《数据科学与工程算法》项目报告

报告题目：LSH 在共同作者关系相似性搜索上的实践

姓　　名：高宇菲

学　　号：10215501422

完成日期：2023.5.11

摘要 [中文]：

为了解决高维相似性搜索带来的“维数灾难”，近似最近邻搜索被提出。其中局部敏感哈希（LSH）是应用最广泛的近似最近邻搜索技术。本文实现了基于$jaccard$相似度的 LSH 算法，并进行了合作关系相似性搜索的实践。实验证明，只要选择合适的算法参数(r, b)，准确率和查询时间分布可以达到 90%和 0.02s 左右，非常具有实用价值。

Abstract [English]:

In order to solve the "dimensional disaster" caused by high-dimensional similarity search, approximate nearest neighbor search is proposed. Among them, locally sensitive hashing (LSH) is the most widely used approximate nearest neighbor search technique. In this paper, we implement the LSH algorithm based on $jaccard$ similarity, and practice the cooperative relationship similarity search. The experiments prove that the accuracy and query time distribution can reach 90% and about 0.02s as long as the appropriate algorithm parameters (r, b) are chosen, which is very practical.

一、 项目概述

为了解决线性搜索不适用于高维数据的问题，学术界和工业界提出了近似最近邻查找(approximate nearest nighbor, ANN)。在许多不要求 100%准确率的应用中，查找近似结果要比查找精确结果快得多。本文聚焦近似最近邻查找中应用最广泛的局部敏感哈希（locality-sensitive hashing, LSH），在共同作者关系相似性搜索问题上进行实践。将研究者合作关系抽象成无向图，对每个研究者，返回和他邻居 Jaccard 相似度最大的前十个结点。

目前，LSH 主要用于解决信息检索中的最近邻搜索问题[1][2]，但在数据挖掘，模式识别，图像搜索数据压缩等任务中也有应用[3]。使用基于欧氏距离的 LSH 寻找相似子图的研究也已经开展[4]，但是目前还没有发现基于 jaccard 相似度的关系相似性搜索研究。

因此，本项目的科学价值在于使用了基于 jaccard 相似度的 LSH 在共同作者搜索中进行实践，并取得了较好的查询速度，有望投入实际应用，有利于学术交流和发展。

二、 问题定义

给定一个数据集$\mathcal{D}$，该数据集描述了$N \in Z_+$位研究者之间的合作关系，研究者编号为$1\sim N$之间的整数。现在将数据集看做一个包含N个节点的无向图。数据集的每一行有两个数字，代表两个节点之间有一条边。我们的目标是构建 LSH 方案，实现对任意一个查询节点$q \in \mathcal{D}$，能够在可接受的时间 δ s 内返回与该节点的邻居近似相似度最大的TOP_K个节点$\{o_i | \ 1 <= i <= K\}$。本文默认$TOP_K = 10$。

相似度的度量使用 jaccard 相似度。给定两个集合 A 和 B， 两者之间的 jaccard 相似度和 jaccard 距离定义如下：

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

$$dist(A,B) = 1 - Jaccard(A,B)$$

本文符号表

N	结点数量
b	最小哈希签名矩阵划分的组数
r	每组的行数
mh_n	总共哈希函数的数量（r * b）
TOP_K	返回的结点数量
mhtab	最小哈希签名矩阵

三、 方法

本项目使用 Python3 作为编程语言。

3.1 数据预处理

图数据的表示方法一般有邻接矩阵和邻接表。考虑到使用邻接矩阵表示会得到非常稀疏的矩阵，所以使用邻接表表示。具体实现上，使用 Python 字典数据

结构，键是整型，代表一个节点编号；值是包含该节点所有邻居的列表。

3.2 定义哈希函数族

MinHash 哈希函数族定义为

$$H(x) = \min\{\pi(x_i)\}$$

其中$x \epsilon \mathcal{D}$是数据集中任一个数据点，$x_i \in x$，π是随机排列函数[5]。即取每次随机排列后序号最小的元素作为最小哈希签名。该哈希函数可以保证每个元素被选为最小哈希签名的概率相等，即

$$\Pr\big(min\{\pi(x)\} = \pi(x_i)\big) = \frac{1}{|x|} .$$

3.3 构建最小哈希签名矩阵

使用一个(mh_n,N)维的 numpy 二维数组作为最小哈希签名矩阵 mhtab，数组元素为$int32$。$mhtab[i,j]$表示第i个哈希函数作用于结点j产生的最小哈希签名，$1 \le i,j \le N$。我们使用$numpy.random$模块中的$shuffle$函数重排邻居结点，并取重排后排在第一个的元素作为邻居集合的最小哈希签名。

3.4 分组并映射到桶

如图 1 所示，将最小哈希签名矩阵划分为b组，其中每组由r行组成。对每一组，都对应r个哈希函数，由这r个最小哈希值组成的列向量决定一个集合被映射到一个大桶中。因此，这个r维列向量可以看作该集合的一个数组签名，b组对应着该集合的b次数字签名。当两个集合在同一组中的数字签名相同时，它们将被映射到同一个大桶中。最终，任意两个集合只要至少有一次被映射到同一个桶中，就被认为是一对候选相似集合。

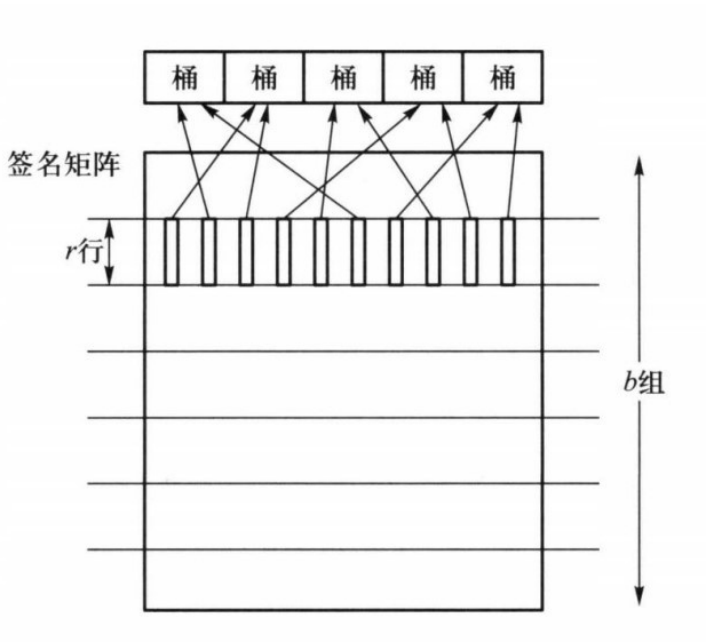


图 1

具体实现时，数字签名使用数组切片转化成的字符串表示，中间用逗号隔开；所有的桶使用一个字典变量$bucket$模拟，键是一个数组签名，值是包含了所有映射到该桶结点的集合。

3.5 在线查询

首先将要查询的结点对应桶中的节点添加到一个候选集（不含查询节点）中，

依次计算查询节点和候选集中节点的$jaccard$相似度，并返回候选集中相似度最高的$Top-K$ 个节点。

具体实现时，候选集用一个空集合初始化。遍历b个组，计算查询节点在b_i组下的数字签名，并将$bucket$ 中键为该数字签名的集合加入候选集。最后从候选集中去除查询节点。

实现$jaccard$函数，参数为两个结点 $n1,n2$，使用集合的交并运算，输出$n1$和$n2$之间的$jaccard$相似度。遍历候选集并计算每个候选节点和查询节点之间的相似度。使用字典$Jlist$保存查询节点与候选集中所有节点的相似度，键是节点编号，值是相似度。然后将$Jlist$按照值从大到小排序，并返回前$Top-K$个(节点，相似度)二元组即可。

四、 实验结果

4.1 实验结果表

以下是 r，b 分布取不同值时的在线查询时间，索引构建时间和准确率。哈希函数个数控制在 10000 以内。

在线查询时间(s)

	r=2	r=3	r=4	r=5
b=1000	0.019	0.005	0.008	0.01
b=2000	0.042	0.0145	0.016	1.8
b=3000	0.036	0.025		
b=4500	0.065			

离线训练时间(哈希签名矩阵构建时间+桶映射时间)(s)

	r=2	r=3	r=4	r=5
b=1000	171.9	211.63	323.56	513.8
b=2000	313.58	451.51	670.05	795.3
b=3000	488.12	656.42		
b=4500	692.66			

准确率

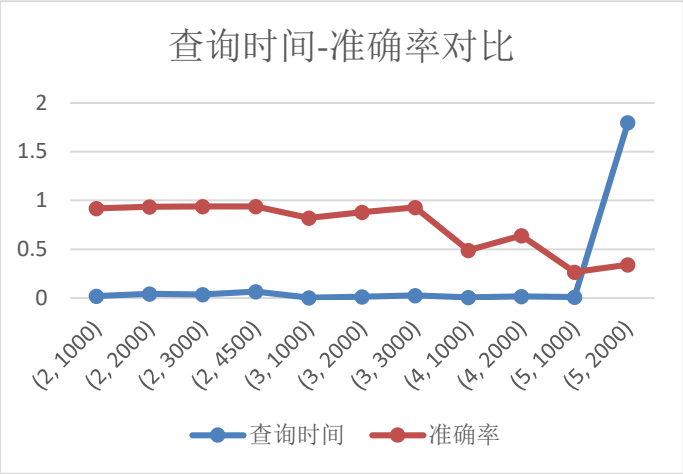
	r=2	r=3	r=4	r=5
b=1000	0.92	0.82	0.49	0.267
b=2000	0.936	0.88	0.64	0.34
b=3000	0.94	0.93		
b=4500	0.94			

内存使用情况：$r=3, b=3000$时，内存占用大小约为 7900MB，内存使用率约为 98%。

表中，准确率测量方法是从所有节点中随机抽取 100 个节点，对每个结点，

计算查询准确率后相加除以 100。单个节点准确率计算使用 LSH 方法查询出来的节点和准确的 Top-K 节点取交集后的个数除以$Top-K$。

将 11 次实验结果的查询时间和准确率绘制成折线图，横坐标是(r,b)二元组。结合上表分析，$r=2$时准确率较高，但是查询时间也更高，原因是$r=2$时近似暴力搜索，LSH 的意义不大，效率仍较低。而r太大会导致哈希到同一个桶中的概率大大降低，错误率上升。可以看出将r，b控制在(3,3000)左右较合理。



4.2 有效性论证

从以上结果表中可以看出，当 r=3, b=3000 时，准确率达到 0.93，查询用时大约为 0.025s，是在实际中可接受的错误率和用时。

4.3 高效性论证

该算法一次查询的时间一定比蛮力搜索更短。因为蛮力搜索需要遍历所有节点，而该算法只需要遍历候选集中的节点。故该方法更高效。

五、 结论

5.1 复杂度分析

建立索引的时间复杂度：建立最小哈希签名矩阵的时间复杂度主要由两个循环组成，是O(N * mh_n)；桶映射的时间复杂度也由两个循环组成，是O(N * b)，因为b ≤ mh_n，所以建立索引的时间复杂度是O(N * mh_n)。

建立索引的空间复杂度：$mhtab$所占空间是O(mh_n * (N + 1))，bucket 所占空间约是O(N + mh_n) = O(N)。

查询的时间复杂度： 遍历一个组的时间是O(1)，所以遍历所有组的时间是$O(b)$；遍历候选集的时间是$O(N)$，排序时间是$O(logN)$，所以查询的时间复杂度是$O(N+logN)=O(N)$。

查询的空间复杂度：候选节点数量小于等于所有节点数量，所以候选集所占空间$O(N)$，$Jlist$所占空间$O(2N)=O(N)$。

5.2 存在的不足

虽然查询时间较短，但是本方法建立索引的时间较长，且随着数据量增加，建立索引的时间也将增加；而且一旦数据有所更改，就要重新建立索引。从这个角度看，本方法效率较低。

5.3 可能的改进方向

本算法可以向缩短索引建立时间的方向改进。具体可以使用 C 语言重写，哈希签名的表示方式也可以不使用字符串，因为使用字符串作为键索引可能使复杂度前的隐含常数较大。如果硬件性能支持，本算法也可以尝试更大的mh_n，以获得更高的准确率。本算法还可以从减小内存使用方面改进。

[1] Paulevé L, Jégou H, Amsaleg L. Locality sensitive hashing: A comparison of hash function types and querying mechanisms[J]. Pattern recognition letters, 2010, 31(11): 1348-1358.

[2] Slaney M, Casey M. Locality-sensitive hashing for finding nearest neighbors [lecture notes][J]. IEEE Signal processing magazine, 2008, 25(2): 128-131.

[3] Jafari O, Maurya P, Nagarkar P, et al. A survey on locality sensitive hashing algorithms and their applications[J]. arXiv preprint arXiv:2102.08942, 2021.

[4] Zhang B, Liu X, Lang B. Fast graph similarity search via locality sensitive hashing[C]//Advances in Multimedia Information Processing--PCM 2015: 16th Pacific-Rim Conference on Multimedia, Gwangju, South Korea, September 16-18, 2015, Proceedings, Part I 16. Springer International Publishing, 2015: 623-633.

[5] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. 2000. Min-wise independent permutations. J. Comput. System Sci.60, 3 (2000), 630–659.