
2020 级

《大数据存储与管理》课程

# 课 程 报 告

# 基于 LSH 的设计和实现

姓　名　刘玺语

学　号　U202015299

班　号　计算机 2001 班

日　期　2023.05.29

# 目　　录

## 一、课题背景

本次课程的选题是基于局部敏感哈希 (LSH: Locality-Sensitive Hashing)的设计和实现。该模型可以近似地求解大数据背景下高维数据的最邻近(NN: Nearest Neighbor)查找问题。

根据个人对相关文献的阅读和实现，该模型的理论大致可以有如下解释：选择合适的哈希函数，使得相似的数据在经过映射后的结果也相近(存放在同一'桶'中)，以此结果作为索引存放数据。这样在查询时只需要经过哈希，从桶中取出远小于规模 N^2 量的数据，大大减少了用于相似性函数的开销。

本次课题实现采用了曼哈顿距离(Manhattan Distance:$\sum_{i=0}^{dim}|a_i-b_i|$)作为向量相似度的参考值。哈希函数的实现基于汉明距离(Hamming Distance:二进制串中不同的位的个数)。

## 二、数据结构

为清晰展示，此处给出各结构的伪代码(详细请参考附录)。

```
Point{
    Int[] vector;
    Int dimension;
}

PointSet{
    Set<Point> points;
    Int size;
    Int max_value;   //向量所有分量中最大的数值；
                     //用于确定映射的二进制串总长
}

HashFunctionCluster{
    Int PointToIndex(Point p){
        binaryString bs;
        for(int c:p.vector){
            //将向量分量 c 映射到含有 c 个 1 的二进制串
            bs.append(CompToBinaryString(p));
        }
        Int index=BinaryToIndex(bs);
        Return index;
    }
}
```

```
HashTable{
    HashFunctionCluster hash_functions;
    HashMap<Integer,List> table_contents;
    Void put(Point p){
        Table_contents.put(hash_functions.PointToIndex(p),p);
    }
    Point[] query(Point p){
        Return table_contents.get(hash_functions.PointToIndex(p));
    }
}

LSH{
    HashTable[] tables;
    Void put(PointSet ps){
        For(Point p:ps){
            For(HashTable ht:tables){ Ht.put(p);}
        }
    }
    Point[] query(Point p,int n){
        List<Point[]> candidates;
        For(HashTable ht:tables){ Candidates.add(ht.query(p));}
        Candidates.distinctAndCalculateAppearTimes();
        Candidates.sortByAppearTimes();
        Candidates=candidates.top(2*n);
        Candidates.calculateDistance(Manhattan,p);
        Candidates.sortByDistance();
        Return Candidates.limit(n);
    }
}
```

## 三、流程分析

映射过程：

哈希函数的实现基于汉明距离(Hamming Distance:二进制串中不同的位的个数)，将向量每一维的分量$a_i$转化为含有$a_i$个'1'的二进制串(将整数值映射到二进制串)。为每张哈希表生成了 k 个哈希函数，对应 2^k 个桶，每个哈希函数在生成时决定了从对应的二进制串中取某一位$s_r$(将二进制串映射到 0-1)，将多次映射后的结果重新组合成一个整数作为哈希索引(将二进制串映射到整数)。以上几步映射后完成了向量到哈希索引的映射。

建表过程：

同时设置有$l$张哈希表，增强哈希的随机性(这是因为该模型是基于选取的哈希函数的特性,以概率保证准确性,利用哈希冲突)。

查询过程：

查询过程中，在各个哈希表中分别计算哈希索引，各个索引指向的桶中的数据汇总，统计各个点出现的次数(在各个不同的哈希函数簇映射下都相近的向量具有更高的可能性相似)，并据此从中选出出现次数最多的一组点作为搜索的候选结果，在这一小范围集合中计算相似度后(曼哈顿距离)取出最接近的 c 个点返回。

## 四、测试分析

系统测试结果如图 4.1 所示。

```
point_set_size: 13107
table_numbers: 512
function_numbers: 30
vector_dimensions: 8
total_block_allocated: 3668535

ACCURATE:
total_time_used: 20
times_calculate_distance: 13107
times_compare_distance: 156116
query_results: [129,111,130,131,88,122,146,114] [153,113,140,127,164,118,131,155]
total_distance: 1218, average_distance: 152.25

LSH Query:
using neglect_weight: 0
total_time_used: 12
query_candidates_list_size: 2790
times_compare_point_rank: 28338
times_compare_distance: 22899
query_results: [129,111,130,131,88,122,146,114] [153,113,140,127,164,118,131,155]
total_distance: 1221, average_distance: 152.625
```

图 4.1 系统运行测试图

分析：

本次运行使用的各项参数为：

数据范围：256*256 点密度：20% 向量维度：8

哈希表： 512 每表哈希函数*30

运行结果：

朴素查询方法(逐点计算距离并根据距离排序)：

运行时间：20ms

距离计算次数：13107 距离比较次数 156116

平均每维度偏移距离：152.25/8=19.03125(偏移半径)

LSH 方法

运行时间：12ms

距离计算次数：2790 权重比较次数(出现次数)：28338

距离比较次数：22889

偏移半径：152.625/8=19.078125

比较：

运行效率(准确率/运行时间)

朴素方法准确率 100%，用时 20ms，效率 1/20=0.05

LSH 方法准确率(偏移半径比值)=朴素法半径/LSH 法半径=99.754%

LSH 方法效率=99.754/12=0.08312

效率优化比=0.08312/0.05=1.663

综上可以得出，LSH 方法在解决该类问题时具有显著的优势，在准确性无较大损失的前提下可以得到 60%以上的性能提升，但也仍需注意额外的空间开销。空间开销达到 数据规模*点密度*哈希表个数。

## 五、课程总结

经过本次实验对大数据背景下的 LSH 存储结构得到了深入的理解和体会，以往在小规模问题中，传统的算法设计课程只涉及了求解精确解的算法，而本次实验基于哈希冲突和概率出发，从另一角度解决 NN 问题，是我在以往的学习过程中所欠缺思考的。经过大数据系列课程，对大数据管理、大数据存储以及大数据分析都有了一定体悟和兴趣。

# 参考文献


• F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, “Beyond Bloom Filters: From Approximate Membership Checks to Approximate State Machines,” Proc. ACM SIGCOMM, 2006.

• Y. Zhu and H. Jiang, “False Rate Analysis of Bloom Filter Replicas in Distributed Systems,” Proc. Int’l Conf. Parallel Processing (ICPP ’06), pp. 255-262, 2006.

• S. Dharmapurikar, P. Krishnamurthy, and D.E. Taylor, “Longest Prefix Matching Using Bloom Filters,” Proc. ACM SIGCOMM, pp. 201-212, 2003.

• L. Fan, P. Cao, J. Almeida, and A. Broder, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol,” IEEE/ACM Trans. Networking, vol. 8, no. 3, pp. 281-293, June 2000.

• B. Xiao and Y. Hua, “Using Parallel Bloom Filters for Multi- Attribute Representation on Network Services,” IEEE Trans. Parallel and Distributed Systems, vol. 21, no. 1, pp. 20-32, Jan. 2010.

• Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, “Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems,” Proc. 28th Int’l Conf. Distributed Computing Systems (ICDCS ’08), pp. 403-410, 2008.

• D. Guo, J. Wu, H. Chen, and X. Luo, “Theory and Network Application of Dynamic Bloom Filters,” Proc. IEEE INFOCOM, 2006.