Finding Similar Items

Similar Items

Problem.

- Search for pairs of items that appear together a large fraction of the times that either appears, even if neither item appears in very many baskets.
  - Such items are considered "similar"

Modeling

- Each item is a set: the set of baskets in which it appears.
  - Thus, the problem becomes: Find similar sets!
  - But, we need a definition for how similar two sets are.

The Jaccard Measure of Similarity

- The *similarity* of sets S and T is the ratio of the sizes of the intersection and union of S and T.
  - *Sim* $(C_1,C_2) = |S \cap T|/|S \cup T|$ = *Jaccard similarity*.

- Disjoint sets have a similarity of 0, and the similarity of a set with itself is 1.

- Another example: similarity of sets {1, 2, 3} and {1, 3, 4, 5} is
  - 2/5.

Applications - Collaborative Filtering

- Products are similar if they are bought by many of the same customers.
  - E.g., movies of the same genre are typically rented by similar sets of Netflix customers.
  - A customer can be pitched an item that is a similar to an item that he/she already bought.

Dual view

- Represent a customer, e.g., of Netflix, by the set of movies they rented.
  - Similar customers have a relatively large fraction of their choices in common.
  - A customer can be pitched an item that a similar customer bought, but that they did not buy.

Applications: Similar Documents (1)

- Given a body of documents, e.g., Web pages, find pairs of docs that have a lot of text in common, e.g.:
  - Mirror sites, or approximate mirrors.
  - Plagiarism, including large quotations.
  - Repetitions of news articles at news sites.

- How do you represent a document so it is easy to compare with others?
  - Special cases are easy, e.g., identical documents, or one document contained verbatim in another.
  - General case, where many small pieces of one doc appear out of order in another, is hard.

Applications: Similar Documents (1)

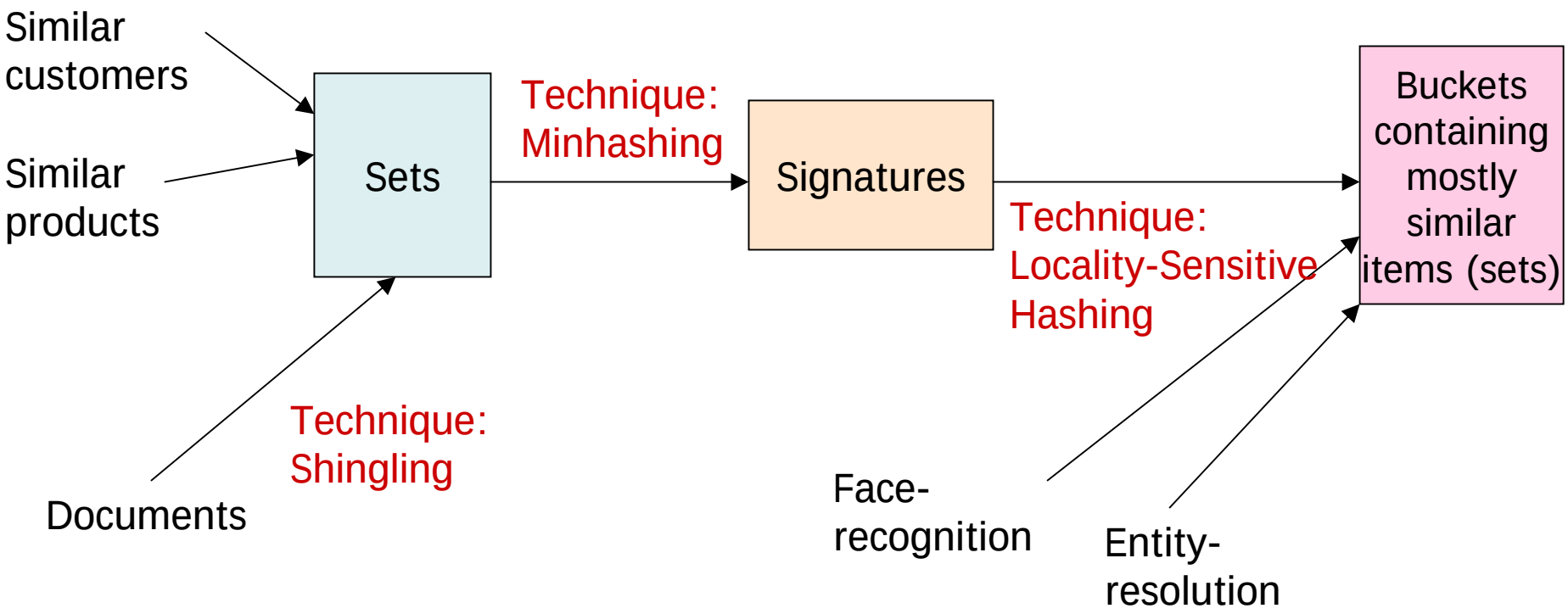
- Represent doc by its set of shingles (or *k* -grams).
- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* characters that appears in the document.

Example.

- *k*=2; doc = abcab.
- Set of 2-shingles = {ab, bc, ca}.

- At that point, doc problem becomes finding similar sets.

Roadmap


Similar customers
Similar products
Documents
Sets
Technique: Shingling
Technique: Minhashing
Signatures
Technique: Locality-Sensitive Hashing
Face-recognition
Entity-resolution
Buckets containing mostly similar items (sets)

Minhashing

- Suppose that the elements of each set are chosen from a "universal" set of n elements $e_0, e_1,\ldots,e_{n-1}$.
- Pick a random permutation of the n elements.
- Then the minhash value of a set S is the first element, in the permuted order, that is a member of S.

Example

- Suppose the universal set is {1, 2, 3, 4, 5} and the permuted order we choose is (3,5,4,2,1).
  - Then, the hash value of set {2, 3, 5} is 3.
  - Set {1, 2, 5} hashes to 5.
  - For another example, {1,2} hashes to 2, because 2 appears before 1 in the permuted order.

Minhash signatures

- We compute *signatures* for the sets by picking a list of *m* permutations of all the possible elements.
  - Typically, *m* would be about 100.
- The signature of a set *S* is the list of the minhash values of *S*, for each of the *m* permutations, in order.

Example

- Universal set is {1,2,3,4,5}, *m* = 3, and the permutations are:
  - π_1= (1,2,3,4,5),
  - π_2= (5,4,3,2,1),
  - π_3= (3,5,1,4,2).
- Signature of S = {2,3,4} is
  - (2,4,3).

Minhashing and Jaccard Distance

Surprising relationship

If we choose a permutation at random, the probability that it will produce the same minhash values for two sets is the same as the Jaccard similarity of those sets.

- Thus, if we have the signatures of two sets S and T, we can estimate the Jaccard similarity of S and T by the fraction of corresponding minhash values for the two sets that agree.

Example

- Universal set is {1,2,3,4,5}, m = 3, and the permutations are: π_1= (1,2,3,4,5), π_2= (5,4,3,2,1), π_3= (3,5,1,4,2).
- Signature of S = {2,3,4} is (2,4,3).
- Signature of T = {1,2,3} is (1,3,3). Conclusion?

Implementing Minhashing

- Generating a permutation of all the universe of objects is infeasible.
- Rather, simulate the choice of a random permutation by picking a random hash function *h*.
  - We pretend that the permutation that *h* represents places element *e* in position *h(e).*
  - Of course, several elements might wind up in the same position.
    - As long as number of buckets is large, we can break ties as we like, and the simulated permutations will be sufficiently random that the relationship between signatures and similarity still holds.

Algorithm for minhashing

- To compute the minhash value for a set $S = \{a_1, a_2, \ldots, a_n\}$ using a hash function *h,* we can execute:

```
V =infinity;
FOR i := 1 TO n DO
    IF h(a_i) < V THEN V = h(a_i);
```

- As a result, V will be set to the hash value of the element of *S* that has the smallest hash value.

Algorithm for set signature

- If we have *m* hash functions $h_1, h_2, \ldots, h_m$, then we can compute *m* minhash values in parallel, as we process each member of *S.*

```
FOR j := 1 TO m DO
    V_j := infinity;
FOR i := 1 TO n DO
    FOR j := 1 TO m DO
      IF h_j(a_i) < V_j THEN V_j = h_j(a_i)
```

Example

S = {1,3,4}
T = {2,3,5}

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

$h(1) = 1$
$g(1) = 3$

$h(2) = 2$
$g(2) = 0$

$h(3) = 3$
$g(3) = 2$

$h(4) = 4$
$g(4) = 4$

$h(5) = 0$
$g(5) = 1$

sig(S) = 1,3
sig(T) = 5,2

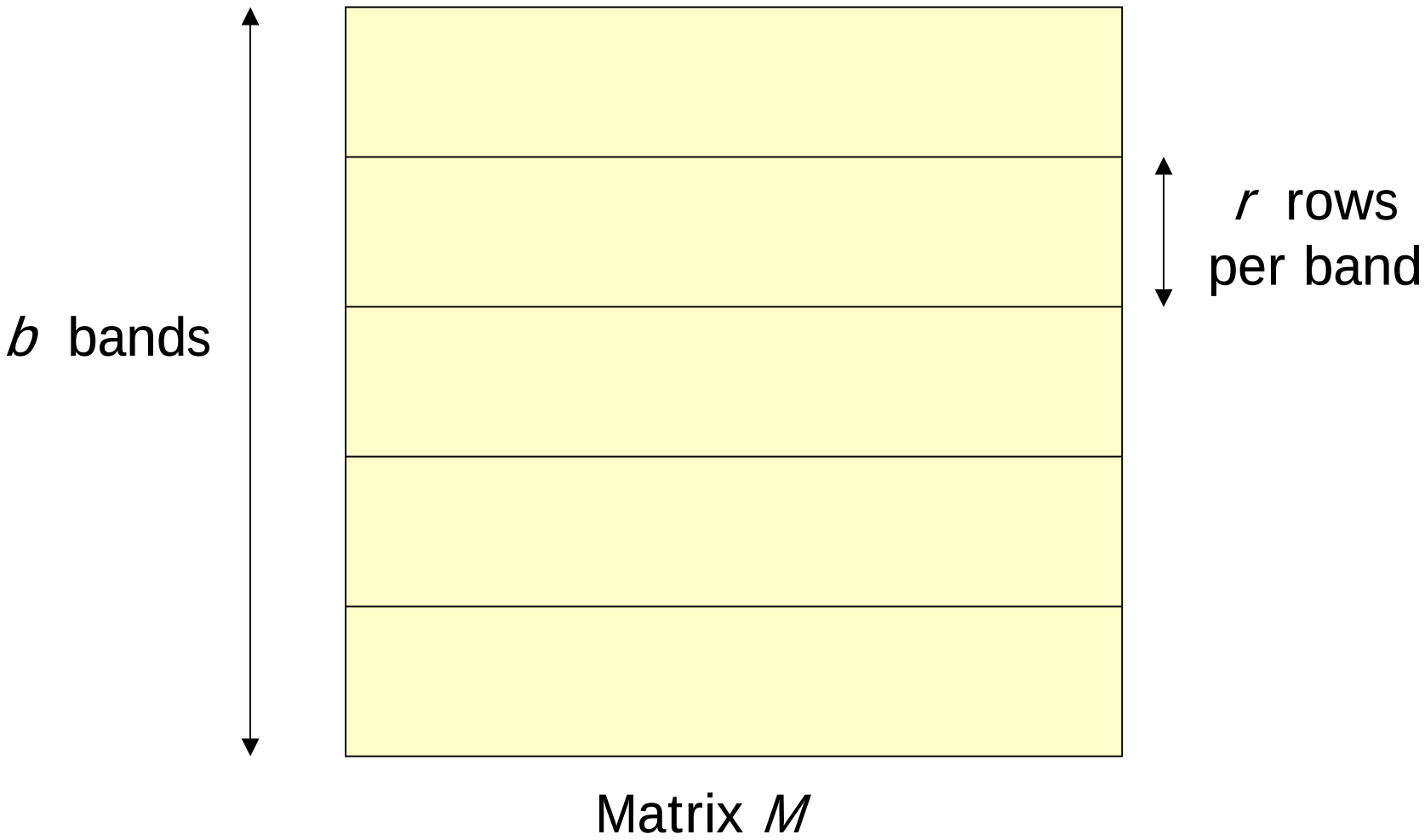
Locality-Sensitive Hashing of Signatures

- **Goal**: Create buckets containing mostly similar items (sets).
  - Then, compare only items within the same bucket.

- Think of the signatures of the various sets as a matrix *M*, with a column for each set's signature and a row for each hash function.

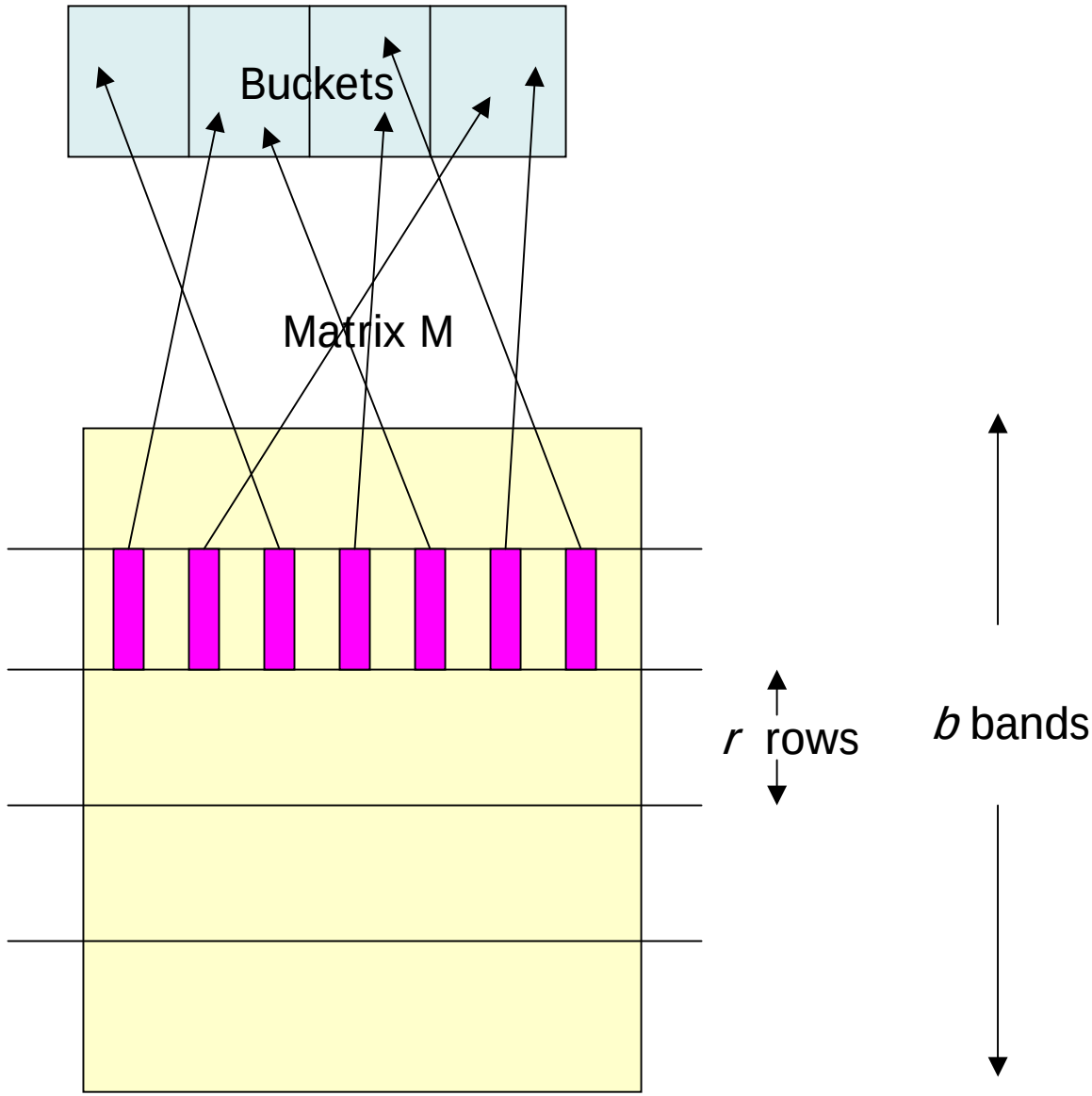
- Big idea: hash columns of signature matrix *M* several times.
- Arrange that (only) similar columns are likely to hash to the same bucket.
- Candidate pairs are those that hash at least once to the same bucket.

Partition Into Bands



Partition Into Bands

- For each band, hash its portion of each column to a hash table with *k* buckets.
- Candidate column pairs are those that hash to the same bucket for at least one band.



Analysis

- Probability that the signatures agree on one row is

  s (Jaccard similarity)
- Probability that they agree on all *r* rows of a given band is

  s^r.
- Probability that they do not agree on all the rows of a band is

  1 - *s^r*
- Probability that for none of the *b* bands do they agree in all rows of that band is

  (1 - *s^r*)^*b*
- Probability that the signatures will agree in all rows of at least one band is

  1 - (1 - *s^r*)^*b*
- This function is the probability that the signatures will be compared for similarity.

Example

- Suppose 100,000 columns.
- Signatures of 100 integers.
- Therefore, signatures take 40Mb.
- But 5,000,000,000 pairs of signatures 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 agree on one particular band:
  - $(0.8)^5 = 0.328$.

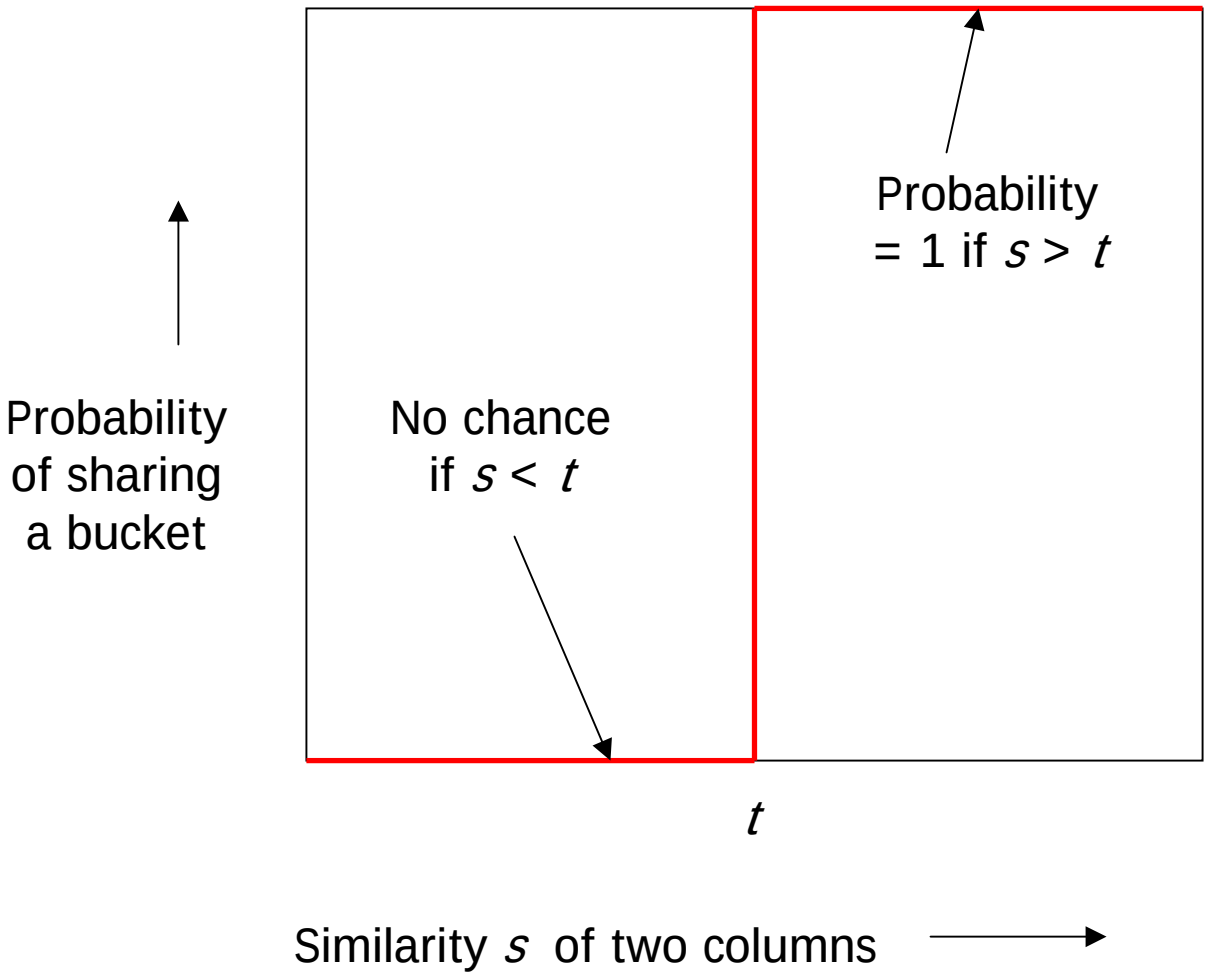
- Probability C_1, C_2 do *not* agree on any of the 20 bands:
  - $(1\text{-}0.328)^{20} = .00035$.
  - i.e., we miss about 1/3000th of the 80%-similar column pairs.

- The chance that we *do* find this pair of signatures together in at least one bucket is 1 - 0.00035,or 0.99965.

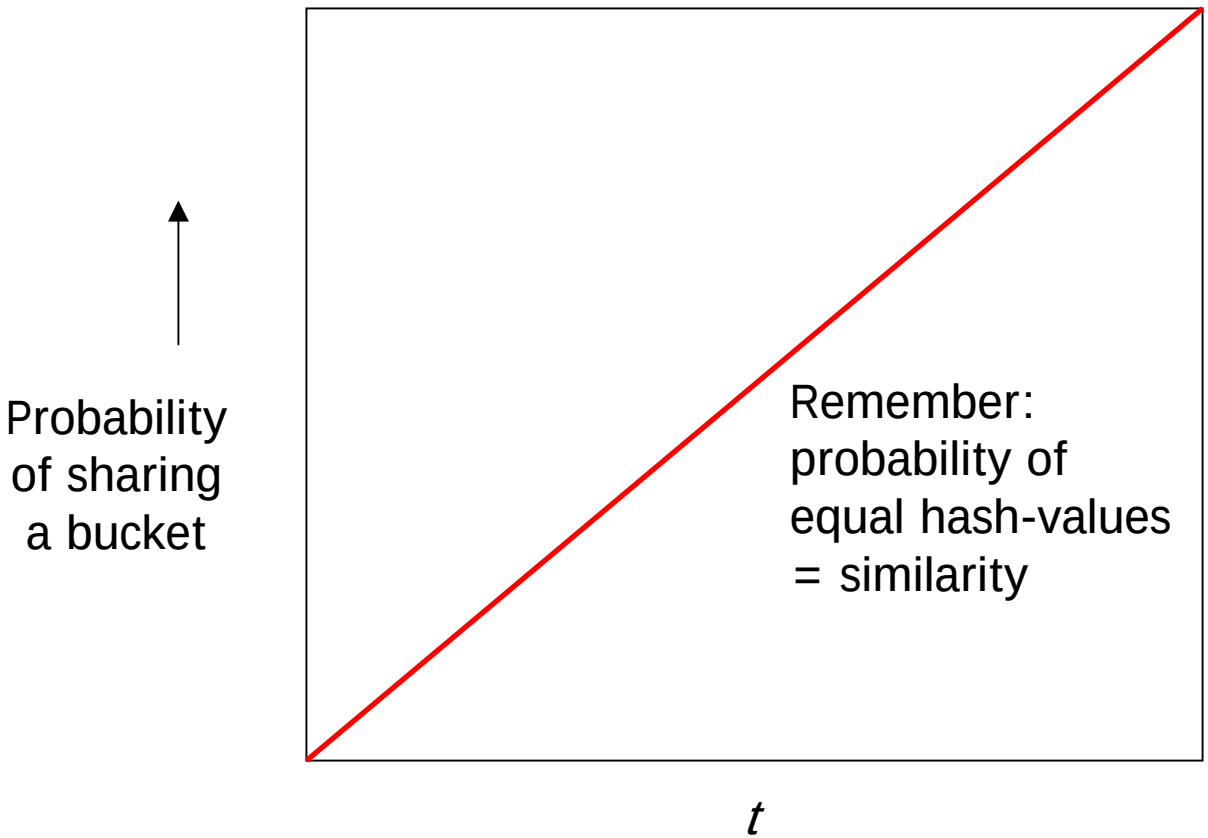
Suppose C_1, C_2 Only 40% Similar

- Probability C_1, C_2 agree on one particular band:
  - $(0.4)^5 = 0.01$.
- Probability C_1, C_2 do *not* agree on any of the 20 bands:
  - (1-0.01)^20 $\approx$ .80
  - i.e., we miss a lot...
- The chance that we *do* find this pair of signatures together in at least one bucket is 1 - 0.80,or 0.20 (i.e. only 20%).

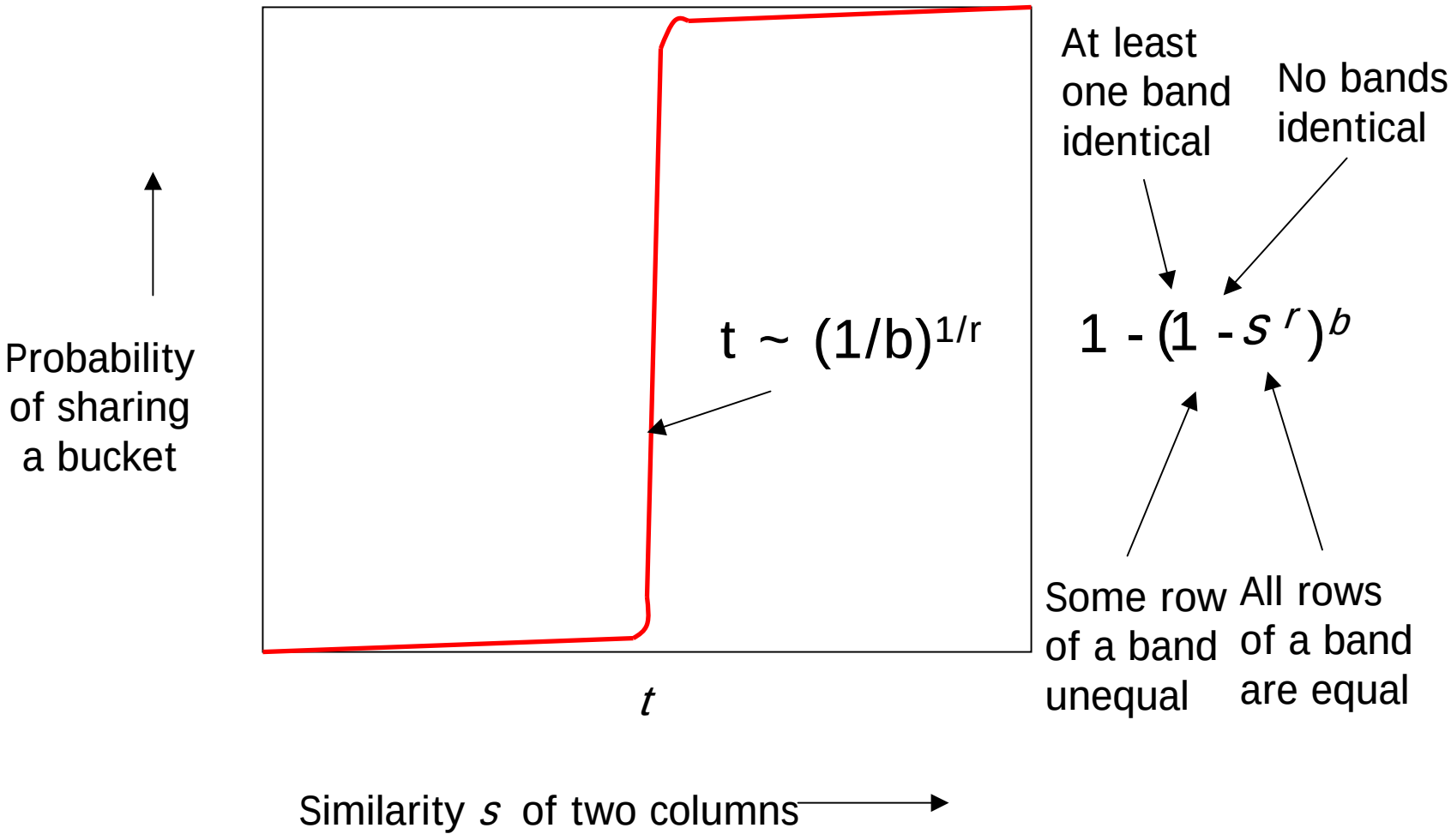
Analysis of LSH – What We Want



What One Row Gives You



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



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 are similar *columns* .