part2 Locality Sensitive Hashing

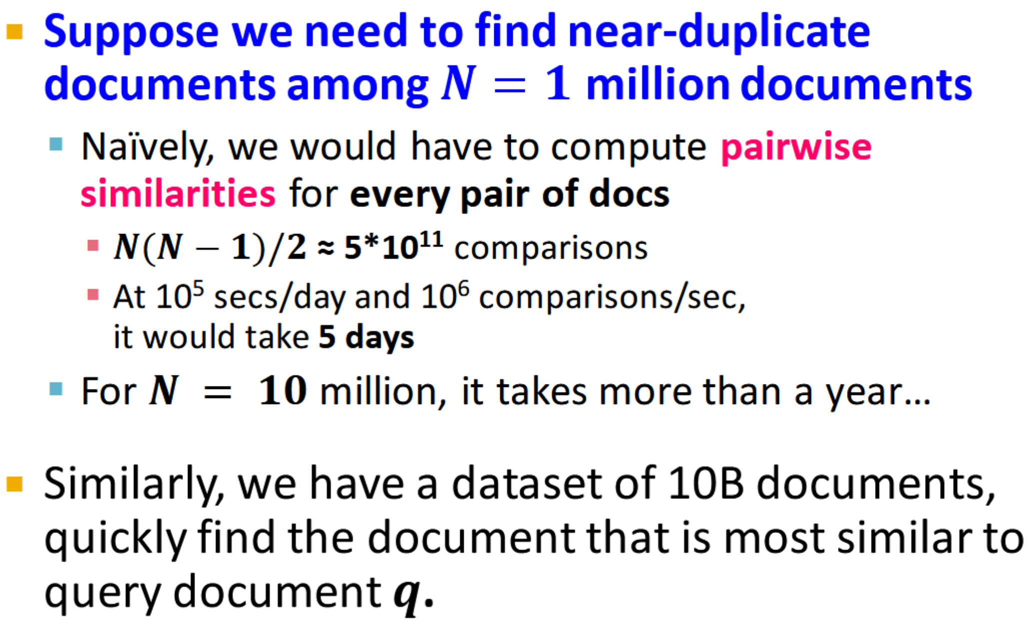
局部敏感哈希是一种计算向量相似度的近似算法。

在很多场景中，我们需要计算大量高维向量的相似度。

- 在文档主题匹配中，对于文档的topic vector进行相似性计算
- 协同过滤中，相似的用户购买相似的物品
- 推荐和搜索场景

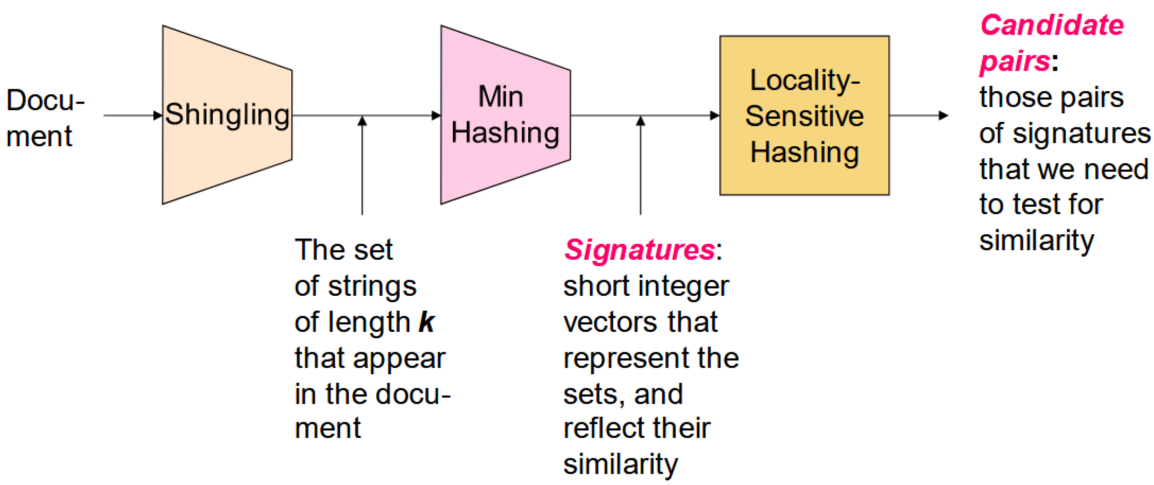
所以我们需要解决的问题为：给出高维向量x_i，定义距离函数$d(x_1, x_2)$，如何找出使得$d(x_i, x_j) <= s$的向量组合(x_i, x_j)。暴力枚举的算法复杂度为$O(N^2)$,如何将复杂度缩短到$O(N)$?

Motivation for LSH



假设我们需要寻找相似的documnets among N = 1 million。整个流程可以被分为三部分。

- Shingling : Converts a document into a set representation
- Min-Hashing: Convert large sets to short signatures, while preserving similarity
- Locality-Sensitive Hashing: Focus on pairs of signatures likely to be from similar documents



Shingling

Step 1: *Shingling:* Converts a document into a set

- A ***k*-shingle** (or ***k*-gram**) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples
- To **compress long shingles**, we can **hash** them to (say) 4 bytes
- **Represent a document by the set of hash values of its *k*-shingles**

将长文档转化为$a\ set\ of\ hash\ values$,例如

- **Example:** **k=2**; document $\mathbf{D_1}$= abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  Hash the shingles: $\mathbf{h(D_1)}$ = {1, 5, 7}
- ***k*** = 8, 9, or 10 is often used in practice

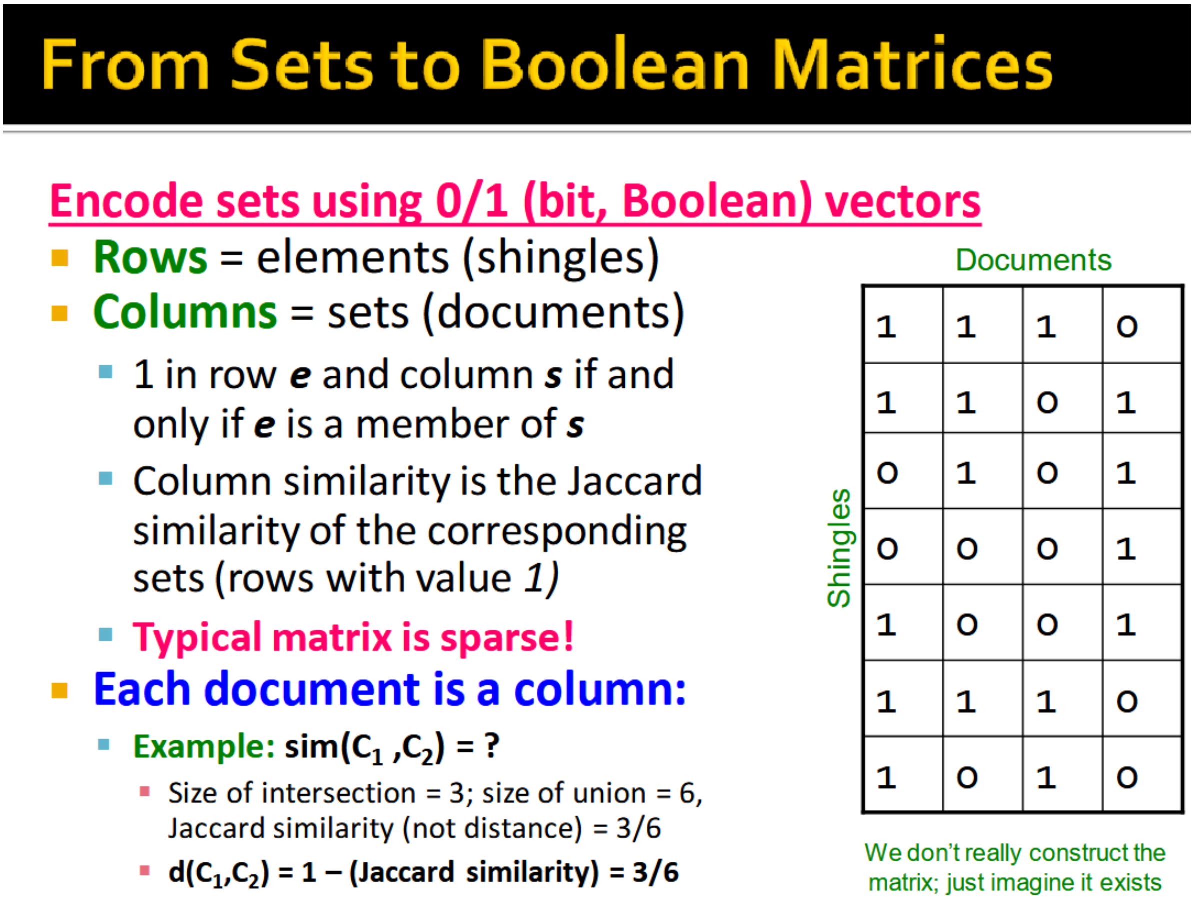
相似的文档会有许多相同的shingling，同时将文档切分后，更改某个单词只会影响k-shingling中的其余k-1个单词。

进行Shingling后的document D_i可以被表示为$C_i = S(D_i)$，即k-shingling的集合。

Jaccard系数是常见的衡量两个向量（或集合）相似度的度量：

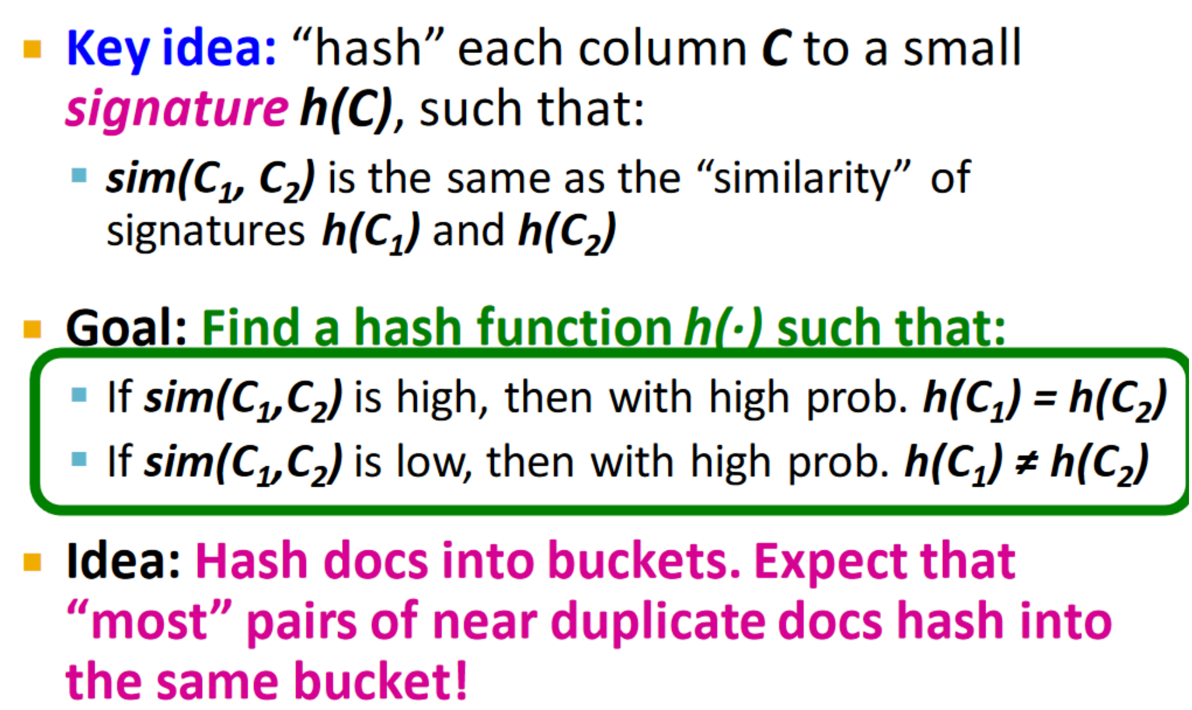
$$sim(D_1, D_2) = |C_1 \bigcup C_2| / |C_1 \bigcap C_2|$$

Jaccard distance : $d(C_1, C_2) = 1 - |C_1 \bigcup C_2| / |C_1 \bigcap C_2|$



对于进行Shingling后的文档向量，我们可以使用01向量来表示。

Min-Hashing



Min-Hashing的思想是寻找到一种将高维稀疏的shingling向量映射到低维向量的方法，同时保证映射前后的向量相似度尽可能的一致。相似的doc向量在映射之后也应该相同的bucket。

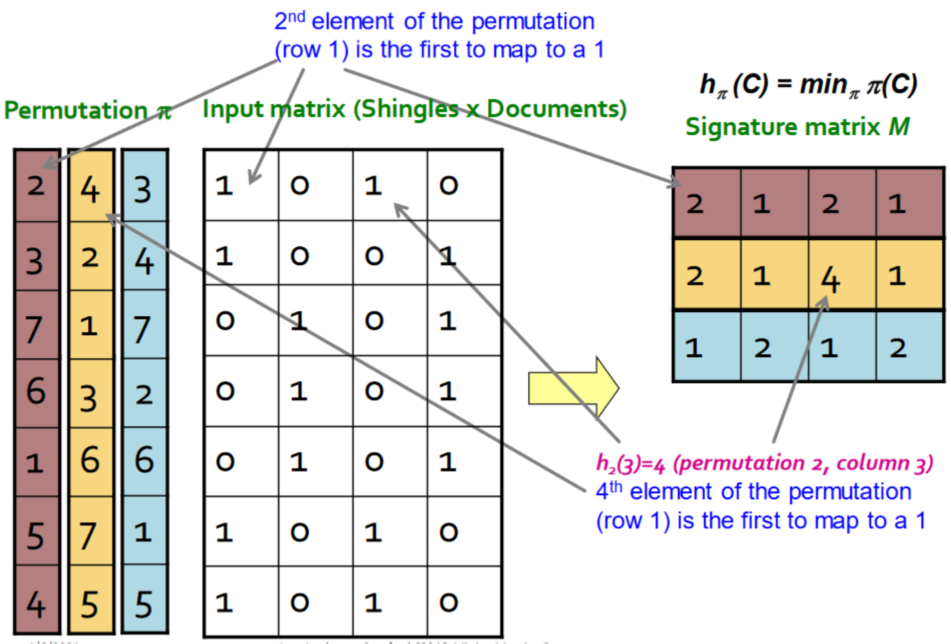
- 如果$sim(C_1, C_2)$很高，那么映射后的hashing函数结果$h(C_1) = h(C-2)$的概率也应该很高
- 如果$sim(C_1, C_2)$很低，那么映射后的hashing函数结果$h(C_1)! = h(C-2)$的概率也应该很高

因此对于向量相似性的度量会影响到Hashing函数的选择，并且并不是所有的相似性度量都会有与支配的Hashing函数。对于$Jaccard$相似性而言，与之对应的Hashing函数是Min-Hashing函数。

Min-Hashing大致流程如下：

- 将Shingling后形成的01矩阵使用随机排列π替换
- 随机排列π代表一串随机顺序的正整数数列，长度与Shingling矩阵的行数相同。
- 对于矩阵的每一列C和一个随机排列数列π，$h_\pi(C) = min_\pi(\pi(C))$,即Hashing的结果是这个随机排列$\pi$对应的最小的$C$中为1的数。

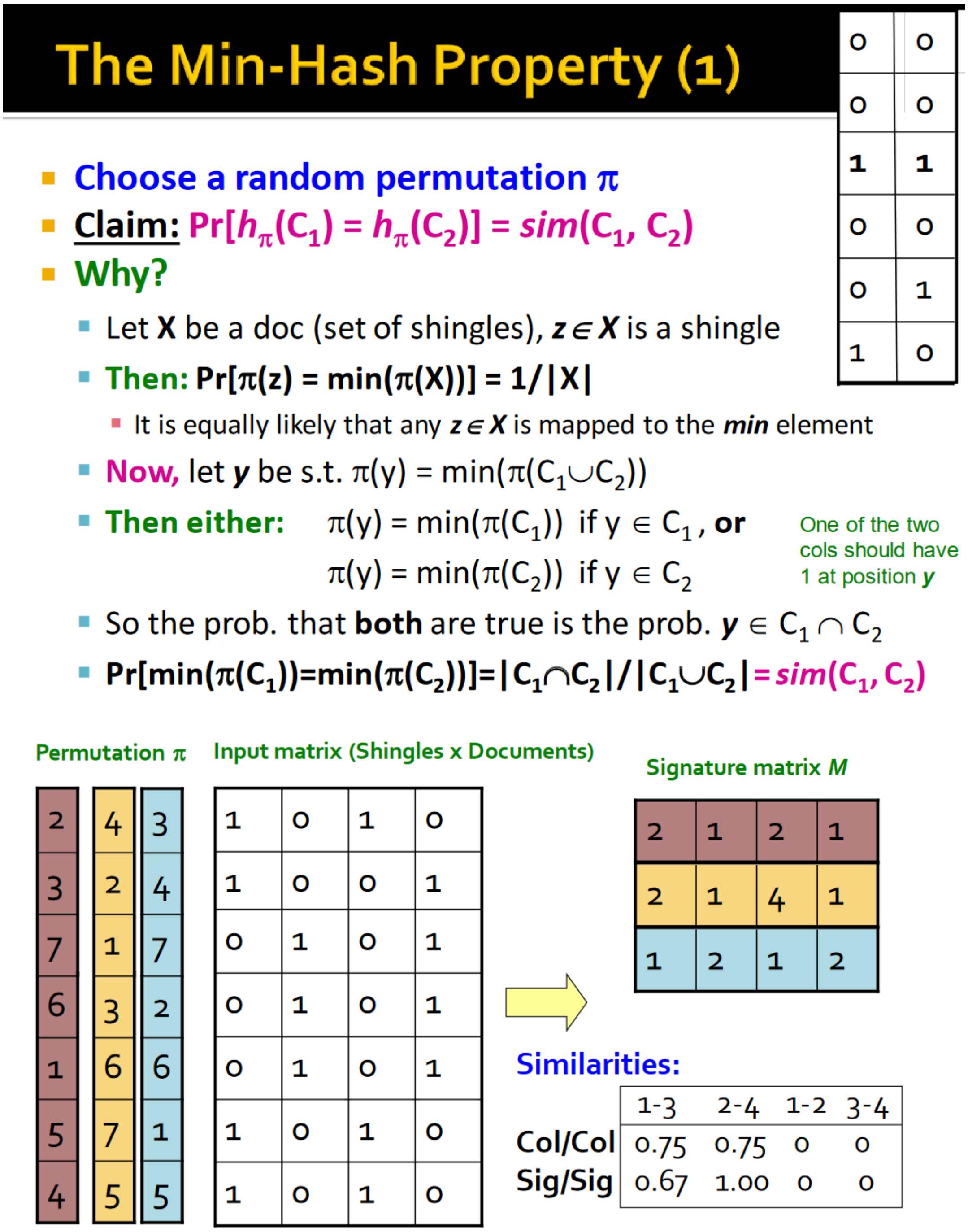
一个随机排列π对于C会形成一个整数，因此Min-Hashing后的signature向量维度就等于我们选择的π个数。



上图给出了三个随机排列π，长度与矩阵的行数相同。我们根据矩阵列向量为1的行，找到对应π的整数值，找到其中最小的为Hashing的结果。上图有三个排列，所以得到的signature结果为三维。

Min-Hashing的性质

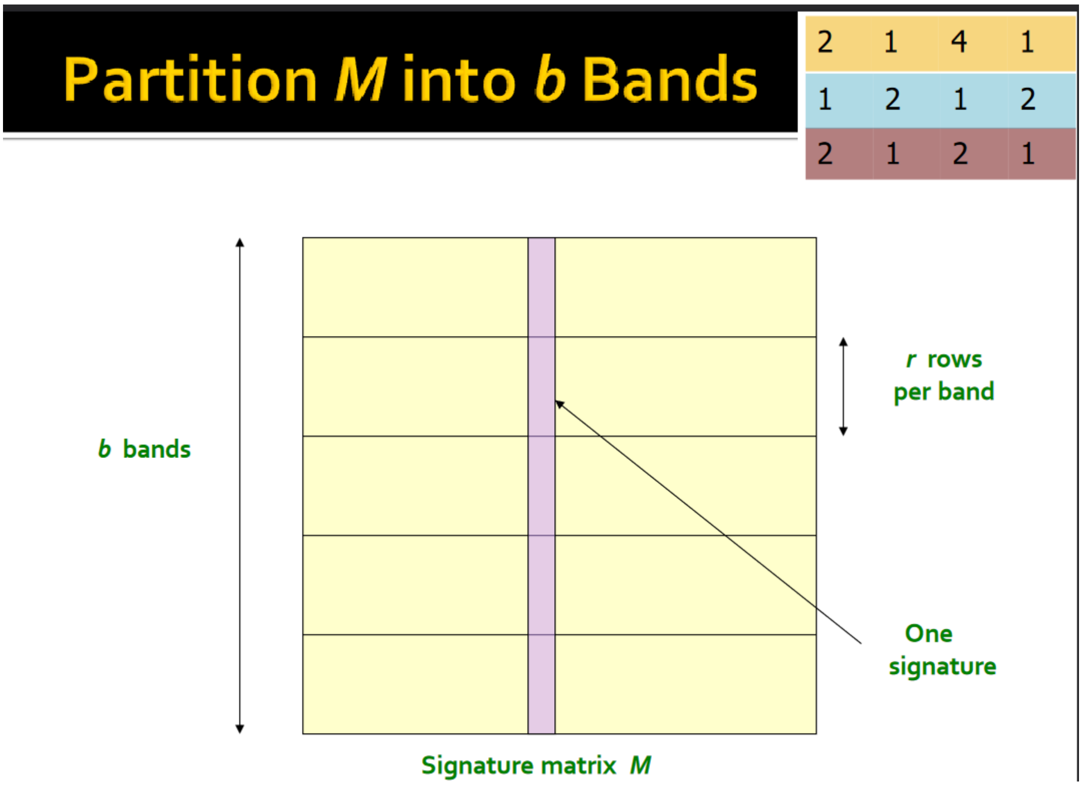
- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ 即对于同一个随机排列π，Min-Hashing结果相同的概率等于C_1 C_2的相似度。



通过Min-Hashing我们可以在降低了doc表示向量的同时保持了向量之间的相似度。Signature向量的维度越接近Shingling向量，向量相似度就会保存的越好。

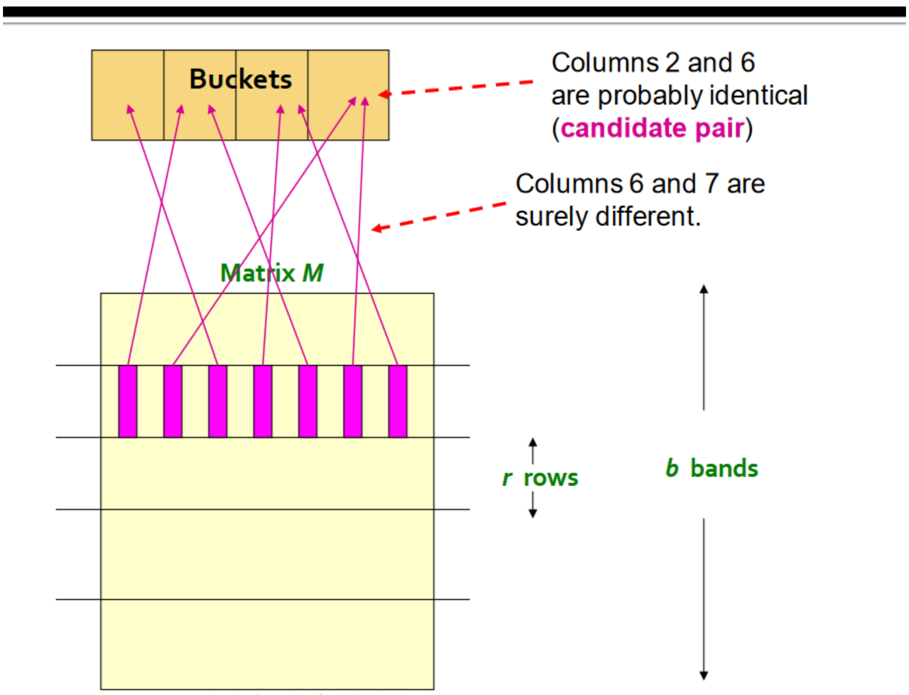
LSH

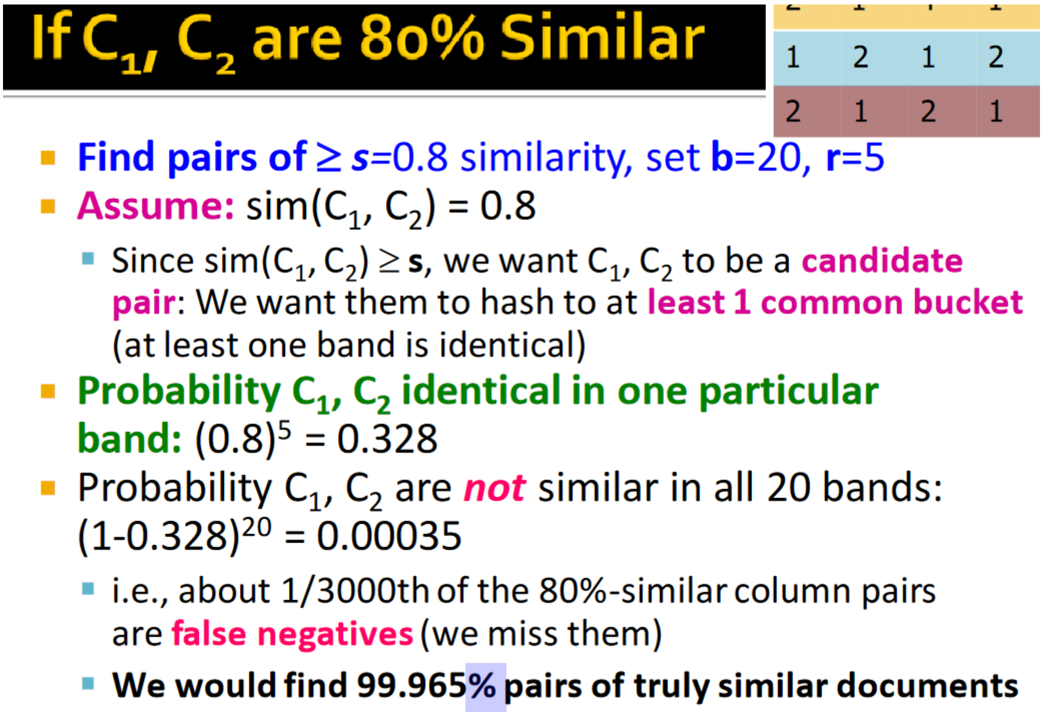
通过Shingling和Min-Hashing我们完成了将doc转化为01向量并且进行了降维的过程，但是如果有N篇文档且N很大时，计算复杂度仍然为$O(N^2)$。LSH主要通过筛选出部分candidate向量来减小N^2的复杂度。



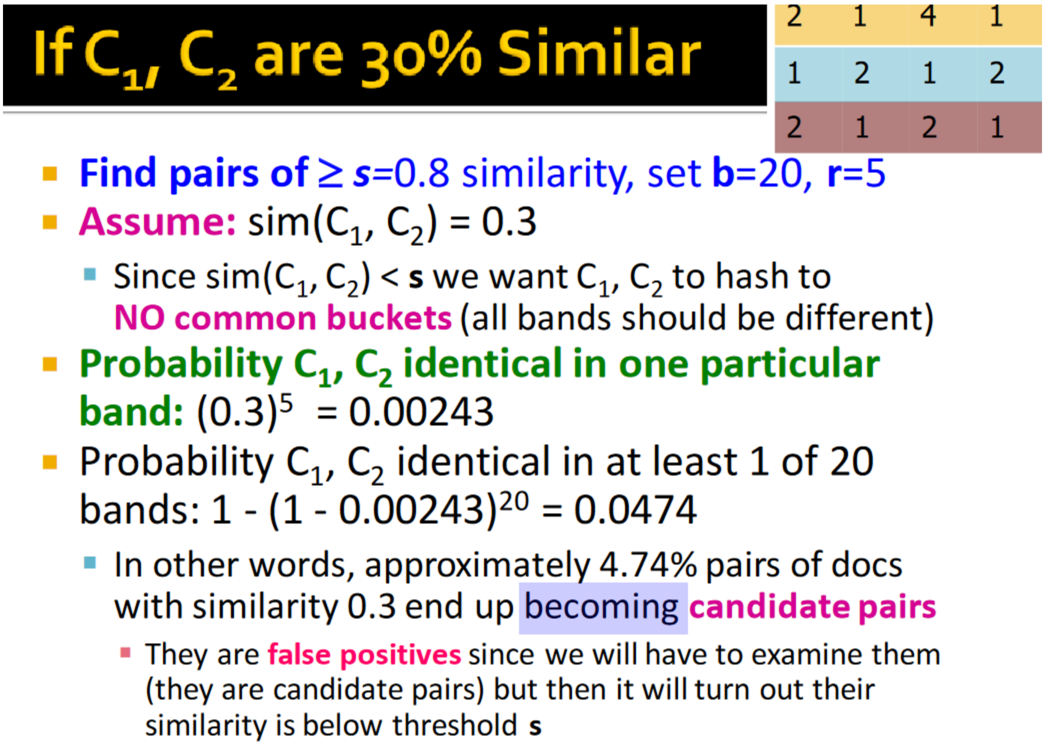
M矩阵的每一列就是Min-Hashing后得到的signature向量。每一个向量在图中被分为了b段（每一列为一个向量），每一段有r 行MinHash值。在任意一个band中分到了同一个桶内，就成为候选相似用户（拥有较大可能相似)。

我们设两个向量的相似度为t，则其任意一个band所有行相同的概率为 t^r，至少有一行不同的概率为 $(1-t^r)$则所有band都不同的概率为 $(1-t^r)^b$,至少有一个band相同的概率为 $1-(1-t^r)^b$ 。

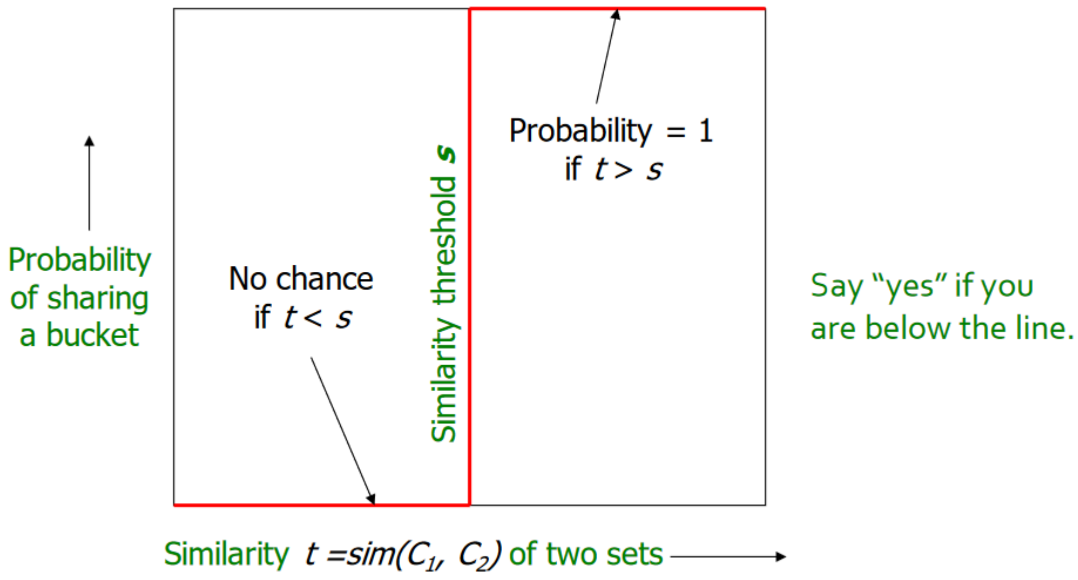




根据上图的假设，我们假设C_1 C_2的相似度为0.8，即大概率是真正相似的向量，signature向量维度为100，分为20个band，band内长度为5。经过计算我们可以得到我们根据LSH算法有99.9%的概率计算为candidate向量。

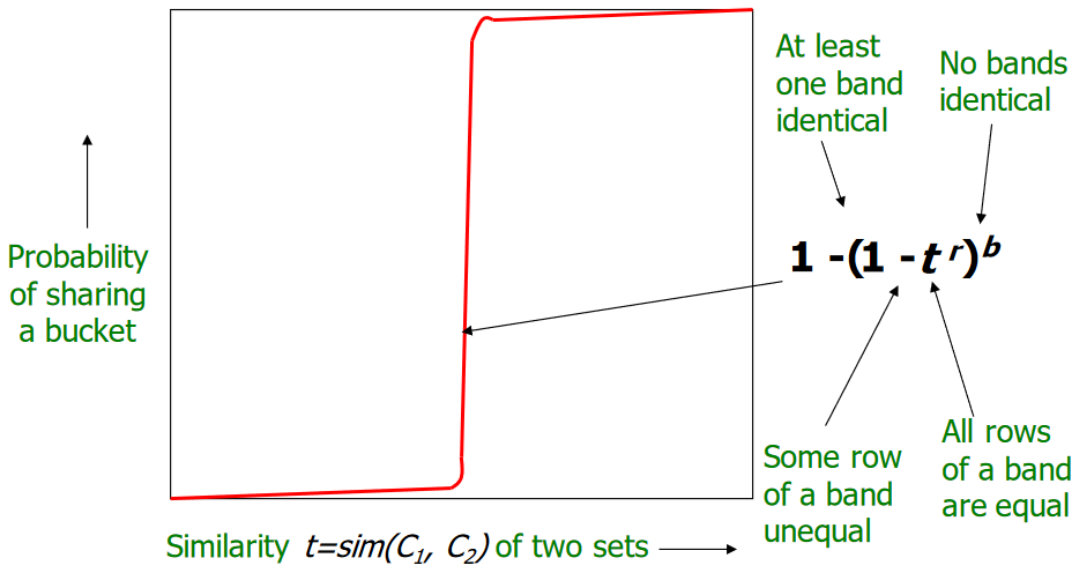


根据上图的假设，我们假设C_1 C_2，即不是真正相似的向量，我们也可以计算得到他们有4.74%被计算为candidate向量。



上图是我们期望构造的判别函数，对于不同的$sim(C_1, C_2)$，给定某个阈值后，小于阈值的向量组合不会被share到一个bucket中，大于阈值的向量组合会100%被share到一个bucket中。

What *b* Bands of *r* Rows Gives You



根据上文的计算，LSH算法构造的判别函数如上图所示，通过选择不同的超参数r和b。我们可以控制LSH的函数图像尽可能逼近理想的曲线。所以现在的问题转化为如何调参b和r来调整曲线的形状？

Summary: 3 Steps

- **Shingling:** Convert documents to set representation
  - We used hashing to assign each shingle an ID
- **Min-Hashing:** Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - We used hashing to get around generating random permutations
- **Locality-Sensitive Hashing:** Focus on pairs of signatures likely to be from similar documents
  - We used hashing to find **candidate pairs** of similarity $\geq$ **s**

我们现在完成了Shingling,Min-Hashing和LSH的流程。关于LSH的原理和调参在下一节中阐述。