基于 LSH 的设计与实现

算法原理分析：

设计 LSH 算法的目标是计算大规模数据的相似度，对于大量数据的相似度计算，如果采用传统方法进行逐一比较，将会耗费大量的时间与空间资源，而且通常进行相似度计算的文档数据维度较高，进一步增加了计算相似度的资源消耗，为了解决上述问题，LSH 算法被设计出来，以牺牲一定的精度为代价大大提升相似度比较的运行效率。

为了进一步了解 LSH 算法，我们先对最小哈希算法做一定的了解，假设有两个文档，我们对其按照字典中的字符数 n，每个字符对应一个比特位，为 1 则说明该文档拥有对应字符，为 0 则说明没有，由此获得两个维度为 n 比特的向量，对于这两个向量，我们可以用 Jaccard 系数计算相似度，公式如下：

$$\mathrm{J}(A,B)=\frac{|A\bigcap B|}{|A\bigcup B|}$$

而最小哈希算法就是一种近似计算 Jaccard 系数的方法，上一段中获取的两个向量我们将其与字典序并列放置，将字典序进行 m 次随机排列，并将两个向量中的对应维度随着字典序置换，对于每次的排列结果，我们按照排列顺序按序取出两个向量中的第一个非 0 维度，将其对应的字典序排列，即可将两个 n 维向量的相似度比较降维成两个 m 维向量的相似度比较，比较这两个 m 维向量相同位的概率，就能得到近似的 Jaccard 系数，其中的 m 取值影响最终的近似程度与计算复杂度，如果 m 取全排列的情况，不难推导，即是准确的 Jaccard 系数，综上所述最小哈希算法可以将原始的特征向量转为更低维度的特征向量，从而减少两个对象之间相似度的计算时间，但是通常情况下，我们需要去计算相似度的数据是海量的，如果对象的数量 N 很大，任意两个对象之间都需要我们去计算相似度，执行次数为 O(N^2)，这仍是一笔不小的资源开销，所以我们需要 LSH 算法来减少相似度计算量。

我们依然先试用最小哈希算法，将高维度的特征向量转变为低维度的 signal 向量，LSH 算法的处理则建立在这些 signal 向量上，将这些 signal 向量分为 b 段，每段长度相同，均为 r，我们按照段数取相同数量的桶组，每个段对应一组桶，按照一定的哈希规则对 r 位数据进行分桶，每个段中的桶组里，分到相同桶的数据即是相似度高的数据对，在计算数据相似度时仅取相似度同桶内数据计算即可，这样就大大减少了计算数据相似度的执行次数。

但是 LSH 算法是一种近似的算法，难免会存在一定的误差，我们后续计算 LSH 算法的精确度，从而进行进一步的分析和改进，这部分内容将会写在理论分析部分。

数据结构的设计：

为了实现对于 LSH 算法的模拟和测试，我们直接取 n 个长度为 d 的数组，每个数组包含 d 个 0 或者 1，来模拟 n 个文档的 d 维特征向量，对其进行最小哈希处理，取排列次数为 m，则转化为 n 个 m 维向量，向量中的数据均为大小在 1~d

的整数，然后我们取一个可以被 m 整除的整数 r，作为每段的维度数，m 整除 r 的结果我们命名为 b 来表示段数，分配 b 个桶组合，根据哈希规则来设定桶的大小，分配结束后即完成 LSH 算法的工作，后续需要寻找相似数据时只需找到与其分配在同一桶内的数据进行计算即可。

操作流程分析：

按照上述设计的数据结构进行操作，我们对其流程进行分析，首先，流程中的大部分操作是确定的，文档数 n 以及文档特征向量维度 d 都是由处理数据决定，并不影响我们的分析结果，我们需要分析的变量包含最小哈希处理的排列次数 m，段维数取值 r，以及分桶的哈希规则，我们分析以下几个指标来进行数据取值的选择：

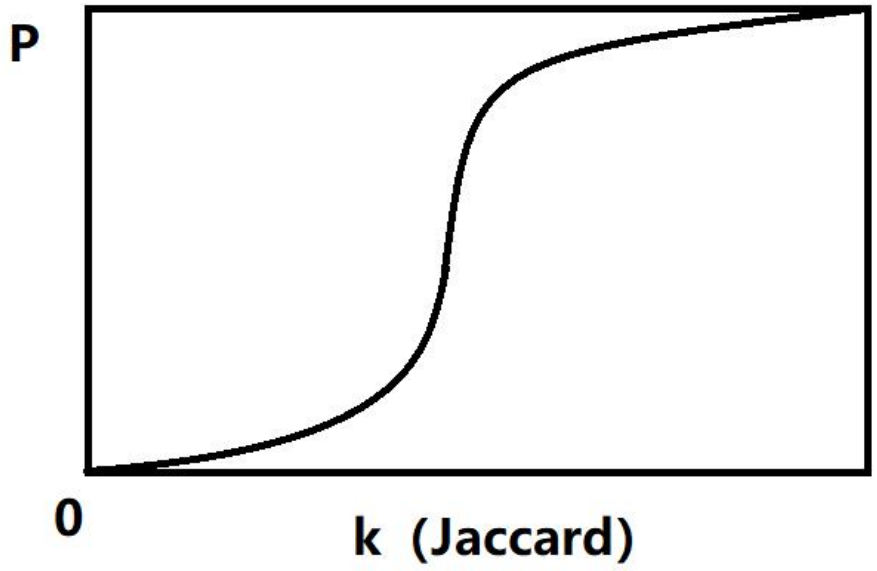
1. 处理过程中所需的空间开销
2. False Positive（相似度低的数据分到一个桶的情况）
3. False Negative（相似度高的数据未被分到一个桶的情况）
4. 分桶选用的不同哈希规则的影响

理论分析：

为了分析指标 1~3，我们需要先对指标 4 就，即处理的哈希规则进行一个确定，为了便于分析，我们首先采用最简单的哈希规则，段向量完全相同的向量划分至同一个桶中，这样做可以很大程度上避免 False Positive 的情况出现，但是代价是对于每个段，需要的桶数较多，最差的情况需要 d^r 个桶，但是考虑到实际情况，如果这样分桶，其中的大部分桶很有可能是空的，我认为此处可以进行一定的优化，初始化桶组中没有桶，每对一个向量进行分桶时，遍历目前桶组的所有桶，若向量哈希匹配，则放入该桶，否则为该向量新增一个桶，加入桶组并将向量放入，对于桶内数据的添加也可采取这种模式，这样可以减少一些不必要的空间消耗。

统计空间开销，共包含 b 个桶组，每个桶组包含最差情况 d^r 个桶，每个桶的空间最差为全部 n 个文档特征向量均分配到一个桶中，但桶组文档总是仍未 n，由于每个 signal 向量被分为了 b 段，共使用了 b 个如上桶组的空间。

分析 False Positive 与 False Negative 情况，假设我们现在正在比较两个文档，其 Jaccard 系数为 k，我们的 LSH 算法将其 signal 向量分为长度为 r 的 b 个段，由前文中对最小哈希处理的介绍可知，两组 signal 向量在某一维度上取值相同的概率即为精确 Jaccard 系数 k，所以在该段内，这两个文档被分到一个桶里的概率为 k^r，则在该段内未被分到一个桶中的概率为 1-k^r，所有段中均未分到一个桶中的概率为（1-k^r）^b，所以最终得出两个文档被判断为高相似度数据对的概率是 1-（（1-k^r）^b），我们画出该函数的大致图像，横轴设置为 Jaccard 系数 k，纵轴为最终概率，图象大致如下：



我们可以看出，该函数存在一段区域，该段区域内两个文档被分到同一个桶内的概率激增，我们不妨设定一个期望的 Jaccard 系数阈值，将其近似视为一个阶跃函数，小于该阈值则不是高相似度数据对，大于该阈值即为高相似数据对，再回到我们的函数式，其中 r 与 b 的取值也影响了该阈值的位置，为了使实际阈值与我们所期待的阈值相近，需要取用合适的 r 与 b 值，而我们所期待的 Jaccard 系数阈值其实就是判断两个文档是否划分为高相似度数据度的相似程度，当期望阈值设置较高时，可能会将部分相似度较高的数据没有分到一个桶中，反之，期望阈值设置较低时，会将部分相似度较低的数据分到一个桶中，这样的情况是可以根据需要解决的问题通过设置合理的阈值来解决，但是由于设置阈值的处理方式是将函数曲线近似为阶跃函数来实现的，因此便会出现 False Positive 与 False Negative 这两种情况，我们设定阈值为 a，假设给定数据中的任意两组数据的相似度（即 Jaccard 系数）组成的数据集是均匀分布的，，则可以通过积分或得出现这两种情况的概率：

False Positive：

$$\int_0^a (1-(1-k\text{^}r)\text{^}b)d(k)$$

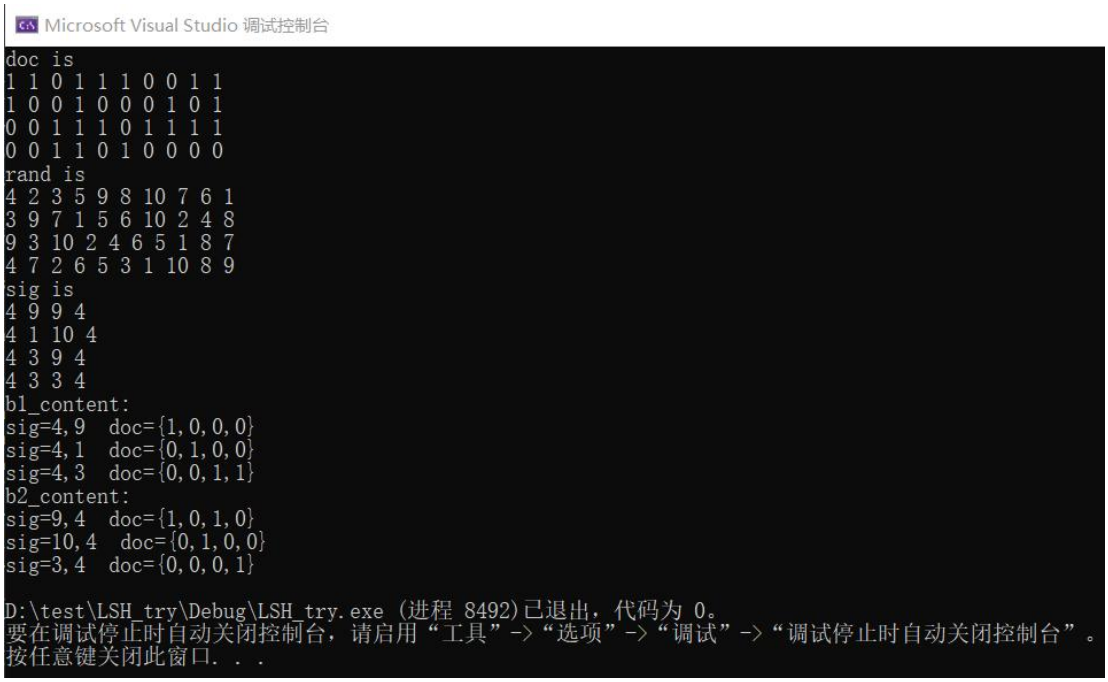
False Negative：

$$\int_a^1 (1-(1-(1-k\text{^}r)\text{^}b))d(k)$$

两式分别计算了将函数近似为阶跃函数的误差部分，即为出现两种误差情况，很明显，在其他变量已确定的情况下，当阈值 k 对应概率为 0.5 时两种情况发生总概率是最小的，这也是我们在取值时要注意的。

实验测试的性能：

如上文所述，我们编写程序模拟最小哈希算法以及 LSH 算法的处理过程，源码见文件 LSH_try.cpp，代码已注释，此处仅展示模拟结果



我们随机生成了四个特征向量为 10 维的文档，并且随机了 4 次最小哈希序列，获得了 4 个降为 4 维的 sig 向量，之后段长设为 2，共分两段，采用向量唯一的分桶策略，将其分入桶，可以看到，段 1 中，文档 3 和 4 被分到一个桶中，段 2 中，文档 1 和 3 被分到一个桶中，为了更直观的看到分桶的正确性，我们计算四个向量两两间的准确 Jaccard 系数以及最小哈希得出的近似值：

	准确值	近似值
1,2：	3/8=0.375	2/4=0.5
1,3：	3/10=0.3	3/4=0.75
1,4：	2/8=0.25	2/4=0.5
2,3：	3/8=0.375	2/4=0.5
2,4：	1/6=0.167	2/4=0.5
3,4：	2/8=0.25	3/4=0.75

可以看出，由于文档间的 Jaccard 系数准确值分布较为密集，分桶出现了许多明显的问题，相似度较高的 1,2 文档以及 2,3 文档均未分到一个桶内，对应 False Negative 情况，相似度并没有很高的 3,4 文档却被分到了同一个桶中，对应了 False Positive 情况，所以实际的分桶结果与 Jaccard 系数准确值并不是十分匹配，但是与最小哈希得出的近似值匹配度却十分高，由此我们可以认为，LSH 算法分桶时采用的完全向量相同的哈希规则是准确度很高的，在这个简单的例子中准确度达到了 100%，而针对出现的问题我认为是最小哈希处理时对于原数据的降维过大，导致数据准确性丢失太多，若将 m 提高至 6，准确性则会提升

不少。

以上是对于 LSH 算法的模拟，对于 LSH 算法，在此我提出我认为能够减小空间开销的方案。

减小空间开销的方案：

首先，LSH 算法的主要空间开销是由于 sig 向量分段而需要设立的大量桶组，每一个段都需要一个桶组去存储分桶结果，我们不妨将分段数减少，从而减少桶组的空间开销，但是这样处理会面临一个问题，由于段数的减少导致每一段的向量维度增加，这时候在采取上文模拟时的完全对应哈希规则就很有可能导致许多相似度较高的文档未被分到同一个桶中，所以要采取新的哈希规则，我的想法是对于对应的维度空间进行分割，将所有的向量一侧端点固定为原点，根据其另一端点的空间位置分桶，设置阈值，当该向量与当前桶内所有元素的欧氏距离在该阈值内，则归入该桶，依照这样的原则进行分桶，可以在大大减少空间开销。

举例说明，假设向量是三维的，每个向量就对应半径为设定阈值的球体，桶内所有球体的重叠空间即为该桶的 key 值。

我们再来分析一下时间开销，将向量某一维度的一次比较视为一次操作，若采取完全对应哈希规则，无论段长度如何，时间复杂度均为 O(d*m)，即文档数与 sig 向量维度的积，但是采用欧氏距离哈希规则来分桶时，最差情况下每个向量的比较都要遍历所有已存在桶内的所有向量，最好情况下是与完全对应哈希规则相同，所以这样的改进是用了一定的时间复杂度来换取了空间复杂度，来减少 LSH 算法的空间开销。

以上是我了解相关知识后对 LSH 算法进行的模拟分析以及提出的一些个人看法和改进方案，可能由于本人知识面仍较浅薄会有不合理甚至错误的地方，还请谅解。

CS2101 杨东升　U202115319