Locality-Sensitive Hashing

Flavius Frasincar

- Slides are based on the course material of Mining Massive Datasets by Jure Leskovic, Anand Rajaraman, and Jeffrey David Ullman (http://www.mmds.org/)

Finding Similar Sets

Applications

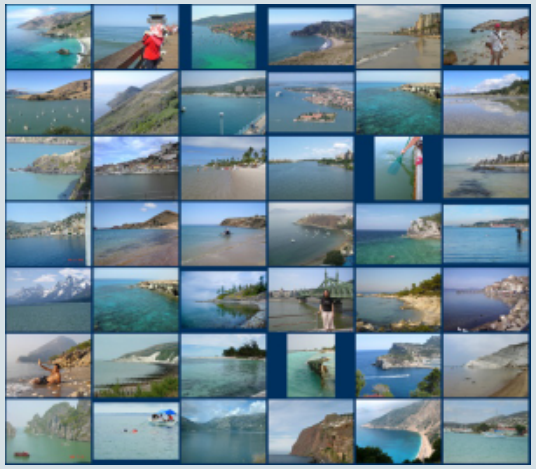
Shingling

Minhashing

Locality-Sensitive Hashing

[Hays and Efros, SIGGRAPH 2007: http://graphics.cs.cmu.edu/projects/scene-completion/]

Scene Completion Problem

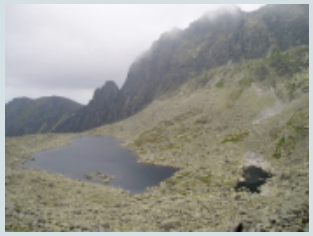


[Hays and Efros, SIGGRAPH 2007: http://graphics.cs.cmu.edu/projects/scene-completion/]

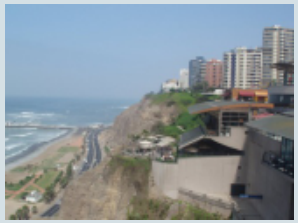
Scene Completion Problem

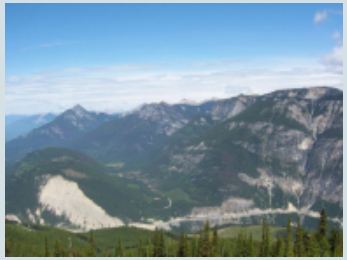
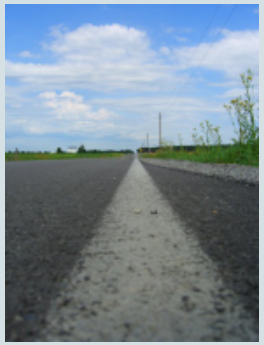
[Hays and Efros, SIGGRAPH 2007: http://graphics.cs.cmu.edu/projects/scene-completion/]

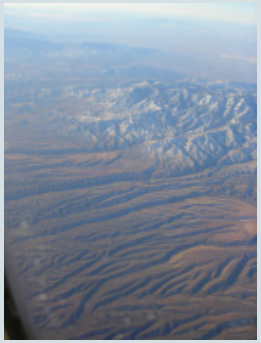
Scene Completion Problem







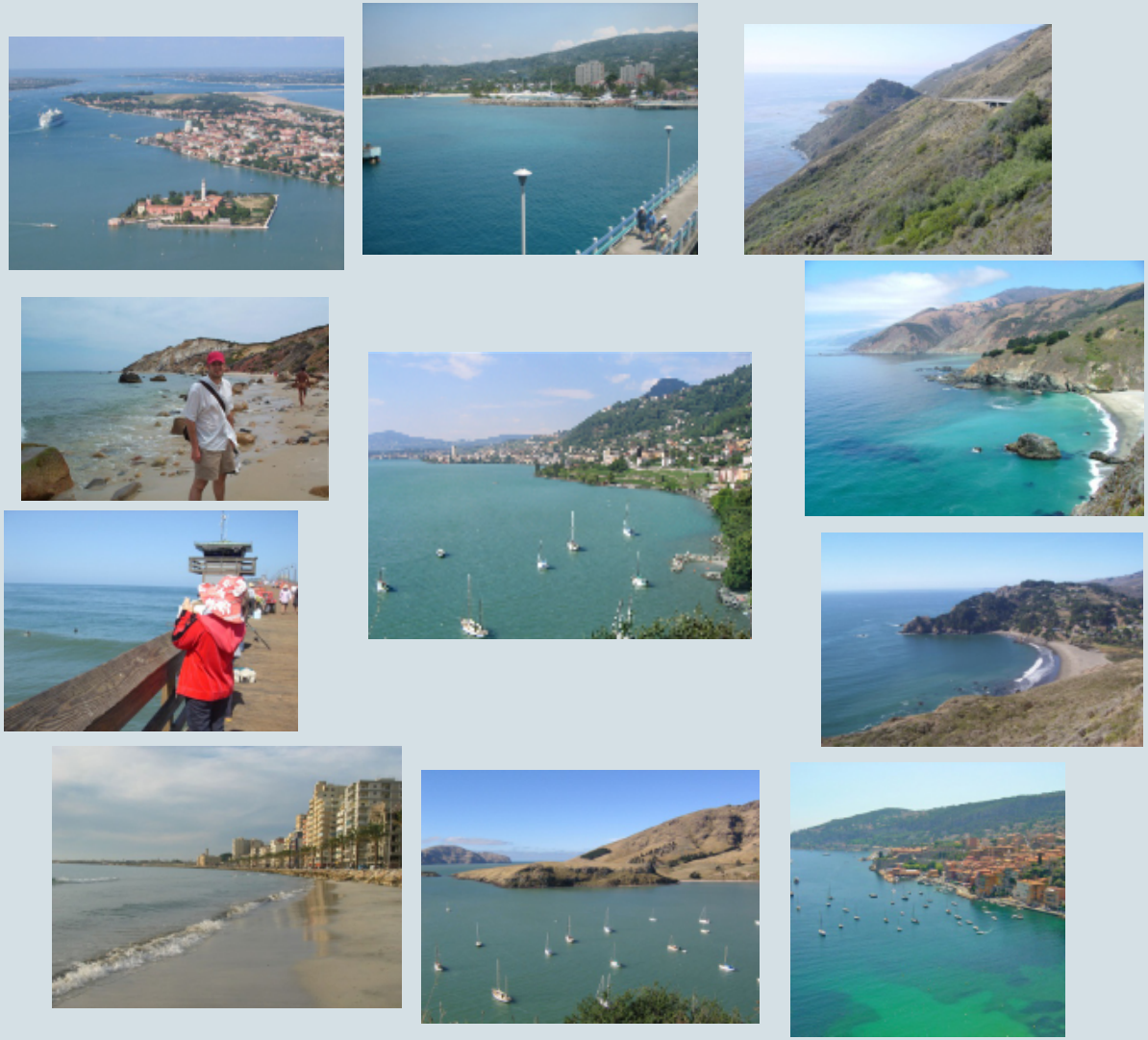



10 nearest neighbors from a collection of 20,000 images

[Hays and Efros, SIGGRAPH 2007: http://graphics.cs.cmu.edu/projects/scene-completion/]

Scene Completion Problem



10 nearest neighbors from a collection of 2 million images

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.
  5. Entity resolution. how do you know 2 products are the same? (various websites)

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 first, then talk about "very."

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.

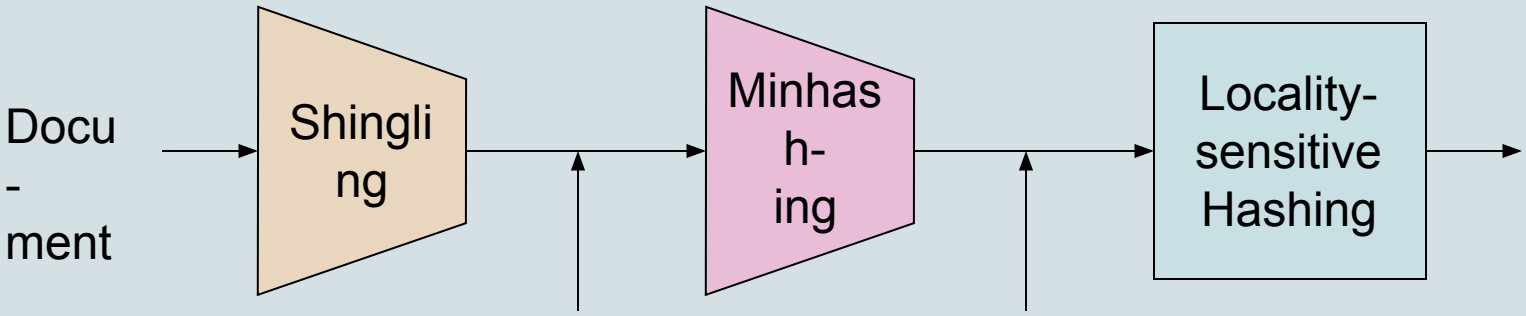
Similar Documents – (2)

- 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. (compressing)
3. *Locality-sensitive hashing*: focus on pairs of signatures likely to be similar.

The Big Picture



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

Candidate pairs: those pairs of signatures that we need to test for similarity.

Shingles

Represent a document by:
Unique k-length strings (or bags)

- 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.

! remove stop words (particles, ...)

Working Assumption

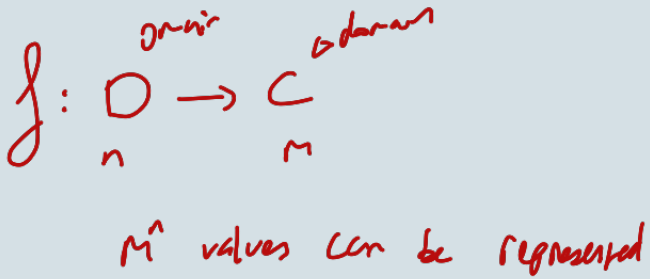
- Documents that have lots of shingles in common have similar text, even if the text appears in different order.
- Careful: you must pick *k* large enough, or most documents will have most shingles.
  - *k* = 5 is OK for short documents; *k* = 10 is better for long documents.

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.
- Problem: Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.

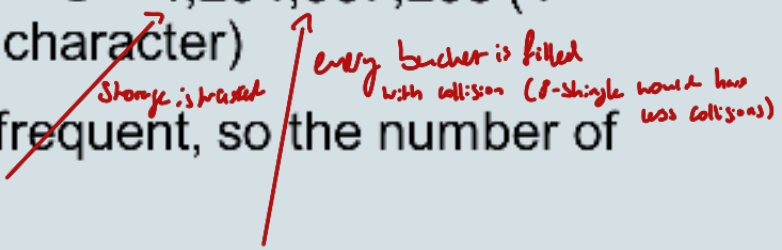
Thought Question

- 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?
- Answer:
  - Number of possible 4-shingles is $2^{32} - 1 = 4{,}294{,}967{,}295$ (4 characters and 1 byte to represent a character)
  - In English around 20 characters are frequent, so the number of different 4-shingles is $20^4 = 160{,}000$
  - Number of different 9-shingles is $20^9 = 512{,}000{,}000{,}000$ when hashed to 4 bytes we expect any sequence of 4 bytes to be possible



MinHashing

Data as Sparse Matrices

Jaccard Similarity Measure

Constructing Signatures

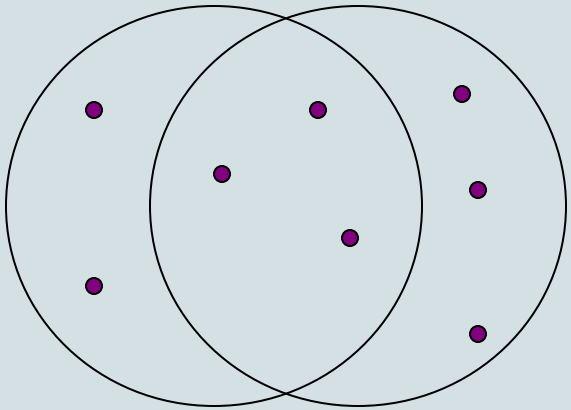
Basic Data Model: Sets

- 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.
  - $Sim\ (C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$. $\leq \frac{1}{2}$

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 = characteristic vector of a set
- Column similarity is the Jaccard similarity of the sets of their rows with 1.
- Typical matrix is sparse:
  - Documents represented by present k-shingles
  - Customers represented by bought books

Example: Jaccard Similarity of Columns

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

not important

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.

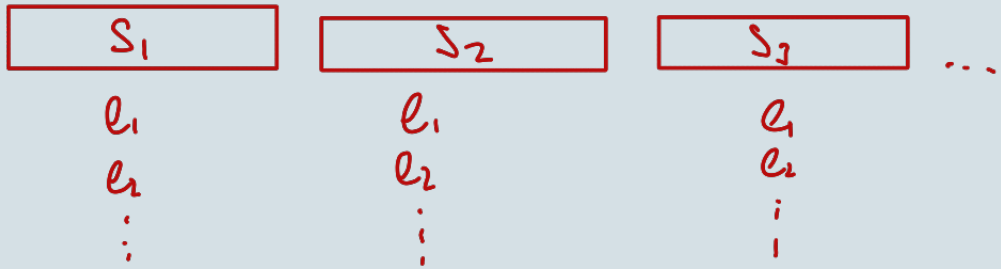
n items

$\binom{n}{2} = \frac{n(n-1)}{2} \approx \frac{n}{2} \cdot n$

pairs

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:
  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).

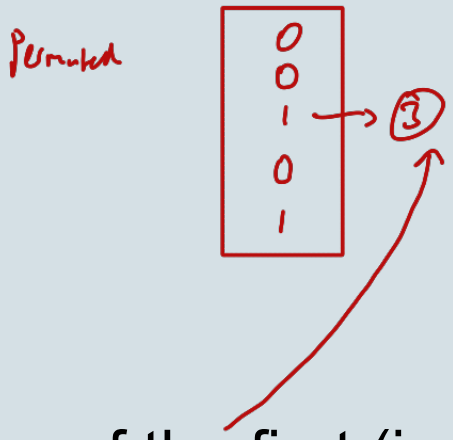
Four Types of Rows

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

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

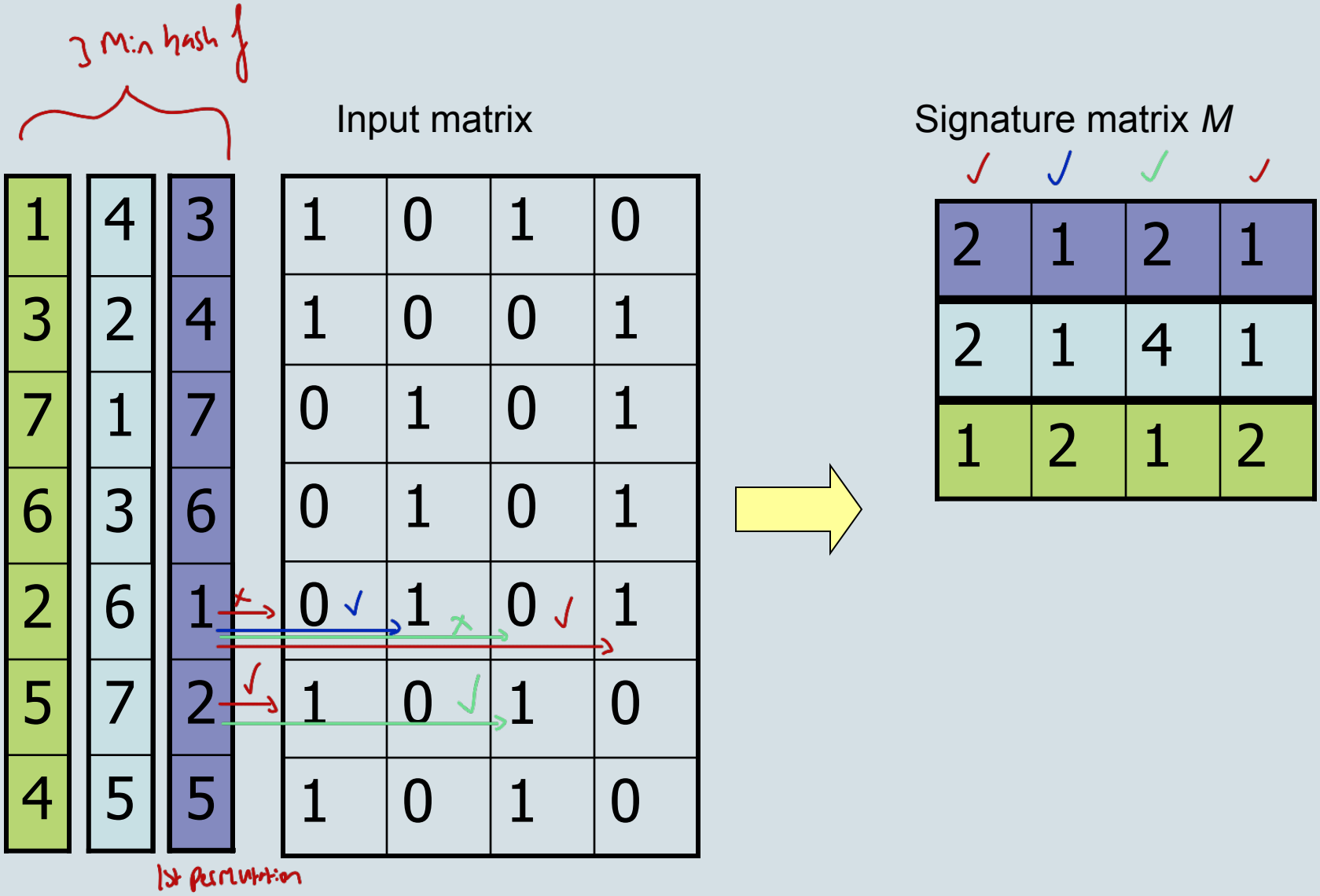
- Also, *a* = # rows of type *a* , etc.
- Note *Sim* (C_1, C_2) = *a* /(*a* +*b* +*c*).

Minhashing

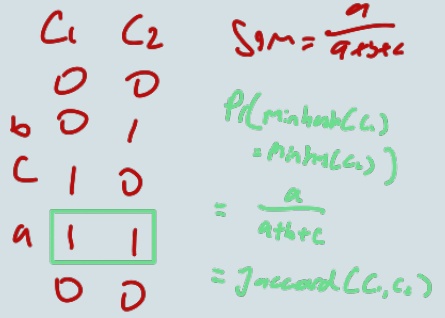


- Imagine the rows permuted randomly.
- Define “hash” function *h* (*C*) = the number of the first (in the permuted order) row in which column *C* has 1.
- Use several (e.g., 100) independent hash functions to create a signature.
- Note: minhashing is a column wide characteristic (and not a set element characteristic)
- Note: a minhash function h is defined by a permutation and a number denoting the first non-zero row

Minhashing Example



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 is a type-*a* row, then $h(C_1) = h(C_2)$. If it is a type-*b* or type-*c* row, then not.
  - Count the number of times it is a type-*a* row versus the total number of times it is a type-*a*, type-*b*, or type-*c* (type-*d* does not count as both columns are 0 and we are looking for 1s)
  - You do not have only 0 in a column, otherwise the set is empty

Similarity for Signatures

- The *similarity of signatures* is the fraction of the hash functions in which they agree.
  - Thinking of signatures as columns of integers, the similarity of signatures is the fraction of rows in which they agree.
- Thus, the expected similarity of two signatures equals the Jaccard similarity of the columns or sets that the signatures represent.
  - And the longer the signatures, the smaller will be the expected error.

NOT Feasible if characteristic Matrix M is huge!

Min Hashing – Example

Input matrix Col/Col

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* Sig/Sig

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

Note: a similarity of zero in the input matrix always triggers a similarity of zero in the signature matrix (the two columns are always minhashed to two different numbers)

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 of 1…billion.
- Representing a random permutation requires 1 billion entries:
  - 4 GBytes for the permutation (4 Bytes for an integer)
  - 100 permutations mean 0.4 TBytes
- Accessing rows in permuted order leads to thrashing.

Implementation – (2)

- A good approximation to permuting rows: pick 100 (?) hash functions.
- For each old 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.
- Pick the range of a hash function larger than the number of rows (or elements in the universal set) in order to avoid collisions (in buckets)

Implementation – (3)

for each row *r*

  for each function h_i **do**

    compute $h_i(r)$

  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	

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

$\lfloor \frac{1}{5} \rfloor = 0$ → $1 \bmod 5 = 1-0 = 1$
$\lfloor \frac{3}{5} \rfloor = 0$ → $3 \bmod 5 = 3-0 = 3$

	Sig1	Sig2
$h(1) = 1$		1	∞
$g(1) = 3$		3	∞
$h(2) = 2$	1	2	$\lfloor \frac{2}{5} \rfloor = 0$ → $2-0 = 2$
$g(2) = 0$	3	0	$\lfloor \frac{5}{5} \rfloor = 1$ → $5-5 = 0$
$h(3) = 3$	1	2	$\lfloor \frac{3}{5} \rfloor = 0$ → $3-0 = 3$
$g(3) = 2$	2	0	$\lfloor \frac{7}{5} \rfloor = 1$ → $7-5 = 2$
$h(4) = 4$	1	2	$\lfloor \frac{7}{5} \rfloor = 0$ → $4-0 = 4$
$g(4) = 4$	2	0	$\lfloor \frac{9}{5} \rfloor = 1$ → $9-5 = 4$
$h(5) = 0$	1	0	$\lfloor \frac{5}{5} \rfloor = 1$ → $5-5 = 0$
$g(5) = 1$	2	0	$\lfloor \frac{11}{5} \rfloor = 2$ → $11-10 = 1$

$= M$

Note: We pass through the original matrix only once,
while maintaining a second matrix M(hash_function, column)!

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:
  - Check for pairs (column, row) which store a 1
  - Sort these pairs by row (instead of column)
- And *always* compute $h_i(r)$ only once for each row.

Locality-Sensitive Hashing

Focusing on Similar Minhash Signatures

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 approximately $5*10^{11}$ column-comparisons.
- 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.
- The number of buckets:
  - Not too large, otherwise you miss truly similar pairs
  - Not too small, otherwise you have to do too many comparisons (too many candidate pairs) bucket contains >2 cols
- Note: **We look only at individual elements, not at the pairs themselves!**

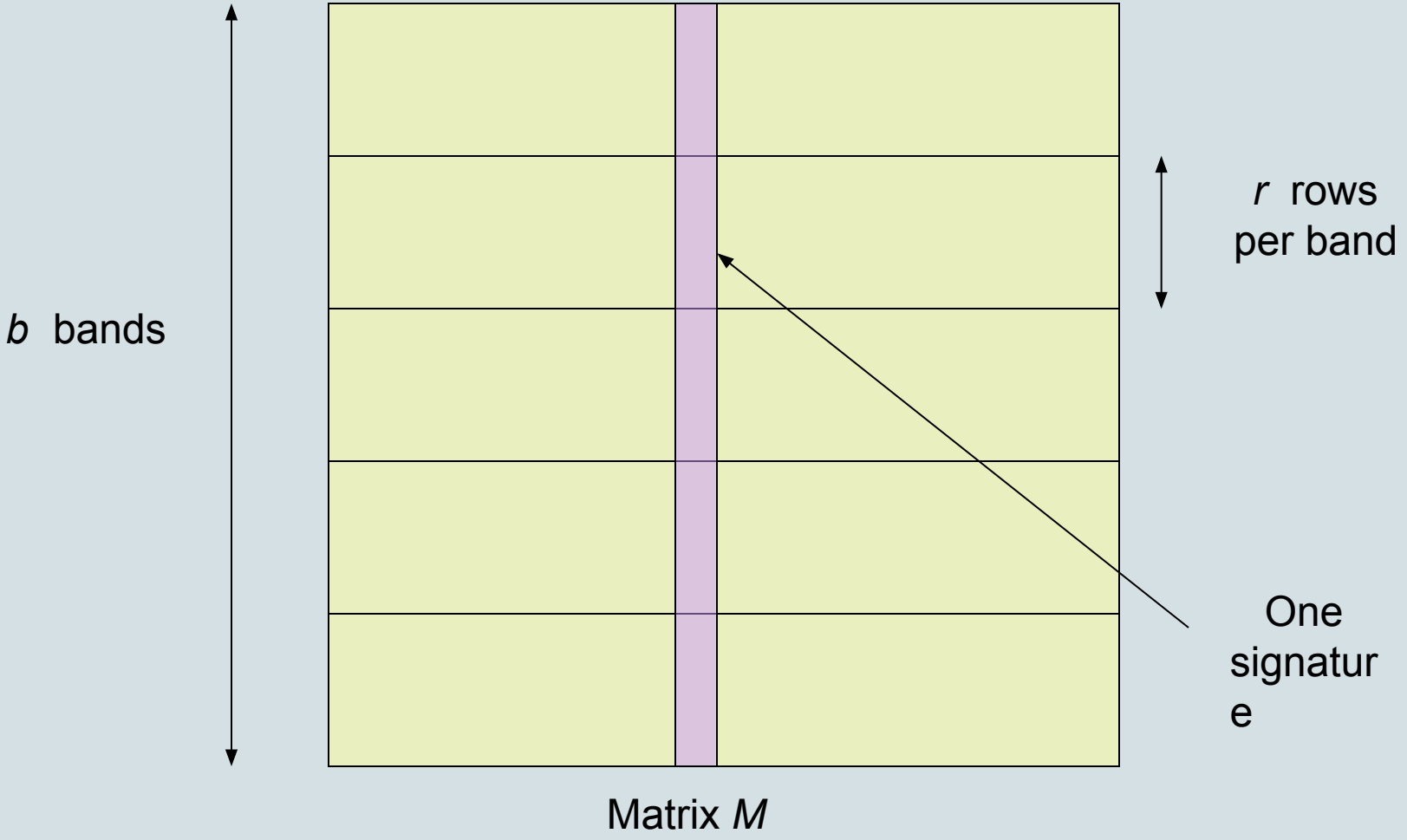
Candidate Generation From Minhash Signatures

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

LSH for Minhash Signatures

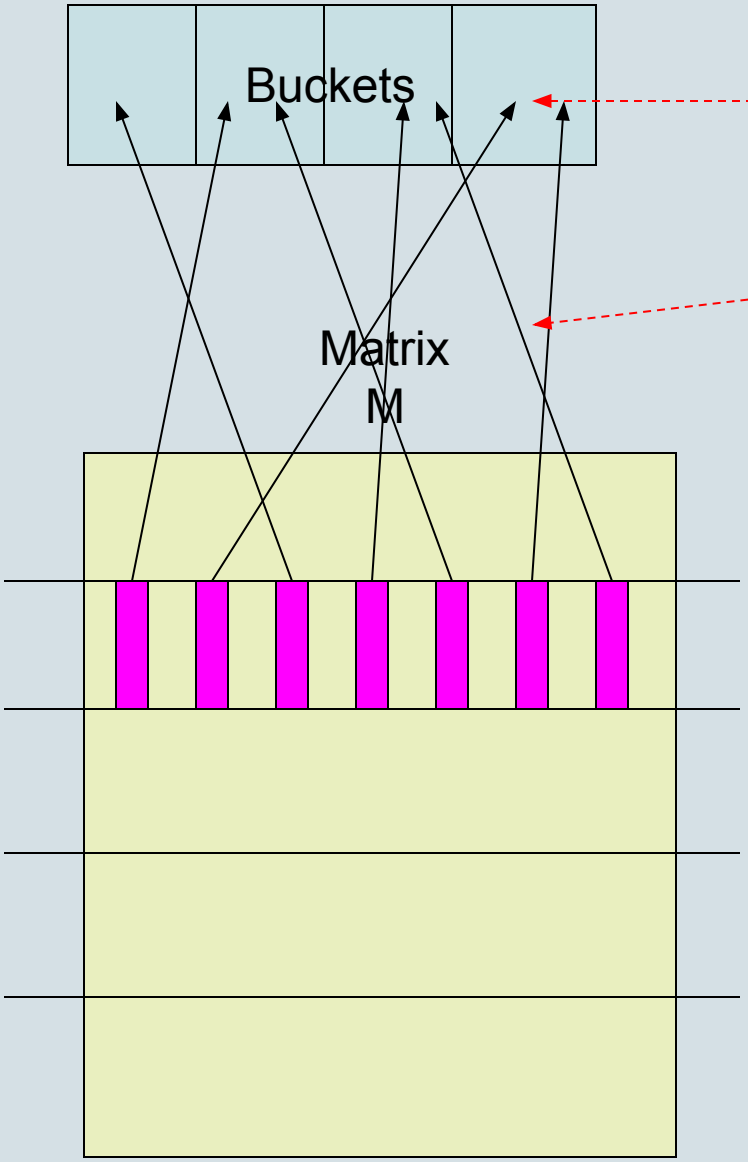
- Big idea: hash columns of signature matrix *M* several times:
  - More precisely we hash a part (band) of the column with one hash function (could be the same for all bands)
- Trick: Arrange that (only) similar columns are likely to hash to the same bucket (shown next).
- Candidate pairs are those 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. → you want identical b length vectors in one buckets.
- *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.



Entire
Columns 2 and 6
are probably identical. Not just the band

We evaluate for all b bands, if for one b agrees (same bucket) the entire cols are candidate pairs

Columns 6 and 7
are
surely different.

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.”

Positives and Negatives

- True positives (TP) = similar pairs that are put in the same bucket
- True negatives (TN) = non-similar pairs that end in different buckets
- False positives (FP) = non-similar pairs that are put in the same bucket
- False negatives (FN) = similar pairs that are put in different buckets
- Tradeoff between FP and FN (if signature vector length fixed)
- Which ones are worse FP or FN?

Positives and Negatives

- True positives (TP) = similar pairs that are put in the same bucket
- True negatives (TN)= non-similar pairs that end in different buckets
- False positives (FP) = non-similar pairs that are put in the same bucket
- False negatives (FN) = similar pairs that are put in different buckets
- Tradeoff between FP and FN (if signature vector length fixed)
- Which ones are worse FP or FN?
  - FN as we do not have the chance to check them to be similar
  - FP usually undergo a similarity check afterwards

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.

rows

Suppose C_1, C_2 are 80% Similar

5 rows in one 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 any of the 20 bands: $(1\text{-}0.328)^{20} = 0.00035$.
  - 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: $(0.4)^5 = 0.01$ (Why?)
  - Probability($minhash_{row}(C_1)$=$minhash_{row}(C_2)$)=Jaccard(C_1,C_2)=0.4
- Probability C_1, C_2 identical in ≥ 1 of 20 bands: ≤ 20 * 0.01 = 0.2 ☹:
  - Among 40% similar sets there are 20% false positives (candidates that need comparison without being 80% similar)
- But false positives much lower for similarities << 40% ☺.
  - For 20% Jaccard similarity we get less than 1% false positives (Why?)
    - $20 * (0.2)^5=20*0.00032=0.0064$

False Positives

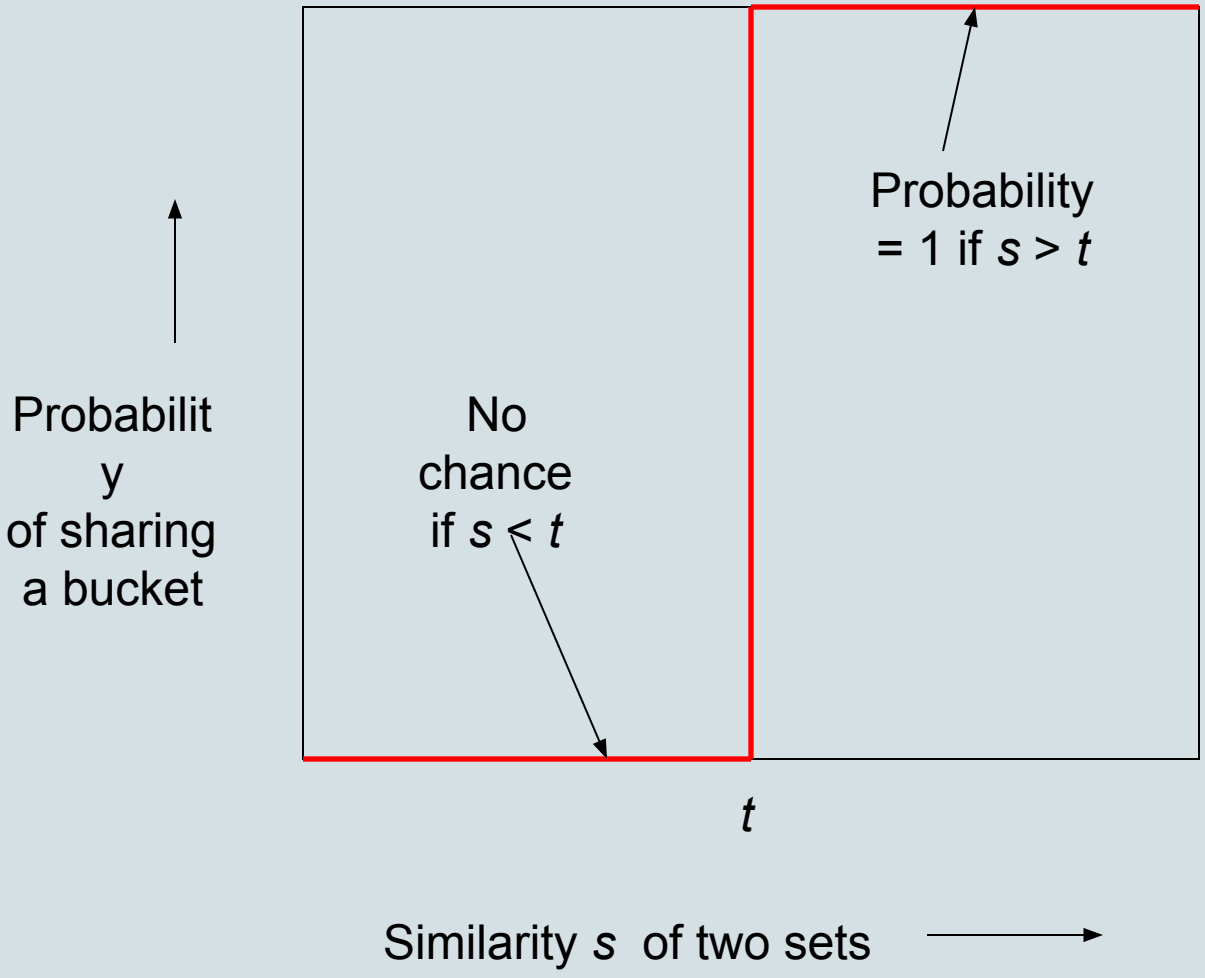
- The number of false positives depends on the distribution of the Jaccard similarities of the underlying sets:
  - All sets have similarities around 0.79, then most of them will be false positives (they will be in the same buckets but do not pass the 80% threshold)
  - All sets have similarities around 0.1, then almost none of them will be false positives (they will not be in the same buckets)

LSH Involves a Tradeoff

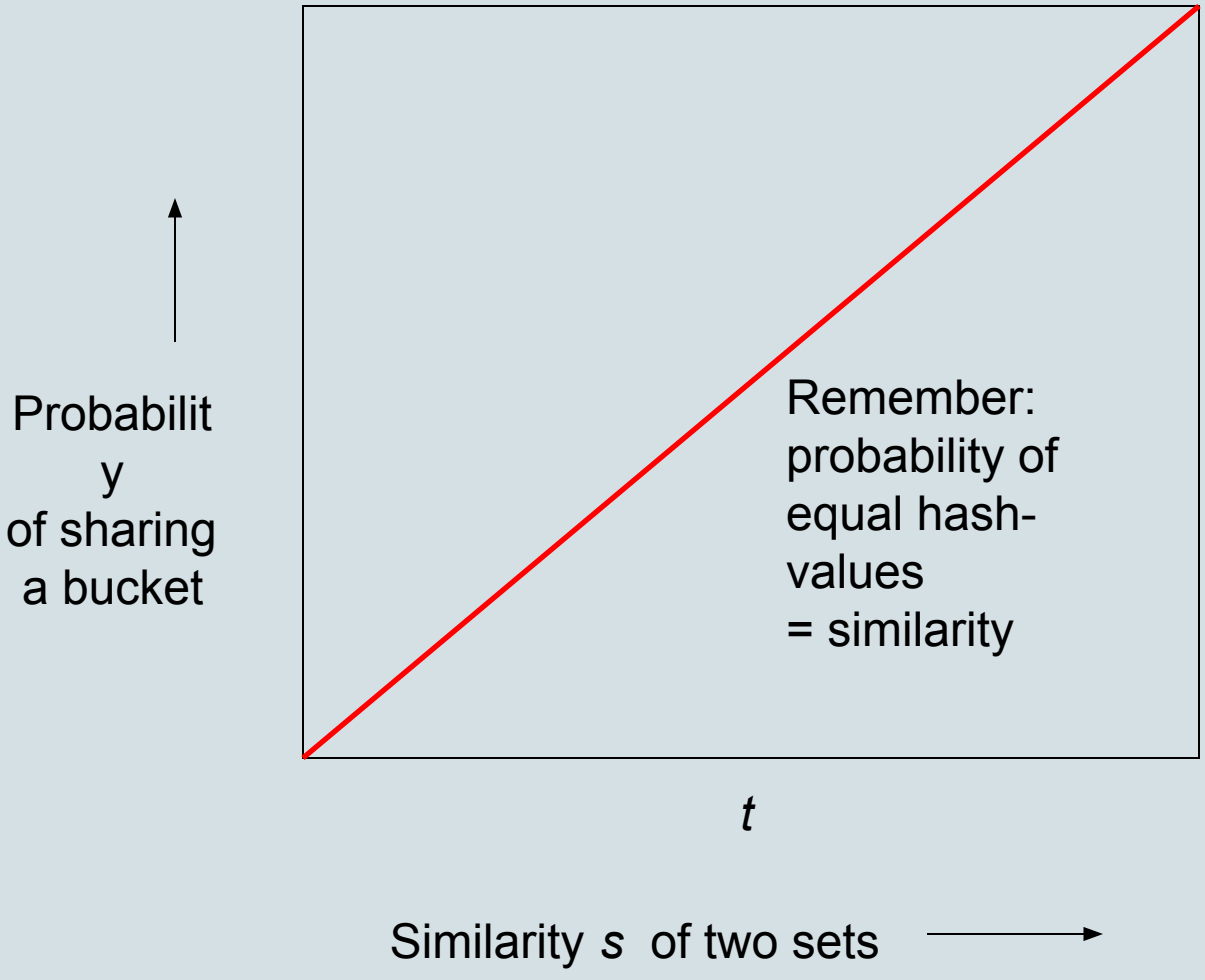
- Pick the number of minhashes, the number of bands, and the number of rows per band to balance false positives/negatives.
- Example: if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up.

you have less chances (20 > 15) to end up in the same bucket.

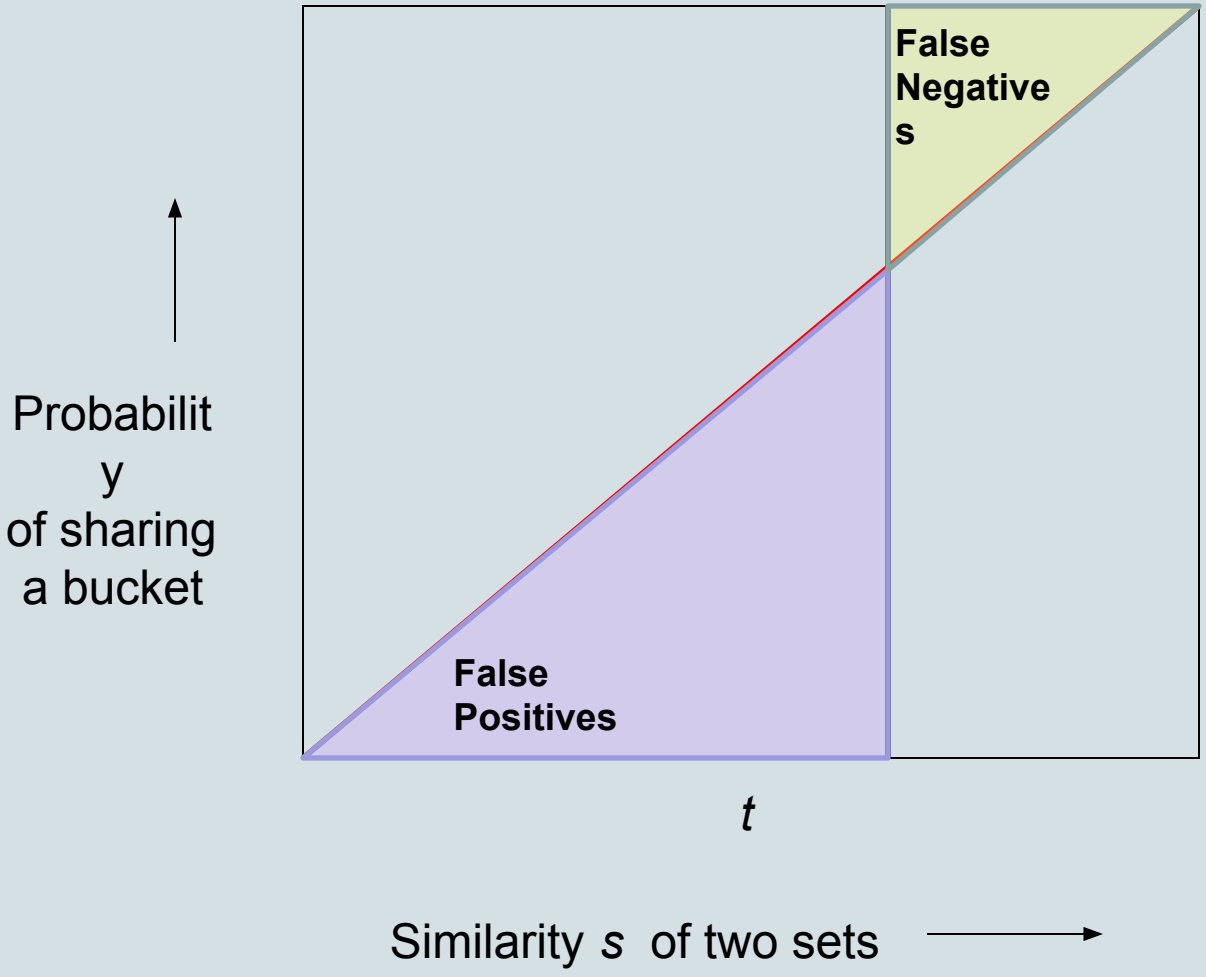
Analysis of LSH – What We Want



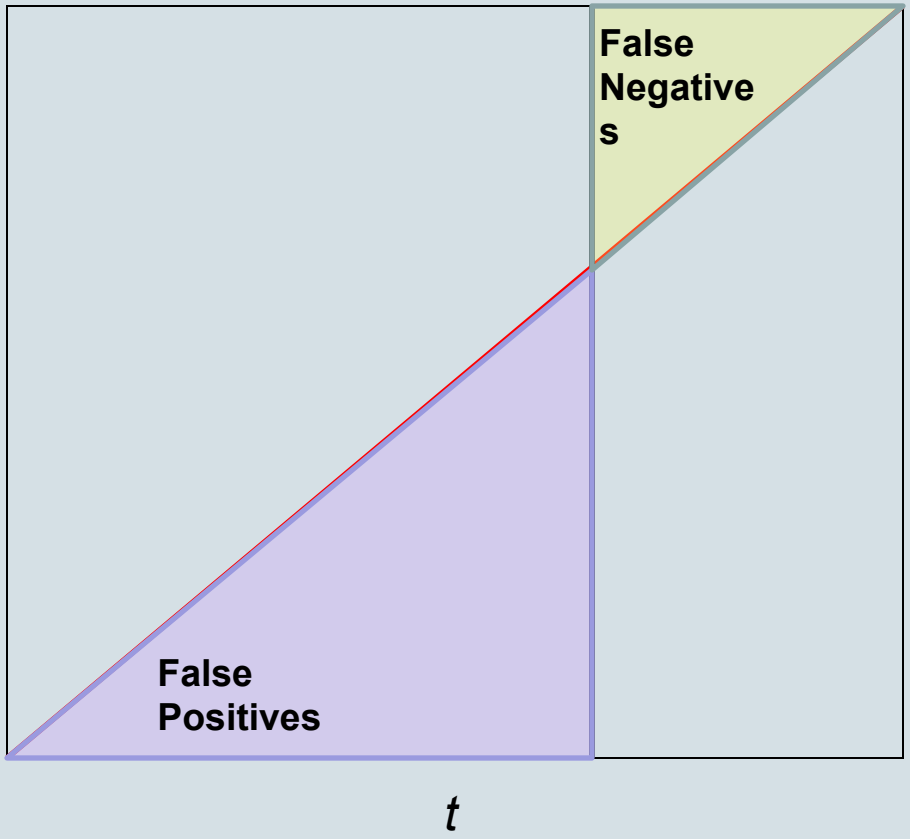
What One Band of One Row Gives You (One Minhash Function)



What One Band of One Row Gives You (One Minhash Function)



What One Band of One Row Gives You (One Minhash Function)



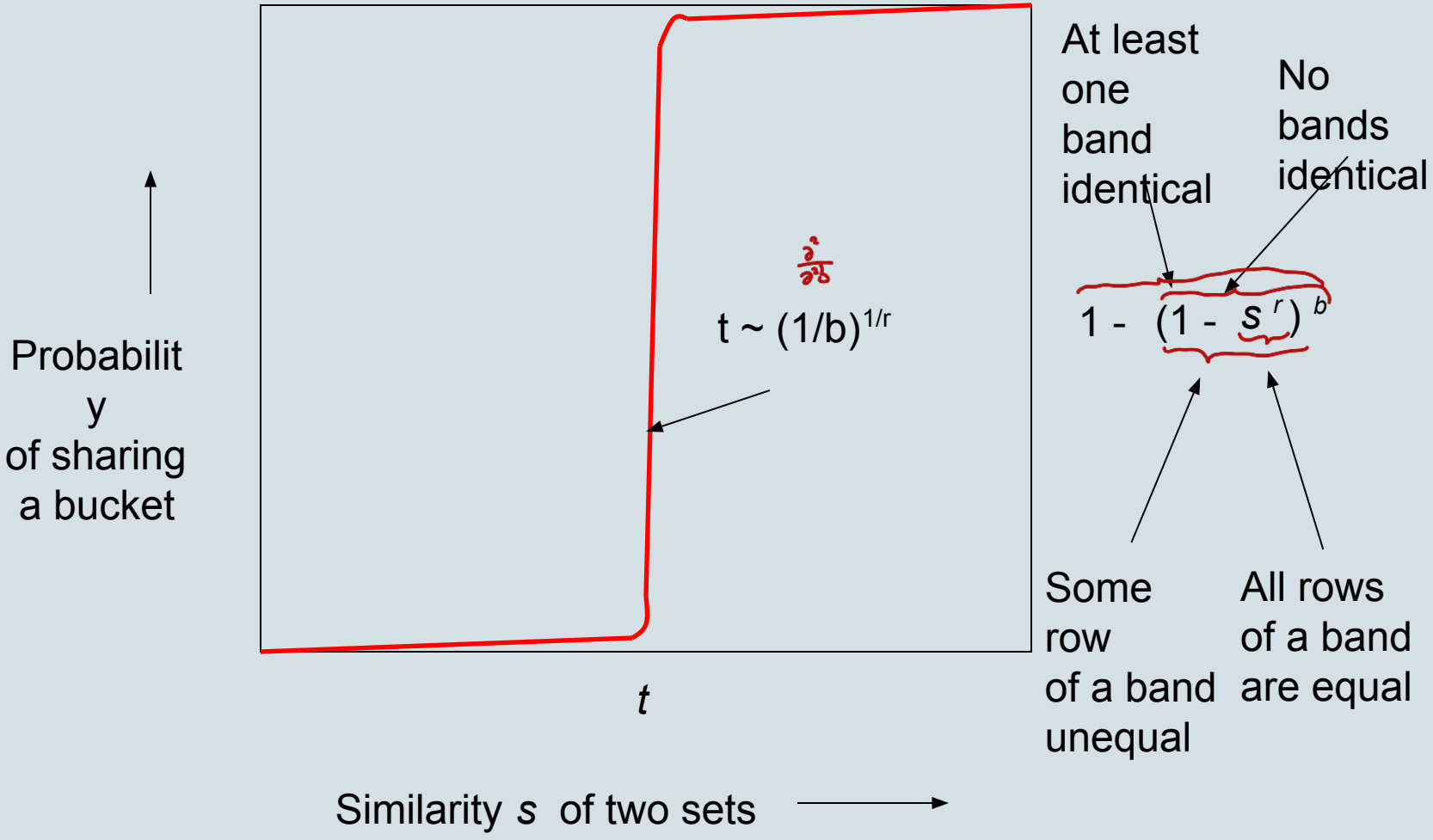
Probability of sharing a bucket

Similarity s of two sets

Not all Jaccard similar sets (similarity bigger than t) make it to the buckets

Some Jaccard non-similar sets (similarity smaller than t) make it to the buckets

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

As b and r get larger this function resembles more and more the step function

$$\lim_{r\to\infty}(1-(1-s^r)^b)=0$$
$$\lim_{b\to\infty}(1-(1-s^r)^b)=1$$
$$\lim_{r,b\to\infty}(1-(1-s^r)^b)=0$$

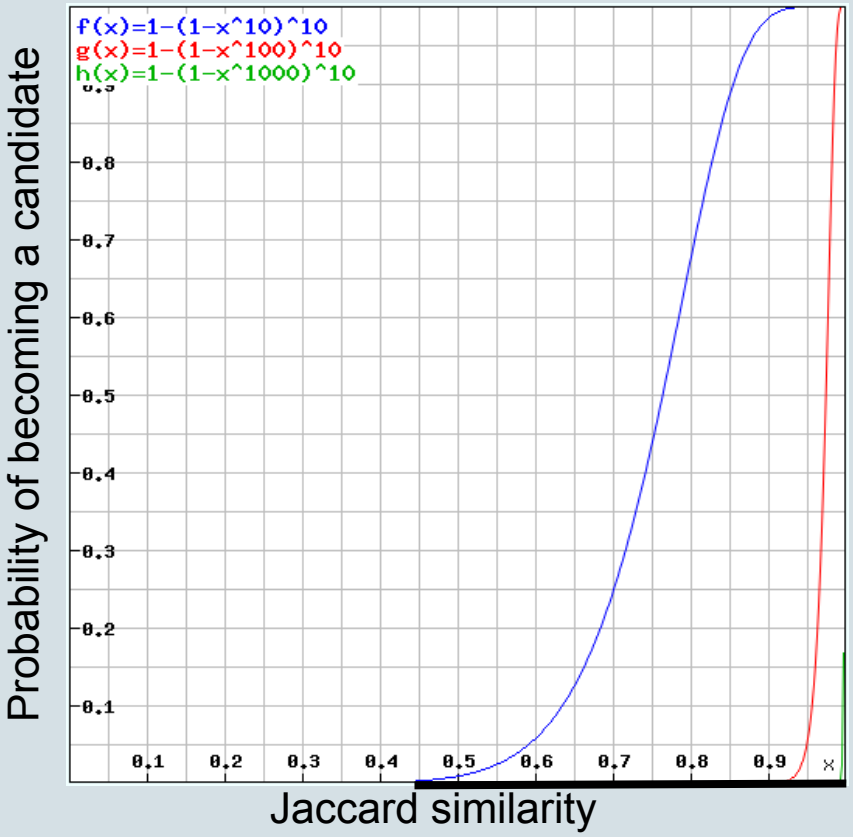
Rise occurs at $(1/b)^{(1/r)}$

$$\lim_{r\to\infty}(1/b)\text{^}(1/r)=1$$
$$\lim_{b\to\infty}(1/b)\text{^}(1/r)=0$$
$$\lim_{r,b\to\infty}(1/b)\text{^}(1/r)=1$$

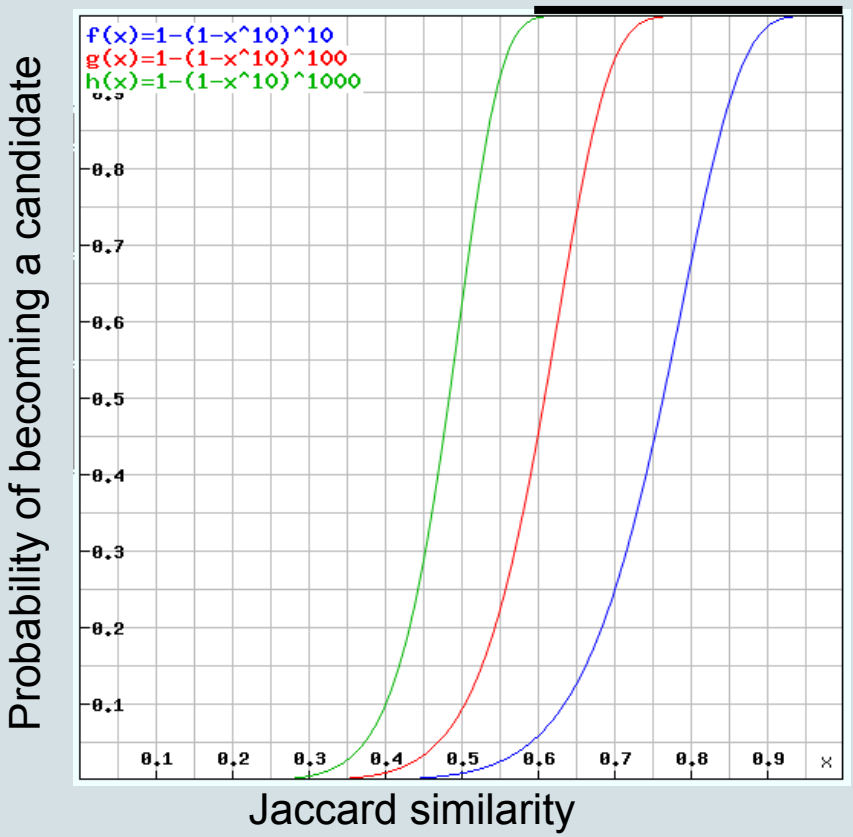
Note: For $r, b \to \infty$ the limit is undefined (assume that they grow equally fast to obtain the given results, i.e., $0^0 = 1$)

b versus r

x = s, r = 10,100,1000, b = 10

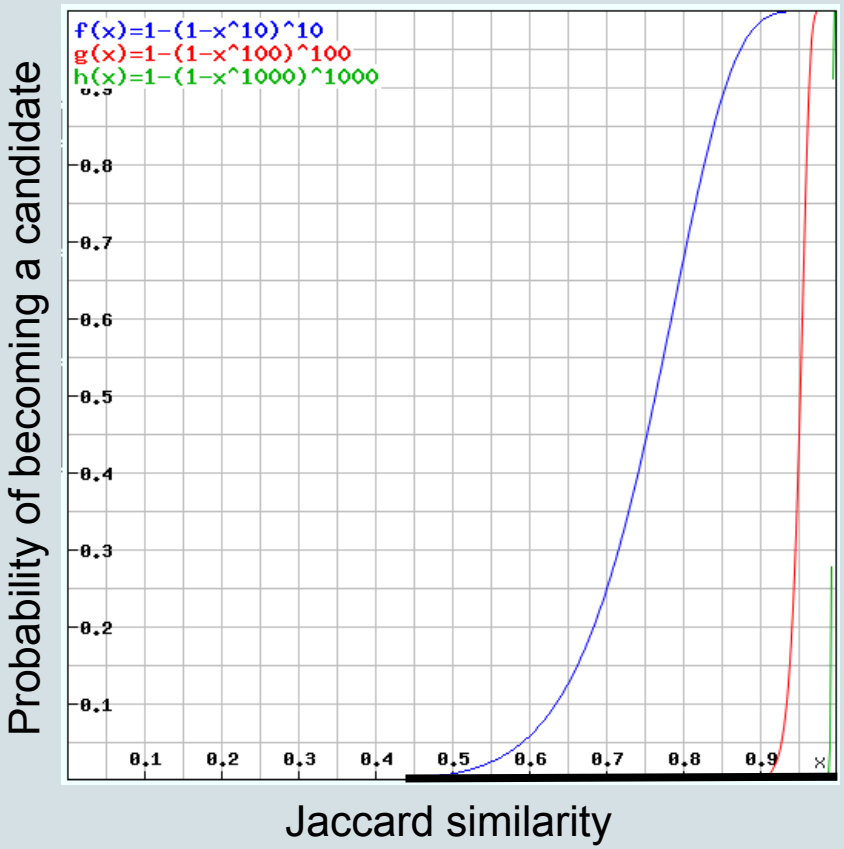


x = s, r = 10, b = 10,100,1000



b versus r

x = s, r = 10, 100, 1000, b = 10,100,1000



More bands more chances

More rows harder match

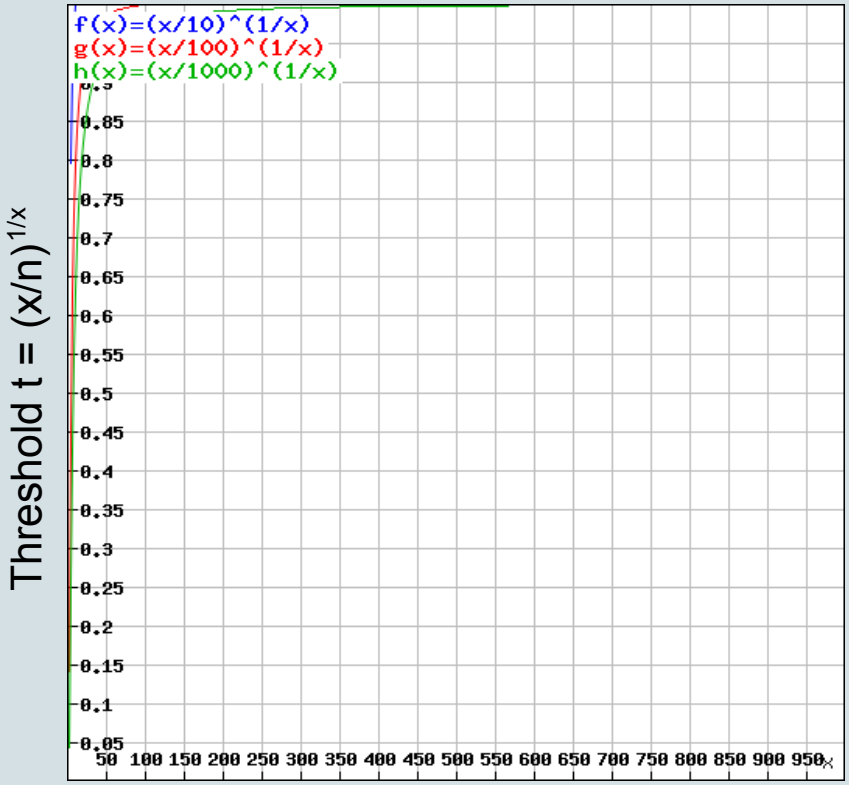
Note: As b and r are increasing with the same speed, r dominates b

Example: $b = 20$; $r = 5$ (n = 100)

- Threshold t = $\sqrt[5]{1/20} \cong 0.55$

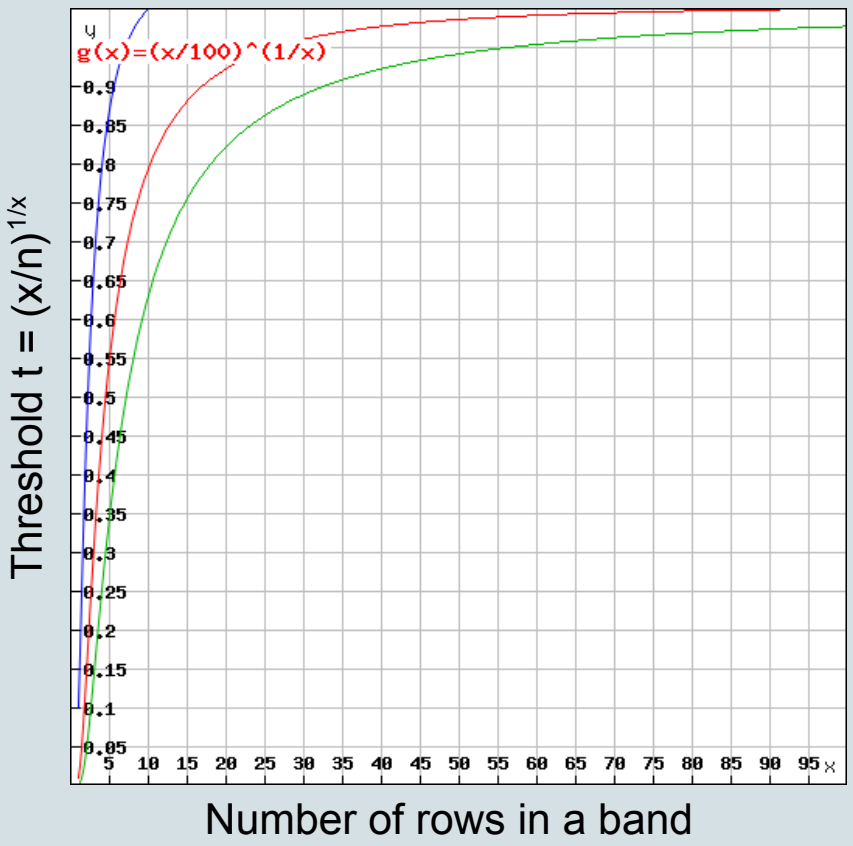
t versus r

x = r, n = 10, 100, 1000



dim sig matrix

x = r, n = 10, 100, 1000



Note: For large signatures it takes more rows to arrive to the same threshold

Setting parameters

- Pick a length n of minhash signatures
- Select threshold t that defines how similar documents have to be in order to be regarded as similar pair (e.g., 80%)
- Pick number of bands b and number of rows per band r such that:
  - $b \times r = n$
  - $(1/b)^{(1/r)} = t$
  - If speed and limit of false positives are important, select b and r to produce a higher threshold than t
  - If avoidance of false negatives is important, select b and r to produce a lower threshold than t

Setting parameters

- n = 100
- t = 0.8 (Jaccard similarity of 80% in order to be considered identical)
  - b x r = 100
  - $(1/b)^{(1/r)} = 0.8$
- b = ?
- r = ?

Setting parameters

- n = 100
- t = 0.8 (Jaccard similarity of 80% in order to be considered identical)
  - b x r = 100
  - $(1/b)^{(1/r)} = 0.8$
- b = 10
- r = 10

Setting parameters

- n = 100
- t = 0.8 (Jaccard similarity of 80% in order to be considered identical)
  - b x r = 100
  - $(1/b)^{(1/r)} = 0.8$
- Avoidance of false positives and speed are important (t = 0.9 > given 0.8)
- b = ?
- r = ?

[Which one should increase compared to previous solution]

Setting parameters

- n = 100
- t = 0.8 (Jaccard similarity of 80% in order to be considered identical)
  - b x r = 100
  - $(1/b)^{(1/r)} = 0.8$
- Avoidance of false positives and speed are important (t = 0.9 > given 0.8)
- b = 5
- r = 20

Setting parameters

- n = 100
- t = 0.8 (similarity of 80% in order to be considered identical)
  - b x r = 100
  - $(1/b)^{(1/r)} = 0.8$
- Avoidance of false negatives is important (t = 0.6 < given 0.8)
- b = 20
- r = 5

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*.

S_{ij} = Sim of interest

Assignment 3

- Exercises: 3.1.1, 3.4.1, 3.4.2, and 3.4.3 (from “Mining of Massive Datasets”)

5 bootstraps (63%)