Finding Similar Sets

Applications

Shingling

Minhashing

Locality-Sensitive Hashing

Distance Measures

Modified from Jeff Ullman

Goals

- Many Web-mining problems can be expressed as finding "similar" sets:
  1. Pages with similar words, e.g., for classification by topic.
  2. NetFlix users with similar tastes in movies, for recommendation systems.
  3. Dual: movies with similar sets of fans.
  4. Images of related things.

Similarity Algorithms

- The best techniques depend on whether you are looking for items that are very similar or only somewhat similar.
- We'll cover the "somewhat" case.

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.

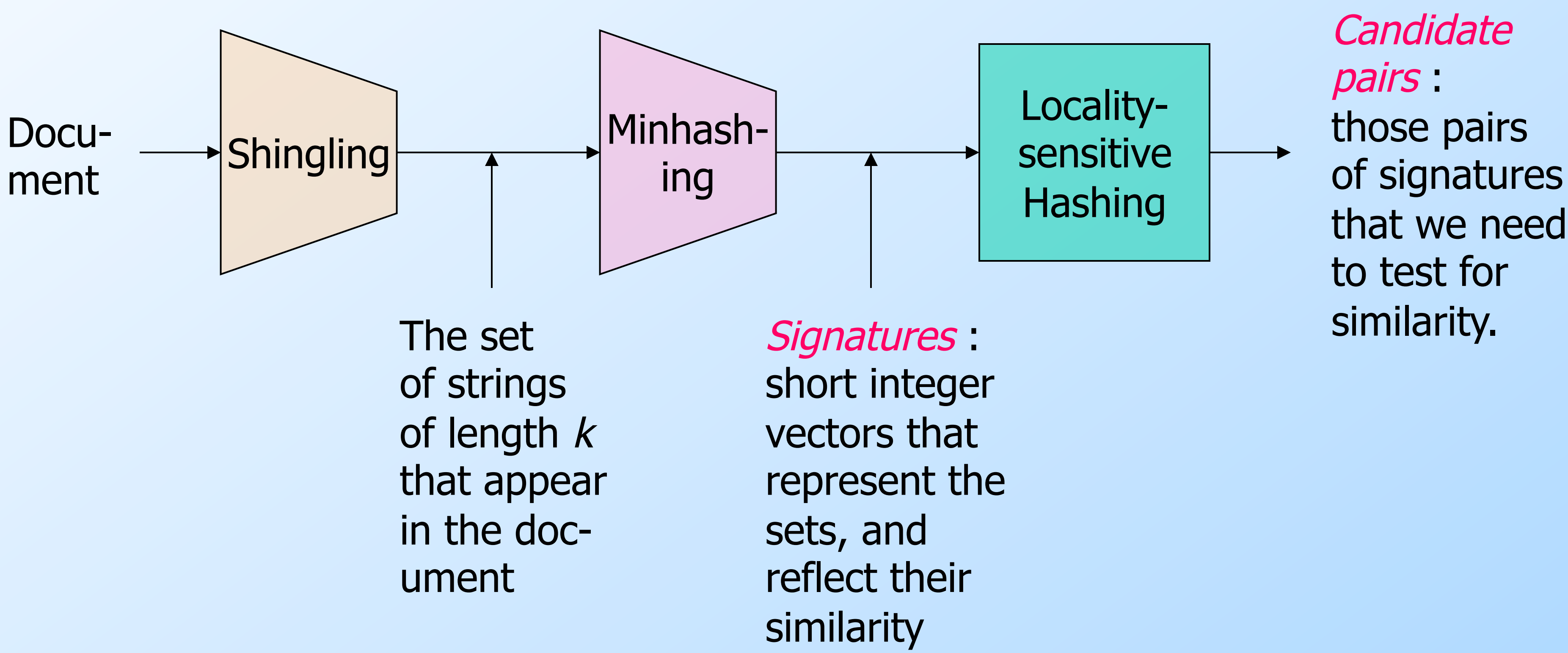
Similar Documents – (2)

- 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.
  - Plagiarism, including large quotations.
  - Similar news articles at many news sites.
    - Application: Cluster articles by "same story."

Three Essential Techniques for Similar Documents

1. *Shingling* : convert documents, emails, etc., to sets.
2. *Minhashing* : convert large sets to short signatures, while preserving similarity.
3. *Locality-sensitive hashing* : focus on pairs of 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.

Shingle Size

- Is k=2 a good choice for size?
- Example: k=2;
- doc1 = abcab. 2-shingles = {ab, bc, ca}.
- doc2 = cabc. 2-shingles = {ab, bc, ca}.

Shingle Size

- 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

- If k=9, to compare shingles we need to compare 9 bytes
- To improve efficiency, we can compress long shingles: hash them to (say) 4 bytes, and
- Represent a doc by the set of hash values of its *k*-shingles.

(aaabbbccc)(abcabcabc)→h(aaabbbccc)h(abcabcabc)

18 bytes → 8 bytes

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?

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?
- Assuming 20 characters are common in English, there are $(20)^4 = 160{,}000$ 4-shingles
- Using 9 shingles there are $(20)^9 >> 2^{32}$

MinHashing

Data as Sparse Matrices

Jaccard Similarity Measure

Constructing Signatures

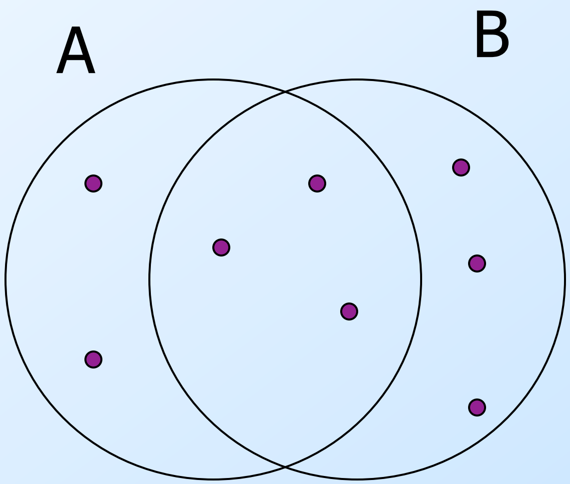
Basic Data Model: Sets

- Many similarity problems can be couched as finding subsets of some universal set that have significant intersection.
- Examples include:
  1. Documents represented by their sets of shingles (or hashes of those shingles).
  2. Similar customers or products.

Jaccard Similarity of Sets

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

Example: Jaccard Similarity



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.

Example: Jaccard Similarity of Columns

	C_1	C_2		
a	0	1		*
b	1	0		*
c	1	1	*	*
d	0	0		
e	1	1	*	*
f	0	1		*

$\text{Sim}(C_1, C_2) = \ 2/5 = 0.4$

Aside

- We might not really represent the data by a boolean matrix.
- Sparse matrices are usually better represented by the list of places where there is a non-zero value.
- But the matrix picture is conceptually useful.

When Is Similarity Interesting?

1. When the sets are so large or so many that they cannot fit in main memory.
2. Or, when there are so many sets that comparing all pairs of sets takes too much time.
3. Or both.

Outline: Finding Similar Columns

1. Compute signatures of columns = small summaries of columns.
2. Examine pairs of signatures to find similar signatures.
   - Requirement: 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:
  1. *Sig* (C) is small enough that we can fit a signature in main memory for each column.
  2. *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	*1 in both columns*
b	1	0	*columns are different*
c	0	1	
d	0	0	*0 in both columns*

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

Minhashing

- History: invented by Andrei Broder in 1997 to detect duplicate pages
- Imagine the rows 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

Permutations

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

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 $a/(a+b+c)$!
- Matrix is sparse – most rows are of type d
- The ratio of type a, b, and c that determine the similarity and the probability that $h(C_1) = h(C_2)$

Surprising Property

- Probability that $h(C_1) = h(C_2)$ is the same as $Sim(C_1, C_2) = a/(a+b+c)$!
- 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.

Similarity for Signatures

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

Min Hashing – Example

Input matrix

1	4	3	1	0	1	0
3	2	4	1	0	0	1
7	1	7	0	1	0	1
6	3	6	0	1	0	1
2	6	1	0	1	0	1
5	7	2	1	0	1	0
4	5	5	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.
- Sorting would take a long time
- Representing a random permutation requires 1 billion entries.

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 permutation.

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

Implementation – (4)

- Often, data is given by column, not row.
  - E.g., columns = documents, rows = shingles.
- If so, sort matrix once so it is by row.
- And *always* compute $h_i(r)$ only once for each row.

Locality-Sensitive Hashing

Focusing on Similar Minhash Signatures

Other Applications Will Follow

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 (10^6 2) =~ $5*10^{11}$ column-comparisons. $\frac{n!}{k!\,(n-k)!}$
- At 1 microsecond/comparison: 6 days.

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.

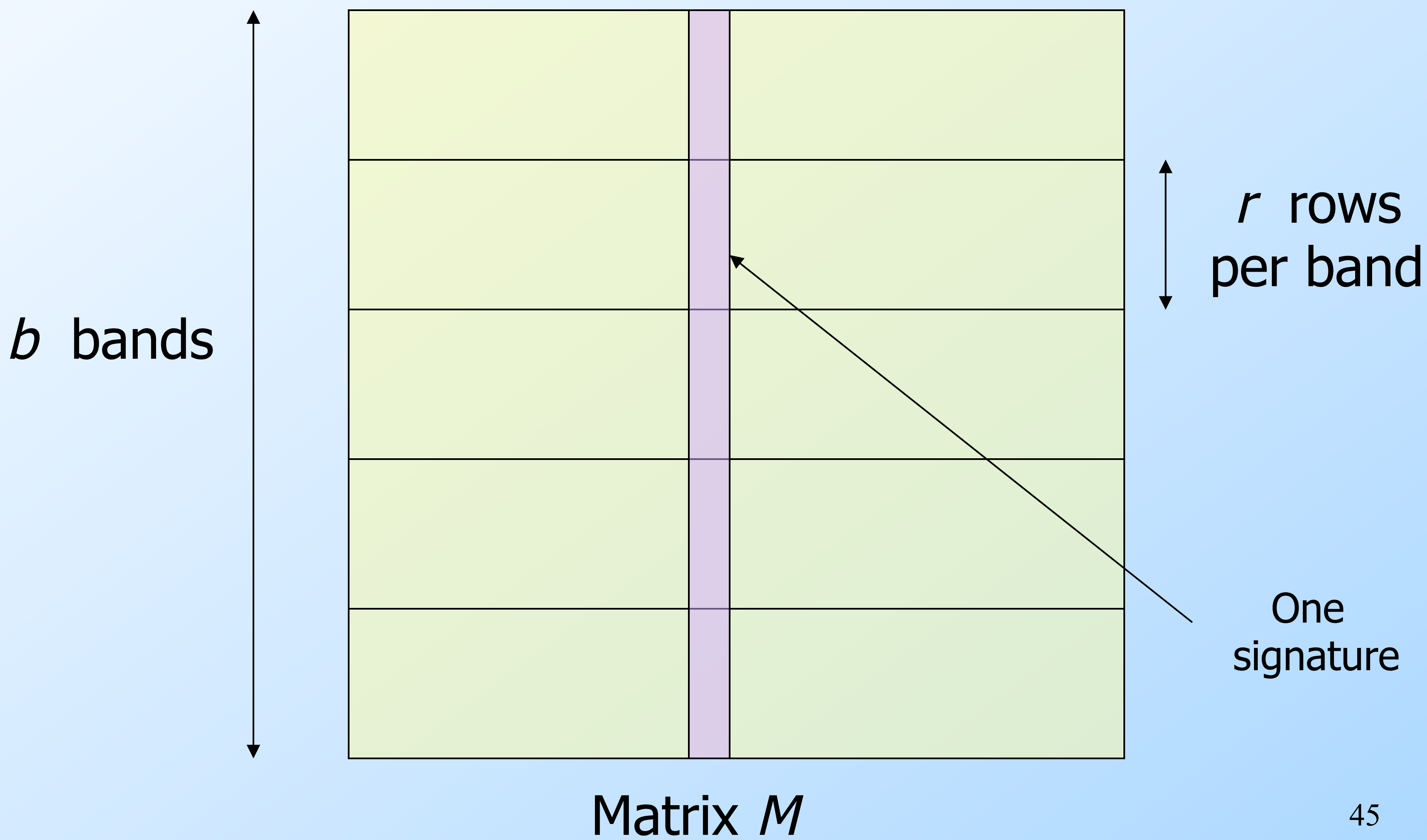
Candidate Generation From Minhash Signatures

- 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

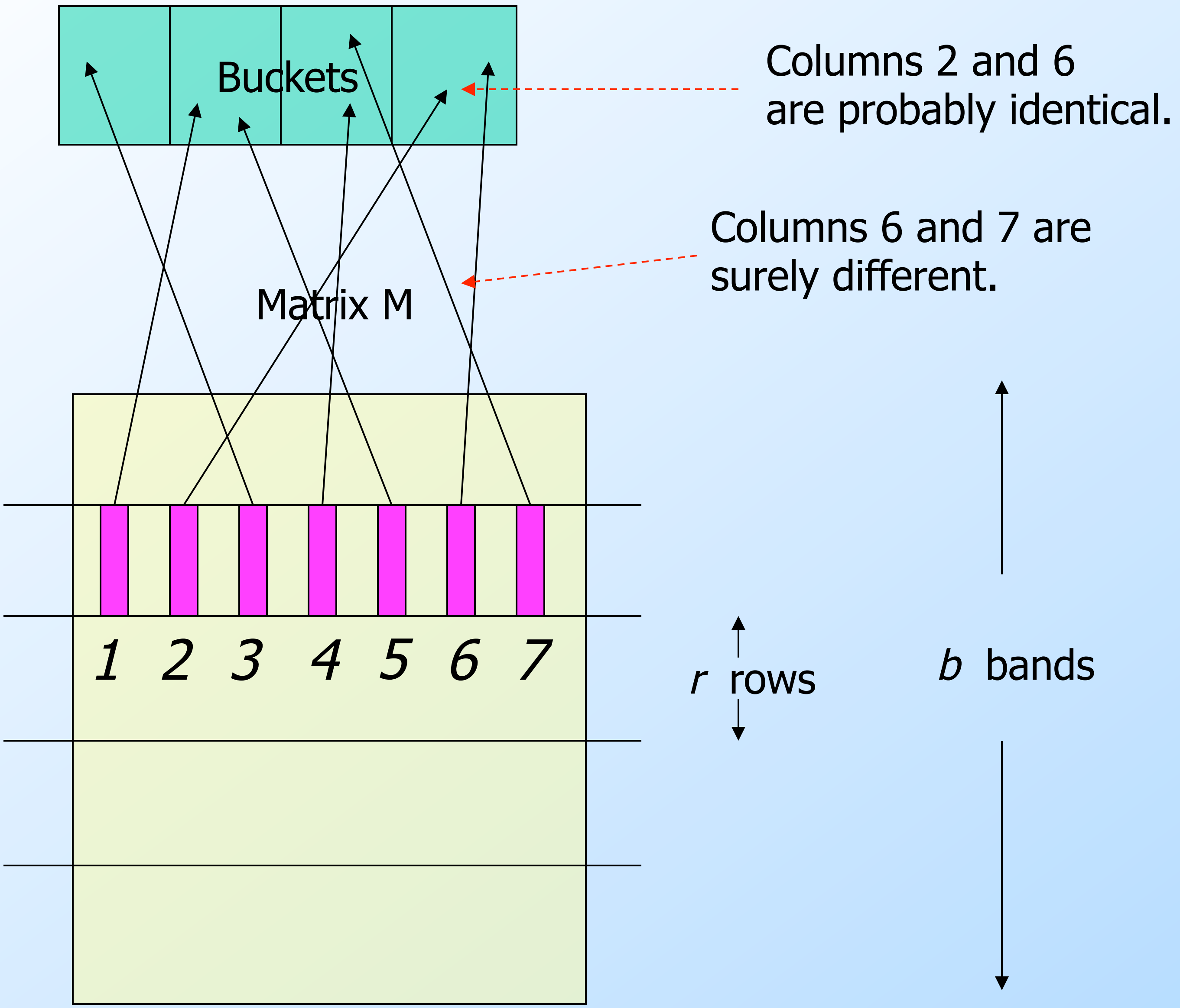
- Big idea: hash columns of signature matrix M several times.
- Arrange that (only) similar columns are likely to hash to the same bucket.
- Candidate pairs are those that hash at least once to the same bucket.

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 $\geq$ 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
1 2 3 4 5 6 7
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: $(1-0.328)^{20} = .00035$.
  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives.

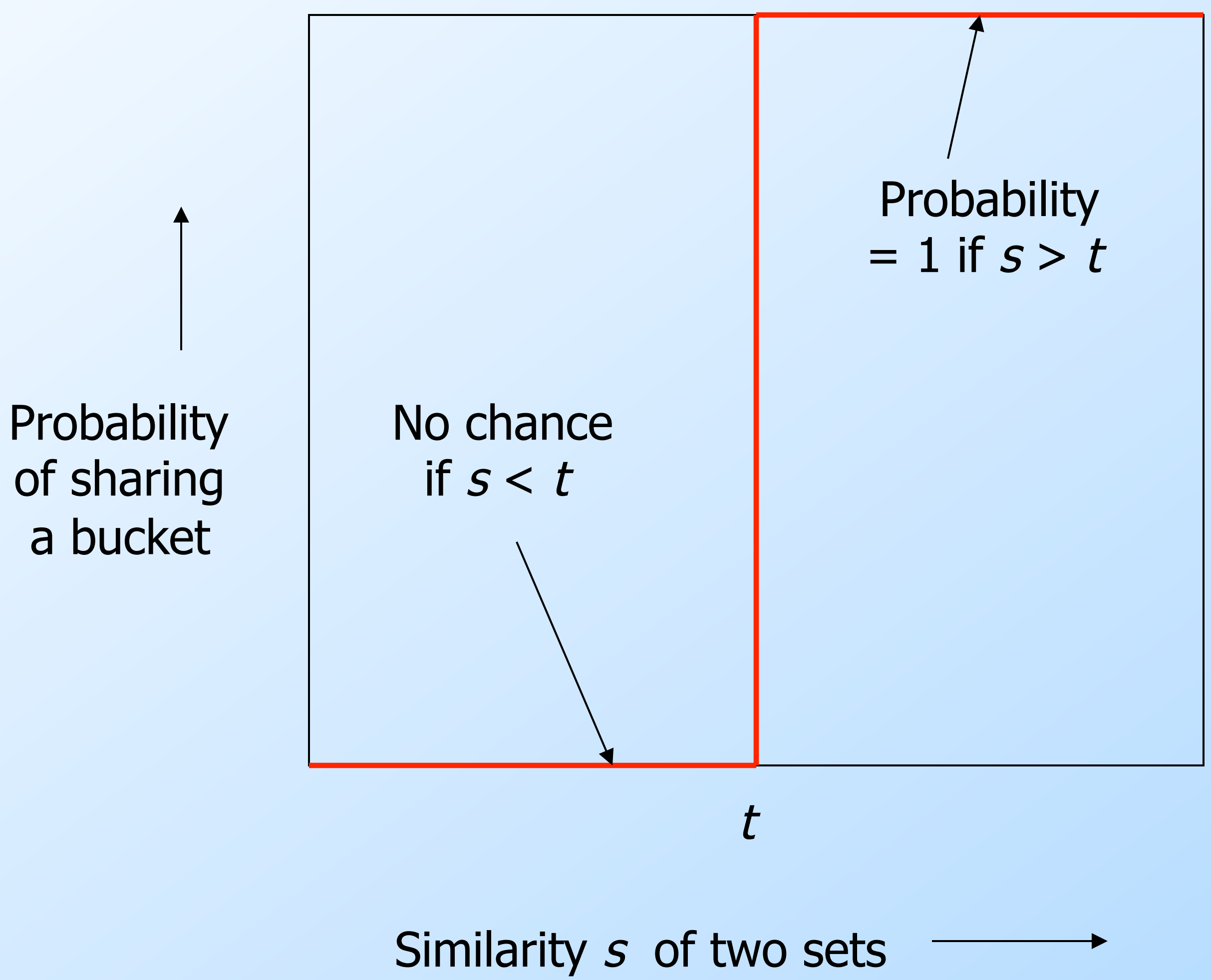
Suppose C_1, C_2 Only 40% Similar

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

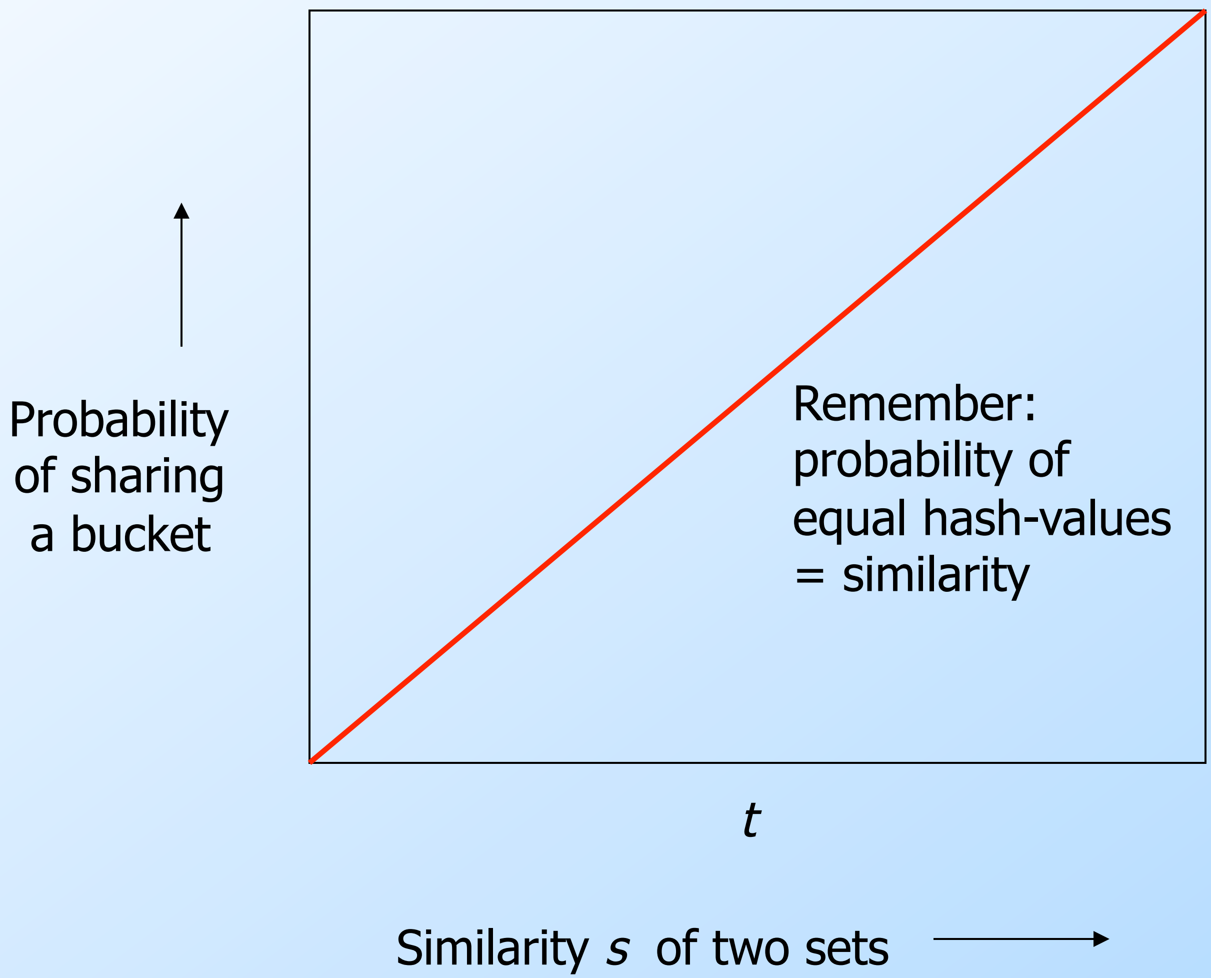
LSH Involves a Tradeoff

- 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.

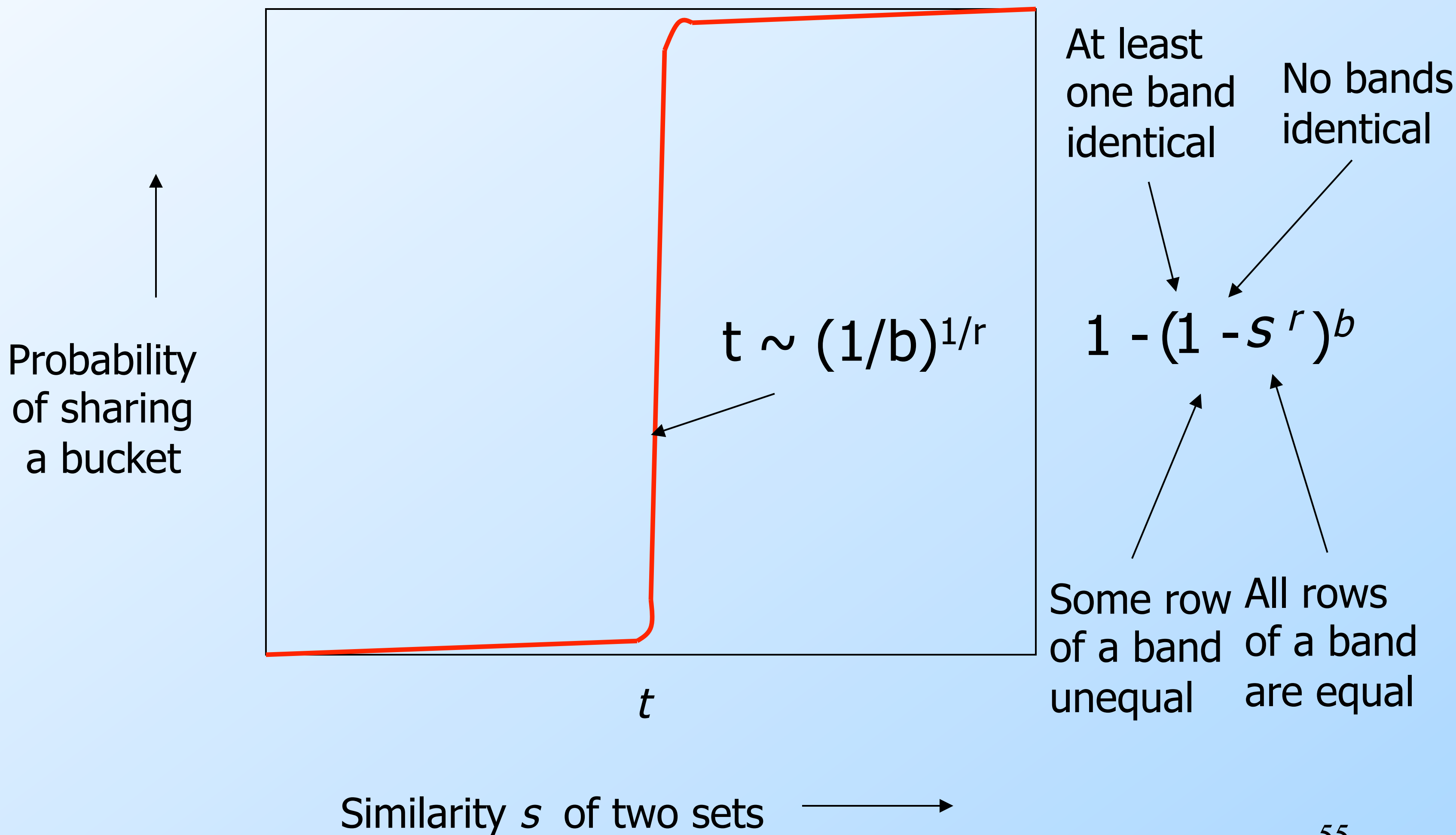
Analysis of LSH – What We Want



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

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* .

Applications of LSH

Entity Resolution

Similar News Articles

Desiderata

- Whatever form we use for LSH, we want :
  1. The time spent performing the LSH should be linear in the number of objects.
  2. The number of candidate pairs should be proportional to the number of truly similar pairs.
- Bucketizing guarantees (1).

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.

Matching Customer Records

- I once took a consulting job solving the following problem:
  - Company A agreed to solicit customers for Company B, for a fee.
  - They then argued over how many customers.
  - Neither recorded exactly which customers were involved.

Customer Records – (2)

- Company B had about 1 million records of all its customers.
- Company A had about 1 million records describing customers, some of whom it had signed up for B.
- Records had name, address, and phone, but for various reasons, they could be different for the same person.

Customer Records – (3)

- 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 – (4)

- 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.

Aside: Hashing Names, Etc.

- How do we hash strings such as names so there is one bucket for each string?
- Possibility: Sort the strings instead.
  - Used in this story.
- Possibility: Hash to a few million buckets, and deal with buckets that contain several different strings.

Aside: Validation of Results

- We were able to tell what values of the scoring function were reliable in an interesting way.
  - Identical records had a creation date difference of 10 days.
  - We only looked for records created within 90 days, so bogus matches had a 45-day average.

Validation – (2)

- By looking at the pool of matches with a fixed score, we could compute the average time-difference, say x, and deduce that fraction (45-x)/35 of them were valid matches.
- Alas, the lawyers didn't think the jury would understand.

Validation – Generalized

- Any field not used in the LSH could have been used to validate, provided corresponding values were closer for true matches than false.
- Example: if records had a height field, we would expect true matches to be close, false matches to have the average difference for random people.

Application: Same News Article

- Recently, the Political Science Dept. asked a team from CS to help them with the problem of identifying duplicate, on-line news articles.
- Problem: the same article, say from the Associated Press, appears on the Web site of many newspapers, but looks quite different.

News Articles – (2)

- 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).

News Articles – (3)

- The team came up with its own solution, that included shingling, but not minhashing or LSH.
  - A special way of shingling that appears quite good for this application.
  - LSH substitute: candidates are articles of similar length.

Enter LSH – (1)

- I told them the story of minhashing + LSH.
- They implemented it and found it faster for similarities below 80%.
  - Aside: That's no surprise. When similarity is high, there are better methods, as we shall see.

Enter LSH – (2)

- Their first attempt at LSH was very inefficient.
- They were unaware of the importance of doing the minhashing row-by-row.
- Since their data was column-by-column, they needed to sort once before minhashing.

New Shingling Technique

- The team observed that news articles have a lot of stop words, while ads do not.
  - "Buy Sudzo" vs. "I recommend that you buy Sudzo for your laundry."
- They defined 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, they biased the mapping 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.

Distance Measures

Distance Measures

- Generalized LSH is based on some kind of "distance" between points.
  - Similar points are "close."
- 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.
- A *Non-Euclidean distance* is based on properties of points, but not their “location” in a space.

Axioms of a Distance Measure

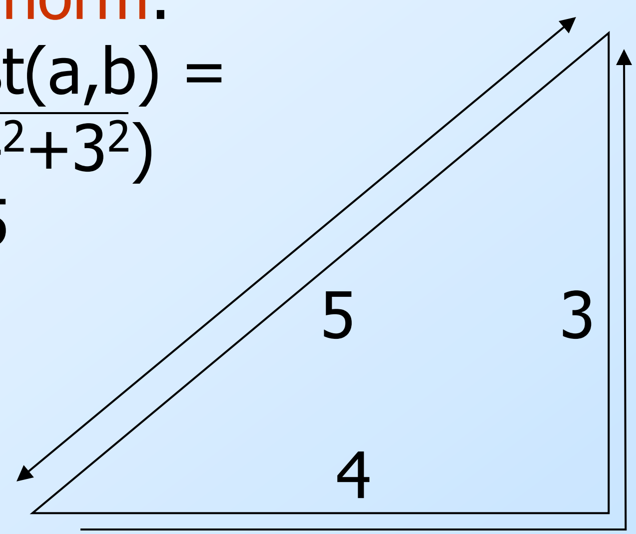
- *d* is a *distance measure* if it is a function from pairs of points to real numbers such that:
  1. $d(x,y) \geq 0$.
  2. $d(x,y) = 0$ iff $x = y$.
  3. $d(x,y) = d(y,x)$.
  4. $d(x,y) \leq d(x,z) + d(z,y)$ (*triangle inequality*).

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.

Examples of Euclidean Distances

L_2-norm:
dist(a,b) =
$\sqrt{(4^2+3^2)}$
= 5



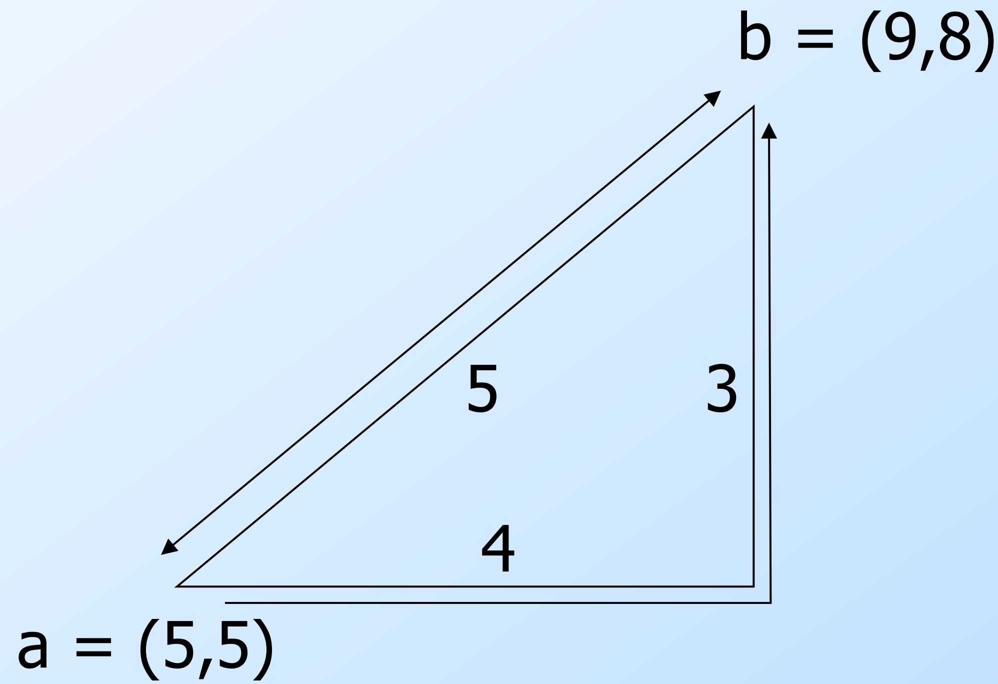
L_1-norm:

dist(a,b) =
4+3 = 7

Another Euclidean Distance

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

Examples of Euclidean Distances



What is the L_∞-norm for a and b?

L_∞-norm:
Max(|9-5|,|8-5|) =
Max(4,3) = 4

Non-Euclidean Distances

- *Jaccard distance* for sets = 1 minus Jaccard similarity.
- *Cosine distance* = angle between vectors from the origin to the points in question.
- *Edit distance* = number of inserts and deletes to change one string into another.
- *Hamming Distance* = number of positions in which bit vectors differ.

Jaccard Distance for Sets (Bit-Vectors)

- Example: $p_1 = 10111$; $p_2 = 10011$.
- Size of intersection = 3; size of union = 4, Jaccard similarity (not distance) = 3/4.
- $d(x,y) = 1 - (\text{Jaccard similarity}) = 1/4$.

Why J.D. Is a Distance Measure

- $d(x,x) = 0$ because $x \cap x = x \cup x$.
- $d(x,y) = d(y,x)$ because union and intersection are symmetric.
- $d(x,y) \geq 0$ because $|x \cap y| \leq |x \cup y|$.
- $d(x,y) \leq d(x,z) + d(z,y)$ trickier (see textbook)

Triangle Inequality for J.D.

$$1 - \frac{|x \cap z|}{|x \cup z|} + 1 - \frac{|y \cap z|}{|y \cup z|} \geq 1 - \frac{|x \cap y|}{|x \cup y|}$$

- Remember: $|a \cap b|/|a \cup b|$ = probability that minhash(a) = minhash(b).
- Thus, $1 - |a \cap b|/|a \cup b|$ = probability that minhash(a) $\neq$ minhash(b).

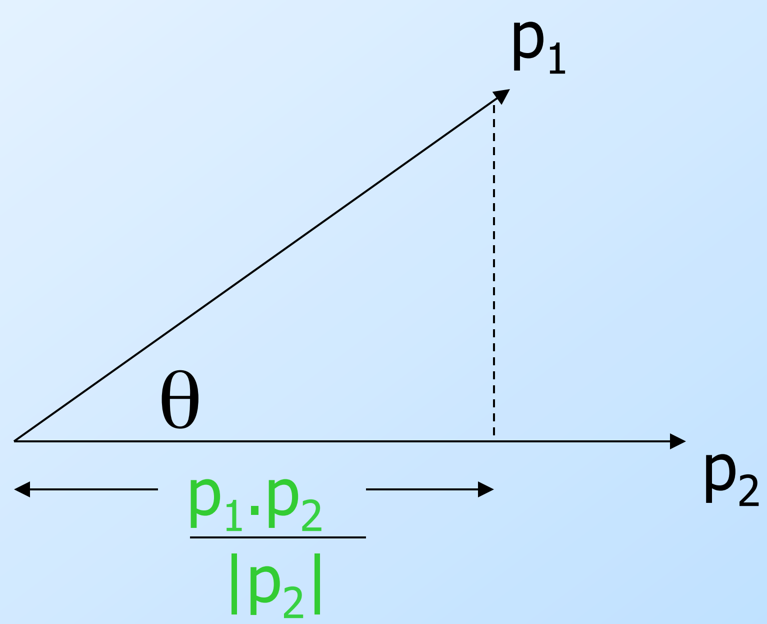
Triangle Inequality – (2)

- Claim: $\text{prob}[\text{minhash}(x) \neq \text{minhash}(y)] \leq \text{prob}[\text{minhash}(x) \neq \text{minhash}(z)] + \text{prob}[\text{minhash}(z) \neq \text{minhash}(y)]$
- Proof: whenever $\text{minhash}(x) \neq \text{minhash}(y)$, at least one of $\text{minhash}(x) \neq \text{minhash}(z)$ and $\text{minhash}(z) \neq \text{minhash}(y)$ must be true.

Cosine Distance

- Think of a point as a vector from the origin (0,0,…,0) to its location.
- Two points' vectors make an angle, whose cosine is the normalized dot-product of the vectors: $p_1.p_2/|p_2||p_1|$.
  - Example: $p_1 = 00111$; $p_2 = 10011$.
  - $p_1.p_2 = 2$; $|p_1| = |p_2| = \sqrt{3}$.
  - $\cos(\theta) = 2/3$; θ is about 48 degrees.

Cosine-Measure Diagram



$d\ (p_1, p_2) = \theta = \arccos(p_1 . p_2 / |p_2||p_1|)$

Why C.D. Is a Distance Measure

- d(x,x) = 0 because arccos(1) = 0.
- d(x,y) = d(y,x) by symmetry.
- d(x,y) $\geq$ 0 because angles are chosen to be in the range 0 to 180 degrees.
- Triangle inequality: physical reasoning. If I rotate an angle from *x* to *z* and then from *z* to *y*, I can't rotate less than from *x* to *y*.

Edit Distance

- The *edit distance* of two strings is the number of inserts and deletes of characters needed to turn one into the other. Equivalently:
- $d(x,y) = |x| + |y| - 2|LCS(x,y)|$.
  - LCS = *longest common subsequence* = any longest string obtained both by deleting from *x* and deleting from *y*.

Example: LCS

- x = *abcde* ; y = *bcduve*.
- Turn x into y by deleting *a*, then inserting *u* and *v* after *d*.
  - Edit distance = 3.
- Or, LCS(x,y) = *bcde*.
- Note: |x| + |y| - 2|LCS(x,y)| = 5 + 6 –2*4 = 3 = edit distance.

Why Edit Distance Is a Distance Measure

- d(x,x) = 0 because 0 edits suffice.
- d(x,y) = d(y,x) because insert/delete are inverses of each other.
- d(x,y) $\geq$ 0: no notion of negative edits.
- Triangle inequality: changing *x* to *z* and then to *y* is one way to change *x* to *y*.

Variant Edit Distances

- Allow insert, delete, and *mutate*.
  - Change one character into another.
- Minimum number of inserts, deletes, and mutates also forms a distance measure.
- Ditto for any set of operations on strings.
  - Example: substring reversal OK for DNA sequences

Hamming Distance

- *Hamming distance* is the number of positions in which bit-vectors differ.
- Example: p_1 = 10101; p_2 = 10011.
- $d(p_1, p_2) = 2$ because the bit-vectors differ in the 3rd and 4th positions.

Why Hamming Distance Is a Distance Measure

- d(x,x) = 0 since no positions differ.
- d(x,y) = d(y,x) by symmetry of "different from."
- d(x,y) $\geq$ 0 since strings cannot differ in a negative number of positions.
- Triangle inequality: changing *x* to *z* and then to *y* is one way to change *x* to *y*.