CSE2525 · DATA MINING · 2026–27

MinHashing & Locality-Sensitive Hashing

Finding similar items among millions — without comparing every pair.

A. Anand · study deck · after *Mining of Massive Datasets*, ch. 3

PART I

Finding similar items at scale

Two walls: sets are huge, and comparing all pairs is quadratic.

RECALL THE NEAR-DUPLICATE SYSTEM

Fast filter → slow check, made real

WHERE WE LEFT OFF

For near-duplicates we wanted cheap candidate generation, then expensive exact verification — and to never materialise all n^2 distances.

MinHash + LSH is exactly how you build that fast filter for set / text similarity. This lecture fills in the “how”.

A SIMILARITY FOR SETS

Jaccard similarity

$$J(A,B) = |A \cap B| \,/\, |A \cup B|$$

EXAMPLE

A = {1,2,3,4}, B = {3,4,5} → |A∩B| = **2**, |A∪B| = **5**, so J(A,B) = **0.4**.

Jaccard is 1 for identical sets, 0 for disjoint sets — and it’s what MinHash will secretly preserve.

TURN DOCUMENTS INTO SETS

Shingling: text → a set of k-shingles

K-SHINGLE

Any substring of length **k** that appears in the document. The document becomes the **set** of its shingles.

```
“the cat sat”  →  k=2 words → { “the cat”, “cat sat” }
```

— Similar documents share many shingles → high Jaccard.

— k too small → everything looks similar; k too large → fragile. k≈5–9 chars for text.

WHY WE CAN'T JUST COMPUTE JACCARD

Two walls

Sets are huge

- Each doc = thousands of shingles
- Storing / intersecting them is costly

Pairs are quadratic

- N docs → N(N−1)/2 pairs
- 1M docs → $\mathbf{5 \cdot 10^{11}}$ comparisons

MinHash attacks the first wall (compress the sets); LSH attacks the second (skip most pairs).

QUESTION

A million documents means about $\mathbf{5 \cdot 10^{11}}$ possible pairs. How could you find the **highly similar** pairs without ever comparing most of them?

Two obstacles: the sets are huge, and the pairs are quadratic.

Think — then → for the answer.

PART II

MinHashing

A short signature that secretly preserves Jaccard.

SETS AS COLUMNS OF A 0/1 MATRIX

The characteristic matrix

element	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

Rows = universe elements, columns = sets. We never build this explicitly — it's a way to *think* about MinHash.

ONE NUMBER PER SET PER PERMUTATION

MinHash = first 1 under a shuffle

DEFINITION

Pick a random permutation π of the rows. The MinHash of set S is the **first row (in π-order) where S has a 1**: $h_\pi(S) = \min_{x \in S} \pi(x)$.

In practice we don't shuffle — we apply k independent hash functions to the row indices and keep each column's running minimum. One pass over the data.

WHY MINHASH WORKS AT ALL

$P(h(A) = h(B)) = J(A, B)$

THE ONE EQUATION TO REMEMBER

Under a random permutation, the probability two sets get the same MinHash equals their Jaccard similarity.

Why: classify rows as both-1 (count X), one-1 (count Y), or both-0 (ignored). Scanning from the top, the first non-both-0 row is equally likely to be any of them. It's a "match" iff it's a both-1 row → $P = X / (X + Y) = J(A,B)$.

COMPRESS COLUMNS TO SHORT SIGNATURES

The signature matrix

hash	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0
h_3	1	2	1	1

Each column is a set's signature — just k numbers (here k=3; real systems use ~100). Estimate $J(S_i, S_j)$ = fraction of rows where their signatures agree.

SIGNATURES STAND IN FOR SETS

Longer signature → better estimate

WHAT WE BOUGHT

Each of k rows agrees with probability J, independently → the fraction of agreeing rows is an unbiased estimate of Jaccard, with error shrinking like $1/\sqrt{k}$.

A giant set is now k small integers. The first wall is gone — but we'd still compare all N^2 signature pairs. That's what LSH fixes.

PART III

Locality-Sensitive Hashing

Hash signatures so similar ones collide — and only compare collisions.

SPLIT THE SIGNATURE INTO BANDS

The banding trick

band	S_1	S_2	S_3	S_4
band 1	a7	c2	a7	a7
band 2	x1	x1	q9	k3
band 3	m4	m4	p8	m4

Split the length-n signature into b bands of r rows (n = b·r). Hash each band to a bucket. Same band value → same bucket.

COLLIDE ONCE → WORTH CHECKING

Candidate = share a bucket in ≥ 1 band

RULE

Two sets are a **candidate pair** if their signatures are **identical in at least one band**. Above: S_1 & S_3 collide in band 1; S_1 & S_4 collide in bands 1 and 3; S_1 & S_2 collide in bands 2 and 3.

Only candidate pairs get the expensive exact check. Most of the N^2 pairs never collide and are never looked at.

TUNE COLLISIONS WITH B AND R

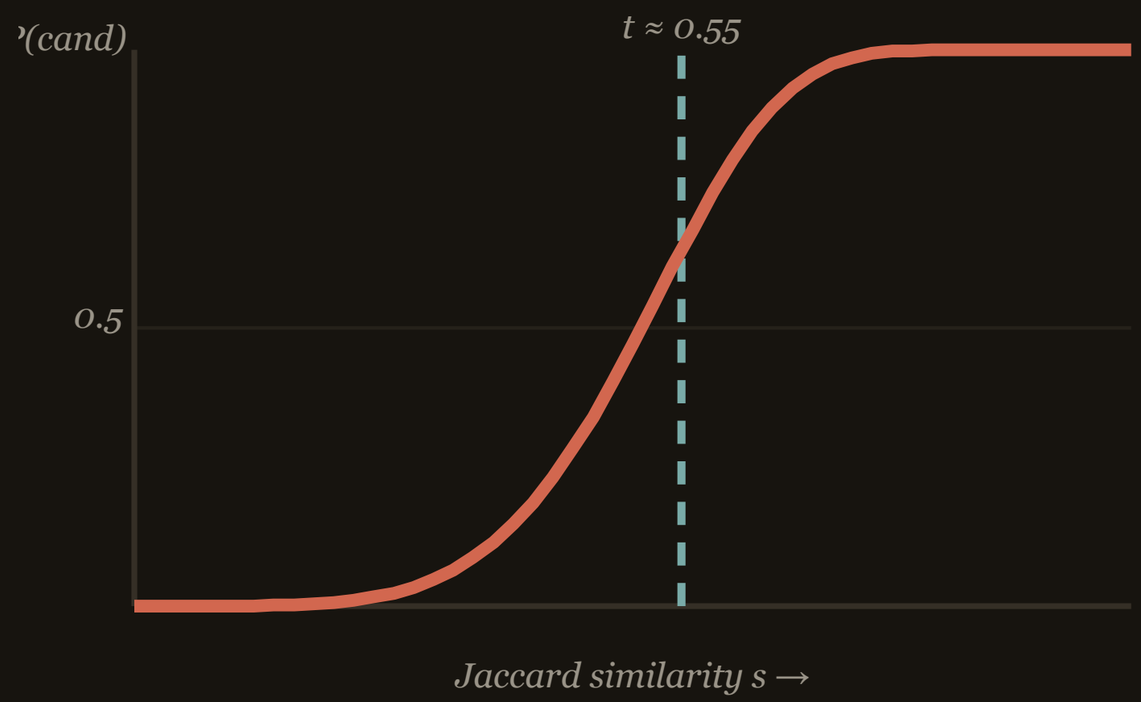
Probability a pair becomes a candidate

$P(candidate) = 1 - (1 - s^r)^b$

- Agree on all r rows of a band: s^r.
- Miss every band: $(1 - s^r)^b$.
- Collide in at least one band: $1 - (1 - s^r)^b$ — an S-curve in s.

B = 20 BANDS · R = 5 ROWS · SIGNATURE = 100

The S-curve and its threshold



The jump sits near $t \approx (1/b)^{1/r} \approx 0.55$. Pairs above *t* almost always become candidates; pairs below almost never do. Choosing b and r chooses the threshold.

STEEPNESS VS. PLACEMENT

Trading false positives against false negatives

You want…	Do this	Cost
Threshold lower	fewer rows r / more bands b	more candidates → more false positives
Threshold higher	more rows r / fewer bands b	risk missing pairs → false negatives
Sharper cutoff	longer signature (bigger b·r)	more hashing work

False positives are cheap — the exact check removes them. False negatives are lost forever, so err toward a slightly lower threshold.

PART IV

The pipeline & where it’s used

Shingle → MinHash → LSH → verify.

END TO END

The whole system on one line

Shingledocs → sets
→
MinHashsets → signatures
→
LSH bandssignatures → candidates
→
Verifyexact Jaccard

THIS IS THE RECIPE FROM THE DISTANCES LECTURE

Cheap candidate generation (LSH) → expensive exact check (Jaccard on the survivors). No N^2 anywhere.

THE SAME FILTER, EVERYWHERE

Where MinHash + LSH show up

— Near-duplicate web pages — deduping a crawl (the original use case).

— Plagiarism & near-duplicate detection — documents, code, answers.

— Entity resolution — matching records that refer to the same thing.

— Recommendation / clustering — fast approximate neighbours at scale.

LSH generalises: different similarities (cosine, Euclidean) have their own locality-sensitive hash families.

TAKE AWAY · AND A TRANSFER EXERCISE

Summary

THE MESSAGE

MinHash shrinks huge sets to short signatures with P(equal) = Jaccard. LSH bands those signatures so only similar pairs collide, giving an S-curve threshold you tune with b and r.

Together they turn an N^2 similarity search into a fast-filter → slow-check pipeline that scales to millions.

Transfer: you must dedup 10M product descriptions at threshold ≈ 0.8. Pick a signature length and a (b, r) split — and say which error you'd rather make.