



中国科学院大学

University of Chinese Academy of Sciences

Mining Massive datasets Finding Similar Items: Locality Sensitive Hashing



Outline

3.0 Motivation

3.1 Finding Similar Items

3.1.1 Shingling

3.1.2 Min-Hashing

3.1.3 Locality-sensitive Hashing

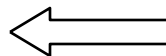
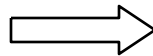
3.2 Theory of LSH

3.3 Amplifying Hash Functions: AND and OR

3.4 LSH for other distance metrics



Scene Completion Problem



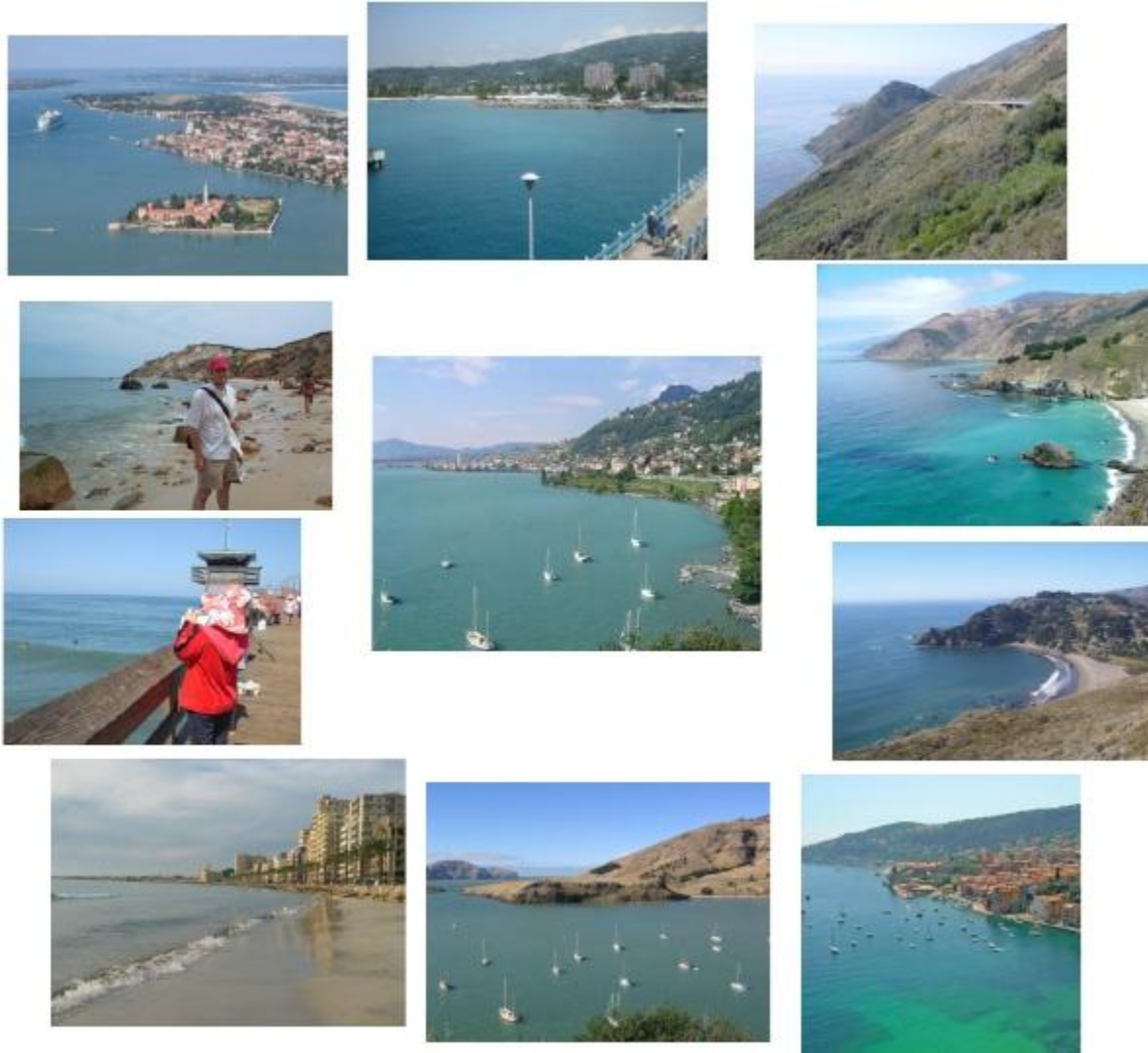
Scene Completion Problem



10 nearestneighbors from a collection of 20,000 images



Scene Completion Problem



10 nearestneighbors from a collection of 2 million images



Motivation

- **Finding similar documents/webpages/images**
 - (Approximate) mirror sites.
Application: Don't want to show both when Google.
 - Plagiarism, large quotations
Application: I am sure you know
 - Similar topic/articles from various places
Application: Cluster articles by "same story".
 - Google image
- **Social network analysis**
 - Finding NetFlix users with similar tastes in movies.
 - e.g., for personalized recommendation systems.



A Common Metaphor

- Many problems can be expressed as finding "similar" sets:
 - Find near-neighbors in high-dim space
- **Example**
 - Pages with similar words
 - For duplicate detection, classification by topic
 - Customers who purchased similar products
 - Products with similar customer sets
 - Images with similar features
 - Users who visited similar websites



How can we compare similarity?

Convert the data (homework, webpages, images) into an object in an **abstract space** that we know how to measure distance, and how to do it efficiently.

What abstract space?

For example, the Euclidean space over \mathbb{R}^d
 L^1 space, L^∞ space, ...



Problem for today's lecture

- Given: High dimensional data points x_1, x_2
 - For example: Image is a long vector of pixel colors
 - $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1 \ 2 \ 1 \ 0 \ 2 \ 1 \ 0 \ 1 \ 0]$
- And some distance function
 - Which quantifies the “distance” between x_1 and x_2
- Goal: Find **all pairs of data points** (x_i, x_j) that are within some distance threshold $d(x_i, x_j) \leq s$
- Note: **Naïve solution would take $O(N^2)$** where N is the number of data points
- **MAGIC: This can be done in $O(N)$!! How?**



Outline

3.0 Motivation

3.1 Finding Similar Items

3.1.1 Shingling

3.1.2 Min-Hashing

3.1.3 Locality-sensitive Hashing

3.2 Theory of LSH

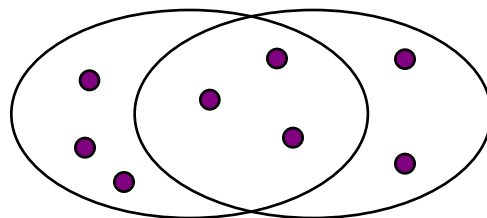
3.3 Amplifying Hash Functions: AND and OR

3.4 LSH for other distance metrics



Distance Measures

- **Goal:** Find near-neighbors in high-dim. space
 - We formally define "near neighbors" as points that are a "small distance" apart
- For each application, we first need to define what "distance" means
- **Today:** Jaccard distance/similarity
 - The **Jaccard similarity** of two sets is the size of their intersection divided by the size of their union:
$$\text{sim}(C1, C2) = |C1 \cap C2| / |C1 \cup C2|$$
 - **Jaccard distance:** $d(C1, C2) = 1 - |C1 \cap C2| / |C1 \cup C2|$



3 in intersection

8 in union

Jaccard similarity = $3/8$

Jaccard distance = $5/8$



Jaccard similarity with clusters

- Consider two sets $A = \{0,1,2,5,6\}$, $B = \{0,2,3,5,7,9\}$. What is the Jaccard similarity of A and B ?
- With clusters: We may have some items which basically represent the same thing. We group objects to clusters. E.g.,

$$C1 = \{0,1,2\}; C2 = \{3,4\}; C3 = \{5,6\}; C4 = \{7,8,9\}$$

For instance: $C1$ may represent action movies, $C2$ may represent comedies, $C3$ may represent documentaries, $C4$ may represent horror movies.

Now we can represent $A_{clu} = \{C1, C3\}$, $B_{clu} = \{C1, C2, C3, C4\}$

$$JS_{clu}(A; B) = JS(A_{clu}; B_{clu}) = |\{C1, C2\}| / |\{C1, C2, C3, C4\}| = 0.5$$



Finding Similar Documents

- **Goal:** Given a large number (N in the millions or billions) of documents, find “near duplicate” pairs
- **Applications:**
 - Mirror websites, or approximate mirrors
 - Don't want to show both in search results
 - Similar news articles at many news sites
 - Cluster articles by “same story”
- **Problems:**
 - Many small pieces of one document can appear out of order in another
 - Too many documents to compare all pairs
 - Documents are so large or so many that they cannot fit in main memory

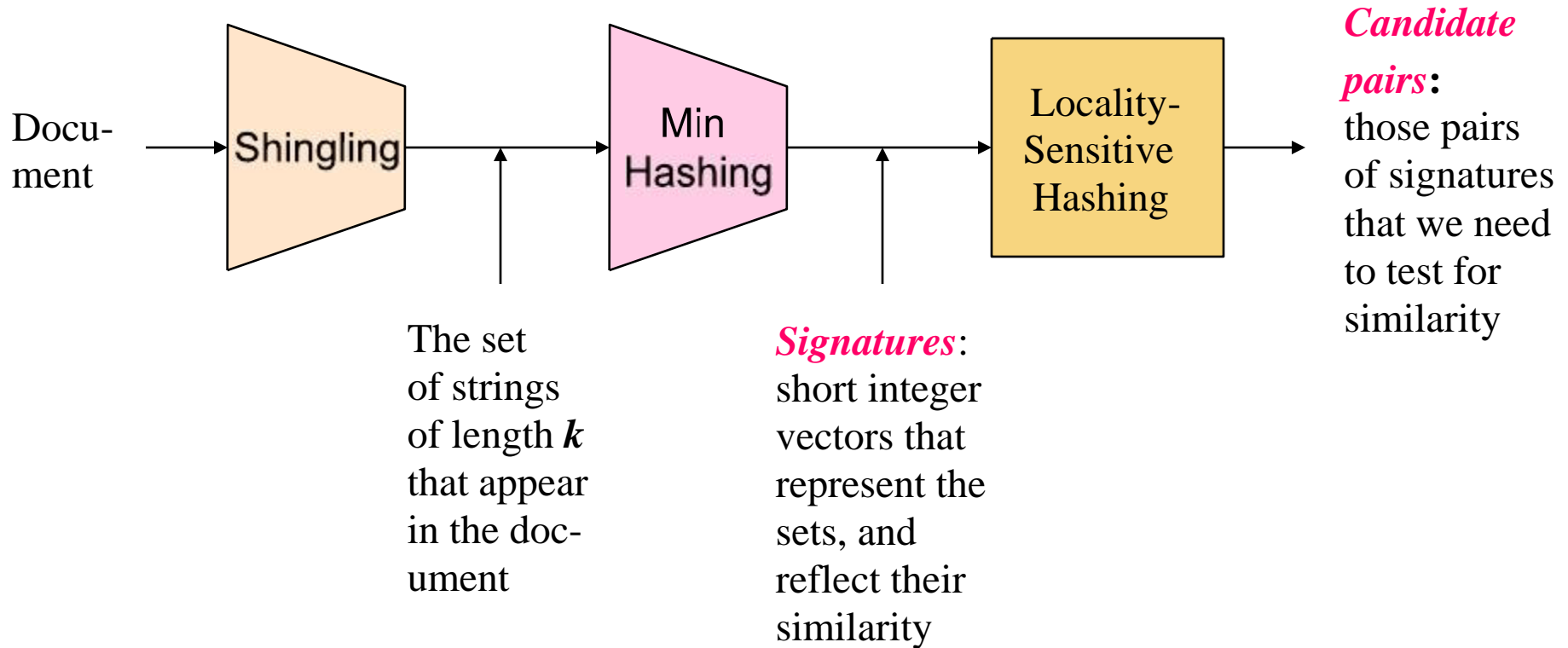


3 Essential Steps for Similar Docs

1. **Shingling**: Convert documents to sets
2. **Min-Hashing**: Convert large sets to short signatures, while preserving similarity
3. **Locality-Sensitive Hashing**: Focus on pairs of signatures likely to be from similar documents
 - **Candidate pairs!**



The Big Picture



Outline

3.0 Motivation

3.1 Finding Similar Items

3.1.1 Shingling

3.1.2 Min-Hashing

3.1.3 Locality-sensitive Hashing

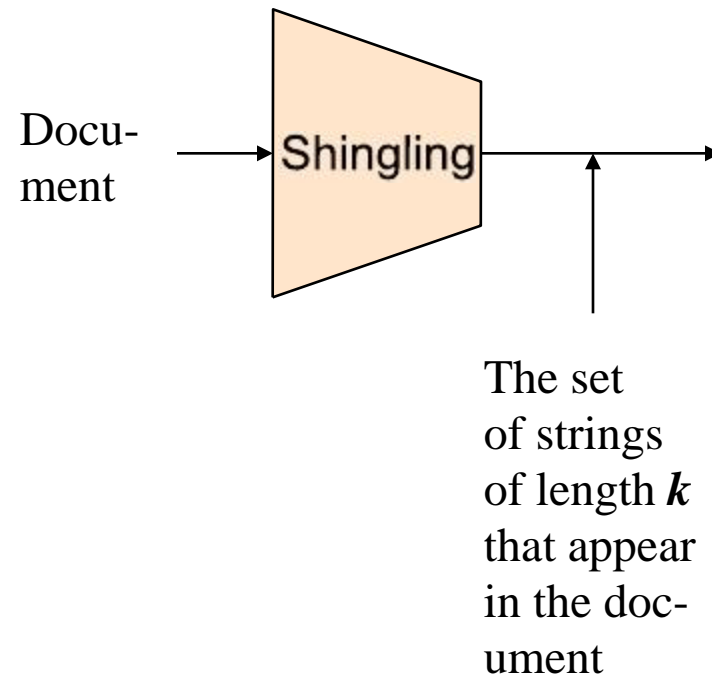
3.2 Theory of LSH

3.3 Amplifying Hash Functions: AND and OR

3.4 LSH for other distance metrics



Shingling



Step 1: **Shingling**:
Convert documents to sets



Documents as High-Dim Data

- Step 1: **Shingling**: Convert documents to sets
- **Simple approaches**:
 - Document = set of words appearing in document
 - Document = set of “important” words
 - Don't work well for this application. **Why?**
- Need to account for ordering of words!
- A different way: **Shingles!**



Define: Shingles

- 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 (ignoring back-space)
- **Example:** $k=2$; document **D1**= abcab
- Set of 2-shingles: $S(D1) = \{ab, bc, ca\}$
 - **Option:** Shingles as a bag (multiset), count ab twice: $S'(D1) = \{ab, bc, ca, ab\}$



Compressing Shingles

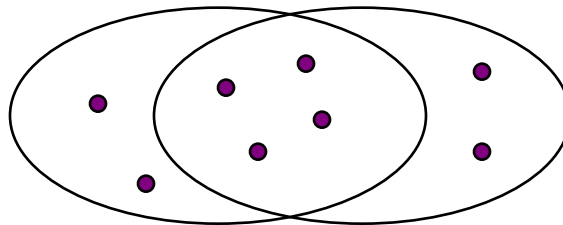
- To **compress long shingles**, we can hash them to (say) 4 bytes
- **Represent a document by the set of hash values of its k-shingles**
 - **Idea**: Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared
- **Example**: $k=2$; document $D1 = \text{abcaab}$
- Set of 2-shingles: $S(D1) = \{\text{ab}, \text{bc}, \text{ca}\}$
- Hash the singles: $h(D1) = \{1, 5, 7\}$



Similarity Metric for Shingles

- Document D1 is a set of its k-shingles $C1=S(D1)$
- Equivalently, each document is a 0/1 vector in the space of k-shingles
 - Each unique shingle is a dimension
 - Vectors are very sparse
- A natural similarity measure is the **Jaccard similarity:**

$$\text{sim}(D1, D2) = |C1 \cap C2| / |C1 \cup C2|$$



Working Assumption

- Documents that have lots of shingles in common have similar text, even if the text appears in different order
- **Careful:** You must pick k large enough, or most documents will have most shingles
 - $k = 5$ is OK for short documents (e.g. e-mails and short essays)
 - $k = 10$ is better for long documents (e.g. novels and papers)



Motivation for Minhash/LSH

- Suppose we need to find near-duplicate documents among $N = 1$ million documents
- Naïvely, we would have to compute **pairwise Jaccard similarities** for every pair of docs
 - $N(N-1)/2 \approx 5 \times 10^{11}$ 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...



Outline

3.0 Motivation

3.1 Finding Similar Items

3.1.1 Shingling

3.1.2 Min-Hashing

3.1.3 Locality-sensitive Hashing

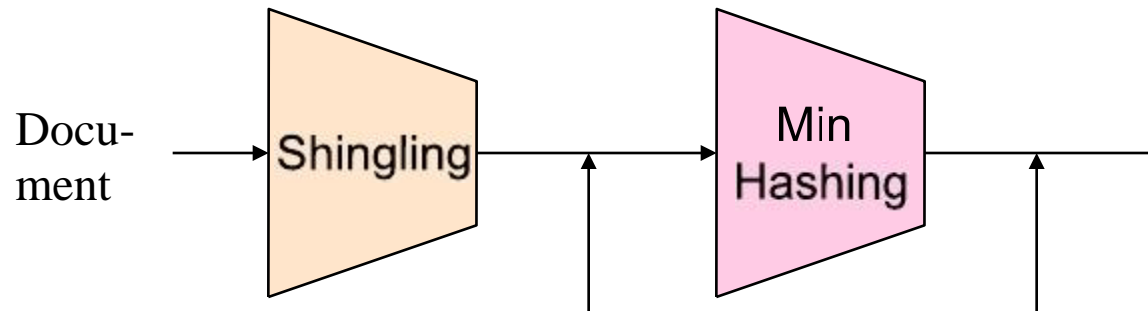
3.2 Theory of LSH

3.3 Amplifying Hash Functions: AND and OR

3.4 LSH for other distance metrics



Min Hashing



The set
of strings
of length k
that appear
in the doc-
ument

Signatures:
short integer
vectors that
represent the
sets, and
reflect their
similarity

Step 2: Min-Hashing: Convert large sets to short signatures, while preserving similarity

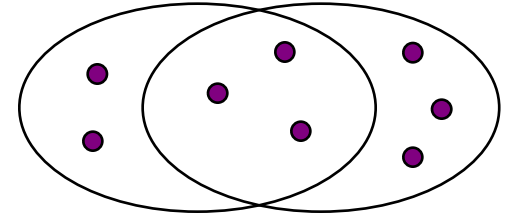


Encoding Sets as Bit Vectors

- Many similarity problems can be formalized as finding subsets that have significant intersection
- Encode sets using 0/1 (bit, boolean) vectors
 - One dimension per element in the universal set
- Interpret **set intersection** as bitwise **AND**, and **set union** as bitwise **OR**

■ **Example:** $C1 = 10111$; $C2 = 10011$

- Size of intersection = 3; size of union = 4,
- Jaccard similarity (not distance) = $3/4$
- Distance: $d(C1, C2) = 1 - (\text{Jaccard similarity}) = 1/4$



From Sets to Boolean Matrices

- **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:** $\text{sim}(C1, C2) = ?$
 - Size of intersection = 3; size of union = 6,
 - Jaccard similarity (not distance) = $3/6$
 - $d(C1, C2) = 1 - (\text{Jaccard similarity}) = 3/6$

Shingles	Documents			
	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



Outline: Finding Similar Columns

- So far:
 - Documents \rightarrow Sets of shingles
 - Represent sets as boolean vectors in a matrix
- Next goal: Find similar columns while computing small signatures
 - Similarity of columns == similarity of signatures



Outline: Finding Similar Columns

- **Next Goal:** Find similar columns, Small signatures
- **Naïve approach:**
 - 1) **Signatures of columns:** small summaries of columns
 - 2) **Examine pairs of signatures** to find similar columns
 - Essential: Similarities of signatures and columns are related
 - 3) **Optional:** Check that columns with similar signatures are really similar
- **Warnings:**
 - Comparing all pairs may take too much time: Job for LSH
 - These methods can produce **false negatives**, and even **false positives**
 - **Optional check can cancel false positives**



Hashing Columns (Signatures)

- **Key idea:** "hash" each column C to a small **signature** $h(C)$, such that:
 - (1) $h(C)$ is small enough that the signature fits in RAM
 - (2) $\text{sim}(C1, C2)$ is the same as the "similarity" of signatures $h(C1)$ and $h(C2)$
- **Goal:** Find a hash function $h(\cdot)$ such that:
 - If $\text{sim}(C1, C2)$ is high, then with high prob. $h(C1) = h(C2)$
 - If $\text{sim}(C1, C2)$ is low, then with high prob. $h(C1) \neq h(C2)$
- Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!



Min-Hashing

- **Goal:** Find a hash function $h(\cdot)$ such that:
 - if $\text{sim}(C1, C2)$ is high, then with high prob. $h(C1) = h(C2)$
 - if $\text{sim}(C1, C2)$ is low, then with high prob. $h(C1) \neq h(C2)$
- Clearly, the hash function depends on the similarity metric:
 - Not all similarity metrics have a suitable hash function
- There is a suitable hash function for the Jaccard similarity: It is called **Min-Hashing**



Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random permutation** π
- Define a "**hash**" function $h_{\pi}(C)$ = the index of the first (in the permuted order π) row in which column C has value 1:

$$h_{\pi}(C) = \min \pi(C)$$

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column



Min-Hashing Example (one element)

$$h_{\pi}(C) = \min \pi(C)$$

$$h_{\pi_1}(C_1) =$$

2
3
7
6
1
5
4

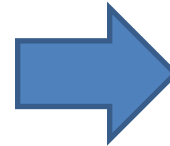
π_1

*

.

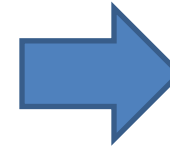
1
1
0
0
0
1
1

C_1



2
3
0
0
0
5
4

Min Item

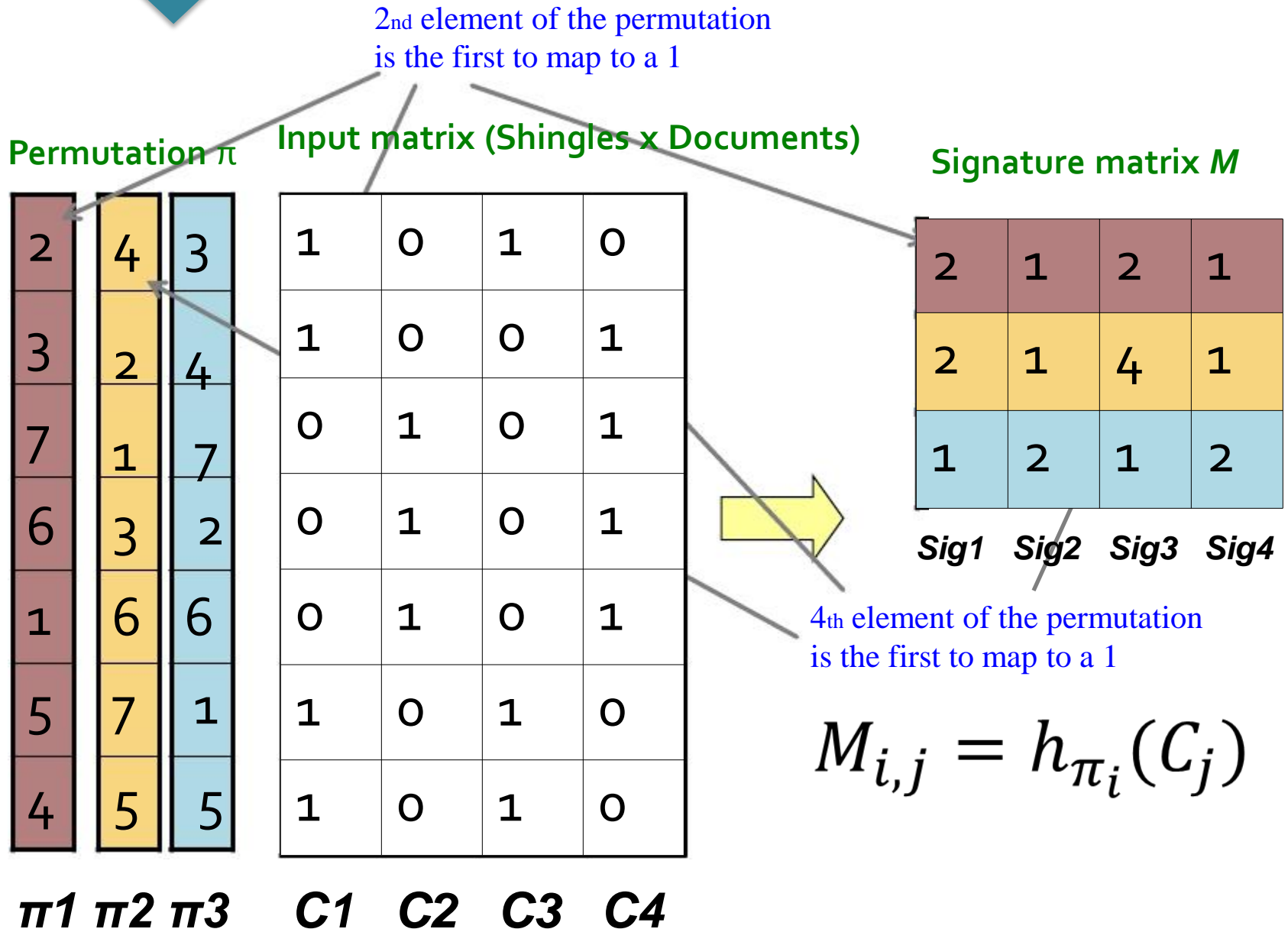


Ignore '0'

2



Min-Hashing Example



The Min-Hash Property

- Choose a random permutation π
- Claim: $\Pr[h\pi(C1) = h\pi(C2)] = \text{sim}(C1, C2)$
- Why?
 - Let X be a doc (set of shingles), $y \in X$ is a shingle
 - Then: $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
 - It is equally likely that any $y \in X$ is mapped to the min element
- Let y be s.t. $\pi(y) = \min(\pi(C1 \cup C2))$
- Then either: $\pi(y) = \min(\pi(C1))$ if $y \in C1$, or $\pi(y) = \min(\pi(C2))$ if $y \in C2$

One of the two cols had to have 1 at position y
- So the prob. that both are true is the prob. $y \in C1 \cap C2$
- $\Pr[\min(\pi(C1)) = \min(\pi(C2))]$
 $= |C1 \cap C2| / |C1 \cup C2| = \text{sim}(C1, C2)$



Similarity for Signatures

- We know: $\Pr[h_{\pi}(C1) = h_{\pi}(C2)] = \text{sim}(C1, C2)$
- Now generalize to multiple hash functions
- The **similarity of two signatures** is the fraction of the hash functions in which they agree
- **Note:** Because of the Min-Hash property, **the similarity of columns** is the same as the **expected similarity of their signatures**

Hence, $\text{sim}(C1, C2) = \text{sim}(\text{Sig1}, \text{Sig2})$ when the number of permutation function is large enough (e.g. 100)



Min-Hashing Example

Permutation ☐

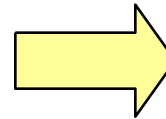
2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

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

We achieved our goal! We “compressed” long bit vectors into short signatures



Outline

3.0 Motivation

3.1 Finding Similar Items

3.1.1 Shingling

3.1.2 Min-Hashing

3.1.3 Locality-sensitive Hashing

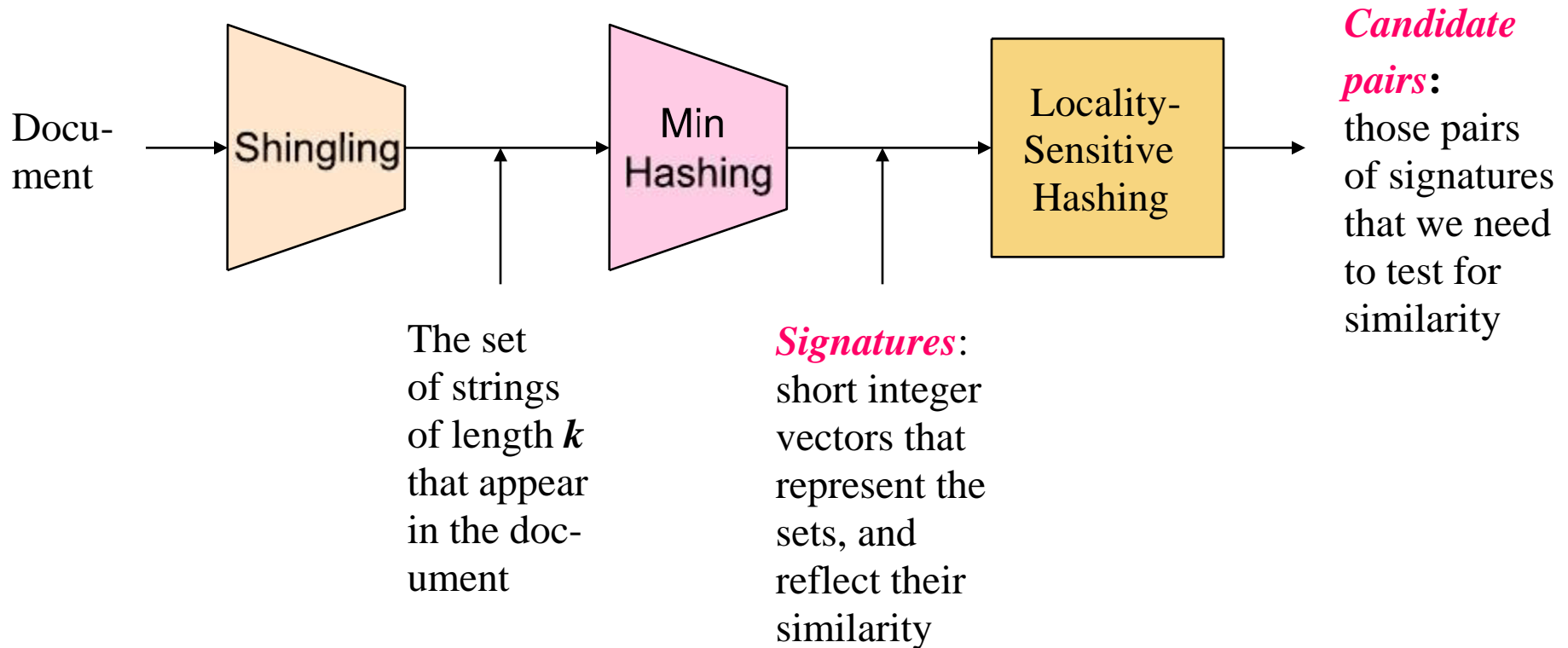
3.2 Theory of LSH

3.3 Amplifying Hash Functions: AND and OR

3.4 LSH for other distance metrics



Locality Sensitive Hashing



Step 3: Locality-Sensitive Hashing:

Focus on pairs of signatures likely to be from similar documents



LSH: First Cut

- Goal: Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s=0.8$)
- LSH - General idea: Use a function $f(x,y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated
- For Min-Hash matrices:
 - Hash columns of signature matrix M to many buckets
 - Each pair of documents that hashes into the same bucket is a candidate pair



Candidates from Min-Hash

- Pick a similarity threshold s ($0 < s < 1$)
- Columns x and y of M are a candidate pair if their signatures agree on at least fraction s of their rows:
- $M(i, x) = M(i, y)$ for at least frac. s values of i
 - We expect documents x and y to have the same (Jaccard) similarity as their signatures

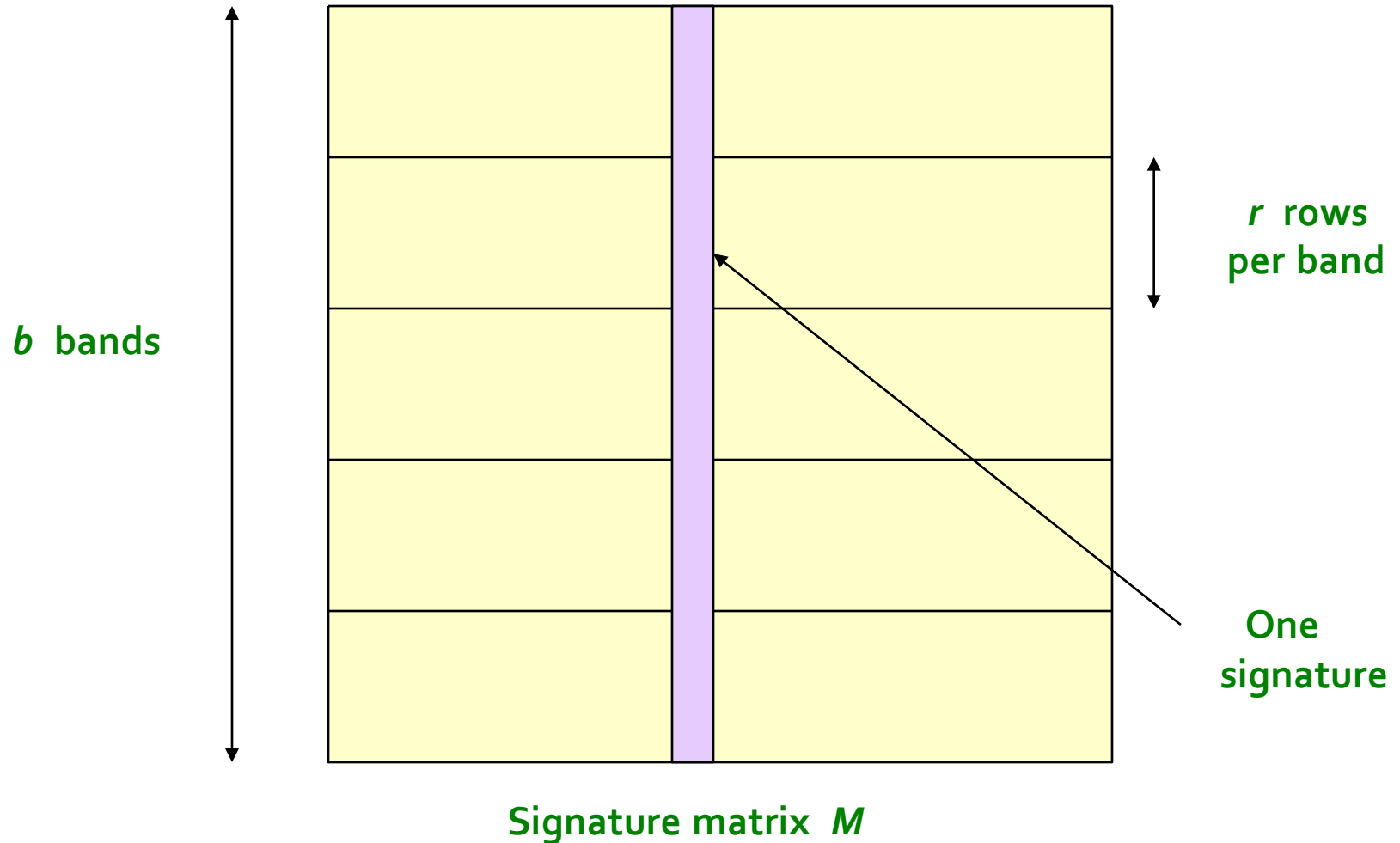


LSH for Min-Hash

- **Big idea:** Hash columns of signature matrix M several times instead of computing all couples of columns
- Arrange that (only) **similar columns** are likely to **hash to the same bucket**, with high probability
- **Candidate pairs** are those that hash to the same bucket



Partition M into b Bands

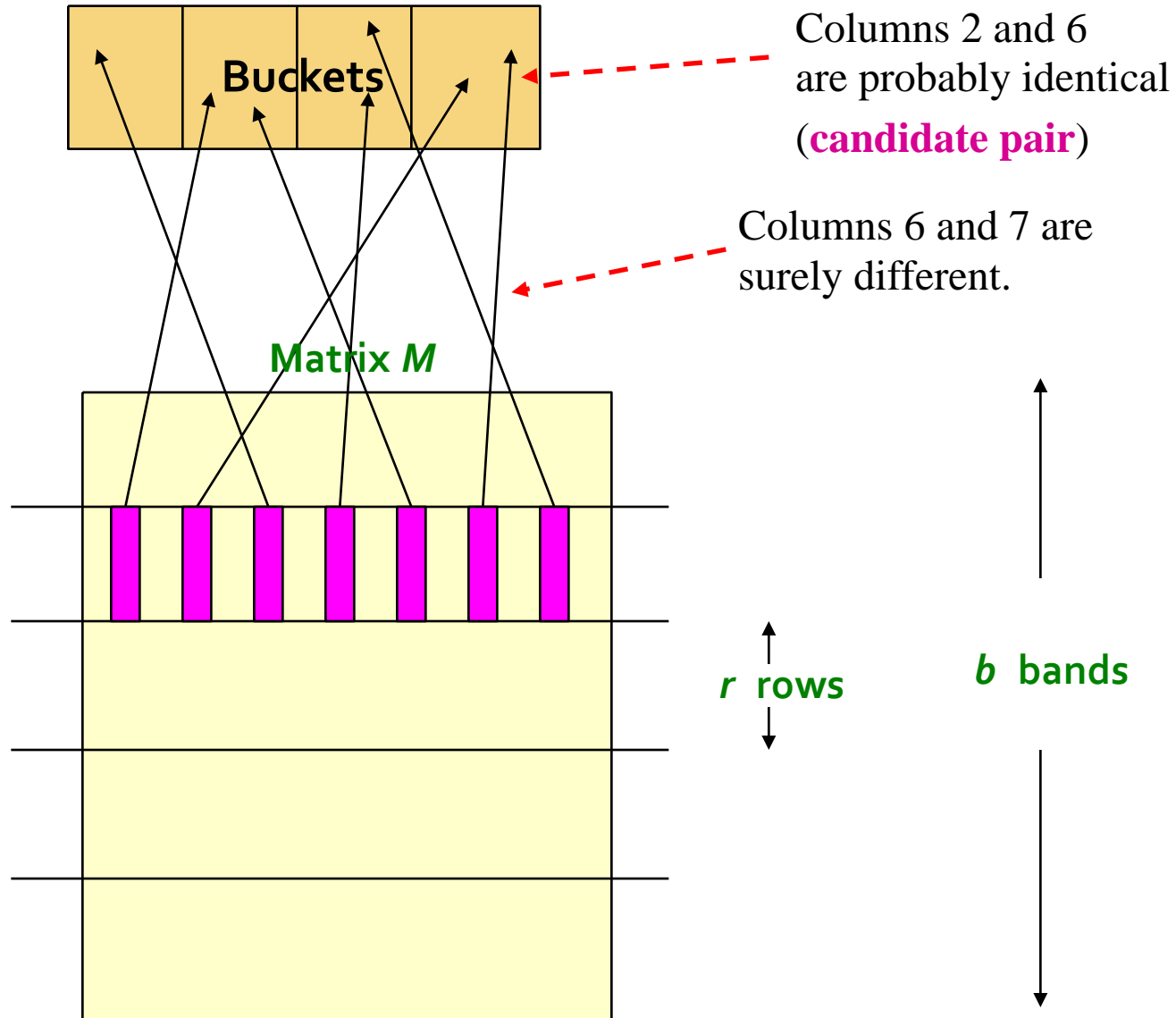


Partition M into Bands

- Divide matrix **M** into **b** bands of **r** rows
- For each band, hash its portion of each column to a hash table with **k** buckets
 - Make **k** as large as possible (at least $k \geq \text{number of documents}$)
- **Candidate** column pairs are those that hash to the same bucket for ≥ 1 band
- Tune **b** and **r** to catch most similar pairs, but few non-similar pairs



Hashing Bands



Simplifying Assumption

- There are **enough buckets** that columns are unlikely to hash to the same bucket unless they are **identical** in a particular band
- Hereafter, we assume that "*same bucket*" means "*identical in that band*"
- Assumption needed only to simplify analysis, not for correctness of algorithm



C1,C2 are 80% Similar

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$ and $r = 5$
- Assume: $\text{sim}(C1, C2) = 0.8$
 - Since $\text{Sim}(C1, C2) \geq s$, we want $C1, C2$ to be **candidate pair**: We want them to hash to at least 1 common bucket (at least one band is identical)
- Probability $C1, C2$ identical in one particular band:
 $(0.8)^5 = 0.328$
- Probability $C1, C2$ are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
 - 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



C1,C2 are 30% Similar

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$ and $r = 5$
- Assume: $\text{sim}(C1, C2) = 0.3$
 - Since $\text{Sim}(C1, C2) < s$, we want $C1, C2$ to be **No candidate pair** (all bands should be different)
- Probability $C1, C2$ identical in one particular band: $(0.3)^5 = 0.00243$
- Probability $C1, C2$ identical in at least 1 of 20 bands: $1 - (1 - 0.00243)^{20} = 0.00474$
 - In other words, approximately 0.474% 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

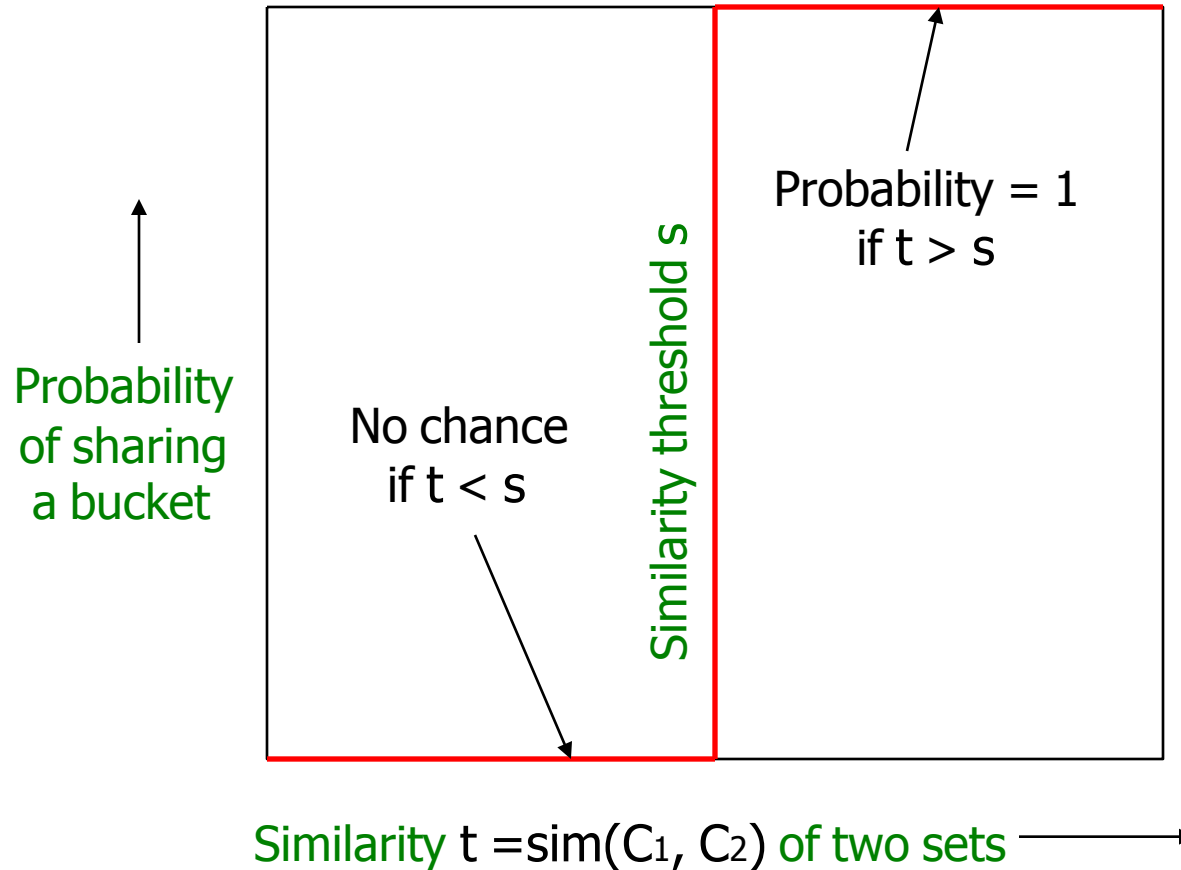


b Bands, r rows/band

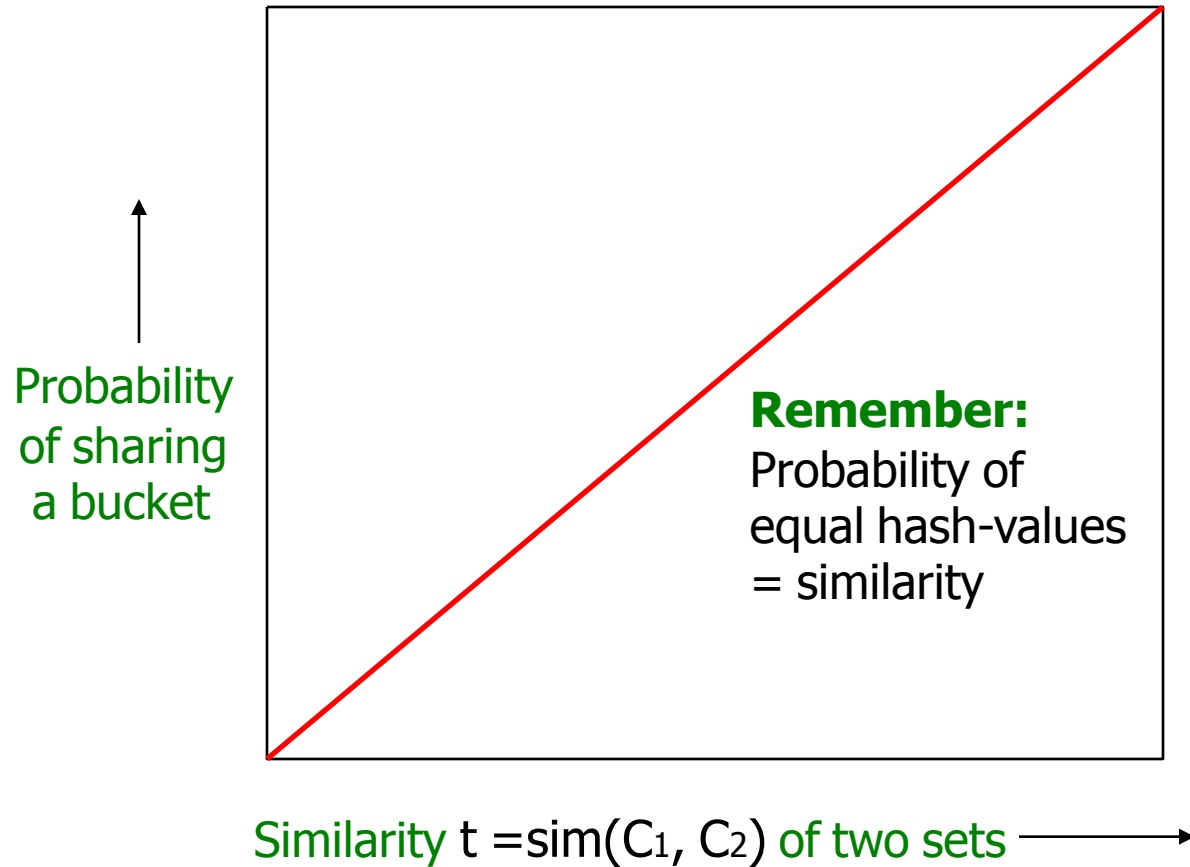
- Columns C_1 and C_2 have similarity t
- Pick any band (r rows)
 - Prob. that all rows in band equal = t^r
 - Prob. that some row in band unequal = $1 - t^r$
- Prob. that no band identical = $(1 - t^r)^b$
- Prob. that at least 1 band identical =
 $1 - (1 - t^r)^b$



Analysis of LSH – What We Want

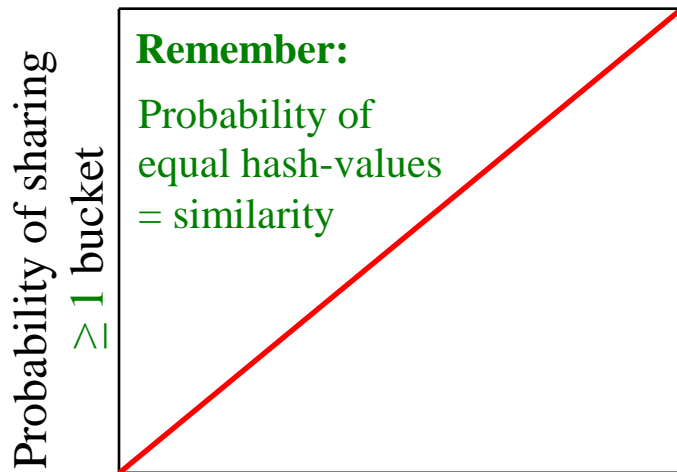


What 1 band of 1 Row Gives You



Recap: The S-Curve

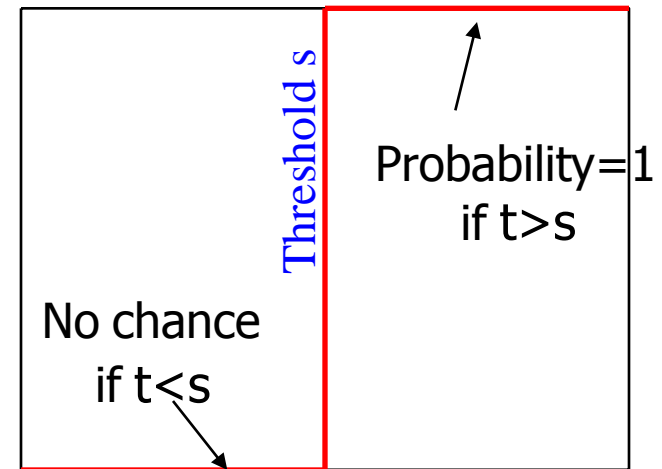
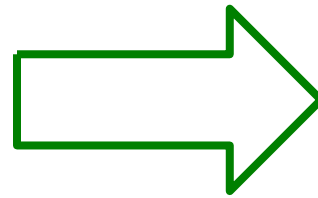
- The S-curve is where the “magic” happens



Similarity t of two sets

This is what 1 hash-code gives you

$$\Pr[h(C_1) = h(C_2)] = \text{sim}(D_1, D_2)$$



Similarity t of two sets

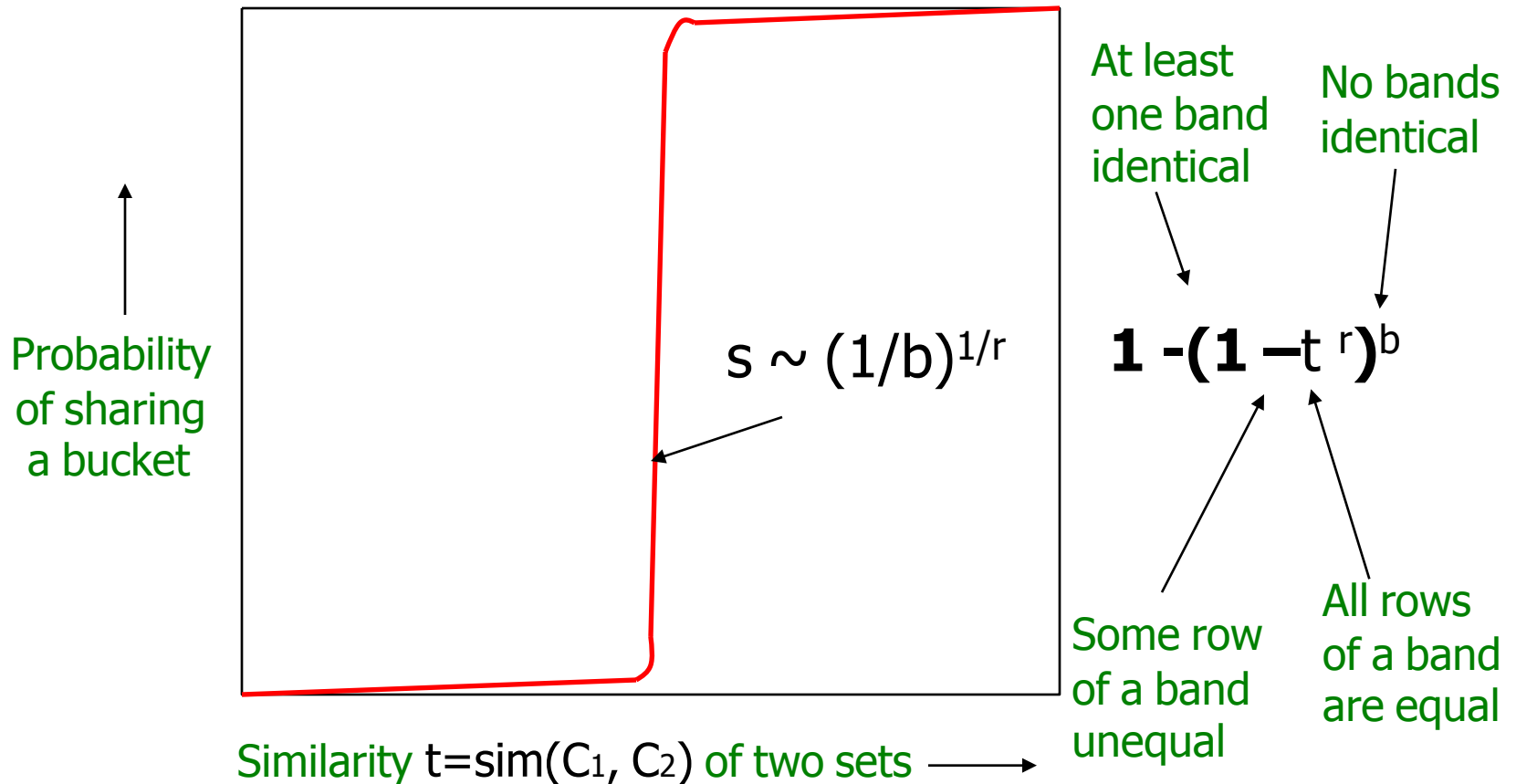
This is what we want!

How to get a step-function?

By choosing r and b !

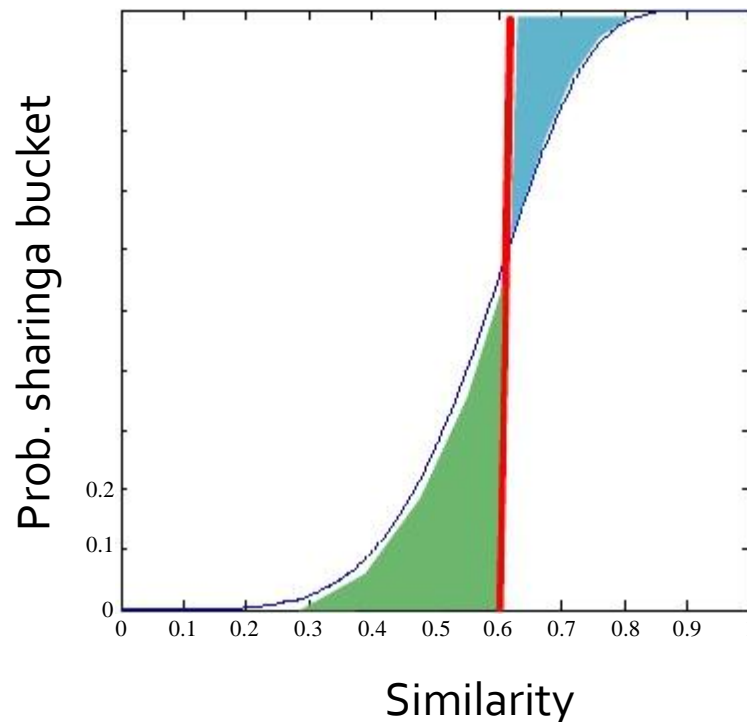


What b Bands of r Rows Gives You



Picking r and b: The S-curve

- Picking r and b to get the best S-curve
 - 50 hash-functions ($r=5$, $b=10$)



Blue area: False Negative rate
Green area: False Positiverate



Optional Check

- Because of false positive/negative existing, we need the **optional check** work to solve the false positive
- Check in main memory that **candidate pairs** really do have similar signatures
- **Optional:** In another pass through data, check that the remaining candidate pairs really represent similar documents



Summary: 3Steps

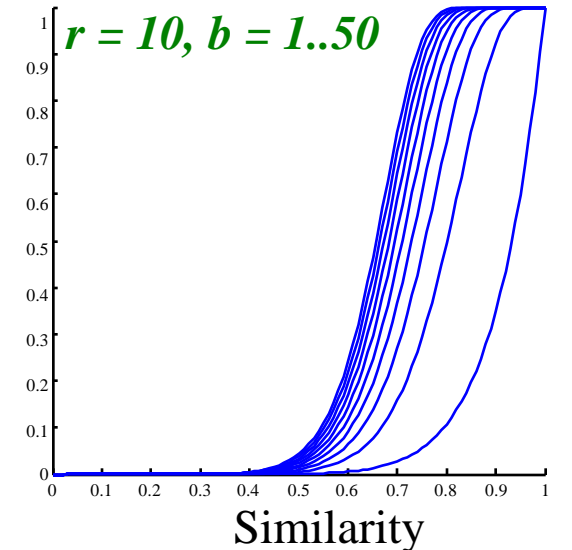
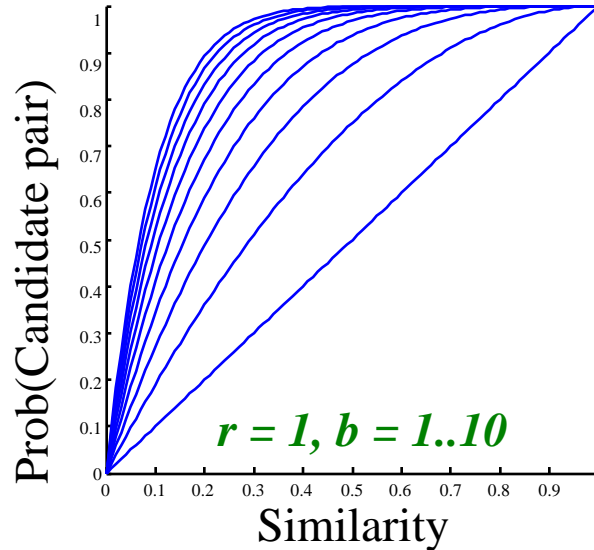
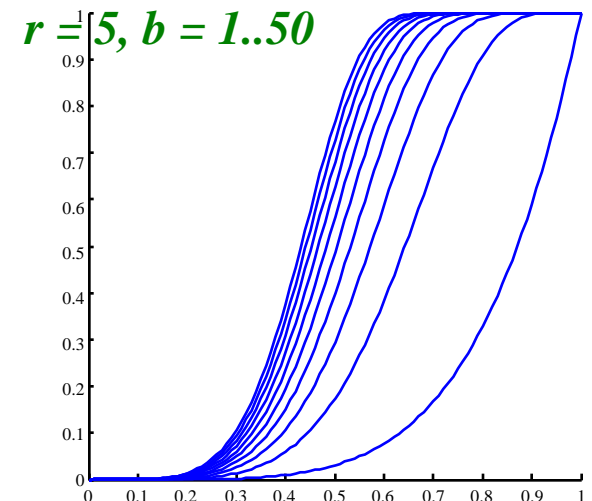
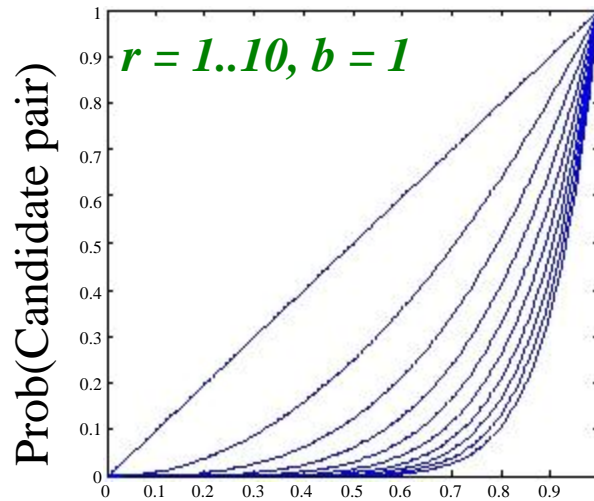
- **Shingling**: Convert documents to sets
 - We used hashing to assign each shingle an ID
- **Min-Hashing**: Convert large sets to short signatures, while preserving similarity
 - We used similarity preserving hashing to generate signatures with property $\Pr[h_{\pi}(C1) = h_{\pi}(C2)] = \text{sim}(C1, C2)$
 - 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 $\geq s$



S-curves as a func. of b and r

Given a fixed threshold s .

We want choose r and b such that the $P(\text{Candidate pair})$ has a “step” right around s .



$$P(C1, C2 \text{ is a candidate pair}) = 1 - (1 - t^r)^b$$



Outline

3.0 Motivation

3.1 Finding Similar Items

3.1.1 Shingling

3.1.2 Min-Hashing

3.1.3 Locality-sensitive Hashing

3.2 Theory of LSH

3.3 Amplifying Hash Functions: AND and OR

3.4 LSH for other distance metrics



Families of Hash Functions

- For Min-Hashing signatures, we got a Min-Hash function for each permutation of rows
- A “hash function” is any function that takes two elements and says whether they are “equal”
 - **Example:** $h(x)=h(y)$ means “h says x and y are equal”
- A **family** of hash functions is any set of hash functions from which we can pick one at random efficiently
 - **Example:** The set of Min-Hash functions generated from permutations of rows



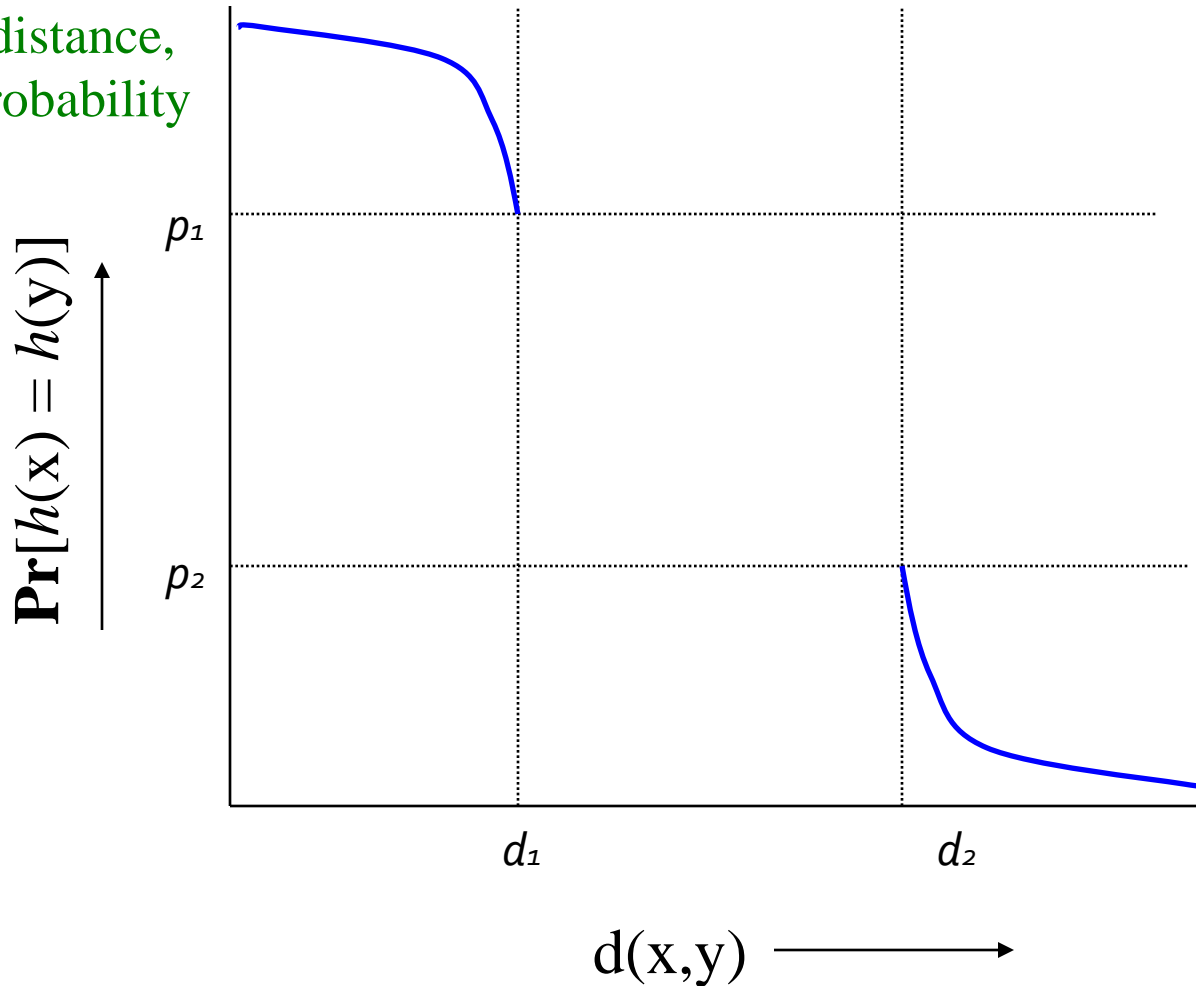
Locality – Sensitive (LS) Families

- Suppose we have a space S of points with a distance measure $d(x, y)$
- A family H of hash functions is said to be $(d1, d2, p1, p2)$ -sensitive if for any x and y in S :
 - 1. If $d(x, y) < d1$, then the probability over all $h \in H$, that $h(x) = h(y)$ is at least $p1$
 - 2. If $d(x, y) > d2$, then the probability over all $h \in H$, that $h(x) = h(y)$ is at most $p2$



A $(d1, d2, p1, p2)$ -sensitive function

Small distance,
high probability



Large distance,
low probability
of hashing to
the same value



LS Family of Min-Hash

- For any hash function $h \in \mathcal{H}$:

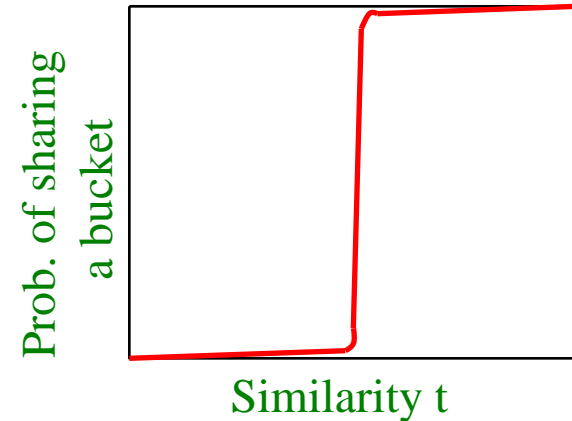
$$\Pr[h(x) = h(y)] = 1 - d(x, y)$$

- Simply restates theorem about Min-Hashing in terms of distances rather than similarities
- For Jaccard similarity, Min-Hashing gives a $(d_1, d_2, (1-d_1), (1-d_2))$ -sensitive family for any $d_1 < d_2$
- Theory leaves unknown what happens to pairs that are at distance between d_1 and d_2
 - **Consequence:** No guarantees about fraction of false positives in that range



Amplifying a LS-Family

- Can we reproduce the "S-curve" effect we saw before for any LS family?



- The "bands" technique we learned for signature matrices carries over to this more general setting
 - So we can do LSH with any $(d1, d2, p1, p2)$ -sensitive family
- Two constructions:
 - AND construction like "rows in a band"
 - OR construction like "many bands"



Outline

3.0 Motivation

3.1 Finding Similar Items

3.1.1 Shingling

3.1.2 Min-Hashing

3.1.3 Locality-sensitive Hashing

3.2 Theory of LSH

3.3 Amplifying Hash Functions: AND and OR

3.4 LSH for other distance metrics



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 , $1 \leq i \leq r$
 - 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**



Subtlety Regarding Independence

- **Independence** of hash functions (HFs) really means that the prob. of two HFs saying "yes" is the product of each saying "yes"
 - But two hash functions could be highly correlated
 - For example, in Min-Hash if their permutations agree in the first one million entries
 - However, the probabilities in definition of a LSH-family are over all possible members of H, H'



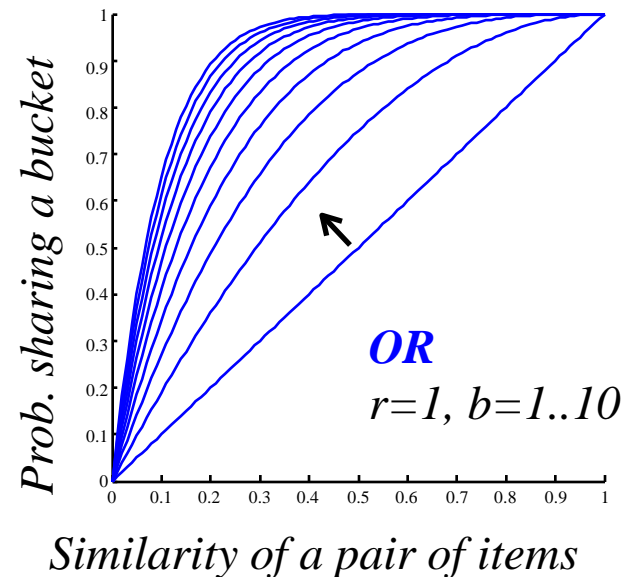
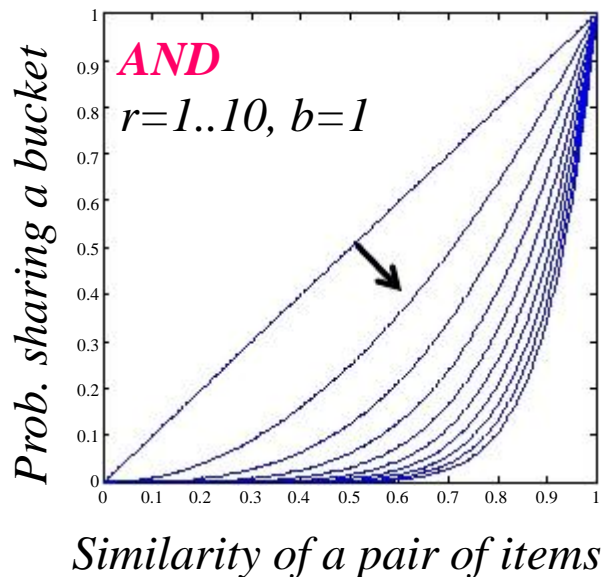
OR of Hash Functions

- Given family H , construct family H' consisting of b functions from H
- For $h = [h_1, \dots, h_b]$ in H' , we say $h(x) = h(y)$ 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



Effect of AND and OR Constructions

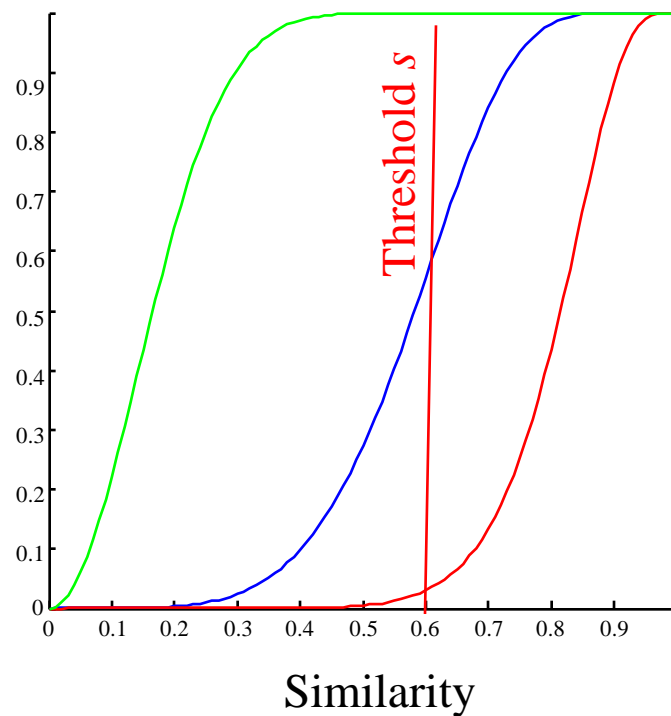
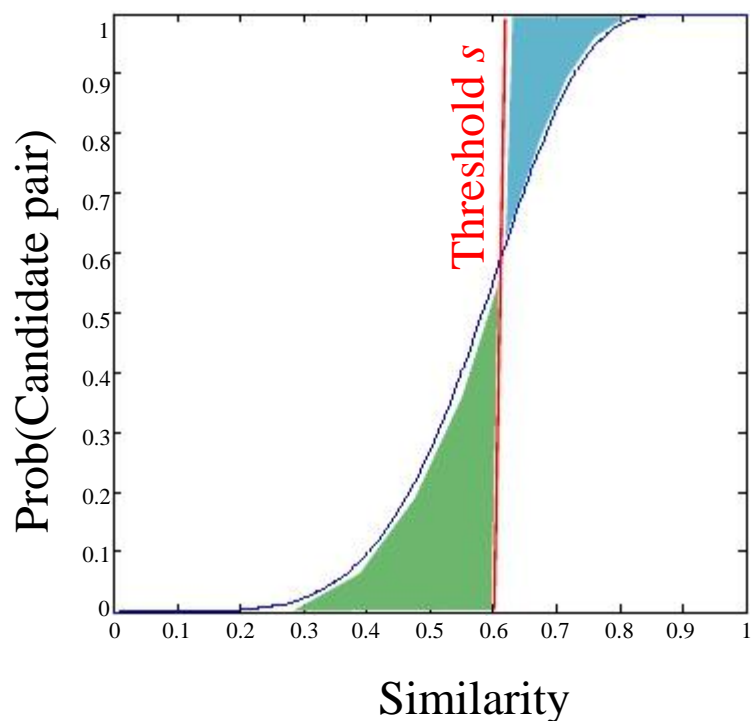
- **AND** makes all probs. shrink, but by choosing r correctly, we can make the lower prob. approach 0 while the higher does not
- **OR** makes all probs. grow, but by choosing b correctly, we can make the upper prob. approach 1 while the lower does not



Picking r and b: the S-curve

■ Picking r and b to get desired performance

- 1. 50 hash-functions ($r = 5, b = 10$)
- 2. 50 hash-functions ($r * b = 50$)



$r=2, b=25$
 $r=5, b=10$
 $r=10, b=5$



Composing Constructions

- Exactly what we did with Min-Hash
 - If bands match in all r values hash to same bucket
 - Cols that are hashed into ≥ 1 common bucket \rightarrow Candidate
- Take points x and y s.t. $\Pr[h(x) = h(y)] = p$
 - H will make (x,y) a candidate pair with prob. P
- Construction makes (x,y) a candidate pair with probability $1-(1-p^r)^b$
 - 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 $b = 4$



AND-OR Composing Constructions

- Each of the two probabilities p is transformed into $1-(1-p^r)^b$.
 - The "S-curve" studied before.
- **Example:** Take H and construct H' by the AND construction with $r = 4$. Then, from H' , construct H'' by the OR construction with $b = 4$.



Table for Function $1-(1-p^4)^4$

p	$1-(1-p^4)^4$
.2	.0064
.3	.0320
.4	.0985
.5	.2275
.6	.4260
.7	.6666
.8	.8785
.9	.9860

Example: Transforms a $(.2,.8,.8,.2)$ -sensitive family into a $(.2,.8,.8785,.0064)$ -sensitive family.



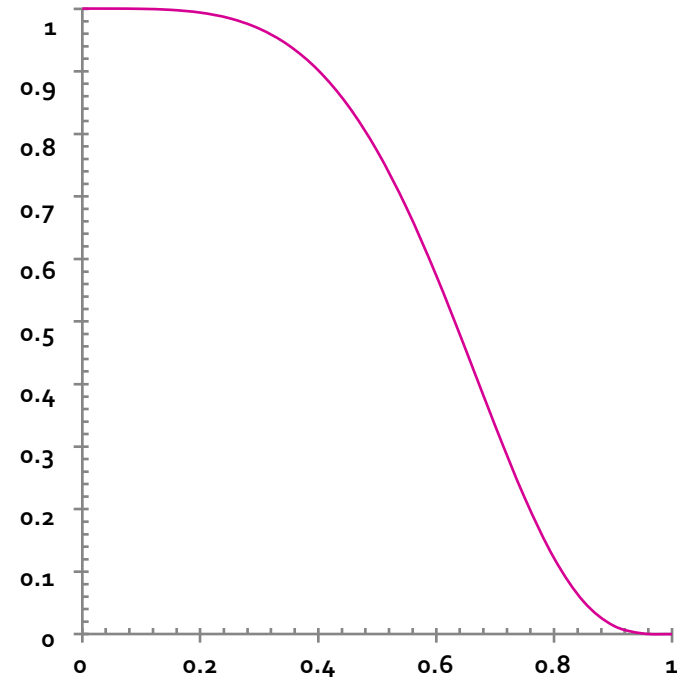
OR-AND Composing Constructions

- Each of the two probabilities p is transformed into $(1-(1-p)^b)^r$.
 - The same S-curve, mirrored horizontally and vertically.
- **Example:** Take H and construct H' by the OR construction with $b = 4$. Then, from H' , construct H'' by the AND construction with $r = 4$.



Table for Function $(1-(1-p^4))^4$

p	$1-(1-p^4)^4$
.1	.0140
.2	.1215
.3	.3334
.4	.5740
.5	.7725
.6	.9015
.7	.9680
.8	.9936



The example transforms a
(.2,.8,.8,.2)-sensitive family into a
(.2,.8,.9936,.1215)-sensitive family



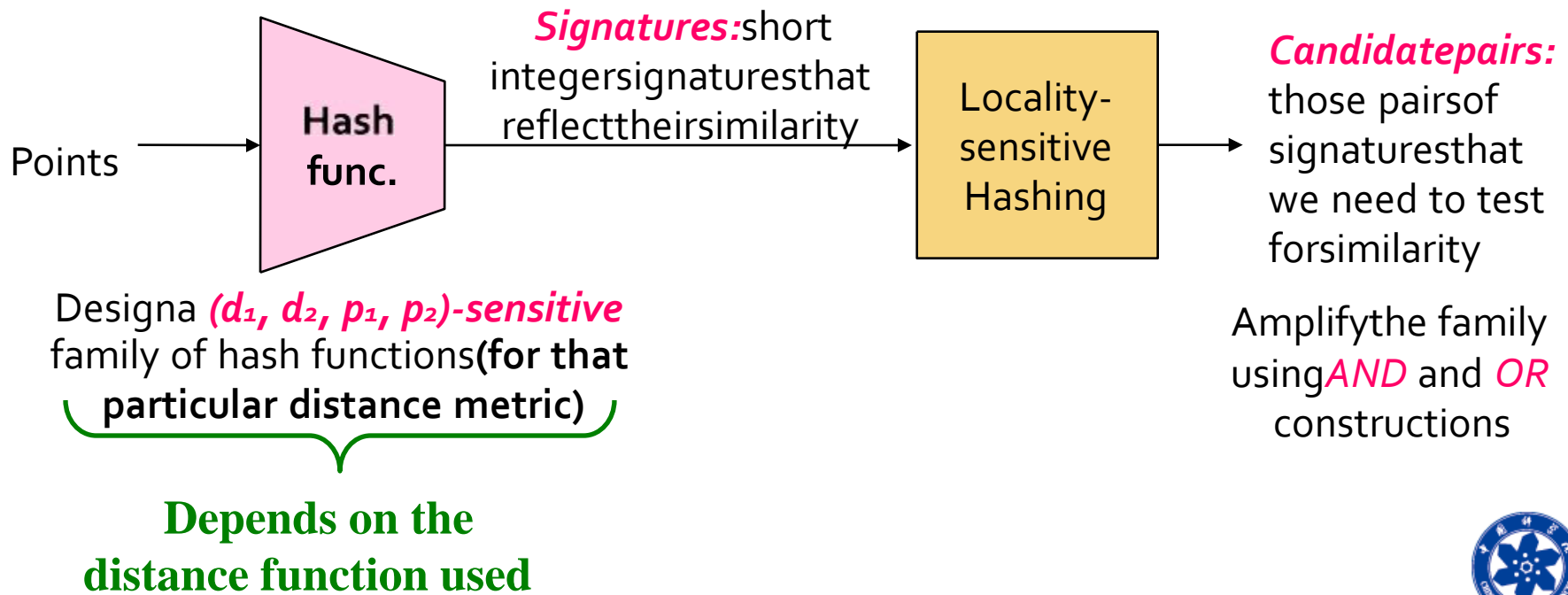
Cascading Constructions

- Transforms a $(.2, .8, .8, .2)$ -sensitive family into a $(.2, .8, .9936, .1215)$ -sensitive family
 - Note this family uses 256 ($=4*4*4*4$) of the original hash functions
 - Note by using 256 hash functions, the $(.2, .8, .8, .2)$ -sensitive family was transformed into a $(.2, .8, .9999996, .0008715)$ -sensitive family
- The closer to 0 and 1 we get, the more hash functions must be used!



LSH for other Distance Metrics

- **Problem:** More generally, we found similar columns in large sparse matrices with high Jaccard similarity
- **Can we use LSH for other distance measures?**
 - E.g., cosine distance: Random hyperplanes
 - E.g., euclidean distance: Project on lines



Outline

3.0 Motivation

3.1 Finding Similar Items

3.1.1 Shingling

3.1.2 Min-Hashing

3.1.3 Locality-sensitive Hashing

3.2 Theory of LSH

3.3 Amplifying Hash Functions: AND and OR

3.4 LSH for other distance metrics



Distance Measures

- Generalized LSH is based on some kind of “distance” between points.
 - Similar points are “close.”
 - Jaccard similarity is not a distance; 1 minus Jaccard similarity is.
- Two major classes of distance measure:
 1. Euclidean
 2. Non-Euclidean



Euclidean VS. Non- Euclidean

- A *Euclidean space* has some number of real valued dimensions and “dense” points.
 - There is a notion of “average” of two points.
 - A *Euclidean distance* is based on the locations of points in such a space.
- Any other space is *Non-Euclidean*.
 - Distance measures for non-Euclidean spaces are based on properties of points, but not their “location” in a space.



Axioms of a Distance Measure

- d is a *distance measure* if it is a function from pairs of points to real numbers such that:
 1. $d(x,y) \geq 0$.
 2. $d(x,y) = 0$ iff $x = y$.
 3. $d(x,y) = d(y,x)$.
 4. $d(x,y) \leq d(x,z) + d(z,y)$ (*triangle inequality*).



Some 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 the differences in each dimension.
 - *Manhattan distance* = distance if you had to travel along coordinates only.



Some Euclidean Distances

- L_∞ *norm*: $d(x,y)$ = the maximum of the differences between x and y in any dimension.
- **Note**: the maximum is the limit as r goes to ∞ of the L_r *norm*: what you get by taking the r^{th} power of the differences, summing and taking the r^{th} root.



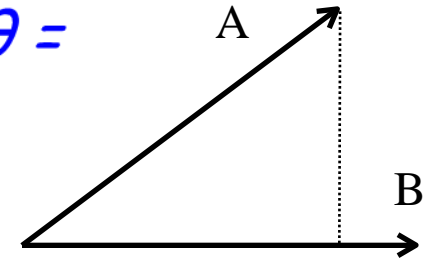
Axioms of a Distance Measure

- *Jaccard distance* for sets = $1 - \text{Jaccard similarity}$.
- *Cosine distance* for vectors = angle between the vectors.
- *Edit distance* for strings = number of inserts and deletes to change one string into another.
- *Hamming Distance* for bit vectors = the number of positions in which they differ.



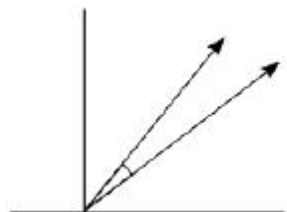
Cosine Distance

- **Cosine distance** = angle between vectors from the origin to the points in question $d(A, B) = \theta = \arccos(A \cdot B / \|A\| \|B\|)$

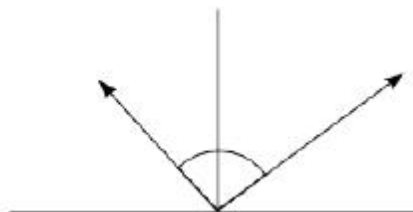


- Has range $0 \dots \pi$ (equivalently $0 \dots 180^\circ$)
 - Can divide θ by π to have distance in range $0 \dots 1$
- Cosine similarity = $1 - d(A, B)$

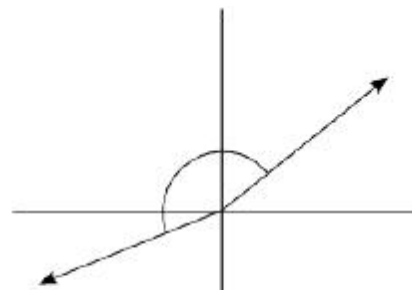
- But often defined as cosine sim: $\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$



Similar scores
Score Vectors in same direction
Angle between them is near 0 deg.
Cosine of angle is near 1 i.e. 100%



Unrelated scores
Score Vectors are nearly orthogonal
Angle between them is near 90 deg.
Cosine of angle is near 0 i.e. 0%



Opposite scores
Score Vectors in opposite direction
Angle between them is near 180 deg
Cosine of angle is near -1 i.e. -100%

- Has range $-1 \dots 1$ for general vectors
- Range $0 \dots 1$ for non-negative vectors (angles up to 90°)

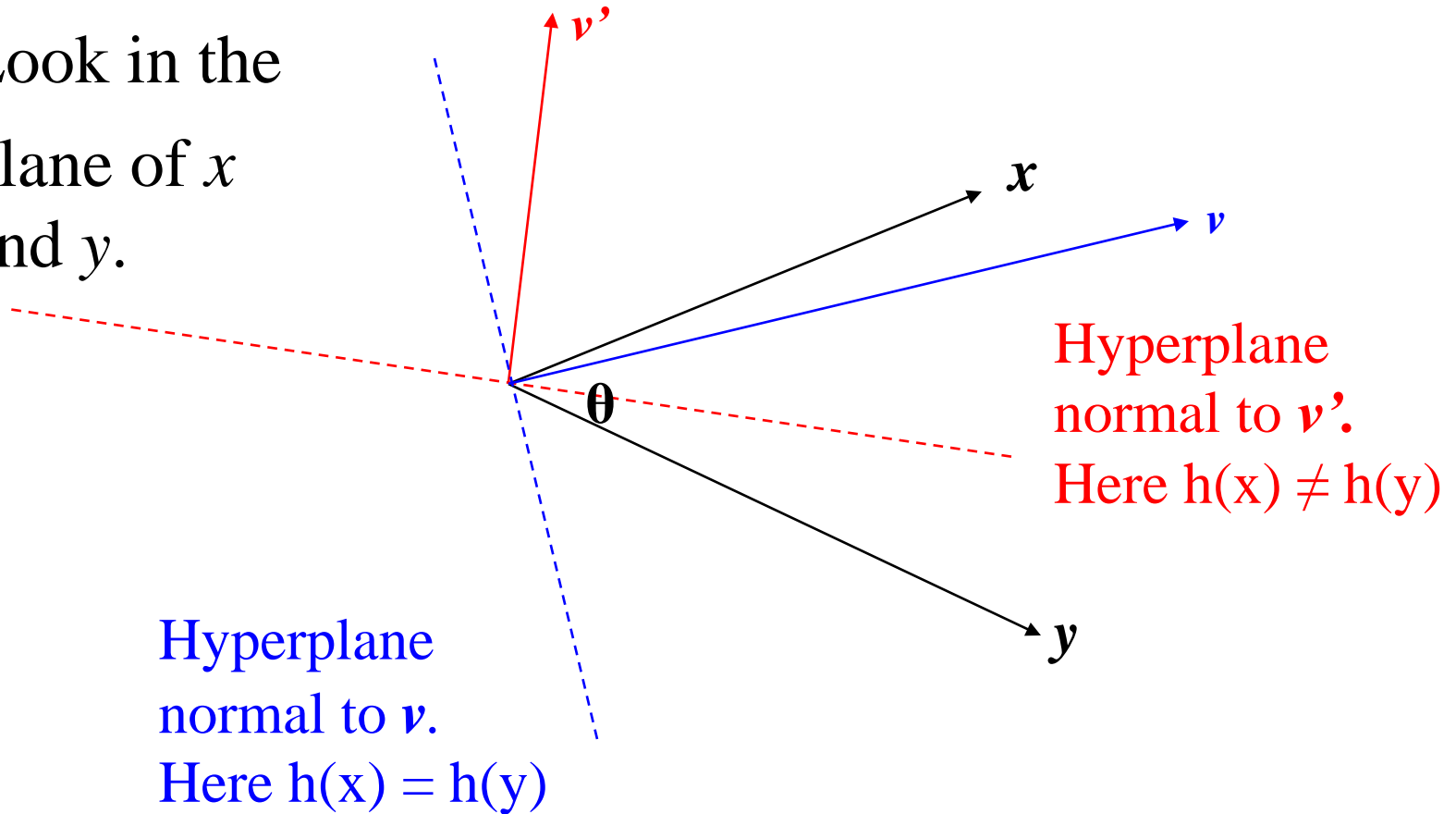
LSH for Cosine Distance

- For cosine distance, there is a technique called **Random Hyperplanes**
 - Technique similar to Min-Hashing
- Random Hyperplanes method is a $(d_1, d_2, (1-d_1/180), (1-d_2/180))$ -sensitive family for any d_1 and d_2
- Pick a random vector v , which determines a hash function h_v with two buckets
- $h_v(x) = +1$ if $v \cdot x \geq 0$; $= -1$ if $v \cdot x < 0$
- LS-family H = set of all functions derived from any vector
- **Claim:** For points x and y ,
$$\Pr[h_v(x) = h_v(y)] = 1 - d(x, y) / 180 \pi$$



Proof of Claim

Look in the
plane of x
and y .



Signatures for Cosine Distance

Random vector

+1	-1	0
-1	-1	0
0	0	0
+1	-1	-1
+1	0	+1
-1	0	+1
-1	0	-1

v1 v2 v3

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

C1 C2 C3 C4

Signature matrix *M*

-1	+1	-1	+1
-1	-1	-1	-1
0	0	0	0

Sig1 Sig2 Sig3 Sig4

$$M_{i,j} = h_{v_i}(C_j)$$

- LSH of cosine distance likes we used the Min-Hash signatures for Jaccard distance
- Amplify using **AND/OR** constructions(*b* bands and *r* rows)

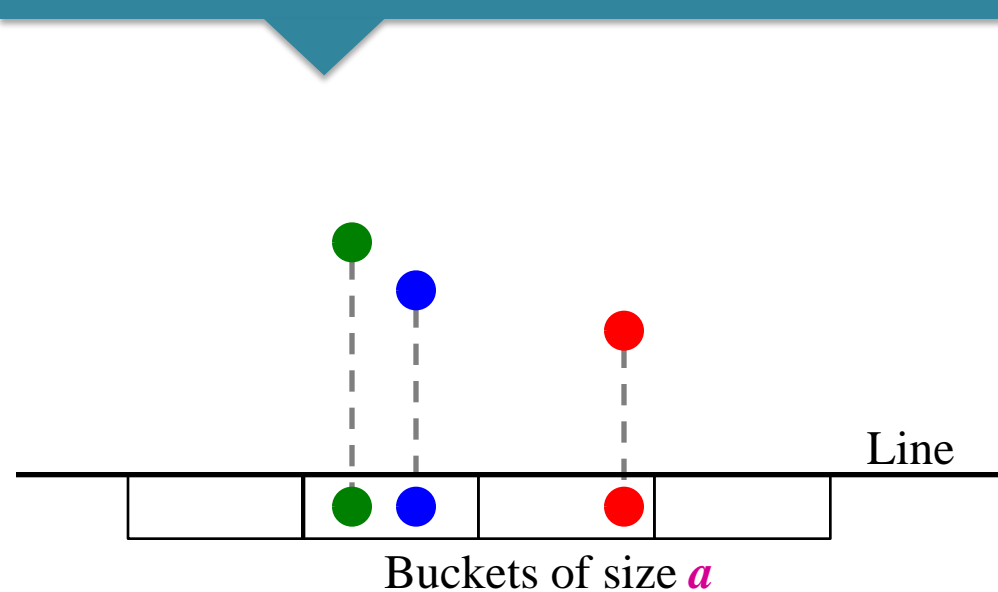


LSH for Euclidean Distance

- **Simple 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
- **Nearby points are always close;**
distant points are rarely in same bucket

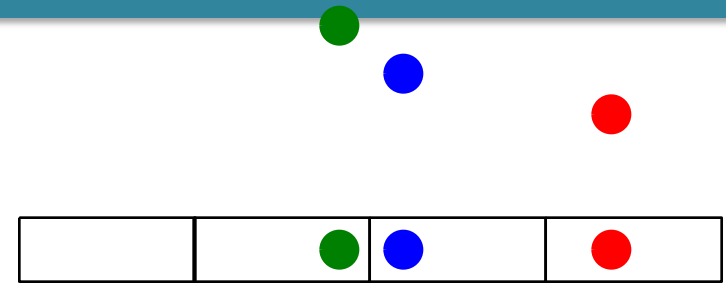


Projection of Points



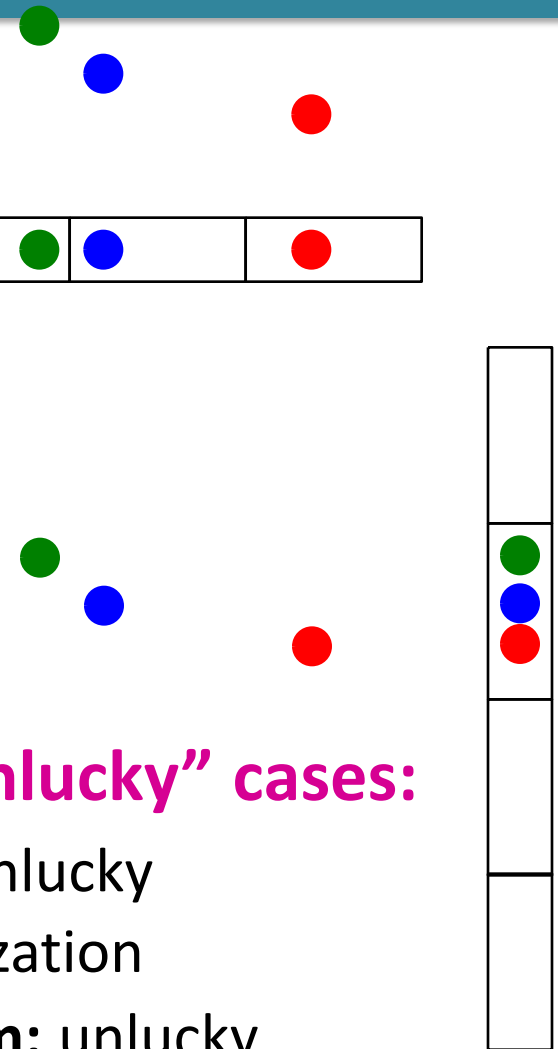
■ “Lucky” case:

- Points that are close hash in the same bucket
- Distant points end up in different buckets

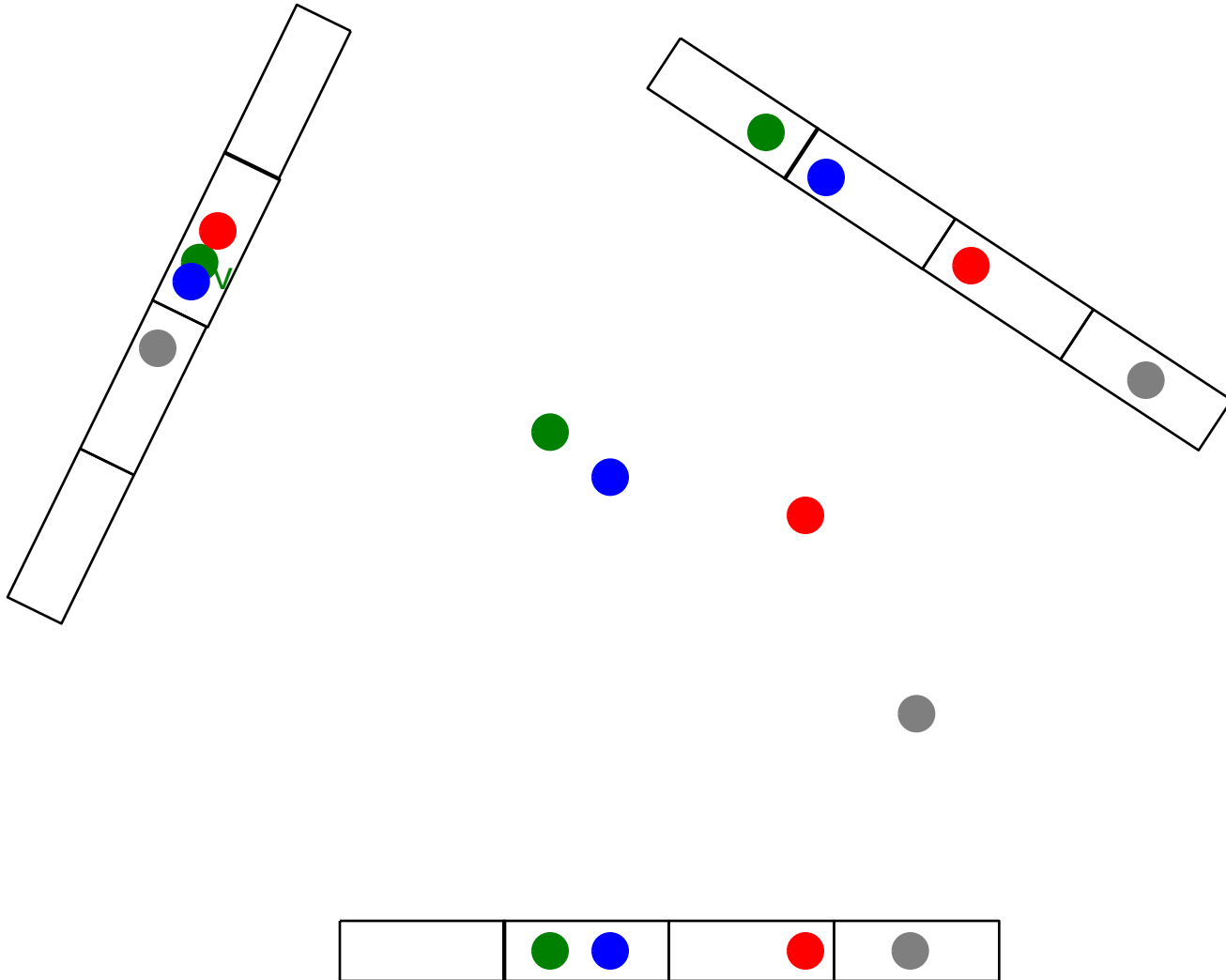


■ Two “unlucky” cases:

- **Top:** unlucky quantization
- **Bottom:** unlucky projection

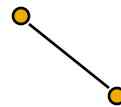


Multiple Projections

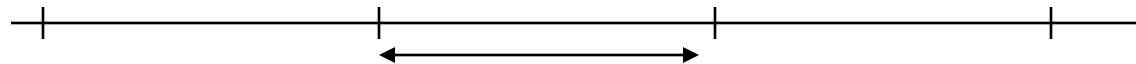


Projection of Points

Points at
distance d



If $d \ll a$, then
the chance the
points are in the
same bucket is
at least $1 - d/a$.



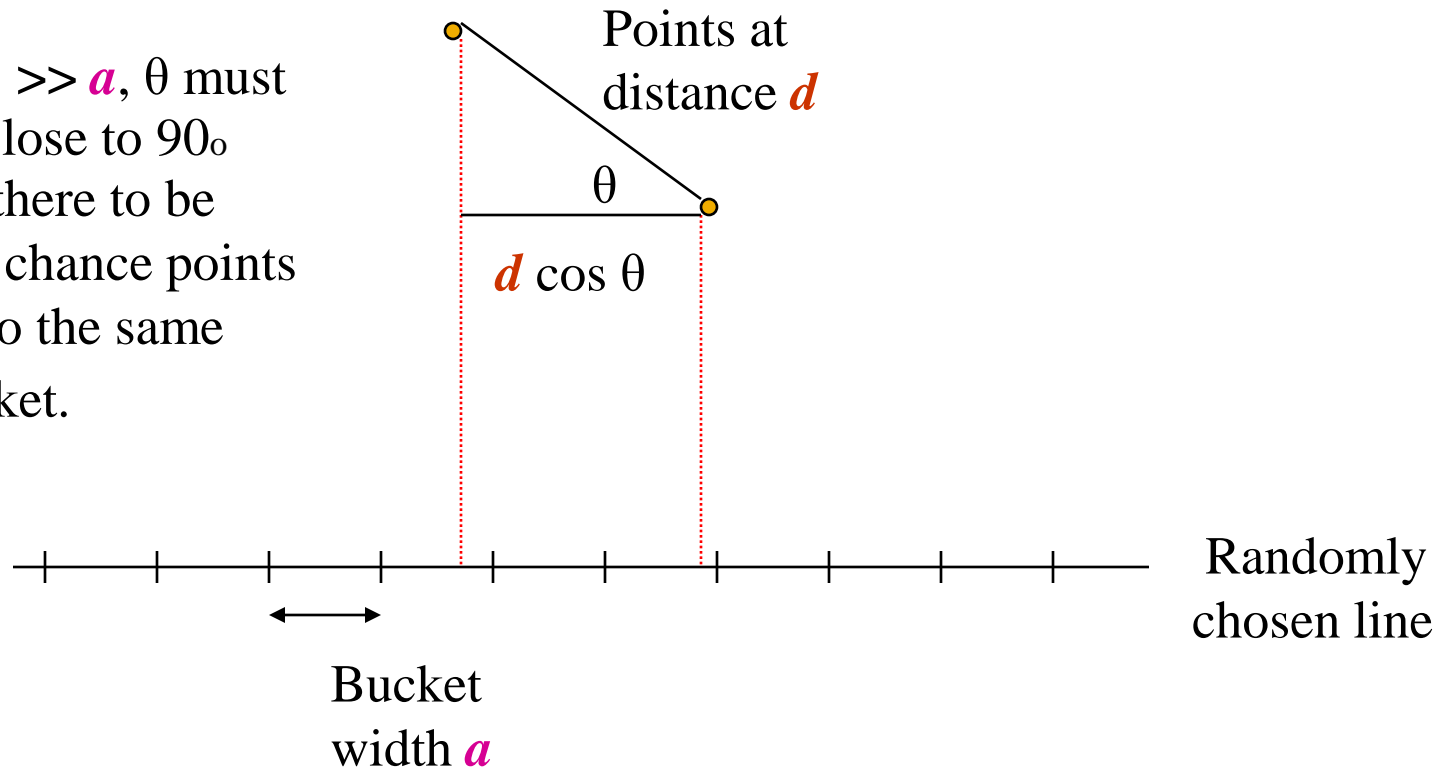
Bucket
width a

Randomly
chosen line



Projection of Points

If $d \gg a$, θ must be close to 90° for there to be any chance points go to the same bucket.



An LS-Family for Euclidean Distance

- If points are distance $d < a/2$, prob. they are in same bucket $\geq 1 - d/a = \frac{1}{2}$
- If points are distance $d > 2a$ apart, then they can be in the same bucket only if $d \cos \theta \leq a$
 - $\cos \theta \leq \frac{1}{2}$
 - $60^\circ < \theta < 90^\circ$, 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



Fixup: Euclidean Distance

- Projection method yields a $(a/2, 2a, 1/2, 1/3)$ -*sensitive* family of hash functions
- For previous distance measures, we could start with an (d_1, d_2, p_1, p_2) -*sensitive* family for any $d_1 < d_2$, and drive p_1 and p_2 to 1 and 0 by AND/OR constructions
- Note: Here, we seem to need $d_1 \leq 4 d_2$
- In the calculation on the previous slide we only considered cases $d < a/2$ and $d > 2a$



Fixup-(2)

- But as long as $d_1 < d_2$, the probability of points at distance d_1 falling in the same bucket is greater than the probability of points at distance d_2 doing so
- Thus, the hash family formed by projecting onto lines is an (d_1, d_2, p_1, p_2) -sensitive family for some $p_1 > p_2$
 - Then, amplify by AND/OR constructions



Tell me and I forget.
Show me and I remember.
Involve me and I understand.

Thank you! Q&A



Appendix: Efficiently Matching Sets of Features with Random Histograms



Motivation

- Set-of-feature representation is popular

- Example: images, video, audio, .etc
- Higher empirical results than global feature

- The set matching problem

$$X = \{x_1, x_2, \dots, x_{n_1}\} \quad Y = \{y_1, y_2, \dots, y_{n_2}\}$$

Given feature similarity measure $s(x_i, y_j)$

- How to define set similarity $S(X, Y)$ meaningfully ?
- How to evaluate $S(X, Y)$ efficiently ?

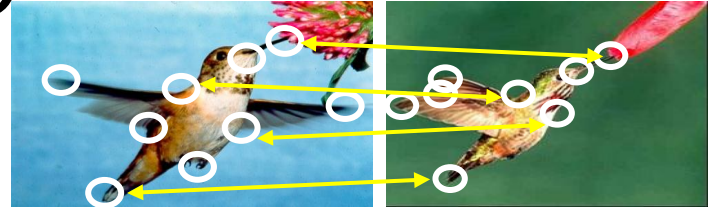


Set Similarity Measures

■ One-to-one match: $O(N^3)$

$$S_1(X, Y) = \max \sum_{\langle x, y \rangle} s(x, y)$$

s.t. each feature used at most once

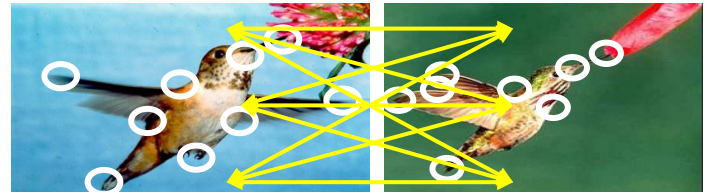


■ Bipartite graph matching

■ Weighted version: EMD[Rubner98]

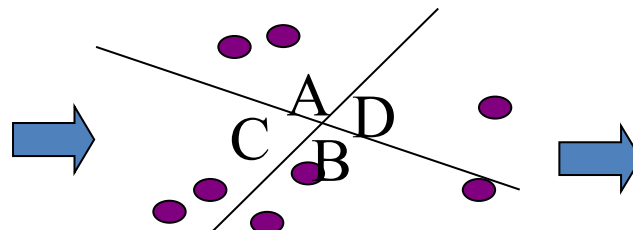
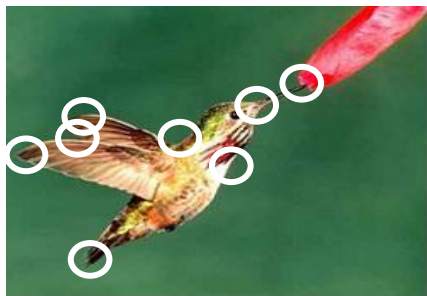
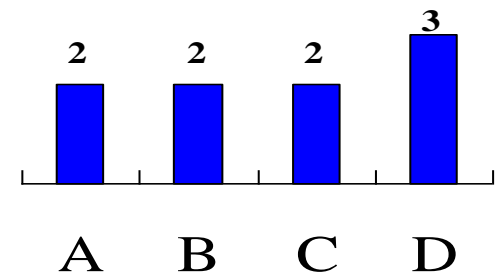
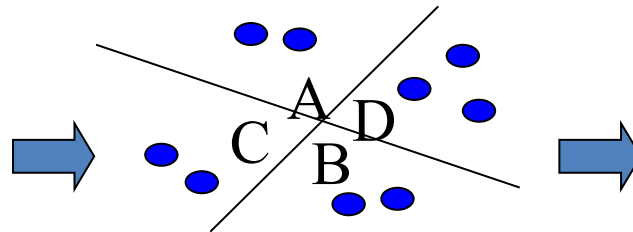
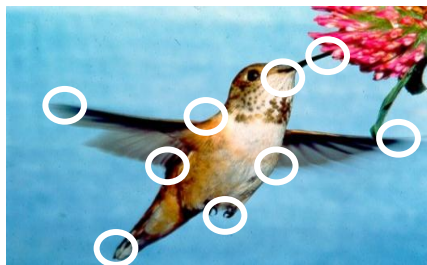
■ One-to-many match: $O(N^2)$

$$S_2(X, Y) = \sum_{x \in X} \sum_{y \in Y} s(x, y).$$

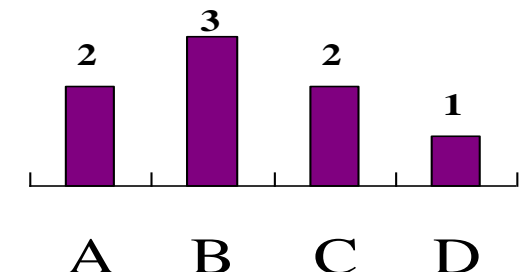


The Histogram Approach

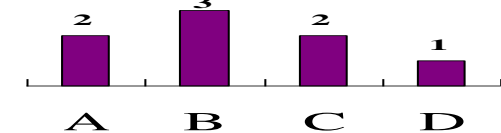
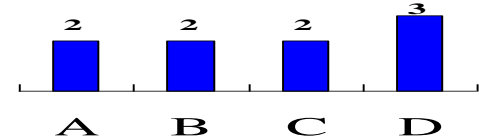
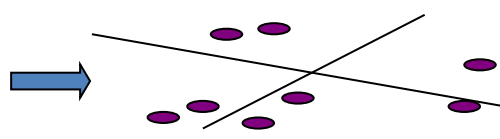
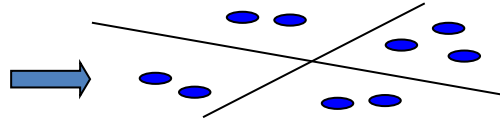
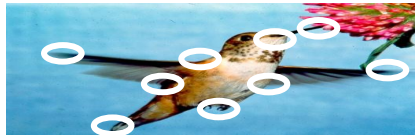
- Embed to histograms via feature space quantization
- Improve online matching speed significantly
- Potential precision loss, but good empirical performance



(feature space)

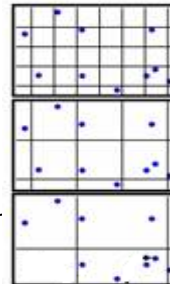


Existing Quantization Methods



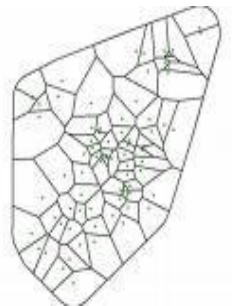
Pyramid match

- Originally uses scalar quantization[Grauman 05]
 - Histograms are sparse
- New version with vector quantization [Grauman07]



Visual words

- Vector quantization
- Time consuming clustering
- Not incremental



Contribution

- A randomized histogram construction
 - Quantization by Locality Sensitive Hashing
 - Compact histogram representation
 - Embedding is fast and parallelizable
 - Support various feature similarity measures
- Evaluation with three task settings
 - Very fast matching speed with good accuracy
 - E.g. 20x speedup over Pyramid Match on Caltech 101

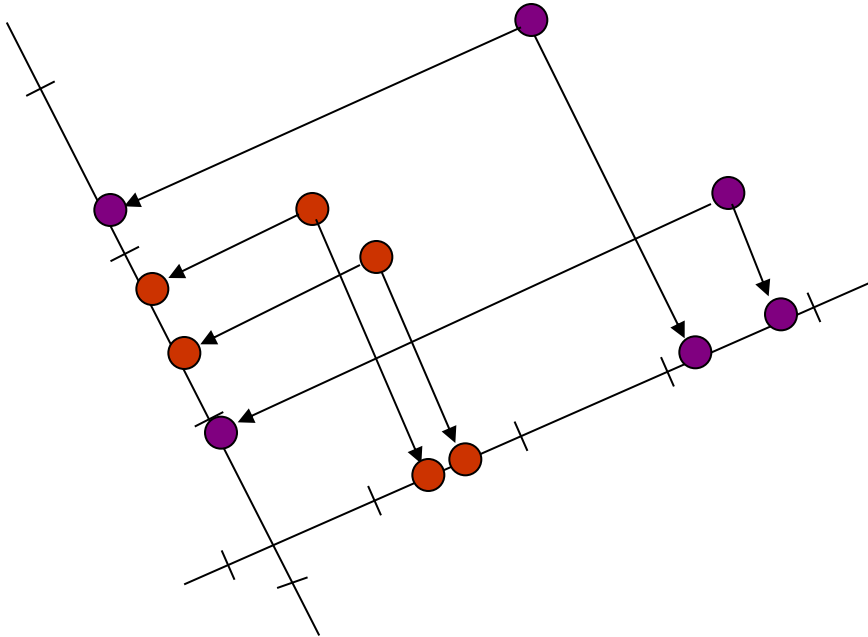


Locality Sensitive Hashing

- Idea: hash functions that similar objects are more likely to have the same hash [Indyk98]

$$\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = s_{\mathcal{H}}(x, y),$$

$s_{\mathcal{H}}$: similarity induced by the LSH family \mathcal{H} .



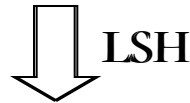
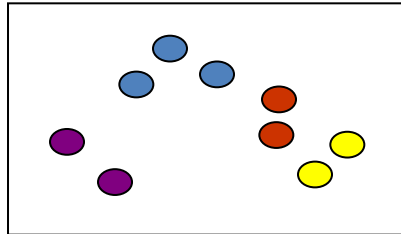
LSHs have been designed for

- L1 and L2 distance
- Cosine similarity
- Hamming distance
- Jaccard index for set similarity
-

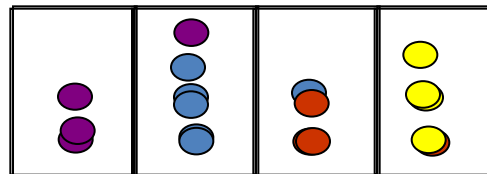


Random Histogram Construction

Features of one object

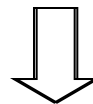


Hash Table



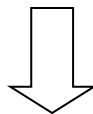
Repeat M times

Histogram



$\langle 2, 4, 2, 3 \rangle$

...



concatenate

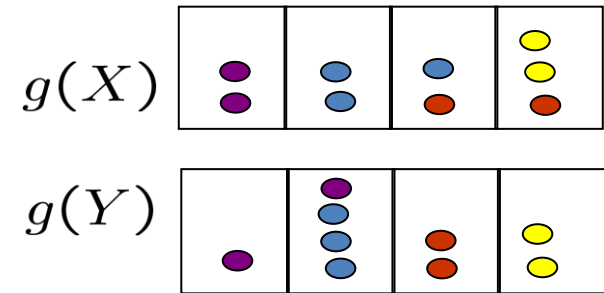
$\langle 2, 2, 2, 3, 1, 4, 2, 2, \dots \rangle$



Matching Random Histograms

■ One-to-one match

$$\begin{aligned} S_1(X, Y) &= \sum_i \min[g_i(X), g_i(Y)] \\ &= 1 + 2 + 2 + 2 = 7 \end{aligned}$$



■ One-to-many match

$$\begin{aligned} S_2(X, Y) &= \sum_i g_i(X) \cdot g_i(Y) \\ &= 2 + 8 + 4 + 6 = 20 \end{aligned}$$

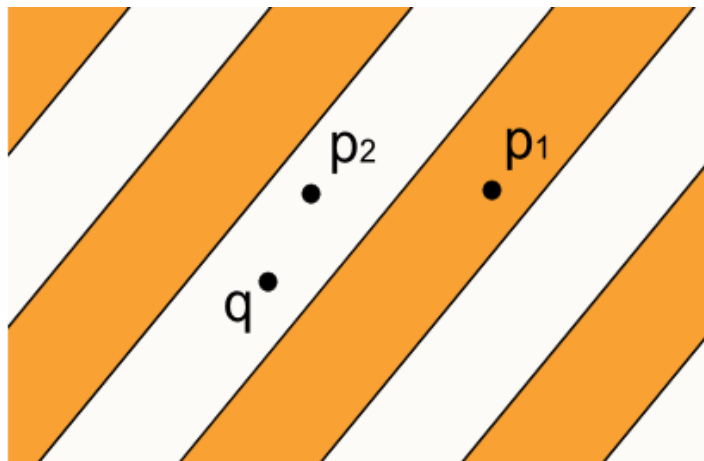
$$E[S_2(X, Y)] = \sum_{x \in X} \sum_{y \in Y} s_{\mathcal{H}}(x, y)$$

Choosing LSH: L2 Distance

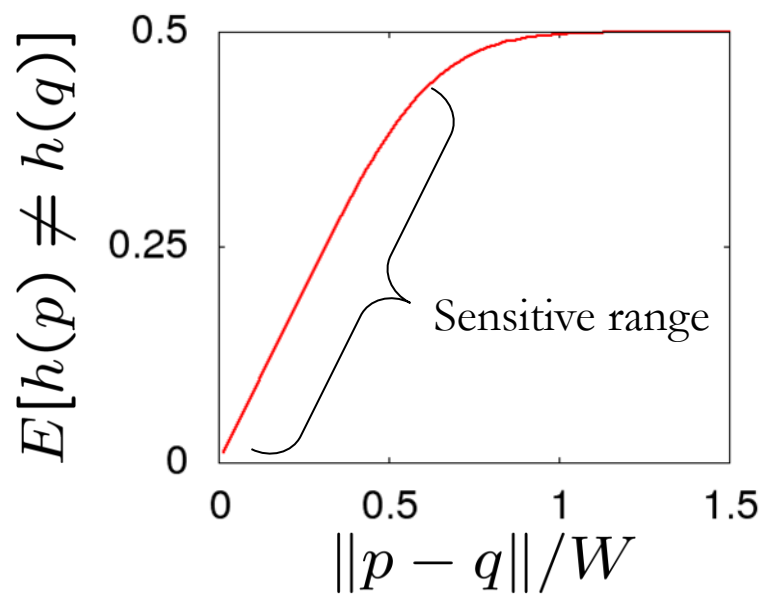
- Least significant bit of the p-stable LSH [Datar04]

$$h(p) = \lfloor \frac{A \cdot p + b}{W} \rfloor \bmod 2$$

- LSB only so that hash values distribute evenly

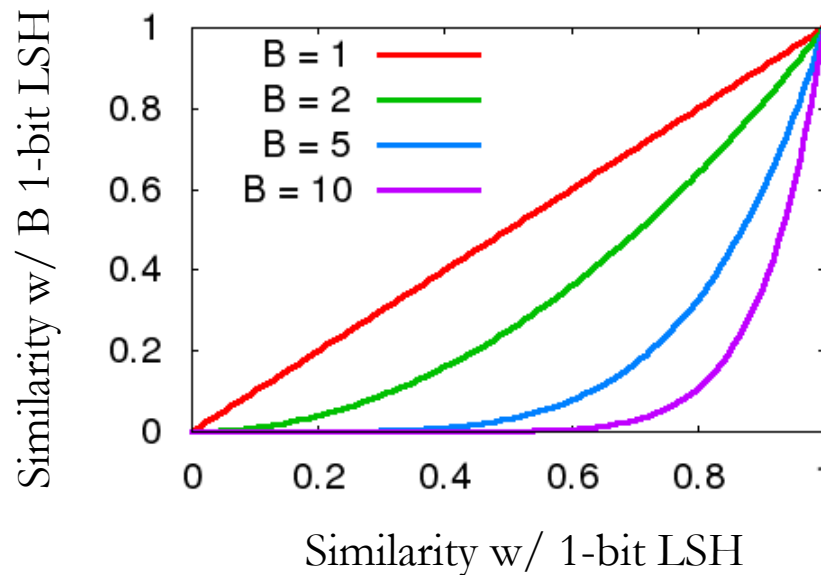


W : stripe width



Choosing LSH Scheme

- Concatenate B 1-bit LSH to make a B -bit LSH
 - Enhance discriminating power
 - Enlarge histogram size to 2^B
 - Tunable tradeoff between space vs. accuracy



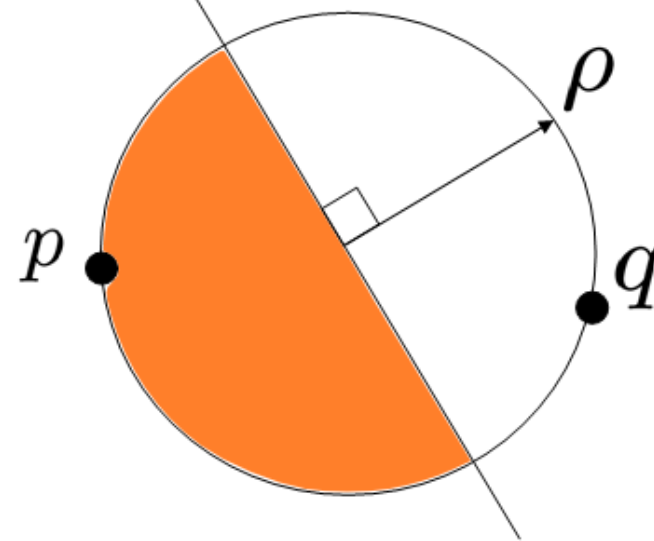
Another LSH: Cosine Similarity

- Cosine Similarity $d_{cos}(p, q) = \cos(p \wedge q)$
- Random hyper-plane sketch [Charikar02]

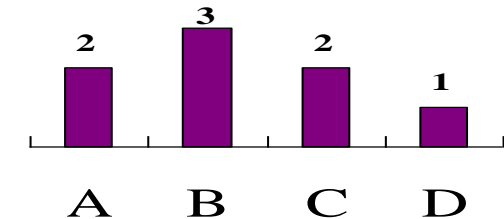
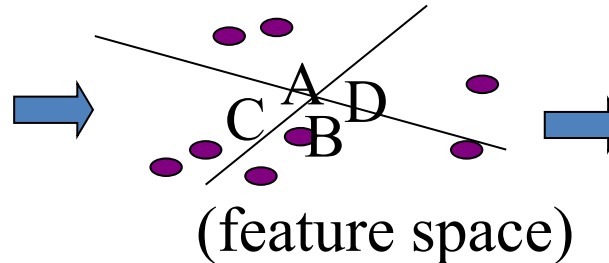
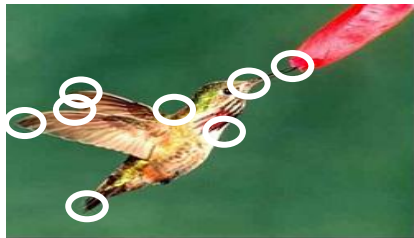
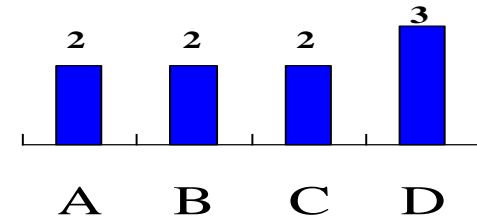
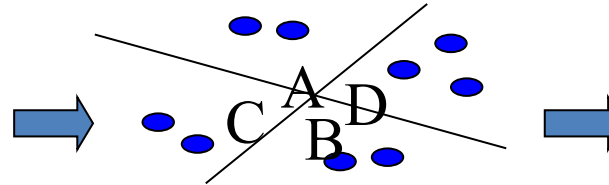
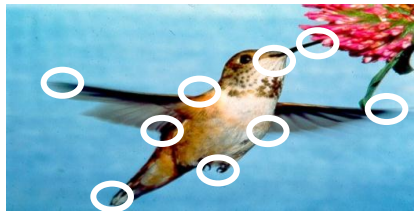
$$h(p) = \begin{cases} 0 & \text{if } \rho \cdot p < 0 \\ 1 & \text{if } \rho \cdot p \geq 0 \end{cases}$$

ρ : normal vector of a random hyperplane

$$E[h(p) \neq h(q)] = \frac{p \wedge q}{\pi}$$



Recap of Our Scheme



Local feature extraction

Quantization with LSH

- L2 distance
- Cosine similarity
-

Histogram matching

- One to one
- One to many

Histogram size = $N2^B$, doesn't explicitly depend on set size!



Evaluation

■ Three tasks

- Object recognition
- Image similarity search
- Near duplicate video detection

■ Performance measure

$$\text{Accuracy} = \frac{\# \text{ correctly labeled}}{\text{total number}}$$

$$\text{Average Precision} = \frac{1}{k} \sum_{i=1}^k \frac{i}{\text{rank}_i}$$

■ Platform: Pentium 4 3.2GHz CPU+ 6GB mem



Recognition: Caltech 101

- Benchmark: 101 categories
- Features: 10D PCA SIFT + 2D position feature
- 15 images/category for training, rest for testing

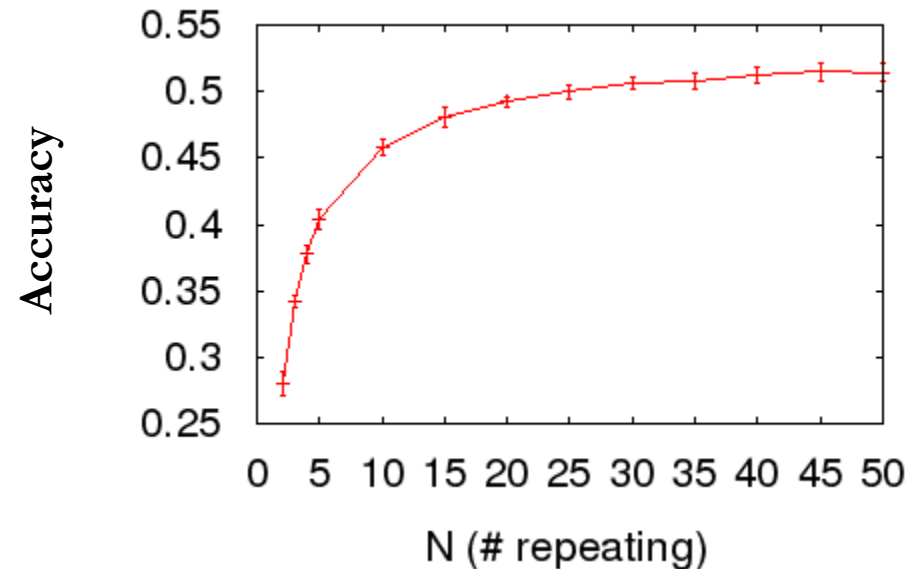
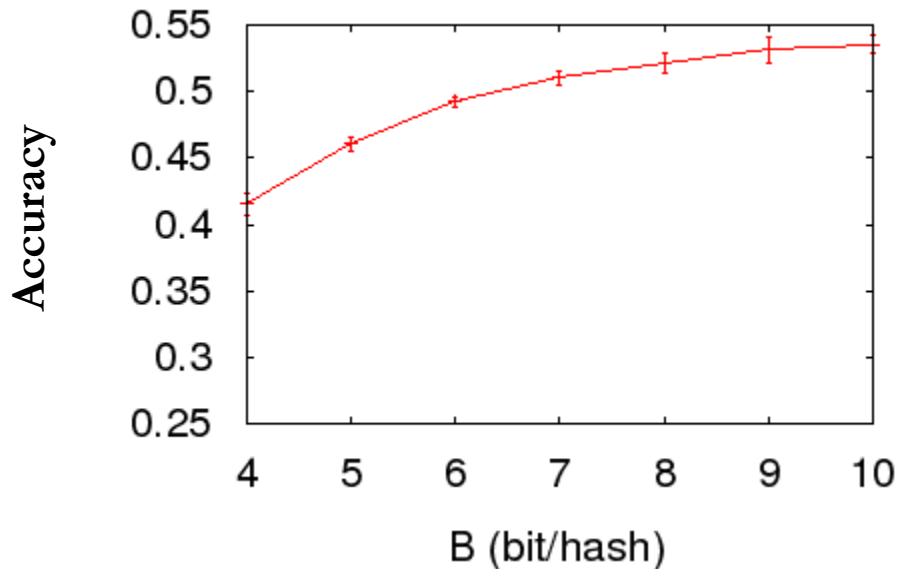
	Grauman05 Pyramid	Zhang07 EMD	Ours A (speed)	Ours B (accuracy)
Accuracy	0.50	0.539	0.497	0.541
Time/match	100 μ s	slow	0.4μs	6μs

All methods use SVM for learning.



Performance vs. Parameters

- Single histogram size: 2^B
- Full histogram size: $N 2^B$



Recognition: Graz-01

- Benchmark: bike, person and background
 - High intra-class scale and pose variation
- DoG detector + SIFT feature
- 100 positive + 100 negative for training
- ROC equal error rate

	Opelt04	Zhang07 EMD	Lazebnik06 spatial pyramid	Ours
Bike	0.865	0.920	0.863	0.883
Person	0.808	0.880	0.823	0.805



Content-Based Image Retrieval

- Benchmark [Lv04]
 - 10K images, 32 sets of manually selected images
- Feature extraction
 - Segmented with JSEG
 - 14-D color & size feature from each segment
- Direct K-NN search

Method	SIMPLIcity EMD like	Lv04 EMD	Ours
Average precision	0.331	0.548	0.548
Time/match	N/A	50 μ s	0.3μs



Near-Duplicate Video Detection

- Benchmark [Wu07]
 - 24 sets of video clips downloaded from Youtube, etc
 - Manually labeled to find near duplicates
- Feature extraction
 - 124 key frames/video clip
 - HSV based color histogram from each key frame

Wu07	SIG_CH	SET_NDK
Average Precision	0.892	0.952
Time/match	“fast”	“minutes”

Ours	SIG_CH	Embedding
Average precision	0.835	0.893
Time/match	0.17ms	4.6ms



Discussion

- Histogram might still be large
 - See paper for two methods of compact representation
 - Further reduction is possible by feature selection
- Precision loss in quantization
 - Vanilla k-NN search is not good enough
 - Can be compensated by machine learning



Conclusion

- Matching sets of features with random histograms
- Efficient
 - fast embedding and matching, good empirical accuracy
 - 20x faster than Pyramid Match (~500 features/image)
 - 150x faster than EMD (~10 features/images)
- Flexible
 - Support various feature and set similarity measures
 - Easy to tradeoff between speed and accuracy

