Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Data Mining
Learning from Large Data Sets

Lecture 2 – Nearest neighbor search

263-5200-00L

Andreas Krause

Announcement

- Homework 1 out by tomorrow

Topics

- **Approximate retrieval**
  - Given a query, find "most similar" item in a large data set
  - *Applications*: GoogleGoggles, Shazam, …
- **Supervised learning** (Classification, Regression)
  - Learn a concept (function mapping queries to labels)
  - *Applications*: Spam filtering, predicting price changes, …
- **Unsupervised learning** (Clustering, dimension reduction)
  - Identify clusters, "common patterns"; anomaly detection
  - *Applications*: Recommender systems, fraud detection, …
- **Interactive data mining**
  - Learning through experimentation / from limited feedback
  - *Applications*: Online advertising, opt. UI, learning rankings, …

Today:

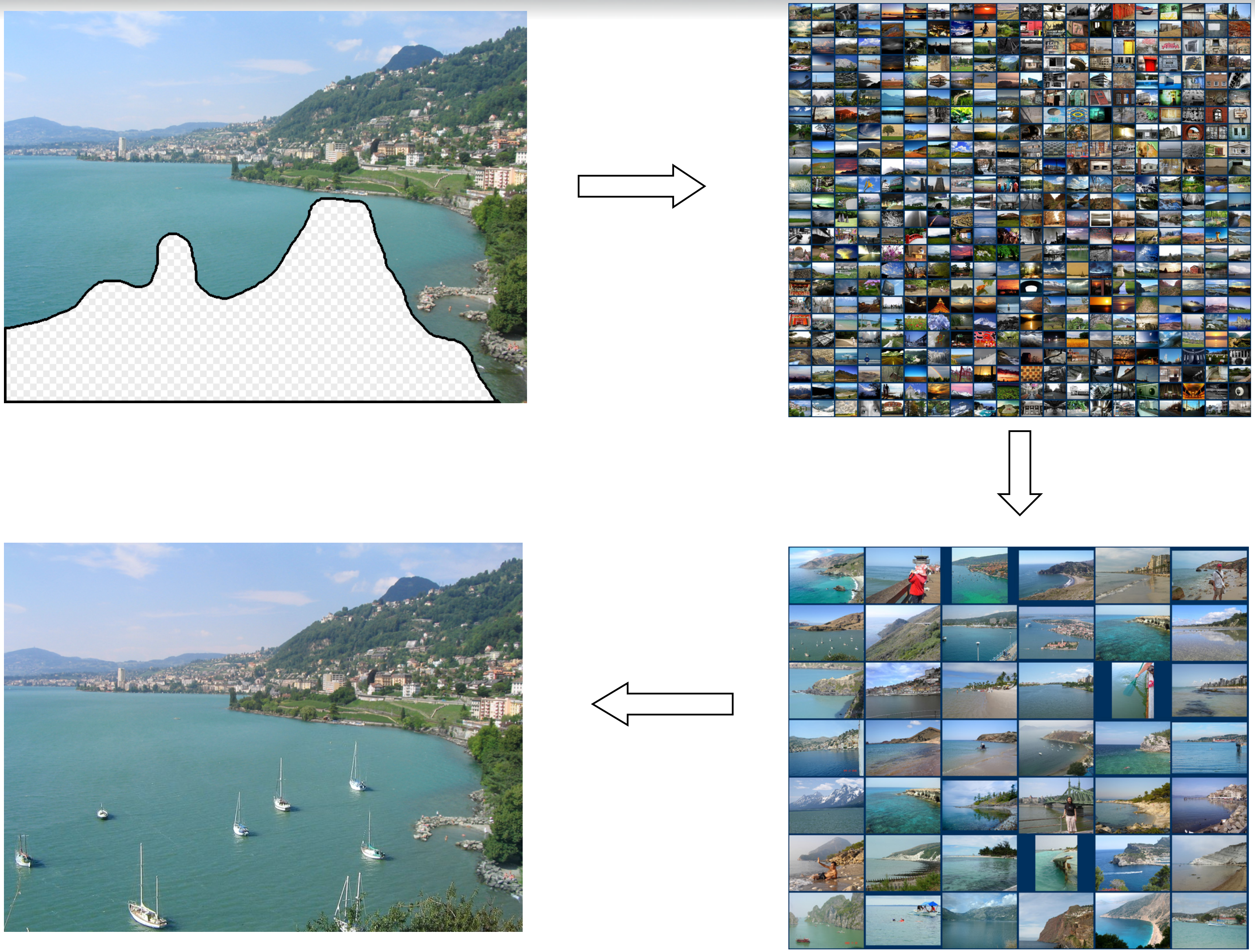
Fast nearest neighbor search in high dimensions

Multimedia retrieval

Google.com

shazam.com

Image completion

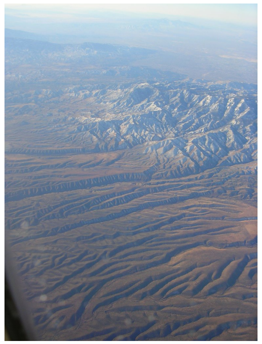


[Hays and Efros, SIGGRAPH 2007]

Nearest-neighbor search

"far"
0,9

"close" 0,01



Properties of distance fn's (metrics)

A function $$d : S \times S \rightarrow \mathbb{R}$$

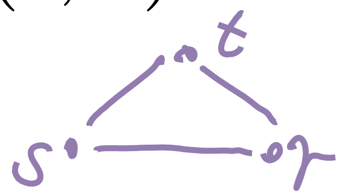
is called a distance function (metric) if it is

Nonnegative: $\forall s, t \in S : d(s,t) \geq 0$

Discerning: $d(s,t) = 0 \Rightarrow s = t$

Symmetric: $\forall s, t : d(s,t) = d(t,s)$

Triangle inequality:



$$\forall s, t, r : d(s,t) + d(t,r) \geq d(s,r)$$

Representing objects as vectors

[.3 .01 .1 2.3 0 0 1.1 …]

The quick brown fox jumps over the lazy dog …

[0 1 0 0 0 1 1 0 1 0 0 0] $\in \mathbb{R}^D$

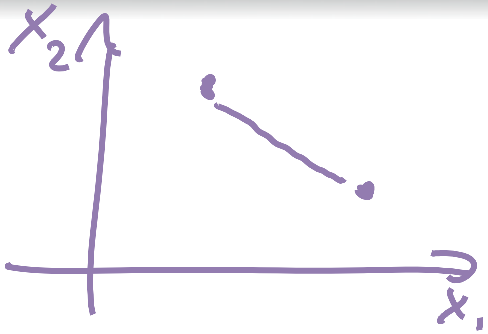
- Often, represent objects as vectors
  - Bag of words for documents
  - Feature vectors for images (SIFT, GIST, PHOG, etc.)
  - …
- Allows to use the same distances / same algorithms for different object types

Examples: Distance of vectors in R^D

- Euclidean distance

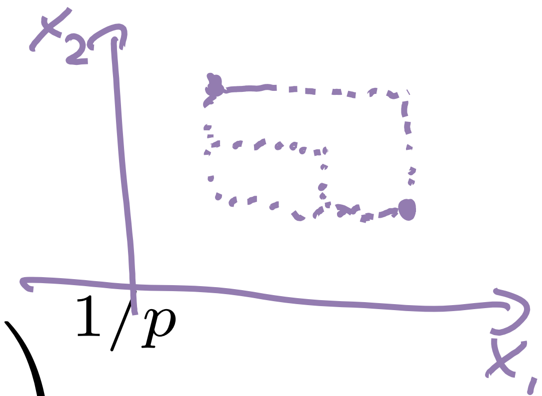
$x = [x_1, \ldots, x_D]$

$$d_2(x,x') = \sqrt{\sum_{i=1}^{D} (x_i - x_i')^2}$$



- Manhattan distance

$$d_1(x,x') = \sum_{i=1}^{D} |x_i - x_i'|$$



- ℓ^p distances:

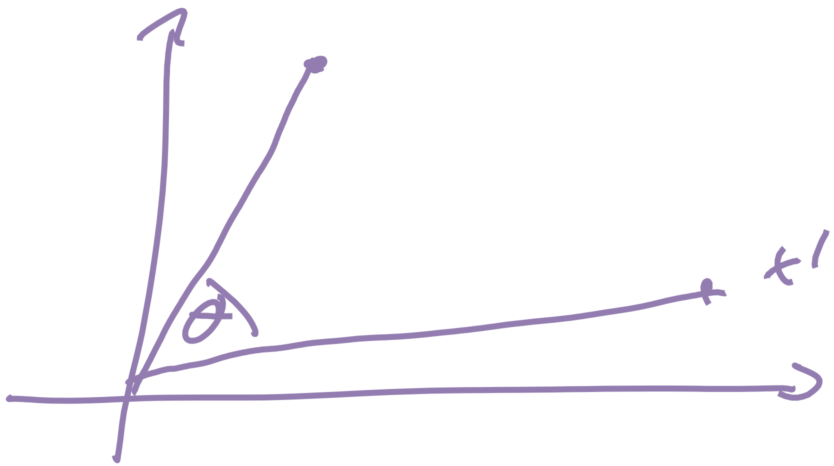
$$d_p(x,x') = \left(\sum_{i=1}^{D} |x_i - x_i'|^p\right)^{1/p}$$

$p=\infty$: $d_\infty(x,x') = \max_i |x_i - x_i'|$

Cosine distance

- Cosine distance

$$d(x, x') = \arccos \frac{x^T x'}{||x||_2 ||x'||_2} = \theta$$



Edit distance

Edit distance: How many inserts and deletes are necessary to transform one string to another?

Example:

- d("The quick brown fox","The quikc brwn fox") =3
- d("GATTACA","ATACAT")

- Allows various extensions (mutations; reversal; …)
- Can compute in polynomial time, but expensive for large texts

➔ We will focus on vector representation

Many real-world problems are high-dimensional

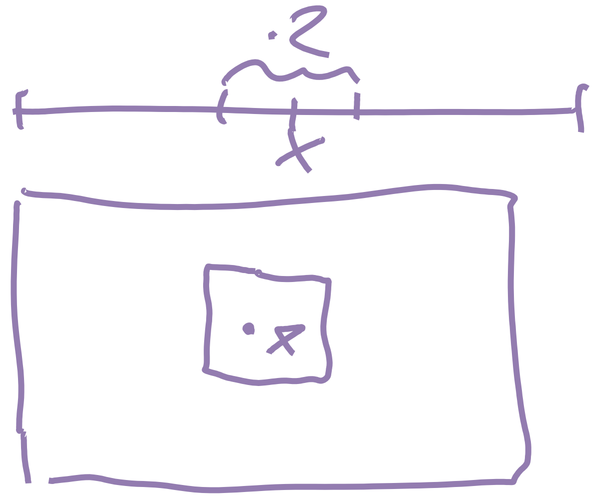
- Text on the web
  - Billions of documents, millions of terms
  - In Bag Of Words representation, each term is a dimension..
- Scene completion, image classification, …
  - Large # of image features
- Scientific data
  - Large number of measurements
- Product recommendations and advertising
  - Millions of customers, millions of products
  - Traces of behavior (websites visited, searches, …)

Curse of dimensionality

- Suppose we would like to find neighbors of maximum distance at most .1 in $[0,1]^D$
- Suppose we have N data points sampled uniformly at random from $[0,1]^D$

D = 1 $\quad \mathbb{E}[\#\text{ Nbrs}] = \frac{N}{5}$

D = 2 $\quad \mathbb{E}[\#\text{ Nbrs}] = \frac{N}{5^2}$



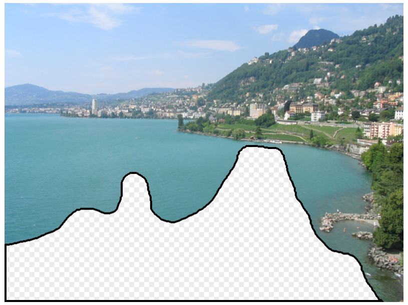
general D $\quad$ " $= \frac{N}{5^D}$

Curse of dimensionality

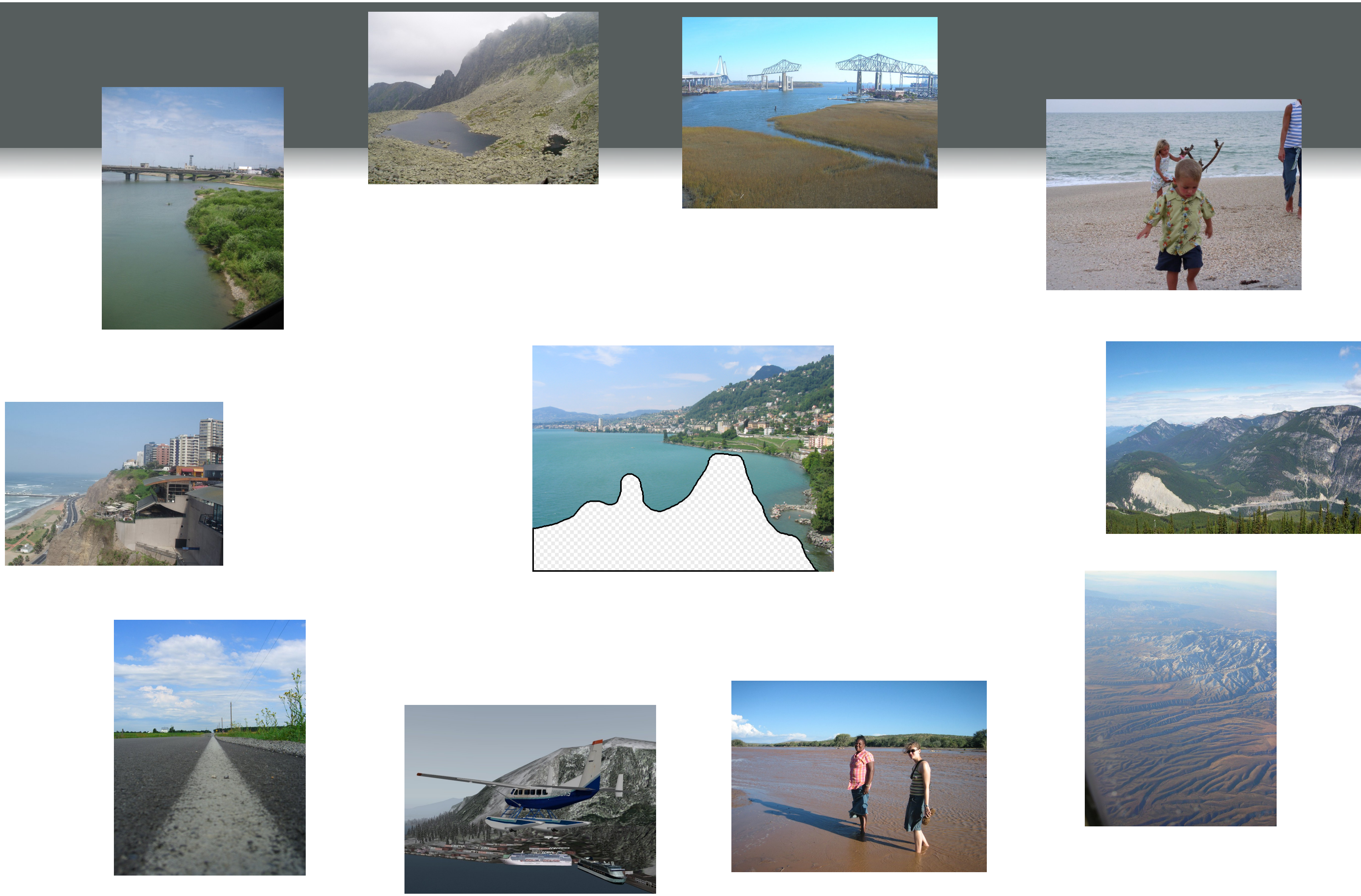
- **Theorem [Beyer et al. '99]** Fix ε>0 and N. Under fairly weak assumptions on the distribution of the data

$$\lim_{D\to\infty} P[d_{\max}(N,D) \leq (1+\varepsilon) d_{\min}(N,D)] = 1$$

The Blessing of Large Data

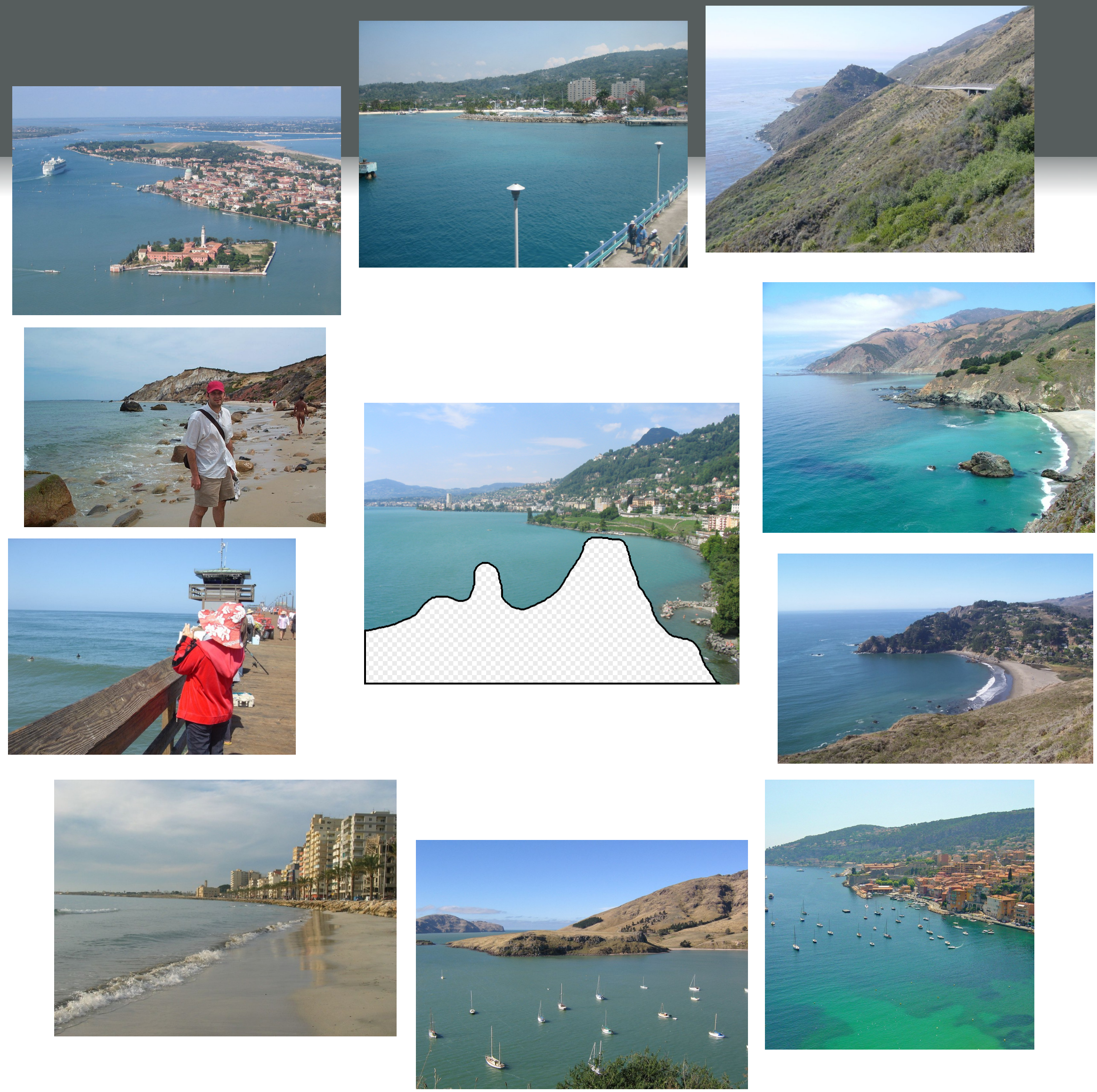


Hays and Efros, SIGGRAPH 2007



10 nearest neighbors from a collection of 20,000 images

Hays and Efros, SIGGRAPH 2007



10 nearest neighbors from a collection of 2 million images

Hays and Efros, SIGGRAPH 2007

Application: Find similar documents

- Find "near-duplicates" among a large collection of documents
  - Find clusters in a document collection (blog articles)
  - Spam detection
  - Detect plagiarism
  - …

- What does "near-duplicates" mean?

Near-duplicates

- Naïve approach:
  - Represent documents as “bag of words”
  - Apply nearest-neighbor search on resulting vectors
- Doesn’t work too well in this setting.

Shingling

- To keep track of word order, extract k-shingles (aka k-grams)
- Document represented as "~~bag~~ set of k-shingles"
- Example: *a b c a b*

2 shingles = { ab, bc, ca }

Shingling implementation

- How large should one choose k?
  - Long enough s.t. the don't occur "by chance"
  - Short enough so that one expects "similar" documents to share some k-shingles

- Storing shingles
  - Want to save space by compressing
  - Often, simply hashing works well (e.g., hash 10-shingle to 4 bytes)

Comparing shingled documents

- Documents are now represented as sets of shingles
- Want to compare two sets
- E.g.: A={1,3,7}; B={2,3,4,7}

Ovelap $|A \cap B| = 2$

Total # $|A \cup B| = 5$

Jaccard distance

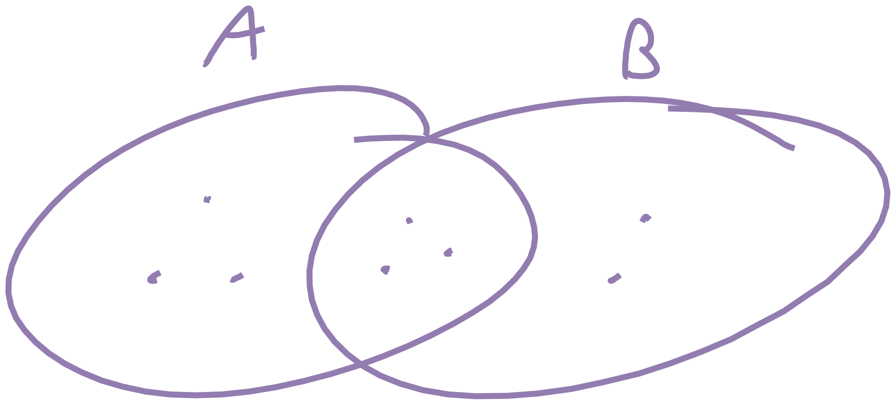
- Jaccard similarity:

$$\mathrm{Sim}(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad \in [0, 1]$$

- Jaccard distance:

$$d(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

Example



$$Sim(A, B) = \frac{3}{8}$$

$$d(A, B) = \frac{5}{8}$$

Near-duplicate detection

- Want to find documents that have similar sets of k-shingles
- Naïve approach:
- For i=1:N
  - For j=1:N
    - Compute d(i,j)
    - If d(i,j) < ε then declare near-duplicate

$O(N^2 D)$

- **Infeasible even for moderately large N** ☹

- **Can we do better??**

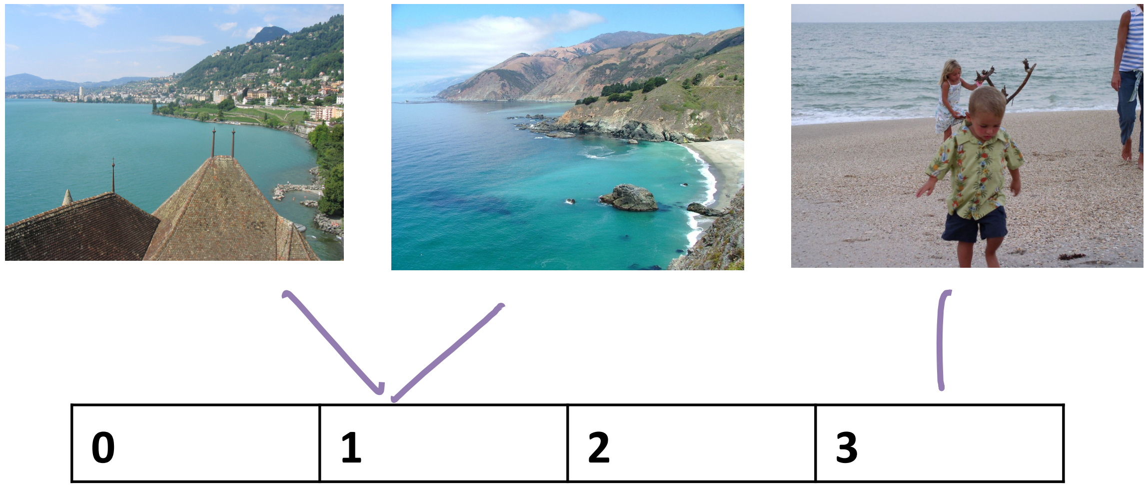
Warm-up

- Given a large collection of documents, determine whether there exist **exact** duplicates?

- Compute hash code / checksum (e.g., MD5) for all documents
- Check whether the same checksum appears twice
- Both can be easily parallelized

Locality sensitive hashing

- **Idea**: Create hash function that maps "similar" items to same bucket



Hashtable

- **Key problem**: Is it possible to construct such hash functions??
  - Depends on the distance function
  - Possible for Jaccard distance!! ☺
  - Some other distance functions work as well

Shingle Matrix

documents

shingles

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

Min-hashing

- Simple hash function, constructed in the following way:

- Use random permutation π to reorder the rows of the matrix
  - Must use same permutation for all columns C!!
- *h*(*C*) = minimum row number in which permuted column contains a 1

$$h(C) = h_\pi(C) = \min_{i:C(i)=1} \pi(i)$$

Min-hashing example

Input matrix

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

⇒ [2, 1, 2, 1]

Min-hashing example

Input matrix

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

h →

2	1	2	1

Min-hashing property

- Want that similar documents (columns) have same value of hash function (with high probability)

- Turns out it holds that

$$\Pr[h(C_1) = h(C_2)] = \mathrm{Sim}(C_1, C_2)$$

Proof

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

$$Sim(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

4 cases		# Occ
1	1	a
1	0	b
0	1	c
0	0	d

$$Sim(C_1, C_2) = \frac{a}{a+b+c}$$

Proof

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

Step through rows in π-order

Stop upon row that contains <u>at least</u> one 1

What's the prob. that row is of type [1 1]

$$P(\text{"}) = \frac{a}{a+b+c}$$

□

Near-duplicate search with Min-Hashing

- Suppose we would like to find all duplicates with more than 90% similarity
- Apply min-hash function to all documents, and look for candidate pairs (documents hashed to same bucket)
- How many 90%-duplicates will we find? ≈ 90%
- How many 90%-duplicates will we miss? ≈ 10%
- How can we reduce the number of misses?

Reducing the "misses"

- Apply multiple *independently random* hash functions
- Consider candidate pair of near duplicates if at least one of the functions hashes to same bucket
- What's the probability of a "miss" with k functions?

$$P(\text{"miss"}) = d(C_1, C_2)^k$$

$$= (1 - s)^k \qquad s = sim(C_1, C_2)$$

Example

- Thus, using multiple independent hash functions can exponentially reduce probability of **misses!**

Min-hash signatures

Input matrix

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

Signature matrix *M*

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

Similarities:

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

Implementing min-hashing

- Difficult to randomly permute a data set with a billion rows
- Even representing a permutation of size 10^9 is expensive
- Accessing rows in permuted order is infeasible (requires random access)

Approximate min-hashing

- Directly represent permutation π through hash function h!

$$\pi(i) = h(i) \equiv ai + b \mod n$$

  - Could happen that h(i)=h(j) for i ≠ j, but this is rare for good h
- **Note**: Will use same notation for h(r) and h(C)

$$h(C) = \min_{i:C(i)=1} h(i)$$

- Suppose h(r)<h(s). Then row r appears before s in π
- Why is this useful?
- Can store h very efficiently
- Allows to process data matrix row-wise..

Example

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

$h(x) = x \bmod 5$

$h(1)=1$, $h(2)=2$, $h(3)=3$, $h(4)=4$, $h(5)=0$

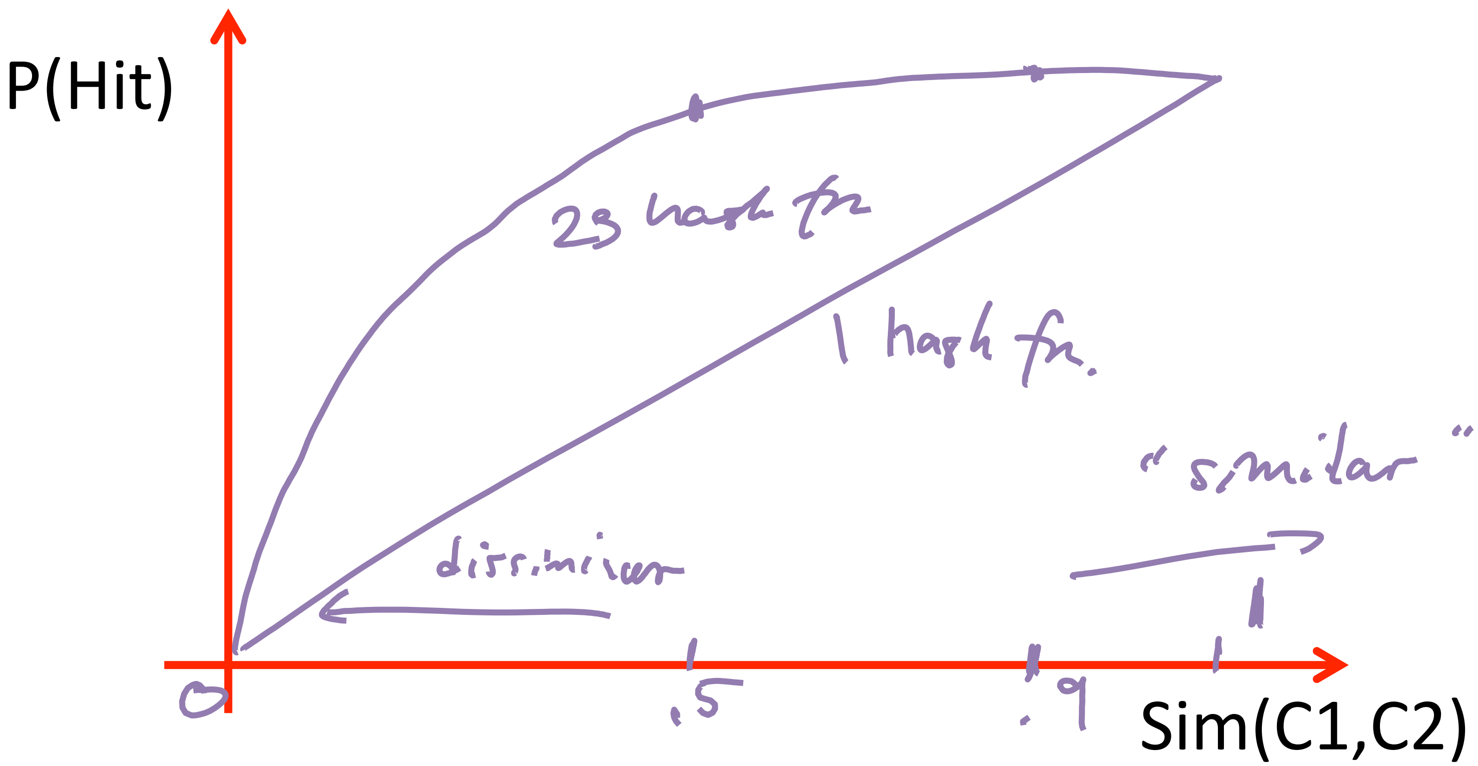
$g(x) = 2x+1 \bmod 5$

$g(1)=3$, $g(2)=0$, $g(3)=2$, $g(4)=4$, $g(5)=1$

$$M = \begin{matrix} 1 & 0 \\ 2 & 0 \end{matrix}$$

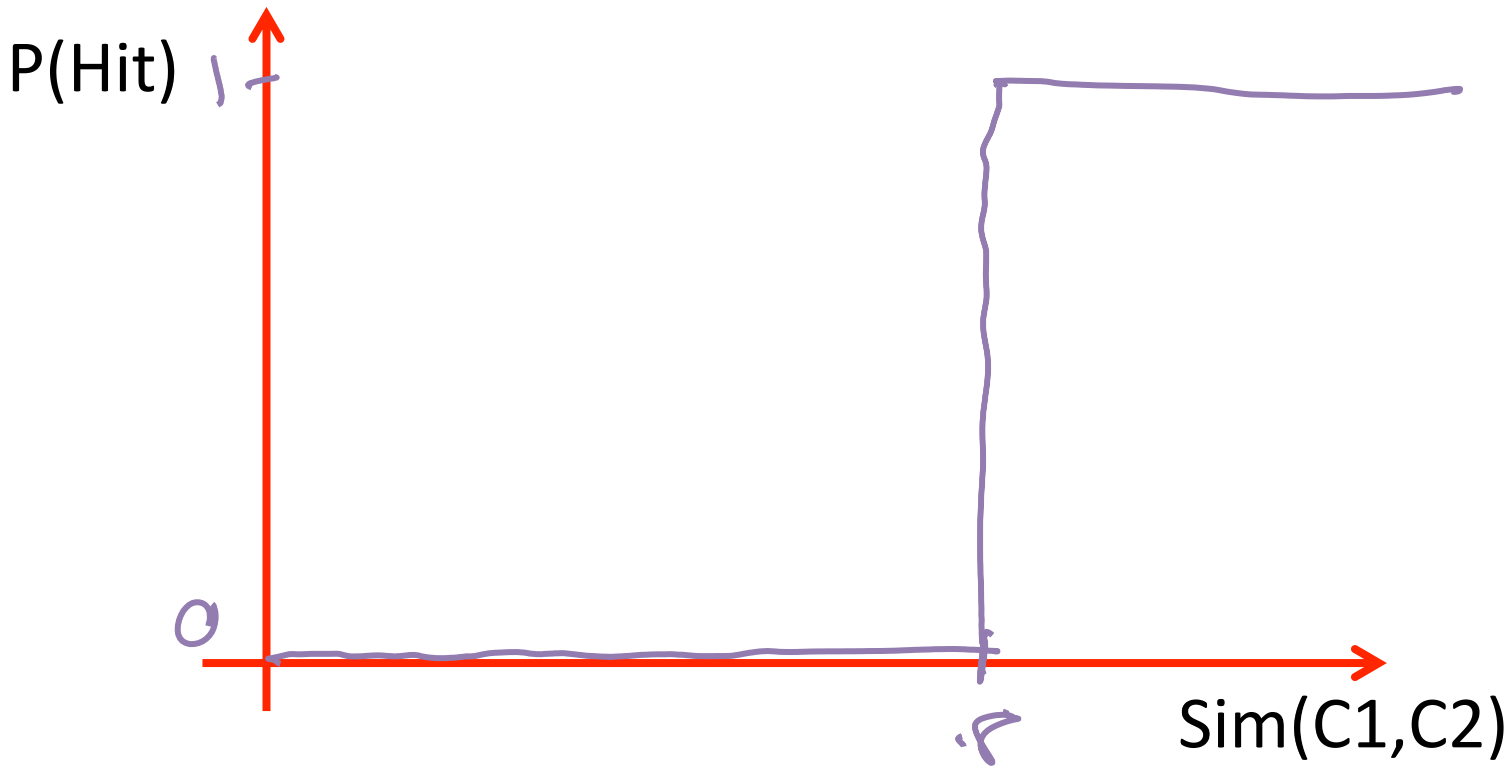
False positives

- Increasing number of hash tables reduces false negative rate ☺
- Also increases false positive rate ☹



False positives

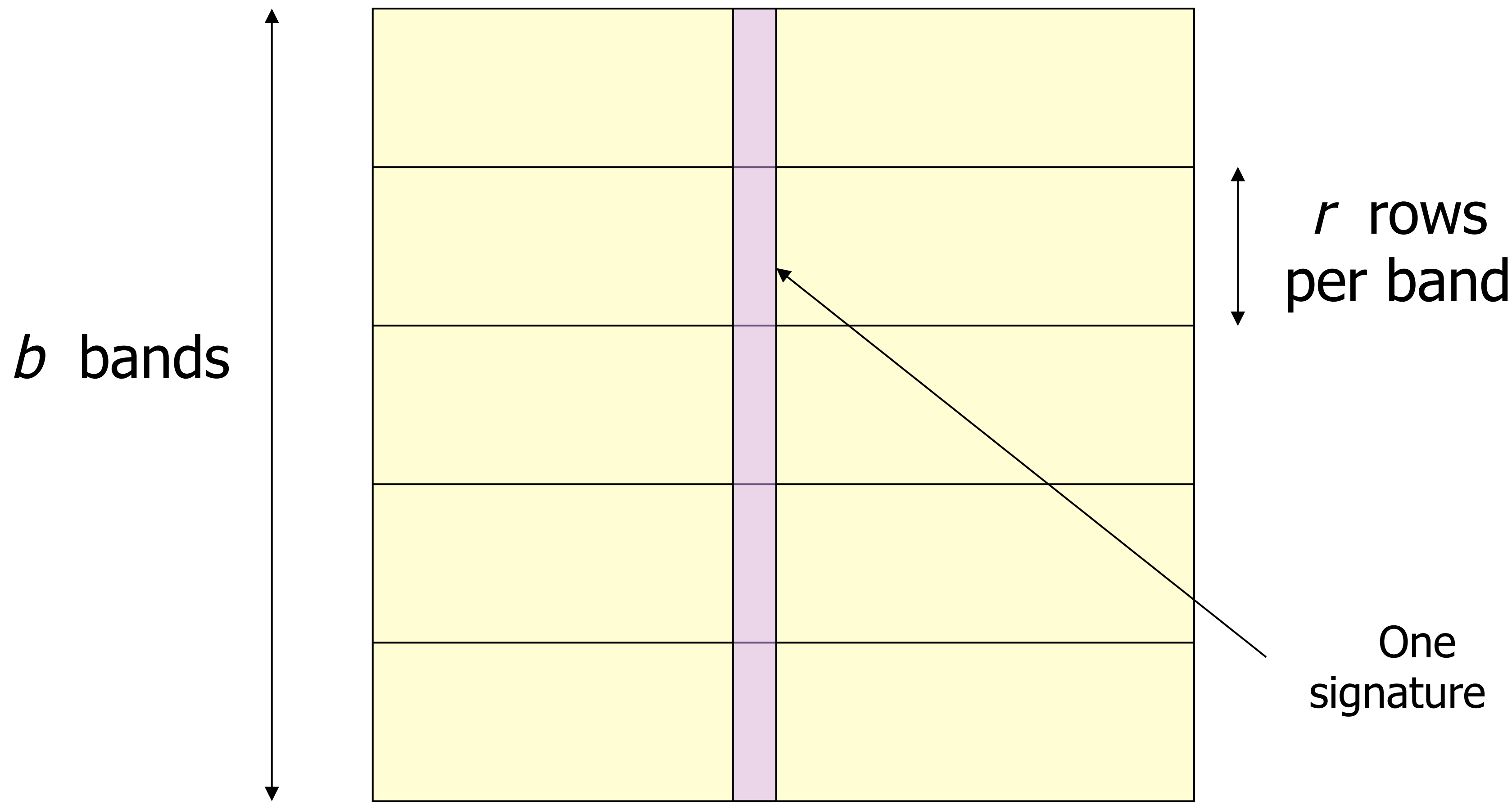
- Ideally want:



Ingenious trick

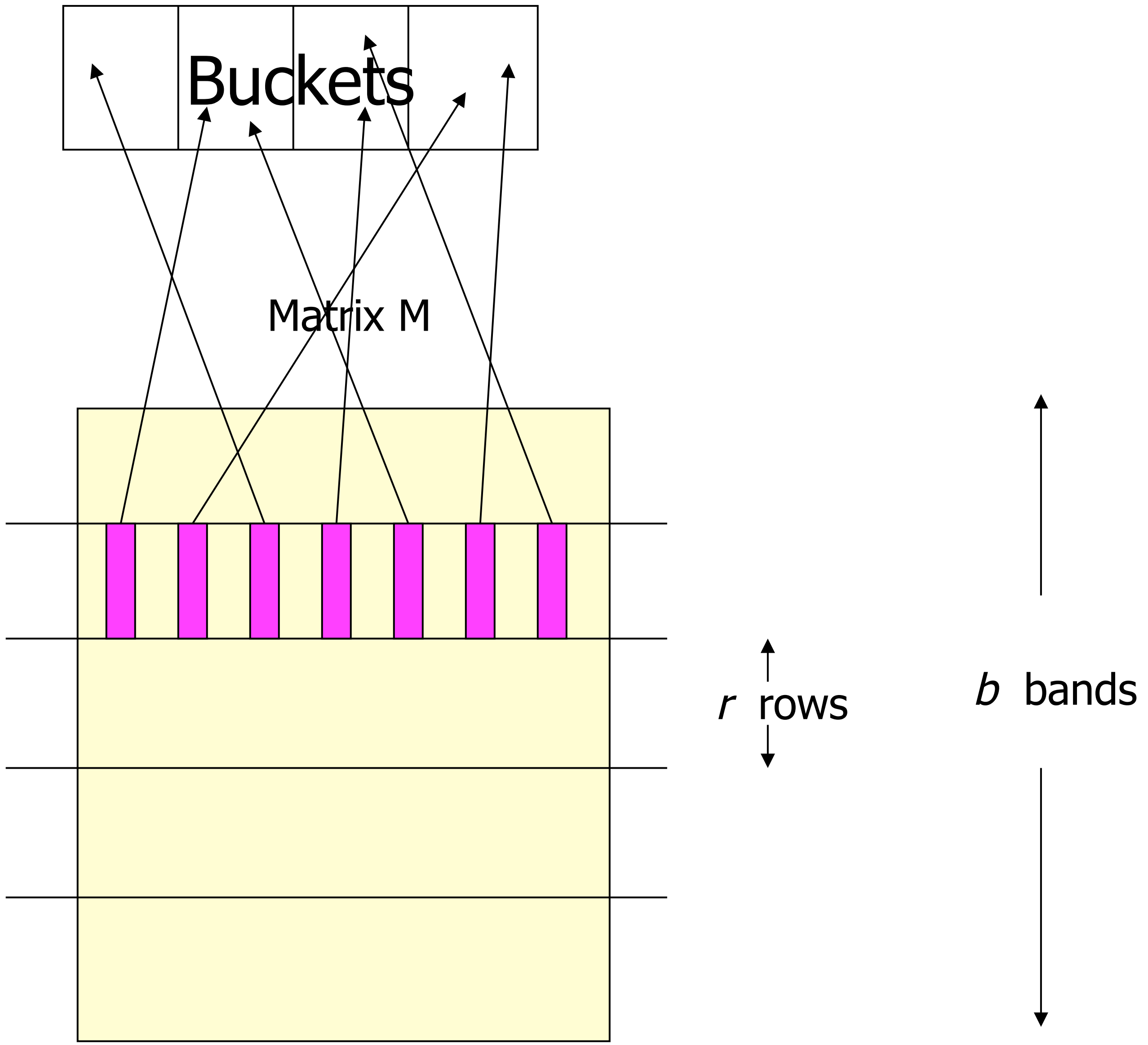
- Signature matrix compactly represents similarity between documents
  - Jaccard distance ~ l1-distance of columns
  - Similar documents have similar signatures
- Naïve approach: Compare any pair of columns to see if their similar
  - Compact representation → faster
  - Still N^2 comparisons ☹
- Will see how to hash columns s.t. with high probability
  - return similar pairs (d(C1,C2) < ε)
  - do not return dissimilar pairs (d(C1,C2) > ε)

Partitioning the signature matrix



Signature Matrix *M*

Hashing bands of M



Hashing the signature matrix

- Signature matrix *M* partitioned into *b* bands of *r* rows.
- One hash table per band, independent hash functions
- For each band, hash its portion of each column to its hash table
  - For purpose of analysis, let's assume there's no "false collisions"
  - Doesn't affect correctness of algorithm
- Candidate pairs are columns that hash to the same bucket for at least one band.
- Why is this useful?

Analysis of partitioning

- Suppose columns M1 and M2 have similarity *s*

$M_i = [B_{i,1} \ldots B_{i,b}]$

For fixed band j, what's the prob. that $B_{1,j}$ and $B_{2,j}$ collide?

$P(h(B_{1,j}) = h(B_{2,j})) = s^r$

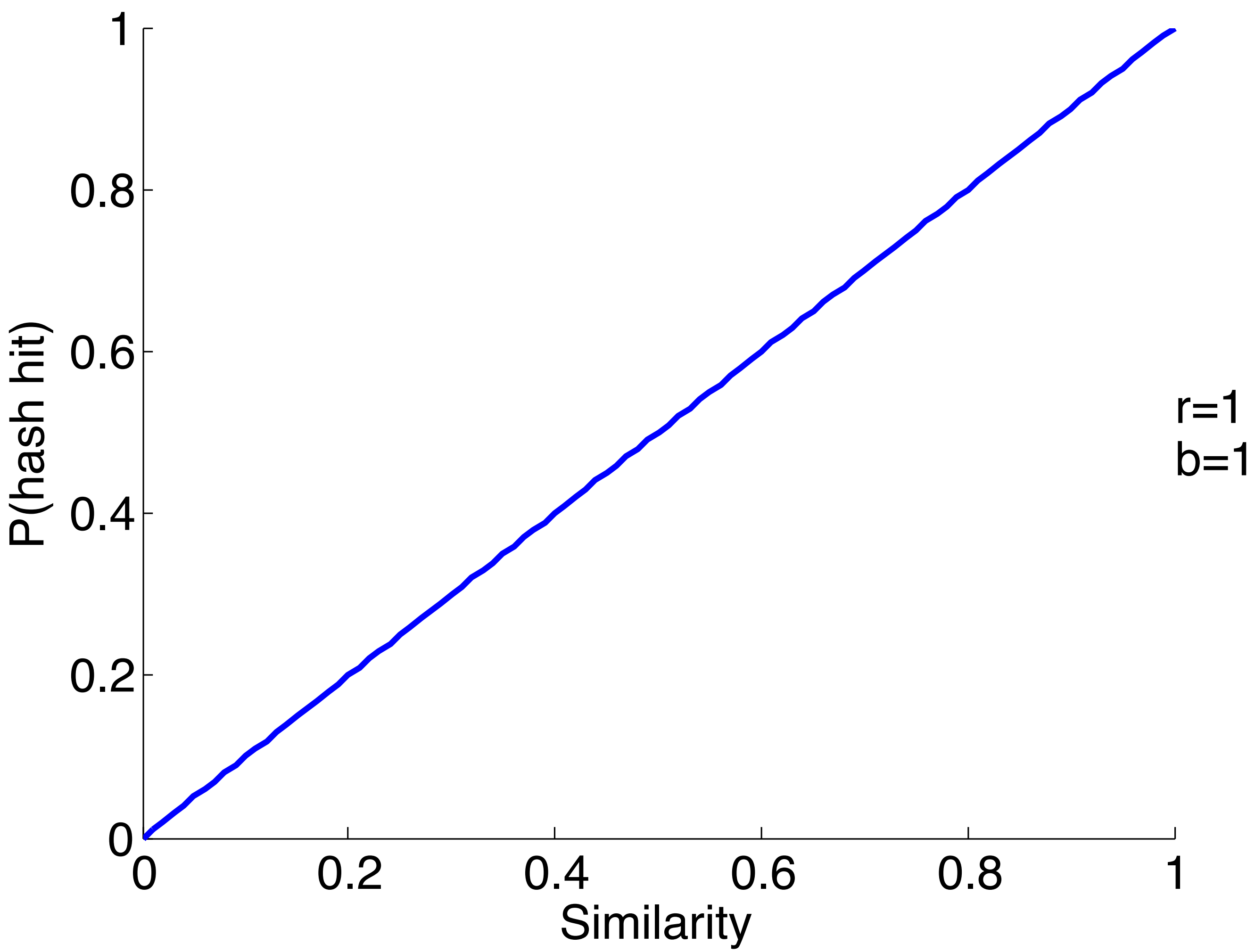
$P(h(B_{1,j}) \neq h(B_{2,j})) = 1 - s^r$

"no collision in jth band"

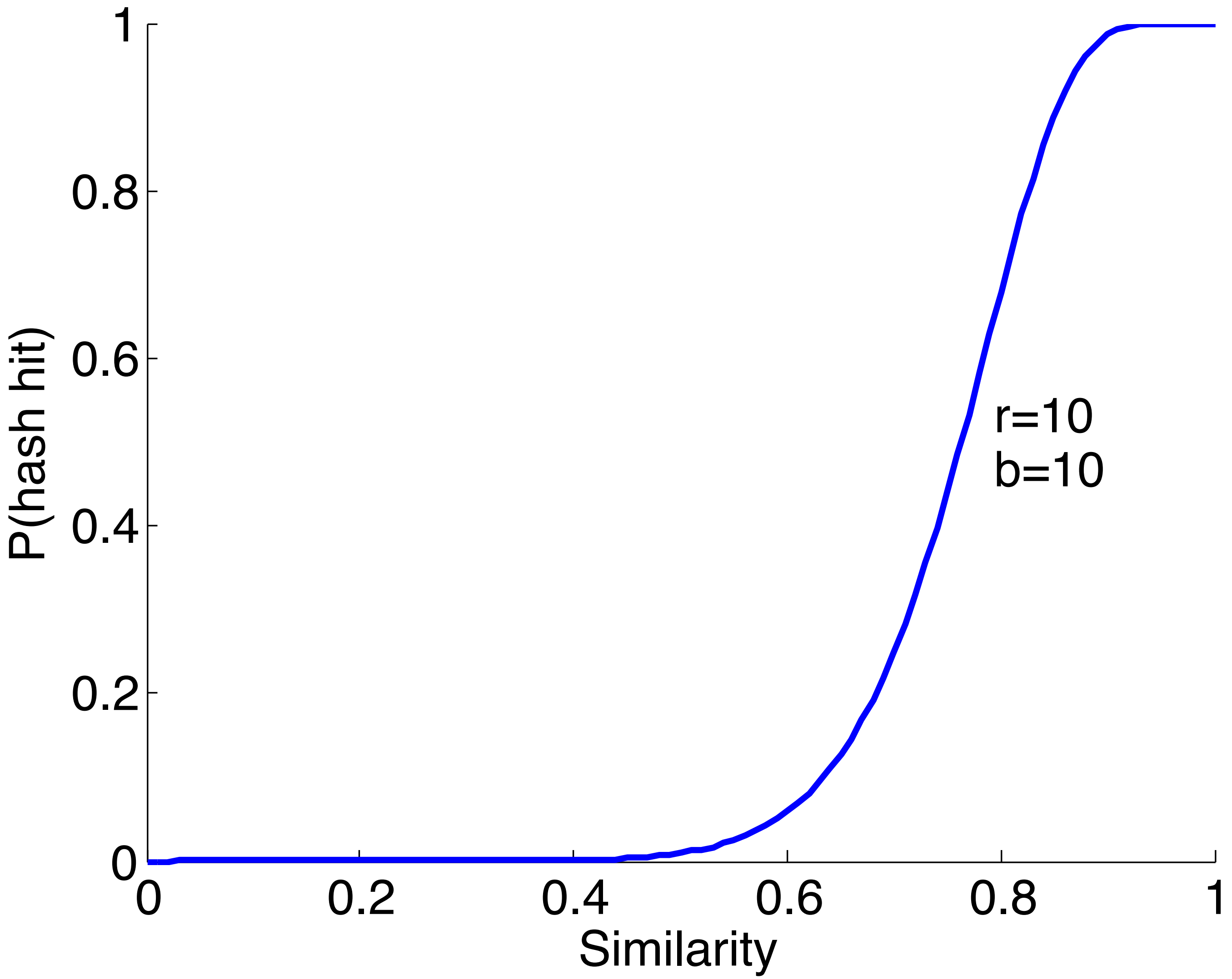
$P(\text{no collision in } \underline{\text{any}} \text{ band}) = (1 - s^r)^b$

$P(\text{collision in } \underline{\text{some}} \text{ band}) = 1 - (1 - s^r)^b$

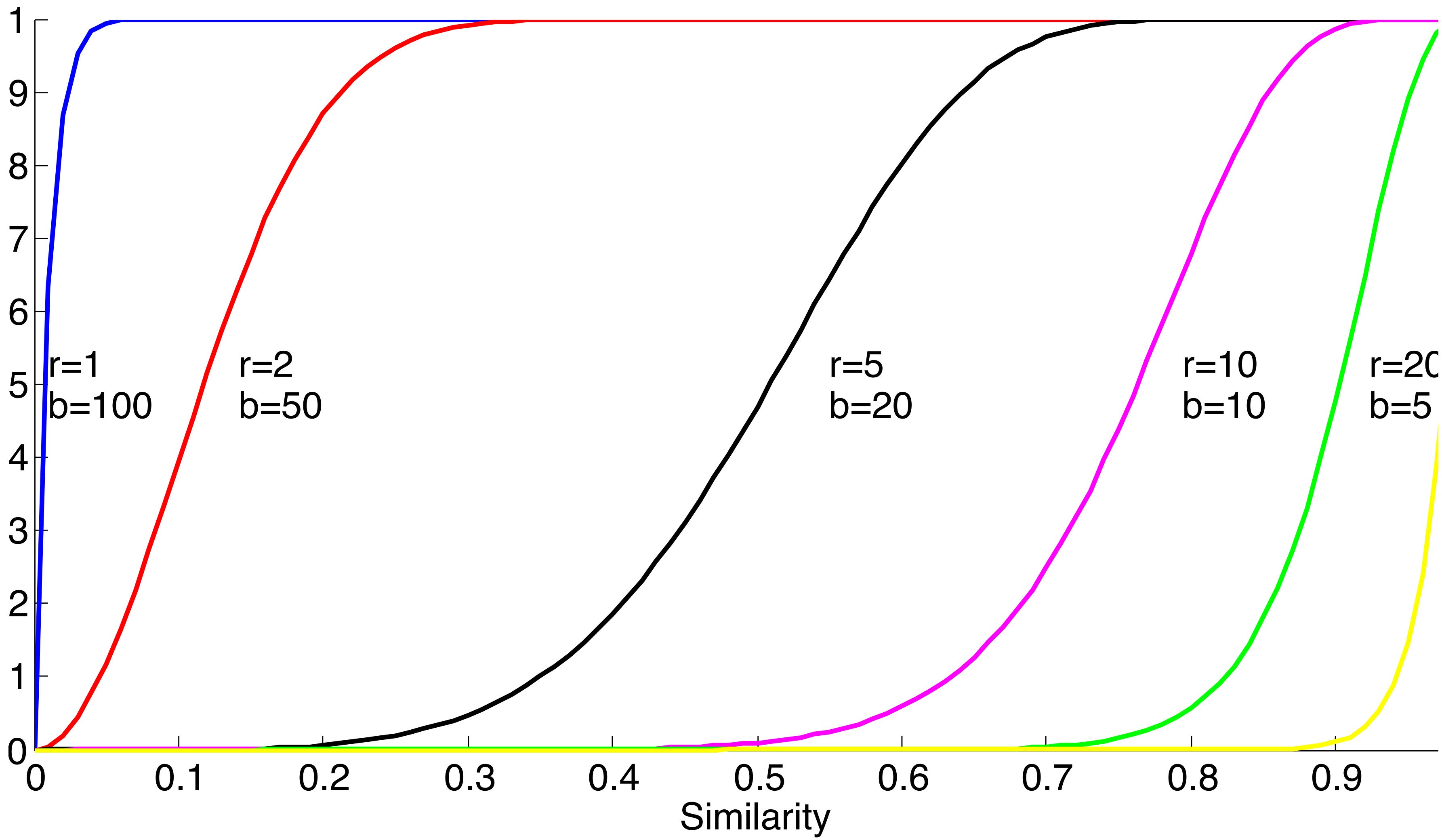
One hash function



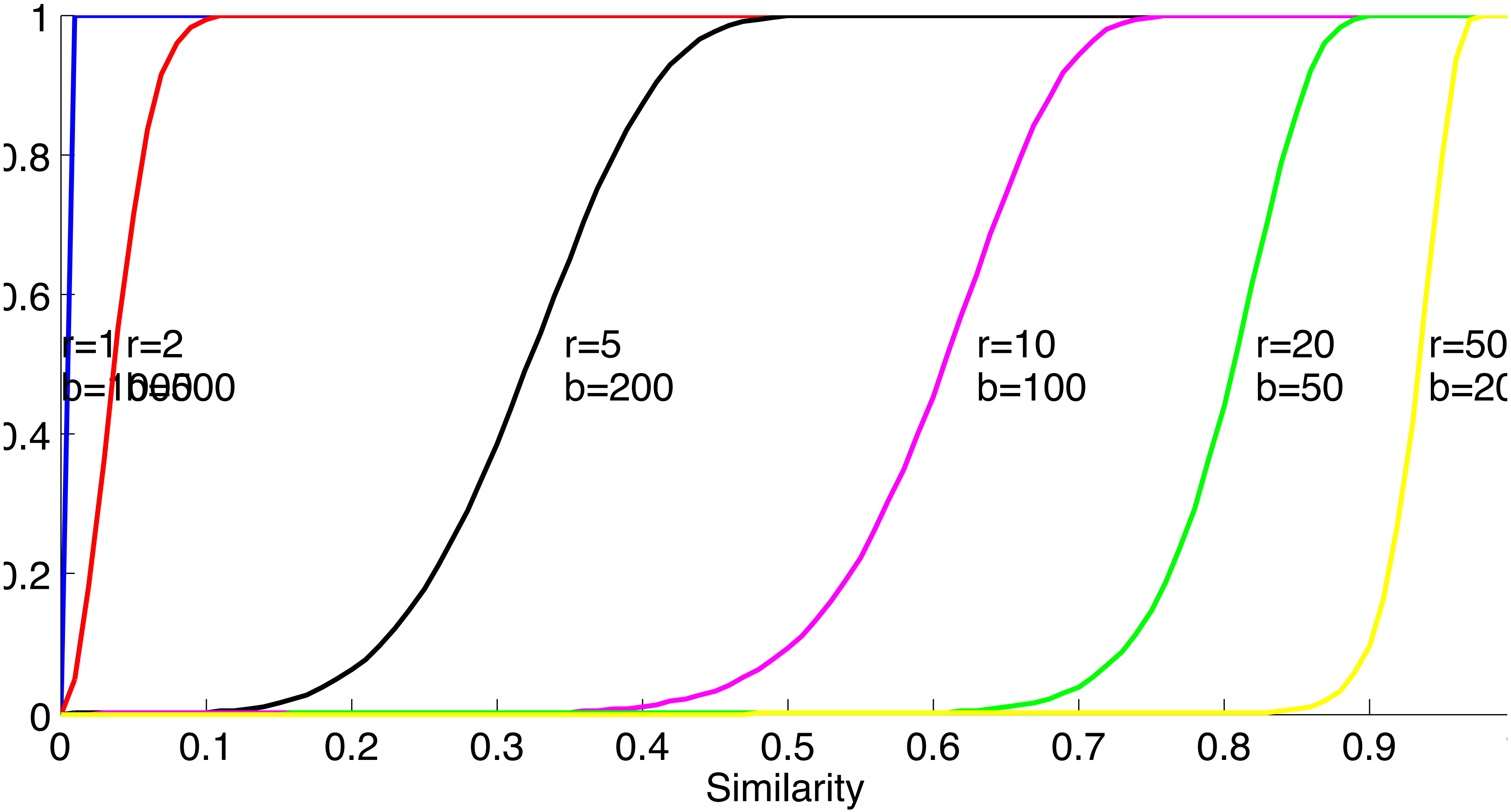
100 hash functions



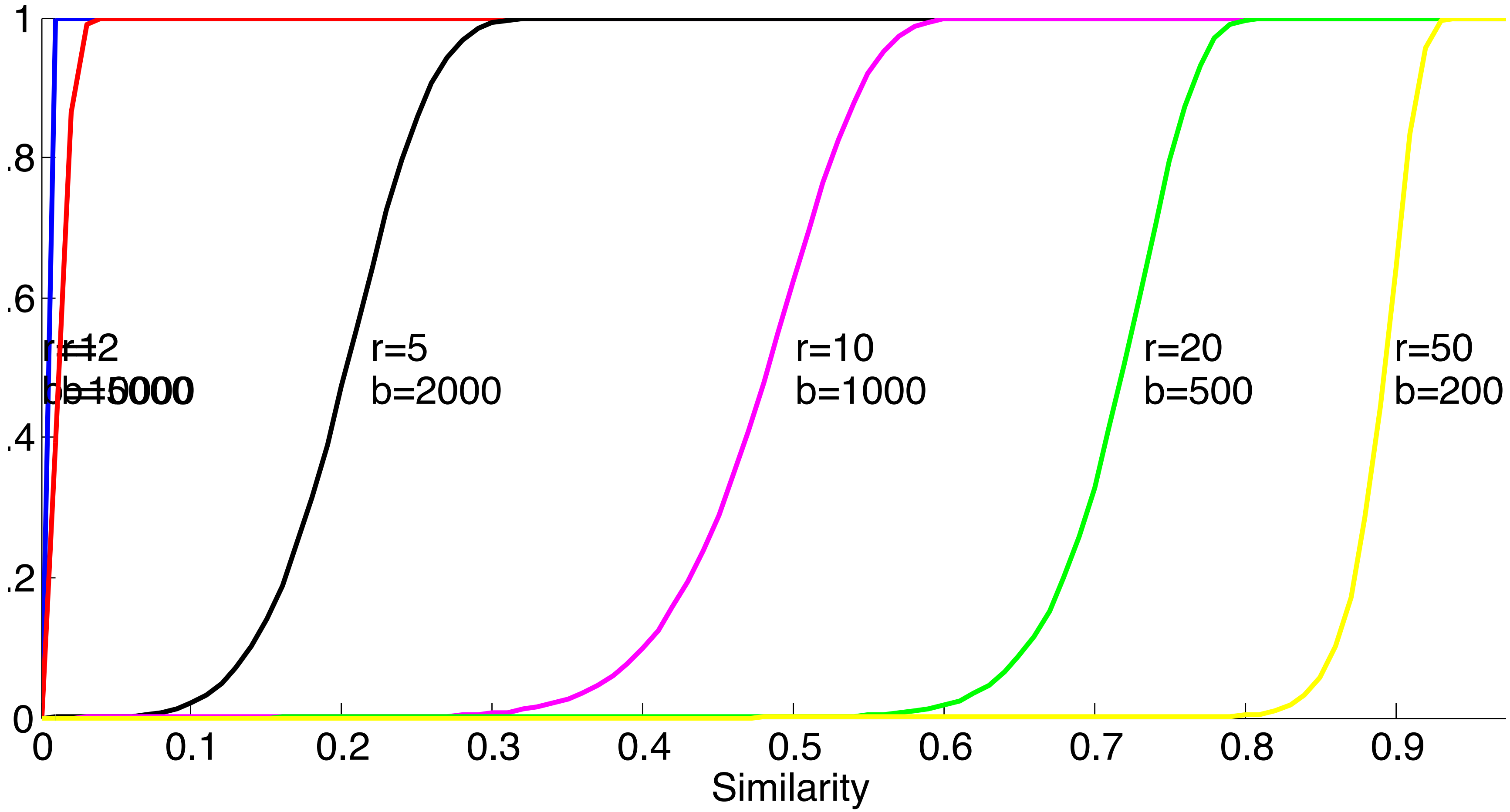
100 hash functions



1000 hash functions



10000 hash functions



Implementation details

- Tune *r* and *b* to achieve desired similarity threshold
- Typically favor
  - few false negatives
  - more false positives
- Do pairwise comparisons of all resulting candidate pairs (in main memory), to eliminate false positives
- Typically also compare the actual documents (needs another pass through the data)

Acknowledgments

- Several slides adapted from the material accompanying the textbook (Anand Rajaraman, Stanford)