15-853:Algorithms in the Real World

Nearest Neighbors in **High Dimensions**

- Curse of dimensionality
- Representing Documents and Products as Sets, Set similarity
- Minhash for compact set signatures
- Locality sensitive hashing

"BigData"

Curse of Dimensionality

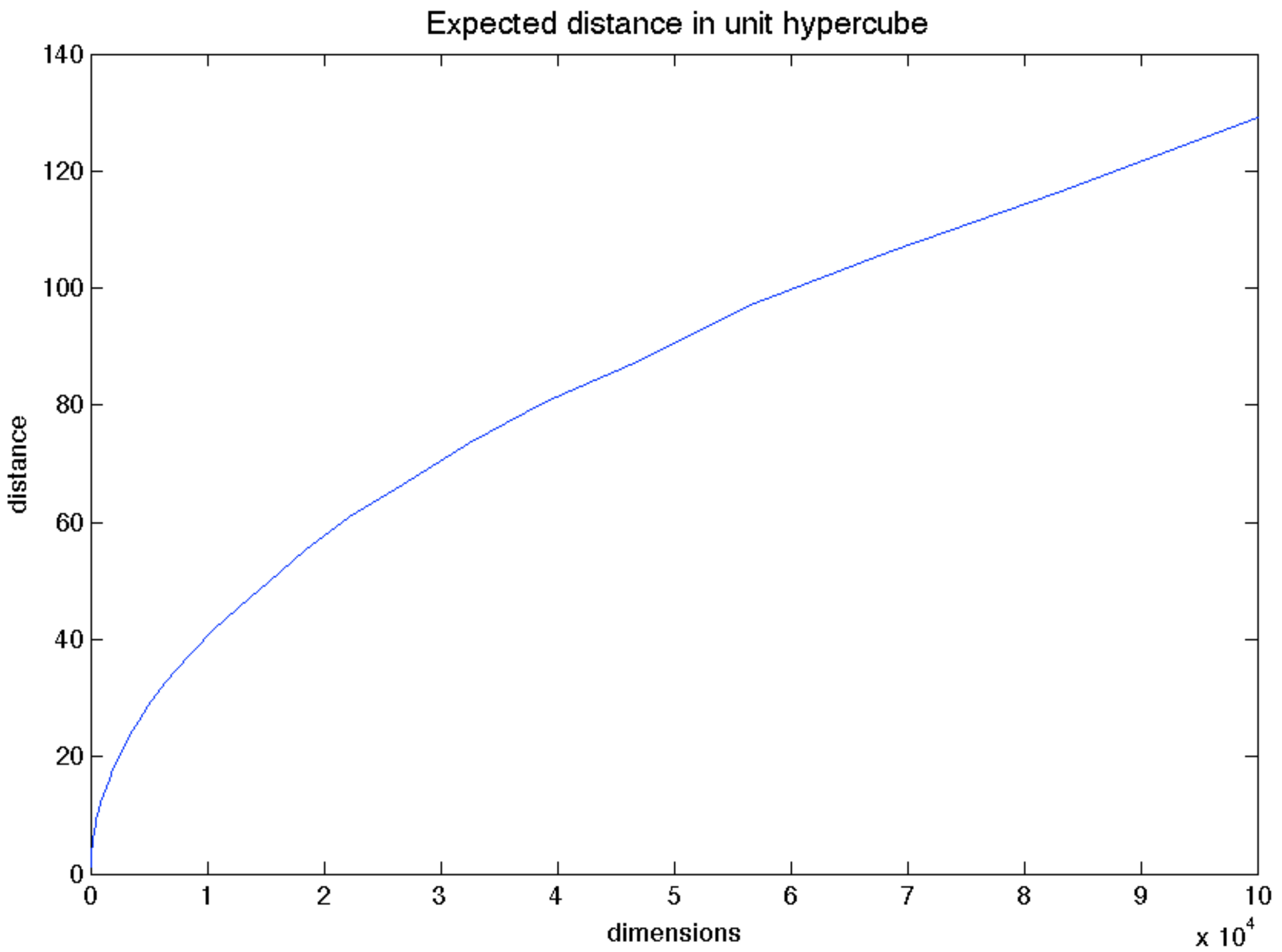
Previously we have learned about spatial decomposition methods such as kd-trees.

Why do these fail with very high dimensions (d >> dozen)?

- What if the dimension is in the thousands, or millions?

Curse of Dimensionality

High-degree spaces are lonely places.



Curse of Dimensionality

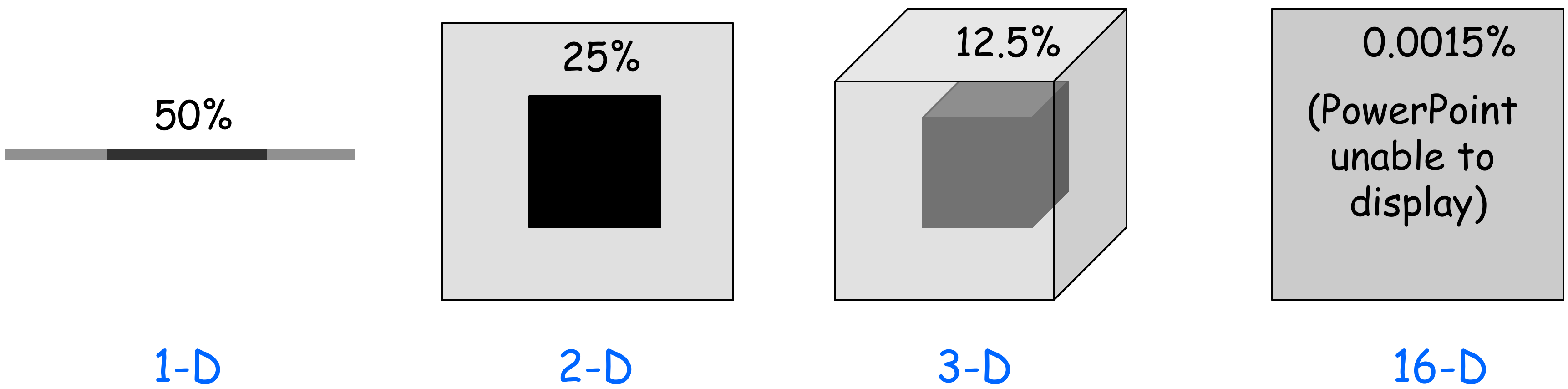
In very high dimensions, the notion of "nearest" may not make much sense anymore.

$$\lim_{dim \to \infty} \frac{\mathrm{dist}_{max} - \mathrm{dist}_{min}}{\mathrm{dist}_{min}} \to 0.$$

Curse of Dimensionality

Rule of thumb: to use KD-trees, the number of points N must be >> 2^d

Rectangular range 0.5 queries in a unit (hyper)cube:



Curse of Dimensionality

To find at least one point in a 0.5-hypercube range, assuming uniform distribution:

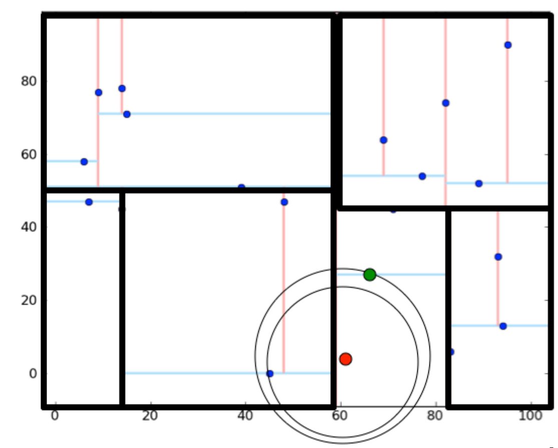
Dimensions	Minimum N
1	2
2	4
3	8
16	65563
10000	galactic
d	2^d

This lecture discusses data with even hundreds of thousands of dimensions.

Curse of Dimensionality

Consider a nearest neighbor search with query point at the origin.

To find any point, need to expand the search range very large fraction of the axis length → Most nodes of the Kd-tree must be considered → No benefit of using Kd-tree.



WORKING WITH HIGH DIMENSIONAL DATA

Challenges

1. Presenting high dimensional objects compactly, so that they can be stored in the RAM and quickly compared for similarity.
   – Today: Min-hash signatures for sets

2. Finding similar items from a collection of high dimensional objects.
   – Today: Locality sensitive hashing based on min-hash

Material based largely on "Mining of Massive Datasets" book by Rajaraman and Ullman (available free for download!)

High Dimensional Data

Examples of high dimensional data:

Representing **documents** as vectors (or sets)

- "bag of words" (TF-IDF weighting)
- shingles (k-substrings)

```
"The course will cover both the theory behind the algorithms and case
studies…"
```

➔ `{the: 3, course: 1, will: 1, …}`

➔ `[0,0, …., 1 …, 3,0,0,… 1,0,.. ]`

➔ For representing **sets**, only **binary values**

Extremely sparse, so we use *sparse vectors*

→ `[(118,1), (107872,1), (200938, 1) ….]`

Note: In practice stop-words like "the" would be removed.

High Dimensional Data (cont.)

Collaborative Filtering

- representing movie as a vector of ratings by users
- representing product by binary vector x: x(j) = 1 if user j bought the item, 0 otherwise

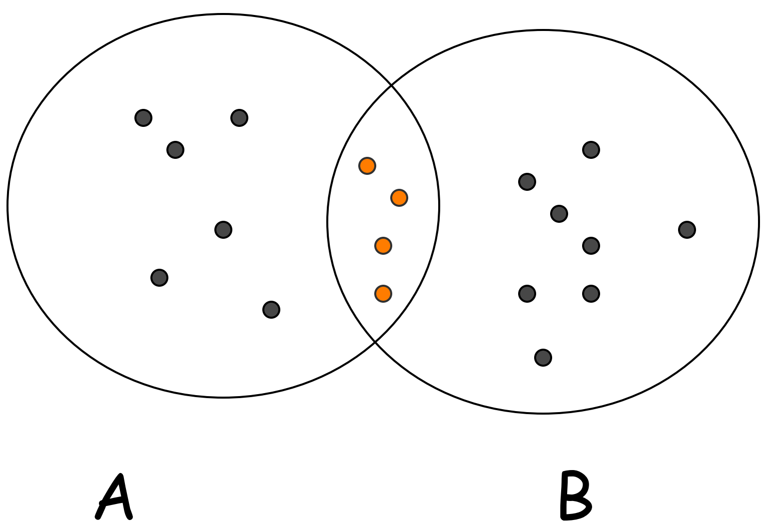
Applications of finding Similar (Nearest) Items

- Filter duplicate docs in search engine
- Plagiarism, mirror pages
- Recommend similar products, movies

Defining Similarity

Similarity metric, “distance”, for **sets**

Jaccard similarity: $$\mathrm{SIM}(A,B) := \frac{A \cap B}{A \cup B}$$



4 common
18 total

SIM(A,B) = 4/18
= 2/9

Similarity-Preserving Signatures

Even sparse, the sets of words, shingles or users/ratings are too big to handle efficiently.

Goal: compute a "signature" for each set, so that similar documents have similar signatures (and dissimilar docs are unlikely to have similar signatures). (Note: "hashes" are one type of signature)

Trade-off: length of signature vs. accuracy

Could we use **cryptographic signatures?**

Characteristic Matrix of Sets

Element num	Set1	Set2	Set3	Set4
0	1	0	0	1
1	0	0	1	0
2	0	1	0	1
3	1	0	1	1
4	0	0	1	0
…				

Stored as a sparse matrix in practice.

Example from "Mining of Massive Datasets" book by Rajaraman and Ullman

Minhashing

Minhash(π) of a set is the number of the row (element) with first non-zero in the **permuted order π**.

Element num	Set1	Set2	Set3	Set4
1	0	0	1	0
4	0	0	1	0
0	1	0	0	1
3	1	0	1	1
2	0	1	0	1
…				

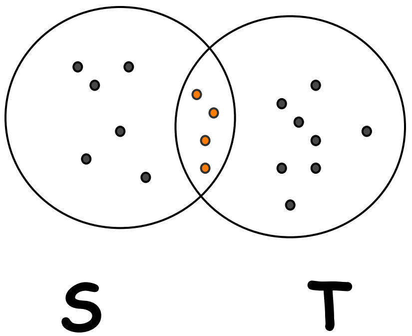
$\Pi=(1,4,0,3,2)$

Example from "Mining of Massive Datasets" book by Rajaraman and Ullman

Minhash and Jaccard similarity

Theorem:

$P(\text{minhash}(S) = \text{minhash}(T)) = SIM(S,T)$



Proof:

X = rows with 1 for both S and T

Y = rows with either S or T have 1, but not both

Z = rows with both 0

Probability that row of type X is before type Y in a random permuted order is ________

Minhash signature

Let $h_1, h_2, \ldots, h_n$ be different minhash functions (i.e different permutations).

Then signature for set S is:

$$SIG(S) = [h_1(S),\ h_2(S),\ \ldots,\ h_n(S)]$$

Now how to compute estimate of the Jaccard similarity between S and T using minhash-signatures?

SIM(S,T) ≈ ratio of equal elements of SIG(S) and SIG(T)

Approximating Minhashes

…. But storing huge permutations is also infeasible.

Solution: use a **random hash function** (for row number) to simulate a permutation.

Properties of random hashes?

We assume the # collisions is small vs. number of items.

Algorithm

For each row $r = 0, 1, \ldots, N-1$ of the characteristic matrix:

1. Compute $h_1(r), h_2(r), \ldots, h_n(r)$
2. For each column c:
   1. If column c has 0 in row r, do nothing
   2. Otherwise, for each $i = 1,2, \ldots, n$ set $SIG(i, c)$ to be $\min(h_i(r), SIG(i, c))$

Note: in practice we need to only iterate through the non-zero elements.

Worked example (on blackboard)

Element num	Set1	Set2	Set3	Set4	x + 1 mod 5	3x +1 mod 5
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3
…						

Signature matrix

	Set1	Set2	Set3	Set4
H1	∞	∞	∞	∞
H2	∞	∞	∞	∞

LOCALITY SENSITIVE HASHING USING MINHASH

Nearest Neighbors

Assume that we construct a 1,000 byte minhash signature for each document.

Million documents can now fit into 1 gigabyte of RAM.

But how much does it cost to find the nearest neighbor of a document?
- Brute force: 1/2 N(N-1) comparisons.

→ Need a way to reduce the number of comparisons.

LSH requirements

A hash function will divide input into large number of **buckets**. To find nearest neighbors for a query item q, we want to only compare with items in the bucket hash(q): “candidates”.

If two A and B are similar, we want the probability that hash(A) = hash(B) be high.

- *False positives*: sets that are not similar, but are hashed into same bucket.
- *False negatives*: sets that are similar, but hashed into different buckets.

LSH based on minhash

(do not get confused about the different "hashes")

Idea:

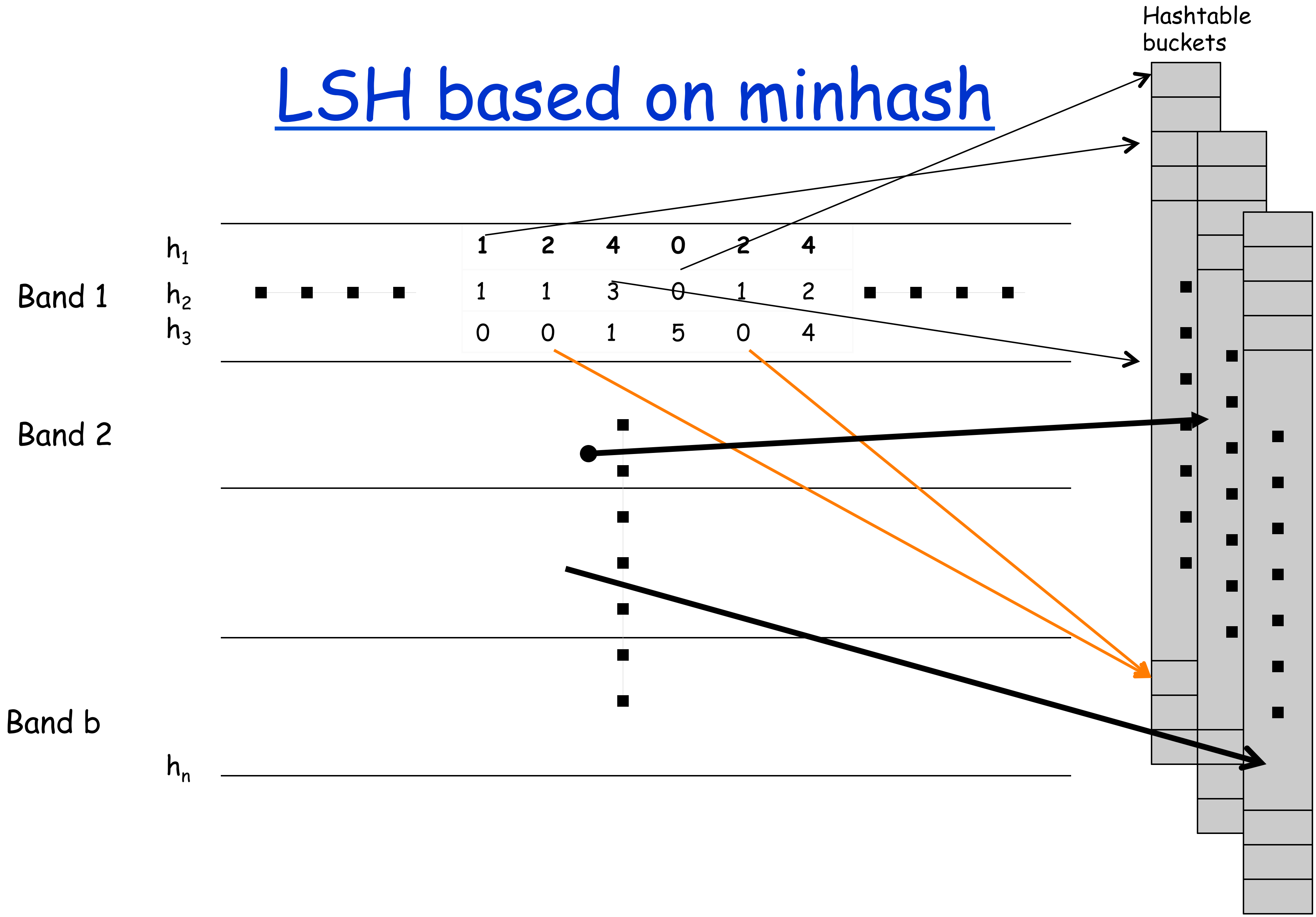
divide the signature matrix rows into **b** bands of **r** rows

hash the columns in each band with a basic hash-function → each band divided to buckets [i.e a hashtable for each band]

If sets S and T have same values in a band, they will be hashed into the same bucket in that band.

For nearest-neighbor, the candidates are the items in the same bucket as query item, in each band.

LSH based on minhash



Analysis

Consider the probability that we find T with query document Q

Let

$s = SIM(Q,T) = P\{ h_i(Q) = h_i(T) \}$

b = # of bands

r = # rows in one band

What is the probability that rows of signature matrix agree for columns Q and T in one band?

Analysis

s = SIM(Q,T)
b = # of bands
r = # rows in one band

Probability that Q and T agree on all rows in a band

s^r

Probability that disagree on at least one row

$1 - s^r$

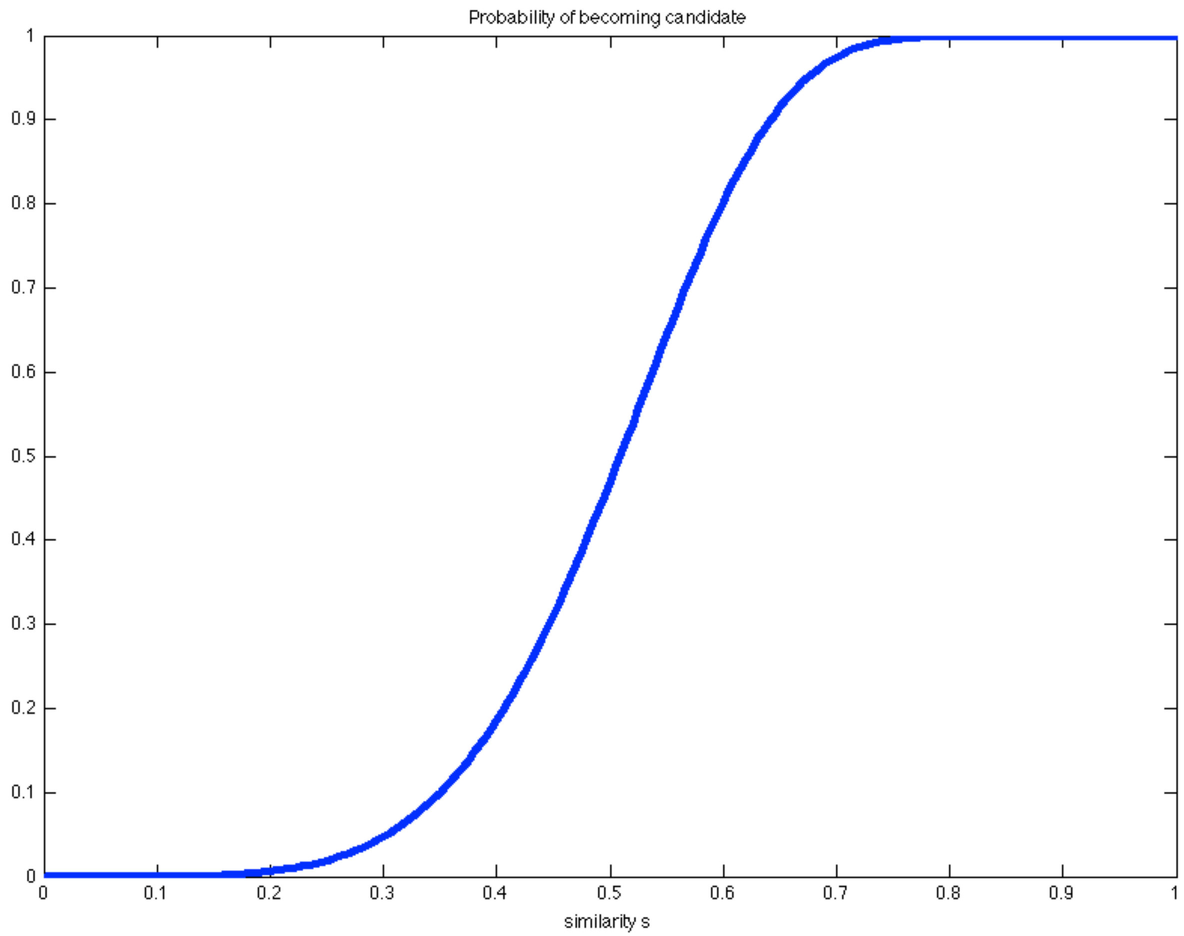
Probability that signatures do not agree on any of the bands:

$(1 - s^r)^b$

Probability that T will be chosen as candidate: _____

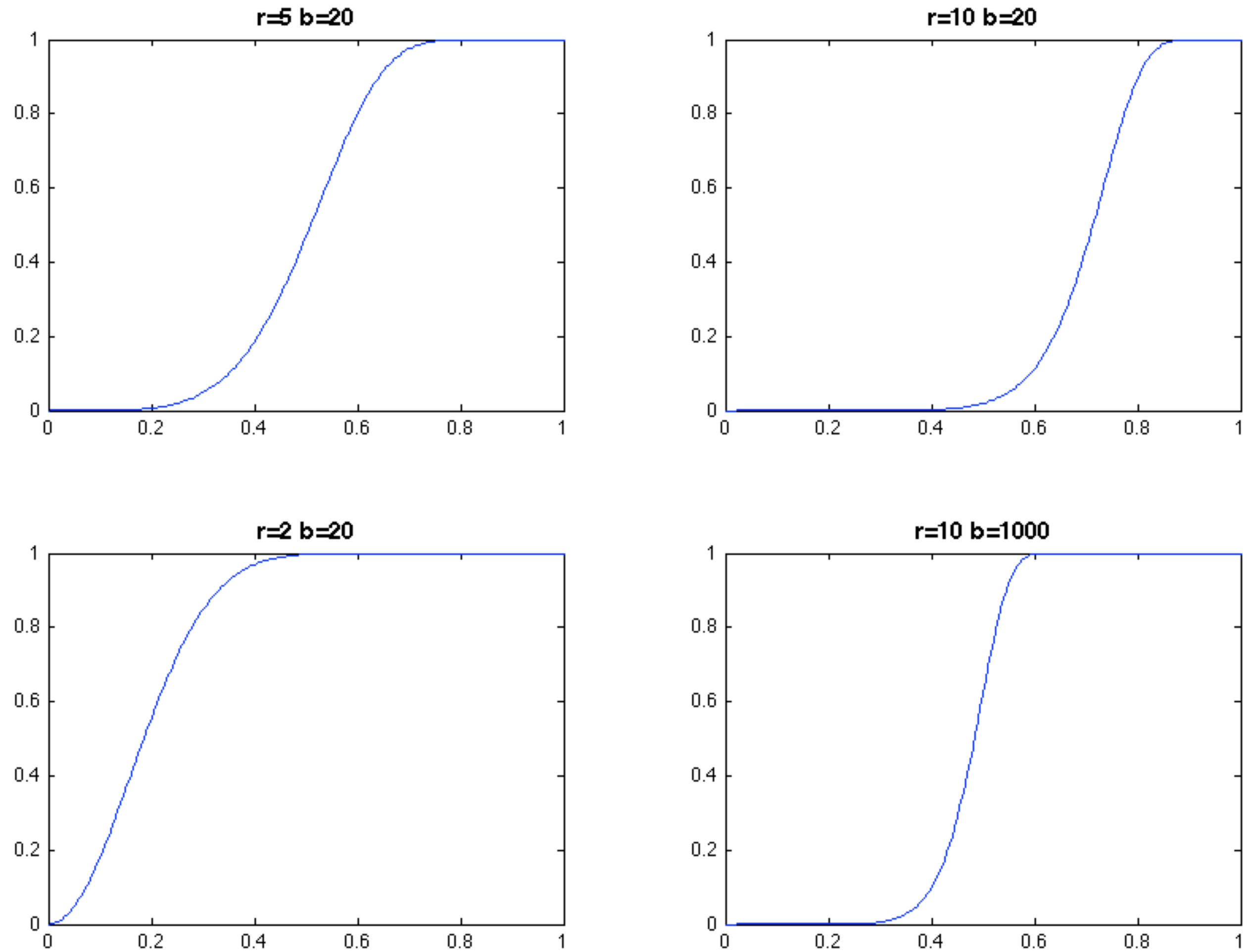
S-curve

$r = 5$
$b = 20$



S-curves

r and b are parameters of the system: trade-offs?



Summary

To build a system that quickly finds similar documents from a corpus:

1. Decide a vector presentation of your documents (bag of words, shingles, etc…)
2. Generate minhash signature matrix for the corpus.
3. Divide signature matrix into bands
4. Store each band-column into a hashtable
5. To find similar documents, compare to candidate documents for each band only in the same bucket (using minhash signatures or the docs themselves) .

More About Locality Sensitive Hashing

Active research area.

Different distance metrics and compatible locality sensitive hash functions:

- Euclidean distance → random projections
- Cosine distance
- Edit distance (strings)
- Hamming distance

Rajaraman, Ullman: Mining of Massive Datasets (available for download)