Locality Sensitive Hashing & ANN

CS 584: Big Data Analytics

Material adapted from
Piotr Indyk (https://people.csail.mit.edu/indyk/helsinki-2.pdf) &
Jure Leskovec and Jeffrey Ulman (http://web.stanford.edu/class/cs246/handouts.html) &
Marc Alban (http://www.cs.utexas.edu/~grauman/courses/spring2008/slides/Marc_Demo.pdf)

Recap: NN

- Nearest neighbor search in Rd is very common in many fields of learning, retrieval, compression, etc.

- Exact nearest neighbor: Curse of dimensionality

Algorithm	Query Time	Space
Full indexing	O(d log n)	$n^{O(d)}$
Linear scan	O(dn)	O(dn)

- Approximate NN

  - KD-trees: optimal space, O(r)d log n query time

Approximate Nearest Neighbor (ANN)

- Idea: rather than retrieve the exact closest neighbor, make a “good guess” of the nearest neighbor

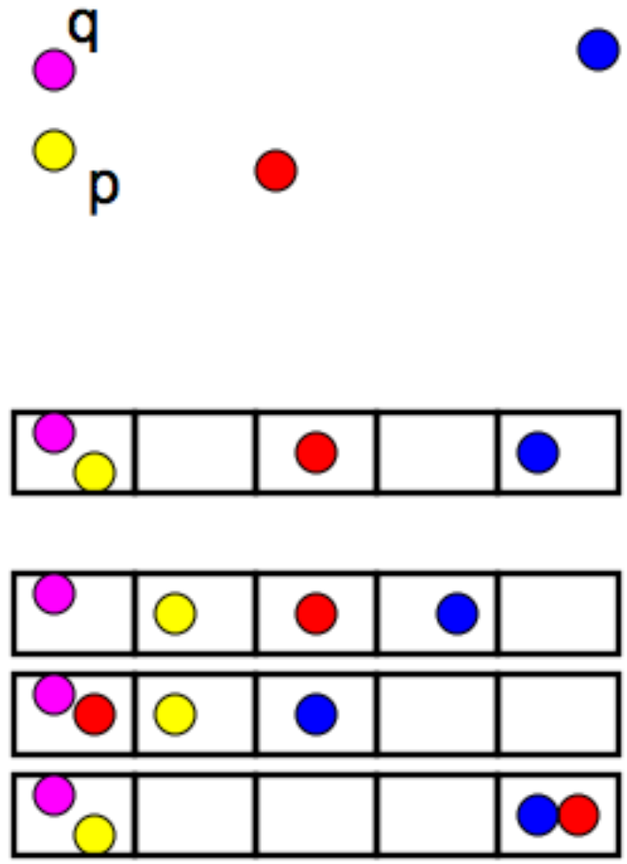
- c-ANN: for any query q and points p:

  - r is the distance to the exact nearest neighbor q

  - Returns p in P, $||p - q|| \leq cr$, with probability at least $1 - \delta,\ \delta > 0$

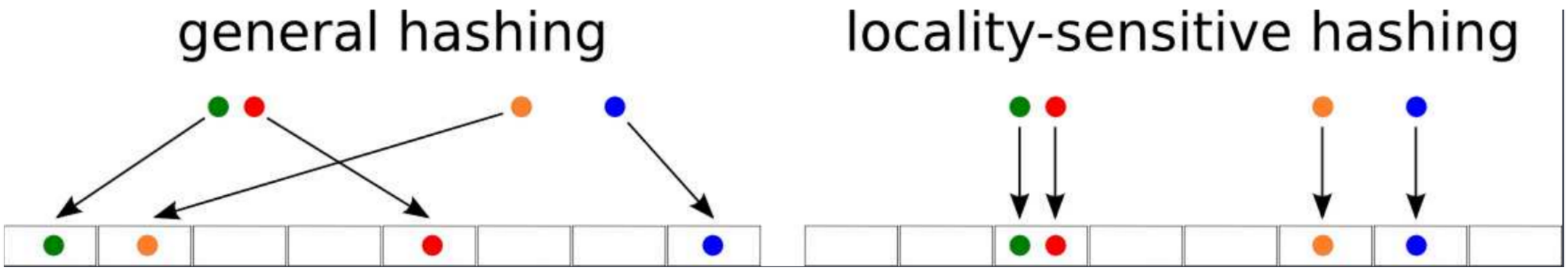
Locality Sensitive Hashing (LSH) [Indyk-Motwani, 1998]

- Family of hash functions
  - Close points to same buckets
  - Faraway points to different buckets
- Idea: Only examine those items where the buckets are shared
- (Pro) Designed correctly, only a small fraction of pairs are examined
- (Con) There maybe false negatives



LSH: Bigfoot of CS

- The mark of a computer scientist is their belief in hashing
  - Possible to insert, delete, and lookup items in a large set in O(1) time per operation
- LSH is hard to believe until you seen it
  - Allows you to find similar items in a large set without the quadratic cost of examining each pair

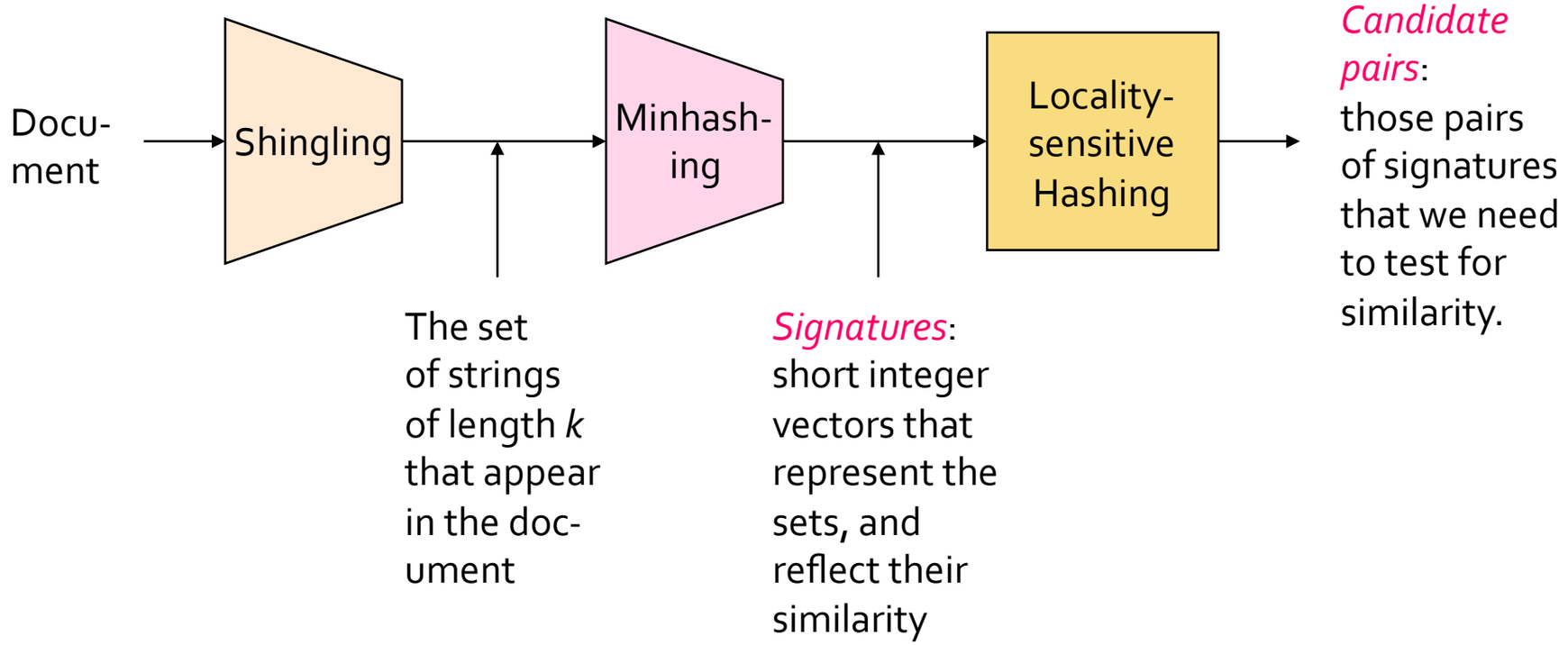


Finding Similar Documents

- Goal: Given a large number of documents, find “near duplicate” pairs
- Applications:
  - Group similar news articles from many news sites
  - Plagiarism identification
  - Mirror websites or approximate mirrors
- Problems:
  - Too many documents to compare all pairs
  - Documents are so large or so many they can’t fit in main memory

Finding Similar Documents: The Big Picture

- Shingling: Convert documents to sets
- Minhashing: Convert large sets to short signatures while preserving similarity
- LSH Query: Focus on pairs of signatures likely to be similar



Shingling: Convert documents to sets

- Account for ordering of words
- A k-shingle (k-gram) for a document is a sequence of k tokens that appears in the document
  - Example: k = 2; document D1 = abcab
    Set of 2-shingles: S(D1) = {ab, bc, ca}
- Represent each document by a set of k-shingles

Shingles and Similarity

- Documents that are generally similar will share many singles
- Changing a word only affects k-shingles within k-1 from the word
  - Example: k = 3, “The dog which chased the cat” versus “The dog that chased the cat”
  - Only 3-shingles replied are g_w, _wh, whi, hic, ich, ch_, h_c
- Reordering paragraphs only affects the 2k shingles that cross paragraph boundaries

Shingles and Compression

- k must be large enough, or most documents will have most shingles (not useful for differentiation)
  - k = 8, 9, 10 is often used in practice
- For compression and uniqueness, hash each single into tokens (e.g., 4 bytes)
- Represent a document by the tokens (set of hash values of its k-shingles)

Finding Similar Documents: Distance Metric

- Each document is a binary vector in the space of the tokens
  - Each token is a dimension
  - Vectors are very sparse

- Natural similarity measure is the Jaccard similarity
  - Size of the intersection of two sets divided by the size of their union
  - Notation: $\mathrm{Sim}(C_1, C_2) = \dfrac{C_1 \cap C_2}{C_1 \cup C_2}$

From Sets to Binary Matrices

- Rows = elements of the universal set (i.e., the set of all tokens)
- Columns = documents
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is Jaccard similarity of the corresponding sets
- Typical matrix is sparse!

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

Why Shingling is Insufficient

- Suppose we need to find near-duplicate items amongst 1 million documents
- Naively, we would have to compute all pairwise Jacquard similarities
  - $N(N-1)/2 = 5 * 10^{11}$ comparisons
  - At 10^5 seconds a day and 10^6 comparisons per second, this would take 5 days!
- If we are looking at 10 million documents, this will take more than 1 year

Hashing Documents

- Idea: Hash each document (column) to a small signature h(C) such that
  - h(C) is “small enough” that it fits in RAM
  - sim(C_1, C_2) is the same as the “similarity” of h(C_1) and h(C_2)
- In other words, you want to use an LSH function
  - If sim(C_1, C_2) is high, then P(h(C_1) = h(C_2)) is high
  - If sim(C_1, C_2) is low, then P(h(C_1) = h(C_2)) is low

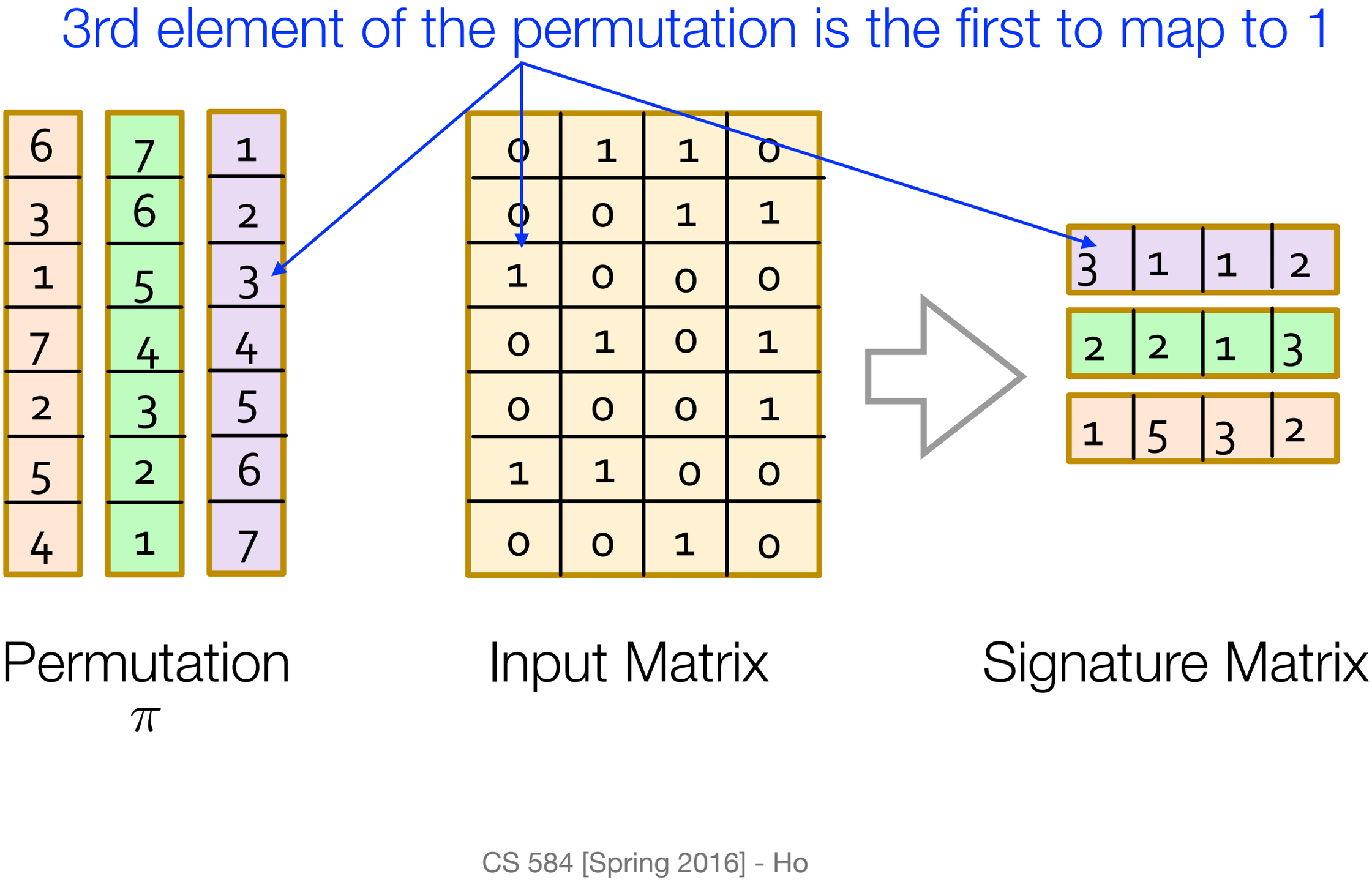
Minhashing

- Hash function depends on the similarity metric
  - Not all similarity metrics have a suitable hash function
- Suitable hash function for Jaccard similarity is minhashing
- Imagine rows of binary matrix permuted under random permutation π
- Hash function is the index of the first (in the permuted order) row in which column C has value 1

$$h_\pi(C) = \min_\pi \pi(C)$$

- Use several independent hash functions (i.e., permutations) to create signature of a column

Example: Minhashing



Minhashing Property

Claim: $P[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$

- X is a document, y is a shingle in document

- Equally likely that any y is mapped to the min element

$$P[\pi(y) = \min(\pi(X))] = 1/|X|$$

- Let y be such that $\pi(y) = \min(\pi(C_1 \cup C_2))$
  (one of the two columns had to have 1 at position y)
  => probability that both are true is $P(y \in C_1 \cap C_2)$

$$P[\min(\pi(C_1)) = \min(\pi(C_2))] = |C_1 \cap C_2|/|C_1 \cup C_2)|$$
$$= \text{sim}(C_1, C_2)$$

Minhashing and Similarity

- The similarity of the signatures is the fraction of the minhash functions (rows) in which they agree

- Expected similarity of two signatures is equal to the Jaccard similarity of the columns

- The longer the signatures, the smaller the expected error

Example: Minhashing and Similarities

Permutation

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

Input Matrix

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

Signature Matrix

3	1	1	2
2	2	1	3
1	5	3	2

	1-2	2-3	3-4	1-3	1-4	2-4
Jaccard	1/4	1/5	1/5	0	0	1/5
Signature	1/3	1/3	0	0	0	0

Minhash Signatures

- Pick K random permutations of the row
    - Permutation rows can be prohibitive for large data, so use row hashing to get random row permutation
- Signature of the document can be represented as a column vector and is a sketch of the contents
- Compression long bit vectors into short signatures as signature is no ~ k bytes!

LSH: Signatures to Buckets

- Hash objects such as signatures many times so that similar objects wind up in the same bucket at least once, while other pairs rarely do

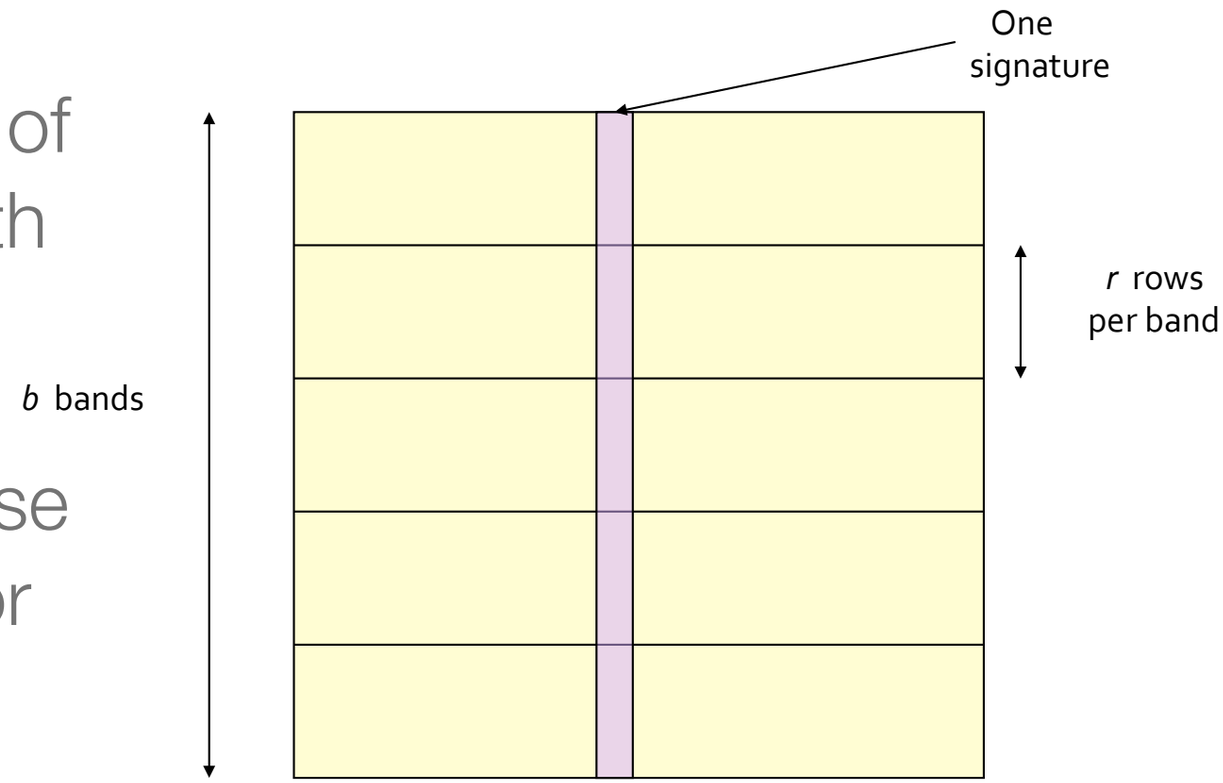
- Pick a similarity threshold t which is the fraction in which the signatures agree to define “similar”

- Trick: Divide signature rows into bands

  - A hash function based on one band

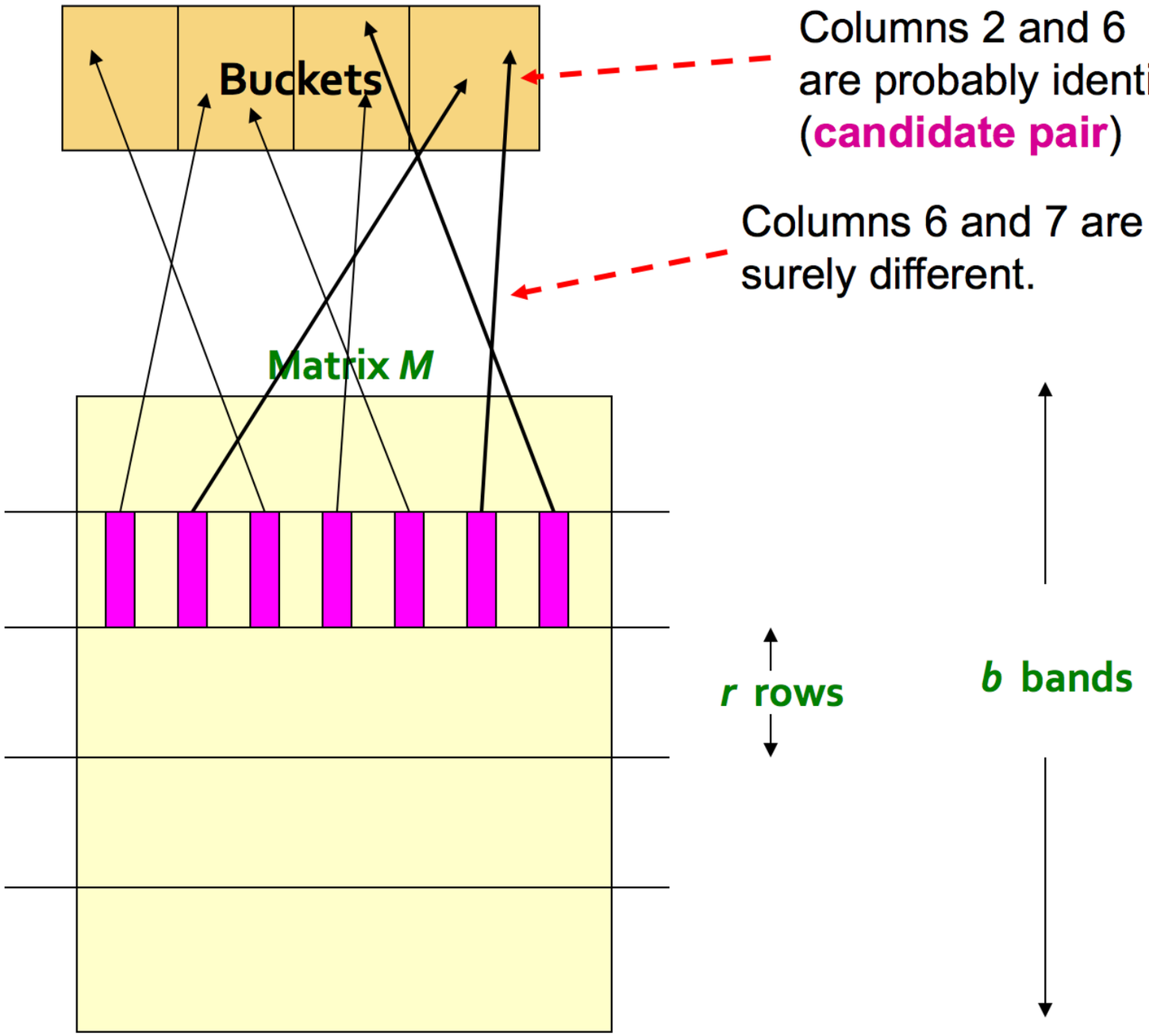
Band Partition

- Divide matrix into b bands of r rows
- For each band, hash its portion of each column to a hash table with k buckets
- Candidate column pairs are those that hash to the same bucket for at least 1 band
- Tune b and r to catch most similar pairs but few non similar pairs



Matrix M

Hash Function for One Bucket



Example of Bands

- Suppose 100k documents (columns)
- Signatures of 100 integers (rows)
- Each signature takes 40MB
- 5B pairs of signatures can take awhile to compare
- Choose 20 bands of 5 integers / band to find pairs of 80% similarity

Find 80% Similar Pairs

- We want C_1, C_2 to be a candidate pair, which is they hash to at least 1 common band

- 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 of the 20 bands: $(1 - 0.328)^{20} = 0.00035$

  - 1/3000th of the column pairs are false negatives (missing the actual neighbors)

What about 30% Similarity?

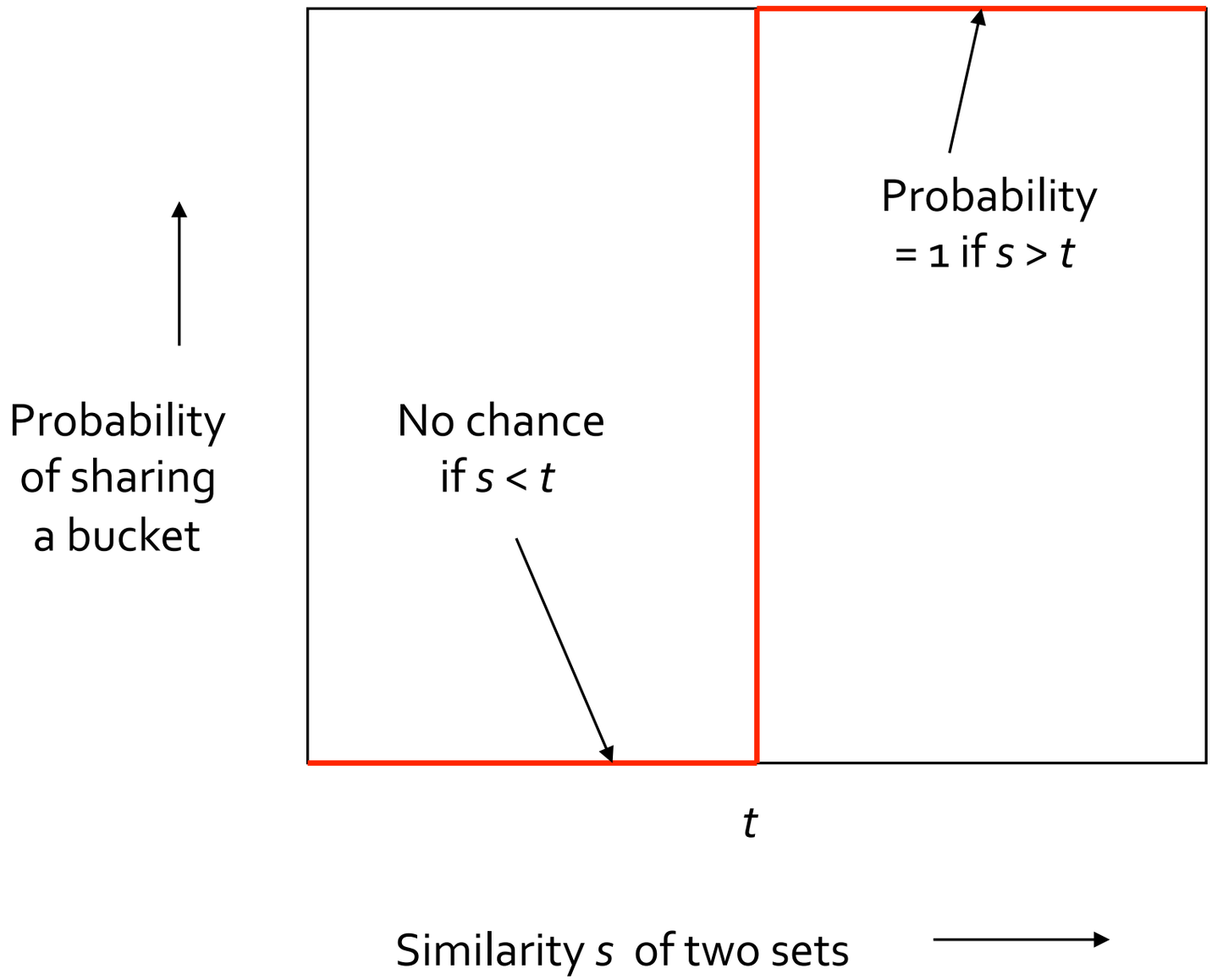
- Since 30% is less than our goal of 80%, we want C_1 and C_2 to hash to NO common buckets

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$

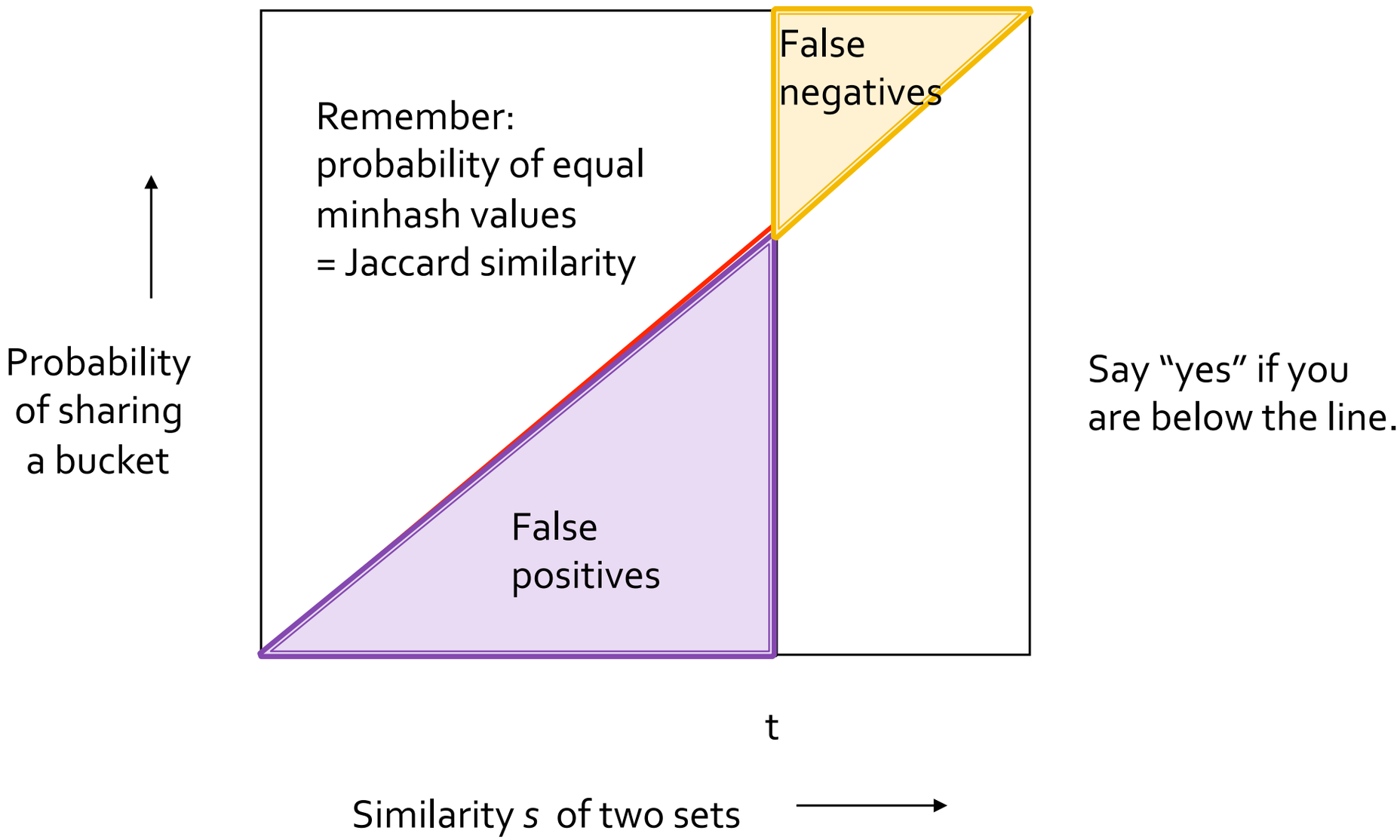
- Probability C_1, C_2 are not similar in all of the 20 bands: $1 - (1 - 0.00243)^{20} = 0.0474$

  - 4.74% pairs of documents with similarity of 0.3% end up being candidate pairs (false positives)

LSH: What We Want



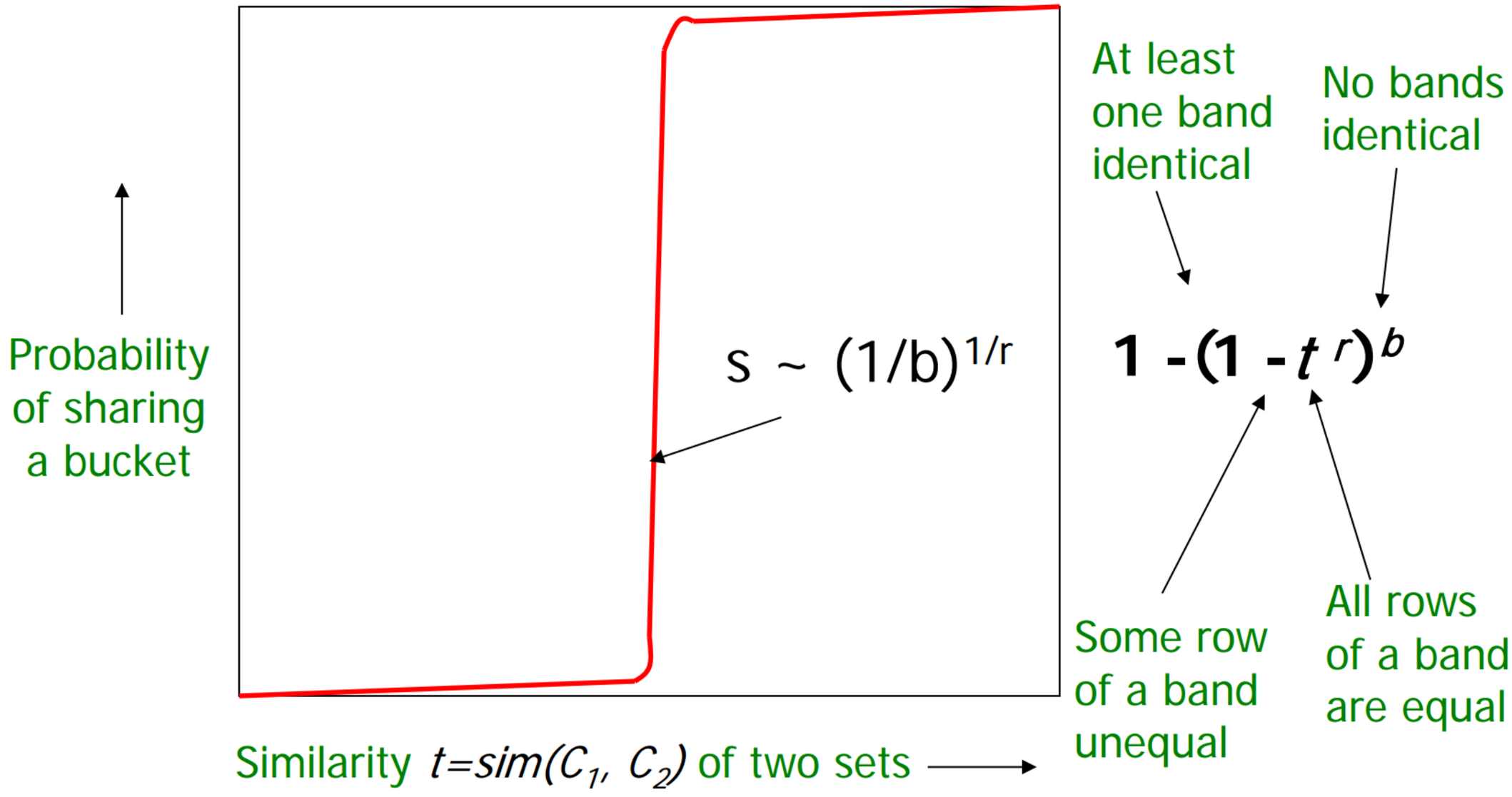
LSH: What One Band of One Row Yields



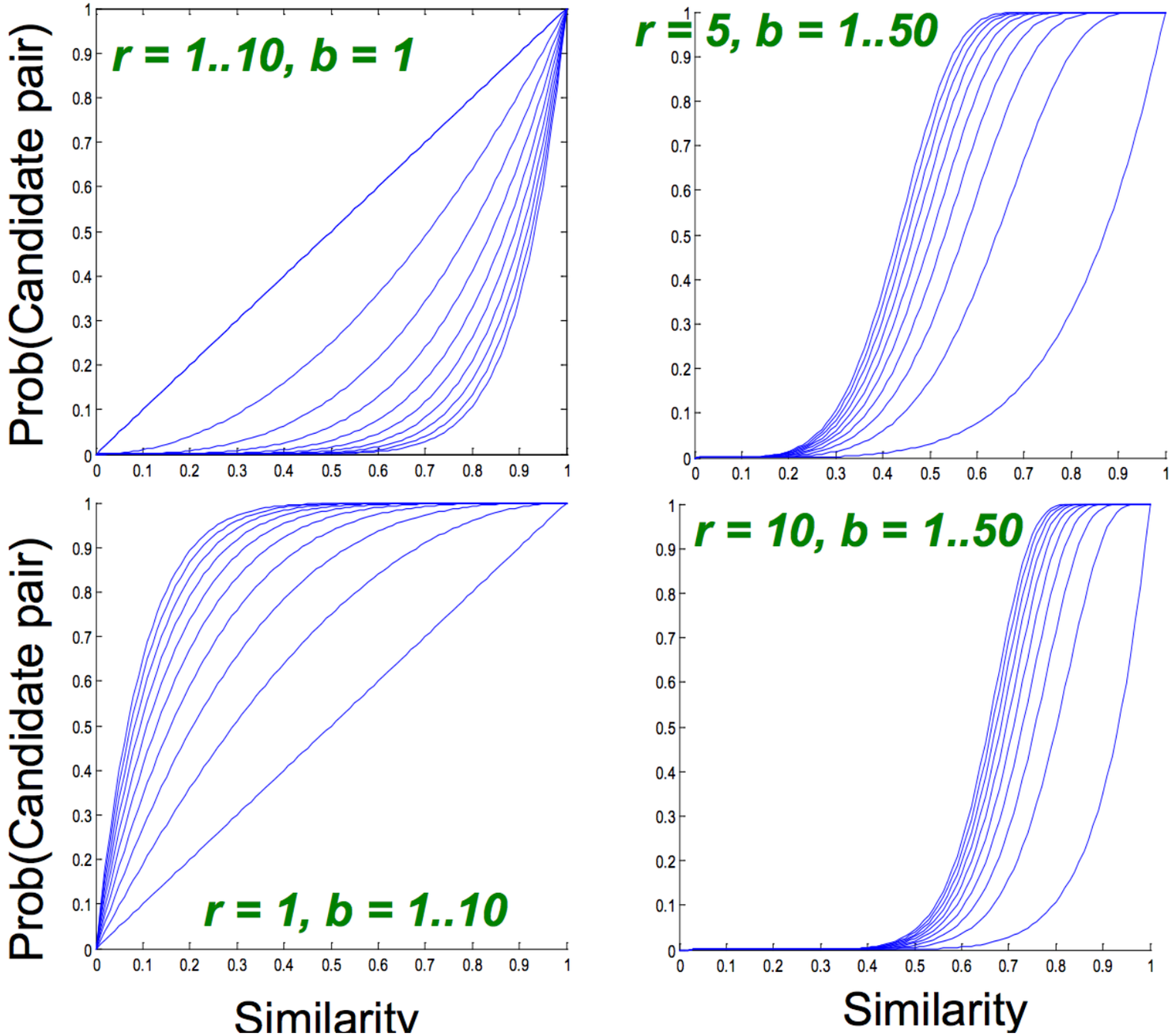
LSH Parameters

- Columns C_1 and C_2 have similarity t
- Pick any band (r rows)
  - Probability that all rows in band equal: t^r
  - Probability unequal: $1-t^r$
- Probability that no band is identical: $(1-t^r)^b$
- Probability that at least one band is identical: $1 - (1-t^r)^b$

LSH: What b Bands of r Rows yields



LSH: S-Curves as a function of b and r



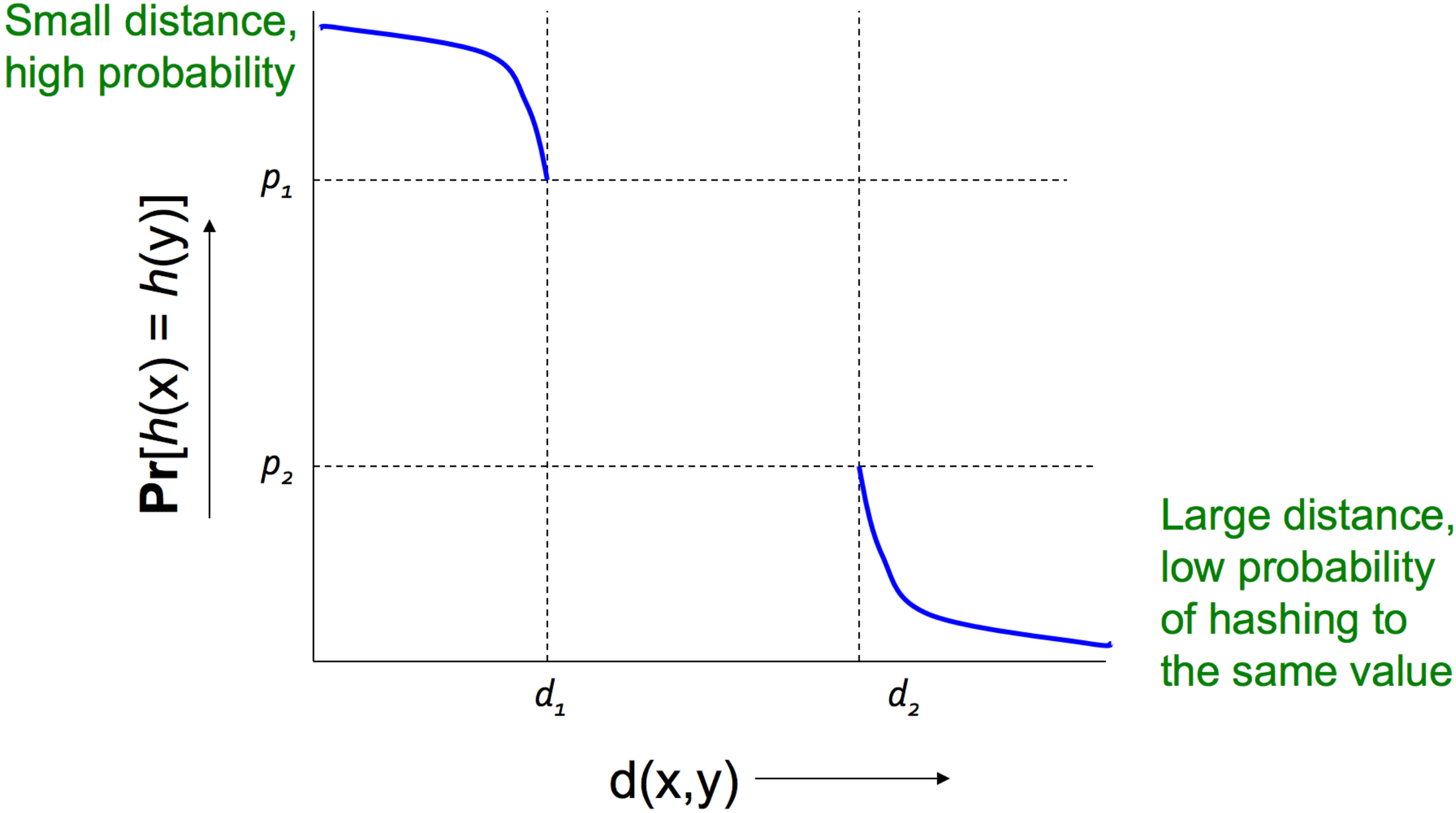
LSH Definition

- Suppose we have a metric space S of points with a distance measure d

- An LSH family of hash functions, $\mathcal{H}(r, cr, P_1, P_2)$, has the following properties for any $q, p \in \mathcal{S}$
  - If $d(p, q) \leq r$, then $P_{\mathcal{H}}[h(p) = h(q)] \geq P_1$
  - If $d(p, q) \geq cr$, then $P_{\mathcal{H}}[h(p) = h(q)] \leq P_2$

- Theory leaves unknown what happens to pairs at distances between r and cr

LSH Family of Hash Functions

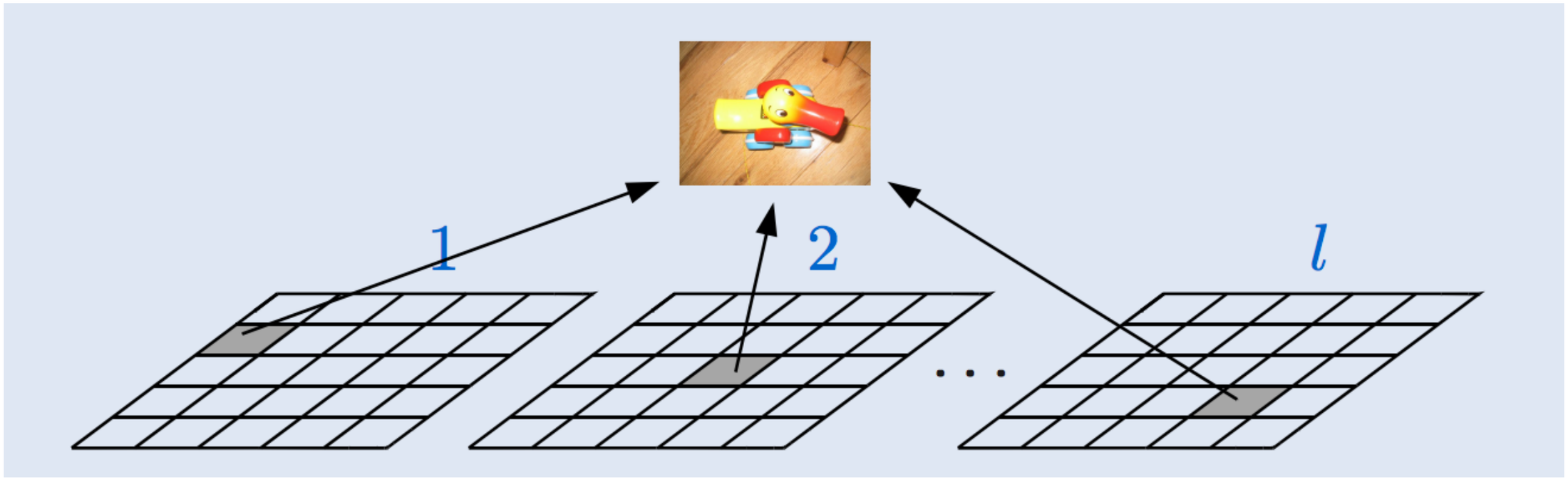


k-bit LSH Functions

- A k-bit locality sensitive hash function (LSHF) is defined as $g(p) = [h_1(p),\ h_2(p),\ \cdots,\ h_k(p)]^\top$
  - Each h_i is chosen randomly from $\mathcal{H}$
  - Each h_i results in a single bit
- P(similar points collide) $\geq 1 - (1 - \frac{1}{P_1})^k$
- P(dissimilar points collide) $\leq (P_2)^k$

LSH Preprocessing

- Select L random k-bit LSHF, g_1, …, g_L
- For all points p, hash p to the buckets $g_1(p)$, …, $g_L(p)$
- Preprocessing space: O(L n)



LSH Querying

- Given a new point q, retrieve the points from buckets g1(q), g2(q), …, until
  - Either the points from all L buckets have been retrieved, or
  - Total number of points retrieved exceeds 3L
- Answer the query based on the retrieved points
- Total Query Time: O(dL)

Hamming Space

- Hamming space is the set of all 2N binary strings of length N

- Hamming distance between two equal length binary strings is the number of positions for which the bits are different

- $\| 1011101, 1001001 \|_H = 2$

- $\| 1110101, 1111101 \|_H = 1$

Hamming Space: Hashing Family

Let a hashing family be defined as $h_i(p) = p_i$ where p_i is the i^{th} bit of p

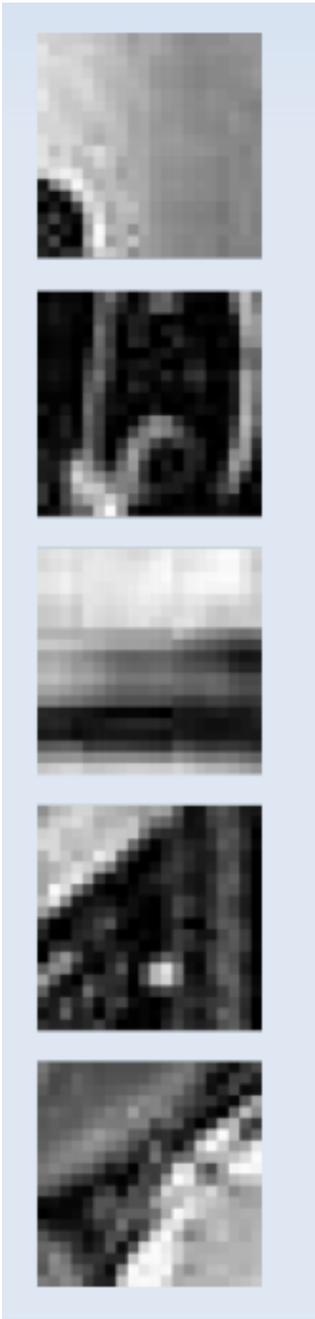
$$P_{\mathcal{H}}[h(p) \neq h(q)] = \frac{||p,q||_H}{d}$$

$$P_{\mathcal{H}}[h(p) = h(q)] = 1 - \frac{||p,q||_H}{d}$$

- Family of hash functions are locality sensitive
- Comparison with Minhash: size of family is only d whereas unlimited supply of minxish functions

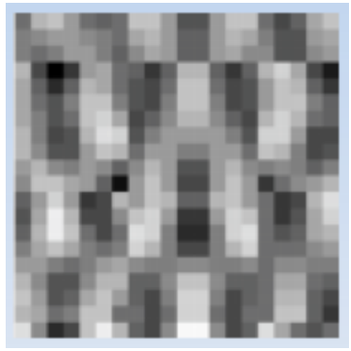
Experiment: Motorcycle Images

- 59,500 20x20 patches taken from motorcycle images
- Each image is represented as 400-dimensional column vectors
- Convert feature vectors into binary strings and use Hamming hash functions
- Denote B as the maximum search length

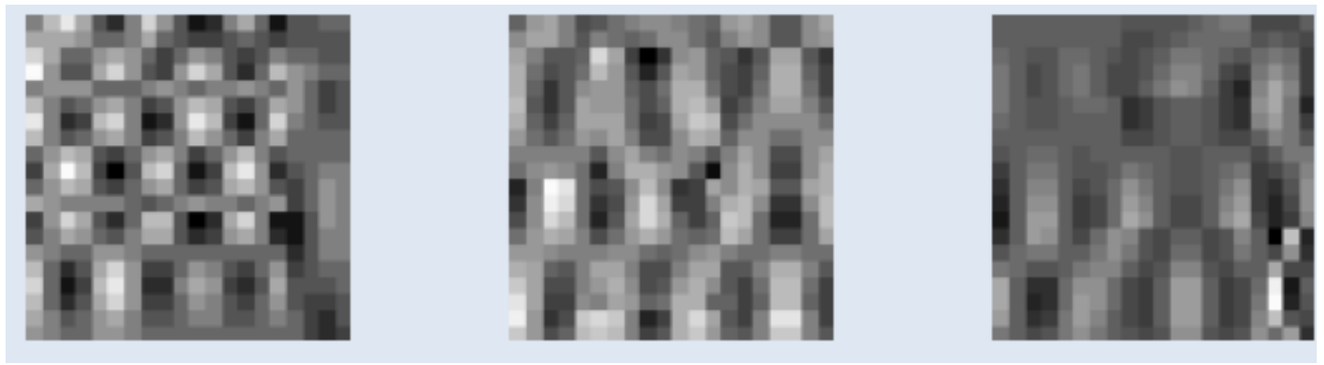


Experiment: Motorcycle Example Query

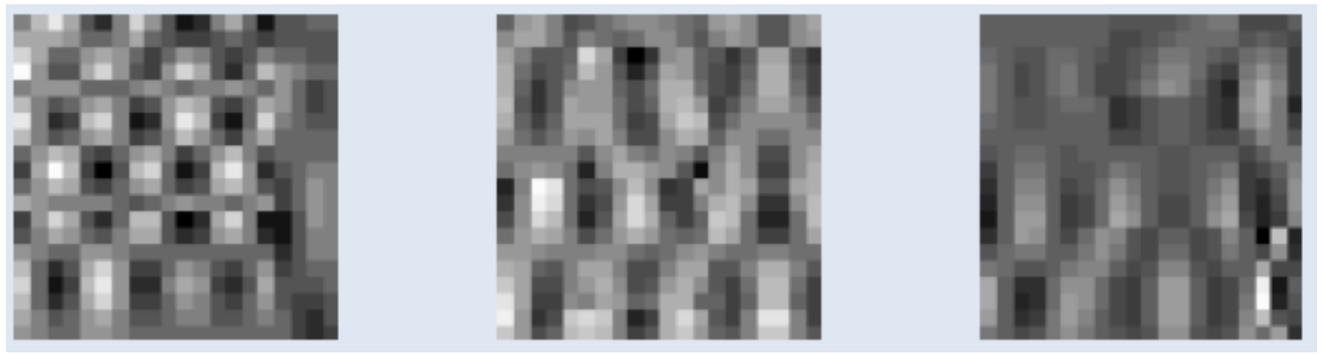
- L = 20, k = 24, B = infinity
- Query =



- Examples searched: 7,722 of 59,500
- Result =

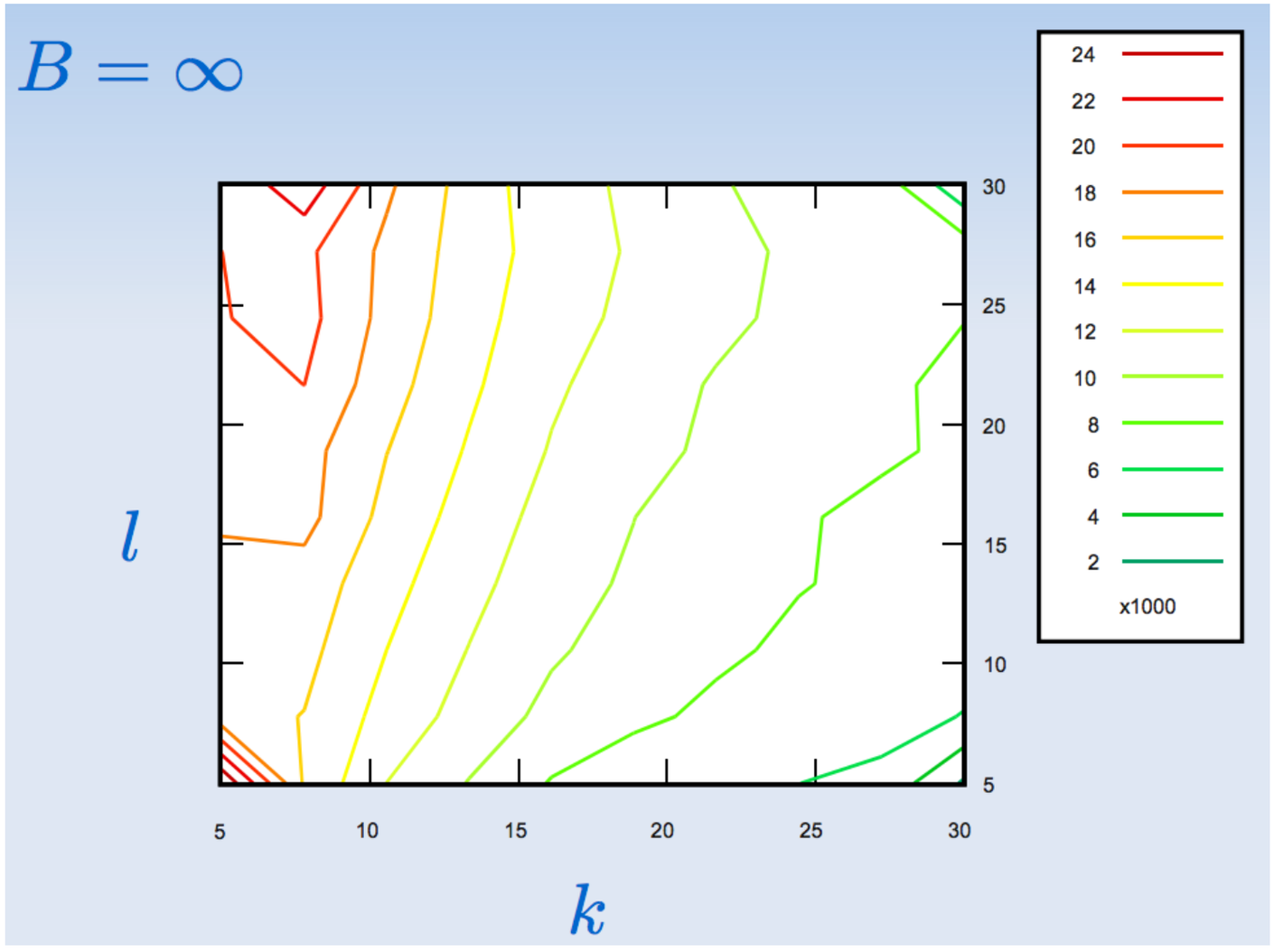


- Exact NN =



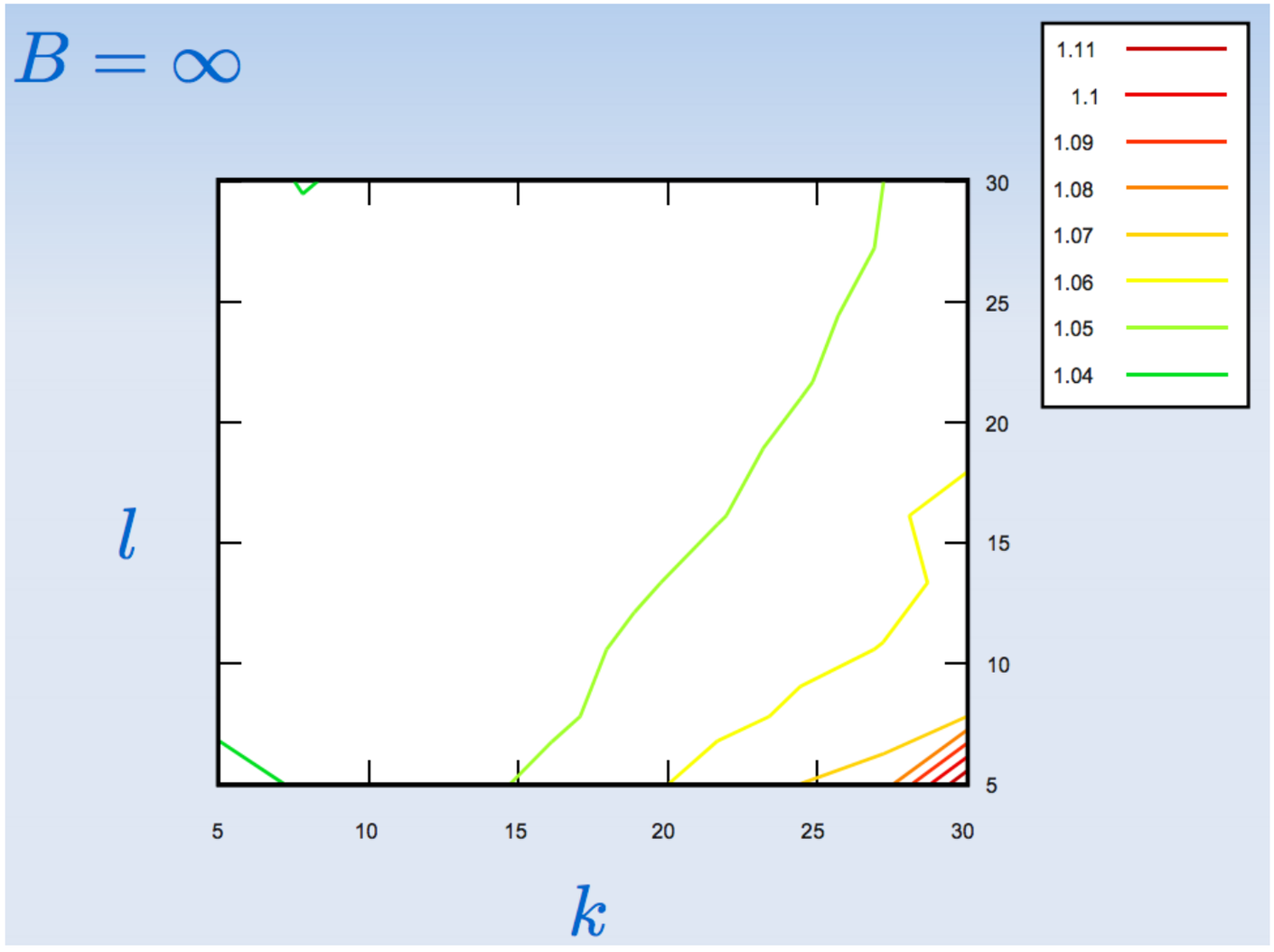
Experiment: Average Search Length

- More hash bits (k) result in shorter searches
- More hash tables (l) result in longer searches



Experiment: Average Approximation Error

- Over hashing (high k) can result in too few candidates to return a good approximation
- Over hashing can cause algorithm to fail



Experiment: Average Approximation Error (2)

- Changing the maximum number of searches requires more bits per hash function (k) and more hash tables (l)