part2 Locality Sensitive Hashing

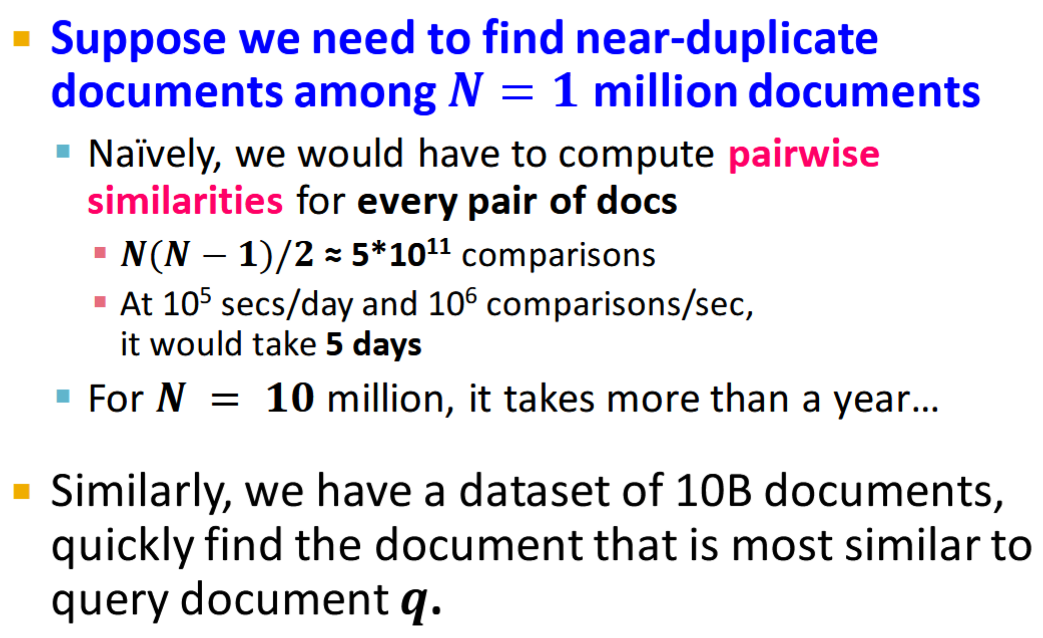
局部敏感哈希是一种计算向量相似度的近似算法。

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$O(N)$?

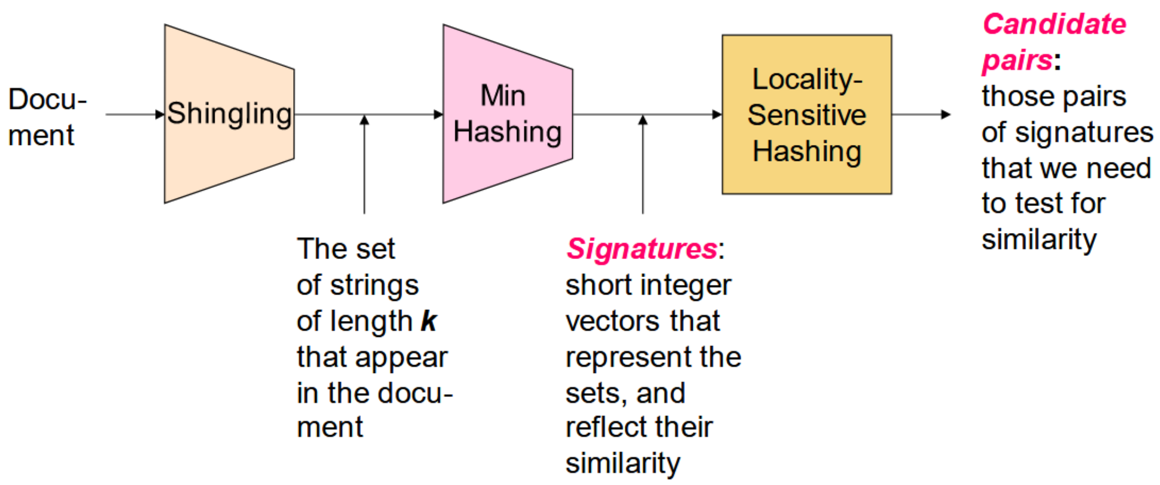
Motivation for LSH



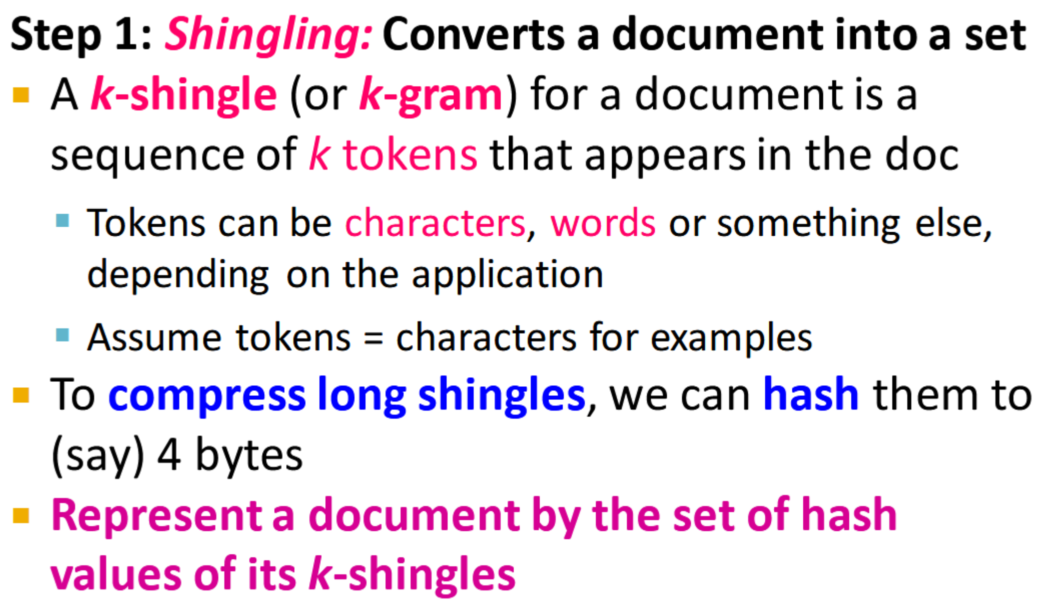
- **Suppose we need to find near-duplicate documents among $N = 1$ million documents**
  - Naïvely, we would have to compute **pairwise similarities** for **every pair of docs**
    - $N(N-1)/2 \approx 5*10^{11}$ comparisons
    - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
  - For $N = 10$ million, it takes more than a year…
- Similarly, we have a dataset of 10B documents, quickly find the document that is most similar to query document q.

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


Docu-ment
Shingling
Min Hashing
Locality-Sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity
The set of strings of length k that appear in the docu-ment
Signatures: short integer vectors that represent the sets, and reflect their similarity

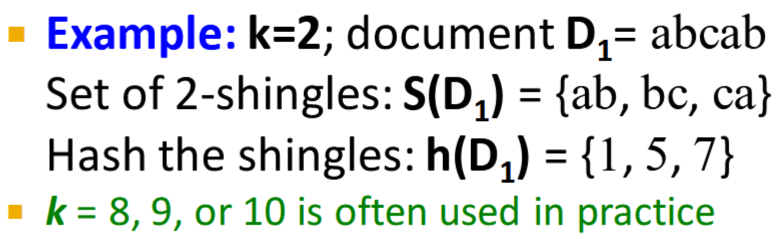
Shingling



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

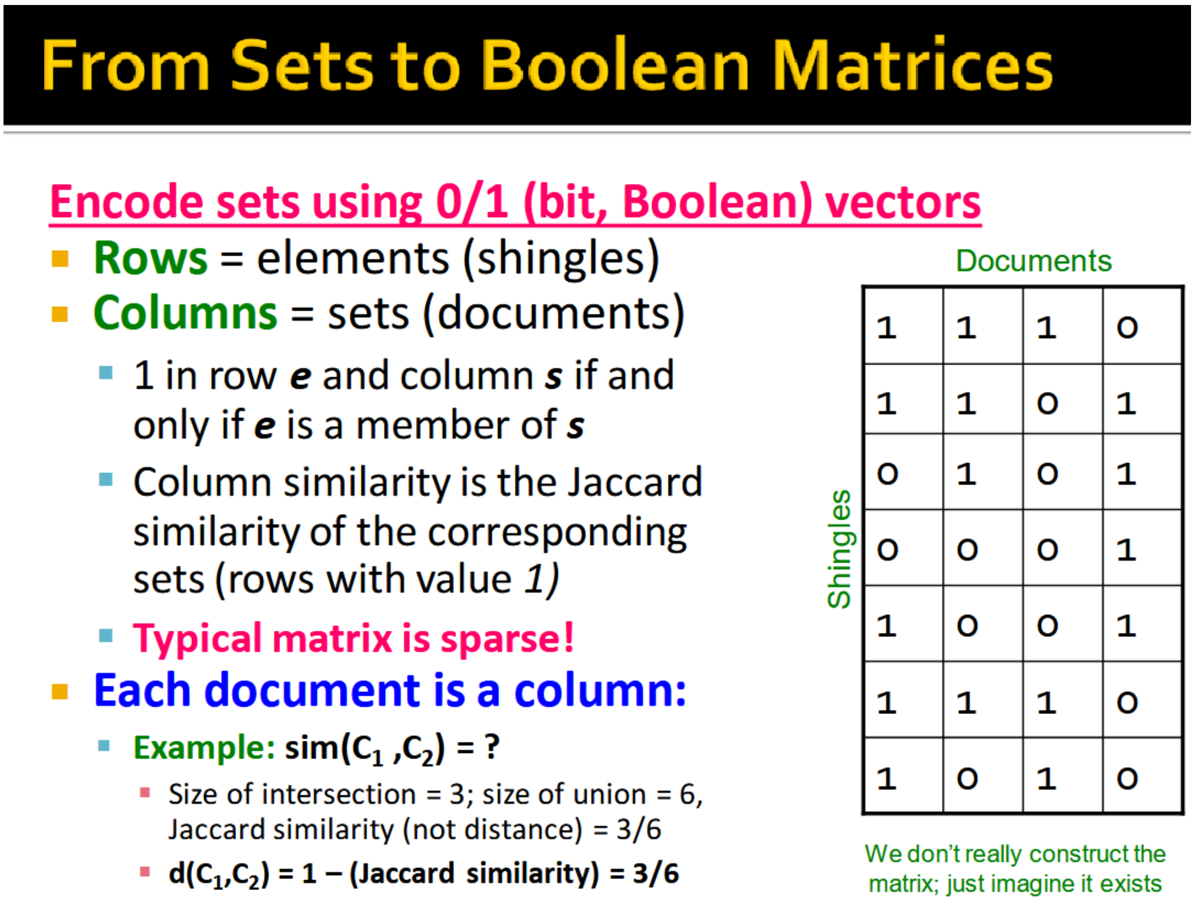
相似的文档会有许多相同的shingling，同时将文档切分后，更改某个单词只会影响k-shingling中的其余k-1个单词。

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



From Sets to Boolean Matrices

Encode sets using 0/1 (bit, Boolean) vectors

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
  - 1 in row ***e*** and column ***s*** if and only if ***e*** is a member of ***s***
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value *1)*
  - **Typical matrix is sparse!**
- **Each document is a column:**
  - **Example: sim(C_1 ,C_2) = ?**
    - Size of intersection = 3; size of union = 6, Jaccard similarity (not distance) = 3/6
    - **d(C_1,C_2) = 1 – (Jaccard similarity) = 3/6**

Documents

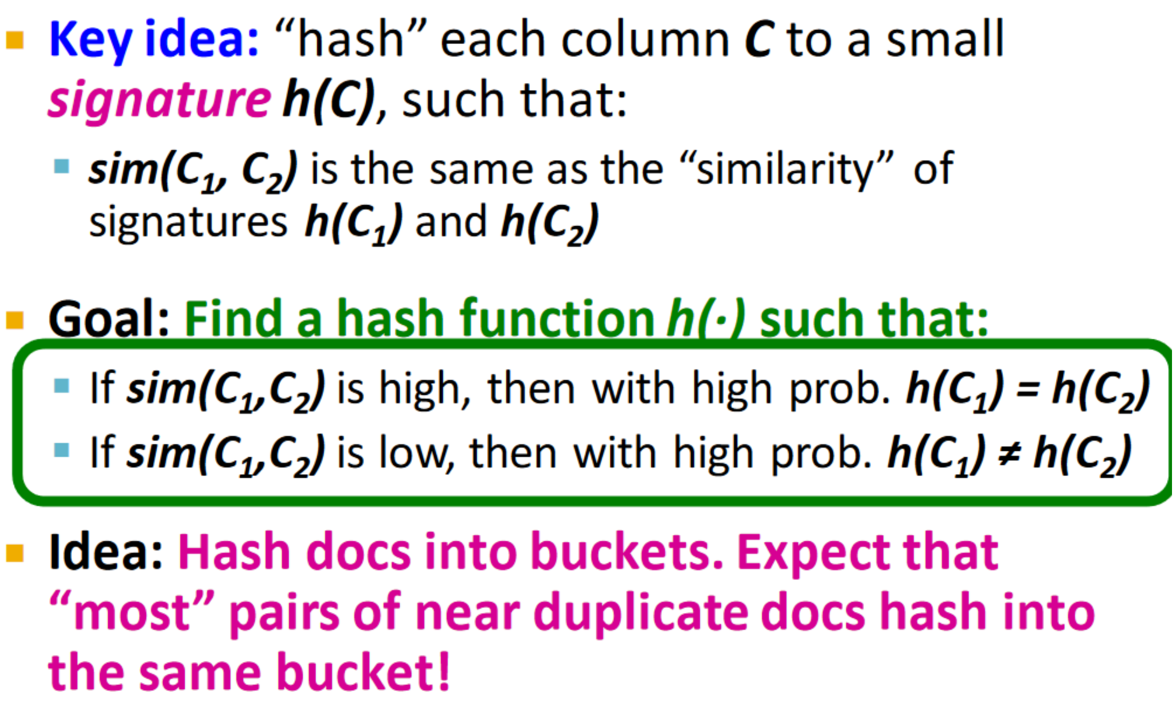
Shingles

1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

We don't really construct the matrix; just imagine it exists

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



- **Key idea:** "hash" each column ***C*** to a small ***signature h(C)***, such that:
  - ***sim(C_1, C_2)*** is the same as the "similarity" of signatures ***h(C_1)*** and ***h(C_2)***
- **Goal: Find a hash function *h(·)* such that:**
  - If ***sim(C_1,C_2)*** is high, then with high prob. ***h(C_1) = h(C_2)***
  - If ***sim(C_1,C_2)*** is low, then with high prob. ***h(C_1) ≠ h(C_2)***
- **Idea: Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!**

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。

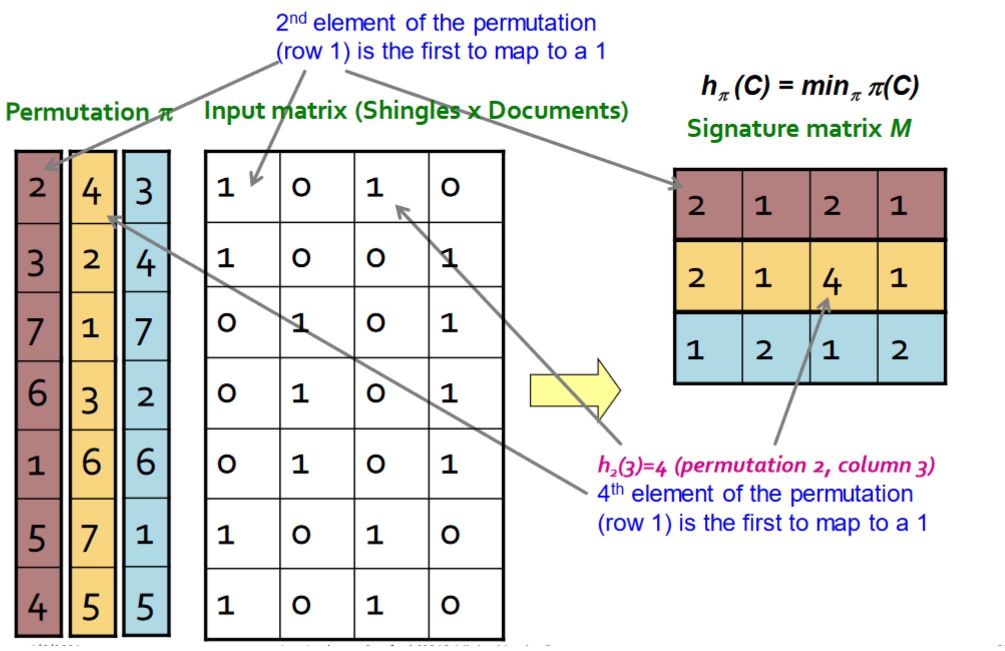
- 如果$sim(C_1, C_2)$很高，那么映射后的hashing函数结果$h(C_1) = h(C-2)$的概率也应该很高
- 如果$sim(C_1, C_2)$很低，那么映射后的hashing函数结果$h(C_1) ! = h(C-2)$的概率也应该很高

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。

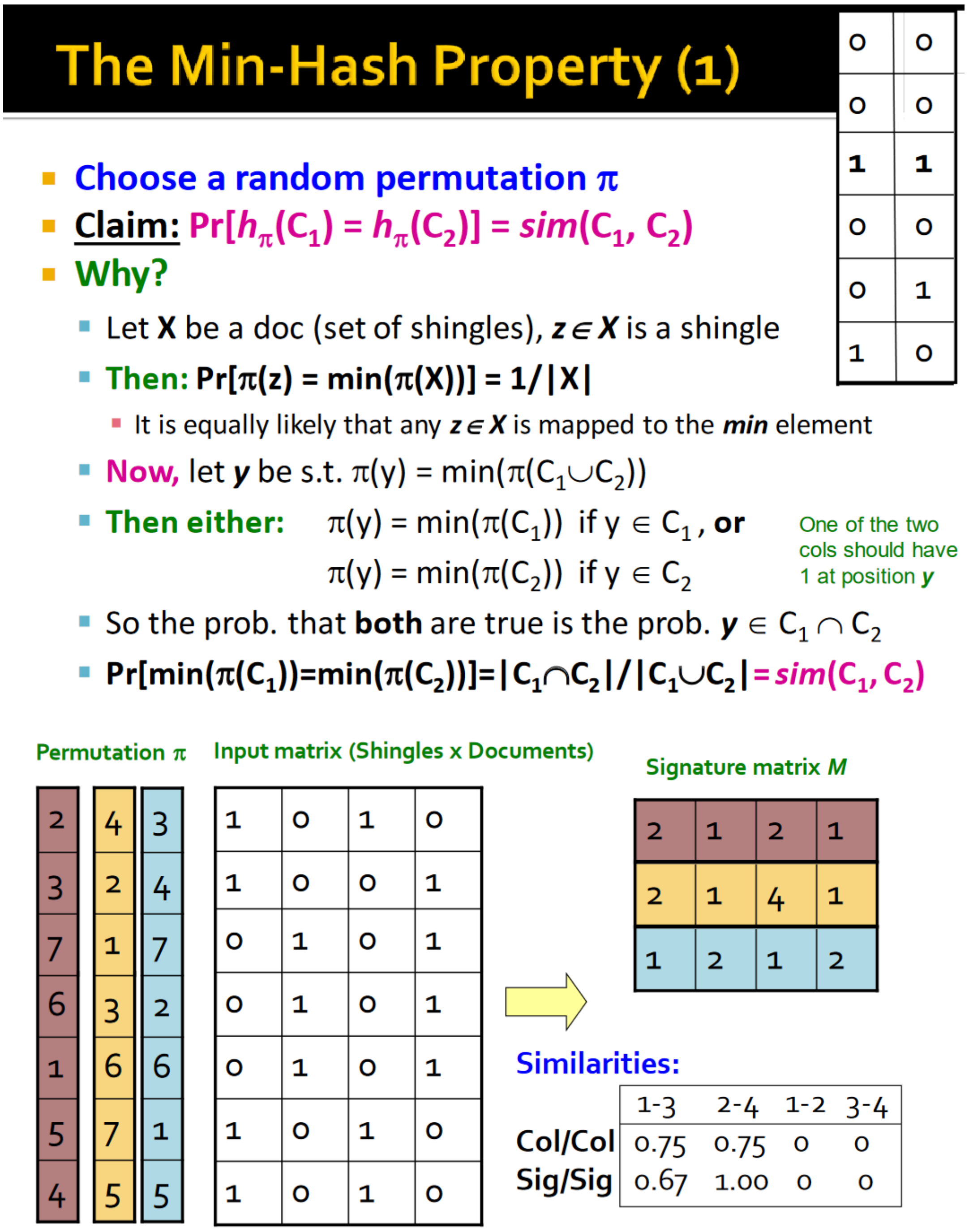

2nd element of the permutation
(row 1) is the first to map to a 1
Permutation π
Input matrix (Shingles x Documents)
h_π(C) = min_π π(C)
Signature matrix M
h_2(3)=4 (permutation 2, column 3)
4th element of the permutation
(row 1) is the first to map to a 1

Permutation π			Input matrix				Signature matrix M			
2	4	3	1	0	1	0	2	1	2	1
3	2	4	1	0	0	1	2	1	4	1
7	1	7	0	1	0	1	1	2	1	2
6	3	2	0	1	0	1				
1	6	6	0	1	0	1				
5	7	1	1	0	1	0				
4	5	5	1	0	1	0				

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。



The Min-Hash Property (1)

0	0
0	0
1	1
0	0
0	1
1	0

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let **X** be a doc (set of shingles), $z \in X$ is a shingle
  - **Then:** $\Pr[\pi(z) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $z \in X$ is mapped to the ***min*** element
  - **Now,** let y be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
  - **Then either:** $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, **or**
    $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$

    One of the two cols should have 1 at position ***y***
  - So the prob. that **both** are true is the prob. $y \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2|= sim(C_1, C_2)$

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

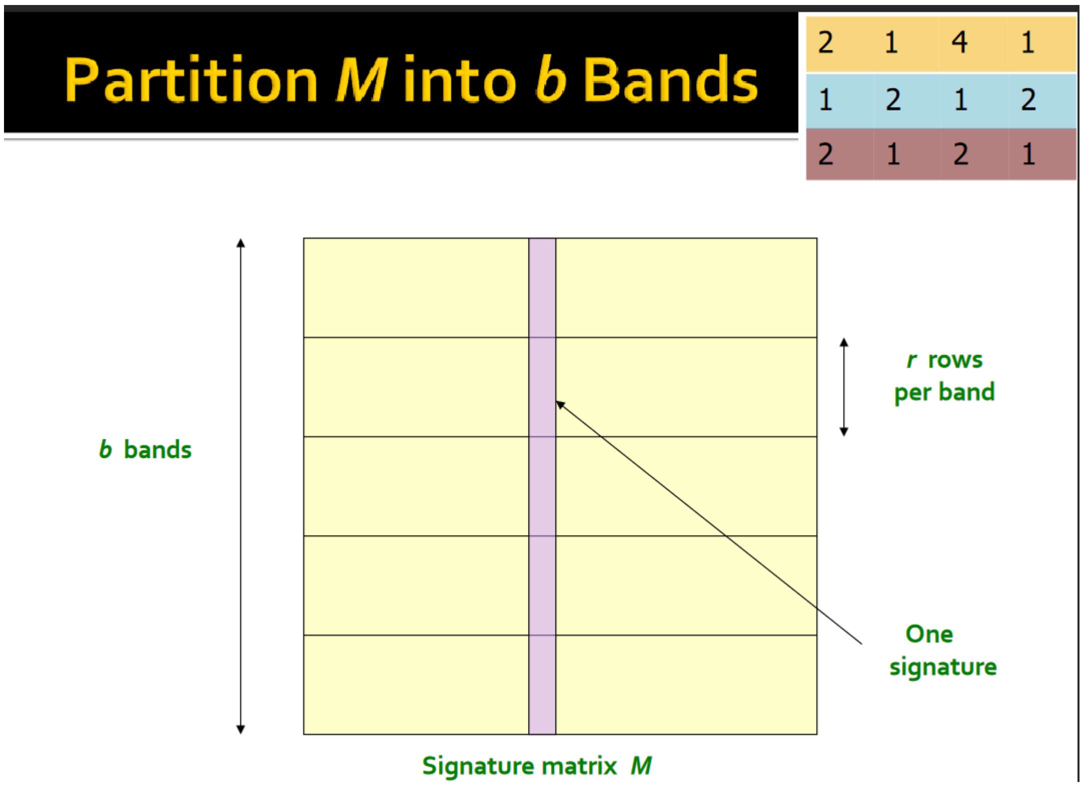
Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

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。

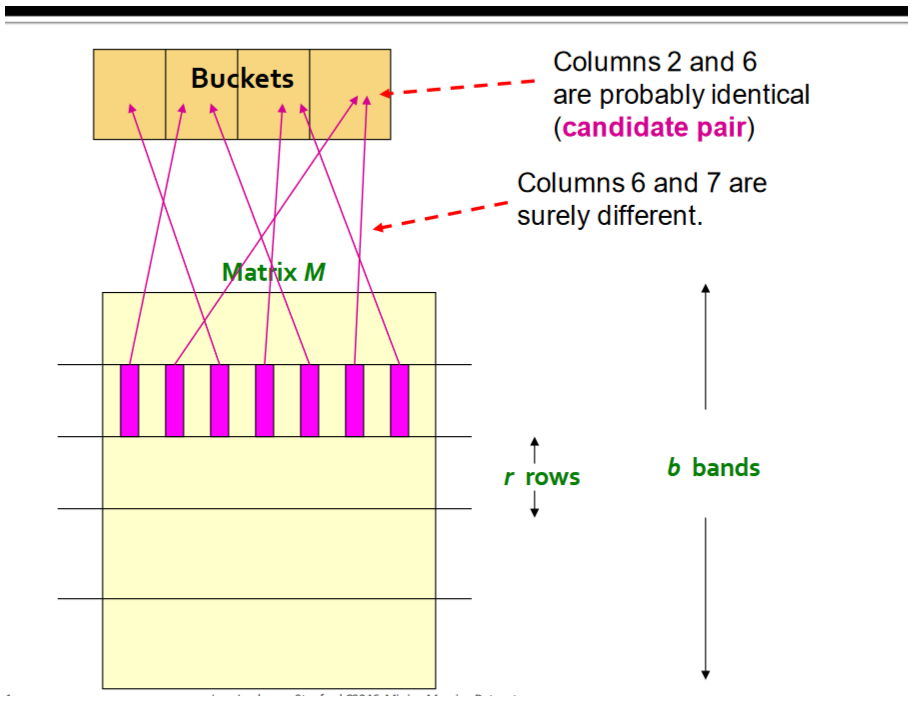
LSH

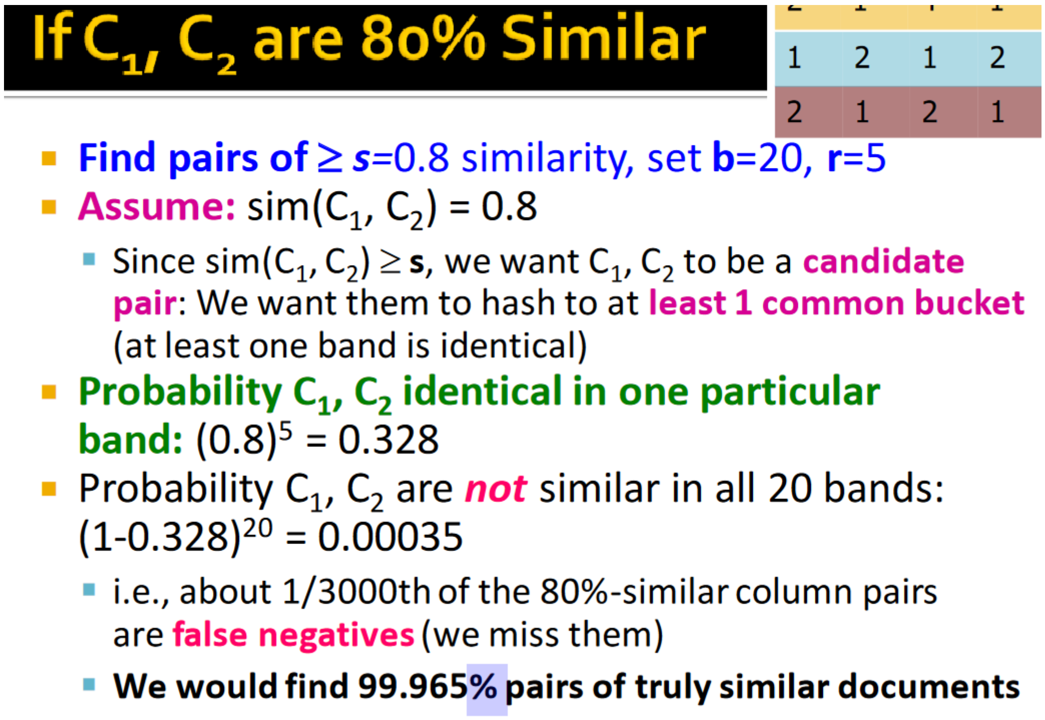
通过Shingling和Min-Hashing我们完成了将doc转化为01向量并且进行了降维的过程，但是如果有N篇文档且N很大时，计算复杂度仍然为$O(N^2)$。LSH主要通过筛选出部分candidate向量来减小N^2的复杂度。



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 $1-(1-t^r)^b$ 。


Buckets
Columns 2 and 6 are probably identical (candidate pair)
Columns 6 and 7 are surely different.
Matrix M
r rows
b bands

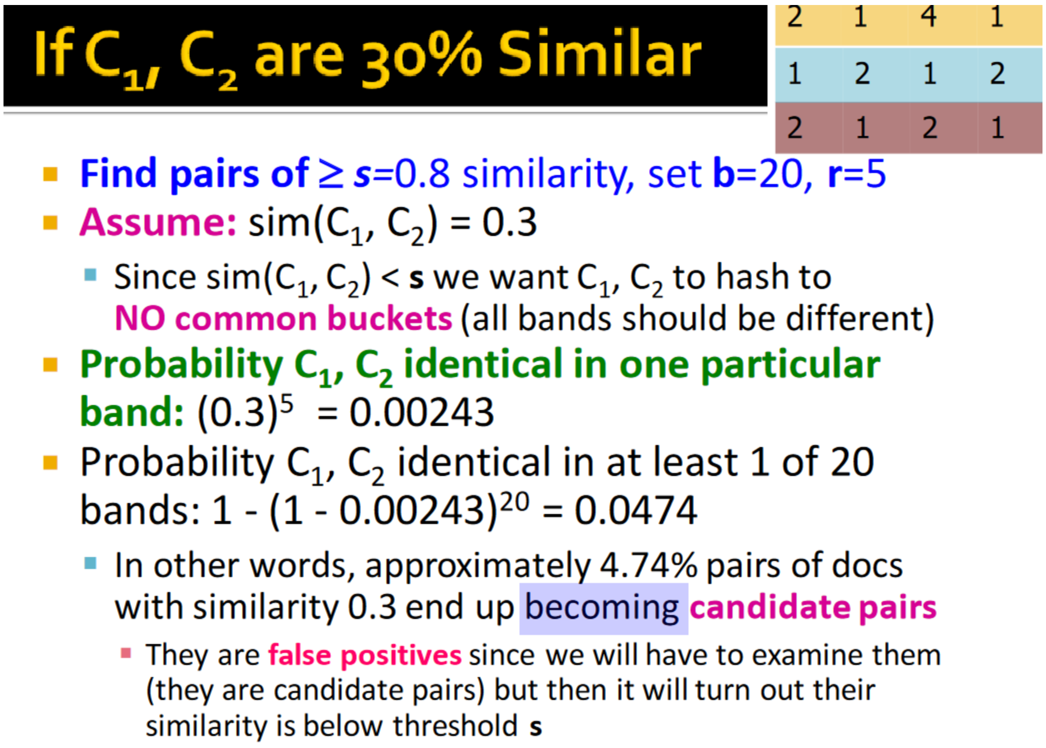


If C_1, C_2 are 80% Similar

1	2	1	2
2	1	2	1

- **Find pairs of** $\geq$ ***s***=0.8 similarity, set **b**=20, **r**=5
- **Assume:** sim(C_1, C_2) = 0.8
  - Since sim(C_1, C_2) $\geq$ **s**, we want C_1, C_2 to be a **candidate pair**: We want them to hash to at **least 1 common bucket** (at least one band is identical)
- **Probability C_1, C_2 identical in one particular band:** $(0.8)^5 = 0.328$
- Probability C_1, C_2 are ***not*** similar in all 20 bands: $(1\text{-}0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
  - **We would find 99.965% pairs of truly similar documents**

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。

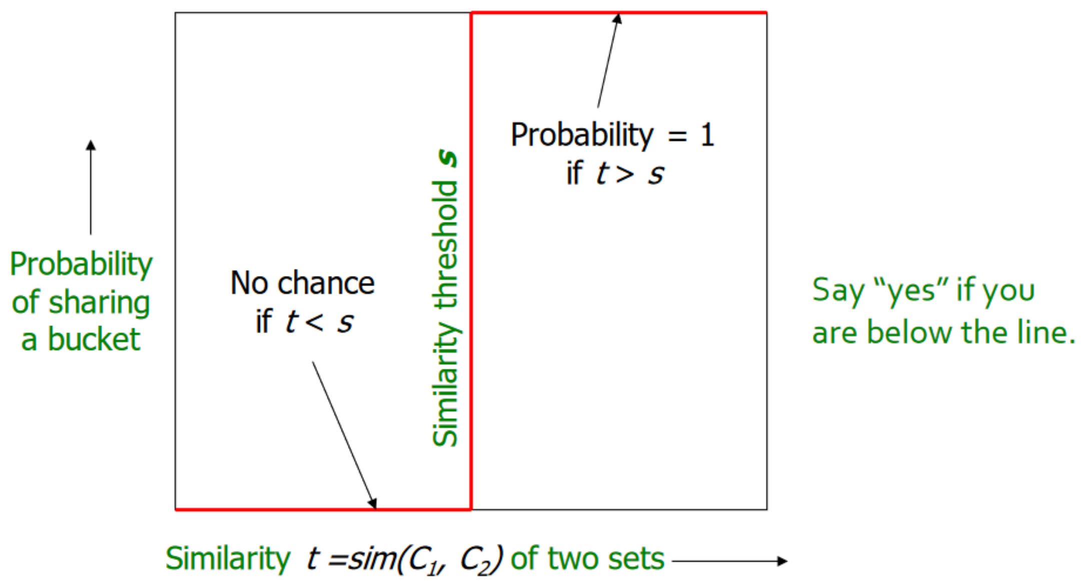


If C_1, C_2 are 30% Similar

2	1	4	1
1	2	1	2
2	1	2	1

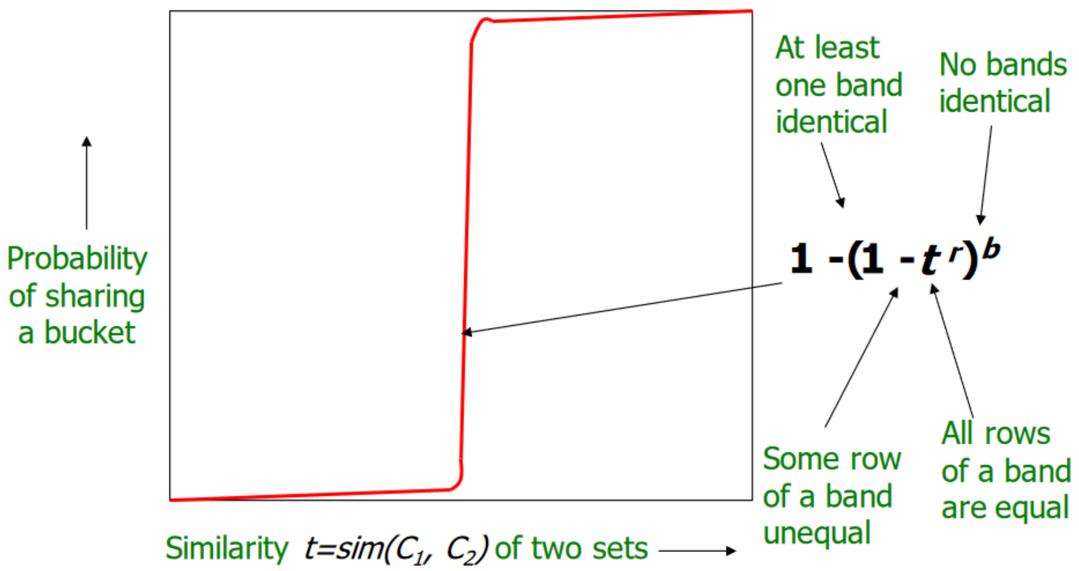
- **Find pairs of** $\geq$ ***s***=0.8 similarity, set **b**=20, **r**=5
- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < **s** we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- **Probability C_1, C_2 identical in one particular band:** $(0.3)^5 = 0.00243$
- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)^{20} = 0.0474$
  - In other words, approximately 4.74% pairs of docs with similarity 0.3 end up becoming **candidate pairs**
    - They are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold **s**

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。

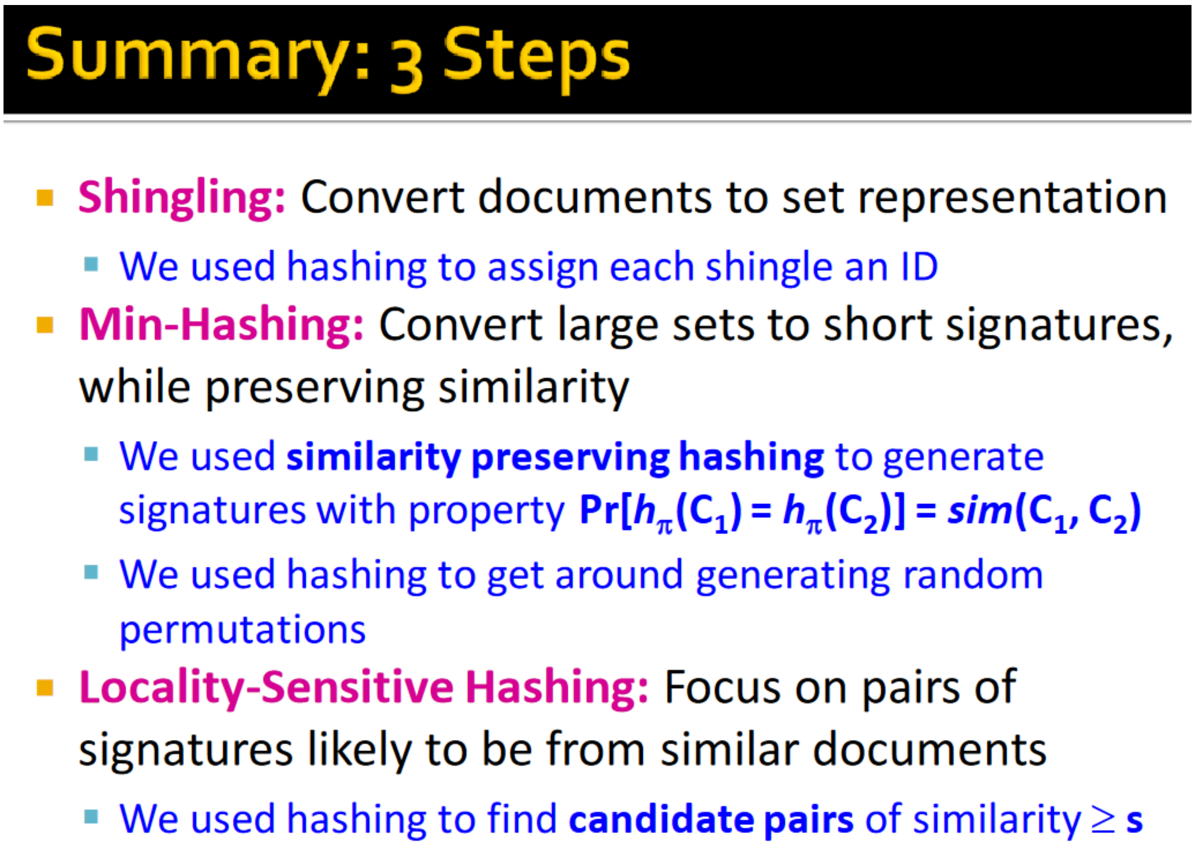

Probability of sharing a bucket
No chance if $t < s$
Similarity threshold s
Probability = 1 if $t > s$
Say "yes" if you are below the line.
Similarity $t = sim(C_1, C_2)$ of two sets

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


Probability of sharing a bucket
At least one band identical
No bands identical
$1-(1-t^r)^b$
Some row of a band unequal
All rows of a band are equal
Similarity $t=sim(C_1, C_2)$ of two sets

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 $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - 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。

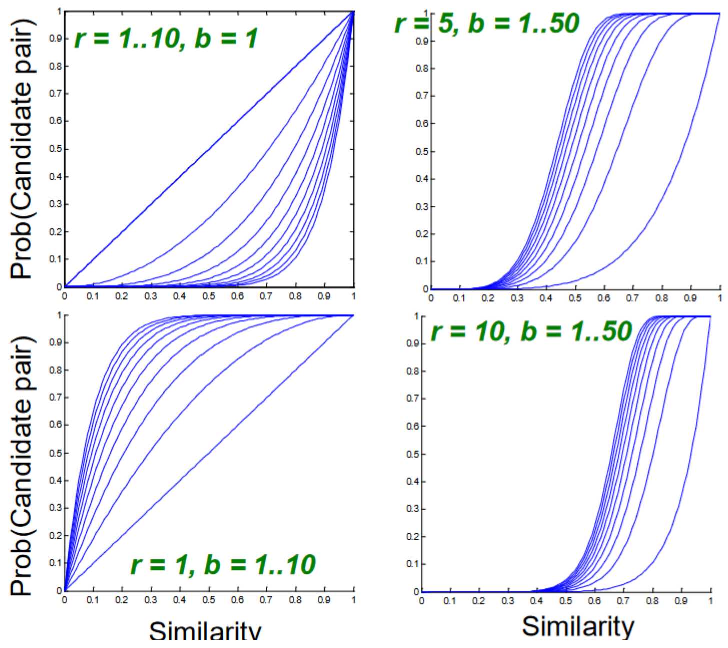
Theory of LSH

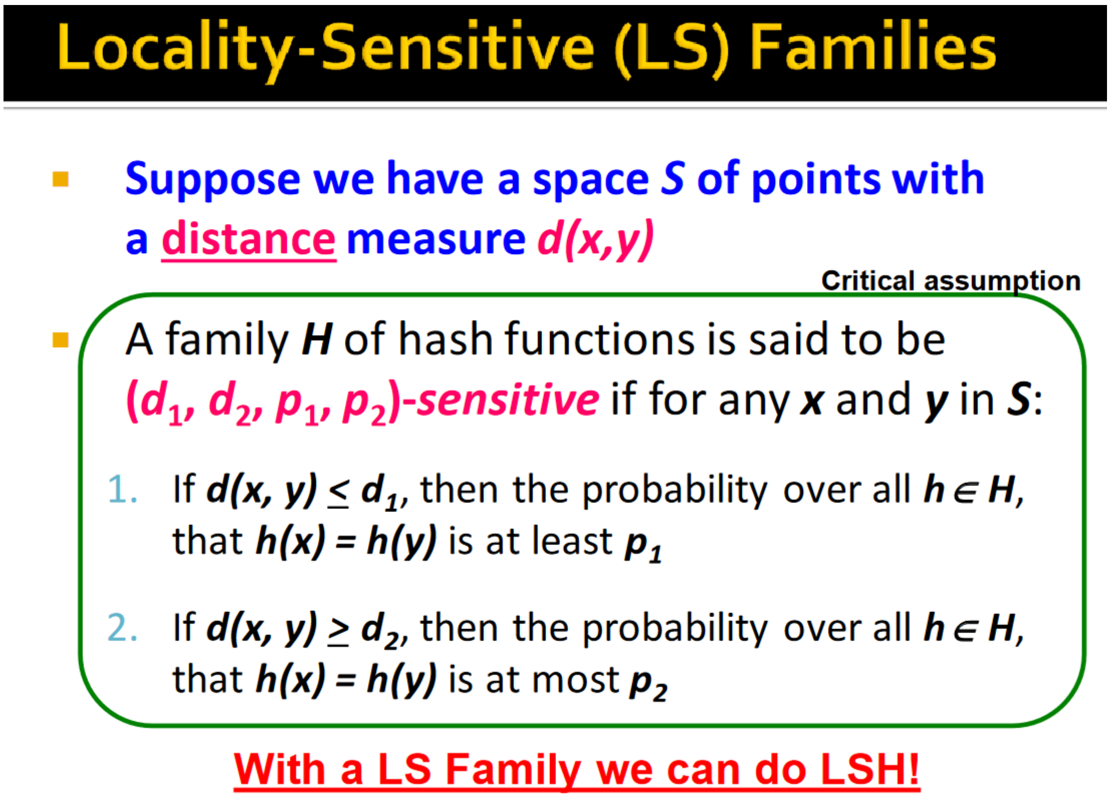
LSH的基本思想是：将原始数据空间中的两个相邻数据点通过相同的映射或投影变换（projection）后，这两个数据点在新的数据空间中仍然相邻的概率很大，而不相邻的数据点被映射到同一个桶的概率很小。也就是说，如果我们对原始数据进行一些hash映射后，我们希望原先相邻的两个数据能够被hash到相同的桶内，具有相同的桶号。对原始数据集合中所有的数据都进行hash映射后，我们就得到了一个hash table，这些原始数据集被分散到了hash table的桶内，每个桶会落入一些原始数据，属于同一个桶内的数据就有很大可能是相邻的，当然也存在不相邻的数据被hash到了同一个桶内。因此，如果我们能够找到这样一些hash functions，使得经过它们的哈希映射变换后，原始空间中相邻的数据落入相同的桶内的话，那么我们在该数据集合中进行近邻查找就变得容易了，我们只需要将查询数据进行哈希映射得到其桶号，然后取出该桶号对应桶内的所有数据，再进行线性匹配即可查找到与查询数据相邻的数据。换句话说，我们通过hash function映射变换操作，将原始数据集合分成了多个子集合，而每个子集合中的数据间是相邻的且该子集合中的元素个数较小，因此将一个在超大集合内查找相邻元素的问题转化为了在一个很小的集合内查找相邻元素的问题，显然计算量下降了很多。

S-curves as a func. of *b* and *r*

Given a fixed threshold ***t***.

We want choose ***r*** and ***b*** such that the ***P(Candidate pair)*** has a "step" right around ***t***.





Locality-Sensitive (LS) Families

- **Suppose we have a space *S* of points with a distance measure *d(x,y)***

Critical assumption

- A family ***H*** of hash functions is said to be ***(d_1, d_2, p_1, p_2)-sensitive*** if for any ***x*** and ***y*** in ***S***:
  1. If ***$d(x, y) \le d_1$***, then the probability over all ***$h \in H$***, that ***$h(x) = h(y)$*** is at least ***p_1***
  2. If ***$d(x, y) \ge d_2$***, then the probability over all ***$h \in H$***, that ***$h(x) = h(y)$*** is at most ***p_2***

With a LS Family we can do LSH!

那具有怎样特点的hash functions才能够使得原本相邻的两个数据点经过hash变换后会落入相同的桶内？这些hash function需要满足以下两个条件：

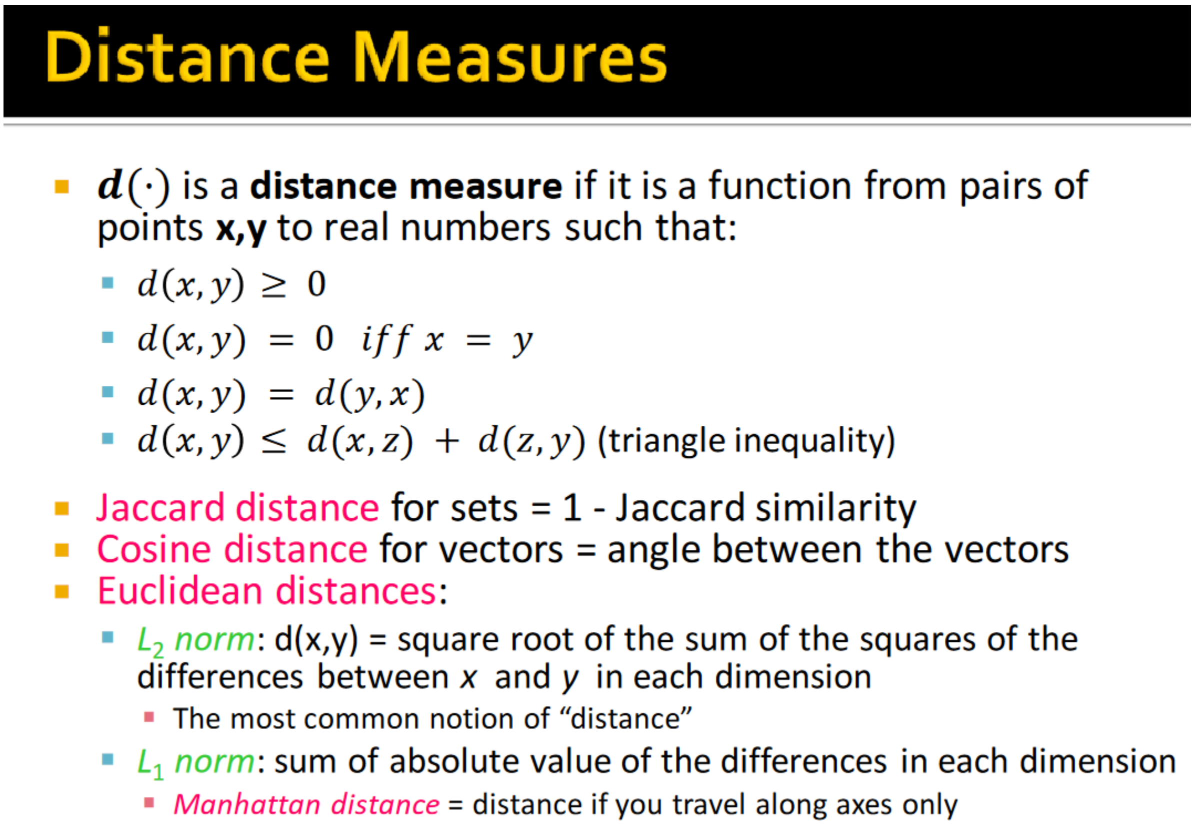
1）如果d(x,y) ≤ d1， 则h(x) = h(y)的概率至少为p1；

2）如果d(x,y) ≥ d2， 则h(x) = h(y)的概率至多为p2；

其中d(x,y)表示x和y之间的距离，d1 < d2， h(x)和h(y)分别表示对x和y进行hash变换。

满足以上两个条件的hash functions称为(d1,d2,p1,p2)-sensitive。而通过一个或多个(d1,d2,p1,p2)-sensitive的hash function对原始数据集合进行hashing生成一个或多个hash table的过程称为Locality-sensitive Hashing。

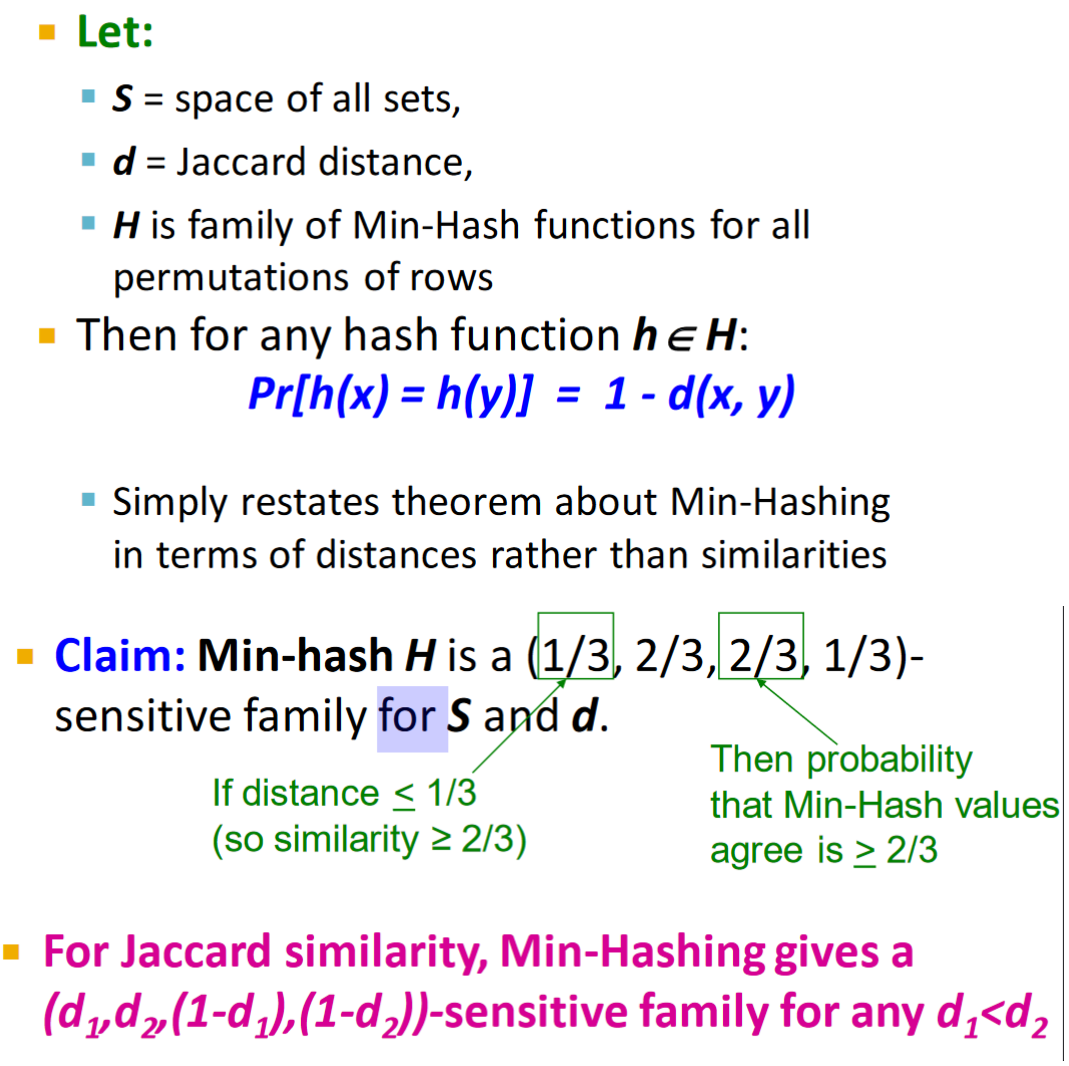
上述的$d(x, y)$可以有如下的表示方法：



Distance Measures

- $d(\cdot)$ is a **distance measure** if it is a function from pairs of points **x,y** to real numbers such that:
  - $d(x,y) \geq 0$
  - $d(x,y) = 0 \ iff \ x = y$
  - $d(x,y) = d(y,x)$
  - $d(x,y) \leq d(x,z) + d(z,y)$ (triangle inequality)
- Jaccard distance for sets = 1 - Jaccard similarity
- Cosine distance for vectors = angle between the vectors
- Euclidean distances:
  - L_2 *norm*: d(x,y) = square root of the sum of the squares of the differences between x and y in each dimension
    - The most common notion of "distance"
  - L_1 *norm*: sum of absolute value of the differences in each dimension
    - *Manhattan distance* = distance if you travel along axes only

关于LS-Family有如下的例子：



- **Let:**
  - ***S*** = space of all sets,
  - ***d*** = Jaccard distance,
  - ***H*** is family of Min-Hash functions for all permutations of rows
- Then for any hash function $h \in H$:

$$Pr[h(x) = h(y)] = 1 - d(x, y)$$

  - Simply restates theorem about Min-Hashing in terms of distances rather than similarities
- **Claim: Min-hash *H*** is a ($\boxed{1/3}$, 2/3, $\boxed{2/3}$, 1/3)-sensitive family for ***S*** and ***d***.

If distance $\leq$ 1/3
(so similarity $\geq$ 2/3)

Then probability that Min-Hash values agree is $\geq$ 2/3

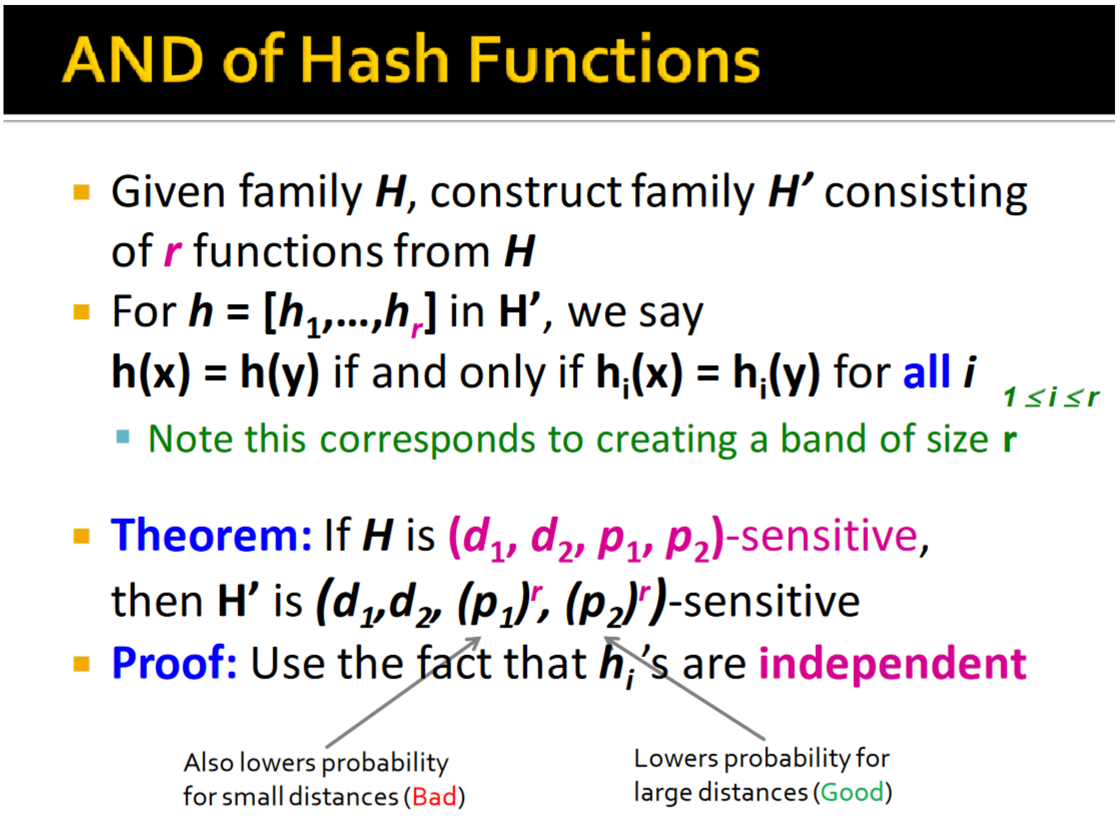
- **For Jaccard similarity, Min-Hashing gives a *(d_1,d_2,(1-d_1),(1-d_2))*-sensitive family for any *d_1<d_2***

我们现在了解了(d1,d2,p1,p2)-sensitive和LSH对应的图像。我们希望可以通过某些操作使得LSH图像在两端尽可能的逼近理想函数的图像。这就是增强LSH（Amplifying LSH。

通过LSH hash functions我们能够得到一个或多个hash table，每个桶内的数据之间是近邻的可能性很大。我们希望原本相邻的数据经过LSH hash后，都能够落入到相同的桶内，而不相邻的数据经过LSH hash后，都能够落入到不同的桶中。如果相邻的数据被投影到了不同的桶内，我们称为false negtive；如果不相邻的数据被投影到了相同的桶内，我们称为false positive。因此，我们在使用LSH中，我们希望能够尽量降低false negtive rate和false positive rate。

Amplify LSH通常由两种构造方法：

- AND construction like "rows in a band"
- OR construction like "many bands"



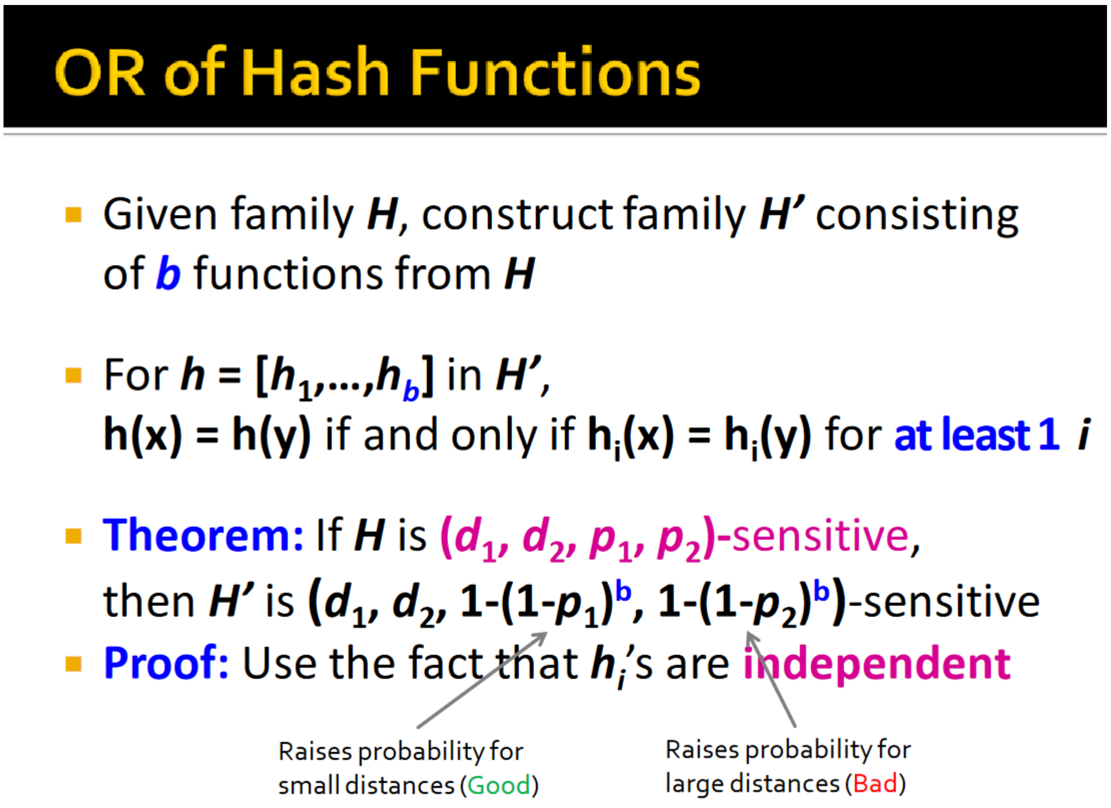
AND of Hash Functions

- Given family *H*, construct family *H'* consisting of *r* functions from *H*
- For *h* = [$h_1,\dots,h_r$] in H', we say h(x) = h(y) if and only if $h_i(x) = h_i(y)$ for all *i* $1 \le i \le r$
  - Note this corresponds to creating a band of size **r**
- **Theorem:** If *H* is (d_1, d_2, p_1, p_2)-sensitive, then H' is $(d_1,d_2, (p_1)^r, (p_2)^r)$-sensitive
- **Proof:** Use the fact that h_i's are **independent**

Also lowers probability for small distances (Bad)
Lowers probability for large distances (Good)

从同一个LSH function family中挑选出k个LSH function，H(X) = H(Y)有且仅当这k个Hi(X) = Hi(Y)都满足。也就是说只有当两个数据的这k个hash值都对应相同时，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。

现有Hash Family H is (d1,d2,p1,p2)-sensitive，构造含有r个function的Family H'，则H' is $(d1, d2, (p1)^r, (p2)^r - sensitive$ 由于概率值p小于等于1，那么在增大r的同时p_1 p_2的值也会相应减小，根据sensitive的定义，我们不希望减小$(p_1)^r$，但是希望减小$(p_2)^r$。



OR of Hash Functions

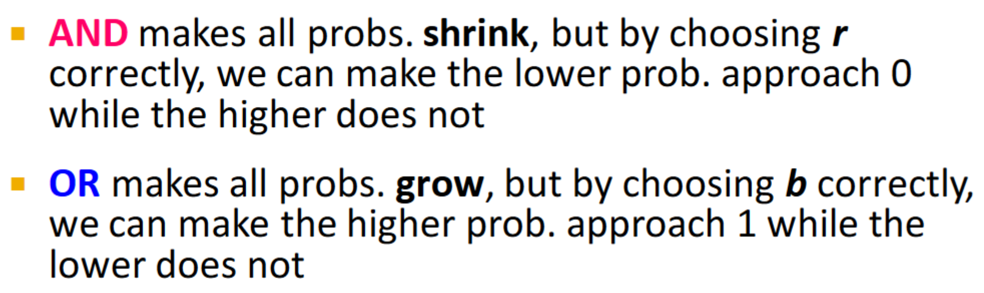
- Given family ***H***, construct family ***H'*** consisting of ***b*** functions from ***H***
- For ***h* = [*h*$_1$,…,*h*$_b$]** in ***H'***, **h(x) = h(y)** if and only if **h$_i$(x) = h$_i$(y)** for **at least 1 *i***
- **Theorem:** If ***H*** is **(d_1, d_2, p_1, p_2)**-sensitive, then ***H'*** is **(d_1, d_2, $1-(1-p_1)^b$, $1-(1-p_2)^b$)**-sensitive
- **Proof:** Use the fact that ***h$_i$***'s are **independent**

Raises probability for small distances (Good)

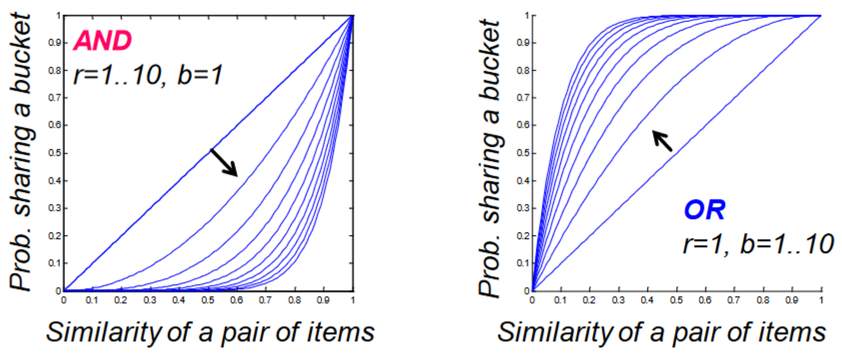
Raises probability for large distances (Bad)

从同一个LSH function family中挑选出k个LSH function，H(X) = H(Y)有且仅当存在一个以上的Hi(X) = Hi(Y)。也就是说只要两个数据的这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。

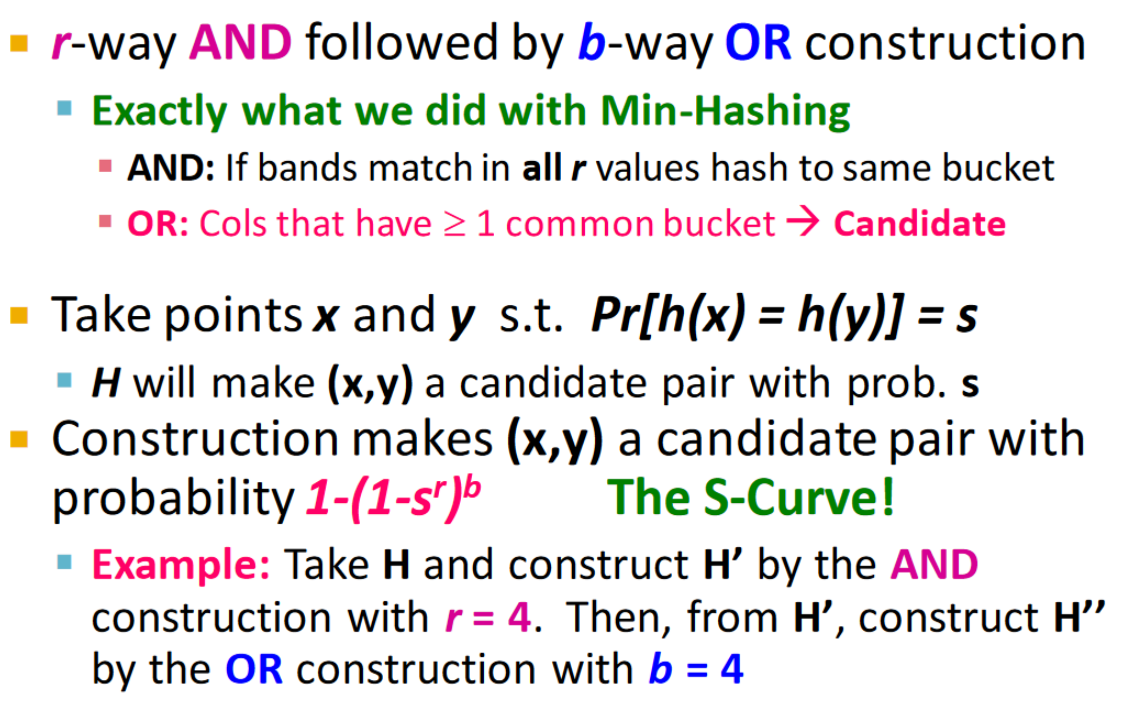
现有Hash Family H is (d1,d2,p1,p2)-sensitive，构造含有r个function的Family H'，则H' is $(d1, d2, 1-(1-p1)^b, 1-(1-p2)^b - sensitive$。同样的，OR操作增大了$1-(1-p1)^b$和$1-(1-p2)^b$。



- **AND** makes all probs. **shrink**, but by choosing ***r*** correctly, we can make the lower prob. approach 0 while the higher does not
- **OR** makes all probs. **grow**, but by choosing ***b*** correctly, we can make the higher prob. approach 1 while the lower does not



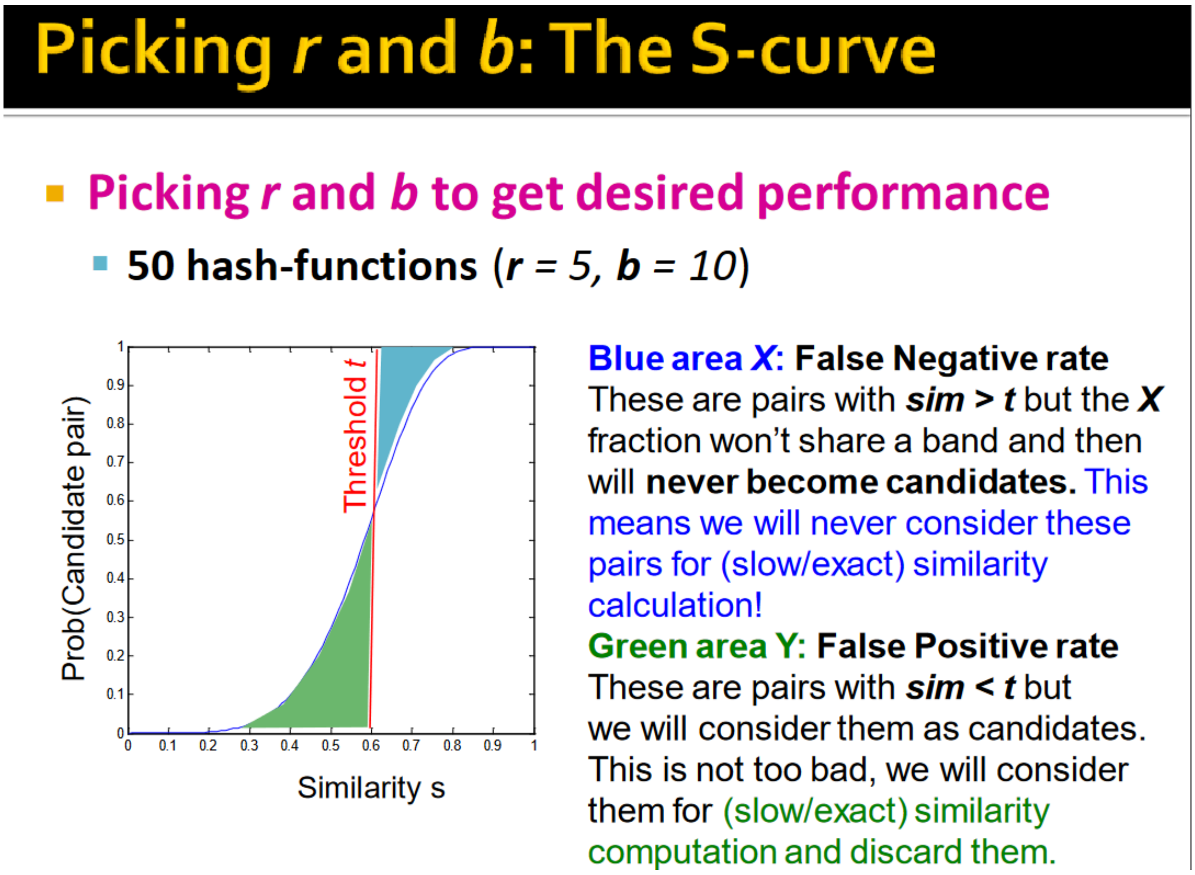
我们希望在增加p_1的同时减小p_2。因此Min-Hashing就是结合了AND和OR操作，使得函数图像的两端向坐标轴逼近。



- r-way **AND** followed by b-way **OR** construction
  - **Exactly what we did with Min-Hashing**
    - **AND:** If bands match in **all** r values hash to same bucket
    - **OR:** Cols that have ≥ 1 common bucket → **Candidate**
- Take points x and y s.t. $Pr[h(x) = h(y)] = s$
  - H will make **(x,y)** a candidate pair with prob. **s**
- Construction makes **(x,y)** a candidate pair with probability $1-(1-s^r)^b$ **The S-Curve!**
  - **Example:** Take **H** and construct **H'** by the **AND** construction with $r = 4$. Then, from **H'**, construct **H''** by the **OR** construction with $b = 4$

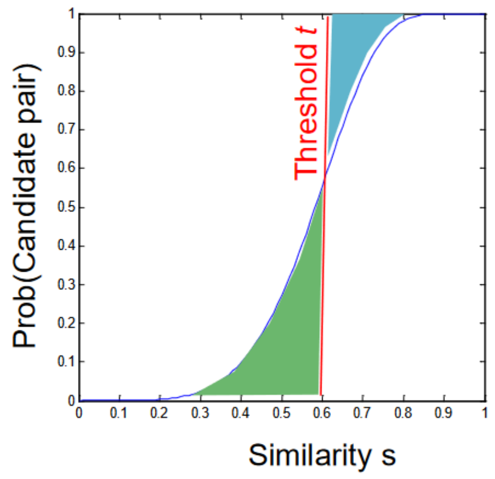
band内所有的值都必须相同可以看做是AND操作，而只要有一个band相似的向量可以被列入candidate可以看做是OR操作。我们通过构造了$1-(1-s^r)^b$函数来构造了我们期望的S型曲线。

那么我们该如何调整超参数b和r呢?



Picking r and b: The S-curve

- **Picking r and b to get desired performance**
  - **50 hash-functions** ($r = 5$, $b = 10$)



Blue area X: False Negative rate
These are pairs with ***sim > t*** but the ***X*** fraction won't share a band and then will **never become candidates.** This means we will never consider these pairs for (slow/exact) similarity calculation!
Green area Y: False Positive rate
These are pairs with ***sim < t*** but we will consider them as candidates. This is not too bad, we will consider them for (slow/exact) similarity computation and discard them.

S型曲线和垂直于y轴阈值的垂线将图像分为两片区域，蓝色区域称之为False Negative Area，这些区域内的向量similarity大于阈值但是任意一个band都不是相同的，即他们不会成为candidate pairs。绿色区域称之为False Positive Area，向量之间的similarity小于阈值但是至少有一个band是相同的，即他们会成为candidate pairs。常见的逻辑是尽可能的减小False Negative的面积，因为落入了蓝色区域的向量不会成为candidate pairs意味我们没有机会找回原本相似的向量组合，但是落入绿色区域的向量组合即使不是相似的向量，我们也可以在 计算相似度的时候将他们筛除。

使用LSH进行对海量数据建立索引（Hash table）并通过索引来进行近似最近邻查找的过程如下:

A .离线建立索引

1. 选取满足(d1,d2,p1,p2)-sensitive的LSH hash functions；
2. 根据对查找结果的准确率（即相邻的数据被查找到的概率）确定hash table的个数L，每个table内的hash functions的个数K，以及跟LSH hash function自身有关的参数；
3. 将所有数据经过LSH hash function哈希到相应的桶内，构成了一个或多个hash table；

B .在线查找

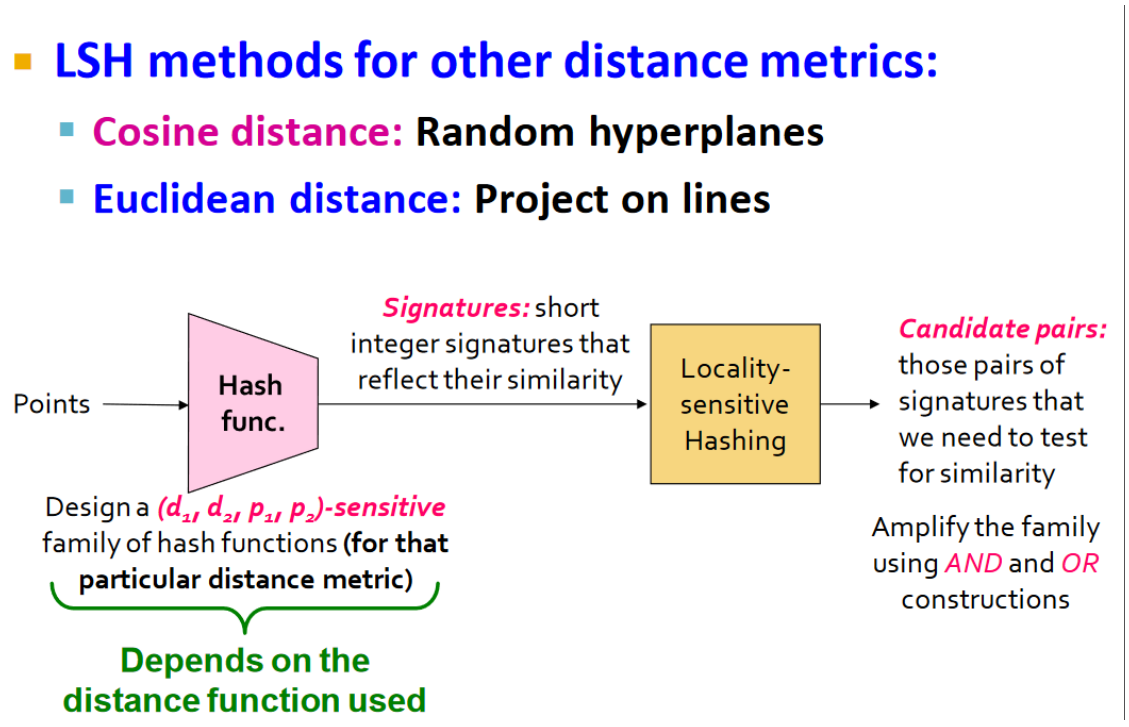
1. 将查询数据经过LSH hash function哈希得到相应的桶号；
2. 将桶号中对应的数据取出；（为了保证查找速度，通常只需要取出前2L个数据即可）；
3. 计算查询数据与这2L个数据之间的相似度或距离，返回最近邻的数据；

LSH在线查找时间由两个部分组成：（1）通过LSH hash functions计算hash值（桶号）的时间；（2）将查询数据与桶内的数据进行比较计算的时间。因此，LSH的查找时间至少是一个sublinear时间。为什么是“至少”？因为我们可以通过对桶内的属于建立索引来加快匹配速度，这时第（2）部分的耗时就从O(N)变成了O(logN)或O(1)（取决于采用的索引方法）。

LSH为我们提供了一种在海量的高维数据集中查找与查询数据点（query data point）近似最相邻的某个或某些数据点。需要注意的是，LSH并不能保证一定能够查找到与query data point最相邻的数据，而是减少需要匹配的数据点个数的同时保证查找到最近邻的数据点的概率很大。

LSH for other distance metrics

上述的LSH算法只是应用于Jaccard Distance，是否可以将LSH扩展到其他的距离度量方法上？

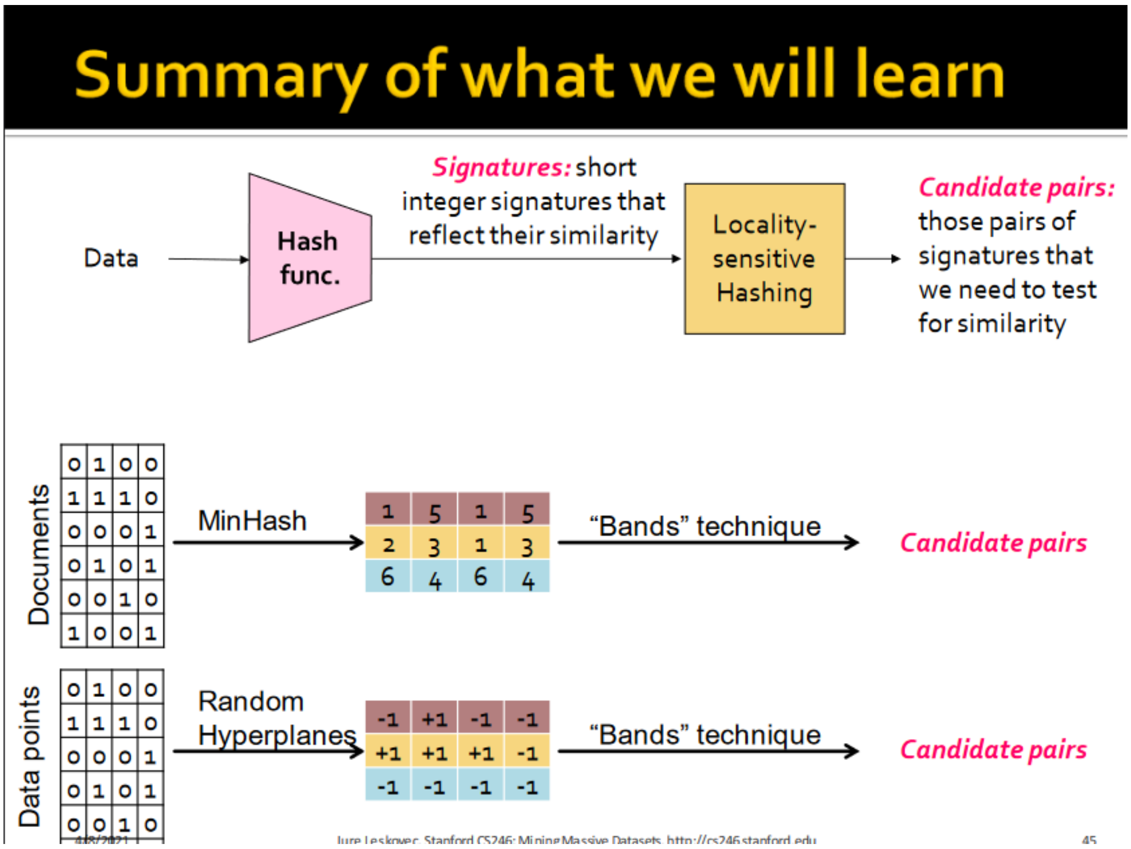


- **LSH methods for other distance metrics:**
  - **Cosine distance: Random hyperplanes**
  - **Euclidean distance: Project on lines**

Points
Hash func.
Signatures: short integer signatures that reflect their similarity
Locality-sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity
Design a *(d_1, d_2, p_1, p_2)-sensitive* family of hash functions **(for that particular distance metric)**
Depends on the distance function used
Amplify the family using *AND* and *OR* constructions

我们将扩展两种距离的度量方式。

- Cosine distance 余弦距离
- Euclidean distance 欧氏距离



Summary of what we will learn

Data
Hash func.
Signatures: short integer signatures that reflect their similarity
Locality-sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity
Documents
MinHash
"Bands" technique
Candidate pairs
Data points
Random Hyperplanes
"Bands" technique
Candidate pairs

0	1	0	0
1	1	1	0
0	0	0	1
0	1	0	1
0	0	1	0
1	0	0	1

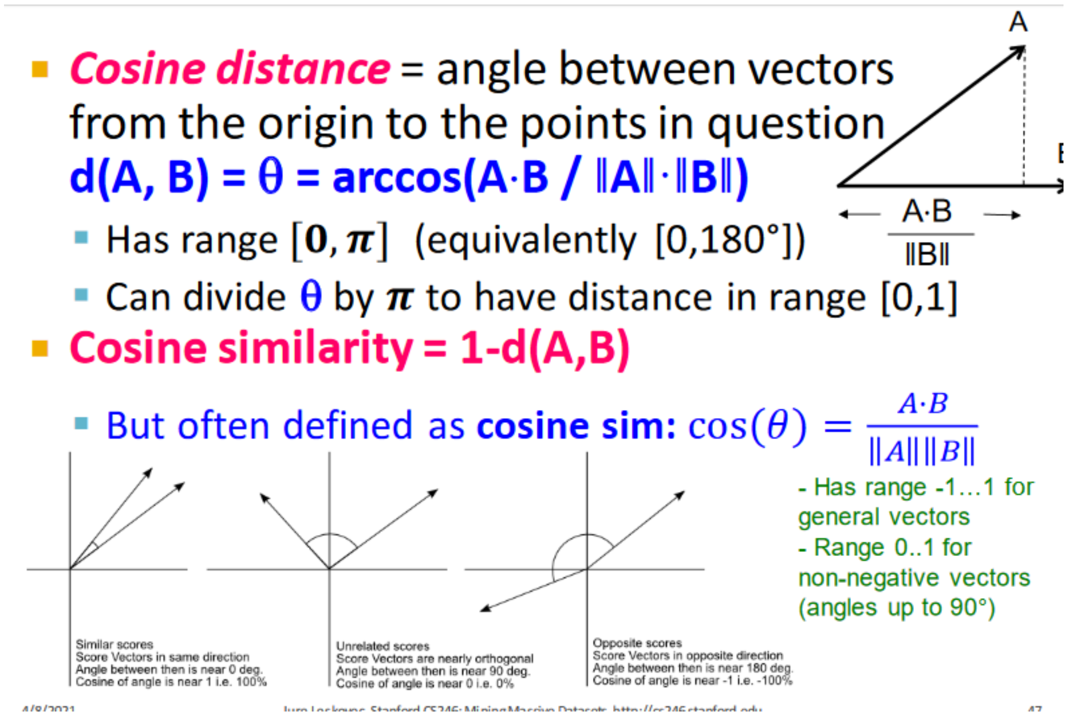
1	5	1	5
2	3	1	3
6	4	6	4

0	1	0	0
1	1	1	0
0	0	0	1
0	1	0	1
0	0	1	0

-1	+1	-1	-1
+1	+1	+1	-1
-1	-1	-1	-1

4/8/2021 Jure Leskovec, Stanford CS246: Mining Massive Datasets, http://cs246.stanford.edu 45

余弦距离定义：实际上定义的是向量的投影长度。



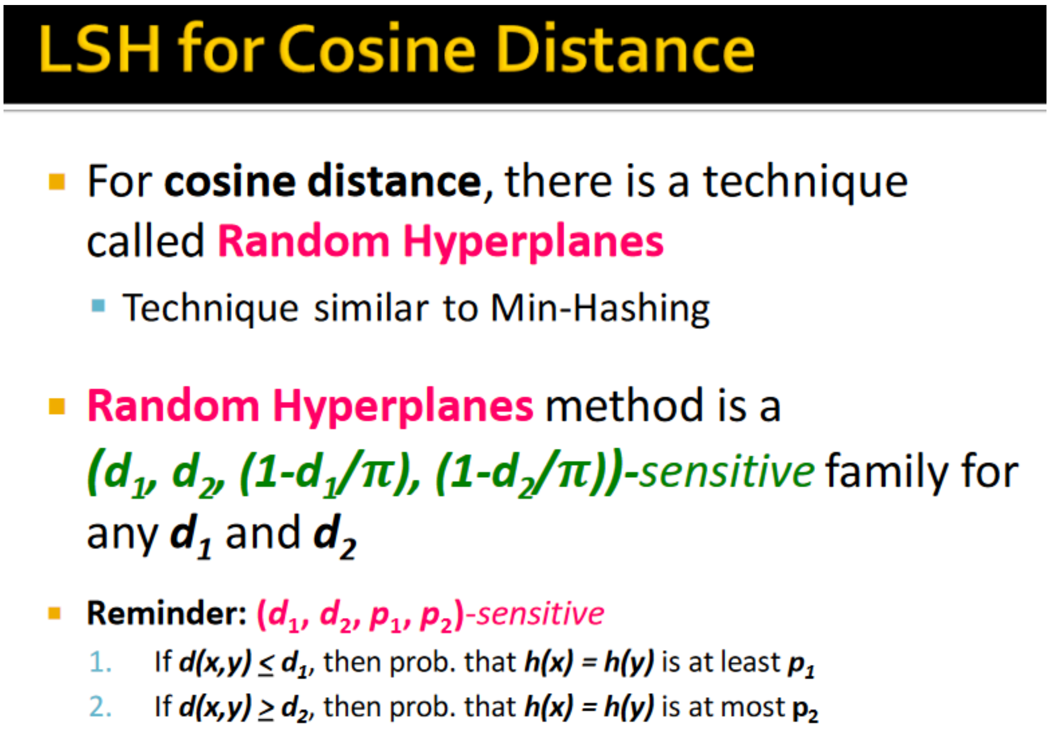
- *Cosine distance* = angle between vectors from the origin to the points in question
  d(A, B) = θ = arccos(A·B / ‖A‖·‖B‖)
  - Has range $[\mathbf{0}, \boldsymbol{\pi}]$ (equivalently [0,180°])
  - Can divide θ by $\boldsymbol{\pi}$ to have distance in range [0,1]
- **Cosine similarity = 1-d(A,B)**
  - But often defined as **cosine sim:** $\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$

- Has range -1…1 for general vectors
- Range 0..1 for non-negative vectors (angles up to 90°)

A
B
$\frac{A \cdot B}{\|B\|}$
Similar scores
Score Vectors in same direction
Angle between then is near 0 deg.
Cosine of angle is near 1 i.e. 100%
Unrelated scores
Score Vectors are nearly orthogonal
Angle between then is near 90 deg.
Cosine of angle is near 0 i.e. 0%
Opposite scores
Score Vectors in opposite direction
Angle between then is near 180 deg.
Cosine of angle is near -1 i.e. -100%

4/8/2021 Jure Leskovec, Stanford CS246: Mining Massive Datasets, http://cs246.stanford.edu 47

对于余弦距离而言，我们引入随机超平面 Random Hyperplanes的概念。

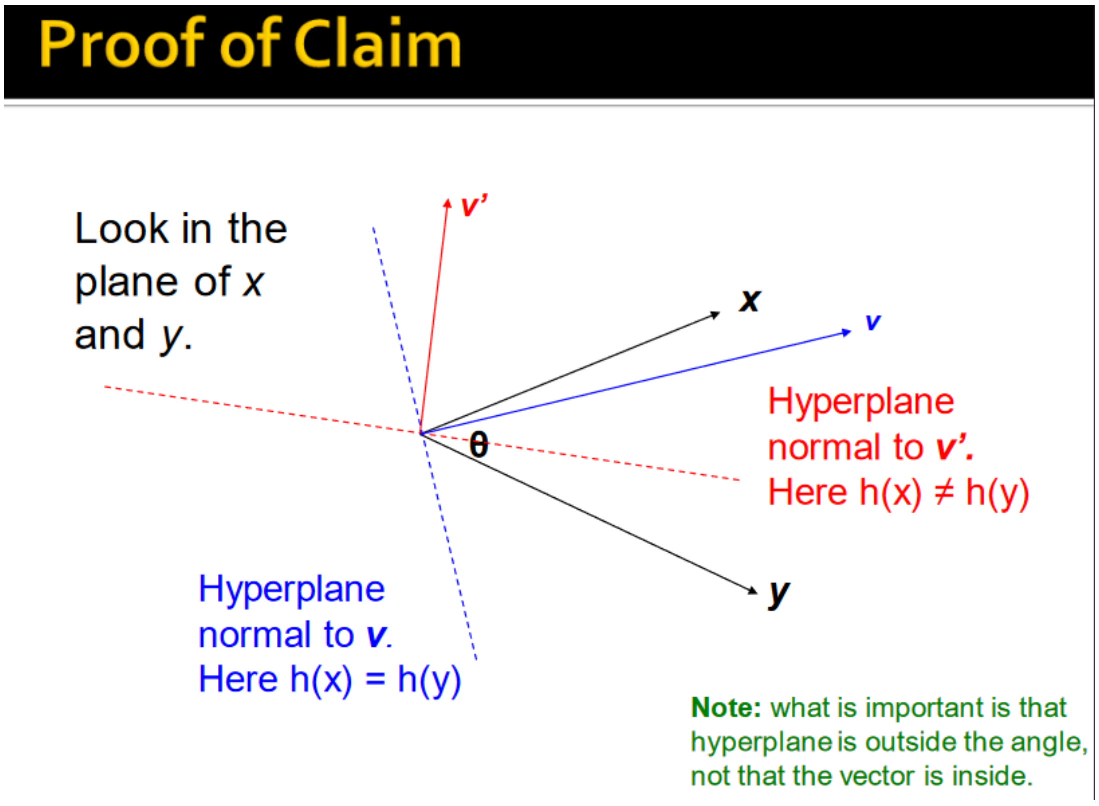


LSH for Cosine Distance

- For **cosine distance**, there is a technique called **Random Hyperplanes**
  - Technique similar to Min-Hashing
- **Random Hyperplanes** method is a ***$(d_1, d_2, (1-d_1/\pi), (1-d_2/\pi))$-sensitive*** family for any d_1 and d_2
- **Reminder:** **(d_1, d_2, p_1, p_2)***-sensitive*
  1. If $d(x,y) \leq d_1$, then prob. that $h(x) = h(y)$ is at least p_1
  2. If $d(x,y) \geq d_2$, then prob. that $h(x) = h(y)$ is at most p_2

首先随机构造向量v，定义Hash Function为$h_v(x) = +1\ if\ vx >= 0; = -1\ if\ vx <= 0$。我们可以得到如下性质：

对于任意点x，y： $Pr[h(x) = h(y)] = 1 - d(x, y)/\pi$ 其中$d(x, y)$为cosine distance。



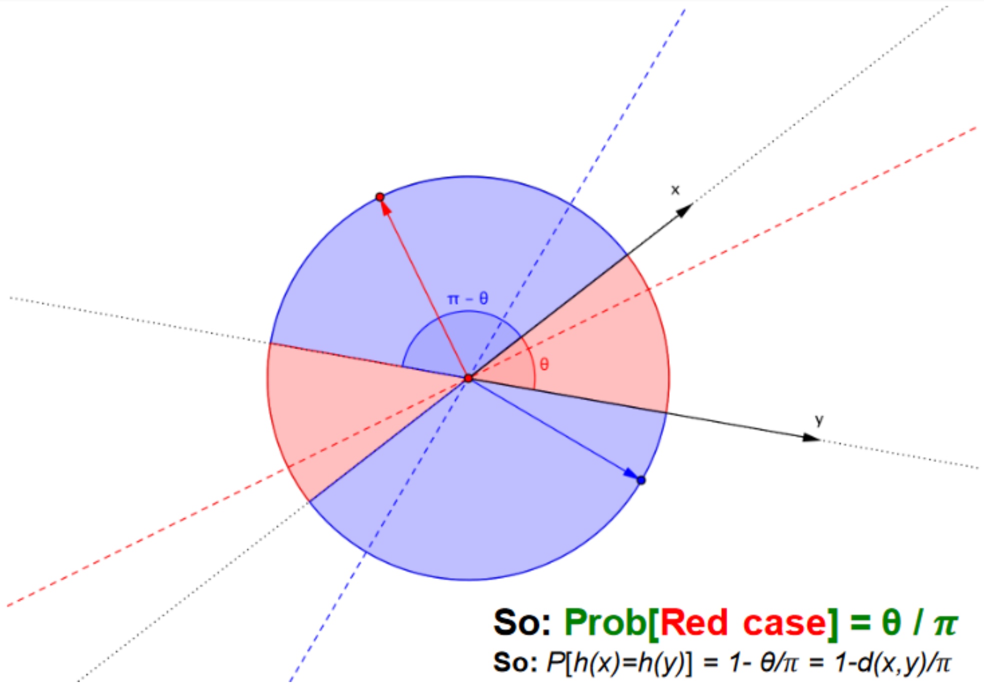
Proof of Claim

Look in the plane of *x* and *y*.

v'
x
v
θ
y
Hyperplane normal to v'. Here h(x) ≠ h(y)
Hyperplane normal to v. Here h(x) = h(y)

Note: what is important is that hyperplane is outside the angle, not that the vector is inside.

由上图我们可知，随机向量v对应的超平面如果将x y向量划分在超平面同侧那么$h(x) == h(y)$，如果划分在异侧则$h(x) != h(y)$

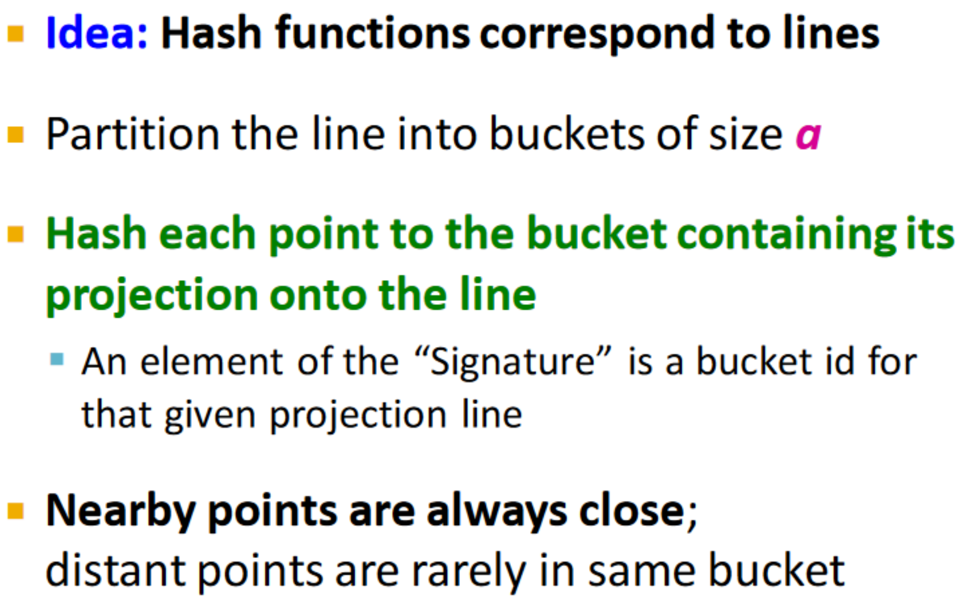

π - θ
θ
x
y
So: Prob[Red case] = θ / π
So: P[h(x)=h(y)] = 1- θ/π = 1-d(x,y)/π

consine distance形成signature向量经过如下流程：

- 随机选择向量v，选择v的数目为signature向量的维度
- 根据向量内积结果得出signature向量（01向量）
- 应用AND OR construction进行LSH操作

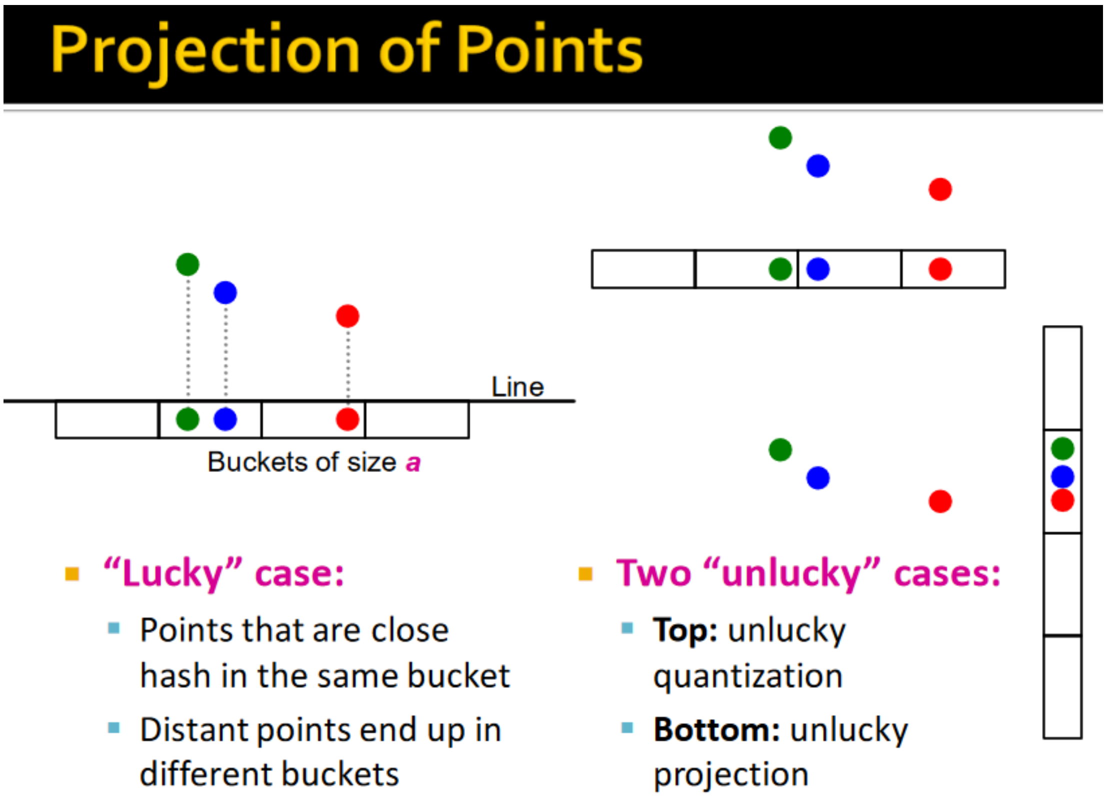
对于v向量的生成，可以将v向量定义为+1，-1向量。原因是当M矩阵维度很高时，我们不需要去随机生成M个随机数且+/-1向量cover the space evenly and not bias in any way。

应用欧氏距离的LSH想法如下：



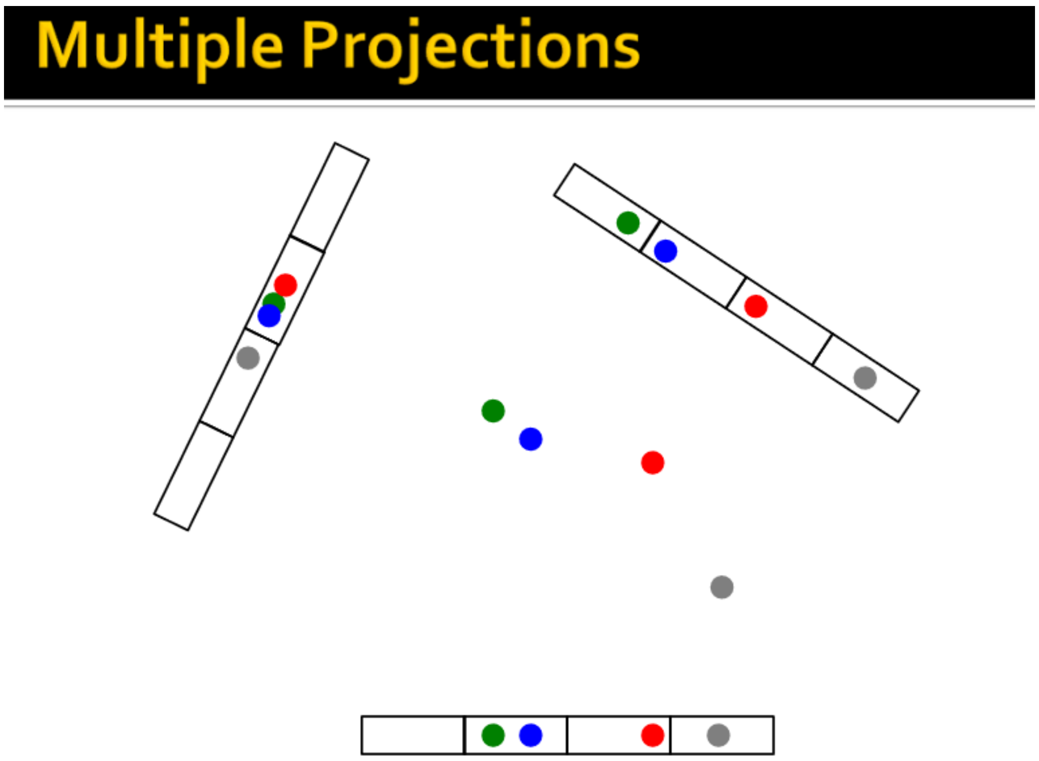
- **Idea: Hash functions correspond to lines**
- Partition the line into buckets of size *a*
- **Hash each point to the bucket containing its projection onto the line**
  - An element of the “Signature” is a bucket id for that given projection line
- **Nearby points are always close;** distant points are rarely in same bucket

对于空间中的点x，我们给出其中的一个投影方向，将这些点的投影结果划分为不同的bucket。如果投影方向设置的恰当，不同点的投影结果就会落到不同的bucket中，如果方向设置的不恰当，不同点的投影结果回落到相同的bucket中。那么如果点在空间中的欧氏距离不同，投影结果落入的bucket也应该是大概率不同的。



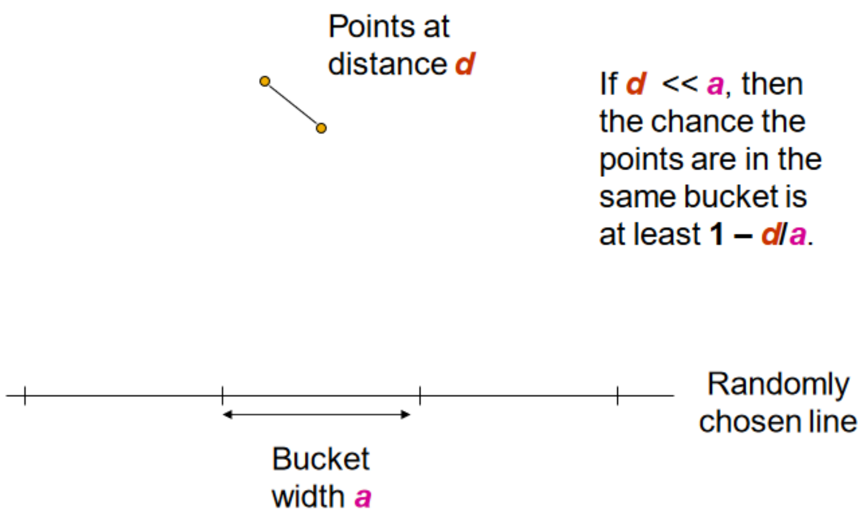
Projection of Points

- **"Lucky" case:**
  - Points that are close hash in the same bucket
  - Distant points end up in different buckets
- **Two "unlucky" cases:**
  - **Top:** unlucky quantization
  - **Bottom:** unlucky projection

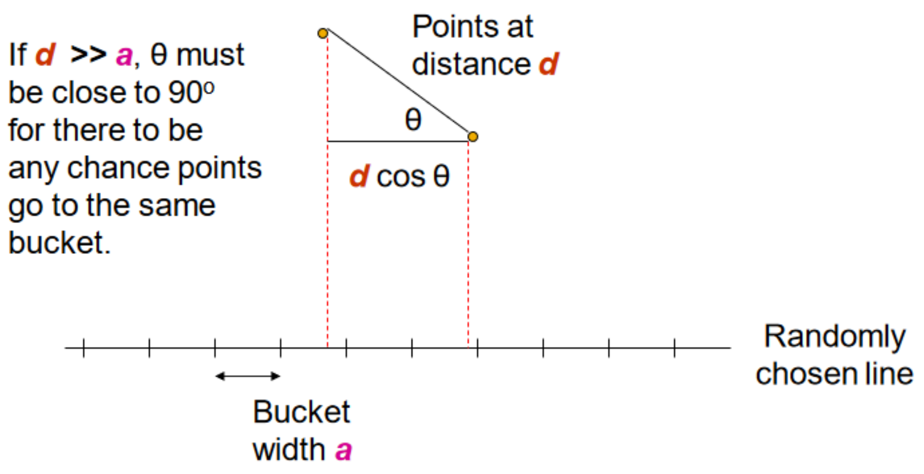


Multiple Projections

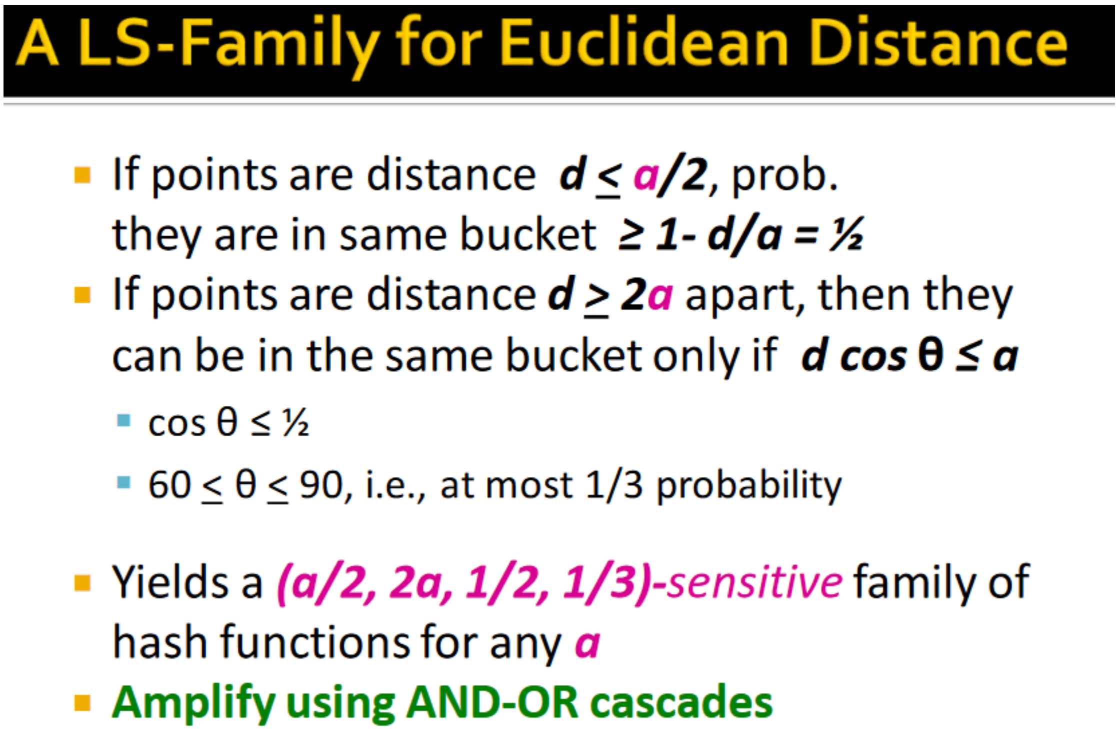
现在假设空间中两点的欧氏距离为d，bucket的宽度为a。如果d远小于a，那么这两个点在投影之后share same bucket的概率则至少为$1-d/a$,即投影方向与两点连线的方向平行。



如果d远大于a，那么两点连线与投影方向的夹角θ需要接近90度才可以使得两点的投影结果落入相同的bucket



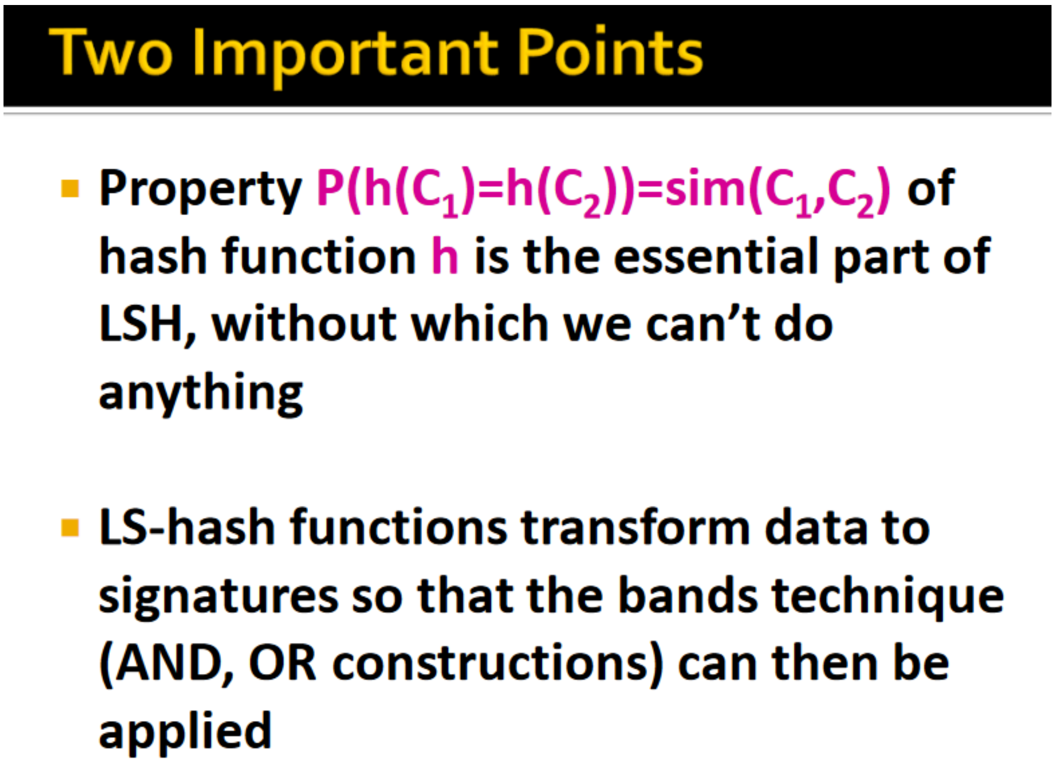
我们可以对任意的band width a 得到一组$(a/2, 2a, 1/2, 1/3)-sensitive$ family。选择多组Randomly chosen line我们就可以应用AND OR construction来构造LSH。



A LS-Family for Euclidean Distance

- If points are distance $d \leq a/2$, prob. they are in same bucket $\geq 1 - d/a = ½$
- If points are distance $d \geq 2a$ apart, then they can be in the same bucket only if $d \cos \theta \leq a$
  - $\cos \theta \leq ½$
  - $60 \leq \theta \leq 90$, i.e., at most 1/3 probability
- Yields a *(a/2, 2a, 1/2, 1/3)-sensitive* family of hash functions for any *a*
- **Amplify using AND-OR cascades**

最后LSH理论有两点最重要的性质：



Two Important Points

- **Property $P(h(C_1)=h(C_2))=sim(C_1,C_2)$ of hash function h is the essential part of LSH, without which we can’t do anything**
- **LS-hash functions transform data to signatures so that the bands technique (AND, OR constructions) can then be applied**

$P(h(C_1) = h(C_2)) = sim(C_1, C_2)$即两个列向量的hash结果相同的概率等于他们之间的similarity。这是所有LSH算法的核心思想。LS-hash fuctions可以将数据转化为signature向量而保存他们之间的相似性，因此我们才可以使用AND OR操作来增强LSH算法的性能。