

Fondamenti di Data Science & Machine Learning

Finding Similar Items

Prof. Giuseppe Polese, aa 2024-25

Free Resources

- ▶ Free book covering the spectrum of big-data algorithms: <http://www.mmids.org>
 - ▶ Mining of Massive Datasets by Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman
- ▶ Self-paced MOOC:
<http://mmids.lagunita.stanford.edu>.
- ▶ Also, Automata MOOC:
<http://automata.lagunita.stanford.edu>

The Big Picture

- ▶ A fundamental **data-mining** problem is to examine data for “similar” items.
- ▶ Another common problem is finding **similar documents**:
 - ▶ **near-duplicate** pages in a collection of Web pages. They could be **plagiarisms**, or **mirrors**.
 - ▶ Detecting similar **news-article** from **news sites**:
 - ▶ **Cluster** articles by “same story.”
- ▶ When merging data sources there might be the need of identifying records that refer to the same person or other entity (**Entity resolution**).

Similarity of Documents

- ▶ we focus on **character-level similarity**, not “similar meaning,” which would require different techniques.
- ▶ testing whether two documents are exact duplicates is easy; just compare them character-by-character, stopping if they ever differ.
- ▶ In many applications documents are not identical, yet they share large portions of text (e.g. **plagiarism**, **mirror pages**, etc).

Plagiarism Detection

- ▶ The **plagiarizer** may extract only some parts of a document.
- ▶ S/He may alter a few words and may alter the order in which sentences appear.
- ▶ Yet the resulting document may still contain 50% or more of the original document.
- ▶ No simple **character by character** document comparison process will detect a sophisticated plagiarism.

Mirror Page Detection

- ▶ Pages of **mirror sites** are quite similar, but are rarely identical. For instance, they might each contain information associated with their host, or links to the mirror sites.
- ▶ It is important being able to detect similar pages, because search engines produce better results if they avoid showing two pages that are nearly identical within the first page of results.

Articles from the same source

- ▶ Often a reporter writes a news article that is distributed to many newspapers, which then publish the article on their Web sites.
- ▶ Each newspaper might change the article by cutting paragraphs, surrounding the article by their own logo, ads, and links to other articles.
- ▶ However, the core of each newspaper's page will be the original article.
- ▶ News aggregators (e.g., [Google News](#)), seek all versions of the article, so as to show only one.

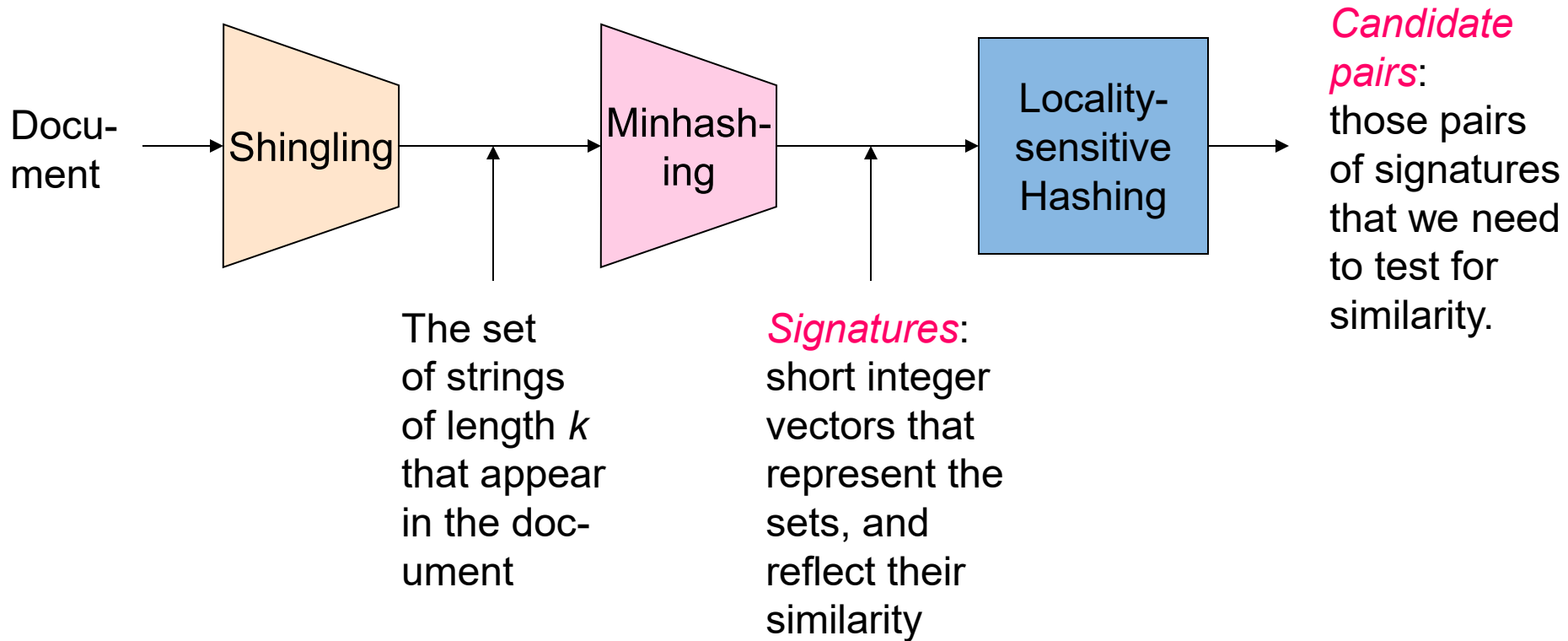
3 Techniques for Similar Documents

1. *Shingling*: convert documents, emails, etc., to sets, so that it is possible using Set-similarity functions (like for instance *Jaccard* function).
2. *Minhashing*: convert large sets to short signatures, while preserving similarity. In this way similarity can be deduced from compressed documents.
3. *Locality-sensitive hashing*: Instead of comparing all possible pairs of items, it focuses on pairs of signatures that are more likely to be similar.

LSH

- ▶ When searching for similar items there may be too many pairs of items to test for similarity.
- ▶ *Locality-Sensitive Hashing* (LSH) compares fewer pairs that are likely to be similar, avoiding the quadratic cost of exhaustive comparisons.
- ▶ In general, items are thrown into buckets using several different “hash functions.” Only pairs of items sharing a bucket for at least one of these hashings are compared.
- ▶ **Upside**: only a small fraction of pairs are ever examined.
- ▶ **Downside**: *false negatives* (pairs never compared)

The Big Picture



Shingling

- ▶ 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 .

Choosing a suitable value k

- ▶ **Intuition:** want enough possible shingles that most docs do not contain most shingles.
- ▶ If k is too small, then most sequences of k characters will appear in most documents, so their shingle-sets will have high similarity, even if they had none of the same sentences or phrases.
- ▶ How large k should be depends on how long typical documents are and how large the set of typical characters is.
- ▶ k should be picked large enough that the probability of any shingle appearing in any given document is low.

A suitable value k for emails

- ▶ For emails, picking $k = 5$ should be fine. In fact, if only letters and white-space character appear in emails then there would be $27^5 = 14,348,907$ possible shingles.
- ▶ Since typical email is much smaller than 14 million characters long, we would expect $k = 5$ to work well.
- ▶ In practice, more than 27 characters appear in emails, and with different probability. Common letters and blanks dominate w.r.t. "z" and some other letters.
- ▶ Thus, even short emails will have many 5-shingles of common letters, and the chances of unrelated emails sharing such shingles is greater than what said above.

Typical k values

- ▶ A good rule of thumb for emails is to imagine that they contain only 20 different characters and estimate the number of k-shingles as 20.
- ▶ For large documents, such as research articles, choice $k = 9$ is considered safe.
- ▶ In practice, for most other types of documents a value $k = 8, 9$, or 10 is often used.

Shingle size for News Articles

- ▶ For the problem of finding similar news articles, it was found that a suitable shingle would be a stop word followed by the next two words, regardless of whether or not they were stop words.
- ▶ The advantage is that news article would contribute more shingles to the set representing the Web page than the surrounding elements.
- ▶ By biasing the set of shingles in favor of the article, Web pages with the same article and different surrounding material have higher similarity.

Hashing Shingles

- ▶ To save space but still make each shingle rare, we can hash them. We can pick a hash function that maps strings of length k to some number of buckets and treat the resulting bucket number as the shingle.
- ▶ A document is then represented by the set of bucket numbers, called **tokens**, of the k -shingles appearing in it.
- ▶ We could map 9-shingles to one of $2^{32} - 1$ bucket numbers. Thus, each shingle is represented by 4 bytes instead of 9, and we can manipulate (hashed) shingles by single-word machine operations.
- ▶ Two documents could (rarely) appear to have shingles in common, when only the hash-values were shared.

Collaborative Filtering

- ▶ Collaborative Filtering is another class of applications where similarity of sets is important.
- ▶ It is a process to recommend users items that were liked by other users with similar tastes.
- ▶ Examples:
 - ▶ Online purchases
 - ▶ Movie Ratings

Online Purchases

- ▶ The database of an online retailer like [Amazon](#) records items bought by customers.
- ▶ Two customers can be considered similar if their sets of purchased items have a high similarity. Likewise, two items that have sets of purchasers with high similarity will be deemed similar.
- ▶ **NB:** while mirror sites might have high similarity values, whereas a low value might still be unusual enough to identify customers with similar tastes.
- ▶ [Jaccard similarity](#) need not be very high to be significant.

Movie Ratings

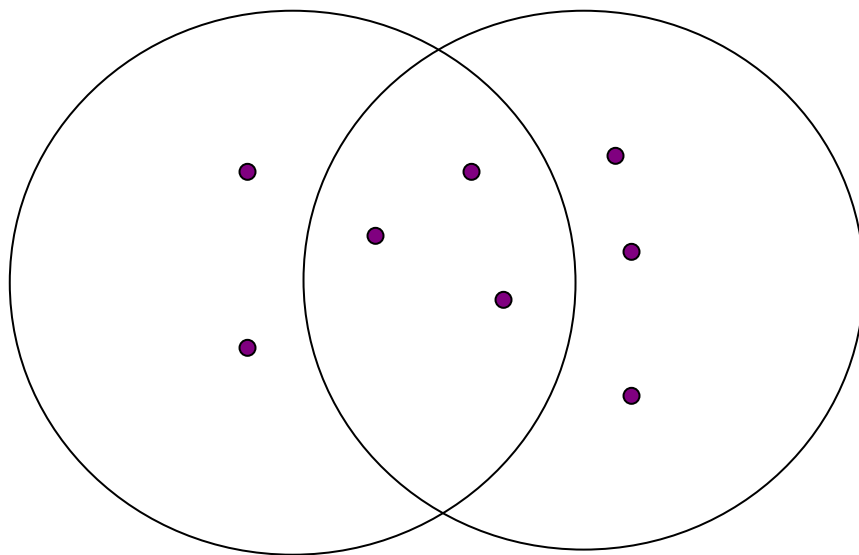
- ▶ Netflix records which movies its customers rented, and the ratings they assigned to them.
- ▶ Movies can be seen as similar if they were rented or rated highly by many of the same customers.
- ▶ Likewise, customers can be seen as similar if they rented or rated highly many of the same movies.
- ▶ Like for Amazon, similarities need not be high to be significant, and clustering movies by genre will make things easier.

Minhashing

Jaccard Similarity Measure
Constructing Signatures

Jaccard Similarity

- ▶ The *Jaccard similarity* of 2 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 = $3/8$

Similarity preserving Set summary

- ▶ Sets of **shingles** are large. Even if we **hash** them to few bytes each, the space needed to store a set is still about four times the space taken by the document.
- ▶ If we have millions of documents, it might not be possible to store all the shingle-sets in main memory.
- ▶ Our goal is to replace sets by compact representations called “**signatures**” in such a way that comparing two signatures provides an estimation of the **Jaccard similarity** of the underlying sets.
- ▶ The larger the signatures the more accurate will be the estimation.

From Sets to Boolean Matrices

For several computations it is helpful to visualize a collection of sets as their **characteristic matrix**:

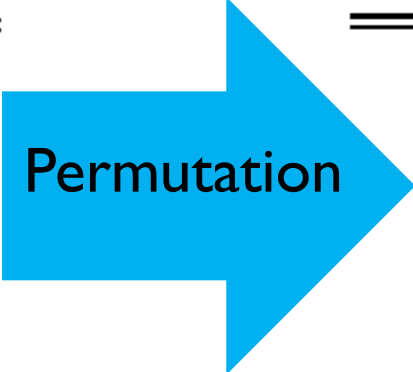
- ▶ The **columns** of the matrix correspond to the sets, and the **rows** correspond to **elements** of the universal set.
 - ▶ **Examples**: the set of all k-shingles or all tokens.
- ▶ 1 in row e and column S if and only if e is a member of S ; else 0.
- ▶ **Column similarity** is the **Jaccard similarity** of the sets of their rows with 1.
- ▶ Typical matrix is sparse.

Set Signatures

- ▶ The **set signatures** we want to construct are composed of the results of a large number of calculations, each of which is a “**minhash**” of the **characteristic matrix**.
- ▶ To **minhash** a set represented by a **column** of the **characteristic matrix**, pick a **permutation** of the rows.
- ▶ The **minhash value** of a column is the **first row number**, in the permuted order, **in which the column has a 1**.
- ▶ This permutation defines a minhash function **h** that maps sets to rows.
- ▶ Although it is not physically possible to permute very large characteristic matrices, the minhash function **h** implicitly reorders the rows of the matrix.
- ▶ We can read the values of **h** by scanning the matrix from the top until we find a 1.

Minhash Example

<i>Element</i>	S_1	S_2	S_3	S_4
<i>a</i>	1	0	0	1
<i>b</i>	0	0	1	0
<i>c</i>	0	1	0	1
<i>d</i>	1	0	1	1
<i>e</i>	0	0	1	0



<i>Element</i>	S_1	S_2	S_3	S_4
<i>b</i>	0	0	1	0
<i>e</i>	0	0	1	0
<i>a</i>	1	0	0	1
<i>d</i>	1	0	1	1
<i>c</i>	0	1	0	1

$$h(S_1) = h(S_4) = a; h(S_2) = c; h(S_3) = b;$$

Minhashing and Jaccard Similarity

- ▶ There is a remarkable connection between **minhashing** and **Jaccard similarity** of minhashed sets.
- ▶ The **probability** that the **minhash** function for a random permutation of rows produces the same value for two sets equals the **Jaccard similarity** of those sets.
- ▶ To see why, rows can be divided into three classes:
 1. Type X rows having 1 in both columns.
 2. Type Y rows having 1 in one of the columns and 0 in the other.
 3. Type Z rows having 0 in both columns.
- ▶ Since the matrix is sparse, most rows will be of type Z. But the ratio of types X and Y rows will determine $\text{SIM}(S_1, S_2)$ and the probability that $h(S_1) = h(S_2)$.

Minhashing and Jaccard Similarity(2)

- ▶ Let x be the number of rows of type X and y those of type Y
- ▶ Then $\text{SIM}(S_1, S_2) = x / (x + y)$
- ▶ The reason is that x is the size of $S_1 \cap S_2$ and $(x + y)$ is the size of $S_1 \cup S_2$.
- ▶ Now, consider the probability that $h(S_1) = h(S_2)$. If we imagine rows permuted randomly, and we proceed from the top, the probability that we shall meet a type X row before a type Y is $x / (x + y)$.
- ▶ But if the first row from the top other than type Z is a type X row, then surely $h(S_1) = h(S_2)$. If it is a type Y , then the set with a 1 gets that row as its minhash value.

Minhashing and Jaccard Similarity(3)

- ▶ However the set with a 0 in that row surely gets some row further down the permuted list.
- ▶ Thus, if we first meet a type Y row, then $h(S_1) \neq h(S_2)$.
- ▶ We conclude the probability that $h(S_1) = h(S_2)$ is $x / (x + y)$, which is also the Jaccard similarity of S_1 and S_2 .

Example: Column Similarity

C₁—C₂

0 1 *

1 0 *

1 1 * *

0 0

1 1 * *

0 1 *

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

Signature Matrix

- ▶ 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 $h_1, h_2 \dots h_n$ to create a *signature* for each column S as the vector $[h_1(S), h_2(S), \dots h_n(S)]$.
- ▶ Result is a *signature matrix*.
- ▶ Even if M is in some compressed form, it is normal for the *signature matrix* to be much smaller than M .

Example: Minhashing

1	0	1	1	0
2	0	0	1	1
3	1	0	0	0
4	0	1	0	1
5	0	0	0	1
6	1	1	0	0
7	0	0	1	0

Input Matrix

3	1	1	2
---	---	---	---

Signature Matrix

Example: Minhashing

7	1	0	1	1	0
6	2	0	0	1	1
5	3	1	0	0	0
4	4	0	1	0	1
3	5	0	0	0	1
2	6	1	1	0	0
1	7	0	0	1	0

Input Matrix

3	1	1	2
2	2	1	3

Signature Matrix

Example: Minhashing

6	7	1	0	1	1	0
3	6	2	0	0	1	1
1	5	3	1	0	0	0
7	4	4	0	1	0	1
2	3	5	0	0	0	1
5	2	6	1	1	0	0
4	1	7	0	0	1	0

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

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.
- ▶ **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.

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 approximates 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.
- ▶ The necessary sorting of the rows would take even more time.
- ▶ Representing a random permutation requires 1 billion entries.
- ▶ Accessing rows in permuted order leads to thrashing.

Implementation through Hashing

- ▶ Fortunately, it is possible to simulate a random permutation by a random **hash** function **h** mapping row numbers to as many buckets **k** as there are rows.
- ▶ A good approximation to permuting rows: pick, say, **100 hash functions**.
- ▶ A hash function will map some pairs of rows to the same bucket and leave some others unfilled. However, as long as **k** is large and there are few collisions, we can pretend that **h** “permutes” row **r** to position **$h(r)$** .
- ▶ Instead of picking **n random permutations** of rows, we pick **n** randomly chosen hash functions **h_1, h_2, \dots, h_n** .

Signature through Hashing

- ▶ Let $SIG(i, C)$ be the element of the signature matrix for the i -th hash function and column C .
- ▶ $SIG(i, C)$ will be the smallest value of $h_i(r)$ for which column C has 1 in row r .
- ▶ Initially, set $SIG(i, C)$ to ∞ for all i and C .
- ▶ For each row r compute $h_1(r), h_2(r), \dots, h_n(r)$
- ▶ For each column C do
 - ▶ If C has a 0 in row r do nothing
 - ▶ Otherwise, for each $i = 1, 2, \dots, n$ set $SIG(i, C)$ to the smallest value between $SIG(i, C)$ and $h_i(r)$.

Hash based Signature Algorithm

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 $SIG(i, C)$ **then**

$SIG(i, C) := h_i(r)$;

end;

Important: so you hash r only once per hash function, not once per 1 in row r .



Example

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

- (1) Row 1 has 1 in C1
 $h(1)=1 < \text{SIG}(h, C1)=\infty$
 $g(1)=3 < \text{SIG}(g, C1)=\infty$

	C1	C2
h	1	∞
g	3	∞

- (3) Row 3 has 1 in C1 and C2
 $h(3)=3 > \text{SIG}(h, C1)=1$
 $h(3) > \text{SIG}(h, C2)=2$
 $g(3)=2 < \text{SIG}(g, C1)=3$
 $g(3) > \text{SIG}(g, C2)=0$

	C1	C2
h	1	2
g	2	0

$$h(x) = x \bmod 5$$

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

- (2) Row 2 has 1 in C2
 $h(2)=2 < \text{SIG}(h, C2)=\infty$
 $g(2)=0 < \text{SIG}(g, C2)=\infty$

- (4) Row 4 has 1 in C1
 $h(4)=4 > \text{SIG}(h, C1)=1$
 $g(4)=4 > \text{SIG}(g, C1)=2$

Signature Matrix Unchanged

Initial Signature Matrix:

Row	C1	C2
h	∞	∞
g	∞	∞

	C1	C2
h	1	2
g	3	0

- (5) Row 5 has 1 in C2
 $h(5)=0 < \text{SIG}(h, C2)=2$
 $g(5)=1 > \text{SIG}(g, C2)=0$

	C1	C2
h	1	0
g	2	0

$\text{SIM}(C1, C2)=1/5$ whereas $\text{SIM}(\text{SIG}(C1), \text{SIG}(C2))=0$
 For bigger matrices the estimation gets closer !!

Explosive column combinations

- ▶ The **number of pairs of documents** may still be too large, even for few documents.
- ▶ Example: 1 million documents, signatures 250 long, hence 1000 bytes per document for the signatures (**signature element = row # or token #, about 4 bytes**).
- ▶ The entire data fits in a gigabyte – less than a typical main memory of a laptop.
- ▶ Thus, there are $\binom{1,000,000}{2}$ or half a trillion pairs of documents. If it takes a microsecond to compute the similarity of two signatures, then it takes almost six days to compute all the similarities on that laptop.

Explosive column combinations

- ▶ Parallelism can reduce time.
- ▶ However, often we want only the most similar pairs or pairs that are above some lower bound in similarity, without investigating every pair.
- ▶ There is a general theory of how to provide such focus, called **locality-sensitive hashing** (LSH) or **near-neighbor search**.
- ▶ We shall consider a specific form of LSH, designed for the particular problem of documents, represented by **shingle-sets**, then **minhashed** to short signatures.

Locality-Sensitive Hashing

**Focusing on Similar Minhash
Signatures**

Other Applications Will Follow

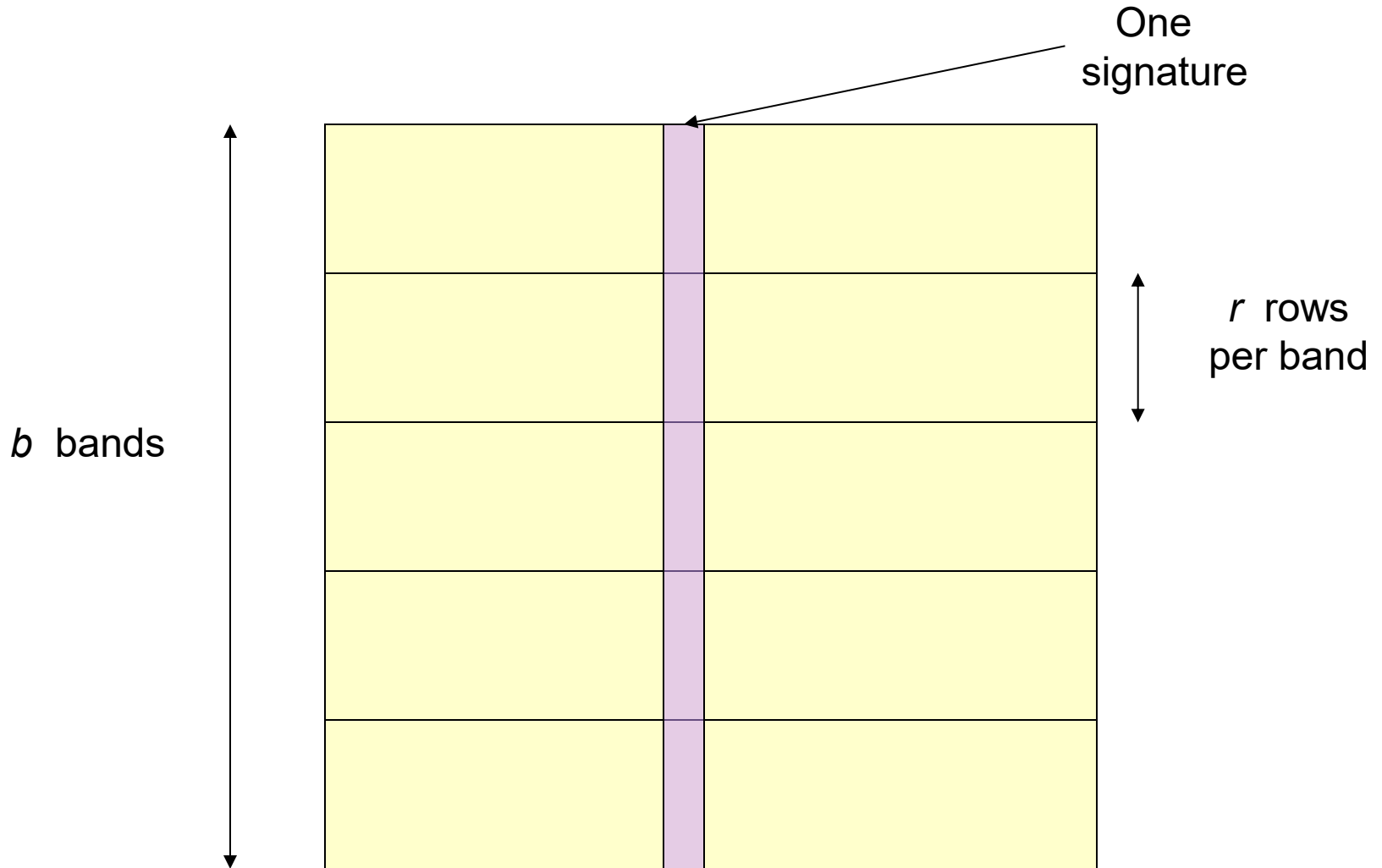
From Signatures to Buckets

- ▶ LSH prescribes to “hash” items several times, so that similar items are more likely to fall in the same bucket.
- ▶ We check for similarity only candidate pairs that hashed to the same bucket for any of the hashings.
- ▶ Define “similar” by a similarity *threshold* t = fraction of rows in which signatures must agree.
- ▶ Those dissimilar pairs that hash to the same bucket are *false positives*.
- ▶ Those similar pairs that do not hash to the same bucket are *false negatives*.

Partition into Bands

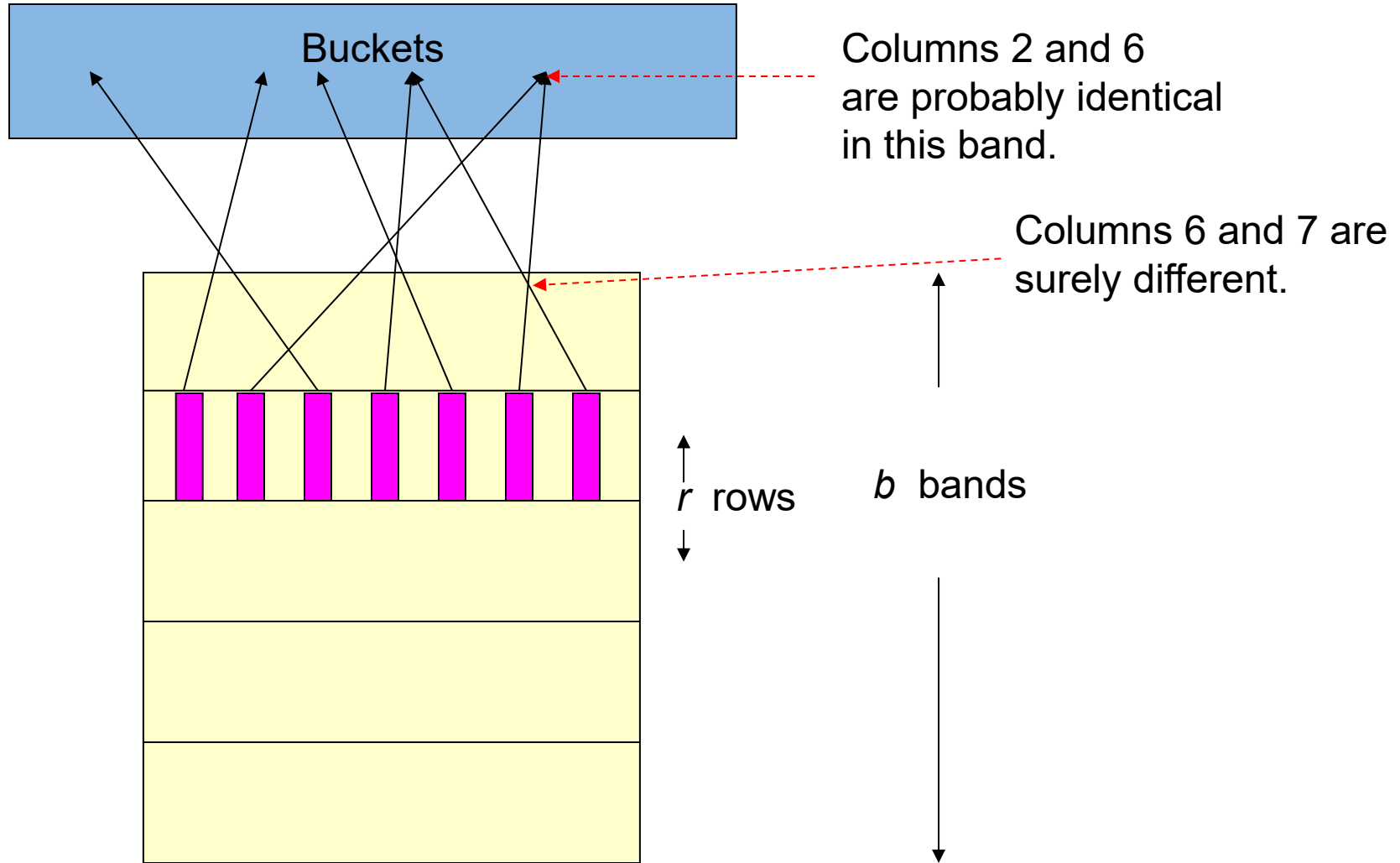
- ▶ Divide matrix M into b bands of r rows.
- ▶ For each band, hash its portion of each column to a hash table with k buckets.
 - ▶ Make k as large as possible.
 - ▶ Use a different hash table for each band.
- ▶ *Candidate* column pairs are those that hash to the same bucket for ≥ 1 band.
- ▶ Tune b and r to catch most similar pairs, but few nonsimilar pairs.

Partition Into Bands



Matrix M

Hash Function for One Bucket



Analysis of Banding Technique

- ▶ Let b the number of bands, r the number of rows in a band, and s the Jaccard similarity of two documents, whose minhash signatures are $SIG(C_1)$, and $SIG(C_2)$.
- ▶ Recall that the probability that $SIG(C_1)$, and $SIG(C_2)$ agree on any row is s . Thus, the probability that C_1 and C_2 are candidate pairs to compare can be computed as follows:
 - ▶ probability their signatures agree in all the rows of a band is s^r ;
 - ▶ probability they disagree in at least one row of a band is $1-s^r$;
 - ▶ probability they disagree in at least one row of each of the bands is $(1-s^r)^b$;
 - ▶ probability they agree in all the rows of at least one band and therefore become a candidate pair, is $1-(1-s^r)^b$;

Example: Bands

- ▶ Suppose 100,000 columns.
- ▶ Signatures of 100 integers.
- ▶ Therefore, all signatures take 40Mb (each signature element is a 4 byte row/token).
- ▶ Want all 80%-similar pairs.
- ▶ $\binom{100,000}{2} \approx 5,000,000,000$ pairs of signatures can take a while to compare.
- ▶ Choose 20 bands of 5 integers/band. Let us compute the probability that two documents are candidate to be compared, based on their Jaccard similarity.

Suppose C_1, C_2 are 80% Similar

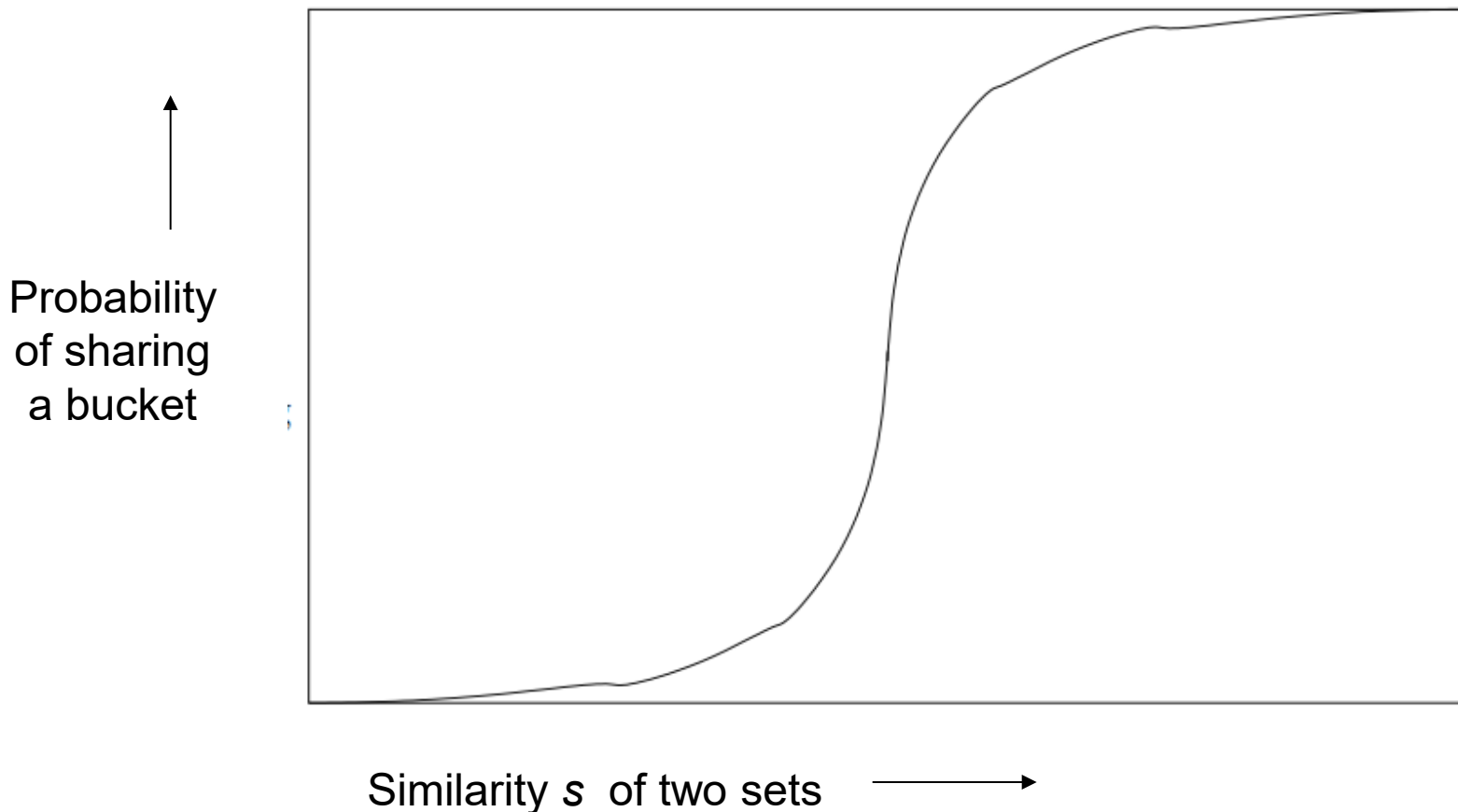
- ▶ Probability C_1, C_2 identical in one particular band: $s^r = (0.8)^5 = 0.328$.
- ▶ Probability C_1, C_2 are *not* similar in any of the 20 bands: $(1 - s^r)^b = (1 - 0.328)^{20} = .00035$.
- ▶ i.e., about 1/3000th of the 80%-similar underlying sets are false negatives.

Suppose C_1, C_2 Only 40% Similar

- ▶ Probability C_1, C_2 identical in any one particular band: $s^r = (0.4)^5 = 0.01$.
- ▶ Probability C_1, C_2 identical in ≥ 1 of 20 bands and hence are candidate pair to be compared:
 $1 - (1 - s^r)^b = 1 - (1 - 0.01)^{20} = 1 - (0.99)^{20} < 0.2$.
- ▶ But false positives much lower for similarities $\ll 40\%$.

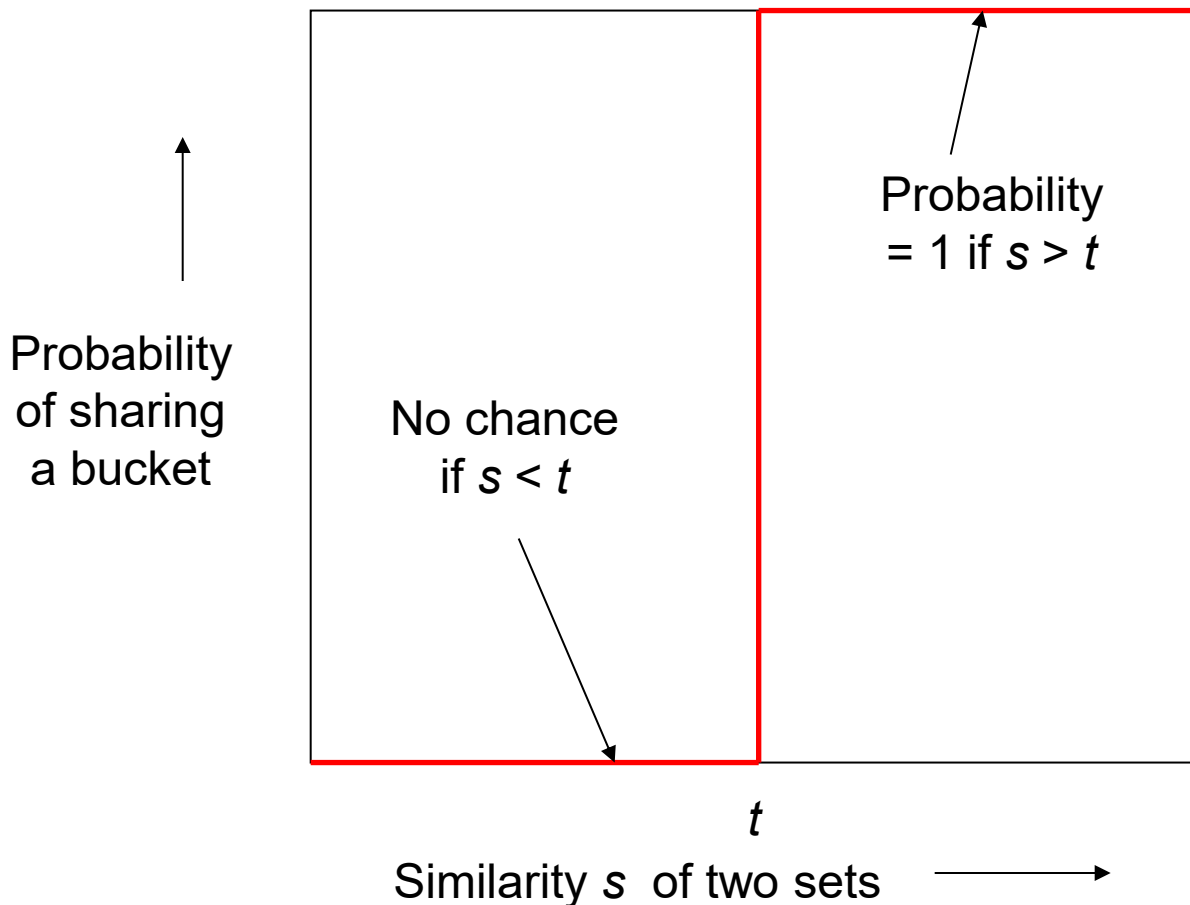
Analysis of LSH

- ▶ Regardless of b and r , this function has the form of an S-curve



Analysis of LSH – What We Want

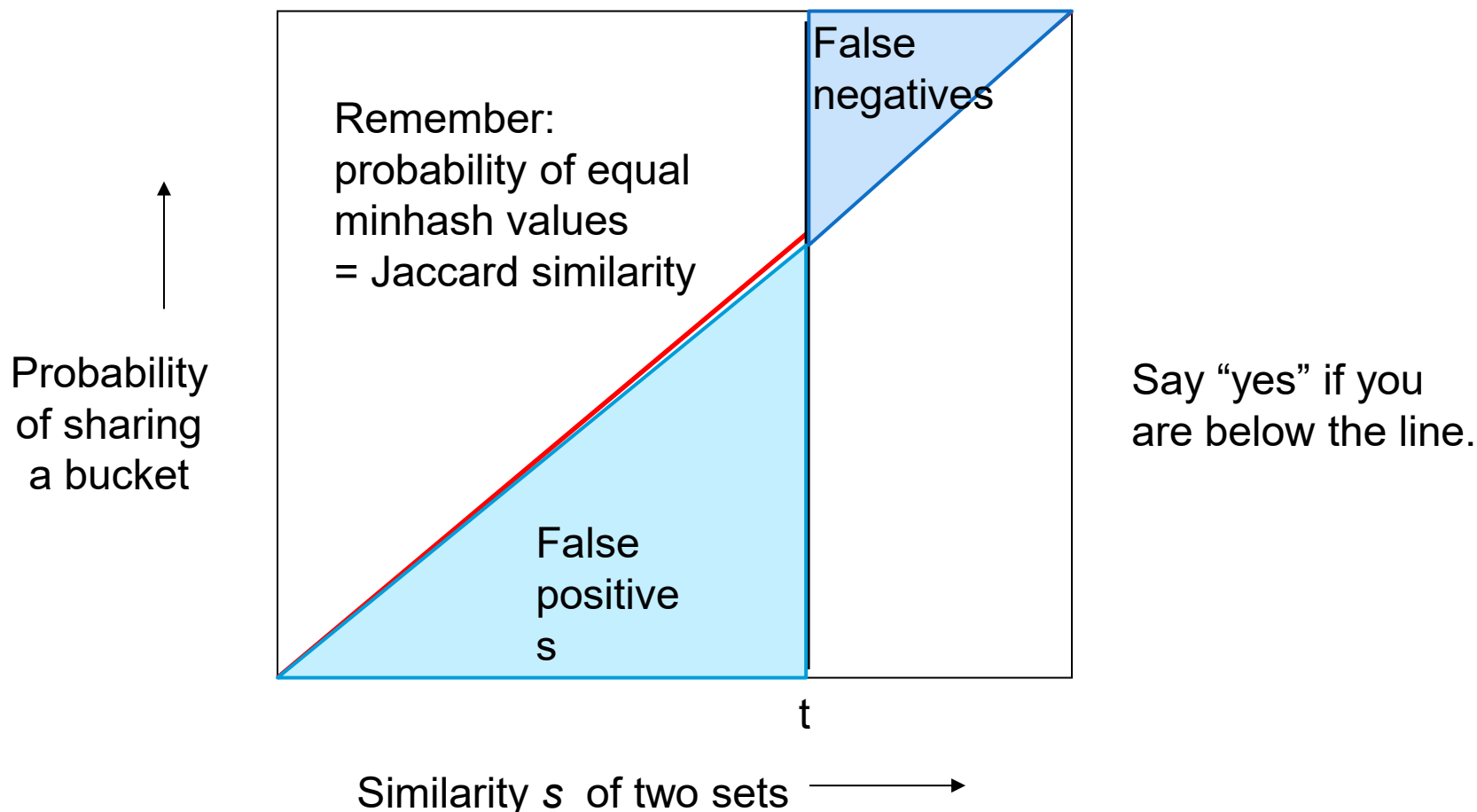
The threshold value t of similarity s at which the probability of becoming a candidate is $\frac{1}{2}$ is function of b and r .



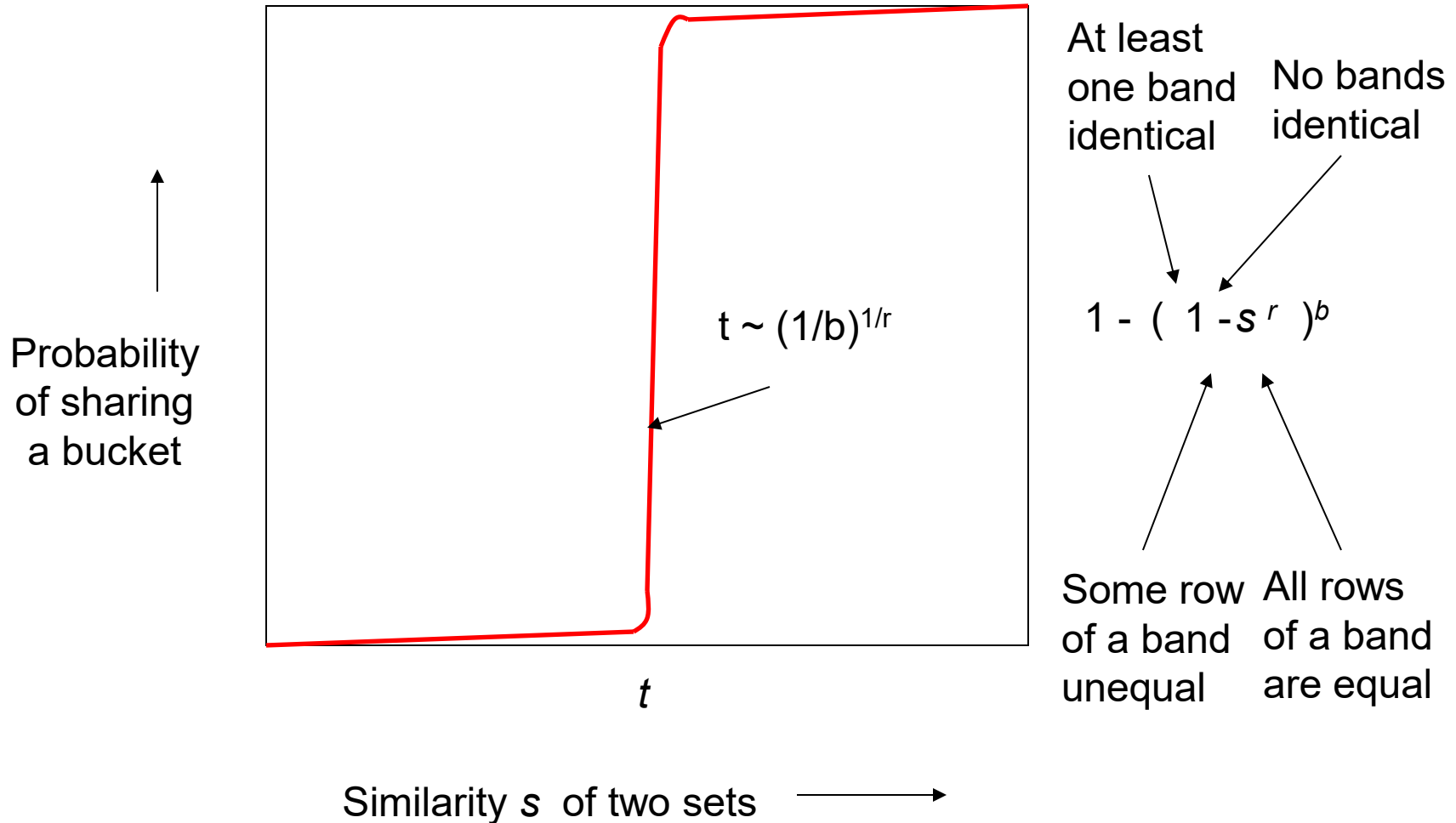
For large b and r we find that pairs with similarity above the threshold are very likely to become candidates, while those below are unlikely. **This is what we want, a step function !**

What One Band of One Row Gives You

- ▶ A single row of the signature matrix gives us a straight line, with lot of **false positives** and **false negative**



What b Bands of r Rows Gives You



Example: $b = 20$; $r = 5$; $(1/b)^{1/r} \sim 0.55$

- ▶ Not exactly a **step function**, but much better than the linear function we get from a single row.

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



Slope here
about 2.3

LSH for Documents: Summary

- ▶ Choose k and construct k -shingles from each document. Optionally, hash k -shingles to shorter bucket numbers.
- ▶ Sort document-shingle pairs to order them by shingle.
- ▶ Pick a length n for minhash signatures and compute minhash signatures for all the documents.
- ▶ Choose a threshold t defining how similar documents should be to be regarded as a “candidate pair.”
- ▶ Pick a number of bands b and a number of rows r such that $b * r = n$, and t is approximately $(1/b)^{1/r}$.

LSH for Documents: Summary(2)

- ▶ To avoid false negatives choose b and r to lower t .
- ▶ To limit false positive and improve efficiency choose b and r to raise t .
- ▶ Construct candidate pairs through the LSH technique.
- ▶ Examine each candidate pair's signatures and determine whether the fraction of components in which they agree is at least t .
- ▶ Optionally, if the signatures are sufficiently similar, go to the documents and check that they are truly similar, rather than, by luck, had similar signatures.

Other Distance Measures

Definition of Distance Measure

- ▶ A **distance measure** on a **space** of points is a **function** $d(x, y)$ that takes two points in the space and returns a **real number**.
- ▶ It satisfies the following axioms:
 1. $d(x, y) \geq 0$ (no negative distances);
 2. $d(x, y) = 0$ if and only if $x = y$ (distances are positive, except for the distance from a point to itself);
 3. $d(x, y) = d(y, x)$ (distance is symmetric);
 4. $d(x, y) \leq d(x, z) + d(z, y)$ (**triangle inequality**. It also says that distance measures behave as the length of a shortest path from one point to another);

Euclidean Distances

- ▶ In an n -dimensional Euclidean space **points** are vectors of n real numbers.
- ▶ The distance measure in this space, referred to as the L_2 -norm, is defined as

$$d([x_1, x_2, \dots, x_n], [y_1, y_2, \dots, y_n]) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

- ▶ It is easy to verify the **four** requirements for a distance measure are satisfied.

Minkowsky and Manhattan Distances

- ▶ Like Euclidean distance, they are defined in an n -dimensional Euclidean space.
- ▶ Minkowsky distance, a.k.a L_r -norm, is defined as
$$d([x_1, x_2, \dots, x_n], [y_1, y_2, \dots, y_n]) = \sum_{i=1}^n (|x_i - y_i|^r)^{1/r}$$
- ▶ For $r = 1$ we get L_1 -norm, also known as Manhattan distance, since it corresponds to the distance when travelling along grid lines, as on Manhattan's streets.
- ▶ Another interesting distance measure is the L_∞ -norm which is the limit as r approaches infinity of L_r -norm
- ▶ As r gets larger, only the dimension with largest difference matters, so L_∞ -norm is defined as $\max |x_i - y_i|$.

Jaccard Distance

- ▶ Is defined as

$$d(x, y) = 1 - \text{SIM}(x, y)$$

- ▶ It is easy to show that this is a Distance function:

1. $d(x, y) \geq 0$, since intersection cannot exceed union;
2. $d(x, y) = 0$ if $x=y$. In fact, $x \cap y = x \cup y = x \Rightarrow 1 - x/x = 0$. Moreover, if $x \neq y \Rightarrow x \cap y$ is less than $x \cup y \Rightarrow d(x, y) > 0$.
3. $d(x, y) = d(y, x)$, since intersection and union are commutative.
4. $d(x, y) \leq d(x, z) + d(z, y)$ since $\text{SIM}(x, y) = \text{probability}(h(x) = h(y))$, hence $d(x, y) = \text{probability}(h(x) \neq h(y))$ for a random minhash function h . Thus, $\text{probability}(h(x) \neq h(y)) \leq \text{probability}(h(x) \neq h(z)) + \text{probability}(h(z) \neq h(y))$, since whenever $h(x) \neq h(y)$ at least one between $h(x)$ and $h(y)$ must be different from $h(z)$.

Cosine Distance

- ▶ Suitable for spaces with dimensions, including Euclidean spaces and their discrete version (i.e- spaces where points are vectors of integer or boolean components).
- ▶ We do not distinguish between a vector and its multiple.
- ▶ The cosine distance of two points is the angle formed by their vectors, expressed in the range 0 to 180 degrees.
- ▶ Given two vectors x and y , we first compute the cosine of their angle, and then apply the arc-cosine function to translate it to an angle in the range 0-180. The cosine is computed as the dot product $x \cdot y$ divided by L_2 -norms of x and y (their Euclidean distance from the origin).

Edit Distance

- ▶ This distance makes sense when points are **strings**.
- ▶ The distance between two strings $x = x_1x_2 \cdots x_n$ and $y = y_1y_2 \cdots y_m$ is the smallest number of insertions and deletions of single characters to convert x to y .
- ▶ The edit distance of 2 strings x and y can be computed as the length of x plus the length of y minus twice the length of their **Longest Common Subsequence (LCS)**.
- ▶ **Example:** The edit distance between the strings $x = abcde$ and $y = acfdeg$ is 3:
 1. Delete b.
 2. Insert f after c.
 2. Insert g after e.

Hamming Distance

- ▶ Given a space of vectors, **Hamming distance** between 2 vectors is the number of components in which they differ.
- ▶ **Hamming distance** is a distance measure:
 1. Clearly it cannot be negative;
 2. If it is 0, then the vectors are identical;
 3. It does not depend on which of 2 vectors we consider first.
 4. If x and z differ in m components, and z and y differ in n components, then x and y cannot differ in more than $m+n$ components.
- ▶ **Hamming distance** is used for Boolean vectors, even if the vectors can have components from any set.

Non Euclidean Spaces

- ▶ Some of the surveyed distance measures are for not Euclidean spaces.
- ▶ A property of Euclidean spaces (useful for clustering) is that the average of a set of points in a Euclidean space always exists and is a point in the space.
- ▶ For instance, the “average” of two sets makes no sense for the Jaccard distance, like the average of strings makes no sense for the space of strings of the edit distance.
- ▶ Vector spaces, for which we suggest the cosine distance, may or may not be Euclidean, depending on whether the vector components are real numbers or integers.

Applications of Locality-Sensitive Hashing

- ▶ **Entity Resolution**: It refers to matching data records that refer to the same real-world entity. The problem is that the similarity of records does not match exactly either the similar-sets or similar-vectors models of similarity.
- ▶ **Matching Fingerprints**: fingerprints can be represented as sets. It is possible to exploit different families of locality-sensitive hash functions from minhashing.
- ▶ **Matching Newspaper Articles**: Here, shingling focuses on the core article in a newspaper's Web page, ignoring extraneous material such as ads and other material.
- ▶ We will see Entity Resolution and Newspaper Articles.

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

- ▶ Given a consulting job for 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, we have previously seen that there are better methods.

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.

