Theory of Locality Sensitive Hashing

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu

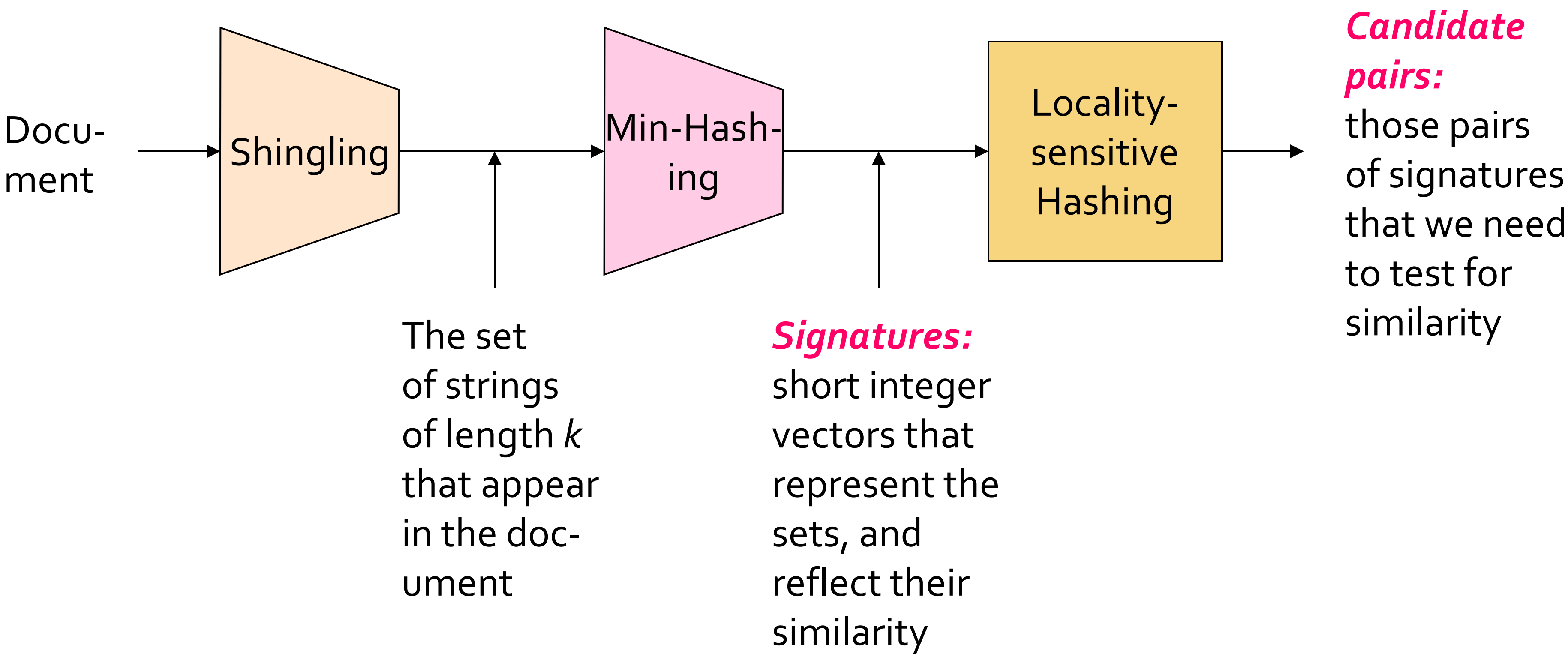
Recap: Finding similar documents

- **Task:** Given a large number (***N*** in the millions or billions) of documents, find "near duplicates"
- **Applications:**
  - Mirror websites, or approximate mirrors
    - Don't want to show both in a single set of search results
- **Problems:**
  - Many small pieces of one document can appear out of order in another
  - Too many documents to compare all pairs
  - Documents are so large or so many that they cannot fit in main memory

Recap: 3 Essential Steps

1. ***Shingling***: Convert docs to sets of items
   - Document is a set of k-**shingles**
2. ***Min-Hashing***: Convert large sets into short signatures, while preserving similarity
   - Want hash func. that $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
     - For the Jaccard similarity Min-Hash has this property!
3. ***Locality-sensitive hashing***: Focus on pairs of signatures likely to be from similar documents
   - Split signatures into bands and hash them
   - Documents with similar signatures get hashed into same buckets: **Candidate pairs**

Recap: The Big Picture



Recap: Shingles

- A *k*-shingle (or *k*-gram) is a sequence of *k* tokens that appears in the document
  - Example: **k=2**; $\mathbf{D_1}$ = abcab
    Set of 2-shingles: $\mathbf{C_1} = S(D_1) = \{ab, bc, ca\}$
- Represent a doc by a set of hash values of its *k*-shingles
- A natural document similarity measure is then the **Jaccard similarity:**

$$\boldsymbol{sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|}$$

  - Similarity of two documents is the Jaccard similarity of their shingles

Recap: Minhashing

- **Prob. $h_\pi(C_1) = h_\pi(C_2)$ is the same as $sim(D_1, D_2)$:**
  $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(D_1, D_2)$

Permutation π

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

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

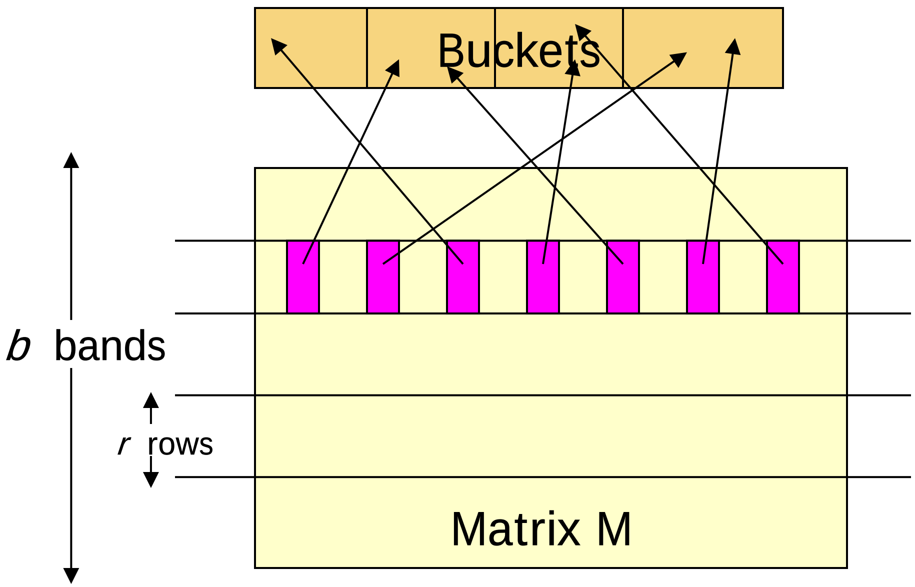
2	1	2	1
2	1	4	1
1	2	1	2

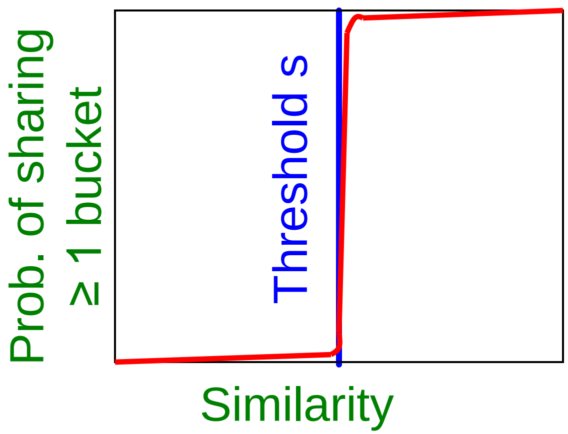
Similarities of columns and signatures (approx.) match!

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

Recap: LSH

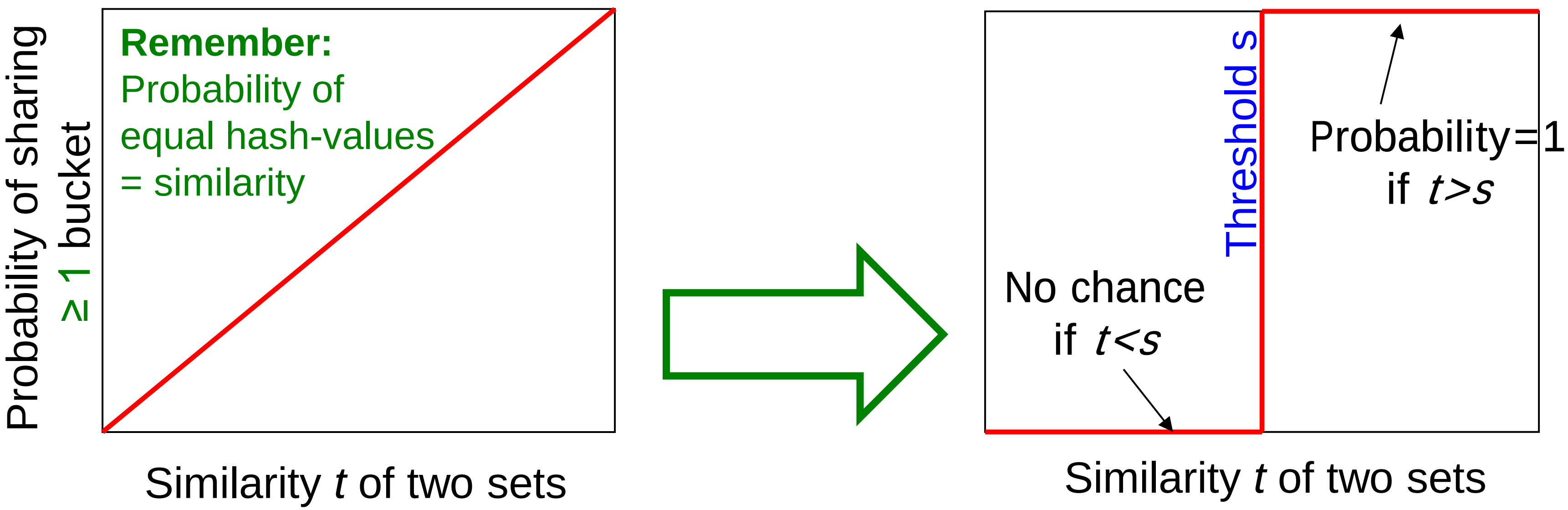
- **Hash columns of the signature matrix *M*:** **Similar columns likely hash to same bucket**
  - Divide matrix *M* into ***b*** bands of ***r*** rows (M=b·r)
  - ***Candidate*** column pairs are those that hash to the same bucket for ≥ **1** band





Recap: The S-Curve

- **The S-curve is where the "magic" happens**



This is what 1 hash-code gives you

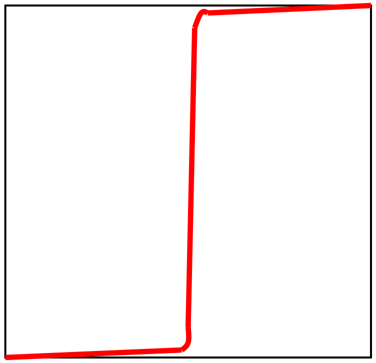
$\mathbf{Pr[}h_\pi\mathbf{(C_1) =} h_\pi\mathbf{(C_2)] =} sim\mathbf{(D_1, D_2)}$

This is what we want!

How to get a step-function?

By choosing *r* and *b*!

How Do We Make the S-curve?



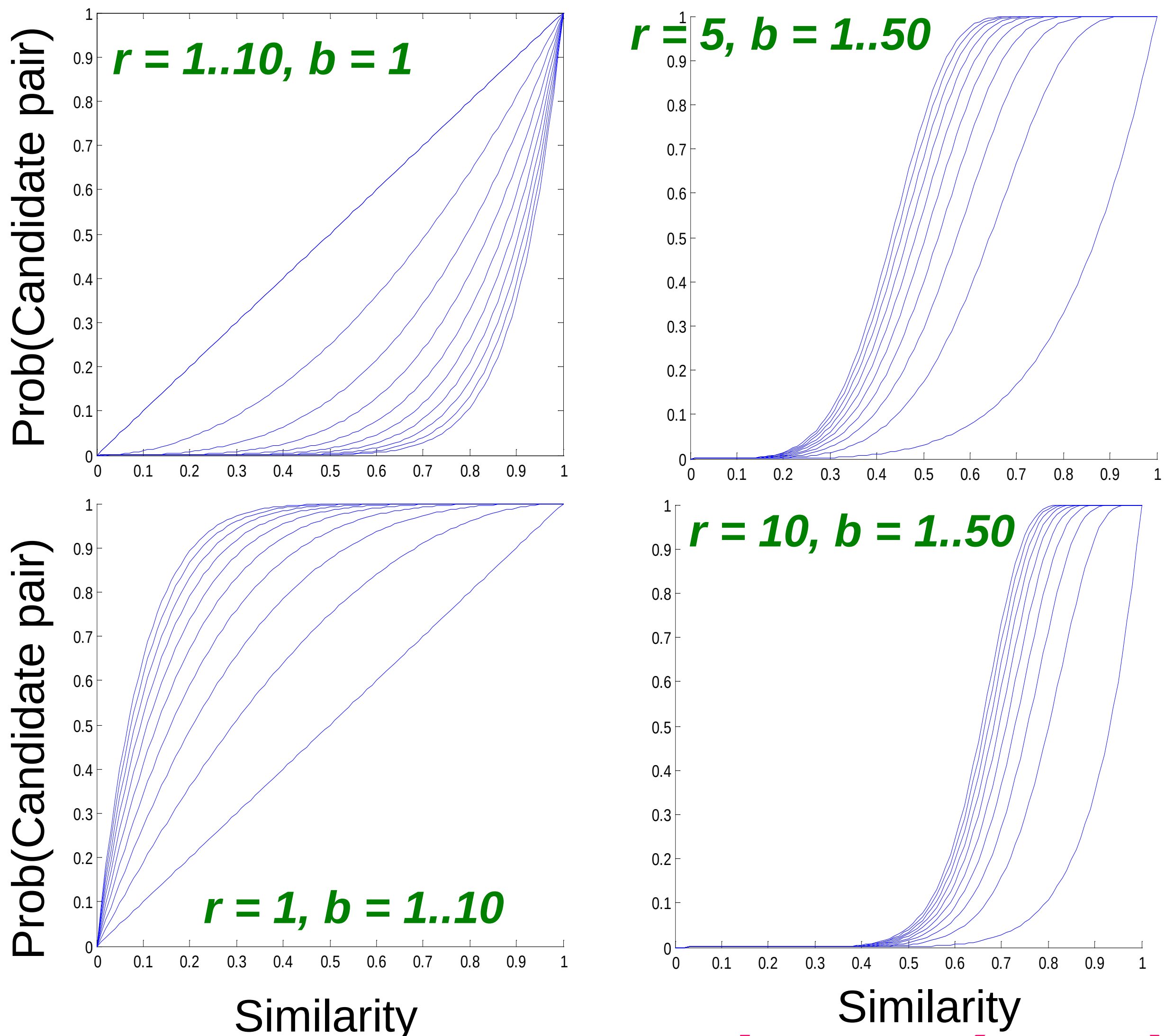
- **Remember:** ***b*** bands, ***r*** rows/band
- Let $sim(C_1, C_2) = t$
- Pick some band (***r*** rows)
  - Prob. that elements in a single row of columns C_1 and C_2 are equal $= t$
  - Prob. that all rows in a band are equal $= t^r$
  - Prob. that some row in a band is not equal $= 1 - t^r$
- Prob. that all bands are not equal $= (1 - t^r)^b$
- Prob. that at least 1 band is equal $= 1 - (1 - t^r)^b$

$$P(C_1, C_2 \text{ is a candidate pair}) = 1 - (1 - t^r)^b$$

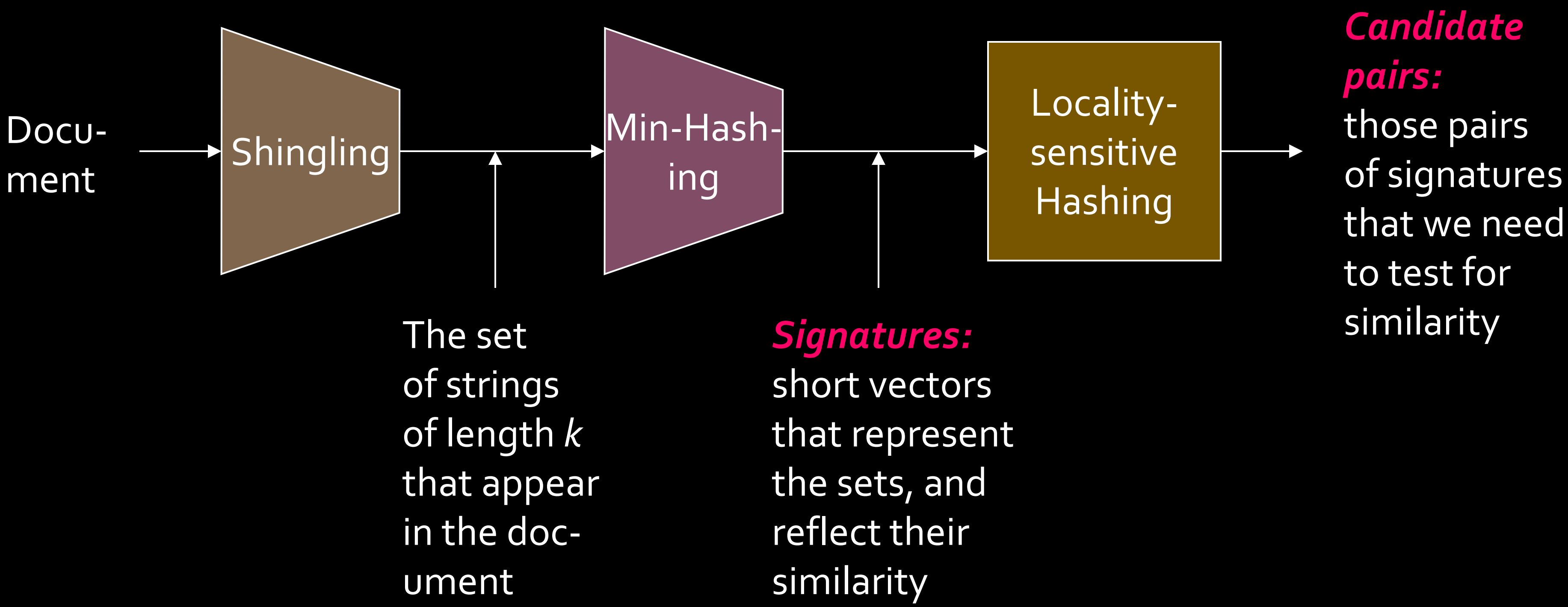
S-curves as a func. of *b* and *r*

Given a fixed threshold ***s***.

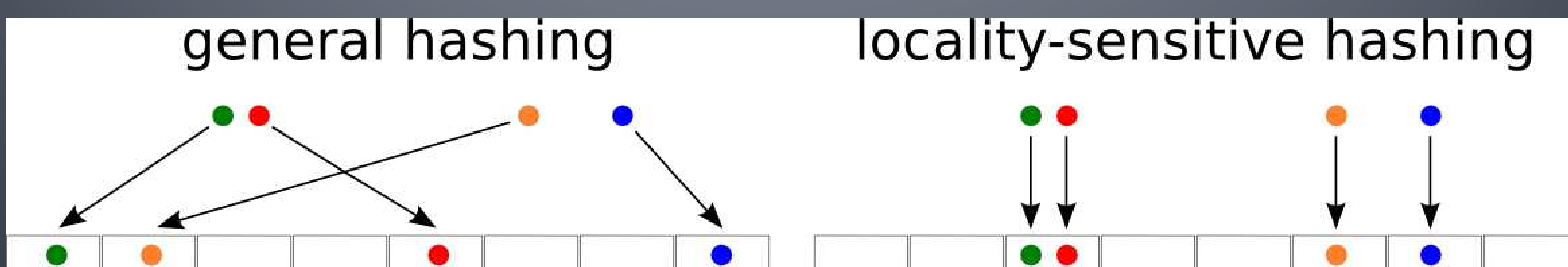
We want choose ***r*** and ***b*** such that the ***P(Candidate pair)*** has a "step" right around ***s***.



$$prob = 1 - (1 - t^r)^b$$



Theory of LSH



Theory of LSH

- **We have used LSH to find similar documents**
  - More generally, we found similar columns in large sparse matrices with high Jaccard similarity
    - For example, customer/item purchase histories
- **Can we use LSH for other distance measures?**
  - e.g., Euclidean distances, Cosine distance
  - **Let's generalize what we've learned!**

Families of Hash Functions

- **For Min-Hashing signatures, we got a Min-Hash function for each permutation of rows**
- A "hash function" is any function that takes ***two*** elements and says whether they are "equal"
  - **Shorthand:** ***h(x) = h(y)*** means "***h says x and y are equal***"
- A ***family*** of hash functions is any set of hash functions from which we can ***pick one at random efficiently***
  - **Example:** The set of Min-Hash functions generated from permutations of rows

Locality-Sensitive (LS) Families

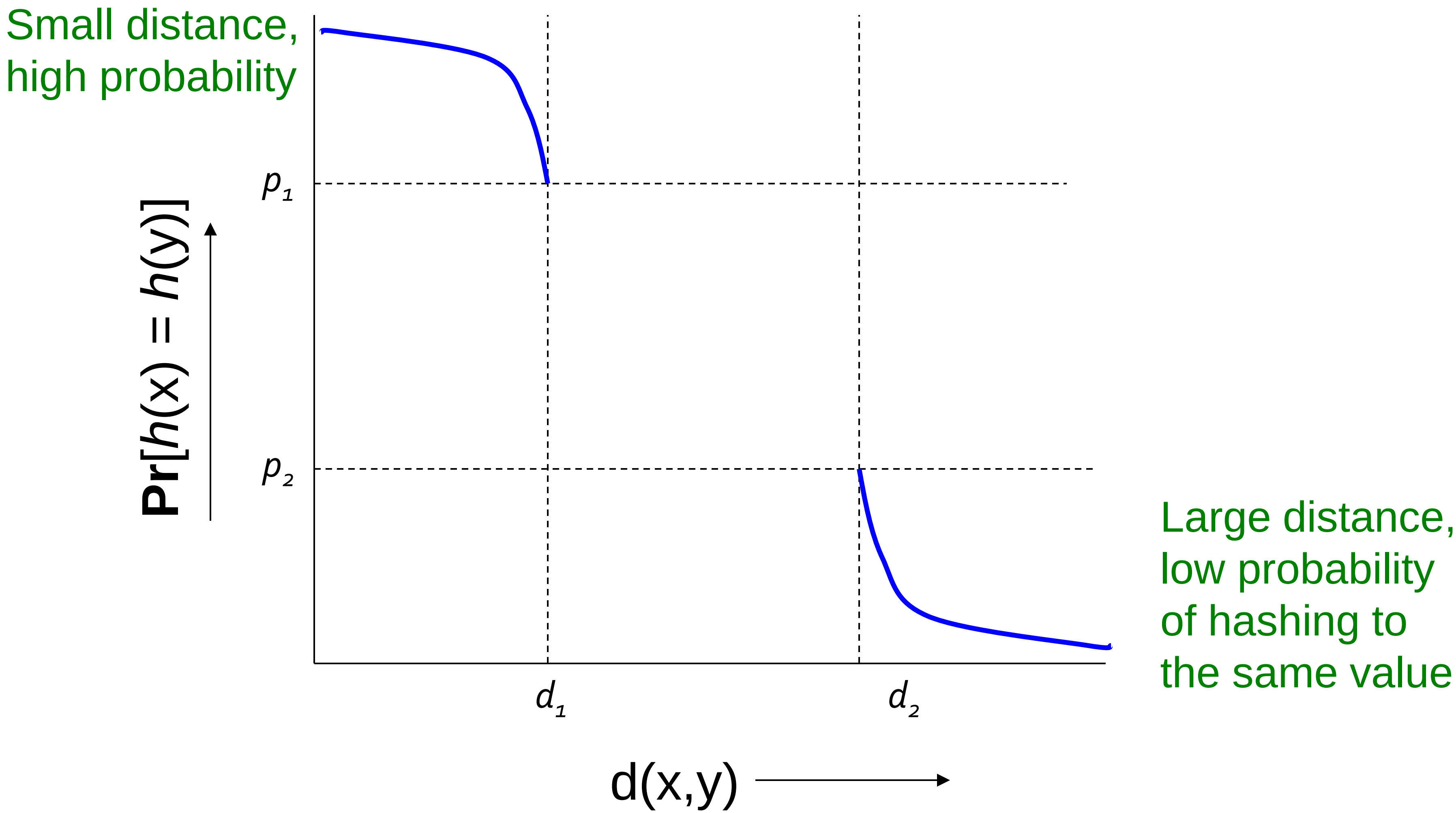
- **Suppose we have a space *S* of points with a distance measure *d(x,y)***

Critical assumption

- A family ***H*** of hash functions is said to be ***(d_1, d_2, p_1, p_2)-sensitive*** if for any ***x*** and ***y*** in ***S***:
  1. If $d(x, y) \leq d_1$, then the probability over all $h \in H$, that $h(x) = h(y)$ is at least p_1
  2. If $d(x, y) \geq d_2$, then the probability over all $h \in H$, that $h(x) = h(y)$ is at most p_2

With a LS Family we can do LSH!

A (d_1,d_2,p_1,p_2)-sensitive function



Example of LS Family: Min-Hash

- **Let:**
  - ***S*** = space of all sets,
  - ***d*** = Jaccard distance,
  - ***H*** is family of Min-Hash functions for all permutations of rows
- Then for any hash function $h \in H$:

$$Pr[h(x) = h(y)] = 1 - d(x, y)$$

  - Simply restates theorem about Min-Hashing in terms of distances rather than similarities

Example: LS Family – (2)

- **Claim: Min-hash *H*** is a ($\boxed{1/3}$, 2/3, $\boxed{2/3}$, 1/3)-sensitive family for ***S*** and ***d***.

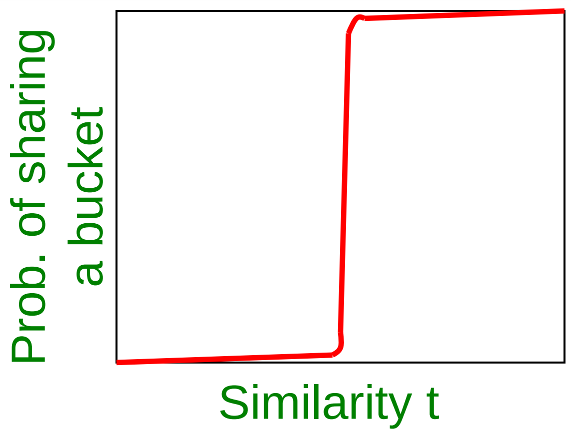
If distance $\leq$ 1/3
(so similarity ≥ 2/3)

Then probability
that Min-Hash values
agree is $\geq$ 2/3

- **For Jaccard similarity, Min-Hashing gives a *$(d_1,d_2,(1-d_1),(1-d_2))$*-sensitive family for any *$d_1<d_2$***
- **Theory leaves unknown what happens to pairs that are at distance between *d_1* and *d_2***
  - **Consequence:** No guarantees about fraction of false positives in that range

Amplifying a LS-Family

- **Can we reproduce the "S-curve" effect we saw before for any LS family?**



- The "**bands**" technique we learned for signature matrices carries over to this more general setting
  - **So we can do LSH with any** **(d_1, d_2, p_1, p_2)-*sensitive* family**

- **Two constructions:**
  - **AND** construction like "rows in a band"
  - **OR** construction like "many bands"

Amplifying Hash Functions: AND and OR

AND of Hash Functions

- Given family ***H***, construct family ***H'*** consisting of ***r*** functions from ***H***
- For $h = [h_1,\ldots,h_r]$ in **H'**, we say **h(x) = h(y)** if and only if $\mathbf{h_i(x) = h_i(y)}$ for **all** ***i*** $1 \le i \le r$
  - Note this corresponds to creating a band of size **r**
- **Theorem:** If ***H*** is (d_1, d_2, p_1, p_2)-sensitive, then **H'** is $(d_1, d_2, (p_1)^r, (p_2)^r)$-sensitive
- **Proof:** Use the fact that h_i's are **independent**

Subtlety Regarding Independence

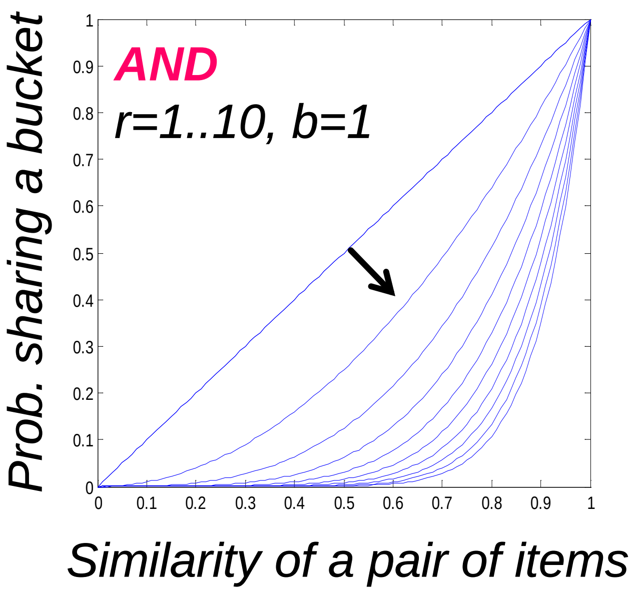
- **Independence of hash functions (HFs) really means that the prob. of two HFs saying "yes" is the product of each saying "yes"**
  - **But** two hash functions could be highly correlated
    - For example, in Min-Hash if their permutations agree in the first one million entries
  - **However**, the probabilities in definition of a LSH-family are over all possible members of ***H***, ***H'***

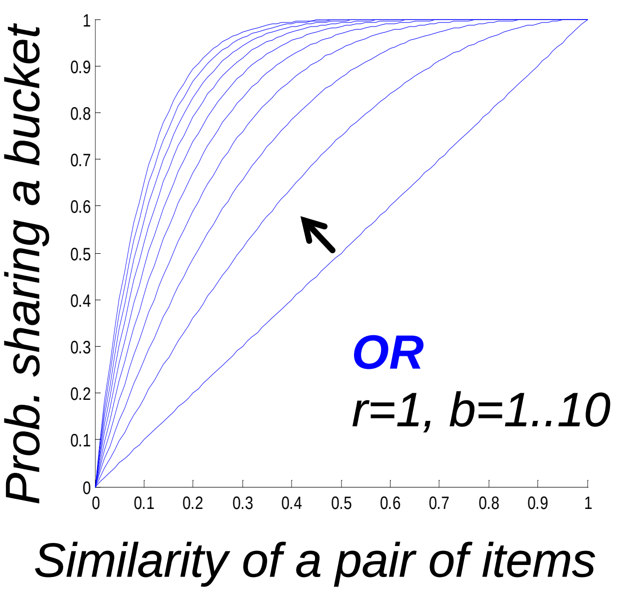
OR of Hash Functions

- Given family H, construct family H' consisting of b functions from H
- For $h = [h_1,\dots,h_b]$ in H', $\mathbf{h(x) = h(y)}$ if and only if $\mathbf{h_i(x) = h_i(y)}$ for **at least 1** i
- **Theorem:** If H is (d_1, d_2, p_1, p_2)-sensitive, then H' is $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$-sensitive
- **Proof:** Use the fact that h_i's are **independent**

Effect of AND and OR Constructions

- **AND** makes all probs. **shrink**, but by choosing ***r*** correctly, we can make the lower prob. approach 0 while the higher does not
- **OR** makes all probs. **grow**, but by choosing ***b*** correctly, we can make the upper prob. approach 1 while the lower does not





Composing Constructions

- ***r*-way AND** followed by ***b*-way OR** construction
  - **Exactly what we did with Min-Hashing**
    - If bands match in all ***r*** values hash to same bucket
    - Cols that are hashed into $\geq$ 1 common bucket → **Candidate**
- Take points ***x*** and ***y*** s.t. $Pr[h(x) = h(y)] = p$
  - ***H*** will make **(x,y)** a candidate pair with prob. **p**
- Construction makes **(x,y)** a candidate pair with probability $1-(1-p^r)^b$ **The S-Curve!**
  - **Example:** Take **H** and construct **H'** by the **AND** construction with $r = 4$. Then, from **H'**, construct **H''** by the **OR** construction with $b = 4$

Table for Function $1-(1-p^4)^4$

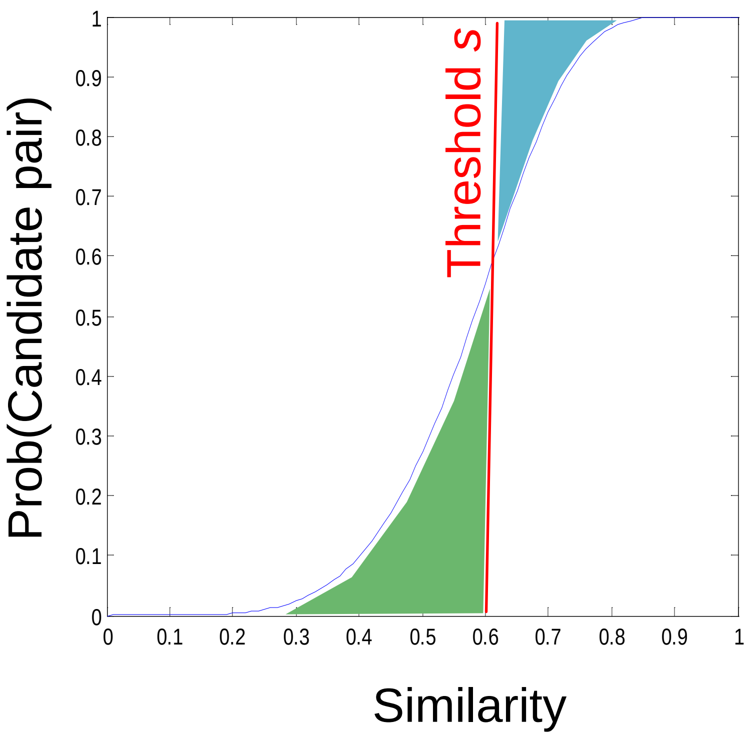
p	$1-(1-p^4)^4$
.2	.0064
.3	.0320
.4	.0985
.5	.2275
.6	.4260
.7	.6666
.8	.8785
.9	.9860

***r* = 4, *b* = 4** transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.8785,.0064)-sensitive family.

How to choose *r* and *b*

Picking *r* and *b*: The S-curve

- **Picking *r* and *b* to get desired performance**
  - **50 hash-functions (*r* = 5, *b* = 10)**

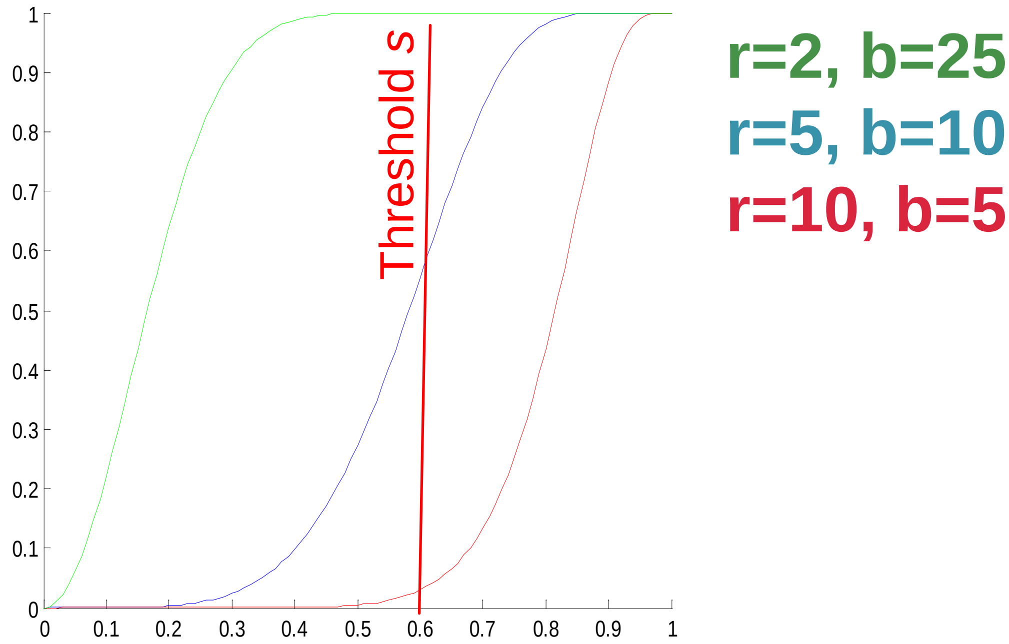


Blue area *X*: False Negative rate
These are pairs with ***sim* > *s*** but the ***X*** fraction won't share a band and then will **never become candidates.** This means we will never consider these pairs for (slow/exact) similarity calculation!

Green area Y: False Positive rate
These are pairs with ***sim* < *s*** but we will consider them as candidates. This is not too bad, we will consider them for (slow/exact) similarity computation and discard them.

Picking *r* and *b*: The S-curve

- **Picking *r* and *b* to get desired performance**
  - **50 hash-functions (*r* * *b* = *50*)**



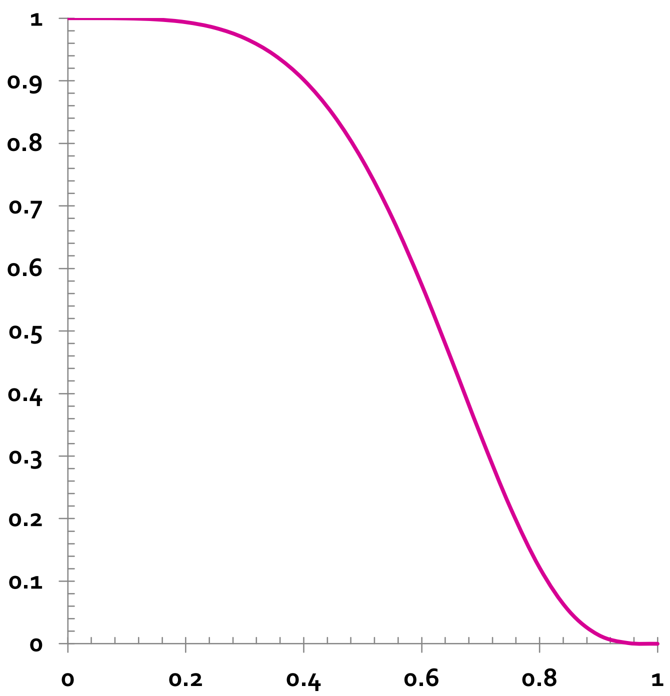
OR-AND Composition

- Apply a ***b***-way **OR** construction followed by an ***r***-way **AND** construction
- Transforms probability ***p*** into $(1-(1-p)^b)^r$
  - **The same S-curve, mirrored horizontally and vertically**

- **Example:** Take **H** and construct **H'** by the **OR** construction with ***b*** = 4. Then, from **H'**, construct **H''** by the **AND** construction with ***r*** = 4

Table for Function $(1-(1-p)^4)^4$

p	$(1-(1-p)^4)^4$
.1	.0140
.2	.1215
.3	.3334
.4	.5740
.5	.7725
.6	.9015
.7	.9680
.8	.9936



The example transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.9936,.1215)-sensitive family

Cascading Constructions

- **Example:** Apply the **(4,4) OR-AND** construction followed by the **(4,4) AND-OR** construction

- **Transforms a (.2, .8, .8, .2)-sensitive family into a (.2, .8, .9999996, .0008715)-sensitive family**
  - **Note this family uses 256 (=4*4*4*4) of the original hash functions**

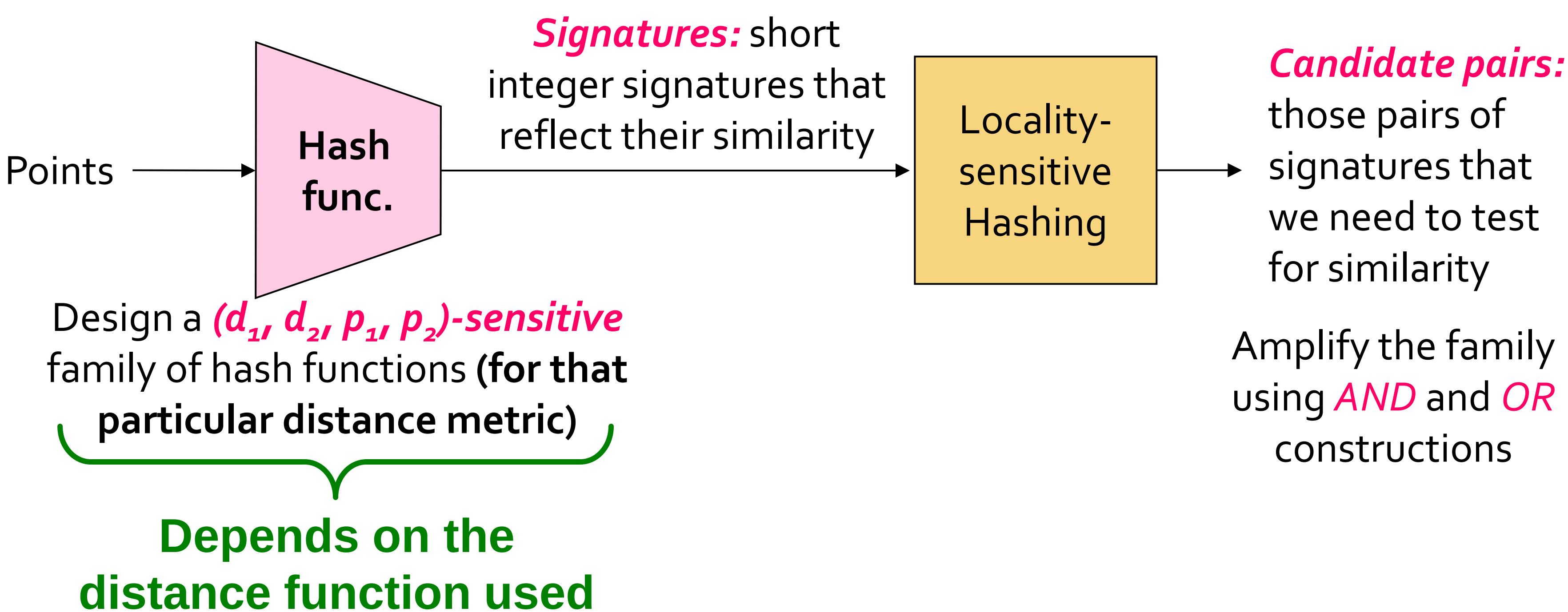
Summary

- Pick any two distances $d_1 < d_2$
- Start with a $(d_1, d_2, (1- d_1), (1- d_2))$-sensitive family
- Apply constructions to **amplify** (d_1, d_2, p_1, p_2)-sensitive family, where p_1 is almost 1 and p_2 is almost 0
- **The closer to 0 and 1 we get, the more hash functions must be used!**

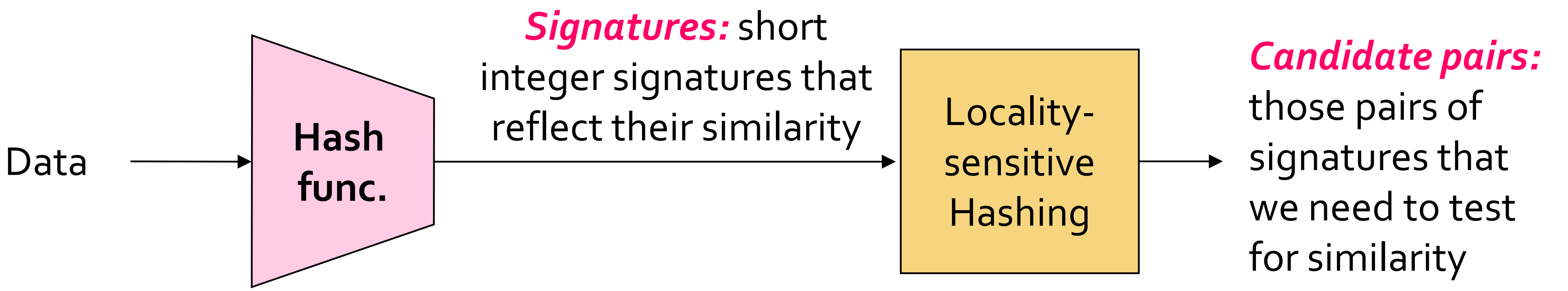
LHS for other distance metrics

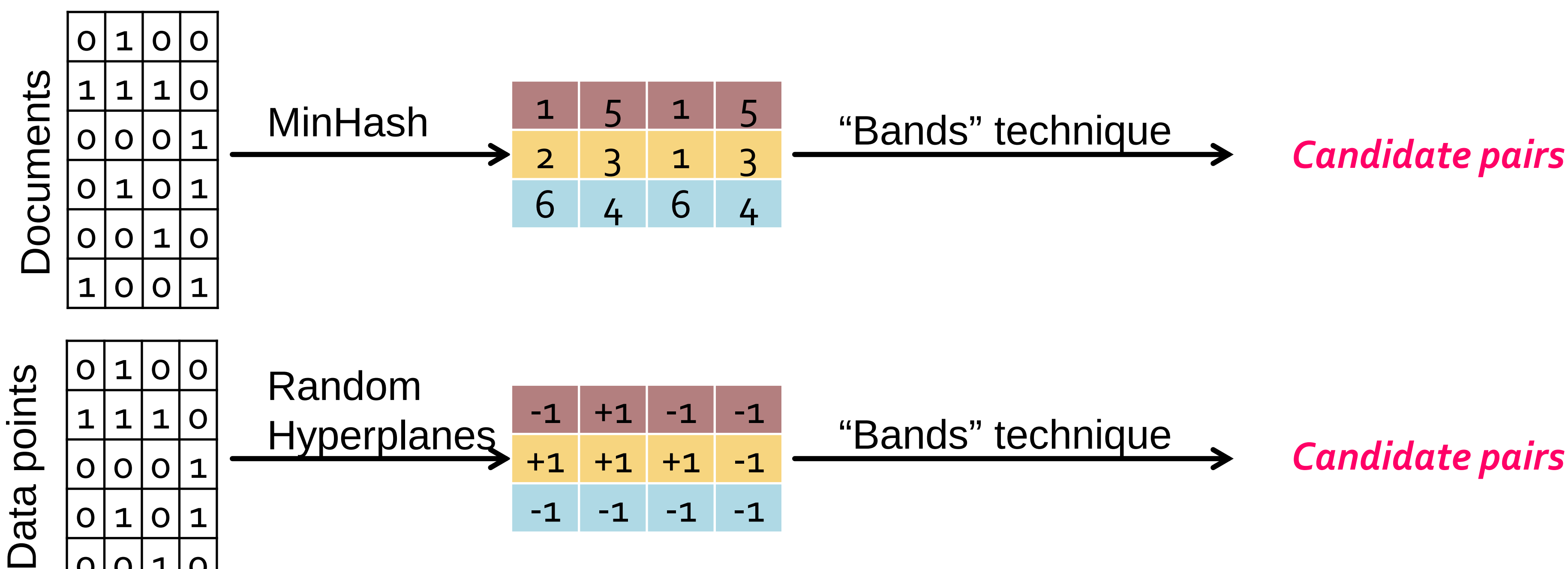
LSH for other Distance Metrics

- **LSH methods for other distance metrics:**
  - **Cosine distance: Random hyperplanes**
  - **Euclidean distance: Project on lines**



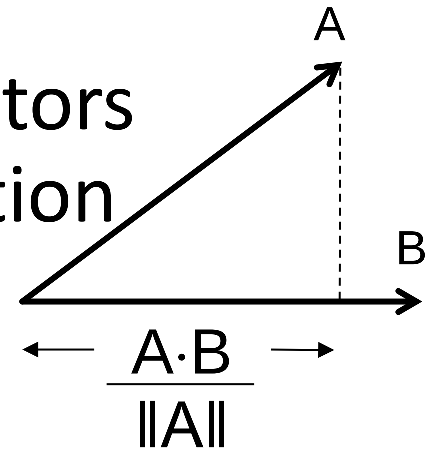
Summary of what we will learn



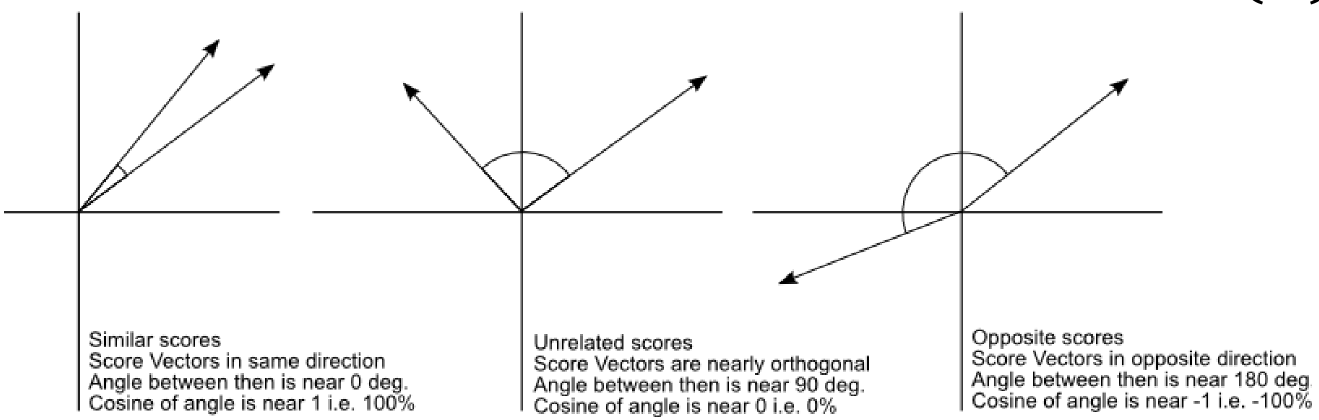


Cosine Distance

- ***Cosine distance*** = angle between vectors from the origin to the points in question
  d(A, B) = θ = arccos(A·B / ‖A‖·‖B‖)
  - Has range $\mathbf{0} \ldots \boldsymbol{\pi}$ (equivalently 0...180°)
  - Can divide θ by $\boldsymbol{\pi}$ to have distance in range 0…1



- **Cosine similarity = 1-d(A,B)**
  - But often defined as **cosine sim:** $\cos(\theta) = \dfrac{A \cdot B}{\|A\|\|B\|}$



- Has range -1…1 for general vectors
- Range 0..1 for non-negative vectors (angles up to 90°)

LSH for Cosine Distance

- For **cosine distance**, there is a technique called **Random Hyperplanes**
  - Technique similar to Min-Hashing

- **Random Hyperplanes** method is a $(d_1, d_2, (1-d_1/\pi), (1-d_2/\pi))$-*sensitive* family for any d_1 and d_2

- **Reminder:** (d_1, d_2, p_1, p_2)-*sensitive*
  1. If $d(x,y) \leq d_1$, then prob. that $h(x) = h(y)$ is at least p_1
  2. If $d(x,y) \geq d_2$, then prob. that $h(x) = h(y)$ is at most p_2

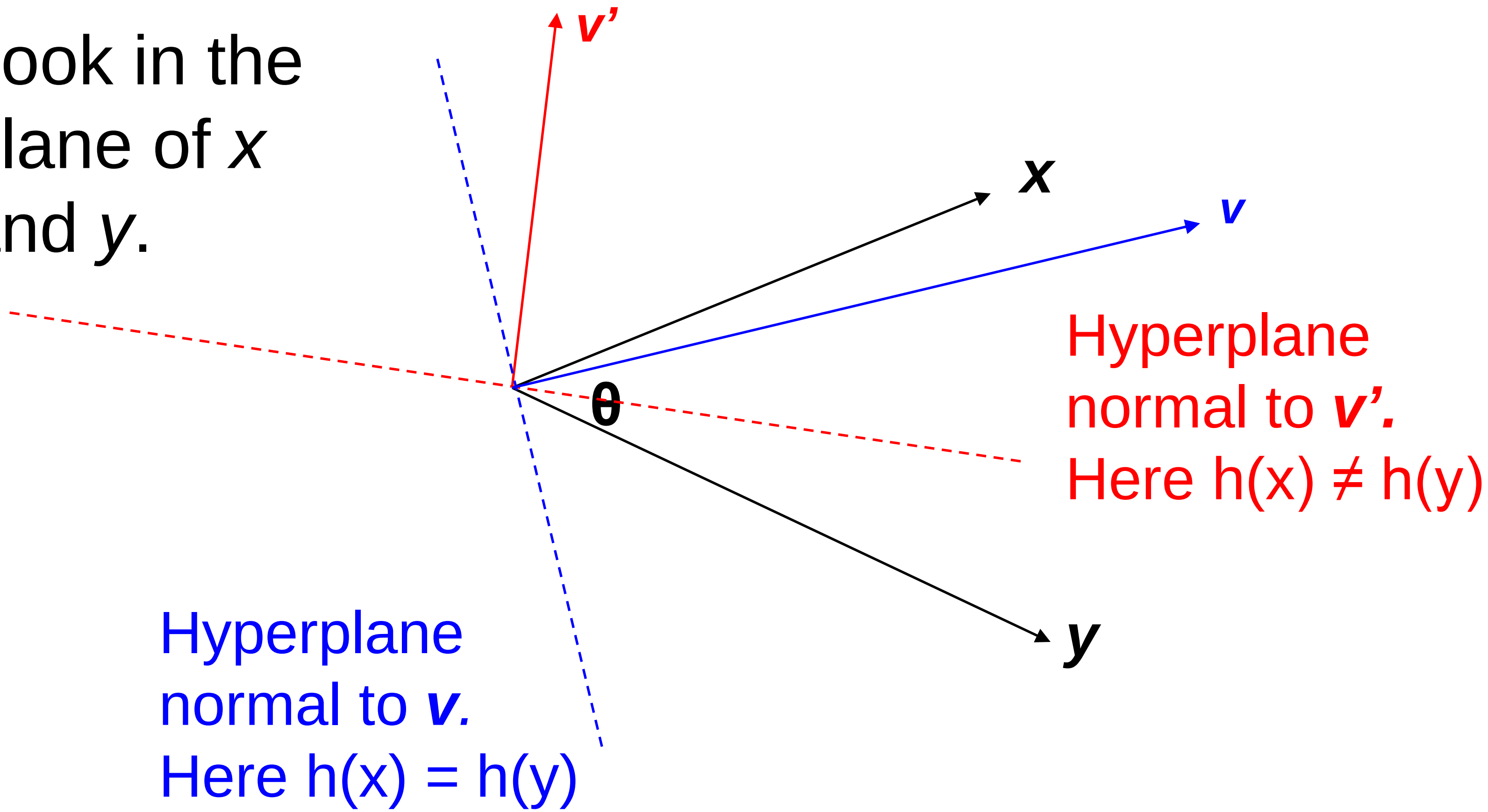
Random Hyperplanes

- Pick a random vector ***v***, which determines a hash function h_v with two buckets
- $\mathbf{h_v(x) = +1}$ if $\mathbf{v \cdot x \geq 0}$; $\mathbf{= -1}$ if $\mathbf{v \cdot x < 0}$
- LS-family ***H*** = set of all functions derived from any vector
- **Claim:** For points **x** and **y**,

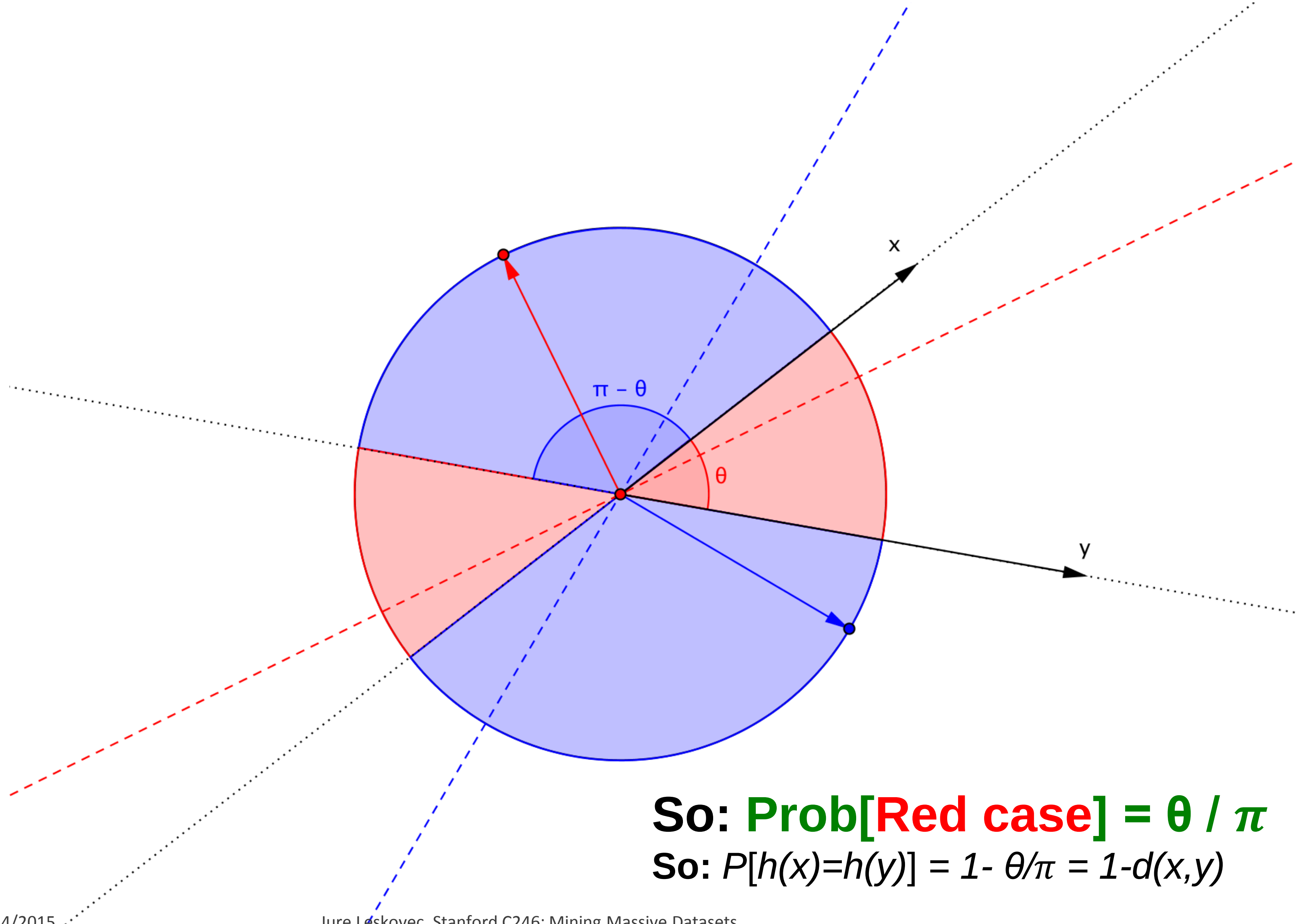
  $$\Pr[h(x) = h(y)] = 1 - d(x,y) / \pi$$

Proof of Claim

Look in the plane of *x* and *y*.



Proof of Claim



So: Prob[Red case] = θ / π

So: $P[h(x)=h(y)] = 1- \theta/\pi = 1-d(x,y)$

Signatures for Cosine Distance

- Pick some number of random vectors, and hash your data for each vector
- The result is a signature (*sketch*) of **+1**’s and **–1**’s for each data point
- Can be used for LSH like we used the Min-Hash signatures for Jaccard distance
- Amplify using **AND/OR** constructions

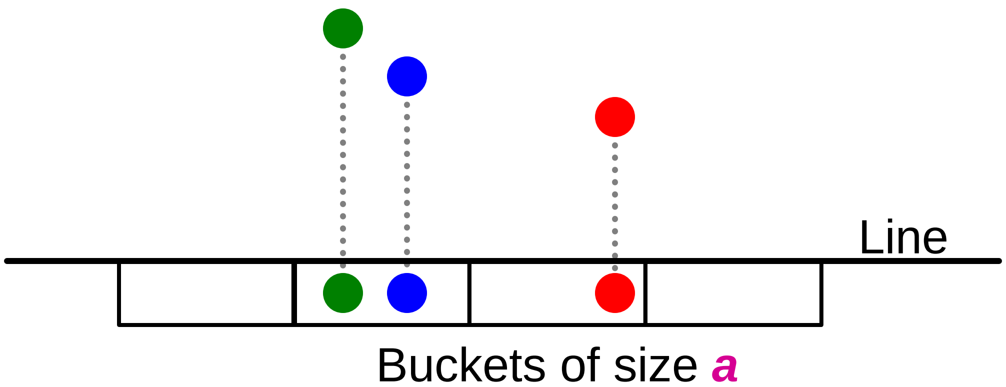
How to pick random vectors?

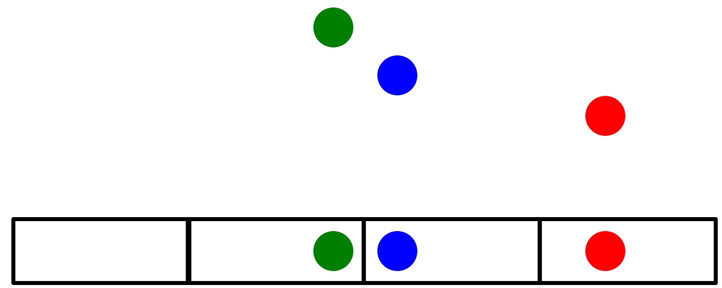
- Expensive to pick a random vector in M dimensions for large M
  - Would have to generate M random numbers
- **A more efficient approach**
  - It suffices to consider only vectors v consisting of +1 and –1 components
  - Why is this more efficient?

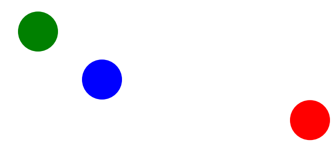
LSH for Euclidean Distance

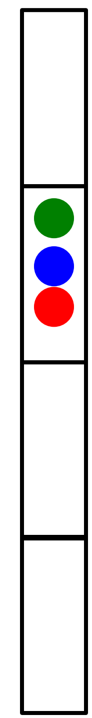
- **Simple idea: Hash functions correspond to lines**
- Partition the line into buckets of size ***a***
- **Hash each point to the bucket containing its projection onto the line**
- **Nearby points are always close**;
  distant points are rarely in same bucket

Projection of Points



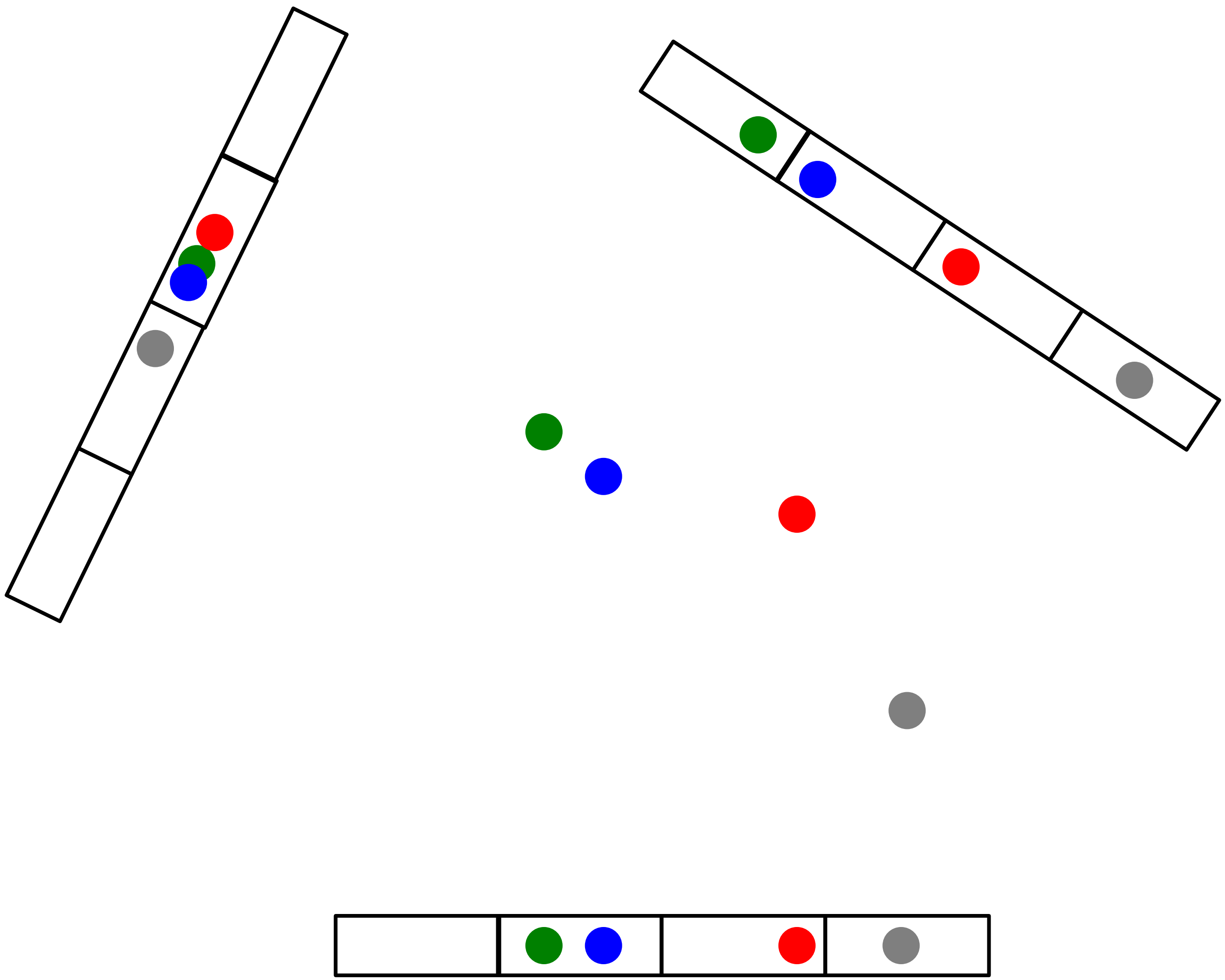




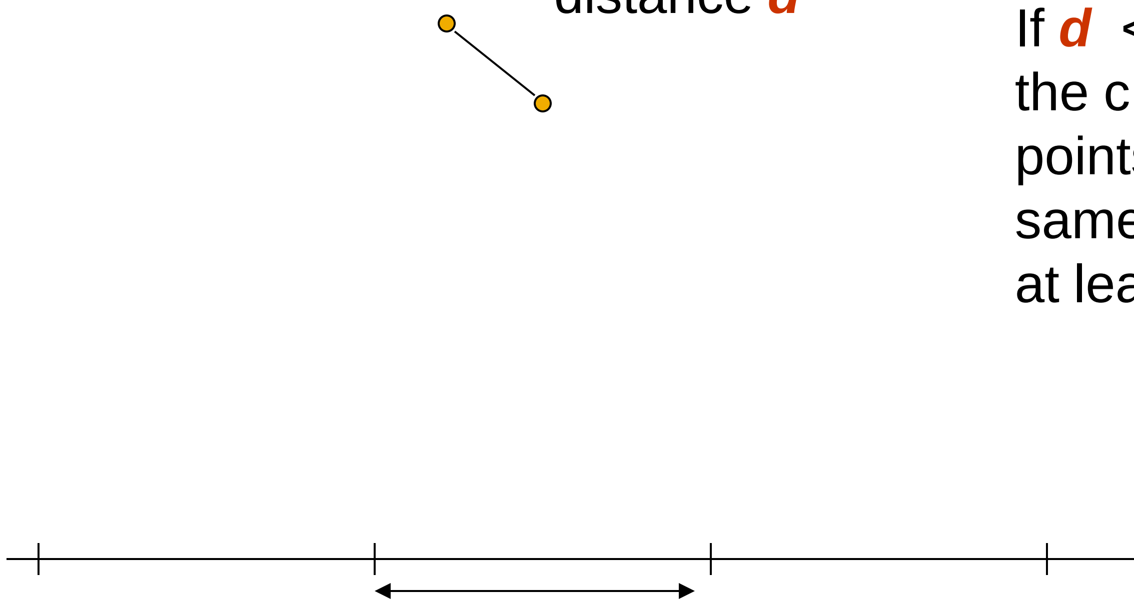


- **"Lucky" case:**
  - Points that are close hash in the same bucket
  - Distant points end up in different buckets
- **Two "unlucky" cases:**
  - **Top:** unlucky quantization
  - **Bottom:** unlucky projection

Multiple Projections



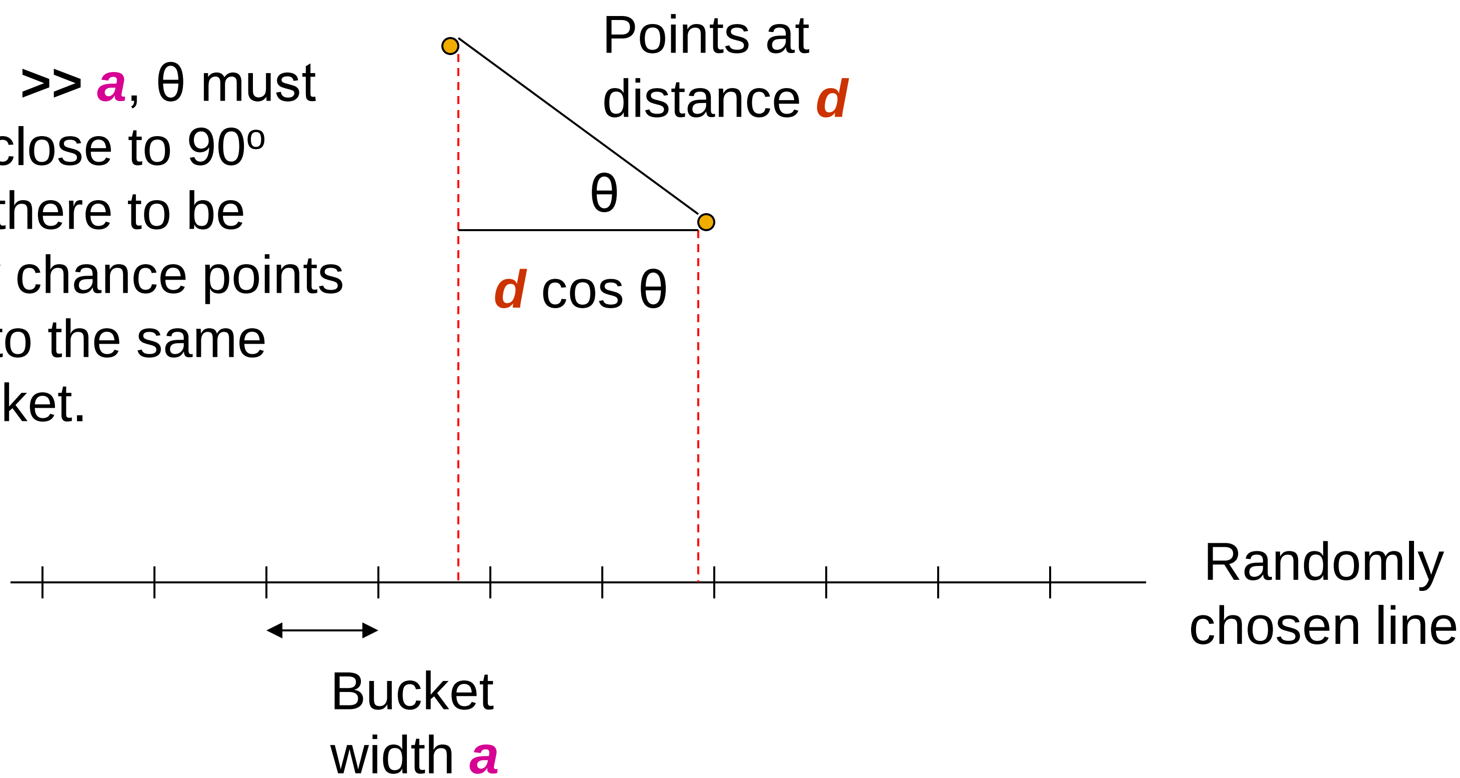
Projection of Points



If ***d*** << ***a***, then the chance the points are in the same bucket is at least **1 –** ***d/a***.

Projection of Points

If ***d*** **>>** ***a***, θ must be close to 90^o for there to be any chance points go to the same bucket.



An LS-Family for Euclidean Distance

- If points are distance $\boldsymbol{d \leq a/2}$, prob. they are in same bucket $\boldsymbol{\geq 1 - d/a = ½}$
- If points are distance $\boldsymbol{d \geq 2a}$ apart, then they can be in the same bucket only if $\boldsymbol{d \cos \theta \leq a}$
  - $\cos \theta \leq ½$
  - $60 \leq \theta \leq 90$, i.e., at most 1/3 probability
- Yields a ***(a/2, 2a, 1/2, 1/3)****-sensitive* family of hash functions for any ***a***
- **Amplify using AND-OR cascades**

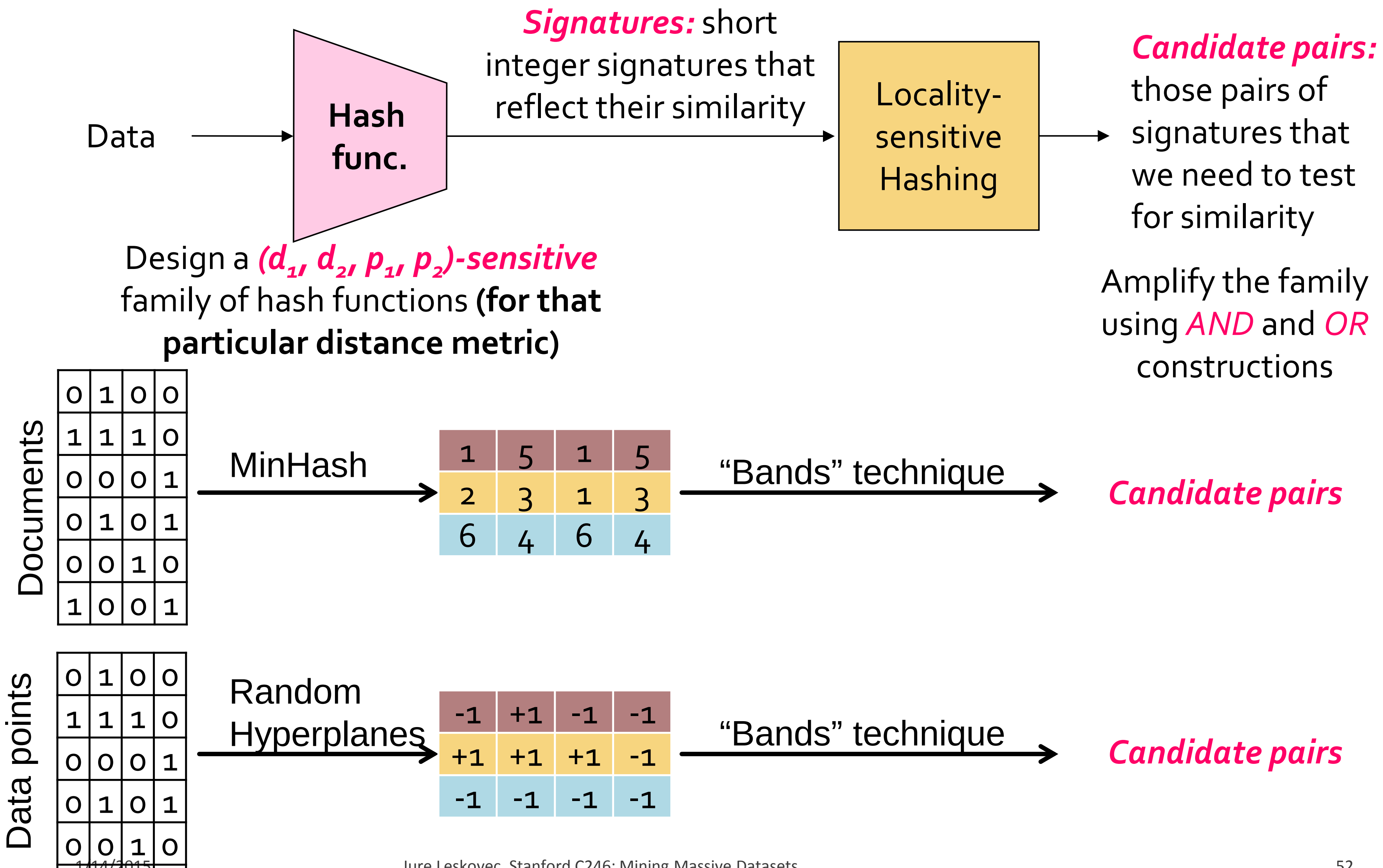
Fixup: Euclidean Distance

- Projection method yields a ***(a/2, 2a, 1/2, 1/3)****-sensitive* family of hash functions
- For previous distance measures, we could start with an (d_1, d_2, p_1, p_2)-sensitive family for any $d_1 < d_2$, and drive p_1 and p_2 to 1 and 0 by **AND/OR** constructions
- Note: Here, we seem to need $d_1 \leq 4\, d_2$
  - In the calculation on the previous slide we only considered cases $d \leq a/2$ and $d \geq 2a$

Fixup – (2)

- But as long as $d_1 < d_2$, the probability of points at distance d_1 falling in the same bucket is greater than the probability of points at distance d_2 doing so
- Thus, the hash family formed by projecting onto lines is an *(d_1, d_2, p_1, p_2)*-sensitive family for **some** $p_1 > p_2$
  - **Then, amplify by AND/OR constructions**

Summary


Data
Hash func.
Signatures: short integer signatures that reflect their similarity
Locality-sensitive Hashing
Candidate pairs: those pairs of signatures that we need to test for similarity
Design a *(d_1, d_2, p_1, p_2)-sensitive* family of hash functions **(for that particular distance metric)**
Amplify the family using *AND* and *OR* constructions
Documents
0 1 0 0
1 1 1 0
0 0 0 1
0 1 0 1
0 0 1 0
1 0 0 1
MinHash
1 5 1 5
2 3 1 3
6 4 6 4
"Bands" technique
Candidate pairs
Data points
0 1 0 0
1 1 1 0
0 0 0 1
0 1 0 1
0 0 1 0
Random Hyperplanes
-1 +1 -1 -1
+1 +1 +1 -1
-1 -1 -1 -1
"Bands" technique
Candidate pairs

Two important points

- Property $P(h(C_1)=h(C_2))=sim(C_1,C_2)$ of hash function h is the essential part of LSH, without it we can't do anything
- LS-hash functions transform data to signatures so that the bands technique (AND, OR constructions) can then be applied