高级算法设计与分析 课程实验报告

实验名称 Min-Hash算法的实现

小组成员：马浩源 黄钰竣 李玲玲 黄铎峻 马杰

一、实验要求

用C/C++ 语言实现Min-hash算法的程序。更换位置敏感哈希的参数b和r，记录实验结果并绘制图表。根据要求计算相应的哈希次数，并通过实验验证计算结果

二、实验内容及实现步骤

2.1随机文档的生成

为了生成Jaccard 相似度应尽可能分散的1000篇文档，我们采用随即替换的方法

* 1. 按照均匀分布的方式从[a~zA~Z]中提取随机字符，重复 500 次，形成文档A1
  2. 随机挑选A1中的某个字符，替换成另一个随机字符，得到文档A2
  3. 以同样的方式处理A2，得到A3，同理得到A4，A5，…，A1000
  4. 我们将生成的1000篇文档以txt的格式进行存储

证明：这种方法生成的随机文档满足“Jaccard相似度尽可能分散”：

* 由于每篇文档大小为 500 字符，且以 8 个字符为一个单位生成shingle，故每篇文档产生 500-8+1=493个shingle。
* 我们设A1和A2是完全相同的两篇文档，故他们产生的 shingle 应完全相同，即他们的 shingle 集合的交集的基数为 493，并集的基数也是493，故他们的 Jaccard 相似度为 493/493=1
* 此时我们修改A2中的字符为另一个随机字符，这就导致了A2产生了至多8 个不同的 shingle，这些新的 shingle 是A1中没有的，故此时他们的交集的基数应为 493-8=485，并集的基数为 493+8=501，此时他们的 Jaccard 系数为 485/501=0.968
* 同理可以计算出A1和A3的相似度为 477/509=0.937，A1和A4的相似度为 469/517=0.907……

故我们生成的文档对会有尽可能分散的相似度

2.2 shingle实现

Shingle一共可以分为两步：

（1）类似于滑动窗口，使用一个可存储8个字母的buffer，以步长为1滑动读取整个文档，将每一步读取到的8个字母存入shingle集合；

（2）将shingle集合中的元素逐一使用CRC32算法映射为32位整数，最终得到所有元素为均为32位整数的shingle集合。

代码实现比较简单，第一步只需调用c++的输入流处理函数，以8个字节（每个单词占1个字节）为单位读取文档。为了实现类似滑动窗口的功能，只需在每次读取后，将文件指针向前移动7个字节，从而使得下一次开始读入的位置与本次读入的位置恰好相隔1个字节，如此便实现了步长为1的滑动窗口读取文档。第二步只需调用CRC32算法将第一步中的每一个元素映射为32位整数即可。

但此处有一个地方需要注意，为了方便后续程序利用shingle集合（如计算相似度矩阵等一系列操作），我们需要将shingle集合按照文档的顺序依次存入列表中（如按A1,A2,A3的顺序读取、处理、存入列表）。因此，实现时最好使用for循环依次按照文件名中数字从小到大的顺序读取，从而保证顺序。但若是使用先获取目录下所有文件名，然后再读取的方式，则需要留意读取到的多个文件名是否是随机排列的。

2.3计算Jaccard相似度

计算1000个文档两两之间的jaccard相似度，生成相似度矩阵，关键在于利用两个文档的shingle集合计算对应特征矩阵中0-1对和1-1对的个数。计算思路主要有两种：

（1）将所有文档的整数shingle集合转化为01序列。在计算两个文档间相似度时，统计两个文档01序列之间的0-1对和1-1对的数量，最后利用统计结果计算得到jaccard similarity。

（2）使用集合运算，计算两个shingle集合之间交集和差集，交集的基数对应1-1对的个数，差集的基数对应0-1对的个数，从而计算出jaccard similarity。

相比于第二种方案，第一种方案获得0-1对和1-1对需要经过两步，即生成序列以及统计两个序列中0-1对和1-1对的数量。这样的两步操作会增加不必要的时间花销（如生成01序列时需遍历0~2^32，统计数量时需遍历整个01序列），因此，我们最终使用了第二种方案计算jaccard相似度。

代码实现同样比较简单，可直接调用集合运算的函数获取交集和差集，再分别获取交集和差集的基数即可获取0-1对和1-1对的数量。

2.4通过min-hash，将shingle文件转换成签名矩阵

本文的min-hash的生成没有像问题描述文档中那样生成特征矩阵保存每个文档所有存在的shingles元素。考虑到每个文档的shingles个数-1，所以为了减小内存开销，在本次作业中实现Min-hash的时候没有生成特征矩阵遍历，而是通过遍历文档中的shingles元素实现。生成一个签名矩阵的实现伪代码为：

伪代码：

初始化签名矩阵的所有元素为unsigned int数值类型最大值

确定一个比-1大的素数=4294967311

遍历每一个哈希函数

生成unsigned int型作为此哈希函数对应的固定系数

遍历每一个文档

遍历文档中每一个shingle元素

根据公式计算经过哈希函数变换之后得到所在的新行。

如果 < 矩阵中哈希函数和文档对应的值，将矩阵中对应的值更新为。否则继续遍历

2.5 Local-Sensitive Hashing实现

基于前面的签名矩阵，这里通过位置敏感哈希（Locality-Sensitive Hashing）将其中碰撞的文档对挑选出来，加入到候选文档对中。

签名矩阵的大小为30\*1000，将签名矩阵划分为b个bands，每个bands有r个分量。这里b、r取不同的值共进行6组实验：（2，15），（3，10），（5，6），（6，5），（10，3），（15，2）。当两篇文档的签名片段完全相同时，这两篇文档才发生碰撞。这里是将band中的分量拼成一个字符串，通过字符出两两比较挑选出碰撞的文档对。

三、实验结果与分析

3.1对六组（b, r）进行1000次实验，得到以下实验结果：

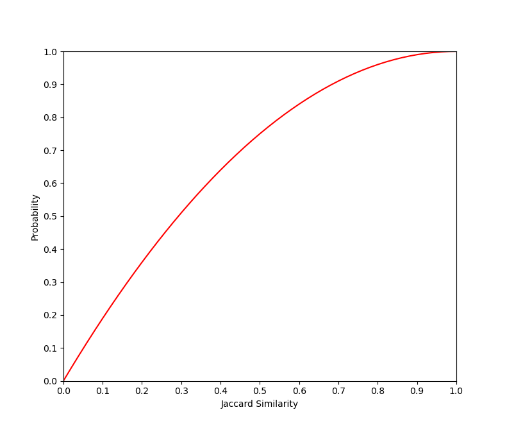
|  |  |
| --- | --- |
|  |  |
| b = 2, r = 15 | b = 3, r = 10 |
|  |  |
| b = 5, r = 6 | b = 6, r = 5 |
|  |  |
| b = 10, r = 3 | b = 15, r =2 |

从上图中可以看到，文档对的Jaccard相似度分布在[0, 1]中，且在六组实验中，文档对的碰撞概率与理论计算的碰撞概率基本一致。

3.2要求Jaccard相似度为80%的文档被选到候选集中的概率为95%，请计算哈希多少次能达到实验预期，b和r应该设置为多大？

要满足上述条件，只需要将r设置为1，增加b的取值。取r = 1， b = 2， 理论概率为：

此时Jaccard Similarity-Probability曲线如下：



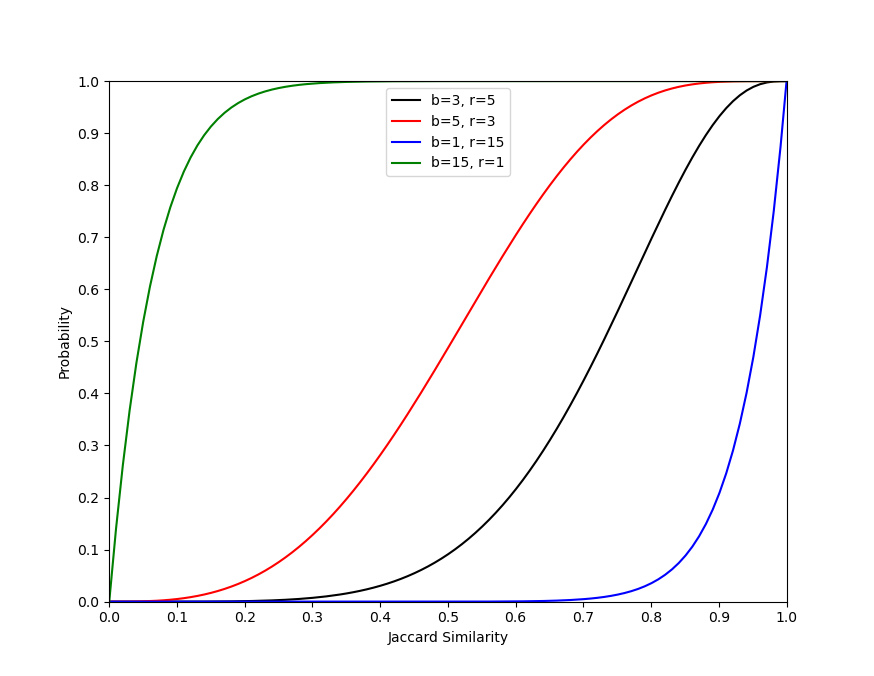
取r = 1， b = 2的Jaccard Similarity-Probability曲线

从上图可以看出，当Jaccard相似度较低时，仍然有较大的概率被选到，导致假阳率比较高，所以应该适当增大r的取值。不妨增加条件：Jaccard相似度小于20%的文档被选到的概率小于5%，即：

取b = 5, r = 3,此时当Jaccard相似度为80%时，被选中的概率为：

当Jaccard相似度为20%时，被选择的概率为：

此时满足条件，哈希次数为15。



当哈希次数为15时的Jaccard Similarity-Probability曲线

四、创新优化

考虑到每个文档的shingles个数62≪2^32-1，所以为了减小内存开销，在本次作业中实现Min-hash的时候没有生成特征矩阵遍历，而是通过遍历文档中的shingles元素实现。