Similarity Analysis and Distance
Min-Hashing
Locality Sensitive Hashing

DATA MINING

Thanks to:
Tan, Steinbach, and Kumar, "Introduction to Data Mining"
Rajaraman and Ullman, "Mining Massive Datasets"

Similarity and Distance

- For many different problems we need to quantify how close two objects are.
- Examples:
  - For an item bought by a customer, find other similar items
  - Group together the customers of site so that similar customers are shown the same ad.
  - Group together web documents so that you can separate the ones that talk about politics and the ones that talk about sports.
  - Find all the near-duplicate mirrored web documents.
  - Find credit card transactions that are very different from previous transactions.
- To solve these problems we need a definition of similarity, or distance.
  - The definition depends on the type of data that we have

Similarity

- Numerical measure of how alike two data objects are.
  - A function that maps pairs of objects to real values
  - Higher when objects are more alike.
- Often falls in the range [0,1], sometimes in [-1,1]

- Desirable properties for similarity
  1. s(p, q) = 1 (or maximum similarity) only if p = q. (Identity)
  2. s(p, q) = s(q, p) for all p and q. (Symmetry)

Similarity between sets

- Consider the following documents

apple releases new ipod	apple releases new ipad	new apple pie recipe

- Which ones are more similar?
- How would you quantify their similarity?

Similarity: Intersection

- Number of words in common

apple releases new ipod

apple releases new ipad

new apple pie recipe

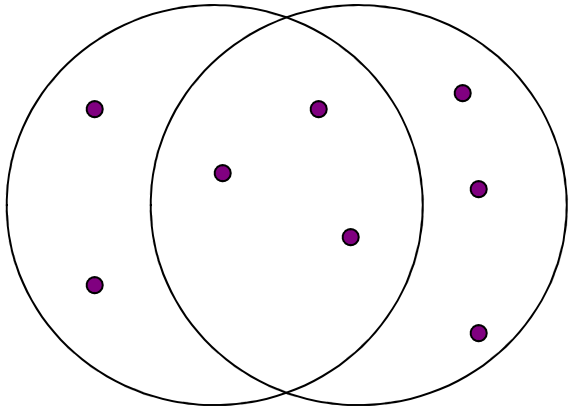
- Sim(D,D) = 3, Sim(D,D) = Sim(D,D) =2
- What about this document?

Vefa rereases new book with apple pie recipes

- Sim(D,D) = Sim(D,D) = 3

Jaccard Similarity

- The Jaccard similarity (Jaccard coefficient) of two sets S_1, S_2 is the size of their intersection divided by the size of their union.
  - JSim $(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$.



3 in intersection.
8 in union.
Jaccard similarity
= 3/8

  - Extreme behavior:
    - Jsim(X,Y) = 1, iff X = Y
    - Jsim(X,Y) = 0 iff X,Y have not elements in common
  - JSim is symmetric

Similarity: Intersection

- Number of words in common

apple releases new ipod

apple releases new ipad

new apple pie recipe

Vefa rereases new book with apple pie recipes

- JSim(D,D) = 3/5
- JSim(D,D) = JSim(D,D) = 2/6
- JSim(D,D) = JSim(D,D) = 3/9

Similarity between vectors

Documents (and sets in general) can also be represented as vectors

document	Apple	Microsoft	Obama	Election
D1	10	20	0	0
D2	30	60	0	0
D2	0	0	10	20

How do we measure the similarity of two vectors?

How well are the two vectors aligned?

Example

document	Apple	Microsoft	Obama	Election
D1	1/3	2/3	0	0
D2	1/3	2/3	0	0
D2	0	0	1/3	2/3

Documents D1, D2 are in the "same direction"
Document D3 is orthogonal to these two

Cosine Similarity

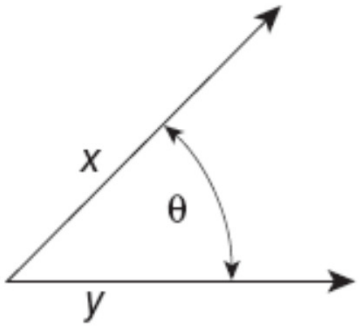


Figure 2.16. Geometric illustration of the cosine measure.

- Sim(X,Y) = cos(X,Y)
  - The cosine of the angle between X and Y
- If the vectors are aligned (correlated) angle is zero degrees and cos(X,Y)=1
- If the vectors are orthogonal (no common coordinates) angle is 90 degrees and cos(X,Y) = 0
- Cosine is commonly used for comparing documents, where we assume that the vectors are normalized by the document length.

Cosine Similarity - math

- If d_1 and d_2 are two vectors, then

  $$\cos(d_1, d_2) = (d_1 \bullet d_2) / \|d_1\| \|d_2\|,$$

  where $\bullet$ indicates vector dot product and $\| d \|$ is the length of vector d.

- Example:

d_1 = 3 2 0 5 0 0 0 2 0 0
d_2 = 1 0 0 0 0 0 0 1 0 2

$d_1 \bullet d_2$ = 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5

$\|d_1\|$ = (3*3+2*2+0*0+5*5+0*0+0*0+0*0+2*2+0*0+0*0)$^{0.5}$ = (42) $^{0.5}$ = 6.481

$\|d_2\|$ = (1*1+0*0+0*0+0*0+0*0+0*0+0*0+1*1+0*0+2*2) $^{0.5}$ = (6) $^{0.5}$ = 2.245

$\cos(d_1, d_2)$ = .3150

Similarity between vectors

document	Apple	Microsoft	Obama	Election
D1	10	20	0	0
D2	30	60	0	0
D2	0	0	10	20

cos(D1,D2) = 1

cos(D1,D3) = cos(D2,D3) = 0

Distance

- Numerical measure of how different two data objects are
  - A function that maps pairs of objects to real values
  - Lower when objects are more alike
- Minimum distance is 0, when comparing an object with itself.
- Upper limit varies

Distance Metric

- A distance function d is a distance metric if it is a function from pairs of objects to real numbers such that:
  1. $d(x,y) \geq 0$. (non-negativity)
  2. $d(x,y) = 0$ iff $x = y$. (identity)
  3. $d(x,y) = d(y,x)$. (symmetry)
  4. $d(x,y) \leq d(x,z) + d(z,y)$ (triangle inequality).

Triangle Inequality

- Triangle inequality guarantees that the distance function is well-behaved.
  - The direct connection is the shortest distance

- It is useful also for proving properties about the data
  - For example, suppose I want to find an object that minimizes the sum of distances to all points in my dataset
  - If I select the best point from my dataset, the sum of distances I get is at most twice that of the optimal point.

Distances for real vectors

- Vectors $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$
- L_p norms or Minkowski distance:

$$L_p(x, y) = [|x_1 - y_1|^p + \cdots + |x_d - y_d|^p]^{1/p}$$

- L_2 norm: Euclidean distance:

$$L_2(x, y) = \sqrt{|x_1 - y_1|^2 + \cdots + |x_d - y_d|^2}$$

- L_1 norm: Manhattan distance:

$$L_1(x, y) = |x_1 - y_1| + \cdots + |x_d - y_d|$$

- L_∞ norm:

$$L_\infty(x, y) = \max\{|x_1 - y_1|, \ldots, |x_d - y_d|\}$$

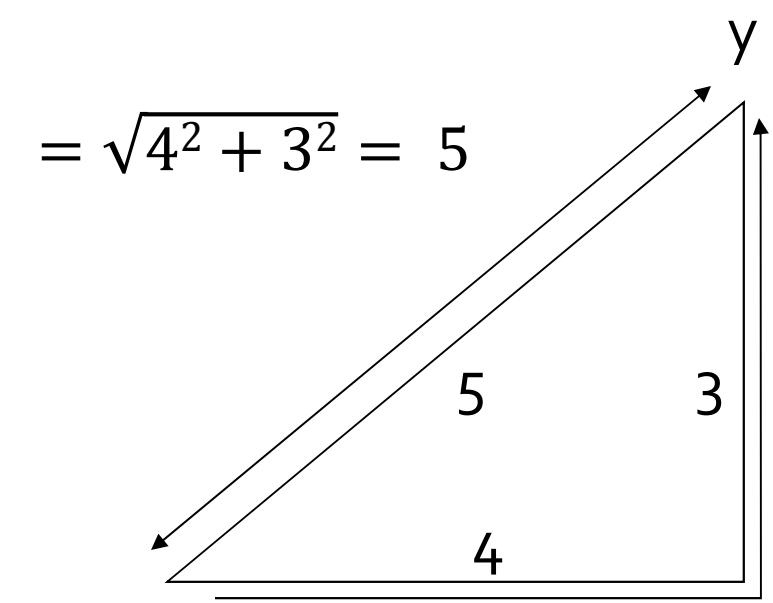
  - The limit of L_p as p goes to infinity.

L_p norms are known to be distance metrics

Example of Distances

L_2-norm:

$$dist(x, y) = \sqrt{4^2 + 3^2} = 5$$



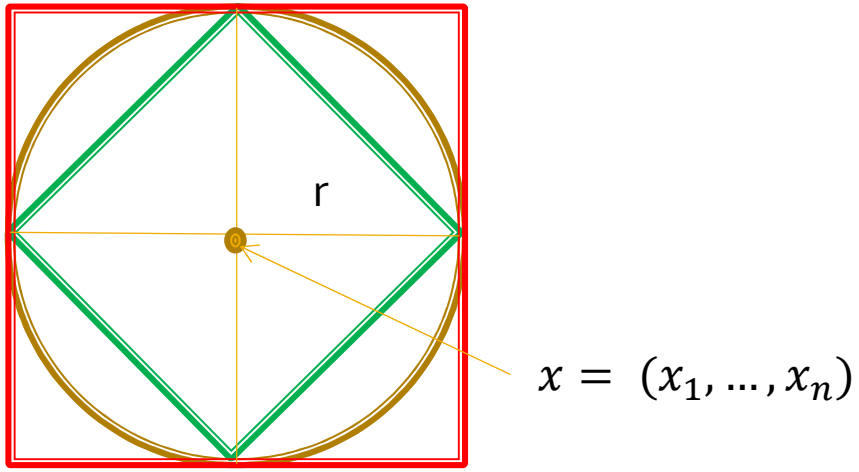
L_1-norm:

$$dist(x, y) = 4 + 3 = 7$$

L_∞-norm:

$$dist(x, y) = \max\{3,4\} = 4$$

Example



Green: All points y at distance $L_1(x,y) = r$ from point x

Blue: All points y at distance $L_2(x,y) = r$ from point x

Red: All points y at distance $L_\infty(x,y) = r$ from point x

L_p distances for sets

- We can apply all the L_p distances to the cases of sets of attributes, with or without counts, if we represent the sets as vectors
  - E.g., a transaction is a 0/1 vector
  - E.g., a document is a vector of counts.

Similarities into distances

- Jaccard distance:

$$JDist(X, Y) = 1 - JSim(X, Y)$$

- Jaccard Distance is a metric

- Cosine distance:

$$Dist(X, Y) = 1 - \cos(X, Y)$$

- Cosine distance is a metric

Why Jaccard Distance Is a Distance Metric

- JDist(x,x) = 0
  - since JSim(x,x) = 1
- JDist(x,y) = JDist(y,x)
  - by symmetry of intersection
- JDist(x,y) ≥ 0
  - since intersection of X,Y cannot be bigger than the union.
- Triangle inequality:
  - Follows from the fact that JSim(X,Y) is the probability of randomly selected element from the union of X and Y to belong to the intersection

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.
    - The L_1 norm for the binary vectors
- Hamming distance between two vectors of categorical attributes is the number of positions in which they differ.
  - Example: x = (married, low income, cheat),
    y = (single, low income, not cheat)
  - d(x,y) = 2

Why Hamming Distance Is a Distance Metric

- 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*.

- For binary vectors if follows from the fact that L_1 norm is a metric

Distance between strings

- How do we define similarity between strings?

weird	wierd
intelligent	unintelligent
Athena	Athina

- Important for recognizing and correcting typing errors and analyzing DNA sequences.

Edit Distance for strings

- The edit distance of two strings is the number of inserts and deletes of characters needed to turn one into the other.
- Example: x = abcde ; y = bcduve.
  - Turn *x* into *y* by deleting a, then inserting u and v after d.
  - Edit distance = 3.
- Minimum number of operations can be computed using dynamic programming
- Common distance measure for comparing DNA sequences

Why Edit Distance Is a Distance Metric

- 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) ≥ 0: no notion of negative edits.
- Triangle inequality: changing *x* to *z* and then to *y* is one way to change *x* to *y*. The minimum is no more than that

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.

- Same for any set of operations on strings.
  - Example: substring reversal or block transposition OK for DNA sequences
  - Example: character transposition is used for spelling

Distances between distributions

- We can view a document as a distribution over the words

document	Apple	Microsoft	Obama	Election
D1	0.35	0.5	0.1	0.05
D2	0.4	0.4	0.1	0.1
D2	0.05	0.05	0.6	0.3

- KL-divergence (Kullback-Leibler) for distributions P,Q

$$D_{KL}(P\|Q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

- KL-divergence is asymmetric. We can make it symmetric by taking the average of both sides
- JS-divergence (Jensen-Shannon)

$$JS(P,Q) = \frac{1}{2} D_{KL}(P\|Q) + \frac{1}{2} D_{KL}(Q\|P)$$

Min-Hashing
LOCALITY SENSITIVE HASHING

Thanks to:
Rajaraman and Ullman, "Mining Massive Datasets"
Evimaria Terzi, slides for Data Mining Course.

Why is similarity important?

- We saw many definitions of similarity and distance
- How do we make use of similarity in practice?
- What issues do we have to deal with?

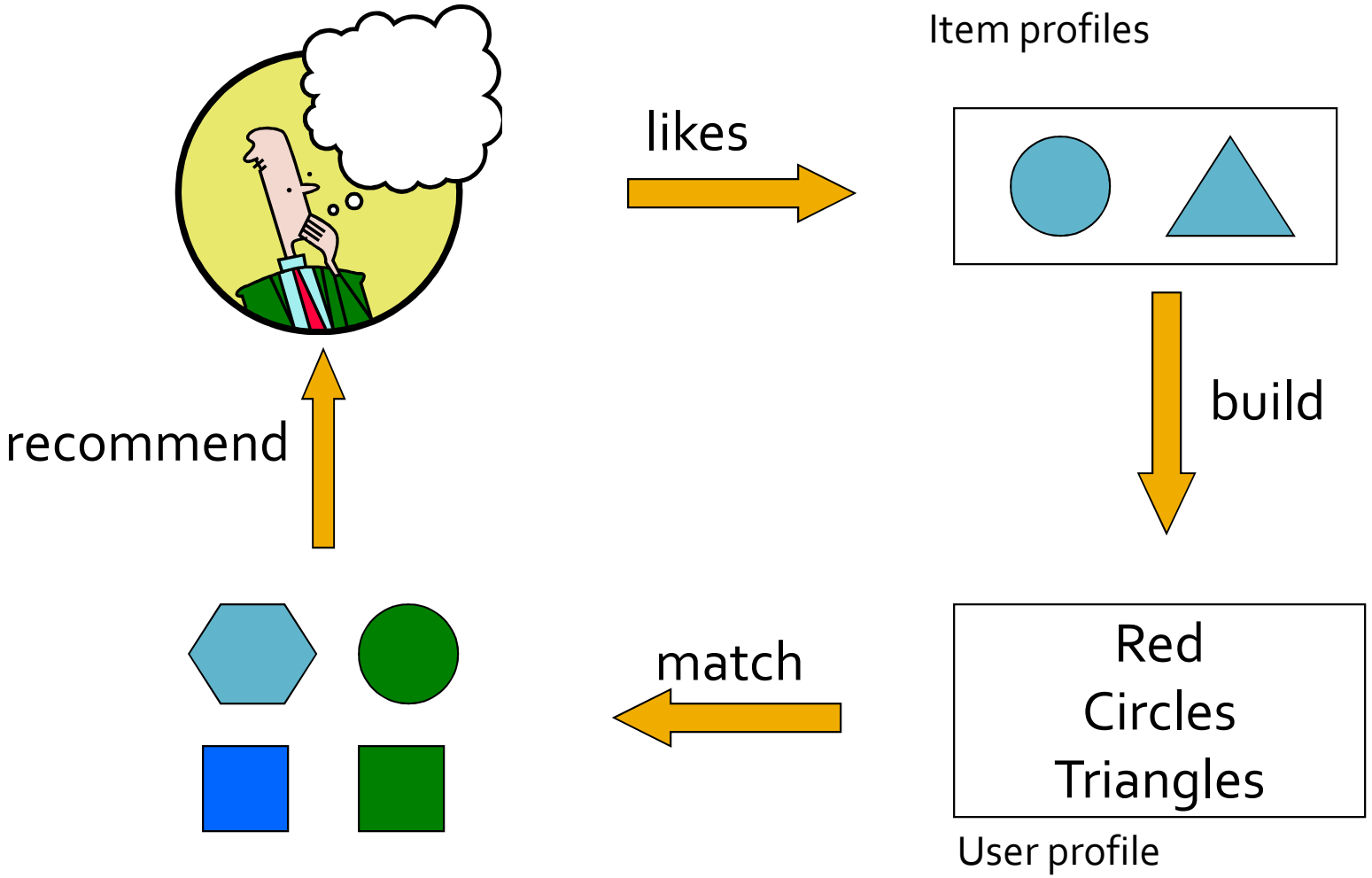
An important problem

- Recommendation systems
  - When a user buys an item (initially books) we want to recommend other items that the user may like
  - When a user rates a movie, we want to recommend movies that the user may like
  - When a user likes a song, we want to recommend other songs that they may like
- A big success of data mining
- Exploits the long tail

Recommendation Systems

- Content-based:
  - Represent the items into a feature space and recommend items to customer C similar to previous items rated highly by C
    - Movie recommendations: recommend movies with same actor(s), director, genre, …
    - Websites, blogs, news: recommend other sites with "similar" content

Plan of action



Limitations of content-based approach

- Finding the appropriate features
  - e.g., images, movies, music
- Overspecialization
  - Never recommends items outside user's content profile
  - People might have multiple interests
- Recommendations for new users
  - How to build a profile?

Recommendation Systems (II)

- Collaborative Filtering (user –user)
  - Consider user c
  - Find set D of other users whose ratings are "similar" to c's ratings
  - Estimate user's ratings based on ratings of users in D

Recommendation Systems (III)

- Collaborative filtering (item-item)
  - For item s, find other similar items
  - Estimate rating for item based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model
- In practice, it has been observed that item-item often works better than user-user

Pros and cons of collaborative filtering

- Works for any kind of item
  - No feature selection needed
- New user problem
- New item problem
- Sparsity of rating matrix
  - Cluster-based smoothing?

Another important problem

- Find duplicate and near-duplicate documents from a web crawl.
- Why is it important:
  - Identify mirrored web pages, and avoid indexing them, or serving them multiple times
  - Find replicated news stories and cluster them under a single story.
  - Identify plagiarism
- What if we wanted exact duplicates?

Finding similar items

- Both the problems we described have a common component
  - We need a quick way to find highly similar items to a query item
  - OR, we need a method for finding all pairs of items that are highly similar.
- Also known as the Nearest Neighbor problem, or the All Nearest Neighbors problem

- We will examine it for the case of near-duplicate web documents.

Main issues

- What is the right representation of the document when we check for similarity?
  - E.g., representing a document as a set of characters will not do (why?)
- When we have billions of documents, keeping the full text in memory is not an option.
  - We need to find a shorter representation
- How do we do pairwise comparisons of billions of documents?
  - If exact match was the issue it would be ok, can we replicate this idea?

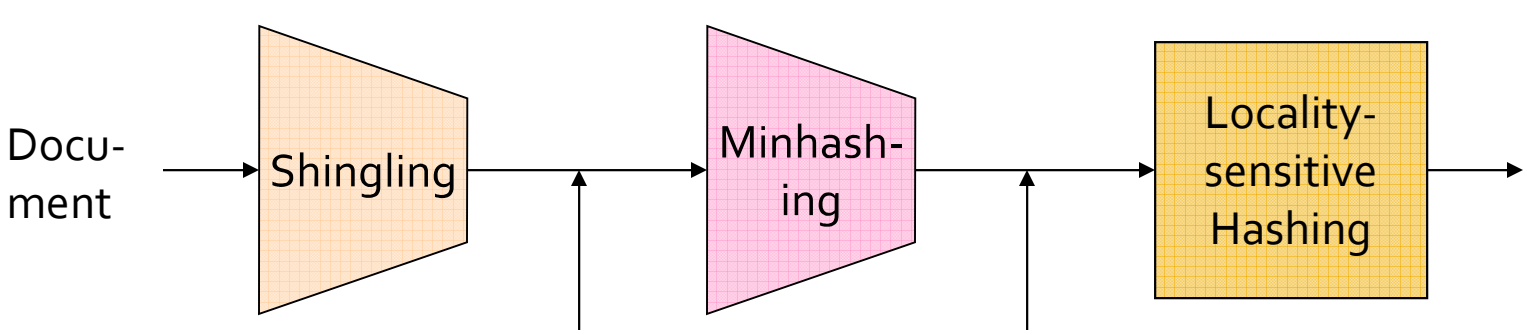
Three Essential Techniques for Similar Documents

1. Shingling : convert documents, emails, etc., to sets.

2. Minhashing : convert large sets to short signatures, while preserving similarity.

3. Locality-Sensitive Hashing (LSH): 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.

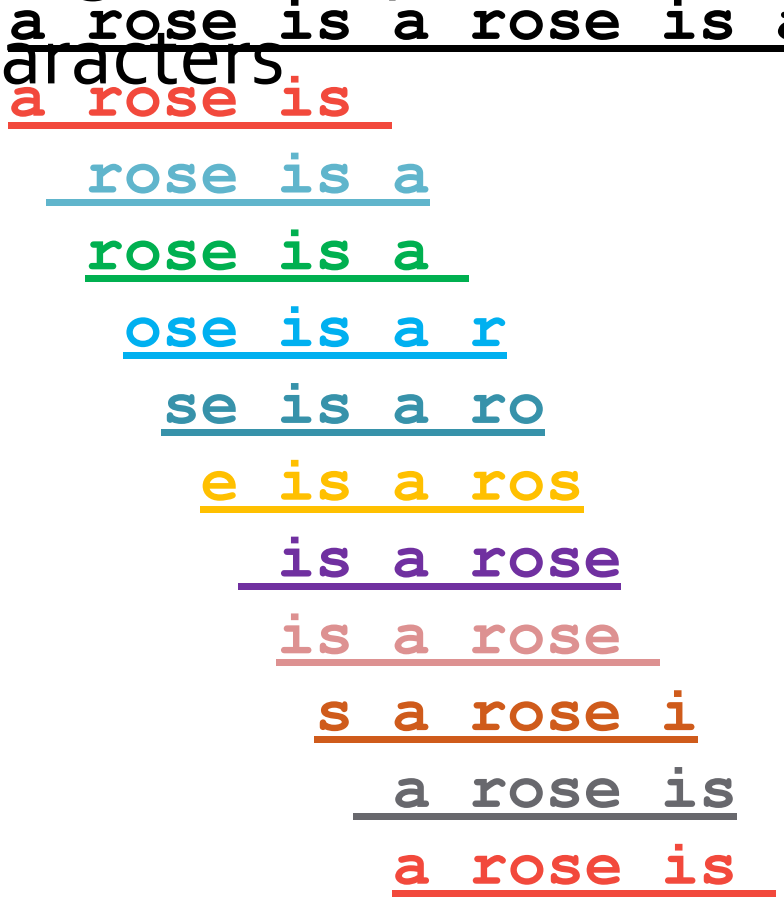
Shingles

- A k -shingle (or k -gram) for a document is a sequence of k characters that appears in the document.
- Example: document = abcab. k=2
  - Set of 2-shingles = {ab, bc, ca}.
  - Option: regard shingles as a bag, and count ab twice.

- Represent a document by its set of k-shingles.

Shingling

- Shingle: a sequence of k contiguous characters



Working Assumption

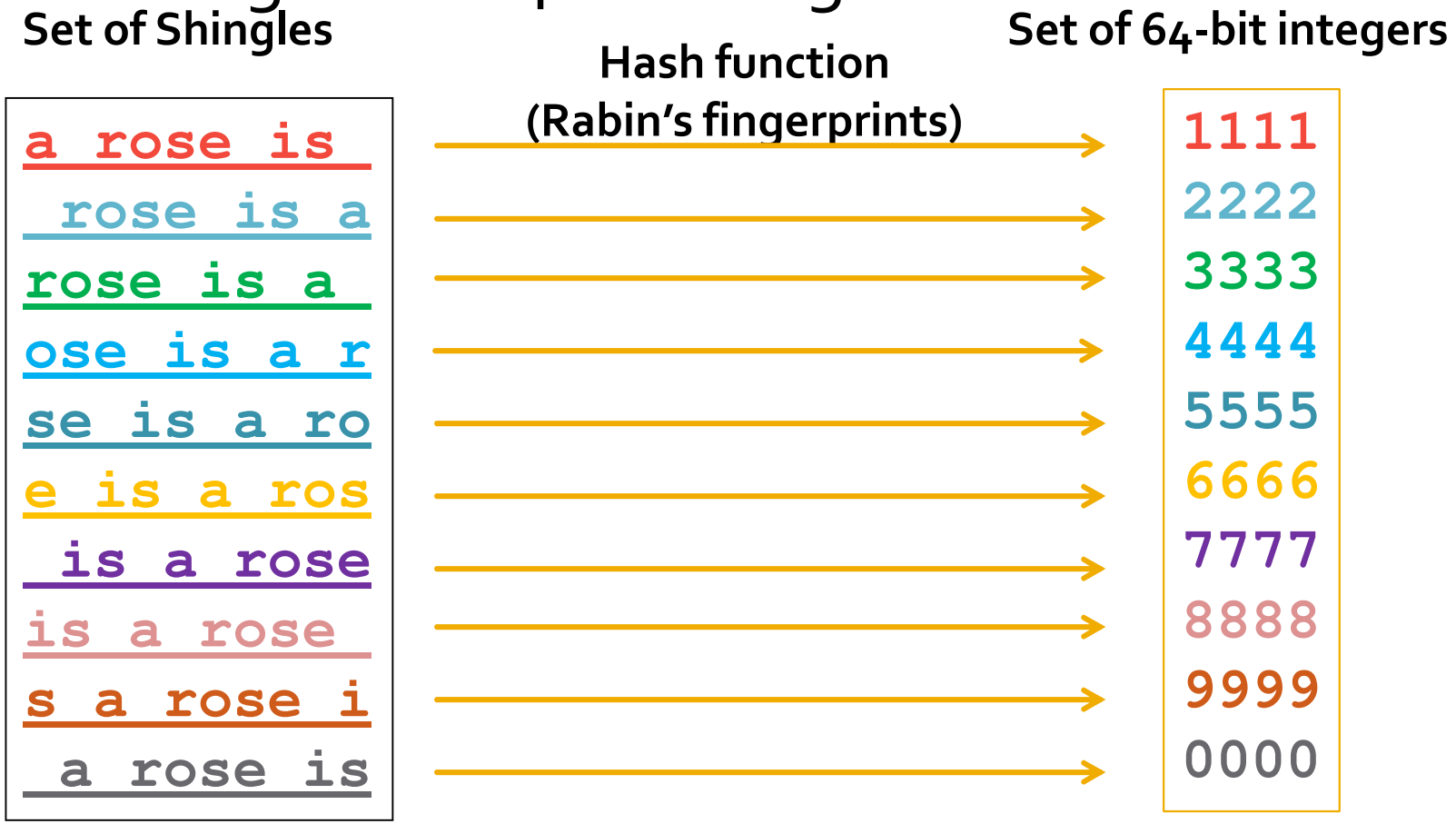
- Documents that have lots of shingles in common have similar text, even if the text appears in different order.
- Careful: you must pick k large enough, or most documents will have most shingles.
  - Extreme case $k = 1$: all documents are the same
  - $k = 5$ is OK for short documents; $k = 10$ is better for long documents.
- Alternative ways to define shingles:
  - Use words instead of characters
  - Anchor on stop words (to avoid templates)

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.
- From now on we will assume that shingles are integers
  - Collisions are possible, but very rare

Fingerprinting

- Hash shingles to 64-bit integers



Basic Data Model: Sets

- Document: A document is represented as a set shingles (more accurately, hashes of shingles)
- Document similarity: Jaccard similarity of the sets of shingles.
  - Common shingles over the union of shingles
  - *Sim* $(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$.
- Although we use the documents as our driving example the techniques we will describe apply to any kind of sets.
  - E.g., similar customers or items.

Signatures

- Problem: shingle sets are too large to be kept in memory.
- Key idea: "hash" each set S to a small signature Sig (S), such that:
  1. Sig (S) is small enough that we can fit a signature in main memory for each set.
  2. Sim (S_1, S_2) is (almost) the same as the "similarity" of Sig (S_1) and Sig (S_2). (signature preserves similarity).
- Warning: This method can produce false negatives, and false positives (if an additional check is not made).
  - False negatives: Similar items deemed as non-similar
  - False positives: Non-similar items deemed as similar

From Sets to Boolean Matrices

- Represent the data as a boolean matrix M
  - Rows = the universe of all possible set elements
    - In our case, shingle fingerprints take values in $[0 \ldots 2^{64}-1]$
  - Columns = the sets
    - In our case, documents, sets of shingle fingerprints
  - M(r,S) = 1 in row r and column S if and only if r is a member of S.
- Typical matrix is sparse.
  - We do not really materialize the matrix

Example

- Universe: U = {A,B,C,D,E,F,G}

- X = {A,B,F,G}
- Y = {A,E,F,G}

- $\text{Sim}(X,Y) = \frac{3}{5}$

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

Example

- Universe: U = {A,B,C,D,E,F,G}

- X = {A,B,F,G}
- Y = {A,E,F,G}

- $\text{Sim}(X,Y) = \frac{3}{5}$

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

At least one of the columns has value 1

Example

- Universe: U = {A,B,C,D,E,F,G}

- X = {A,B,F,G}
- Y = {A,E,F,G}

- $\text{Sim}(X,Y) = \frac{3}{5}$

	X	Y
A	1	1
B	1	0
C	0	0
D	0	0
E	0	1
F	1	1
G	1	1

Both columns have value 1

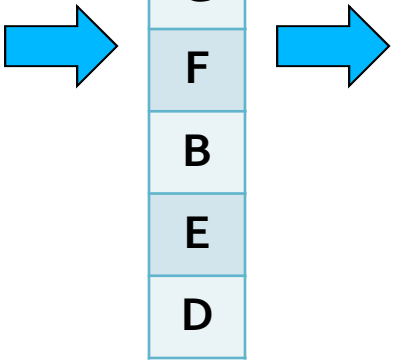
Minhashing

- Pick a random permutation of the rows (the universe U).
- Define "hash" function for set S
  - h(S) = the index of the first row (in the permuted order) in which column S has 1.
  - OR
  - h(S) = the index of the first element of S in the permuted order.
- Use k (e.g., k = 100) independent random permutations to create a signature.

Example of minhash signatures

- Input matrix

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

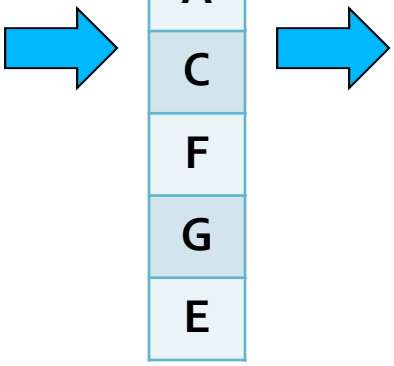


		S_1	S_2	S_3	S_4
1	A	1	0	1	0
2	C	0	1	0	1
3	G	1	0	1	0
4	F	1	0	1	0
5	B	1	0	0	1
6	E	0	1	0	1
7	D	0	1	0	1
		1	**2**	**1**	**2**

Example of minhash signatures

- Input matrix

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

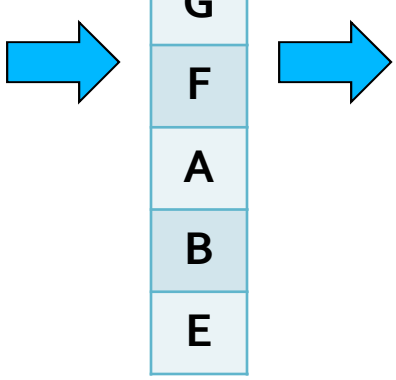


		S_1	S_2	S_3	S_4
1	D	0	1	0	1
2	B	1	0	0	1
3	A	1	0	1	0
4	C	0	1	0	1
5	F	1	0	1	0
6	G	1	0	1	0
7	E	0	1	0	1
		2	1	3	1

Example of minhash signatures

- Input matrix

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0



C
D
G
F
A
B
E

		S_1	S_2	S_3	S_4
1	C	0	1	0	1
2	D	0	1	0	1
3	G	1	0	1	0
4	F	1	0	1	0
5	A	1	0	1	0
6	B	1	0	0	1
7	E	0	1	0	1
		3	1	3	1

Example of minhash signatures

- Input matrix

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

≈

Signature matrix

	S_1	S_2	S_3	S_4
h_1	1	2	1	2
h_2	2	1	3	1
h_3	3	1	3	1

- Sig(S) = vector of hash values
  - e.g., $Sig(S_2) = [2,1,1]$
- Sig(S,i) = value of the i-th hash function for set S
  - E.g., $Sig(S_2,3) = 1$

Hash function Property

$$Pr(h(S_1) = h(S_2)) = Sim(S_1, S_2)$$

- where the probability is over all choices of permutations.
- Why?
  - The first row where one of the two sets has value 1 belongs to the union.
    - Recall that union contains rows with at least one 1.
  - We have equality if both sets have value 1, and this row belongs to the intersection

Similarity for Signatures

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

	S_1	S_2	S_3	S_4
A	1	0	1	0
B	1	0	0	1
C	0	1	0	1
D	0	1	0	1
E	0	1	0	1
F	1	0	1	0
G	1	0	1	0

≈

Signature matrix

S_1	S_2	S_3	S_4
1	2	1	2
2	1	3	1
3	1	3	1

	Actual	Sig
(S_1, S_2)	0	0
(S_1, S_3)	3/5	2/3
(S_1, S_4)	1/7	0
(S_2, S_3)	0	0
(S_2, S_4)	3/4	1
(S_3, S_4)	0	0

Zero similarity is preserved
High similarity is well approximated

- With multiple signatures we get a good approximation

Is it now feasible?

- Assume a billion rows
- Hard to pick a random permutation of 1…billion
- **Even representing a random permutation requires 1 billion entries!!!**
- How about accessing rows in permuted order?
- ☹

Being more practical

Approximating row permutations: pick k=100 hash functions $(h_1,\ldots,h_k)$

for each row **r**

for each hash function h_i

compute $h_i(r)$

for each column **S** that has **1** in row **r**

if $h_i(r)$ is a smaller value than **Sig(S,i) then**

Sig(S,i) = $h_i(r)$;

In practice this means selecting the function parameters

In practice only the rows (shingles) that appear in the data

$h_i(r)$ = index of shingle r in permutation

S contains shingle r

Find the shingle r with minimum index

Sig(S,i) will become the smallest value of $h_i(r)$ among all rows (shingles) for which column **S** has value **1** (shingle belongs in S); *i*.e., $h_i(r)$ gives the min index for the **i**-th permutation

Algorithm – All sets, k hash functions

Pick k=100 hash functions $(h_1,\ldots,h_k)$

In practice this means selecting the hash function parameters

for each row **r**

  for each hash function h_i

    compute $h_i(r)$

    for each column **S** that has **1** in row **r**

      if $h_i(r)$ is a smaller value than **Sig(S,i)** **then**

        Sig(S,i) = $h_i(r)$;

Compute $h_i(r)$ only once for all sets

Example

x	Row	S1	S2	h(x)	g(x)
0	A	1	0	1	3
1	B	0	1	2	0
2	C	1	1	3	2
3	D	1	0	4	4
4	E	0	1	0	1

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

$g(x) = 2x+3 \bmod 5$

h(Row)	Row	S1	S2
0	E	0	1
1	A	1	0
2	B	0	1
3	C	1	1
4	D	1	0

g(Row)	Row	S1	S2
0	B	0	1
1	E	0	1
2	C	1	0
3	A	1	1
4	D	1	0

	Sig1	Sig2
$h(0) = 1$	1	-
$g(0) = 3$	3	-
$h(1) = 2$	1	2
$g(1) = 0$	3	0
$h(2) = 3$	1	2
$g(2) = 2$	2	0
$h(3) = 4$	1	2
$g(3) = 4$	2	0
$h(4) = 0$	1	0
$g(4) = 1$	2	0

Finding similar pairs

- Problem: Find all pairs of documents with similarity at least t = 0.8
- 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 $5*10^{11}$ column-comparisons.
- At 1 microsecond/comparison: 6 days.

Locality-Sensitive Hashing

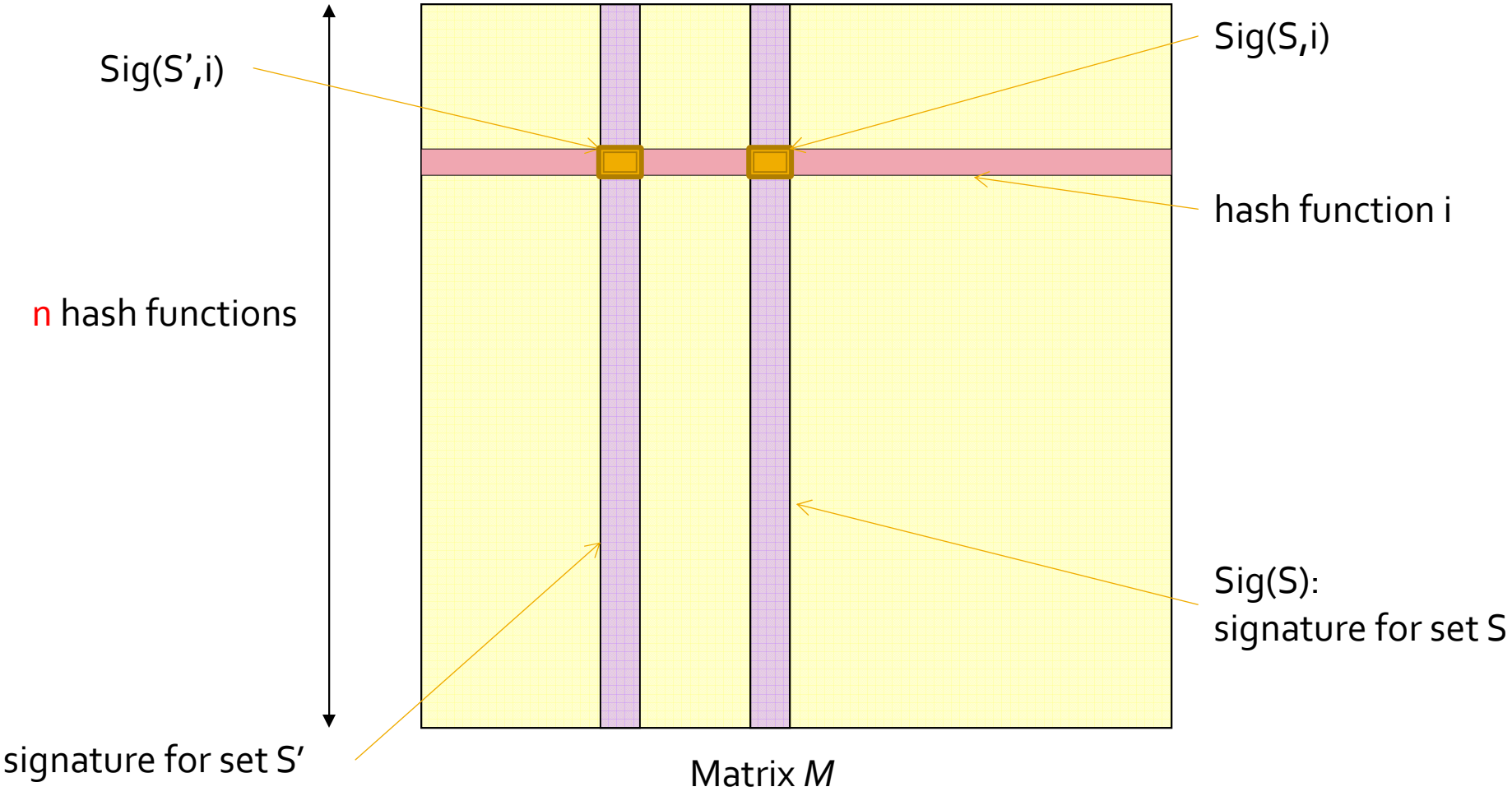
- What we want: a function f(X,Y) that tells whether or not X and Y is a candidate pair: a pair of elements whose similarity must be evaluated.
- A simple idea: X and Y are a candidate pair if they have the same min-hash signature.
  - Easy to test by hashing the signatures.
  - Similar sets are more likely to have the same signature.
  - Likely to produce many false negatives.
    - Requiring full match of signature is strict, some similar sets will be lost.

! Multiple levels of Hashing!

- Improvement: Compute multiple signatures; candidate pairs should have at least one common signature.
  - Reduce the probability for false negatives.

Signature matrix reminder

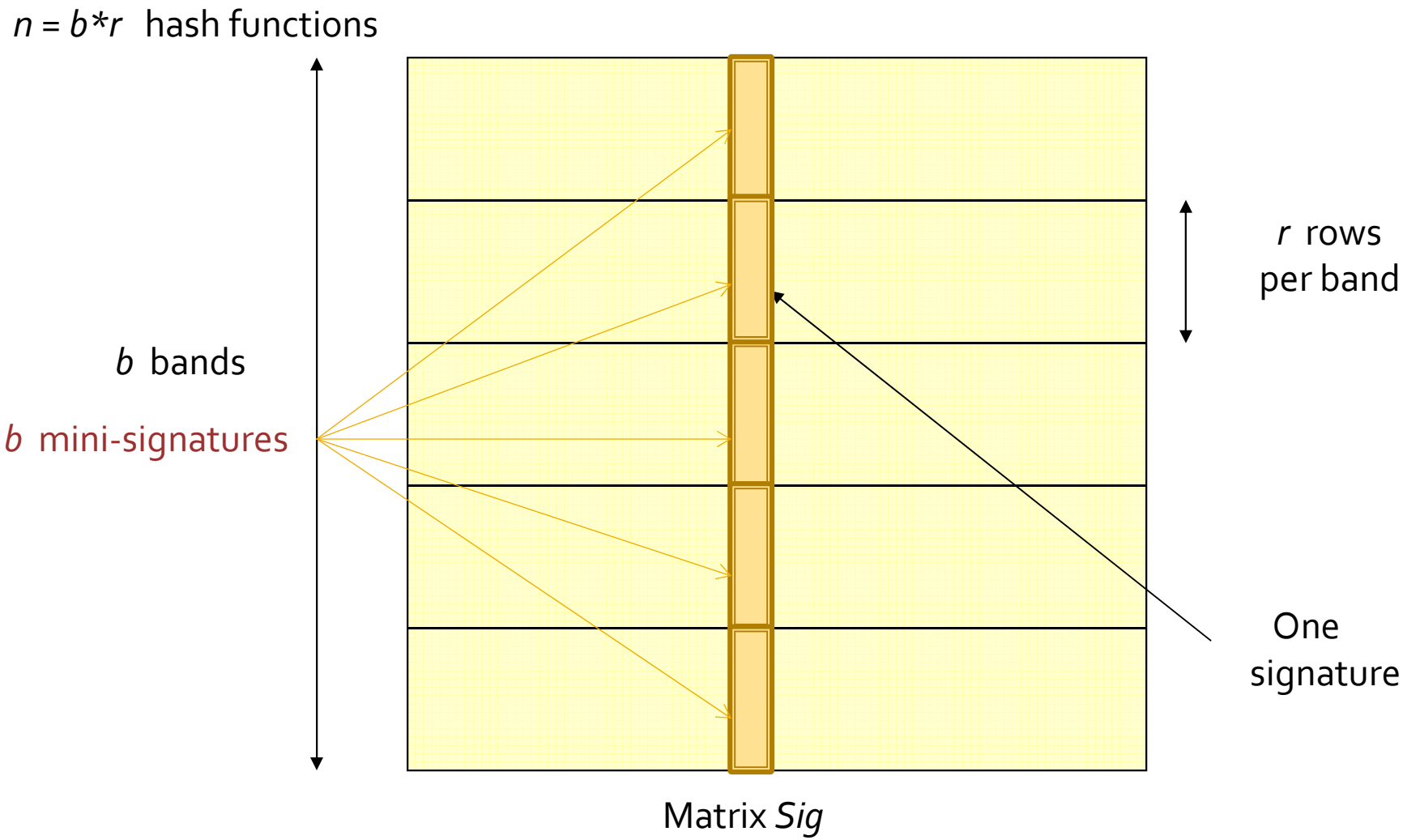
Prob(Sig(S,i) == Sig(S',i)) = sim(S,S')



Partition into Bands – (1)

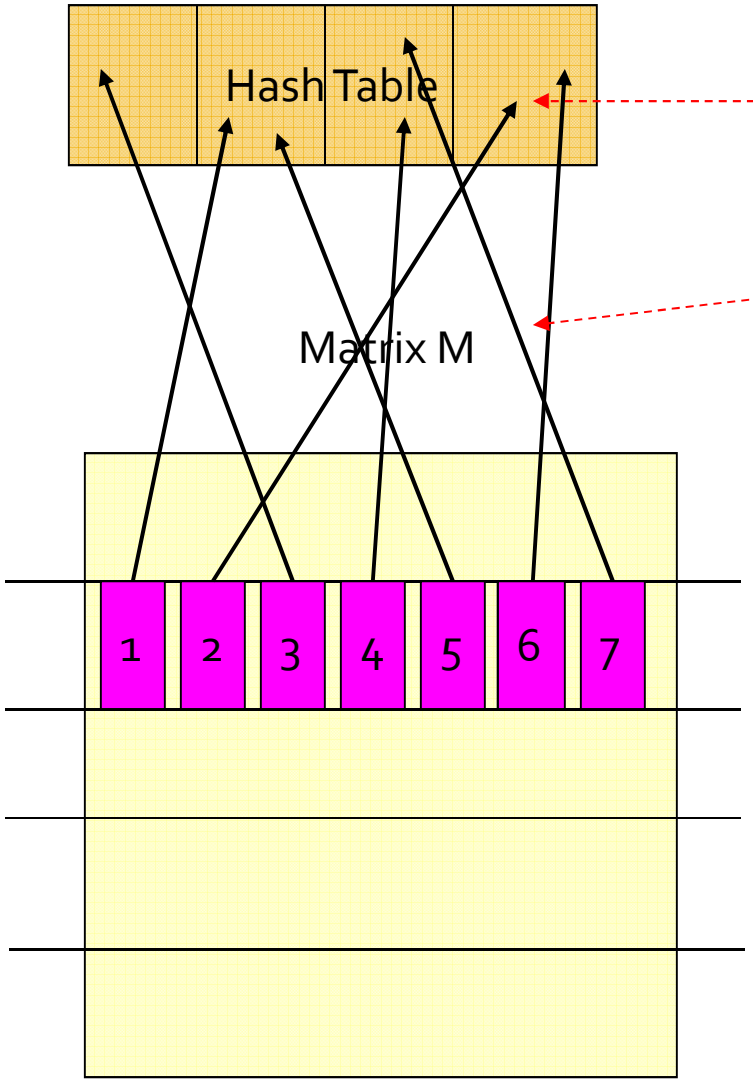
- Divide the signature matrix Sig into *b* bands of *r* rows.
  - Each band is a mini-signature with r hash functions.

Partitioning into bands



Partition into Bands – (2)

- Divide the signature matrix Sig into *b* bands of *r* rows.
  - Each band is a mini-signature with r hash functions.
- For each band, hash the mini-signature to a hash table with *k* buckets.
  - Make *k* as large as possible so that mini-signatures that hash to the same bucket are almost certainly identical.



Columns 2 and 6 are (almost certainly) identical.

Columns 6 and 7 are surely different.

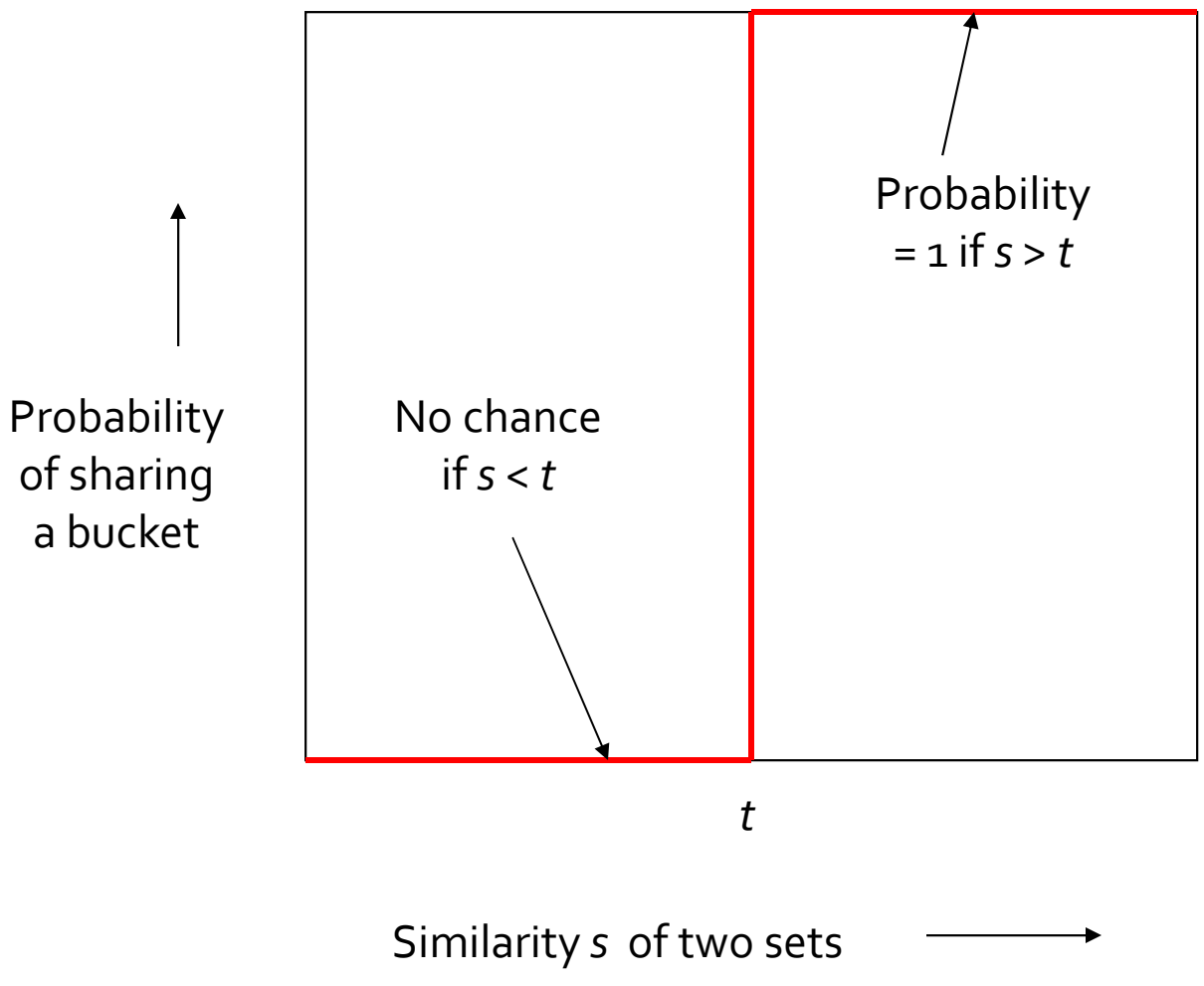
r rows

b bands

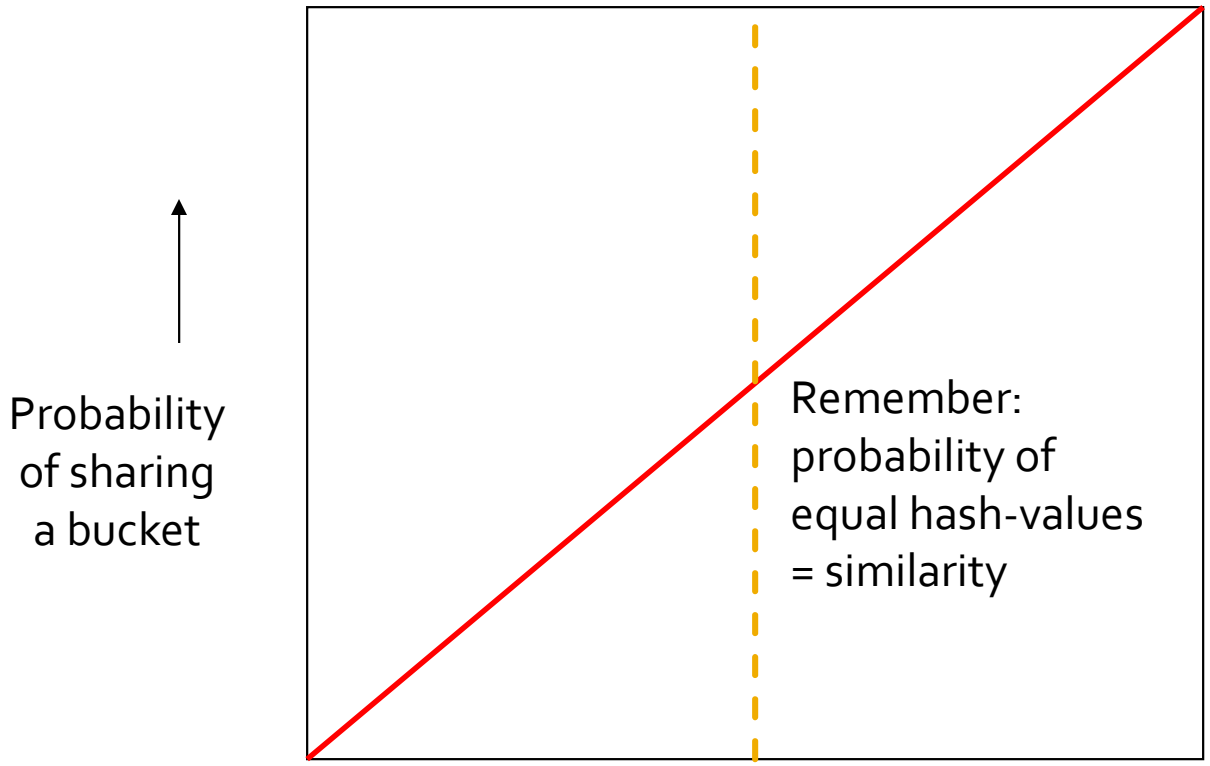
Partition into Bands – (3)

- Divide the signature matrix Sig into *b* bands of *r* rows.
  - Each band is a mini-signature with r hash functions.
- For each band, hash the mini-signature to a hash table with *k* buckets.
  - Make *k* as large as possible so that mini-signatures that hash to the same bucket are almost certainly identical.
- Candidate column pairs are those that hash to the same bucket for at least 1 band.
- Tune *b* and *r* to catch most similar pairs, but few non-similar pairs.

Analysis of LSH – What We Want

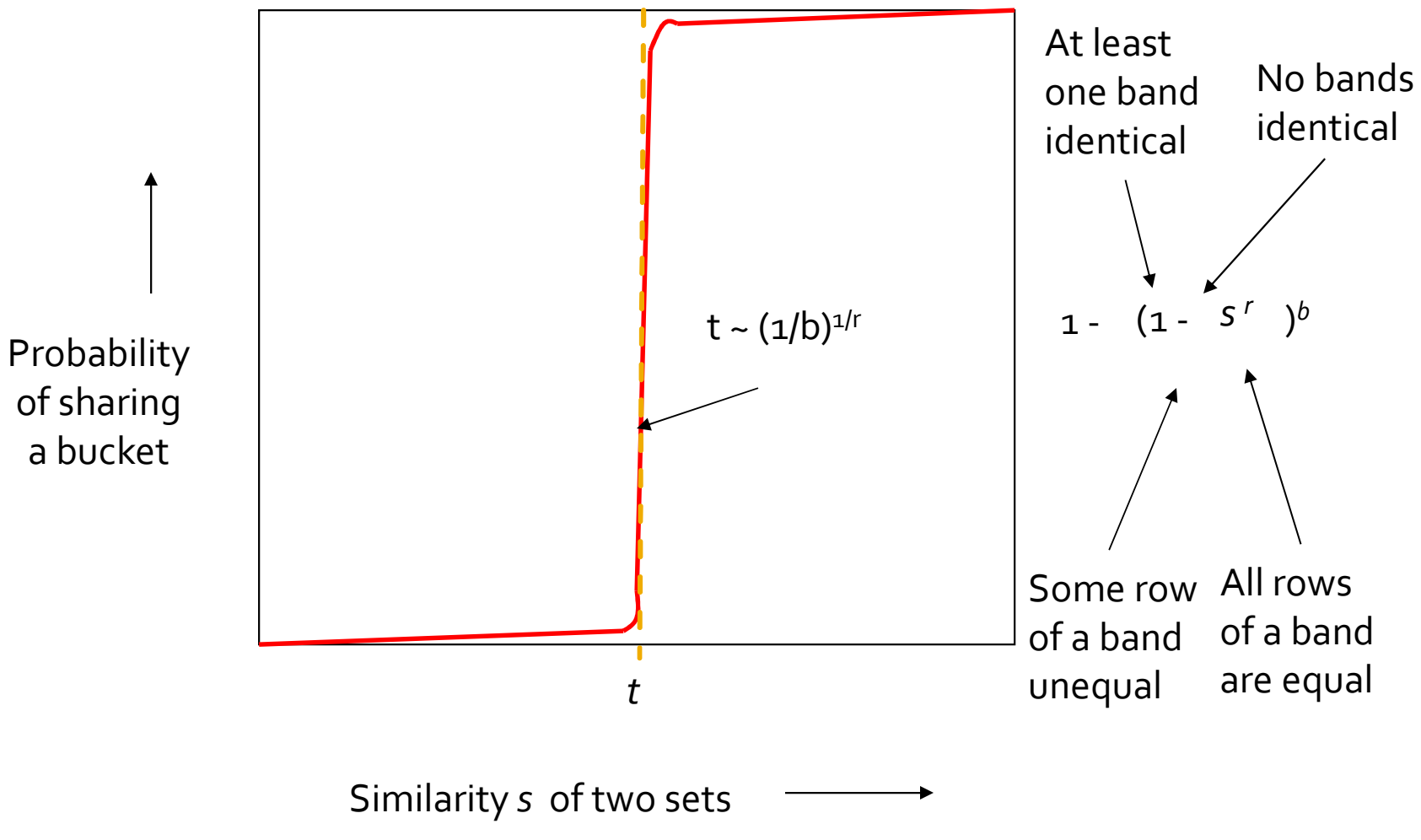


What One Band of One Row Gives You



Single hash signature

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

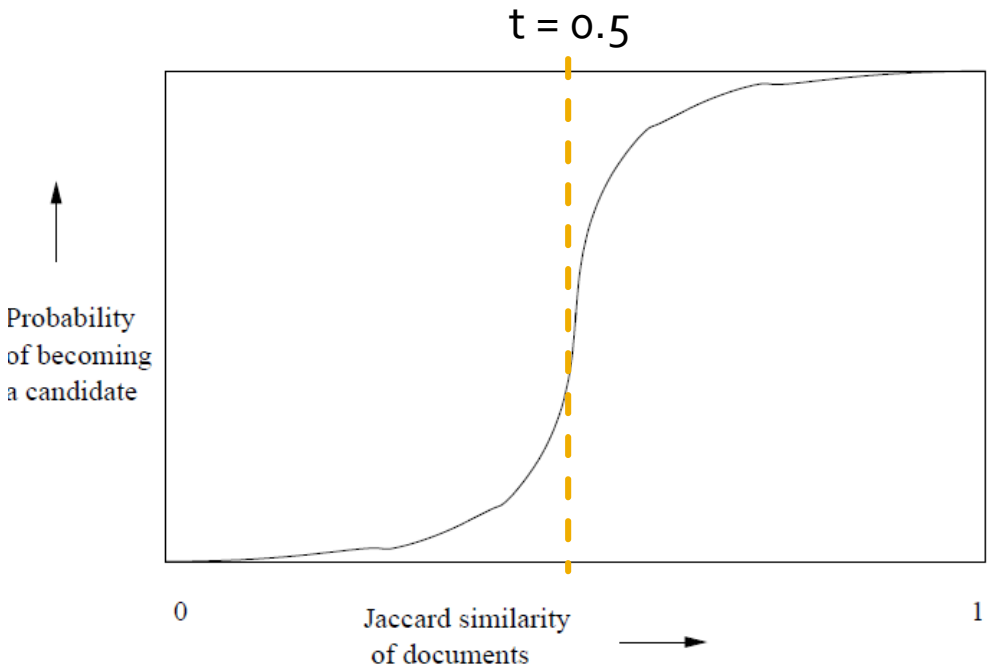


Figure 3.7: The S-curve

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* .

Locality-sensitive hashing (LSH)

- Big Picture: Construct hash functions **h: $R^d \rightarrow U$** such that for any pair of points **p,q**, for distance function **D** we have:
  - If **$D(p,q) \leq r$**, then **$Pr[h(p)=h(q)] \geq \alpha$** is high
  - If **$D(p,q) \geq cr$**, then **$Pr[h(p)=h(q)] \leq \beta$** is small
- Then, we can find close pairs by hashing

- LSH is a general framework: for a given distance function **D** we need to find the right **h**
  - **h** is **(r, cr, α, β)-**sensitive

- For cosine distance, there is a technique analogous to minhashing for generating a $(d_1,d_2,(1-d_1/180),(1-d_2/180))$- sensitive family for any d_1 and d_2.
- Called *random hyperplanes*.

Random Hyperplanes

- Pick a random vector *v*, which determines a hash function h_v with two buckets.
- $h_v(x)$ = +1 if v.x > 0; = -1 if v.x < 0.
- LS-family **H** = set of all functions derived from any vector.
- Prob[h(x)=h(y)] = 1 – (angle between *x* and *y* divided by 180).

Signatures for Cosine Distance

- Pick some number of vectors, and hash your data for each vector.
- The result is a signature (*sketch*) of +1's and –1's that can be used for LSH like the minhash signatures for Jaccard distance.

Simplification

- We need not pick from among all possible vectors v to form a component of a sketch.
- It suffices to consider only vectors v consisting of +1 and –1 components.

Signatures for Cosine Distance

- Pick some number of vectors, and hash your data for each vector.
- The result is a signature (*sketch*) of +1's and –1's that can be used for LSH like the minhash signatures for Jaccard distance.

Simplification

- We need not pick from among all possible vectors v to form a component of a sketch.
- It suffices to consider only vectors v consisting of +1 and –1 components.

Course Recap

- **Coupon Collector Problem**
- **Reservoir Sampling**
- **Hadoop** = MapReduce + HDFS
- **Information Retrieval** = TD-IDF
- **Association Analysis** = Frequent Itemsets + Frequent Rules (sorted by **Lift**)
- **Similarity Analysis** = Shingling + Min-Hash + Locality Sensitive Hashing
- Min-Hashing = compact representation of sets that maintains similarity.
- LSH = Find similar candidates using hashing.