KNN 的各种实现

汪加林

2016 年 12 月 15 日

1 Introduction

本文主要是介绍 KNN 算法的不同实现，包括最原始的 KNN 实现，K-d Tree 的 KNN 实现，Locality Sensitive Hashing 的 KNN 实现。

下面是本说明文档的内容组织结构:

- 代码说明
- KNN 算法的原始实现
- KNN 算法的 K-D Tree 实现
- KNN 算法的 LSH 实现
- 实验
- 类的说明

2 代码说明

本代码主要对代码中的主类 KNN 进行说明。主类 KNN 主要进行了 KNN 算法的原始实现、K-D 树和 LSH 实现的性能对比。

2.1 参数说明

- k: kNN 中的 k，也就是需要寻找 k 个最邻近点
- $minimum$：K-D tree 算法中，构建树的时候，叶子节点包含的样本数目的最大值

- $alpha$: K-D tree 算法中，当查找时，判断节点是否与超球体相交时，用来对超球体的半径进行放缩，$alpha$ 越大，超球体的半径越小，查找效率越高，求的也为近似解。
- h：LSH 算法中, 用来对区域进行划分的线的数目，将空间划分为 $2 * h$ 空间
- $table$：LSH 算法中，hash 表的数目
- $spliStandard$：K-D tree 算法中，节点分裂时选择最优划分节点属性的标准
- $distType$：距离度量的类型

3 KNN 算法的原始实现

3.1 基本思想

KNN 算法的原始实现思路很直接，就是计算所有样本到目标点的距离，取前 K 个。在实现的时候可以用个优先队列去保存最近邻点的信息。

3.2 复杂度分析

复杂度为 $Nlogk$，$logk$ 是用于优先队列插入比较用的。

4 KNN 算法的 K-D Tree 实现

4.1 基本思想

参考李航老师的 $<$ 统计学习方法 $>$。

4.2 如何判断超球体和超平面是否相交

在这里补充一下，李航老师还有很多其他地方讲的很清楚，不过我还是卡在了这个地方。希望给那些和我遇到同样问题的同学一点启示。

判断最直接的方法是算出目标点到超平面的距离，看其是否小于当前的最近的距离。可是，沿着这个思路仔细想想就会发现这个想法有很多问题。

- 计算点的超平面的距离复杂度太高，甚至不可行
- 如果以平面包含的数据点来算的话，就和遍历所有样本数据没有区别了

因此，要换一种思路，上面的想法是想通过计算距离来确认是否相交？那么我们既然确认相交不容易，但是确认一定不相交却是很简单的。因此实际中是采用排除法来进行判断的。下面以二维为例进行一下讲解，假设目标点位置为 (x, y)。

1. 首先判断目标点在 $0 < x < 7, 0 < y < 4$ 这个区域内，计算最邻近距离 $minDist$，回溯到父节点;

2. 接下来排除父节点是否和目标点是否相交，利用的是父节点的划分属性值 $(y = 4)$，可知 $|y - 4| < minDist$，无法排除；

3. 递归至叶子节点 $7 < x < 10, 4 < y < 10$，遍历其中样本，计算、更新最短距离;

4. 回溯到父节点 $7 < x < 10, 0 < y < 10$, 此区域都已访问过，继续回溯到根节点

5. 接下来判断根节点是否相交，利用的是根节点的划分属性值 $(x = 7)$, 可知 $|x - 7| < minDist$，无法排除；

6. 因为根节点的左节点已访问过，因此递归到 $7 < x < 10, 0 < y < 10$ 区域，利用当前节点的划分属性值 $(y = 7)$, 可知 $y - 7 > minDist$，因此，当前节点的右子节点（也就是 $7 < x < 10, 6 < y < 10$ 区域）一定和圆不相交；

7. 遍历至 $7 < x < 10, 0 < y < 6$ 区域，此节点是叶子节点，遍历其中样本，计算、更新最短距离。

4.3 复杂度分析

复杂度和数据的分布有很大关系，$logNlogk - Nlogk$, 最差的时候接近线性，如下图所示。

并且，随着数据维度的增加，复杂度也是急剧下降，呈指数趋势，如下图。

5 KNN 算法的 LSH 实现

5.1 基本思想

对于 K-D tree 算法，在数据维度很高的时候，性能会急剧下降。因此提出了局部敏感哈希（LSH）的算法。可以得到近似最优解，LSH 获得最优解是通过概率保证的，而不是百分之百。

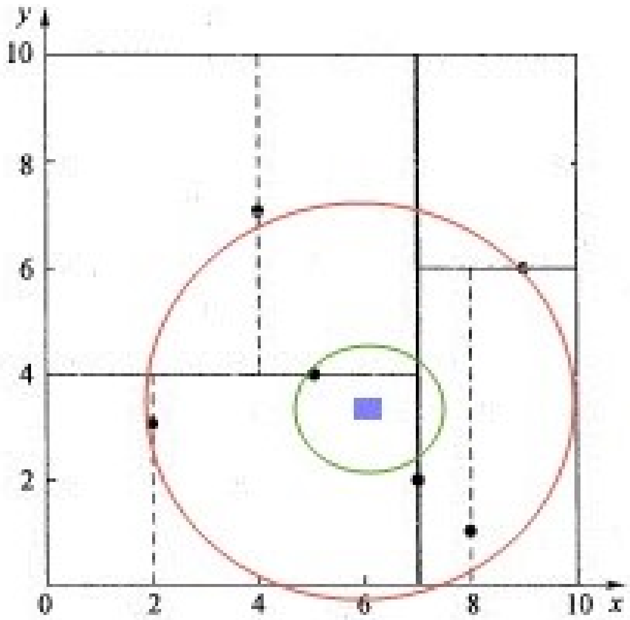

Figure 1: K-D Tree 查找

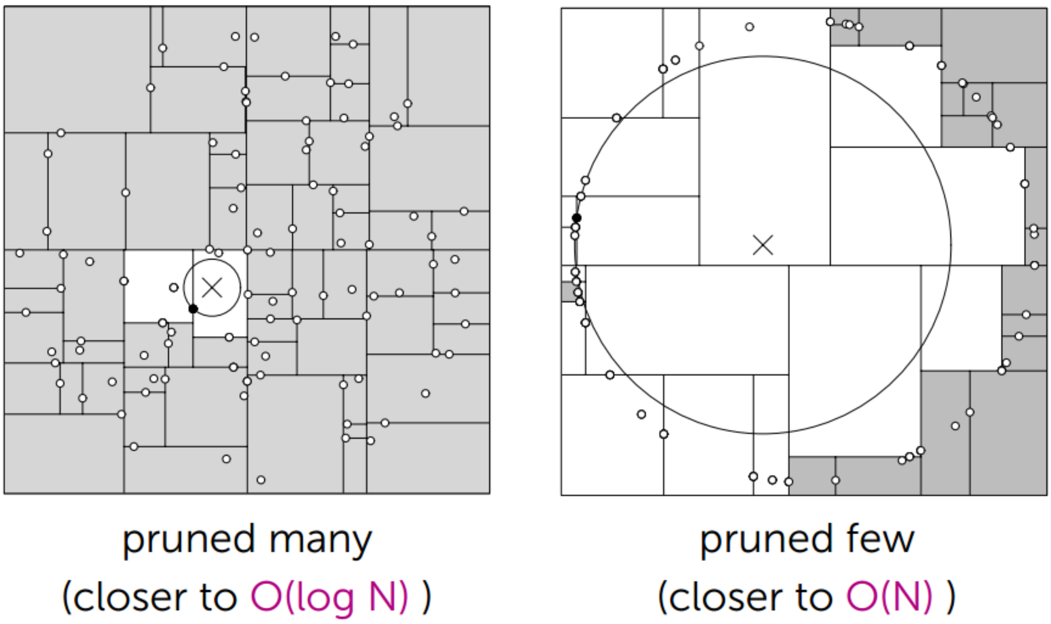

Figure 2: K-D Tree 复杂度相关 1

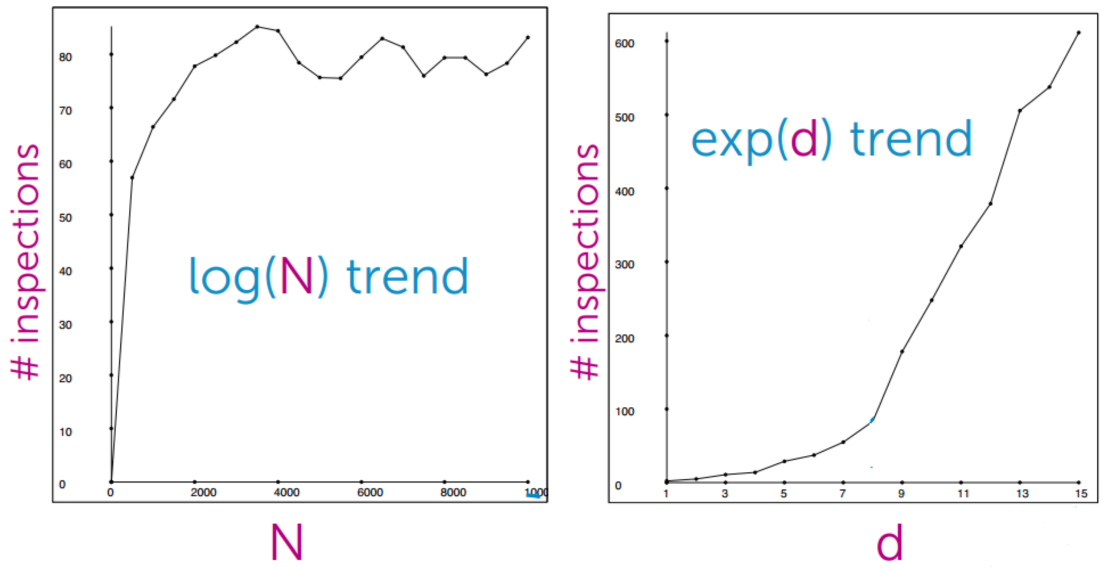

Figure 3: K-D Tree 复杂度相关 2

通过产生超平面将数据空间划分为很多个 bin（区域），并且创建 hash 表; 然后看目标点属于哪个区域。然后在目标点所在区域内通过查询 hash 表查找最近邻点。

那么就会带来下面的问题:

- 如何划分数据空间最好；
- 万一最近邻点不在目标区域呢？

我们应该是想越近的点划分在一个区域越好！但是，神奇的是，我们可以任意产生超平面 (此超平面穿过原点) 对数据空间进行分割，依然能达到不错的效果。如图 4所示，当两个点很近的时候，分在不同区域的概率很低。同时为了提高效率，我们需要产生适当多的超平面进行划分。

因此，有以下两个直观的结论:

- h 越大，划分的区域越多，效率越高, 但被误分在不同区域的概率越大；
- 查询邻近的区域越多，找到最邻近点的概率越大。

每个点所属区域的索引由二进制向量表示，每一位都由一个超平面决定。如图 5所示，将点的数据代入超平面方程，如果大于 0，则该位为 1；反之则为 0。

即使分在不同区域的概率很低，但是依然存在。有下面两种解决方案：

- 创建更多的 hash 表 (由不同的超平面产生)，然后只查询目标在不同的 hash 表中所属的区域；
- 查询相邻的区域。相邻区域的二进制索引可以直接将 $h$bit 的向量中任一位翻转即可得到, 例如可由 $(0,0,1)$ 得到 $(1,0,1)$、$(0,1,1)$、$(0,0,0)$。

1. 产生 h 个超平面划分数据空间
2. 给每个数据点计算在每个超平面所对应的取值，并且转化为二进制的索引
3. 使用 $h$bit 大小的二进制向量来表示每个区域的索引
4. 创建 hash 表
5. 对于查询点 x, 计算所属点的区域，通过 hash 表在该区域查询最邻近点，然后再在相邻的区域中寻找直到达到时间限制。

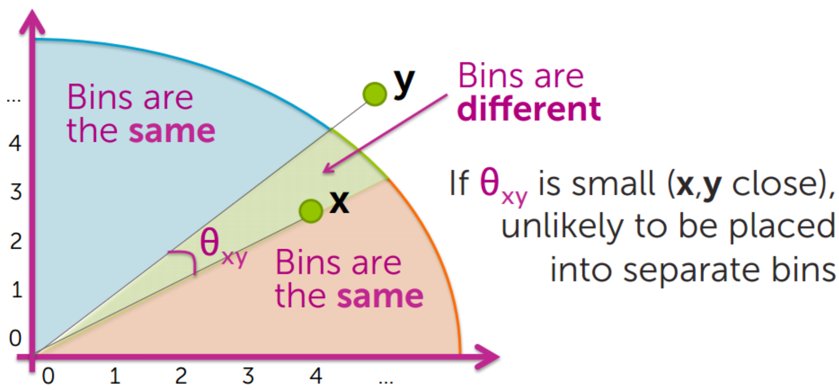

Figure 4: Random Split

5.2 复杂度分析

LSH 的复杂度和 h 和我们想要创建的表的数目有关; 但毫无疑问，非常快，可能牺牲了一点点准确率。

6 实验

15 万组随机产生的 8 维的数据,k=3, KNN 原始方法花费时间 8，K-D tree 花费时间 3, lsh 花费时间 1。

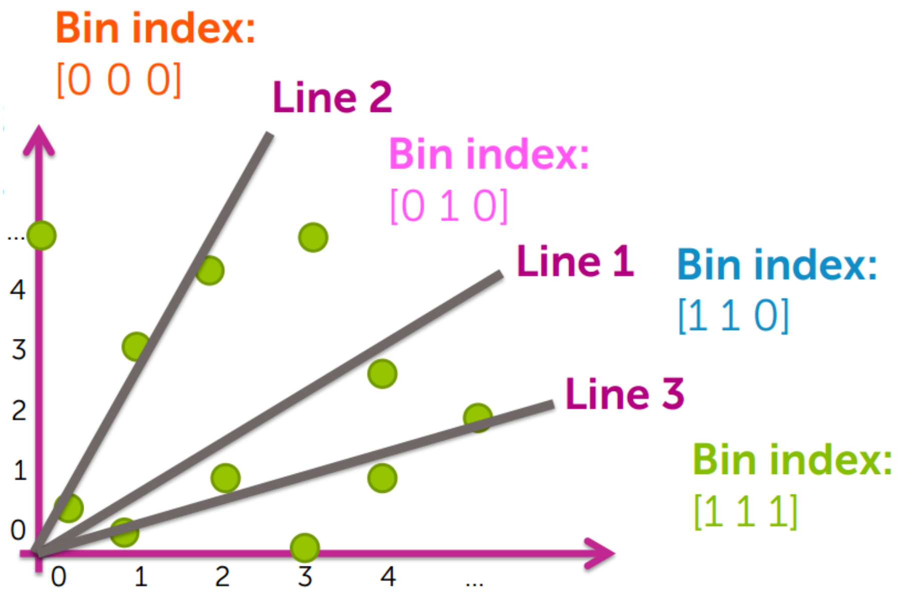

Figure 5: 空间分割

7 类的说明

KNN 主类，主要用于三种方法的性能比较

KNNOrigin 最原始的 KNN 实现

KDTree KNN 的 K-D tree 实现

LSH KNN 的 LSH 实现

NearstPoint 表示最邻近点的信息，索引和到目标点的距离

CalDistance 计算距离的类，有不同的距离度量方式

ChooseSplitDim 构建 K-D tree 时，选择最优的节点分裂属性

Node 表示 K-D tree 中的节点

PriorityQueue 优先队列, 用来保存最邻近点的信息

Queue 队列，用于保存构建 K-D tree 过程中的节点信息

TwoDimE 当选择最优分裂特征时，需要将样本按特征值排序，同时还要记住样本的索引，因此构建 [特征值，索引] 的元素类

TwoDimComp 用于比较 TwoDimE 的大小