Finding Similar Items

Shingling
Minhashing
Locality-Sensitive Hashing

Jeffrey D. Ullman
Stanford University

Administrivia

- Wednesday, January 13
  Computer Forum Career Fair
  11am - 4pm
  Lawn between the Gates and Packard Buildings
- Policy for HW2 regarding tagging of subproblems:
  - Do it.
  - There will be deductions of up to 5 points for failure to tag, minus another 2 points if the cover sheet is missing.

The Big Picture

- It has been said that the mark of a computer scientist is that they believe hashing is real.
  - I.e., it is possible to insert, delete, and lookup items in a large set in O(1) time per operation.
- *Locality-Sensitive Hashing* (LSH) is another type of magic that, like Bigfoot, is hard to believe is real, until you've seen it.
- It lets you find pairs of similar items in a large set, without the quadratic cost of examining each pair.

LSH: The Bigfoot of CS

- LSH is really a family of related techniques.
- In general, one throws items into buckets using several different "hash functions."
- You examine only those pairs of items that share a bucket for at least one of these hashings.
- Upside: designed correctly, only a small fraction of pairs are ever examimed.
- Downside: there are *false negatives* – pairs of similar items that never even get considered.

Some Applications

- We shall first study in detail the problem of finding (lexically) similar documents.
- Later, two other problems:
  - *Entity resolution* (records that refer to the same person or other entity).
  - News-article similarity.

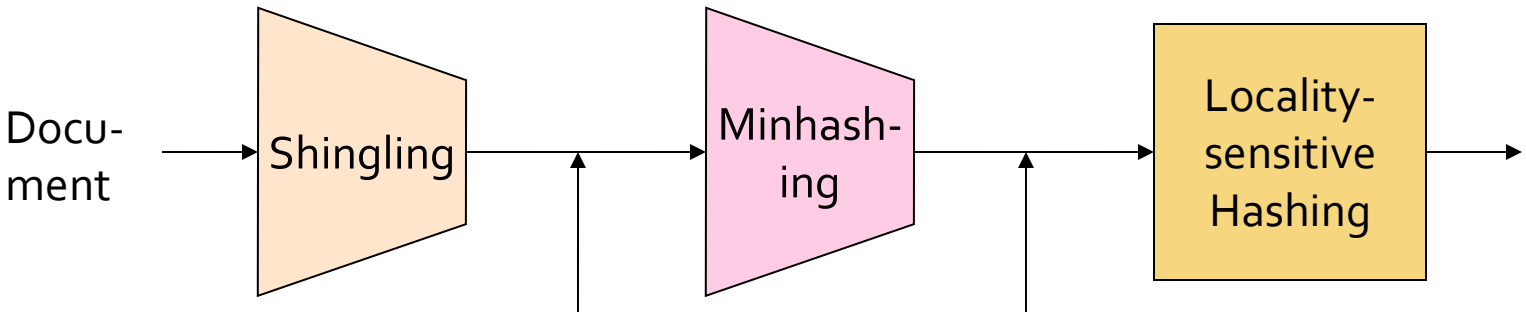
Similar Documents

- Given a body of documents, e.g., the Web, find pairs of documents with a lot of text in common, such as:
  - 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 (lists of integers), while preserving similarity.
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

- 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}.
- Represent a doc by its set of *k*-shingles.

Shingles and Similarity

- Documents that are intuitively similar will have many shingles in common.
- Changing a word only affects k-shingles within distance k-1 from the word.
- Reordering paragraphs only affects the 2k shingles that cross paragraph boundaries.
- Example: k=3, “The dog which chased the cat” versus “The dog that chased the cat”.
  - Only 3-shingles replaced are g_w, _wh, whi, hic, ich, ch_, and h_c.

Compression Option

- Intuition: want enough possible shingles that most docs do not contain most shingles.
- Character strings are not “random” bit strings, so they take more space than needed.
  - k = 8, 9, or 10 is often used in practice.

Tokens

- To save space but still make each shingle rare, we can hash them to (say) 4 bytes.
  - Called *tokens*.
- Represent a doc by its tokens, that is, 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.

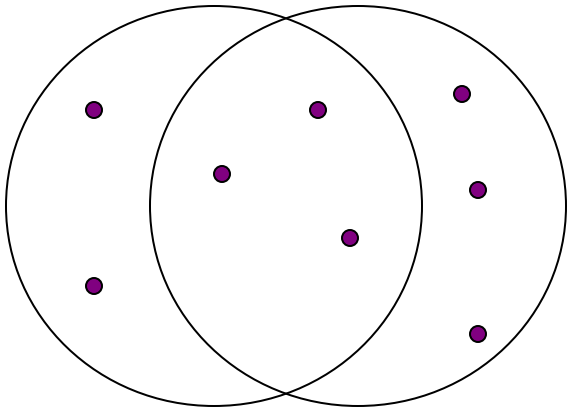
Minhashing

Jaccard Similarity Measure
Constructing Signatures

Jaccard Similarity

- 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.
  - Examples: the set of all k-shingles or all tokens.
- Columns = sets.
- 1 in row *e* and column *S* if and only if *e* is a member of *S*; else 0.
- *Column similarity* is the Jaccard similarity of the sets of their rows with 1.
- Typical matrix is sparse.

Example: Column Similarity

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

$Sim(C_1, C_2) =$

$2/5 = 0.4$

Four Types of Rows

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

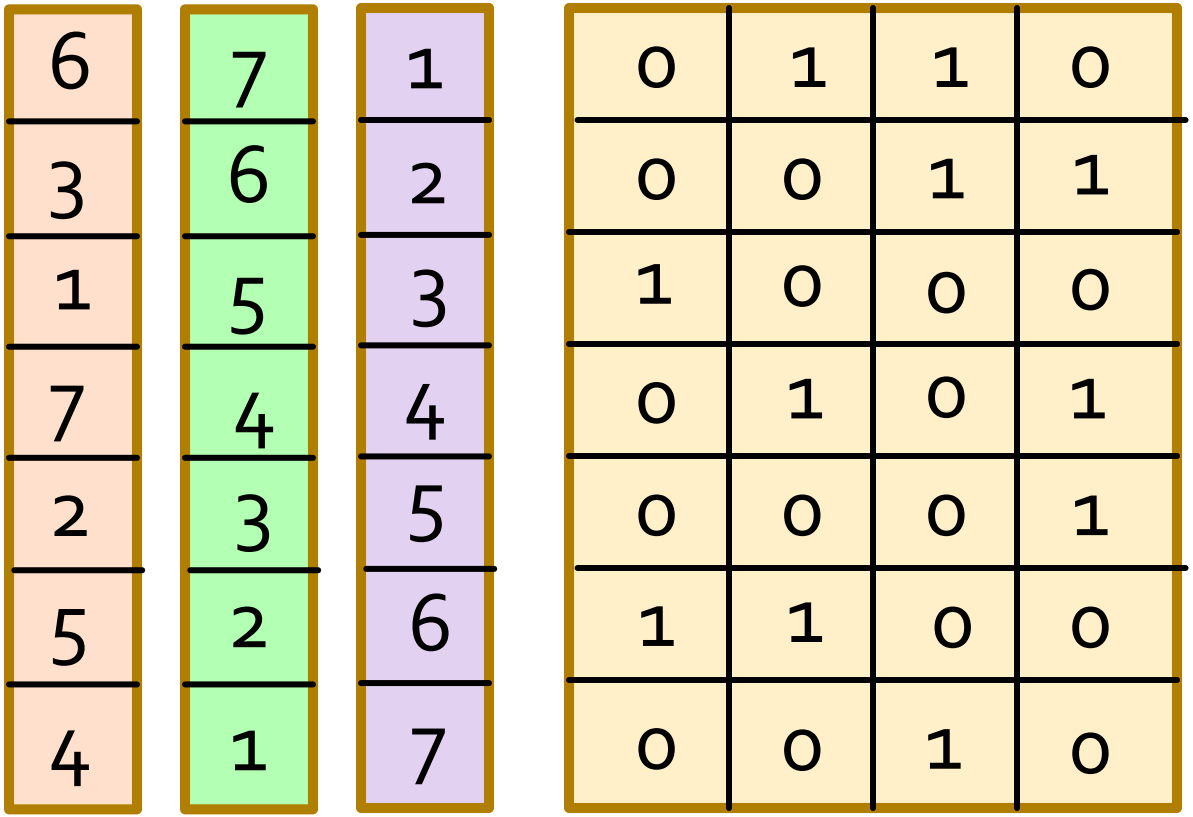
	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

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

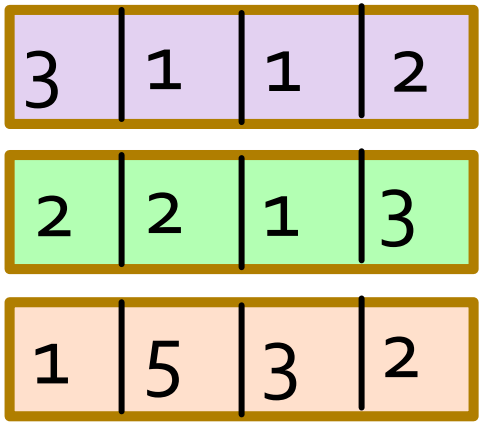
Minhashing

- Permute the rows.
  - Thought experiment – not real.
- Define *minhash function* for this permutation, $h(C)$ = the number of the first (in the permuted order) row in which column C has 1.
- Apply, to all columns, several (e.g., 100) randomly chosen permutations to create a *signature* for each column.
- Result is a *signature matrix*: columns = sets, rows = minhash values, in order for that column.

Example: Minhashing



Input Matrix



Signature Matrix

A Subtle Point

- People sometimes ask whether the minhash value should be the original number of the row, or the number in the permuted order (as we did in our example).
- Answer: it doesn't matter.
- You only need to be consistent, and assure that two columns get the same value if and only if their first 1's in the permuted order are in the same row.

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.

Similarity for Signatures

- The *similarity of signatures* is the fraction of the minhash functions (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.

Example: Similarity

Columns 1 & 2:
Jaccard similarity 1/4.
Signature similarity 1/3

Columns 2 & 3:
Jaccard similarity 1/5.
Signature similarity 1/3

Columns 3 & 4:
Jaccard similarity 1/5.
Signature similarity 0

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

Input Matrix

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

Signature Matrix

Implementation of Minhashing

- Suppose 1 billion rows.
- Hard to pick a random permutation of 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, say, 100 hash functions.
- Intuition: the hash of the row numbers is the order of the corresponding permutation.
- 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.

Implementation – (3)

for each row *r* **do begin**
  for each hash 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 smaller than $M(i, c)$ **then**
          $M(i, c) := h_i(r)$;
end;

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.
  - Example: columns = documents, rows = shingles.
- If so, sort matrix once so it is by row.

Locality-Sensitive Hashing

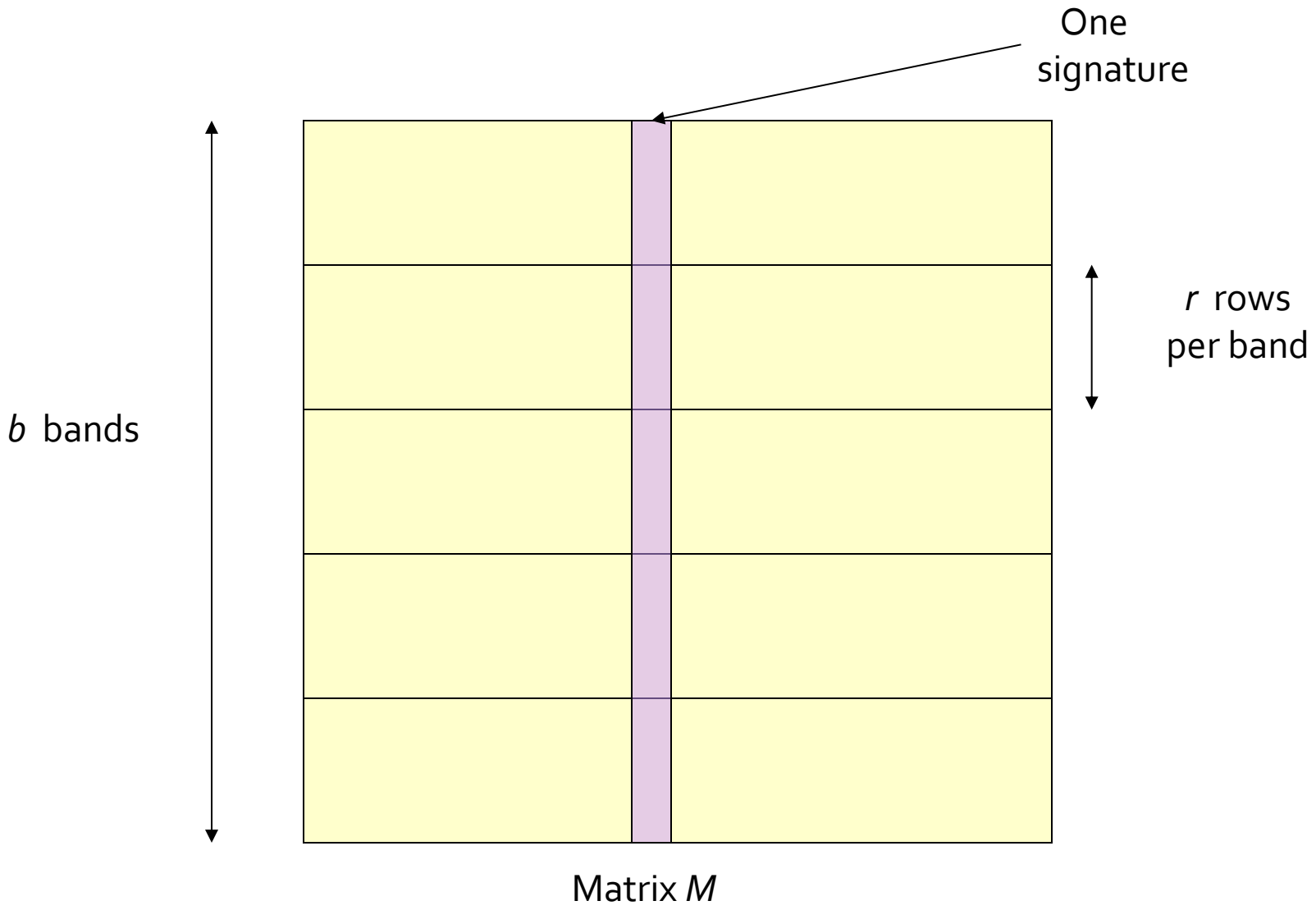
Focusing on Similar Minhash Signatures

Other Applications Will Follow

From Signatures to Buckets

- Remember: we want to 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.
  - *Candidate pairs* are those that share a bucket.
- Pick a similarity threshold t = fraction of rows in which the signatures agree to define "similar."
- Trick: divide signature rows into bands.
  - Each hash function based on one band.

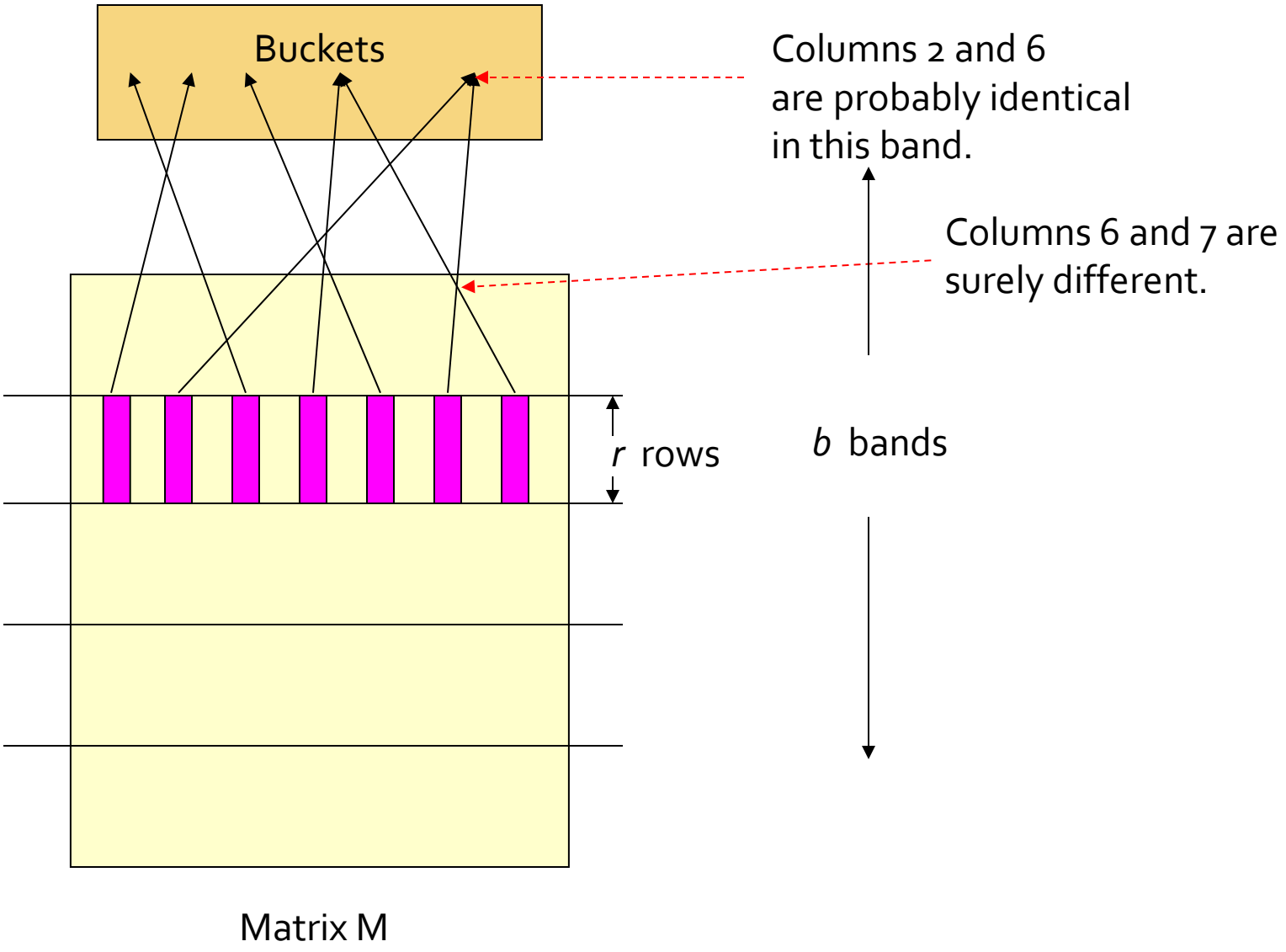
Partition Into Bands



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.

Hash Function for One Bucket



Example: 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.

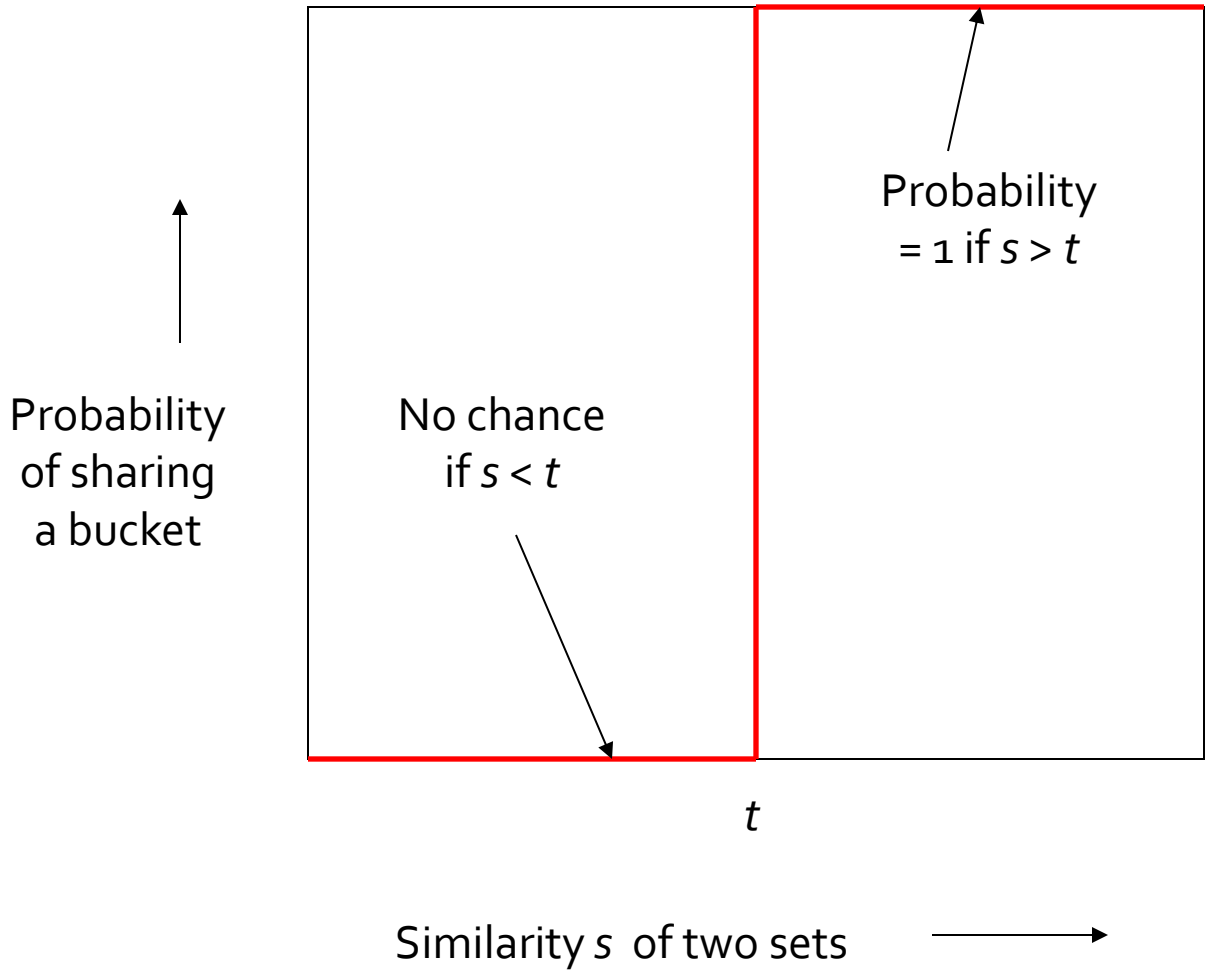
Suppose C_1, C_2 are 80% Similar

- 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 underlying sets are false negatives.

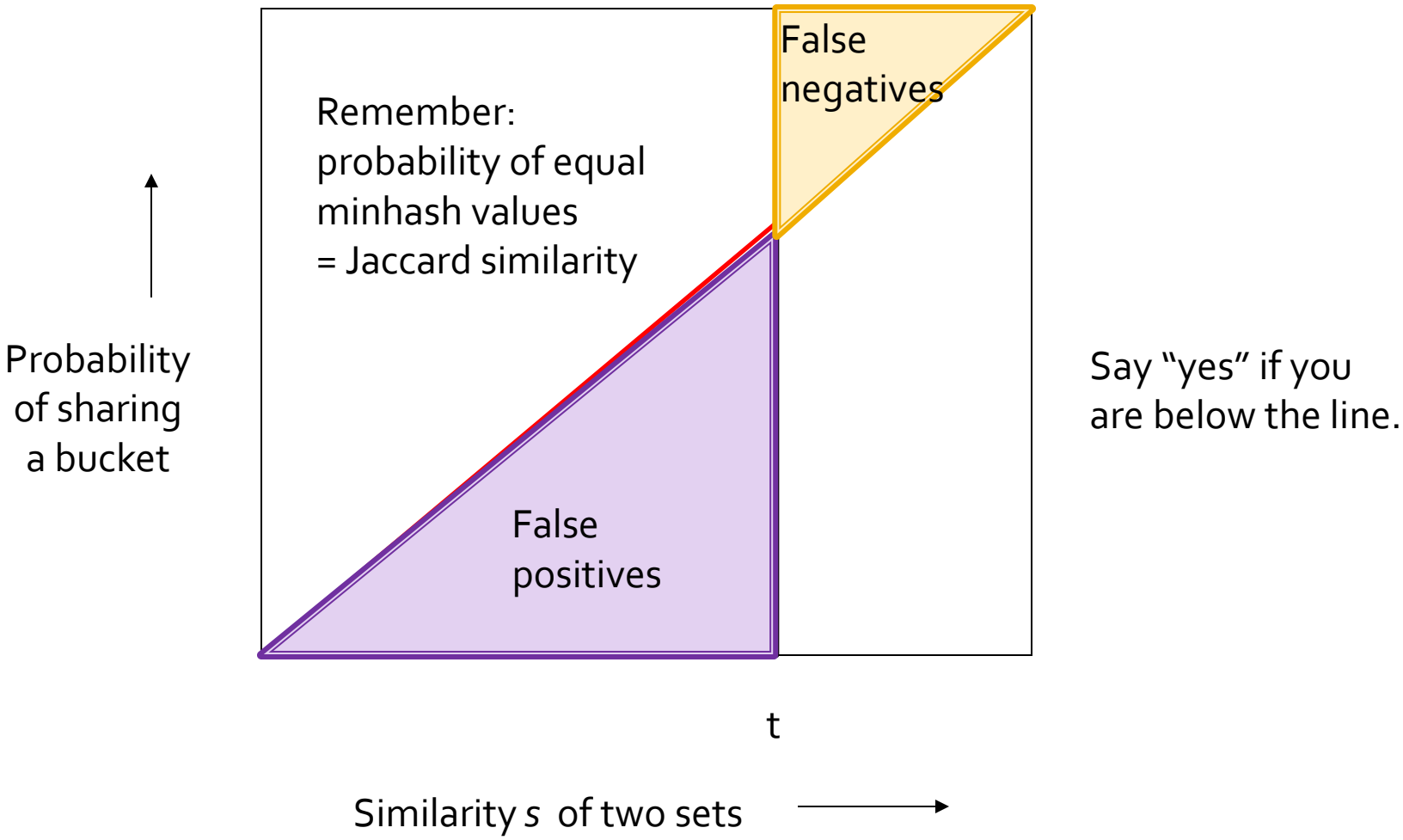
Suppose C_1, C_2 Only 40% Similar

- 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: $1 - (0.99)^{20} < 0.2$.
- But false positives much lower for similarities << 40%.

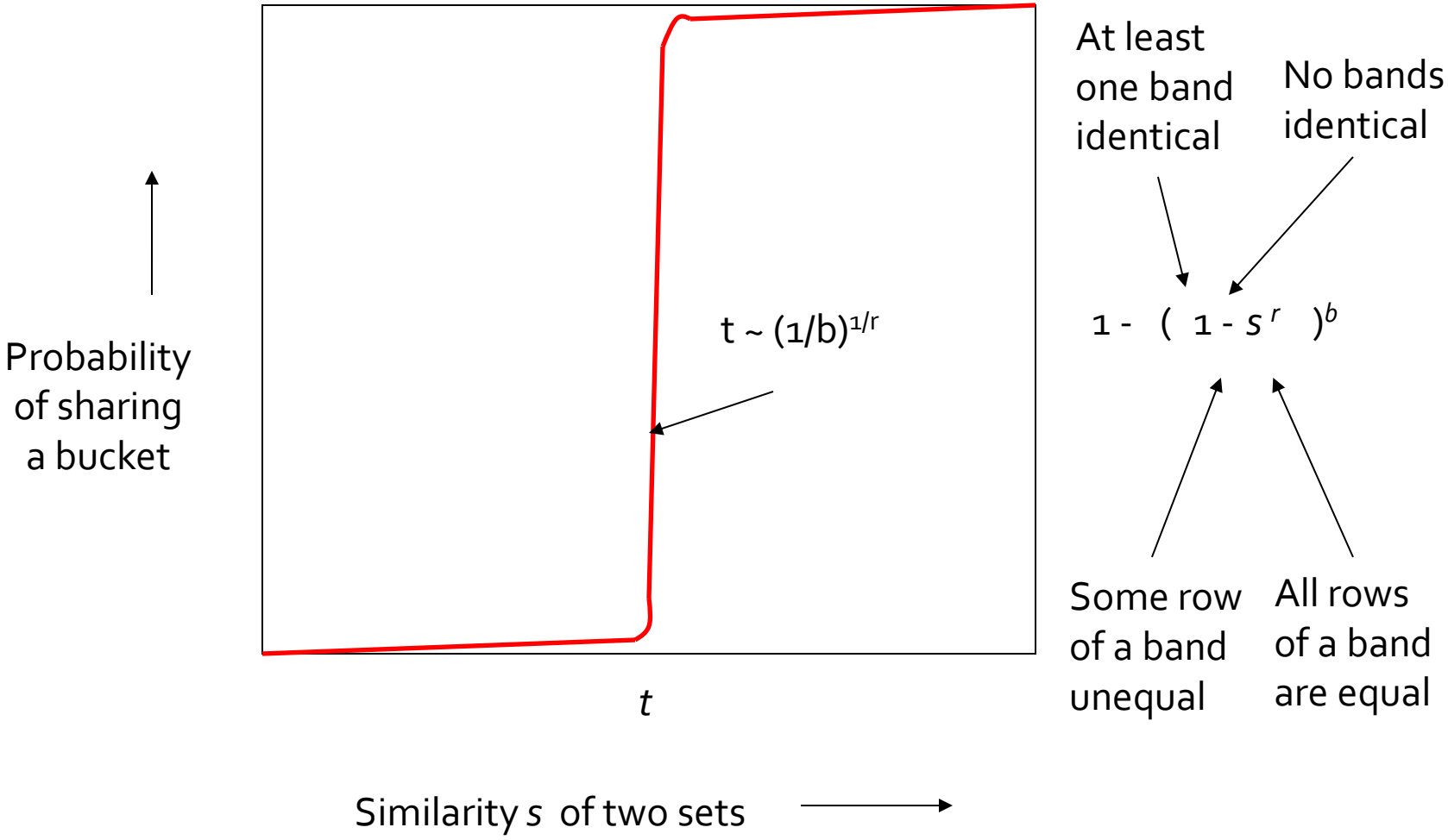
Analysis of LSH – What We Want



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 for Documents: Summary

- Tune b and r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures.
- Check that candidate pairs really do have similar signatures.
- Optional: In another pass through data, check that the remaining candidate pairs really represent similar *sets*.

Entity Resolution

Similarity of Records
A Simple Bucketing Process
Validating the Results

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)

- Each company had about 1 million records describing customers that might have been sent from A to B.
- Records had name, address, and phone, but for various reasons, they could be different for the same person.
  - E.g., misspellings, but there are many sources of error.

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 that the LSH scheme identified as candidates; report high scores as matches.

Customer Records – (4)

- Problem: $(1 \text{ million})^2$ 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.

- Problem: How do we hash strings such as names so there is one bucket for each string?
- Answer: Sort the strings instead.
- Another option was to use 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 an average creation-date difference of 10 days.
- We only looked for records created within 90 days of each other, so bogus matches had a 45-day average difference in creation dates.

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.

Similar News Articles

A New Way of Shingling
Bucketing by Length

Application: Same News Article

- The Political-Science Dept. at Stanford 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

- 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 the similarity threshold is high, there are better methods – see Sect. 3.9 of MMDS.

Enter LSH – (2)

- Their first attempt at minhashing 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.

Specialized 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.