大数据存储系统与管理报告

姓　名：　韩茂卿

学　院：　计算机科学与技术学院

专　业：　数据科学与大数据技术

班　级：　大数据 2102

学　号：　U202115354

指导教师：　施展

分数	
教师签名	

2024 年 4 月 21 日

目 录

1 数据结构的设计

1.1 LSH 概述

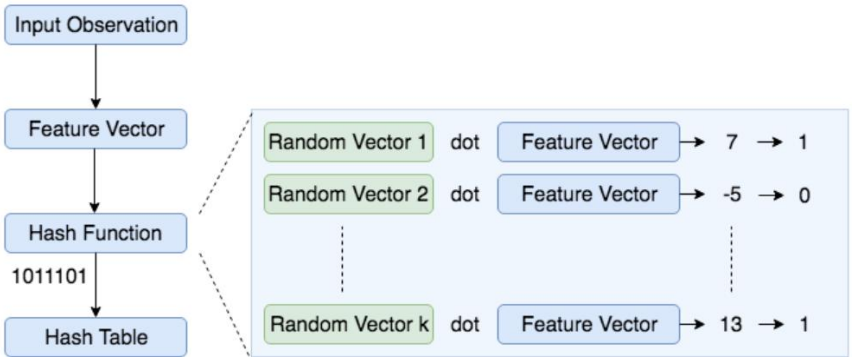
LSH 是一种基于哈希的算法，内核认为更相似的点会哈希到同一个桶中，即如果特征空间中有两个点彼此靠近，它们很可能具有相同的哈希值，如果 a 和 b 靠近，则 Pr（h（a） == h（b）） 为高，如果 a 和 b 相距较远，则 Pr（h（a） == h（b）） 为低，识别近距离物体的时间复杂度是亚线性的。

LSH 与传统哈希的主要区别在于，传统哈希试图避免冲突，但 LSH 旨在最大化相似点的冲突。在传统哈希中，对输入的微小扰动可以显着改变哈希，但在 LSH 中，轻微的失真将被忽略，以便轻松识别主要内容。哈希冲突使相似的项目具有相同哈希值的可能性很高。

而为了减少误判，提高准确率，往往采用多个 hash function，也因此会极大提高空间使用率。

1.2 数据结构

本实验采用的是通过随即映射得到哈希码来代替分桶，以此减少空间开销，将输入与随机向量相乘，然后得到的位值的正负来确定输入哈希码位值是 1 还是 0。通过与多个随机向量相乘，得到多个哈希表，这些哈希表的值进行与操作和或操作，以此来降低误判率。



```
class HashTable:
    def __init__(self, hash_size, inp_dimensions):
        self.hash_size = hash_size
        self.inp_dimensions = inp_dimensions
        self.hash_table = dict()
        self.projections = np.random.randn(self.hash_size, inp_dimensions)

    def generate_hash(self, inp_vector):
        bools = (np.dot(inp_vector, self.projections.T) > 0).astype('int')
        return ''.join(bools.astype('str'))

    def __setitem__(self, inp_vec, label):
        hash_value = self.generate_hash(inp_vec)
        self.hash_table[hash_value] = self.hash_table\
            .get(hash_value, list()) + [label]

    def __getitem__(self, inp_vec):
        hash_value = self.generate_hash(inp_vec)
        return self.hash_table.get(hash_value, [])
```

构建多个哈希表的 LSH:

```
class LSH:
    def __init__(self, num_tables, hash_size, inp_dimensions):
        self.num_tables = num_tables
        self.hash_size = hash_size
        self.inp_dimensions = inp_dimensions
        self.hash_tables = list()
        for i in range(self.num_tables):
            self.hash_tables.append(HashTable(self.hash_size, self.inp_dimensions))

    def __setitem__(self, inp_vec, label):
        for table in self.hash_tables:
            table[inp_vec] = label

    def __getitem__(self, inp_vec):
        results = list()
        for table in self.hash_tables:
            results.extend(table[inp_vec])
        return list(set(results))
```

2 操作流程分析

1. 将 n 个输入归一化，得到[n,k]维向量
2. 创建[x,k]的随机向量，其中是 x 是哈希码的长度，即等价于哈希桶的个数为 2^x 个，k 是特征向量的维度。
3. 将输入向量与随机向量的转置点乘，计算点积，即：[n,k]向量点乘[k,x]随机向量，从而得到[n,x]向量结果，也就是 n 个输入对应的 x 维哈希码。如果点积的结果为正数，则将位值赋值为 1，否则为 0。
4. 产生新的随机向量，重复，得到多个输入与哈希码的对应，将这些哈希码相异或从而减少 value 的个数。
5. 将减少后的哈希码部分相与部分相或，得到一个输入最终的哈希码。
6. 将相同的哈希码的输入归于一类，得到结果。

3 理论分析

为减少空间开销，将一个输入通过不同的随机函数得到的哈希码相异或，从而将多个哈希码减少为一个哈希码，减少空间开销。

同时为保证准确率，将哈希码按位相与，即当且仅当多个哈希码都满足相等，也就是说两个输入的多个哈希码都对应相同时，才会被投影到相同的桶内，只要有一个不满足就不会被投影到同一个桶内，从而降低 false negtive。

相应的，将哈希码按位相或，也就是只要两个输入的这 k 个 hash 值中有一对以上相同时，就会被投影到相同的桶内，只有当这 k 个 hash 值都不相同时才不被投影到同一个桶内。从而降低 false postive。

4 实验性能

将 k 个哈希函数的结果向量进行相异或，使得原本的 k 个哈希结果归一，实现高维到低维的降维，减少了空间占用。

但是相应的，在减少空间的同时也会带来准确率的下降，增加合并的向量结果个数，会减少空间使用，增加用来相与和相或的结果向量会增加准确率，减少误判，因此选择合适的用来合并的向量数量，能够找到最优解。