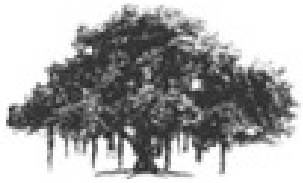


IIIT-H

Web Mining
Lecture 3: Similarity Search

Manish Gupta

7th Aug 2013

Slides borrowed (and modified) from
http://infolab.stanford.edu/~ullman/mining/2009/
http://cmp.felk.cvut.cz/~chum/papers/chum-minhash-BMVC08.ppt
http://infolab.stanford.edu/~ullman/mining/cs345-lsh.ppt

Recap of Lecture 2: Relevance Ranking

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

- Vector Space Model
  - Cosine similarity
  - Index Elimination
  - Champion Lists
  - Static Quality Scores
  - High and Low Lists
  - Impact-Ordered Postings
  - Cluster Pruning
  - Parametric and Zone Indexes
  - Tiered Indexes
  - Query Parsers
- Evaluation Measures
  - Precision
  - Recall
  - Prec@k
  - Interpolated Precision
  - AUC
  - MAP
  - NDCG
  - Kappa
  - A/B tests

Today's Agenda

- Shingling
- Min-hash
- Locally Sensitive Hashing (LSH)
- Applications of LSH

Today's Agenda

- **Shingling**
- Min-hash
- Locally Sensitive Hashing (LSH)
- Applications of LSH

Goals

- Many web-mining problems can be expressed as finding “similar” sets
  - Pages with similar words, e.g., for classification by topic.
  - NetFlix users with similar tastes in movies, for recommendation systems.
  - Movies with similar sets of fans.
  - Images of related things.

Example Problem: Comparing Documents

- Goal: common text, not common topic.
- Special cases are easy, e.g., identical documents, or one document contained character-by-character in another.
- General case, where many small pieces of one doc appear out of order in another, is very hard.

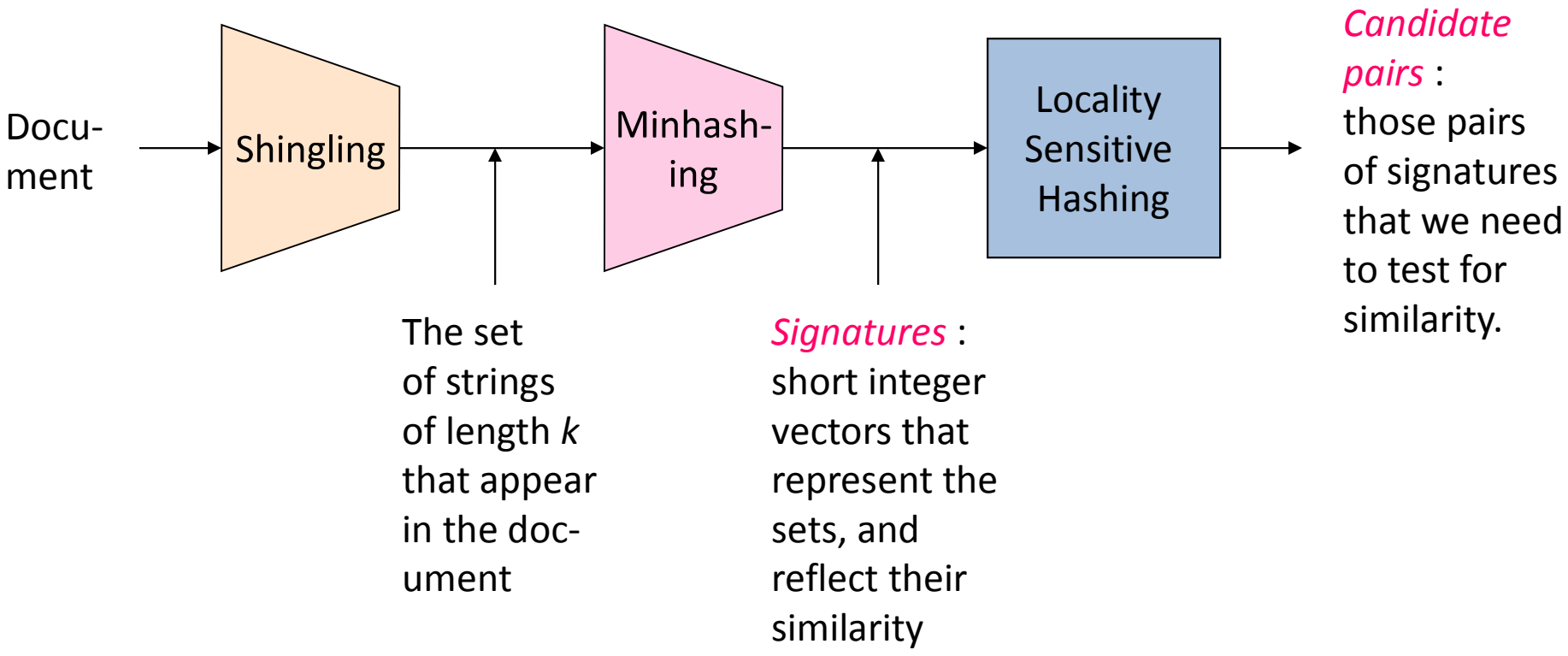
Similar Documents

- Given a body of documents, e.g., the Web, find pairs of documents with a lot of text in common, e.g.:
  - Mirror sites, or approximate mirrors.
    - Application: Don't want to show both in a search result.
  - Plagiarism, including large quotations.
  - Similar news articles at many news sites.
    - Application: Cluster articles by "same story."

Three Essential Techniques for Similar Documents

- *Shingling* : convert documents, emails, etc., to sets.
- *Minhashing* : convert large sets to short signatures, while preserving similarity.
- *Locally Sensitive Hashing (LSH):* Find pairs of documents (or signatures) likely to be similar

The Big Picture



Shingles

- A *k* -shingle (or *k* -gram) for a document is a sequence of *k* characters that appears in the document.
- Example: k=2; doc = abcab. Set of 2-shingles = {ab, bc, ca}.
  - Option: regard shingles as a bag, and count ab twice.
- Represent a doc by its set of *k*-shingles.

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; $k = 10$ is better for long documents.

Shingles: Compression Option

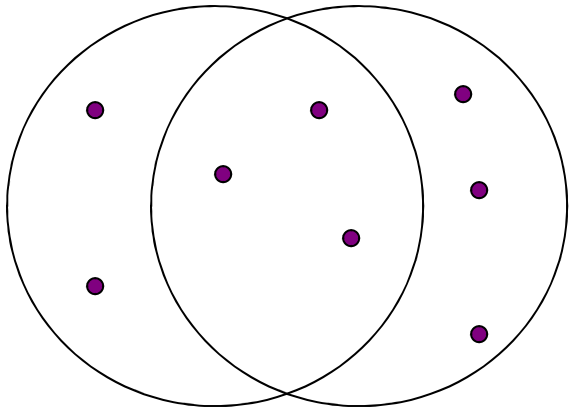
- To compress long shingles, we can hash them to (say) 4 bytes.
- Represent a doc by the set of hash values of its *k*-shingles.
- Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.

Today's Agenda

- Shingling
- **Min-hash**
- Locally Sensitive Hashing (LSH)
- Applications of LSH

Jaccard Similarity of Sets

- The *Jaccard similarity* of two sets is the size of their intersection divided by the size of their union.
  - *Sim* $(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$.



3 in intersection.
8 in union.
Jaccard similarity
= 3/8

From Sets to Boolean Matrices

- Rows = elements of the universal set.
- Columns = sets.
- 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 sets of their rows with 1.
- Typical matrix is sparse.

C_1	C_2
0	1
1	0
1	1
0	0
1	1
0	1

Sim (C_1, C_2) =2/5 = 0.4

Outline: Finding Similar Columns

1. Compute signatures of columns = small summaries of columns.
2. Examine pairs of signatures to find similar signatures.
   - Essential: similarities of signatures and columns are related.
3. Optional: check that columns with similar signatures are really similar.

Warnings

1. Comparing all pairs of signatures may take too much time, even if not too much space.
   - A job for Locality-Sensitive Hashing.
2. These methods can produce false negatives, and even false positives (if the optional check is not made).

Signatures

- Key idea: "hash" each column *C* to a small *signature* *Sig* (C), such that
  - *Sig* (C) is small enough that we can fit a signature in main memory for each column.
  - *Sim* (C_1, C_2) is the same as the "similarity" of *Sig* (C_1) and *Sig* (C_2).

Four Types of Rows

- Given columns C_1 and C_2, rows may be classified as:

	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

- Also, *a* = # rows of type *a* , etc.
- Note *Sim* (C_1, C_2) = *a* /(*a* +*b* +*c*).

Minhashing

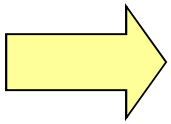
- Imagine the rows are permuted randomly.
- Define "hash" function h (C) = the number of the first (in the permuted order) row in which column C has 1.
- Use several (e.g., 100) independent hash functions to create a signature.

Minhashing Example

Input matrix

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

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

Surprising Property

- The probability (over all permutations of the rows) that $h(C_1) = h(C_2)$ is the same as $Sim(C_1, C_2)$.
- Both are $a/(a+b+c)$!
- Why?
  - Look down the permuted columns C_1 and C_2 until we see a 1.
  - If it's a type-*a* row, then $h(C_1) = h(C_2)$. If a type-*b* or type-*c* row, then not.

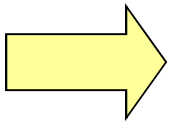
Min Hashing – Example

- The *similarity of signatures* is the fraction of the hash functions in which they agree.

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

Input matrix

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

Minhash Signatures

- Pick (say) 100 random permutations of the rows.
- Think of *Sig* (C) as a column vector.
- Let *Sig* (C)[i] =

  according to the *i* th permutation, the number of the first row that has a 1 in column *C*.

Implementation – (1)

- Suppose 1 billion rows.
- Hard to pick a random permutation from 1…billion.
- Representing a random permutation requires 1 billion entries.
- Accessing rows in permuted order leads to thrashing.

Implementation – (2)

- A good approximation to permuting rows: pick 100 hash functions.
- For each column c and each hash function h_i, keep a "slot" $M(i, c)$.
- Intent: $M(i, c)$ will become the smallest value of $h_i(r)$ for which column c has 1 in row r.
  - I.e., $h_i(r)$ gives order of rows for i th permuation.

Implementation – (3)

for each row *r*

  for each column *c*

    if c has 1 in row *r*

      for each hash function h_i **do**

        if $h_i(r)$ is a smaller value than $M(i, c)$ **then**

          $M(i, c) := h_i(r)$;

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$

$g(x) = 2x+1 \bmod 5$

	Sig1	Sig2
$h(1) = 1$	1	-
$g(1) = 3$	3	-
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

Finding Similar Pairs

- Suppose we have, in main memory, data representing a large number of objects.
  - May be the objects themselves.
  - May be signatures as in minhashing.
- We want to compare each to each, finding those pairs that are sufficiently similar.

Checking All Pairs is Hard

- While the signatures of all columns may fit in main memory, comparing the signatures of all pairs of columns is quadratic in the number of columns.
- Example: 10^6 columns implies $5*10^{11}$ column-comparisons.
- At 1 microsecond/comparison: 6 days.

Today's Agenda

- Shingling
- Min-hash
- **Locally Sensitive Hashing (LSH)**
- Applications of LSH

Locality-Sensitive Hashing

- General idea: Use a function f(x,y) that tells whether or not *x* and *y* is a *candidate pair* : a pair of elements whose similarity must be evaluated.
- For minhash matrices: Hash columns to many buckets, and make elements of the same bucket candidate pairs.

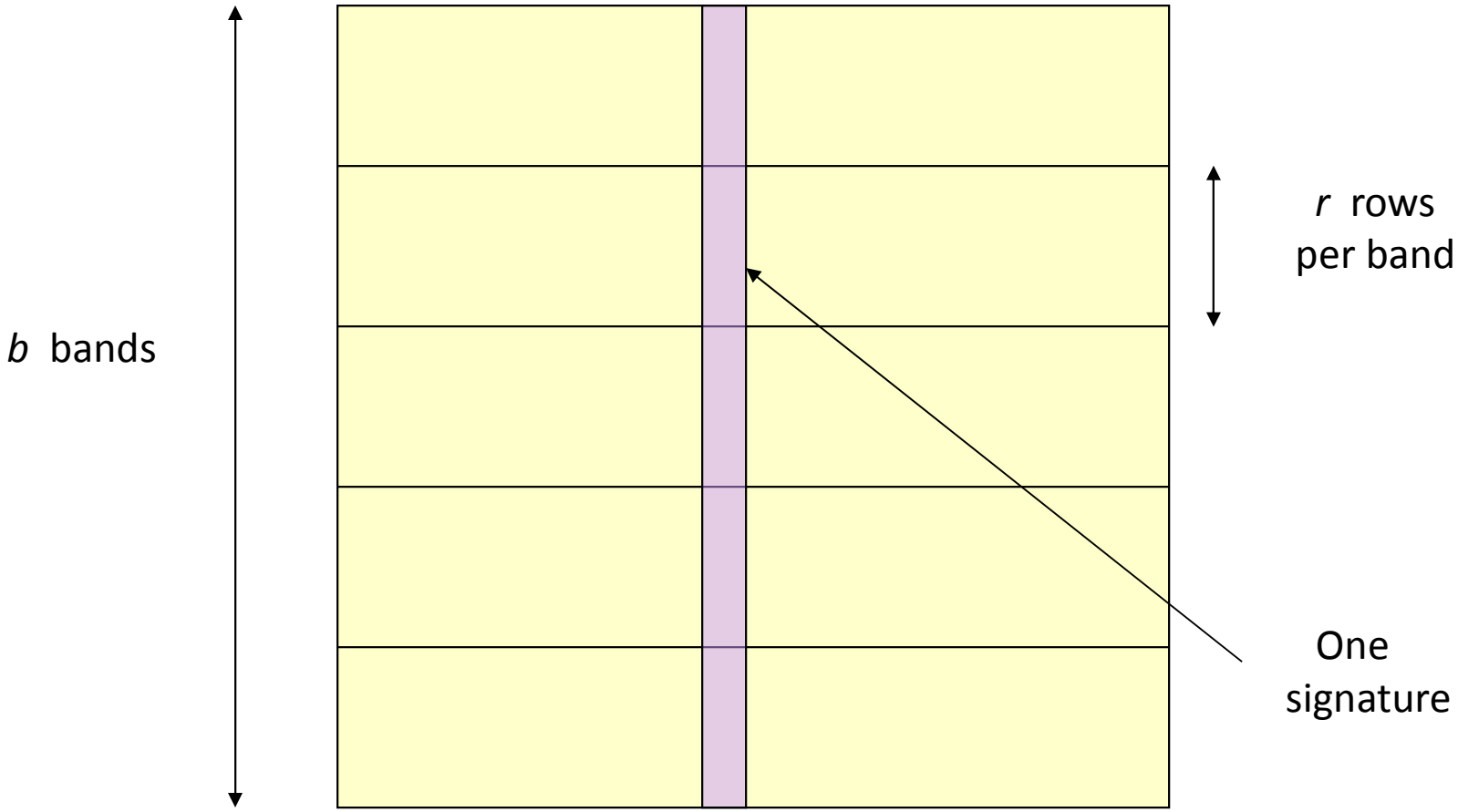
Candidate Generation From Minhash Signatures

- Pick a similarity threshold *s*, a fraction < 1.
- A pair of columns *c* and *d* is a *candidate pair* if their signatures agree in at least fraction *s* of the rows.
  - I.e., $M(i, c) = M(i, d)$ for at least fraction *s* values of *i*.

LSH for Minhash Signatures

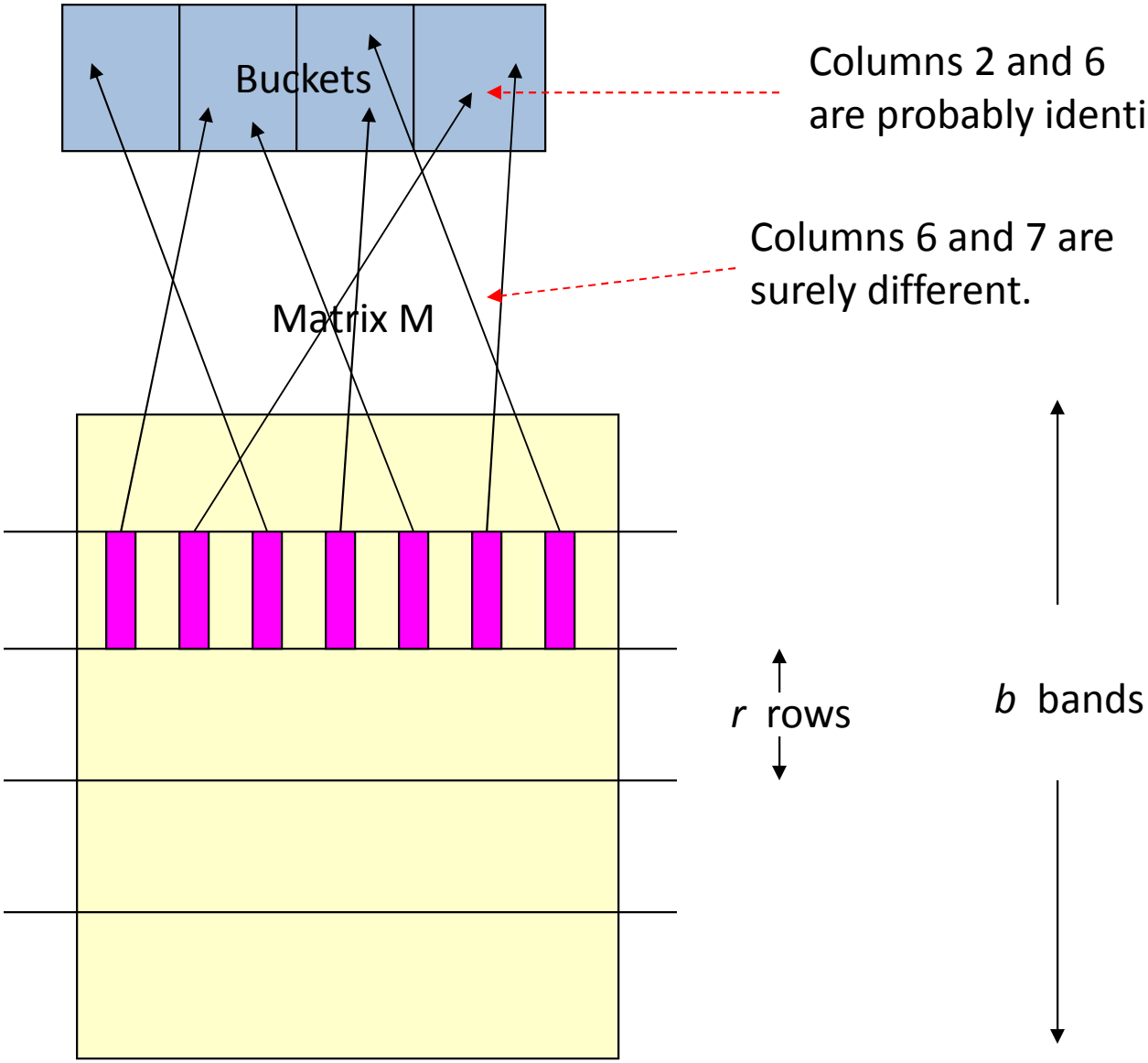
- Big idea: hash columns of signature matrix *M* several times.
- Ensure that (only) similar columns are likely to hash to the same bucket.
- Candidate pairs are those that hash at least once to the same bucket.

Partition Into Bands



Partition into Bands – (2)

- 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.
- *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 nonsimilar pairs.


Buckets
Columns 2 and 6
are probably identical.
Columns 6 and 7 are
surely different.
Matrix M
r rows
b 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."

Example: Effect of Bands

- Suppose 100,000 columns.
- Signatures of 100 integers.
- Therefore, signatures take 40Mb.
- Want all 80%-similar pairs.
- 5,000,000,000 pairs of signatures can take a while to compare.
- Choose 20 bands of 5 integers/band.

Suppose C_1, C_2 are 80% Similar

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$.
- Probability C_1, C_2 are *not* similar in any of the 20 bands: $(1\text{-}0.328)^{20} = .00035$.
  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives.

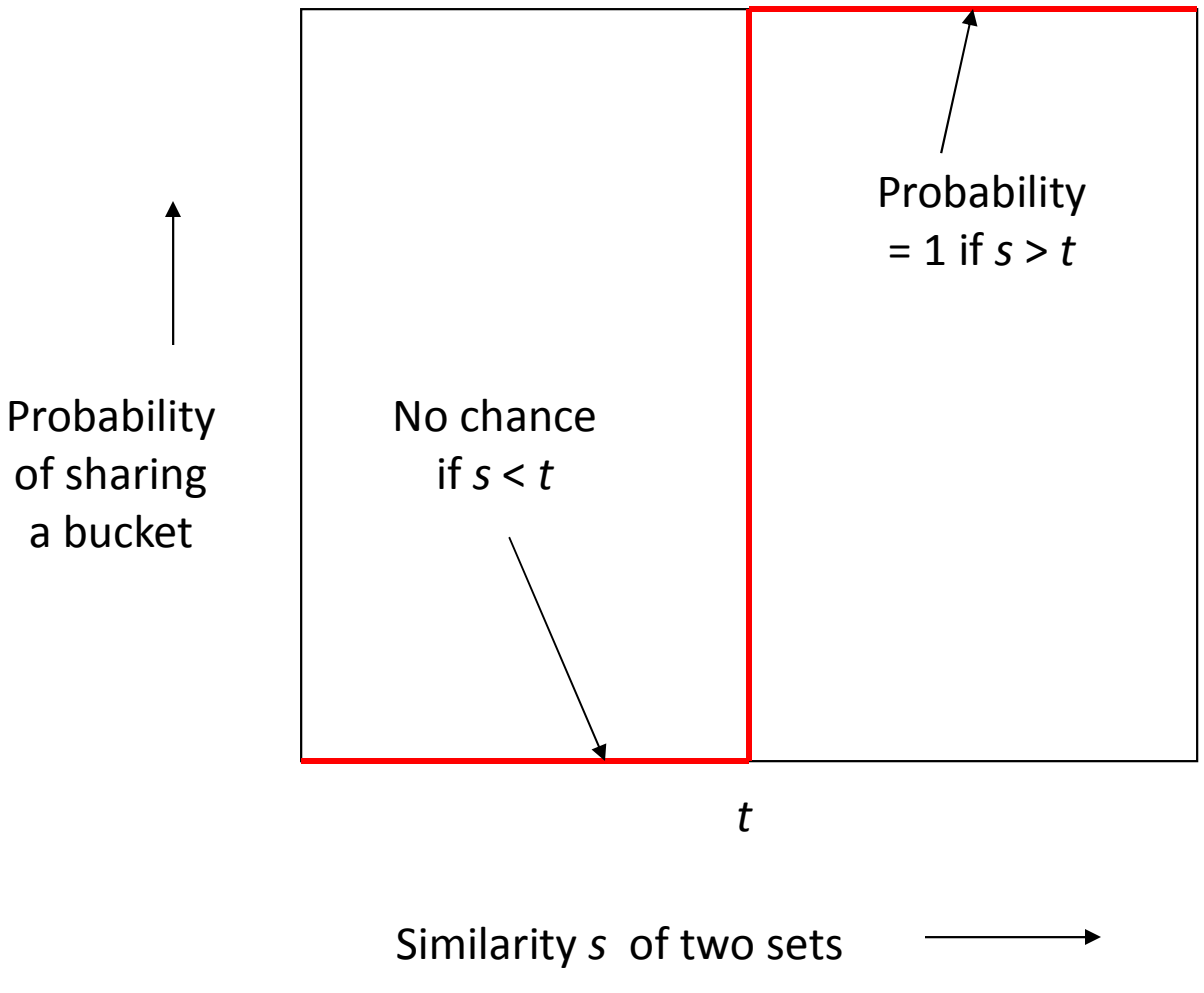
Suppose C_1, C_2 Only 40% Similar

- Probability C_1, C_2 identical in any one particular band: $(0.4)^5 = 0.01$.
- Probability C_1, C_2 identical in $\geq$ 1 of 20 bands: $\leq 20 * 0.01 = 0.2$.
- But false positives much lower for similarities << 40%.

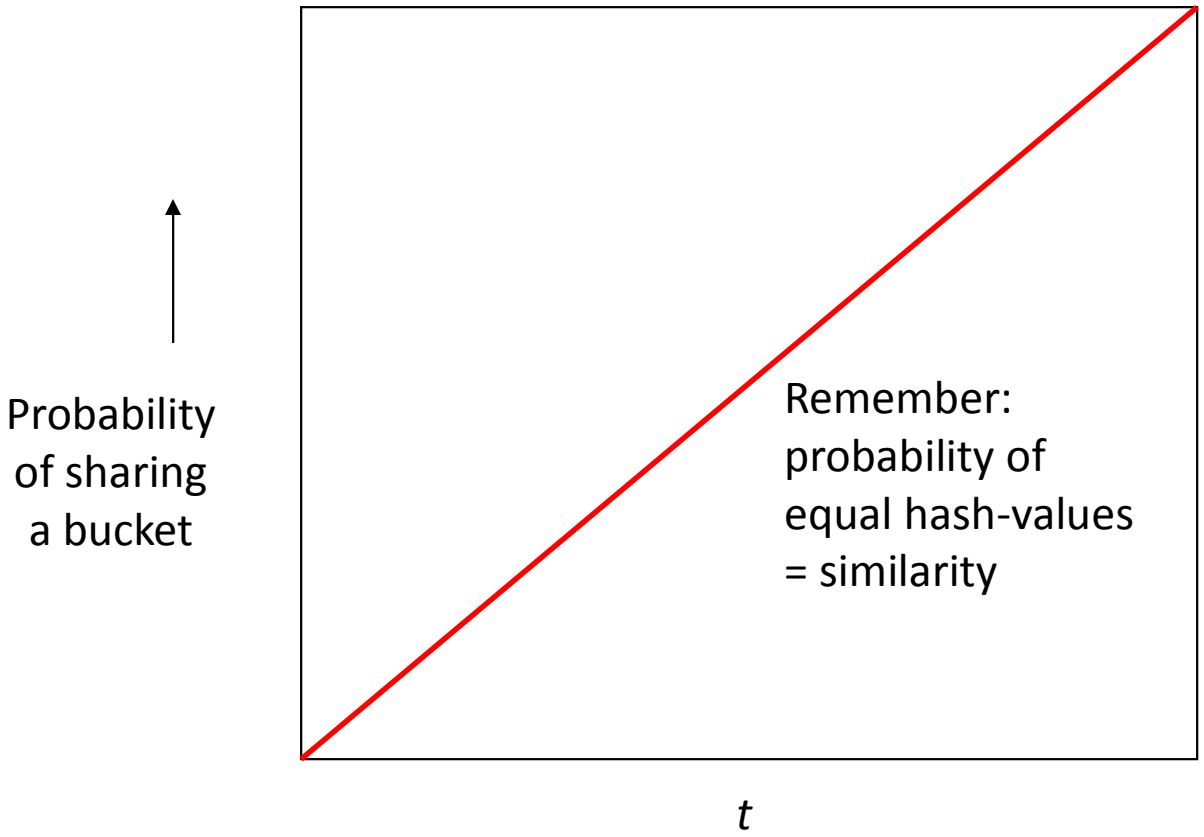
LSH Involves a Tradeoff

- Pick the number of minhashes, the number of bands, and the number of rows per band to balance false positives/negatives.
- Example: if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up.

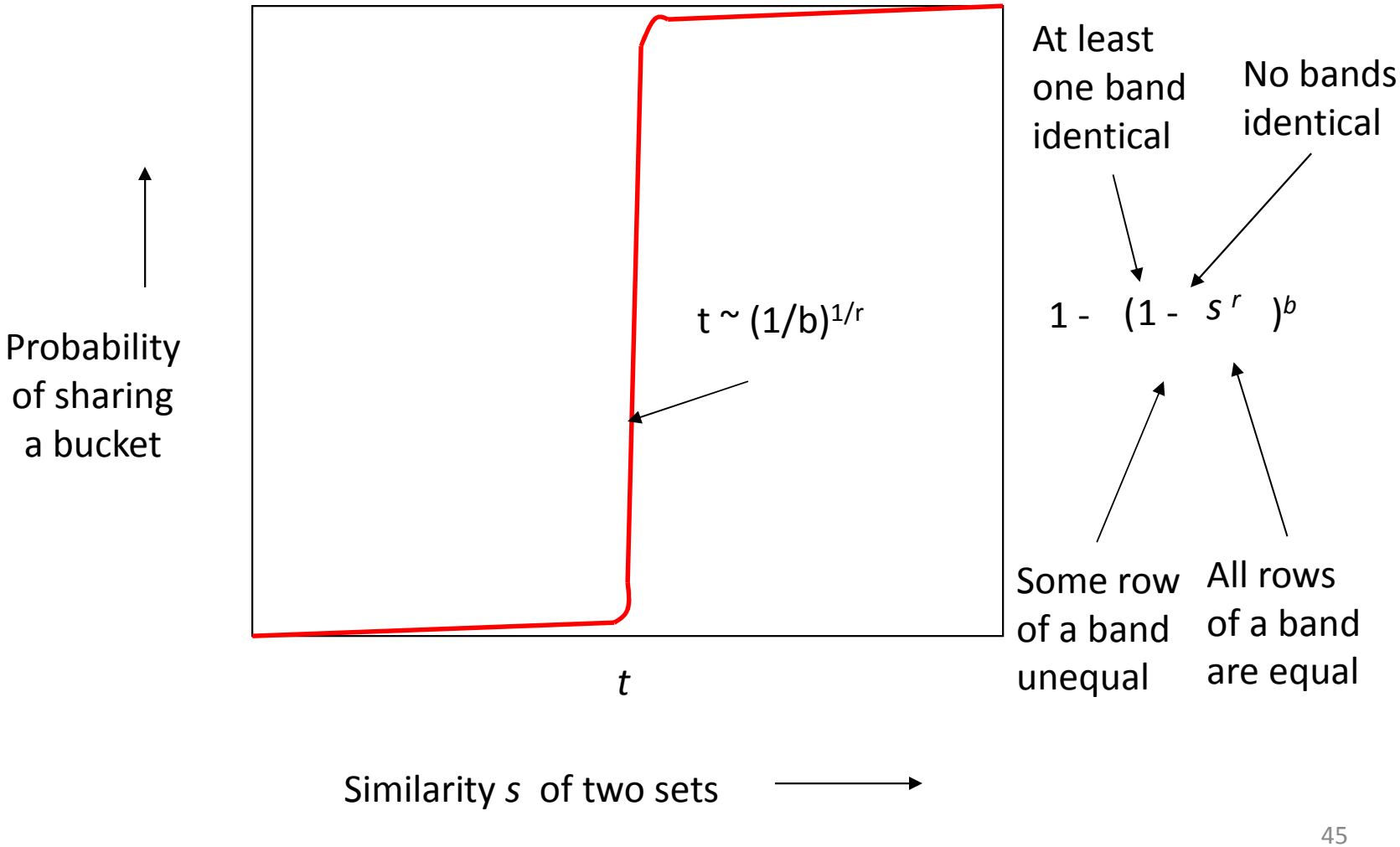
Analysis of LSH – What We Want



What One Band of One Row Gives You



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



Example: $b = 20$; $r = 5$

s	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

Analysis of the Banding Technique

- The probability that the signatures agree in all rows of one particular band is s^r.
- The probability that the signatures do not agree in at least one row of a particular band is $1 - s^r$.
- The probability that the signatures do not agree in all rows of any of the bands is $(1 - s^r)^b$.
- The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

LSH Summary

- Tune to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures.
- 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 *sets*.

Today's Agenda

- Shingling
- Min-hash
- Locally Sensitive Hashing (LSH)
- **Applications of LSH**

Entity Resolution

- The *entity-resolution* problem is to examine a collection of records and determine which refer to the same entity.
  - *Entities* could be people, events, etc.
- Typically, we want to merge records if their values in corresponding fields are similar.

Customer Records – (1)

- Step 1: Design a measure ("*score* ") of how similar records are:
  - E.g., deduct points for small misspellings ("Jeffrey" vs. "Jeffery") or same phone with different area code.
- Step 2: Score all pairs of records; report high scores as matches.

Customer Records – (2)

- Problem: (1 million)^2 is too many pairs of records to score.
- Solution: A simple LSH.
  - Three hash functions: exact values of name, address, phone.
    - Compare iff records are identical in at least one.
  - Misses similar records with a small differences in all three fields.

Duplicate News Articles

- Problem: the same article, say from the Associated Press, appears on the Web site of many newspapers, but looks quite different.
- Each newspaper surrounds the text of the article with:
  - It's own logo and text.
  - Ads.
  - Perhaps links to other articles.
- A newspaper may also "crop" the article (delete parts).

New Shingling Technique

- News articles have a lot of stop words, while ads do not.
  - "Buy Sudzo" vs. "I recommend that you buy Sudzo for your laundry."
- Define a *shingle* to be a stop word and the next two following words.

Why it Works

- By requiring each shingle to have a stop word, the mapping gets biased from documents to shingles so it picked more shingles from the article than from the ads.
- Pages with the same article, but different ads, have higher Jaccard similarity than those with the same ads, different articles.

Take-away Messages

- Computing similarity between two sets of objects is a core functionality needed by multiple components of a “web mining system”
- We studied a technique for similarity search for dataset with large number of objects (“big data”). This involves three steps: shingling, minhashing and locality sensitive hashing

Further Reading

- Chapter 3: "Finding Similar Items" from Mining of Massive Datasets
  - http://infolab.stanford.edu/~ullman/mmds.html
- Chapter 3 (Web Search and Information Retrieval) from Mining the Web
  - http://www.cse.iitb.ac.in/soumen/mining-the-web/
- http://en.wikipedia.org/wiki/MinHash#CITEREFBroder1997
- http://en.wikipedia.org/wiki/Locality_sensitive_hashing

Preview of Lecture 4: Link Analysis Algorithms

- PageRank
- HITS (Hypertext-Induced Topic Selection)
- Topic-Specific Page Rank
- Spam Detection Algorithms: TrustRank

Disclaimers

- This course represents opinions of the instructor only. It does not reflect views of Microsoft or any other entity (except of authors from whom the slides have been borrowed).
- Algorithms, techniques, features, etc mentioned here might or might not be in use by Microsoft or any other company.
- Lot of material covered in this course is borrowed from slides across many universities and conference tutorials. These are gratefully acknowledged.

Thanks!

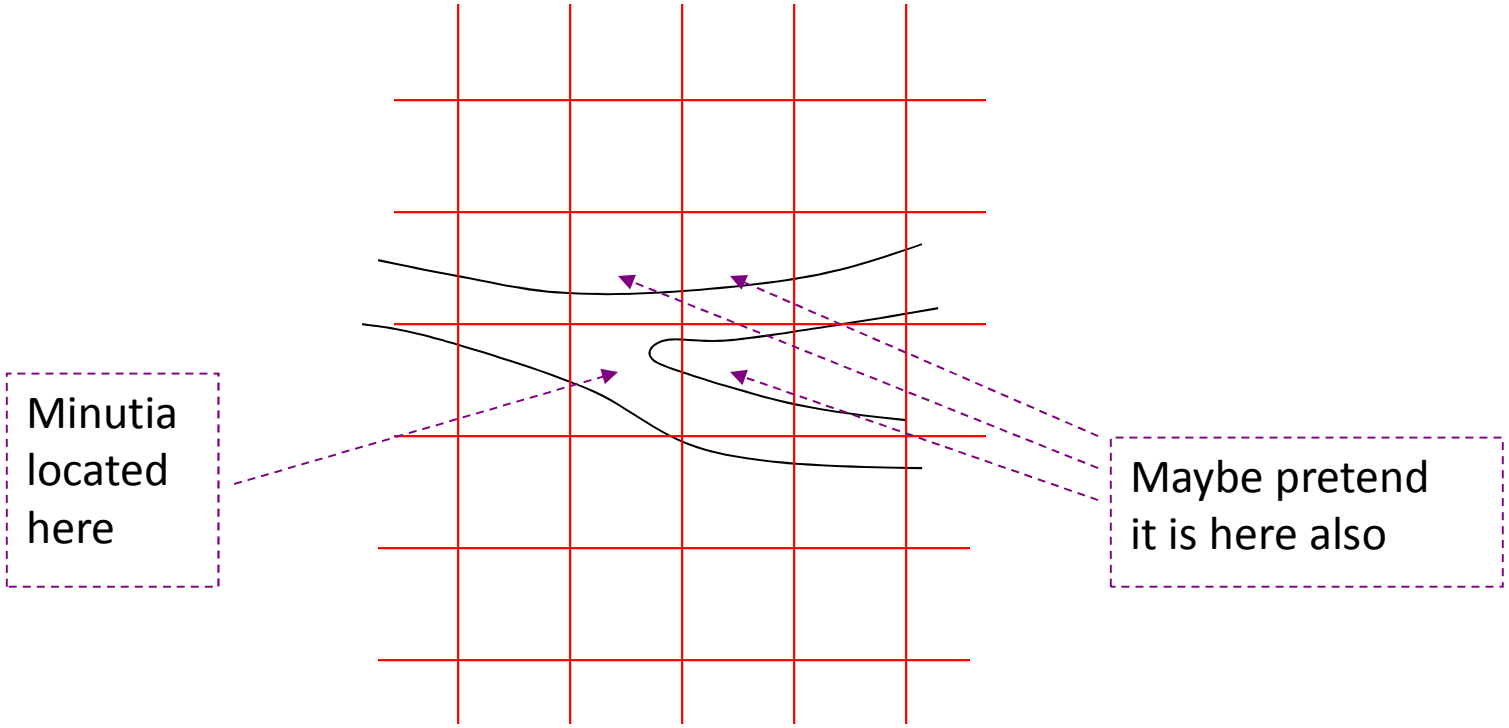
Fingerprint Comparison

- Represent a fingerprint by the set of positions of *minutiae*.
  - These are features of a fingerprint, e.g., points where two ridges come together or a ridge ends.

LSH for Fingerprints

- Place a grid on a fingerprint.
  - Normalize so identical prints will overlap.
- Set of grid points where minutiae are located represents the fingerprint.
  - Possibly, treat minutiae near a grid boundary as if also present in adjacent grid points.

Discretizing Minutiae



Applying LSH to Fingerprints

- Make a bit vector for each fingerprint's set of grid points with minutiae.
- We could minhash the bit vectors to obtain signatures.
- But since there probably aren't too many grid points, we can work from the bit-vectors directly.

LSH/Fingerprints – (2)

- Pick 1024 sets of 3 grid points, randomly.
- For each set of points, fingerprints with 1 for all three points are candidate pairs.
- Suppose typical fingerprints have minutiae in 20% of the grid points.
- Suppose fingerprints from the same finger agree in at least 80% of their points.
- Probability two random fingerprints each have 1 in all three points = $(0.2)^6 = .000064$.

Example: Continued

First image has 1 in a point

Second image of same finger also has 1.

- Probability two fingerprints from the same finger each have 1's in three given points = $((0.2)(0.8))^3 = .004096$.
- Prob. for at least one of 1024 sets of three points = $1-(1-.004096)^{1024} = .985$.
- But for random fingerprints: $1-(1-.000064)^{1024} = .063$.

1.5% false negatives

6.3% false positives

Alternative Interpretation

- Given 2 sets A and B, we want to estimate Jaccard(A,B), i.e. $J(A,B)$.
- A,B $\subset$ {1,2,…,n}
- Let π be a random permutation from {1,2,…,n} to {1,2,…,n} (i.e., one of the n! permutations)
- $P(\min\{\pi(A)\} = \min\{\pi(B)\}) = J(A,B)$
- Given enough permutations, can estimate $J(A,B)$ accurately
- The proof follows on the next slide

Estimating Jaccard using Minhash

- What is $P(\min\{\pi(A)\} = \min\{\pi(B)\})$
- What fraction of n! permutations satisfy $\min(\pi(A)) = \min(\pi(B))$?
- If $x = \min(\pi(A)) = \min(\pi(B))$, then $\pi^{-1}(x)$ must belong to $A \cap B$
- Pick the range to which A ∪ B is mapped in $\binom{n}{|A \cup B|}$ ways
- Remaining $n - |A \cup B|$ elements not chosen may be permuted every way: $(n - |A \cup B|)!$
- Element to be mapped to the minimum value in the range can be chosen in $|A \cap B|$ ways
- Remaining elements in $A \cup B$ can be permuted in $(|A \cup B| - 1)!$ ways
- $\binom{n}{|A \cup B|} (n - |A \cup B|)! \, |A \cap B| \, (|A \cup B| - 1)! = \frac{|A \cap B|}{|A \cup B|} n!$

Word Weighting for min-Hash

- For minhash over word sets, the hash functions π should be chosen such that each word has an equal chance of being a minhash.
  - Such a scheme helps minhash result into Jaccard similarity
  - $P(\min\{\pi(A)\} = \min\{\pi(B)\}) = \frac{|A \cap B|}{|A \cup B|}$
- For minhash over weighted word sets, i.e., when each word w has a weight d_w
  - Hash functions h should be chosen such that the probability of word w being a minhash is proportional to d_w
  - $P(\min\{\pi(A)\} = \min\{\pi(B)\}) = \frac{\sum_{w \in A \cap B} d_w}{\sum_{w \in A \cup B} d_w}$

Thought Question

- Why is it better to hash 9-shingles (say) to 4 bytes than to use 4-shingles?
- Hint: How random are the 32-bit sequences that result from 4-shingling?

LS Families of Hash Functions

- Suppose we have a space *S* of points with a distance measure *d*.
- 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 $d(x,y) \leq d_1$, then prob. over all *h* in **H**, that $h(x) = h(y)$ is at least p_1.
  2. If $d(x,y) \geq d_2$, then prob. over all *h* in **H**, that $h(x) = h(y)$ is at most p_2.

LSH for other Distance Functions

- Let *S* = sets, *d* = Jaccard distance, **H** is formed from the minhash functions for all permutations. For Jaccard similarity, minhashing gives us a $(d_1,d_2,(1-d_1),(1-d_2))$-sensitive family for any $d_1 < d_2$.
  - Prob[h(x)=h(y)] = 1-d(x,y).
- d=Cosine Distance. Random hyperplanes or random projections. $(d_1,d_2,(1-d_1/180),(1-d_2/180))$- sensitive family
  - If x is data and v is random hyperplane, $h_v(x)$ = +1 if v.x > 0; = -1 if v.x < 0
  - Prob[h(x)=h(y)] = 1 – (angle between *x* and *y* divided by 180).
- D=Euclidean distance.
  - Hash functions are derived by choosing random lines and projecting points onto those lines.
  - Each line is broken into fixed-length intervals, and the function answers “yes” to a pair of points that fall into the same interval.