通用相似度量的K近邻图构造

摘要

K近邻图构造是许多web应用中的一项重要操作，包括协同过滤、相似性搜索、数据挖掘和机器学习。已经存在的K-NNG构造方法既不能很好的扩展，也不适用于某些相似性度量。本文提出的NN-Descent方法，是一个简单又高效的构建近似K-NNG的方法。我们的方法基于局部搜索，空间开销最小，不依赖任何共享全局索引。因此，它特别适用于数据结构分布在网络上的大规模应用。我们已经通过各种数据集和相似性度量表明，所提出的方法通常收敛到90%以上的召回率。

## 简介

K邻近图是顶点集合V上的有向图，边从顶点 到和它最相近的K个顶点（在给定的度量方式下，比如：余弦距离、L2距离等）。K邻近图构建在许多web相关应用里是一个重要的操作：比如**协同过滤：用相似等级模型连接的用户来构建K-NNG（K临近图），基于活跃用户的邻居来做推荐**；基于内容的搜索系统，当数据集已经固定，离线构建的KNNG比在线的KNN搜索更可行。在数据挖掘和机器学习领域KNNG也是关键的数据结构。

基于暴力方法的KNNG构造只适用于小的数据集（开销O（n的平方））。在KNNG的构造和KNN的搜索上已经投入了大量的研究工作，并且提出了大量的方法，但是已知的方法要么不能上规模，要么限定于特定的度量方法。

Paredes提出了在普通度量空间上具有低复杂度的2种KNNG构建方法，但是这两种方法都需要全局的数据结构和难以跨机器并行化。基于递归数据划分和空间填充曲线，提出了一种计算l2距离的高效的方法。基于递归数据划分和空间填充曲线，提出了一种计算l2距离的有效方法，但是他们不能自然的推广到其他距离度量或者一般的相似度度量。（**也就是说这个方法也不太行**）

K-NN搜索的索引数据是一个被广泛研究的密切相关的开放问题。一个KNN图可以采用重复的调用KNN搜索每个数据集中的对象的方法来构建。为一般度量空间和欧氏空间设计了各种基于树的数据结构。然而，他们都有上面提到的扩展性的问题。局部感知哈希（LSH）是在近似KNN搜索上比较有前途的方法。这些哈希算法为了一系列不同的度量方法而设计，包括汉明距离、 距离 、余弦相似度。然而要完成精确的估算，这些开销仍然很大，为新的相似性度量设计一个高效的哈希函数是非常重要的。(**就是说LSH方法也不太行**)

在文本检索社区，提出了基于预过滤的神经网络图构造方法，也就是说，所有相似性搜索或者相似性连接。NN图不同于KNNG，它是在所有相似点之间建立无向边。当NN图通常是非常稀疏和不连续的时，这些方法在具有紧密相似阈值的情况下是有效的。（**这一段不知道说的好像也没啥用**）

因此，高效的KNNG构建仍然是一个开放的问题，已知的解决办法还没有做到通用，高效和可扩展。在这篇论文中，我们提出了NN-Descent，一个简单又高效的KNNG构建算法，满足这些问题的需求，这个算法有如下一些特征：

* 通用性。 我们的方法适用于任意相似oracle--------- 一个产出两个对象相似度打分的方法。
* 扩展性。随着数据集大小的增长，我们的方法只能看到召回率的轻微下降。对于我们实验室的所有数据集，计算的开销大概是O()。 我们的方法主要对每个数据项局部的信息进行操作，本质上适合于像MapReduce这样的分布式计算环境。（就是数据集变大性能也好，也可以分布式搞。）
* 空间高效。原则上，唯一需要的数据结构就是一个近似的KNNG，也是最后的输结果。我们的方法可以迭代的改进现有的图。对于优化，或者在分布式的实现中，最小化额外的数据。（后面这句话估计就是说节省空间）
* 速度快，并且精度高。我们用真实的数据集证明，我们的方法通常收敛到90%以上的召回率。
* 易于实现。我们的单节点实现了本文提出的所有的优化方法，只有200行左右的C++代码。不包含I/O和评估代码。（看来还有多节点分布式的实现方法）

我们比较我们的方法和2个已经存在的方法： Recursive Lanczos Partitioning and Locality Sensitive Hashing。结果就是我们的方法好呗。

## 2. 方法

### 2.1 符号和背景

V是数据集， N是数据集的大小。 N = |V|。

相似度度量： 。 （这里应该表示的是V中的节点到另外一个节点的距离？）

对于V中任意的点v， ， 用 代表v的K-NN。比如，K个和v最相近的节点。

**,** 这个就是反过来，表示这个点v属于那些点的K最邻近的点。

用B[v] 和 R[v] 来存储和 中的数据。

**,** 表示这两个数据合起来的，称为v的**一般邻居**。

采用堆的数据结构来组织，更新开销O(logK)

**距离度量d**

用来表示，距离越近，表示相似度越大。也就有了

表示距离在r之内的点的集合

**,**

满足存在一个c满足如下条件，则称**度量空间V是受限**。

对于任意的v，都有

满足这个条件的，最小的c，叫做V的**生长常数**。C是维度概念的概括，并抓住了数据集的复杂性。（**这么看这个c肯定是存在的，c的值越小，表示数据分布越均匀**。）

### 2.2 基本的算法

我们的算法基于以下的基本原则：一个邻居的邻居很有可能是邻居。换一句话说，如果我们有一个近似的k-NN的每个点，那么我们可以通过探索每个点的邻居的邻居来改善这种近似估计。

当V是一个增长受限的度量空间时，这个观测值可以通过下面的启发式参数来量化。

c表示是V的生长常数，定义**K**=

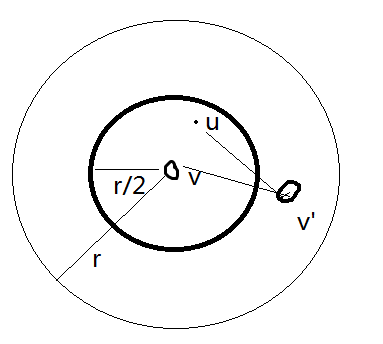
假设我们已经有一个近似KNNG叫做B。

对于每一个  **,** 定义 (**这里就是k邻居的k邻居**) 是我们要去探索然后提升B的点的集合。

如果B本身的精度已经相当好，对于一定的固定半径r，对于所有的**，** 包含的K个邻居都均匀的分布在中，然后假设某些事件的独立性和 ， 我们可以得出结论很可能包含K个在中的邻居。（大概意思就是如果这个图精度已经比较好的情况下，邻居的邻居应该还是在一个比较小的范围内。）

换句话说，我们希望通过探索V中每个节点v的使近似K近邻的最大距离减半。（**估计的意思是之前的图的K近邻不一定是那么好的，可能不是很集中，经过探索更新一遍之后，K近邻会更加集中在一个小的范围之内。**）

对于任意的在能在 能找到的，又在中能找到的点u，(就是对于v来说，又在v的r/2的半径距离之内，又属于v的邻居的邻居u)，我们至少能有一个，使得 且 。（这里其实说的就是v的邻居的邻居的意思。）。



**这里的可能在r/2的外面也可能在r/2的里面**

任何在r/2范围之内的有可能满足这些需求：

1. 也在v的半径r的范围内，于是 Pr{} >= K / || 。 (这里的意思是 **在v的半径为r的范围内的时候， 属于 v的邻居的概率 大于等于 K/（v的半径为r的范围内的节点的个数）**)（感觉这里的意思就是点在这里分布的比较均匀）
2. d(u,) <= d(u,v) + d(v, ) <= r, 于是Pr{ 。（这里的意思是u到的距离，小于等于 u到v的距离加上 v到的距离。 然后也小于等于r，这里为啥小于等于r不太理解，不只是这里的含义是假设小于等于r还是肯定小于等于r，**应该是假设**。 **大概意思就是，当u和 离得比较近的情况下，小于r的情况下，u属于 的邻居的概率大于等于 K/的范围是r里面的节点的个数）**）
3. ，并且 。（这里的意思是，前面一个表示是和之前的度量空间受限的概念是一样的， 后面的含义感觉也是一堆假设。）

（感觉前面这三点似乎都是假设。。。。）

整合1-3，并且假设独立性，我们可以得到：

(这里的意思是**u属于v的邻居的邻居的概率**，大于等于 K除以 v的半径为r/2范围内的节点数量的平方，这些理论都不知道为啥是这样)

总共，对于我们有个候选点， 于是：

(意思是说u属于v的邻居的邻居的概率，约等于 K除以 v半径是r/2之内的元素的个数)**但是上面又是平方，这里对自己对前面的概率的理解有点错误了，这个问题先放到这里吧。**

整个数据集的直径用 表示。上面的启发式论证表明，只要我们选择一个足够大的K（依赖生长常数的值），甚至我们从一个随机的KNN图开始，通过探索每个顶点的邻居的邻居，我们有可能在半径为的范围内，找到每个节点的K项。这个过程可以重复以进一步缩小半径，直到找到最近的邻居.

我们基本的NN-Descent算法在**算法1**中描述。我们从一个随机近似估计KNN开始，通过不断的迭代，比较每个节点和他邻居的邻居，包括KNN和反向的KNN，来提升近似估计。当没有再提升的时候，停止迭代。

近似KNN图可以看做是KN函数，每个函数是n个对象和它的k个nn的距离。算法简单的模拟梯度下降的方法来最小化KN函数，因此叫NN-descent。 但是不像常规的梯度下降方法那样只是探索很小的固定半径内的邻居，我们的函数定义在离散的V集合，我们探索的半径由上一次迭代的近似KNN图来决定。实际上，半径的值从一个很大的值开始，初始近似值是最近形成的，随着迭代的次数增加会慢慢缩小（我们在半径范围内检查的对象数量大致相同，**这里就有点奇怪了，半径缩小了，对象数量还相同？**）。通过迭代逐步缩小搜索半径的思想类似于小世界网络的分散搜索。（**啥叫小世界网络？**）。大多数点可以在几次迭代中达到真正的KNN。

基本算法已经在许多数据集上表现得非常好。在实践中，它可以通过多种方式进行改进，如本节其余部分所述。

### 2.3 局部连接

给定一个点v和他的邻居们，在上的**局部连接**就是计算里面每一对的p,q的相似度，并且用这个相似度更新B[p]和B[q]。每个点探索其邻居的邻居的操作可以通过每个点邻域上的一个局部连接来实现。

a -> b -> c， 代表b属于a的K最邻近，c属于b的K最邻近。（方向不重要，因为还考虑反向KNN）

在基本的算法中，我们比较a和c两次，当在a或者c周围探索的时候比较一次（只有当a>c才可以避免比较）**（这里没太看懂，为啥比较2次）。** 同样的，在a和c之间比较由上的局部连接保证。（**这里也不懂，为啥局部连接保证比较**）

即使计算量保持不变，局部连接极大地提高了算法的数据局部性，使算法的执行效率大大提高。假设的平均大小是，在基本算法中，探索每个点的邻居接触到了个点，在每个点上的局部连接，相反的，只接触到了个点。（这里是为啥也没看懂）

对于单机实现，通过提高cache命中率局部连接优化可以提升几个百分点到几倍的速度。对于Mapreduce的实现，局部连接能极大的减少数据在机器之间的复制。

**算法1 NNDescent**

数据： 数据集V，相似度度量： 。， K

结果： KNN 列表B

{

B[v]

(这里的意思是： 对于任意v， 从数据集V中取K个样本 和无穷大关联？？？)

While(1){

得到反向邻居R

得到，是正向的邻居和反向的邻居的并集。

C = 0 // 更新计数

遍历 数据集V中的v{

For( ) {

// 这里的意思是u1 是v的一般邻居，u2是u1的一般邻居

l =  **// l 是v到u2的相似度度量，v到一般邻居的一般邻居的相似度度量**

**c = c+ UPDATENN（B[v], <u2,l>） // 应该是采用u2和l来调整堆B[v]，如果有调整会返回1，没有调整返回0**

}

}

Return B if c=0

// 如果循环之后都没有改变了，就返回B。 （感觉这个算法要遍历很多次啊）

}

}

Function SAMPLE(S, n){

Return 从集合S中取n个样本

}

Function Reverse(B){

(这里的意思是 对于数据集V中任意的v，v如果是某个点u的邻居，则将u加入到R[v]中)

Return R

}

Function UPDATENN ( H, <u,l, ….>){

更新KNN堆H，如果改变了返回1， 否则返回0；

(但是这里u,l为啥没有用上？)

}

### 2.4 增量搜索

随着算法的运行，每次迭代中进入K-NNG的新项越来越少。因此，在每次迭代中执行完整的局部连接是浪费的，因为在以前的迭代中许多的对已经比较过了。我们采用如下的增量搜索策略来避免冗余计算：

* 我们在每个节点上增加一个bool标记，当一个节点插入到列表里的时候，标记初始化为true。
* 在局部连接上，两个节点中，只有当其中有新的节点的时候，才进行比较。当一个节点参与到局部连接之后，就标记为false。

### 2.5 采样/抽样

我们的方法仍然还有2个问题。

1. 当K的值比较大的时候，局部连接的开销比较大。甚至只有KNN中的节点用在局部连接的时候，每次迭代的开销是次相似度比较。（这里的N是啥？）当反向KNN也考虑的时候，因为反向KNN的大小没有限制，开销会更大，
2. 两个点可能都一个以上的其他点连接，当局部连接在他们共同的邻居上进行的时候，会计算多次。

我们使用采样策略来避免这两个问题：

* 在局部连接之前，我们从KNN的项中，抽样出个项，标记为true， 在每个节点进行局部连接的时候用， 。在每次迭代完，只有这些被抽样的节点标记为false。（**只有抽样的点才标记fasle？**）
* 反向KNN列表采用抽样的节点单独构建，并且这些节点标记为false。 这些列表被再次抽样，于是每个最多有个项。（**这是为啥？**）
* 对采样对象进行局部连接，并在采样对象和旧项之间进行局部连接。

注意，在当前迭代中标记为true但未被采样的对象仍有机会在未来迭代中被采样，如果它们没有被更好的近似替换的话。

我们发现，即使只有少量的抽样，该算法也能收敛到可接受的召回率。 虽然根据采样率， 精度和开销都会下降，但是开销会下降的更快。（见4.4.2节）。 参数被用来平衡精度和速度。

### 2.6 提前终止

算法自然终止是KNN图不再提升了。 实际上，在每次迭代中KNN图的更新次数减少的很快。后面几次迭代，没有做啥实际性的工作了。我们用如下的策略提前结束迭代：

我们在每次迭代计算KNN列表的更新次数，当次数小于的时候停止， 是精度参数，我们使用的默认值是0.001

### 2.7 完整算法

在**算法2**中，我们合并了之前提到的4个优化算法，作为完整的NN-Descent算法。在这篇文章中，我们主要感兴趣的是独立的相似度度量。 可以针对特定的相似性度量进行优化。例如，如果相似度度量用的距离度量，三角不等式可以用来避免不必要的计算。

我们的优化不足以确保两个对象之间的相似性只计算一次。完全消除冗余计算需要O()空间的表，这样对于大的数据集，空间开销太大。 空间高效的近似，比如bloom filter，是可能的，但是带有额外的计算开销，将只是对相似度度量计算开销很大的有帮助。**（这里就是说其他的辅助方法也不是太好使？好像也没提出来更好的方法？）**

**算法2 NNDescentFull**

数据： 数据集V，相似度度量： 。， K， ，

结果： KNN 列表B

{

B[v]

(这里的意思是： 对于任意v， 从数据集V中取K个样本 和无穷大关联？？？并标记为true)

while (true){

并行处理V中的每个v{

old[v] = 在B[v]中是false标记的所有项

new[v] = 在B[v]中是true标记的项

标记B[v]中抽样的项为false

}

= old的反转

= new的反转

c = 0 // 更新计数的

并行处理V中的每个v{

old[v] = old[v] Sample(

new[v] = new [v] Sample(

for **u1, u2 new[v], u1 < u2** or **u1 new[v], u2 old[v]** {

l =

// c 和 B[.]是同步的

c = c + UPDATENN(B[u1] ,<u2, l ,true> )

c = c + UPDATENN(B[u2], <u1,l ,true>)

}

}

如果 c <， Return B

}

}

Function SAMPLE(S, n){

Return 从集合S中取n个样本

}

Function UPDATENN ( H, <u,l, ….>){

更新KNN堆H，如果改变了返回1， 否则返回0；

(但是这里u,l为啥没有用上？)

}

Function Reverse(B){

(这里的意思是 对于数据集V中任意的v，v如果是某个点u的邻居，则将u加入到R[v]中)

Return R

}

### 2.8 在MapReduce上的实现

我们的算法在MapReduce下面可以很容易的实现。一条记录的组成包括一个key，和这个key到他的邻居的距离的列表。一次迭代可以由两个MapReduce操作完成：

1. Mapper操作发出输入记录和反向的KNN项；reducer 合并KNN和反向KNN
2. Mapper执行一个局部连接并且发出输入记录和比较的顶点对。（我有时候写的顶点/节点，有时候写的对象，都一个意思）。 Reducer合并每个key节点的邻居们，只保持前K项。

## 实验步骤

这里后面是实验数据，就不写了。

主要有几个相似度的度量方法：

L1， L2： 用在图片数据集Corel、音频数据集TIMIT、3D图像数据集上

Cosine、Jaccard相似度度量： 主要用在文本相关性上

Earth mover’s Distance (EMD): 主要用户基于内容的图片检索上。

**默认参数：**

K： 20， DBLP数据集上用的是50，说是这个数据集上比较能，要提高K的值才能到90%的召回率。

采样率：。 默认1.0，有的实验室用的是0.5，速度会变快，但是精度降低。

终止阈值：。 默认0.001

特征的维度，不是越高越好。推荐20-50