怎么比较？

Re: [绿柚子](#) 2014-11-15 15:58发表



回复Grace_0642：这个是根据split域来比较的，貌似不是您说的这样。

58楼 [绿柚子](#) 2014-11-13 23:24发表



博主，你的k-d tree的方差貌似算错了。

Re: [v_JULY_v](#) 2014-11-13 23:36发表



回复Grace_0642：你好，你说的是具体哪一节的问题？

Re: [绿柚子](#) 2014-11-15 15:00发表



回复v_JULY_v：2.2.

=====

6个二维数据点{(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)}构建kd树的具体步骤为：

确定：split域=x。具体是：6个数据点在x，y维度上的数据方差分别为39，28.63，所以在x轴上方差更大，故split域值为x；

Re: [小白菜加油](#) 2015-05-19 09:43发表



回复Grace_0642: 方差确实是写错了的, 建议楼主改过来

57楼 绿柚子 2014-11-13 18:03发表



随着树的深度增加, 循环的选取坐标轴, 作为分割超平面的法向量。对于3-d tree来说, 根节点选取x轴, 根节点的孩子选取y轴, 根节点的孙子选取z轴, 根节点的曾孙子选取x轴, 这样循环下去。=====博主, 这个也可以么? 我记得的是在方差最大的维度上面做划分。

56楼 绿柚子 2014-11-13 17:12发表



博主, 有什么好的书推荐学习么? 谢谢

55楼 sxsyfb 2014-11-11 18:05发表



非常感谢博主如此辛苦的给我们分享这些内容! 谢谢你! 这些好东西需要慢慢消化。同时支持楼主后续继续为我们分享你的所学所思。期待.....

54楼 qq10__ 2014-09-28 10:45发表



这么好的博客居然没人评论?

53楼 yang345109993 2014-09-21 14:39发表



很喜欢博主的文章, 刚刚用豆约翰博客备份专家备份了您的全部博文。

52楼 _KDH 2014-09-03 16:21发表



感谢博主

51楼 MarkandRun 2014-08-15 11:34发表



博主, 6个数据点在x, y维度上的数据方差分别为39, 28.63, 这两个数据怎么算出来的啊?

50楼 feizigegeyeye 2014-03-20 09:08发表



多谢博主的文章, , , , 要多来学习看来。

49楼 hzwzjun 2014-01-30 13:39发表



非常感谢楼主的工作, 受益匪浅!

在图像统计中还有一个Metric 比较常用的是earth mover's distance。楼主分享和么多, 只有这么点信息回报了!

48楼 x_hh 2013-12-26 17:46发表



6个数据点在x, y维度上的数据方差分别为39, 28.63, 这两个数据怎么算出来的, 我看网上好多都是这个数据, 今天还和人讨论着, 就是算不出来, 不知道是网上的数据错的还是我们的方法错了, 大侠指点一下

47楼 瓜瓜东西 2013-12-07 22:45发表



虽然我看不到, 但我知道楼主很牛掰

46楼 一生爱花 2013-11-03 21:54发表



求博主联系方式, 有很多问题想请教! 最好有QQ

Re: v_JULY_v 2013-11-04 13:43发表



回复u012710853: 你好, 咱们可以在微博私信上联系: <http://weibo.com/julyweibo>, 多谢关注。

45楼 thinkercui 2013-10-30 22:31发表



```
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;
```

默认target点一定在Kd_point 的一个子空间中, 但是, 当target不在其中的任一子空间时, 会丢失对一个子空间的搜索。所以, 用该伪代码写的程序, 能够得到的最近邻都值得怀疑, 最直接的结果就是, 本来用其他方法实现的最近邻算法能够正确分类, 而本算法则会分错!!!

另外, 楼主伪代码中用到的有些成员变量, 在节点结构定义的时候并没有出现, 导致理解困难, 望及时改正。

Re: [ai543064193](#) 2015-06-30 17:11发表



回复qq1184810369: 对 我也感觉是 那个feature好难理解~你现在理解了吗? 求指点~5555555

44楼 [youlan9115](#) 2013-10-25 22:38发表



楼主好强大! 我决定要把你的博文通通学习一遍~~~

43楼 [fengjianbang](#) 2013-09-16 14:57发表



楼主, 您好, 最近您这篇文章看了很多遍, 但是一直有一个问题不能理解。就是kd树里面确定分割维的时候, 你的那个方差值是怎么算出来的, 还请赐教啊。

42楼 [xxc1605629895](#) 2013-09-11 09:56发表



大神, 请问 k-d树查询算法中回溯的时候用到的 Max_dist 是什么呢?

Re: [雨宫夏一](#) 2015-07-09 13:59发表



回复xxc1605629895: Max_dist的意思就是那个圆的半径, 证据代码的意思是, 如果发现有点的距离小于圆的半径, 那么要向上回溯。

41楼 [wzlang](#) 2013-09-05 09:20发表



博主您好, 小弟使用KD-BBF算法用于SIFT时发现速度很慢, 比穷举法要慢很多很多。

40楼 [catTom](#) 2013-08-26 02:20发表



看您把问题都研究挺深入的, 可以写一下后缀数组吗? 我对于后缀数组的计数排序实现很不明白

39楼 [JackDanny](#) 2013-08-05 20:06发表



很喜欢博主的文章, 刚刚用豆约翰博客备份专家备份了您的全部博文。

38楼 [MATH_F](#) 2013-08-03 00:50发表



或者说为什么kd树中删除的时候一会儿比较x 一会儿比较y 有什么标准嘛 以及 那个示例好像不是kd树啊。。

37楼 [MATH_F](#) 2013-08-03 00:19发表



KD树的删除

它得替代节点要么是 (a,b) 左子树中的X坐标最大值的结点, 要么是 (a,b) 右子树中x坐标最小值的结点
这个是不是写错了
右子树种Y最小吧 是嘛?

36楼 [孟小子](#) 2013-07-07 11:48发表



很喜欢博主的文章, 刚刚用豆约翰博客备份专家备份了您的全部博文。

35楼 [见死不救No1](#) 2013-06-02 20:08发表



楼主真厉害, 学习了!

34楼 [wz88423511](#) 2013-05-23 15:33发表



<http://www.cs.umd.edu/class/spring2008/cmsc420/L19.kd-trees.pdf>
你怎么看待不存在右子树的时候寻找左子树最大值的问题

33楼 [wz88423511](#) 2013-05-23 15:33发表



<http://www.cs.umd.edu/class/spring2008/cmsc420/L19.kd-trees.pdf>
文章中提到了关于右子树不存在的情况下寻找左子树中的最大值可能存在的问题，求赐教，你怎么看待这个问题

32楼 [wz88423511](#) 2013-05-23 15:31发表



<http://www.cs.umd.edu/class/spring2008/cmsc420/L19.kd-trees.pdf>
看一下文章中对delete kd-tree的看法
Delete in kd-trees --- No Right Subtree
Possible idea: Find the max in the left subtree?
Why might this not work?
求指教，你怎么看待这个问题

31楼 [燎原](#) 2013-05-03 10:43发表



博主，您在K-D树如何构建的那个例子中，举例有6个点，(2,3)、(5,4)。。。它们在X,Y维度上的方差是怎样计算的啊？就是用X方向上的坐标值求得平均值，再按照方差的计算公式去求吗？可是我算出来的数据跟您的差别好大好大啊，还烦请博主详述一下吧。。。我真的是小白。。。。

30楼 [奇异果](#) 2013-04-30 13:06发表



在选择用哪个维度切分的时候，是顺序循环选择，还是每次选择都选择方差最大的呢？如5维(a,b,c,d,e)，是abcdeabcdeab...这样选，还是每次都计算方差选最大的？
再有，允许相邻两次选择同一维度进行划分吗，如abb这种？

Re: [wz88423511](#) 2013-05-23 15:35发表



回复liukang0618：采用方差最大。是为了找到点分布最分散的维度，这样可以平衡做右子树，所以可以连续对同一维度进行切割的。

29楼 [飞信天下](#) 2013-03-25 17:26发表



楼主，您好！我有个问题请教您哈！
`int kdtree_bbf_knn(....)函数里，大循环while里通过expl=explore_to_leaf(expl,feat,min_pq);`

.....
`n+=insert_into_nbr_array(tree_feat,_nbrs,n,k); (1)`
这里，我注意到程序仅仅只能把也kd树的叶子才能运行语句（1），例如在上面例子中的6个二维数据点{(2,3)，(5,4)，(9,6)，(4,7)，(8,1)，(7,2)}中无法与 非叶子的根节点例如（5,4）比较。因为目标点也可能离（5，4）更近。
实际上在bbf算法里搜索kd树时都会在搜索路径上确定最佳点，即不断寻找离目标最短距离的节点。求关注！

Re: [Day_plot](#) 2013-10-26 11:48发表



回复google0802：按照Rob Hess的代码，其实他已经将kd-tree的构造改了，和博主介绍的不一样，改成了将所有数据都设成了叶子节点。你用上面的例子构造一个kd-tree，打印出所有节点就知道了

Re: [飞信天下](#) 2013-10-30 21:15发表



回复Day_plot：你的意思是说rob hess的代码的kd树把所有的数据当为叶子？我试一试打印所有叶子，谢谢你哈！

28楼 [daduzifufu](#) 2013-01-16 21:46发表



学习的近似误差, 学习的估计误差区别是什么？

27楼 [heroalone2012](#) 2013-01-10 16:09发表



博主你好，最近一直在学习你这篇博文，受益匪浅
但我有个弱弱的问题，希望博主指点。
你代码中struct kd_node中的struct feature应该是储存用来决定该点的所有特征数据点
这个struct feature结构体内具体应该选择什么样的形式？
如果是std::vector，那后面还可以用malloc(sizeof(struct kd_node))分配内存吗？

如果是二维数组，不得不一开始就确定数组的长度，那对于大型点云数据整个kd树建立过程内存开销又太庞大大了。所以我觉得这两种都不合适啊。

最近被导师盯着做点云的k近邻，我导师说了，寒假前如果做不出，就不用回家过年了。。。跪求指点啊！！

Re: [ai543064193](#) 2015-06-30 17:14发表



回复heroalone2012: 您现在理解了吗? 我也是遇到这个问题了 求指点 ~不胜感激~

26楼 [visionfans](#) 2012-12-28 00:15发表



好文章，加油

25楼 [v_JULY_v](#) 2012-12-17 11:56发表



预告

本文之后，待写的几篇文章罗列如下：

- 1、数据挖掘中所需的概率论与数理统计知识，估计年底前完成；
- 2、机器学习中相关的降维方法，如PCA/LDA等等，明年年初完成；
- 3、神经网络入门导论，明年1月内完成；
- 4、程序员编程艺术第二十八章，时间待定；
- 5、...

欢迎期待。July、二零一二年十二月十七日。

24楼 [swuxd](#) 2012-12-11 20:29发表



非常好评论一下

Re: [v_JULY_v](#) 2012-12-12 23:13发表



回复swuxd: thank you, 文章2.6节、kd树近邻搜索算法的改进: BBF算法 刚做了再次修改, 有何问题, 欢迎各位读者不吝指出, 谢谢。

23楼 [lvjun93](#) 2012-12-10 19:41发表



楼主，你在写完svm和knn之后，能否解释一下2012百度机器学习与数据挖掘校招的笔试题最后一题??

3.一个厂商准备上线一个在线图片分类算法，数据量大，服务器资源有限，查询时间需要快一些，请在linear SVM, logistic regression, discriminate regression等算法中选择一个实现，并说明理由和分析，并与k-NN进行优势与劣势比较。

Re: [v_JULY_v](#) 2012-12-11 00:28发表



回复lvjun93: 建议你先去了解一下这几个算法:

linear SVM: http://blog.csdn.net/v_july_v/article/details/7624837;

logistic regression..;

discriminate regression..;

k-NN: http://blog.csdn.net/v_july_v/article/details/8203674。

22楼 [empty16](#) 2012-12-09 13:01发表



楼主博文观点思路很明确，学习目标也很清晰，目标坚定，道路清晰。给我们这些学习之人带来很多思路，讲解了很多好多算法，谢谢楼主

21楼 [lovingdiana](#) 2012-12-05 15:39发表



正好在弄SIFT算法和KD-TREE搜索算法，楼主的博文实在是对我帮助很大，谢谢哈！期待您更好的博文，不知道楼主有没有关于SIFT改进算法的研究？

Re: [v_JULY_v](#) 2012-12-05 19:29发表



回复lovingdiana: SIFT改进算法，以后有机会研究下:-)

20楼 [jAmEs_](#) 2012-12-04 17:31发表



很强大，要看需要慢慢消化

19楼 maxiao001 2012-12-01 20:51发表



楼主确实全程高能。

18楼 yangyangfuture 2012-12-01 18:46发表



2.确定: Node-data = (7,2)。具体是: 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域位数据点(7,2)。这样, 该节点的分割超平面就是通过(7,2)并垂直于: split=x轴的直线x=7;

这段中, 提到6个数据的中值是7, 中值是中位数的意思吧? 应该是6吧! ?

Re: v_JULY_v 2012-12-01 22:03发表



回复nmcfyangyang: "6个二维数据点{(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)}, 根据x维上的值将数据排序, 6个数据的中值为7"

这里所谓的中值非中位数的意思, 而是中间大小的值, 6个数据点按照x维上的值排序后, 是{(2,3), (4,7), (5,4), (7,2), (8,1), (9,6)}, 所以中值为点(7,2)中的7 (不存在x值为6的点)。

Re: FreeApe 2016-06-05 21:55发表



回复v_JULY_v: {(2,3), (4,7), (5,4), (7,2), (8,1), (9,6)}
5和7都是中数, 可以选择5?

17楼 天才在左疯子 in 右 2012-11-29 21:14发表



有点小深奥

16楼 meiguang1028 2012-11-29 15:00发表



最近也在做关于应用knn算法的研究, 读了一下博主的文章, 感觉把KNN算法介绍的非常通俗。但是我觉得可以再介绍一些nonparametric的内容, 毕竟knn是非parametric的一种方法, 以及knn为何有效 (也许这部分内容需要一些数学证明, 为了通俗, 所以博主省略了)。还有就是距离的选择应该是根据应用场合而定, 个人感觉是这样的。所以觉得没有比较利用这么大的篇幅来介绍。最后想说的是, 非常喜欢博主写的文章, 以后多向博主学习, 希望看到更多的技术文章。

Re: v_JULY_v 2012-11-29 20:26发表



回复meiguang1028: 非常好的建议, 谢谢! 我也了解下nonparametric, 你有相关的资料介绍么?

Re: meiguang1028 2012-11-30 13:19发表



回复v_JULY_v: pattern classification, 很经典的一个教材, 里头nonparametric章节介绍的比较详细。

Re: v_JULY_v 2012-11-30 14:23发表



回复meiguang1028: 好! 是模式分类那一本书吧, 晚上回去一定好好看下

15楼 v_JULY_v 2012-11-28 21:10发表



一群友欲编译本文源码, 但不知"宏"的定义, 后知刚学编程。另一群友提示说, 学编程要从基础一点一点开始, 不要一上来就搞算法, 偶强烈赞同。自己09年10月C、C++、数据结构, 10年底算法, 12年5月开始DM&ML, 近期遂补数学。何谓基础, 就是你容易忽略的简单的初级的东西, 少说也得花心思好好学3年。

14楼 江南烟雨 2012-11-27 16:07发表



膜拜一下~学习一下, 算法很难, 也很重要~明年找工作之前一定要把这里的算法都自己实现一遍~~

Re: v_JULY_v 2012-11-27 18:28发表



回复xiajun07061225: 期待你自己的实现~

13楼 UESTC_HN_AY_GUOBO 2012-11-26 10:28发表



首先 谢谢博主的劳动
其实在KD tree之前呢 其实还有很多过度的知识介绍

我能力有限，如果博主想深入 可以看看similarity search and metric space approach

还有其实knn算法是很经典了，例如在kdtree 也罢 还是mtree也罢 都会有基于树形结构的knn算法，但是knn算法也是独立的一个算法

很想搞清楚 如果把空间数据进行了kdtree 或者 mtree的索引，然后使用knn 那么与直接使用knn算法在空间数据上 结果会有不同吗？

Re: [v_JULY_v](#) 2012-11-27 10:56发表



回复UESTC_HN_AY_GUOBO：谢谢。

“如果把空间数据进行了kdtree 或者 mtree的索引，然后使用knn 那么与直接使用knn算法在空间数据上 结果会有不同吗？”

我觉得你的问题等价于：索引有什么用？

12楼 [v_JULY_v](#) 2012-11-26 00:19发表



全文第二次修订完毕，补充了kd树的插入，删除情况的图例分析。

11楼 [BestChenqi](#) 2012-11-24 17:01发表



顶礼膜拜中

10楼 [mandycool](#) 2012-11-24 10:48发表



kd树做高维数据还是很困难，即使加入bbf，现在用的较多的是randomized kd-tree，lz也可添加介绍。将继续关注lz~~

Re: [UESTC_HN_AY_GUOBO](#) 2012-11-26 10:31发表



回复mandycool：现在主要是在kdtree的基础上有了mtree或者mvp-tree

其实关键还是pivot的选择

度量空间中算法还是怎么能减少 距离计算

使用度量空间的法则 过滤距离计算

Re: [mandycool](#) 2012-11-26 15:56发表



回复UESTC_HN_AY_GUOBO：大致查了一下，mtree和mvp-tree是2000年前提出的吧，感觉是比较老的技术了；现在在kd树的基础上，有做基变换的方法，有做pca的方法；根据flann作者的说法，multi randomized kd-tree经过比较是比较高效的了；另外还有在每一层上做聚类的方法，据说效果也不错；如有误，欢迎指正

Re: [v_JULY_v](#) 2012-11-26 18:37发表



回复mandycool：有相关的资料链接或论文么？

Re: [mandycool](#) 2012-11-26 19:31发表



回复v_JULY_v：FAST APPROXIMATE NEAREST NEIGHBORS WITH AUTOMATIC ALGORITHM CONFIGURATION

Re: [v_JULY_v](#) 2012-11-26 19:39发表



回复mandycool：建议你写一篇blog记录下学习心得或体会哈

Re: [mandycool](#) 2012-11-26 19:47发表



回复v_JULY_v：自己比较浮躁，没法写的像lz这么细致，还配图的；等做完毕设看能不能写点吧。。。

Re: [v_JULY_v](#) 2012-11-26 19:56发表



回复mandycool：好的，期待！到时候向你文章学习

Re: [mandycool](#) 2012-11-26 21:07发表



回复v_JULY_v: lz的博文写的很好, 我才该多学习, 哈哈

Re: [v_JULY_v](#) 2012-11-26 10:54发表



回复UESTC_HN_AY_GUOBO: 针对你说的

"关键还是pivot的选择

度量空间中算法还是怎么能减少 距离计算

使用度量空间的法则 过滤距离计算"

你有何体会或使用心得么?

Re: [mandycool](#) 2012-11-26 19:41发表



回复v_JULY_v: 刚看了一下mvp-tree, 是利用三角形不等式来缩小搜索区域的, 不过mvp-tree的目标稍有不同, 查询的是到query点的距离小于某个值r的点; 另外作者test的数据集只有20维, 不知道上百维以后效果如何

Re: [v_JULY_v](#) 2012-11-30 22:25发表



回复mandycool: 补充了球树, VP树, MVP树的相关简介, 资料还是有限, 无法进一步深入..

Re: [mandycool](#) 2012-11-30 23:48发表



回复v_JULY_v: Distance-based indexing for high-dimensional metric spaces
vp-tree和mvp-tree均可参见此篇, 中文资料确实很少

Re: [v_JULY_v](#) 2012-12-01 11:20发表



回复mandycool: 网络还真不给力, 半天下不下来, 打不开

Re: [mandycool](#) 2012-11-26 15:58发表



回复v_JULY_v: 减少距离计算的一个思路是做embedding, 通过不等式排除掉一部分点

Re: [v_JULY_v](#) 2012-11-24 11:44发表



回复mandycool: 好, 非常感谢, 我了解下randomized kd-tree !

9楼 [leifeng457327997](#) 2012-11-24 00:13发表



太深奥了

8楼 [Felven](#) 2012-11-23 16:58发表



最近上研究生学 模式识别, 机器学习, 人工智能, 再加上自己在实验室帮老师搞数据挖掘, 觉得这些技术很不错, 学好绝对有很大帮助。

Re: [v_JULY_v](#) 2012-11-23 17:01发表



回复jj12345jj198999: 研究生有机会在实验室里 做各种实验, 真心幸福丫

7楼 [v_JULY_v](#) 2012-11-23 12:11发表



OK, 本文基本修订完毕。

Re: [v_JULY_v](#) 2012-11-23 16:54发表



回复v_JULY_v: 补充了本文完整源码的下载地址。

6楼 [-无-为-](#) 2012-11-22 13:07发表

顶一下



5楼 [syzcch](#) 2012-11-21 13:07发表



july 还是一如既往的给力

Re: [v_JULY_v](#) 2012-11-22 14:32发表



回复syzcch：多谢关注哦！

4楼 [SuperFC](#) 2012-11-21 09:27发表



KD树：统计、计算、挖掘！

3楼 [铭毅天下](#) 2012-11-20 23:15发表



引用“zhucegepp”的评论：

博主本科？

去了什么公司？

貌似QQ，博主说过。。

Re: [v_JULY_v](#) 2012-11-21 00:16发表



回复wojiushiwo987：NO， :-)

2楼 [zhucegepp](#) 2012-11-20 23:03发表



博主本科？
去了什么公司？

Re: [Mini校长](#) 2012-11-26 10:52发表



回复zhucegepp：我觉得可以去google，现在博主在上海

1楼 [potentialman](#) 2012-11-20 22:53发表



mark一下 明天考linux 考了再静心来看。

Re: [v_JULY_v](#) 2012-11-22 00:47发表



回复potentialman：文章补充了第一部分中的皮尔逊相关系数等距离度量表示法，不知你昨晚来看了没？

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

- 全部主题
- Hadoop

AWS

移动游戏

Java

Android

iOS

Swift

智能硬件

Docker

OpenStack
- VPN

Spark

ERP

IE10

Eclipse

CRM

JavaScript

数据库

Ubuntu

NFC

WAP

jQuery
- BI

HTML5

Spring

Apache

.NET

API

HTML

SDK

IIS

Fedora

XML

LBS

Unity
- Splashtop

UML

components

Windows Mobile

Rails

QEMU

KDE

Cassandra

CloudStack

FTC
- coremail

OPhone

CouchBase

云计算

iOS6

Rackspace

Web App

SpringSide

Maemo
- Compuware

大数据

aptech

Perl

Tornado

Ruby

Hibernate

ThinkPHP

HBase

Pure

Solr
- Angular

Cloud Foundry

Redis

Scala

Django

Bootstrap

