Ques How to determine 'K'

D_n → D_{train} (70%)

D_n → D_{Test} (30%)

One idea

	Train	acc on D_{test} → $\frac{\text{no of correctly pts}}{\text{Total \# pts}}$
K=1	D_{train}	0.78
K=2	"	0.82
K=3	"	0.85
⋮		⋮

$$D_{test} = \left\{ (x_i, y_i) \mid x_i \in R^d,\ y_i \in \{0,1\} \right\}_{i=1}^{n_{test}}$$

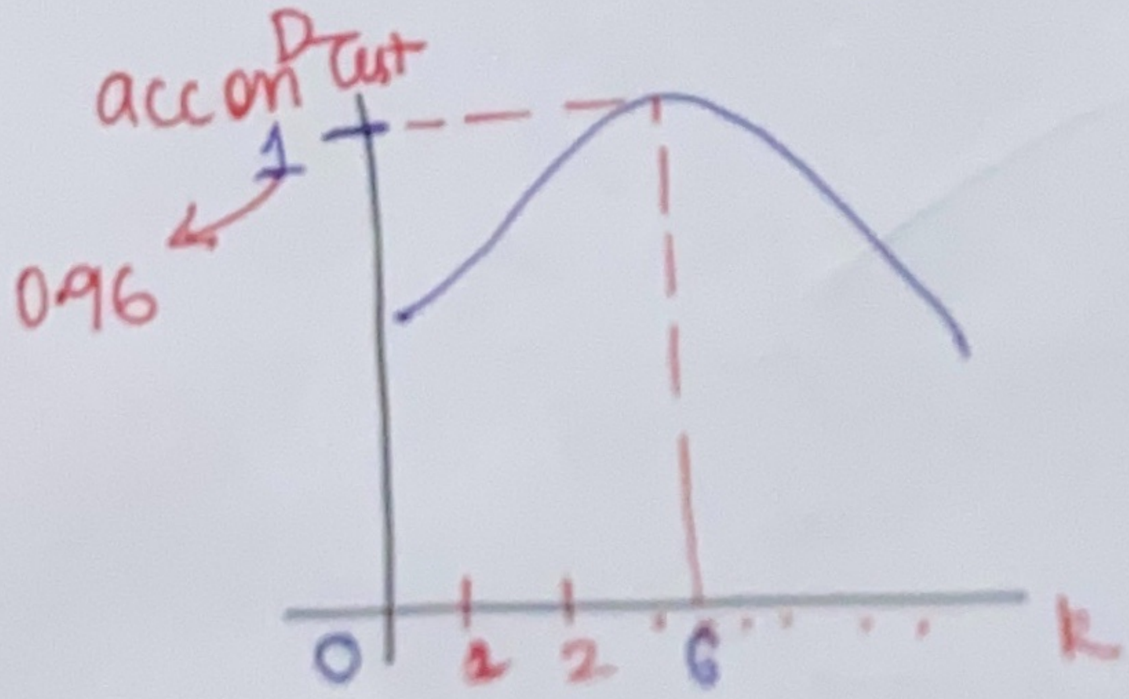
$x_q \rightarrow y_q$

$x_q = x_2$

$x_q = x_3$

⋮

$x_q = x_{n\,test}$



(Typically)
K = 6 gives me best accuracy on D_{test} when using D_{train} as training data

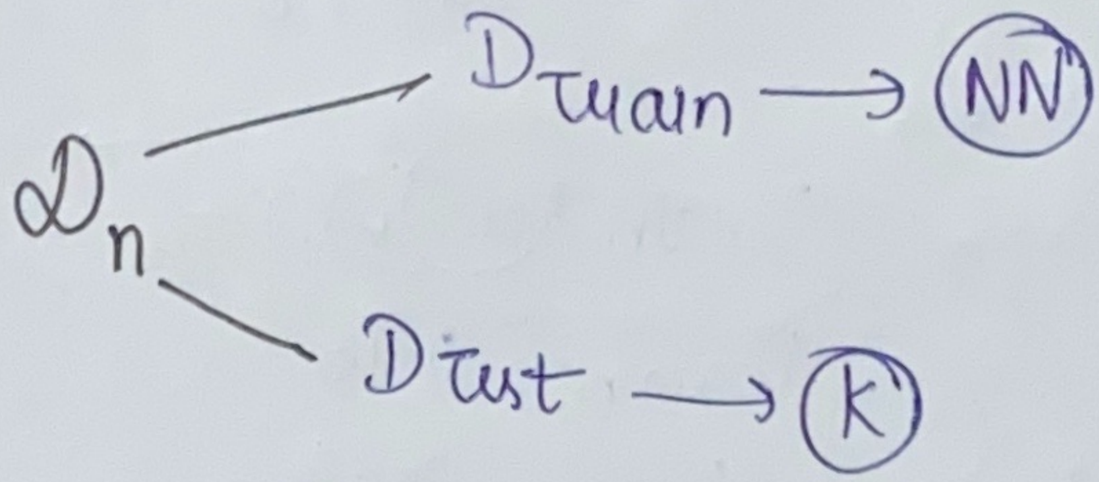
Using D_{train} & 6-NN on Amazon food reviews dataset I got an accuracy of 96%.

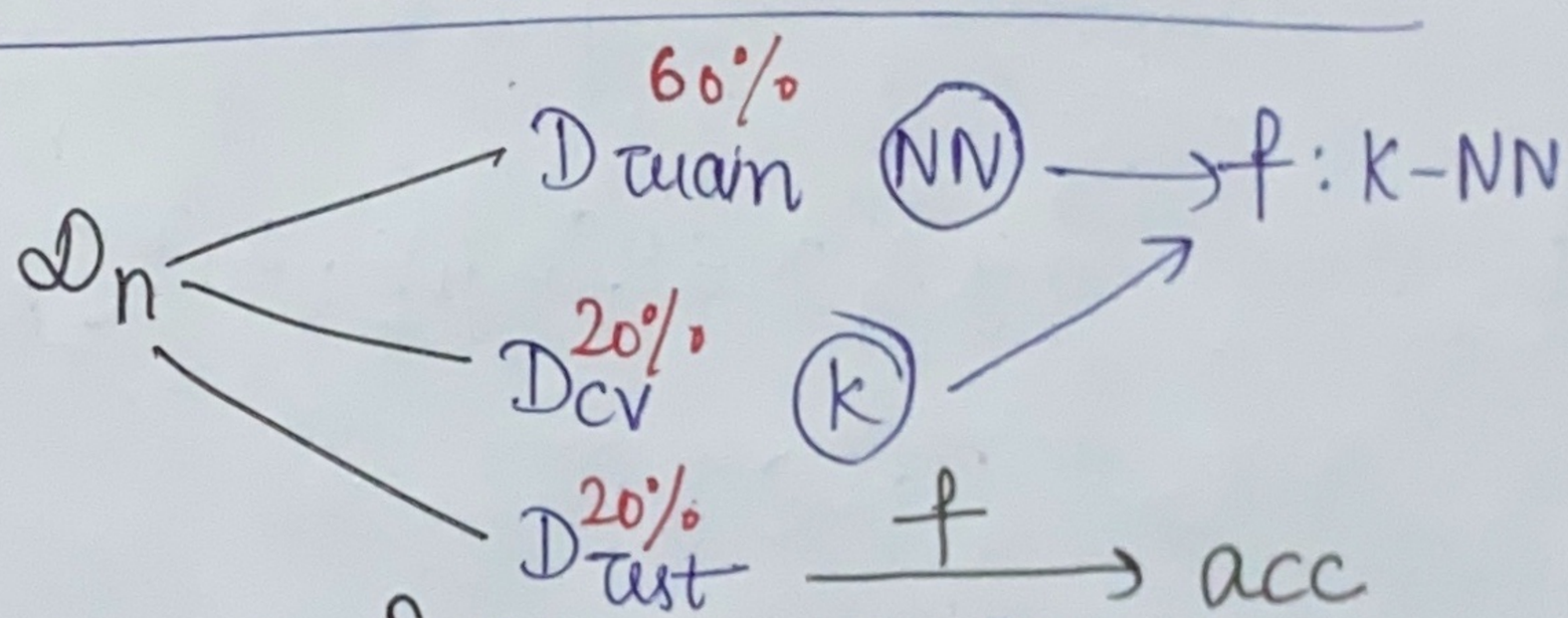
~~Obj~~ perform well on "unseen" pts

↓

not in Train & Test

Cross Validation



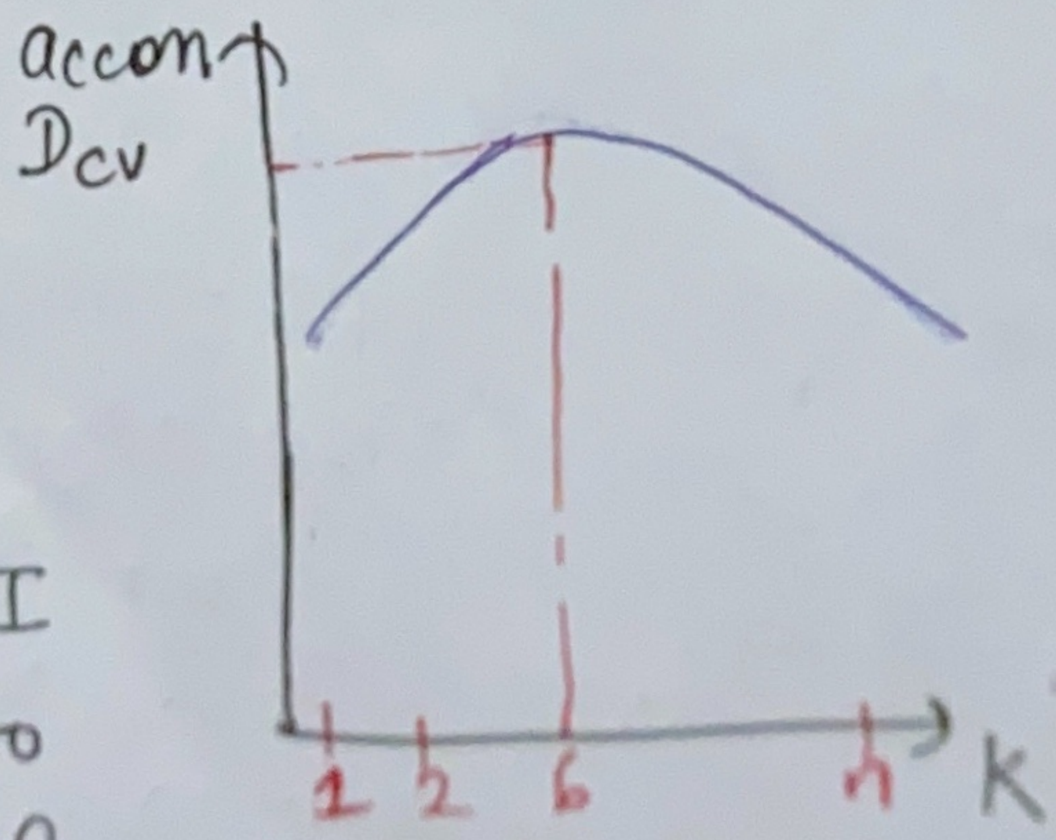


[can I now say 6-NN has acc of 93% on unseen data?]

(unseen data)

→ x_i in D_{test} as x_q

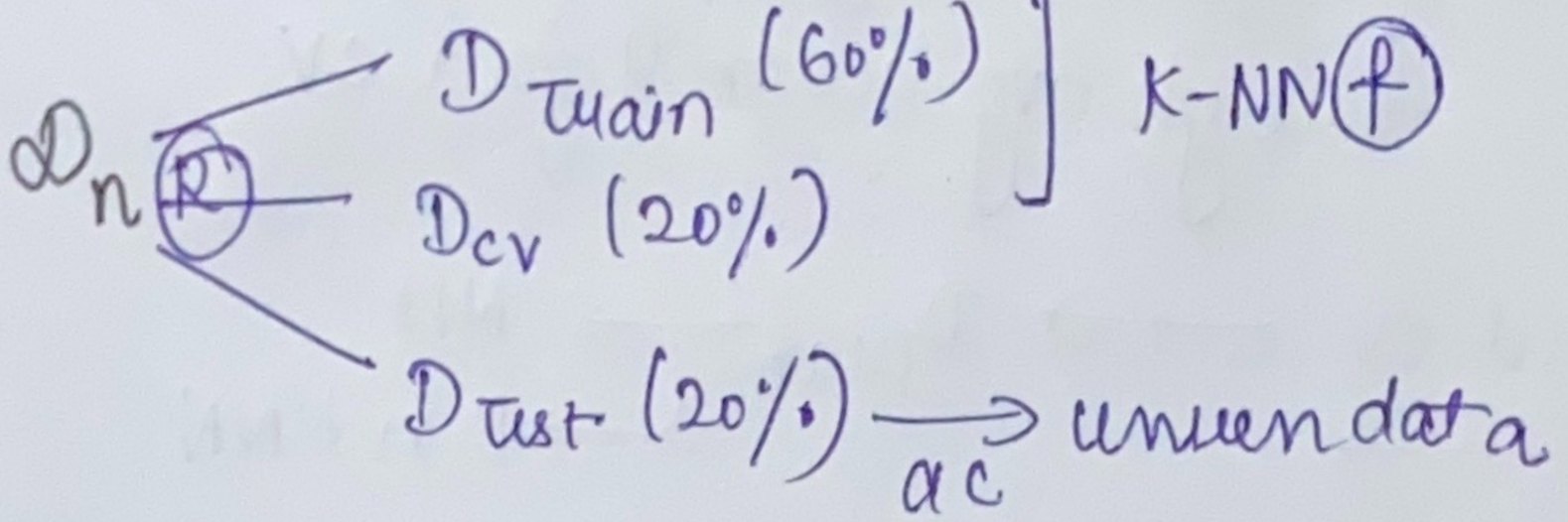
→ acc 93%



using D_{train} as training data I found 6-NN to have an acc of 93% on unseen data

→ generalisation accuracy 7% = generalisation error

K'- fold cross Validation

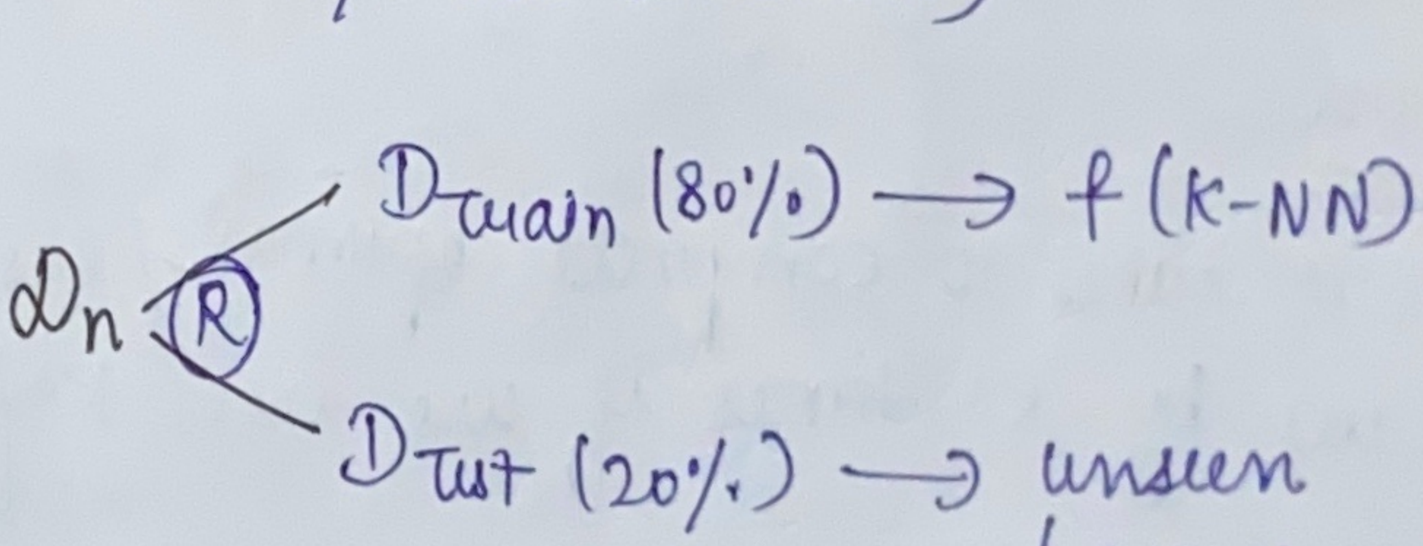


Problem

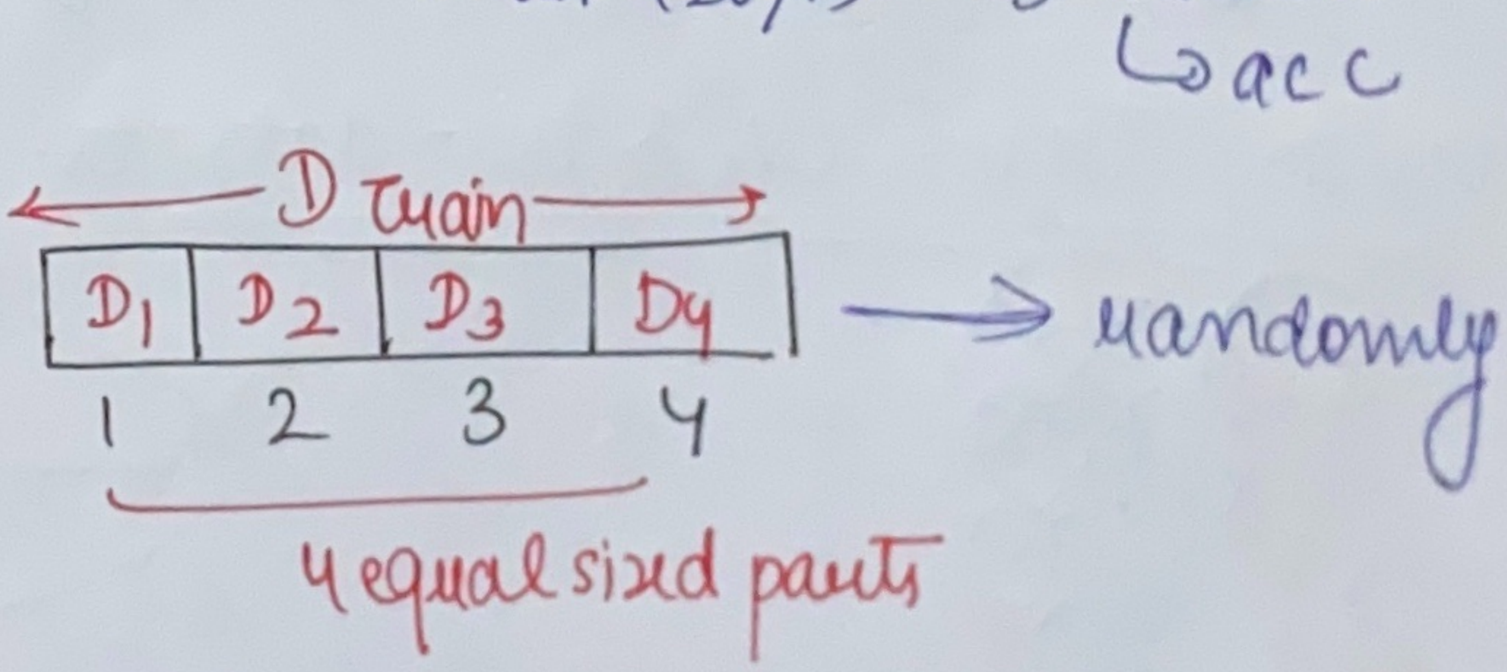
only using 60% of data to compute NN] 80% to compute NN
20% → CV
20% Test (unseen)

Solution

①



②



③

		Train	CV	acc on CV
4 fold cross validation (4 times)	K=1	$D_1 D_2 D_3$	D_4	a_4
	K=1	$D_1 D_2 D_4$	D_3	a_3
	K=1	$D_1 D_3 D_4$	D_2	a_2
	K=1	$D_2 D_3 D_4$	D_1	a_1
again repeat	K=2	$D_1 D_2 D_3$	D_4	a_4
	K=2	⋮	⋮	⋮
	K=2			

avg (a_1, a_2, a_3, a_4) = a

acc on CV dataset

f: 3-NN ⟶ D_{train}

↓

through avg acc on 4 fold CV

k' fold CV ⟶ D_{train} ⟶ NN

↳ k in KNN

↓

what is right k'

k'=4 (k'=10) k'=100

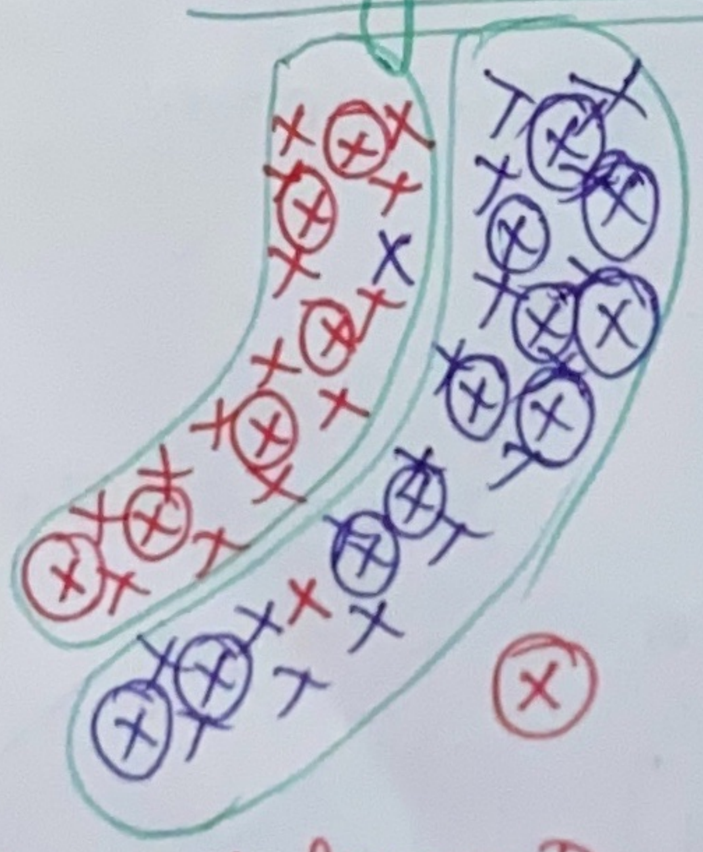
↓

rule of thumb: 10 fold CV

Limitation

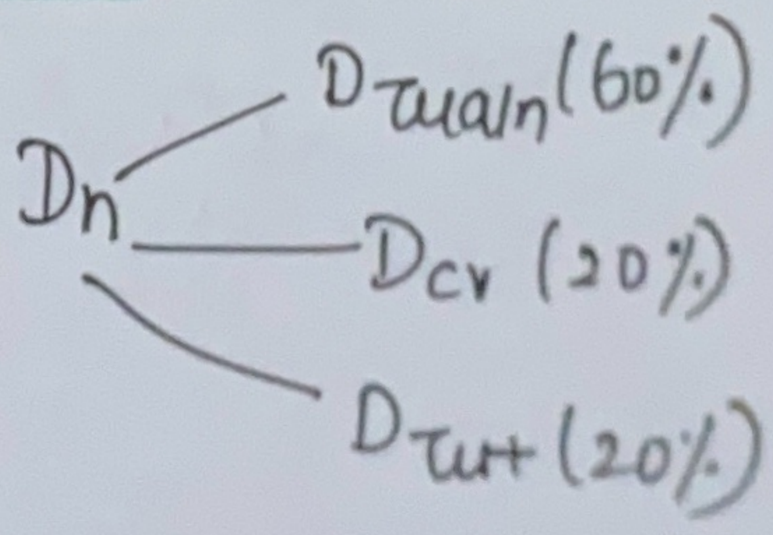
time it takes to compute optimal / last K in K-NN increases by k' times if we use k' fold CV

Visualizing Train, CV & test datasets:



(x): pt from D_{CV}

(x): pt from D_{CV}



x: a datapoint in D_{train}

x: the " " "

① D_{train} & D_{CV} do not overlap perfectly

② If there are many pts from D_{train} in region then it is likely to find many pts from

③ If there are very few tr/te pts in region from D_{train} then it is unlikely to find tr/te from D_{cv} in that region

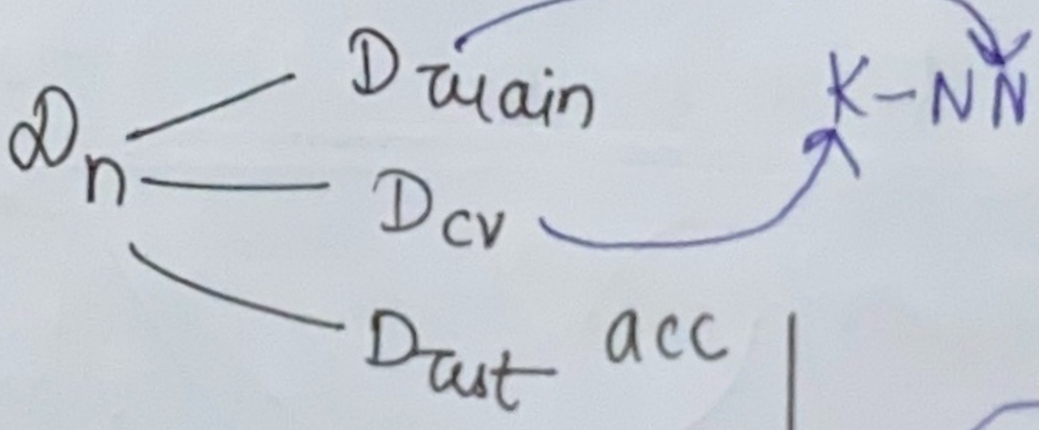
Overfitting vs Underfitting

K-fold cv on D_{cv} → best k → neither overfit nor underfit

Ques How we can be sure that we are not underfitting or overfitting → (plot)

$$\max\uparrow \quad acc = \frac{\#\text{ correctly class pts}}{\text{total}\#\text{ pts}} = 0.93$$

$$\min\downarrow \quad error = 1 - acc = 0.07\% = 7\%$$



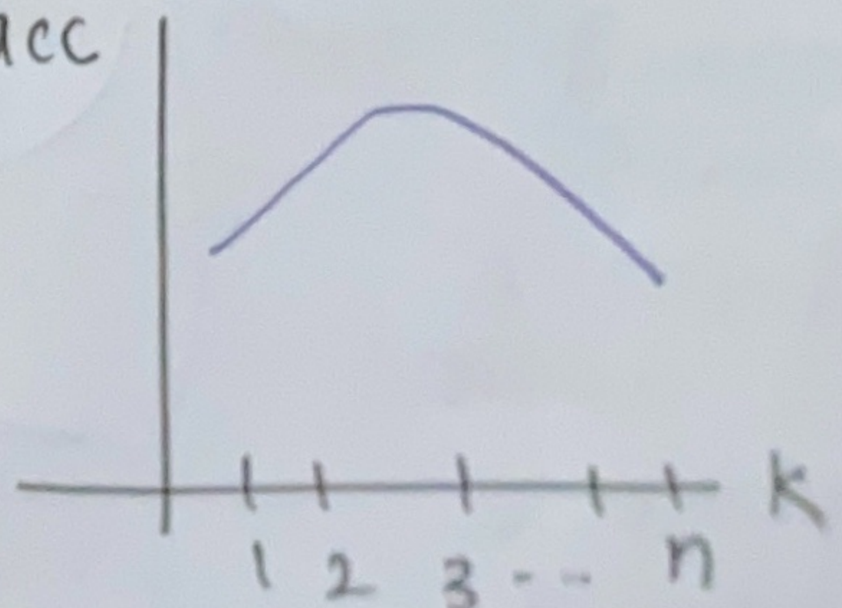
error = 1 − acc

↑ acc on D_{cv}

for each pt in D_{cv}

x_q = pt

$y_q \Rightarrow y_i$



Training error

- for each x_i in D_{train}
  - find 2 nearest neigh to x_i from D_{Train}
  - majority vote to get class label
  - if y_i = class label accurate else error

$D_{train} \rightarrow$ NN

$D_{CV} \rightarrow$ K

f[k-NN]

---

what is train error if 2-NN ?

$D_{train} \rightarrow$ training

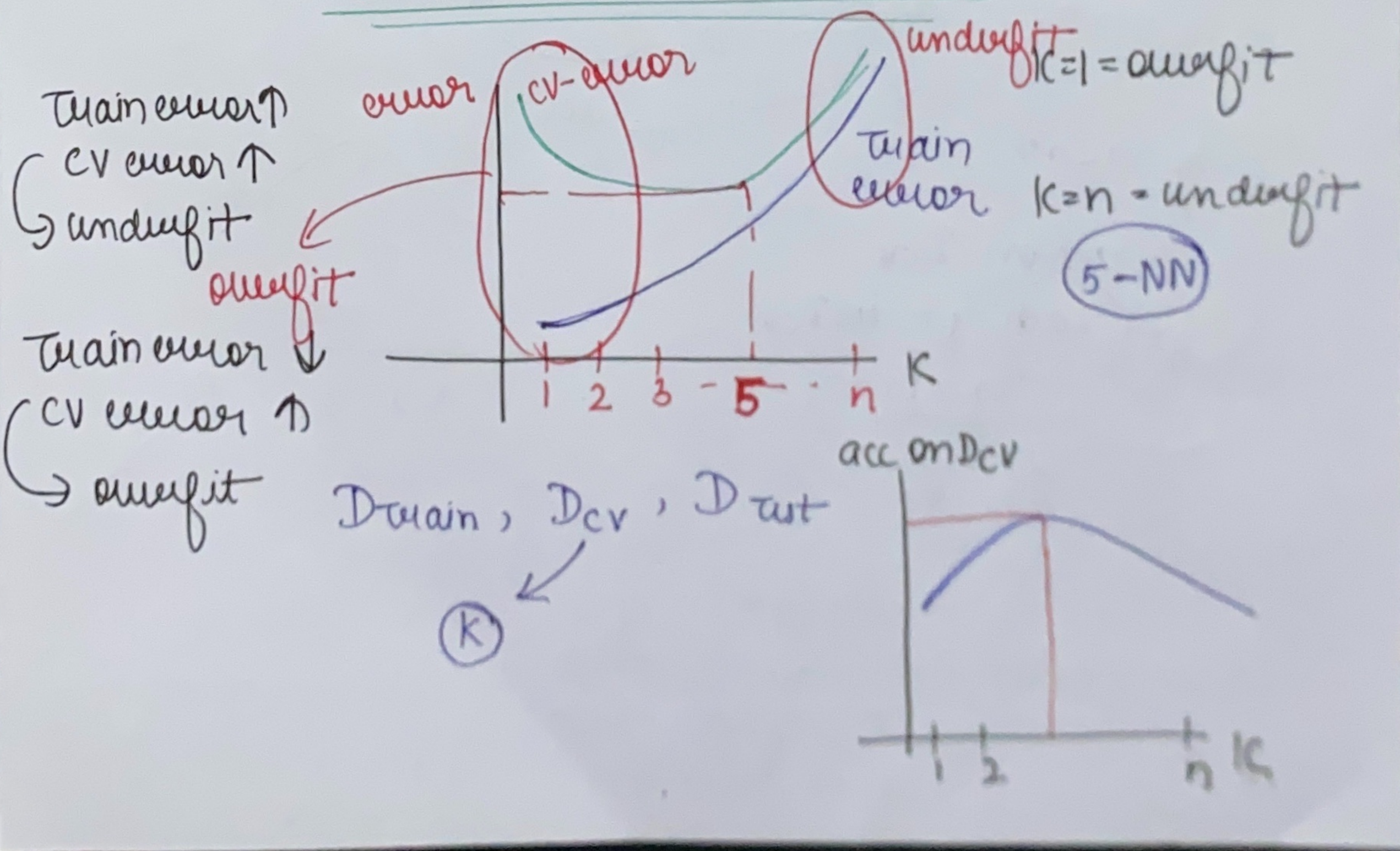
$D_{train} \rightarrow$ acc/error } train error

what is error of 2-NN on D_{CV}

$Train = D_{train}$

D_{CV} : error on D_{CV}

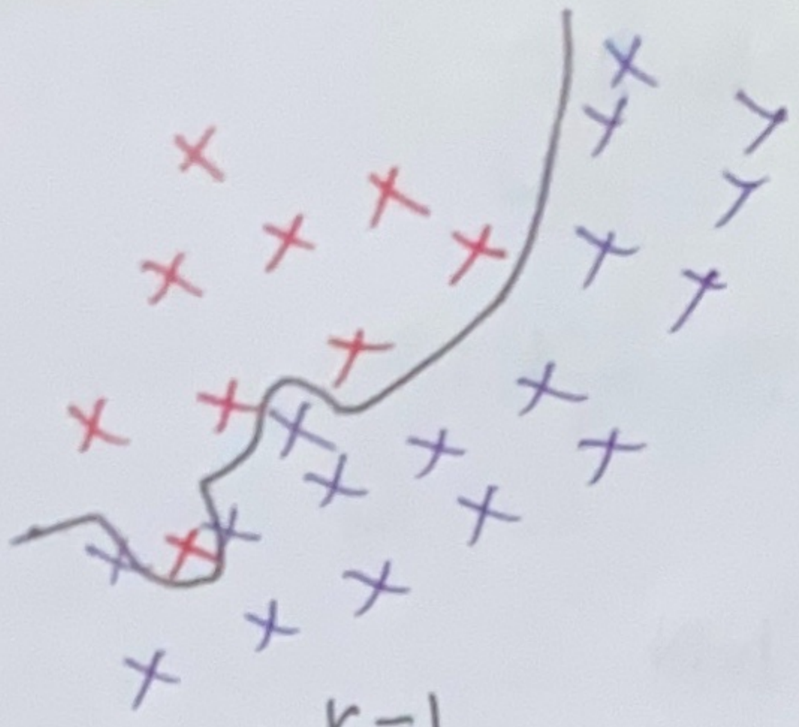
error; train error; validation error



D_{train} , D_{CV} , D_{test}

↙

(K)



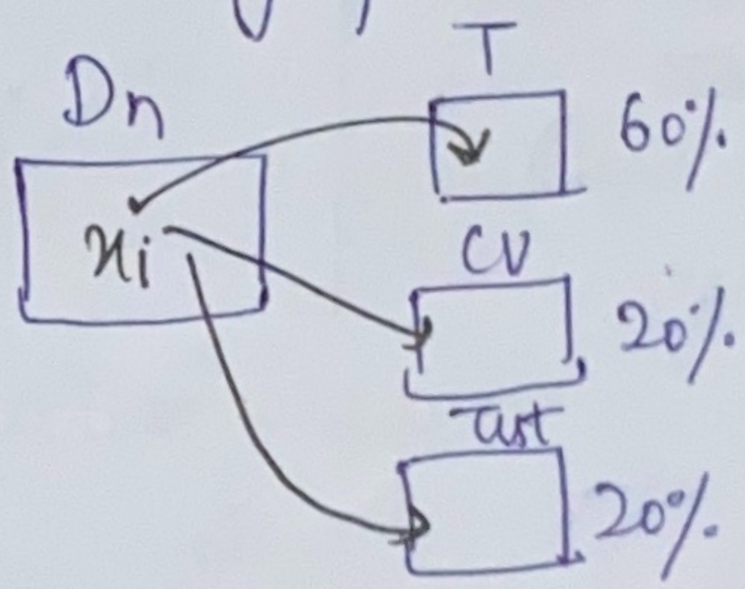
K=1

Train error is low
CV error is high
] k=1

CV error is low
Train error is not zero
] k=5

Amazon Food Review

D_n → D_{train}, D_{cv}, D_{test}] Randomly split

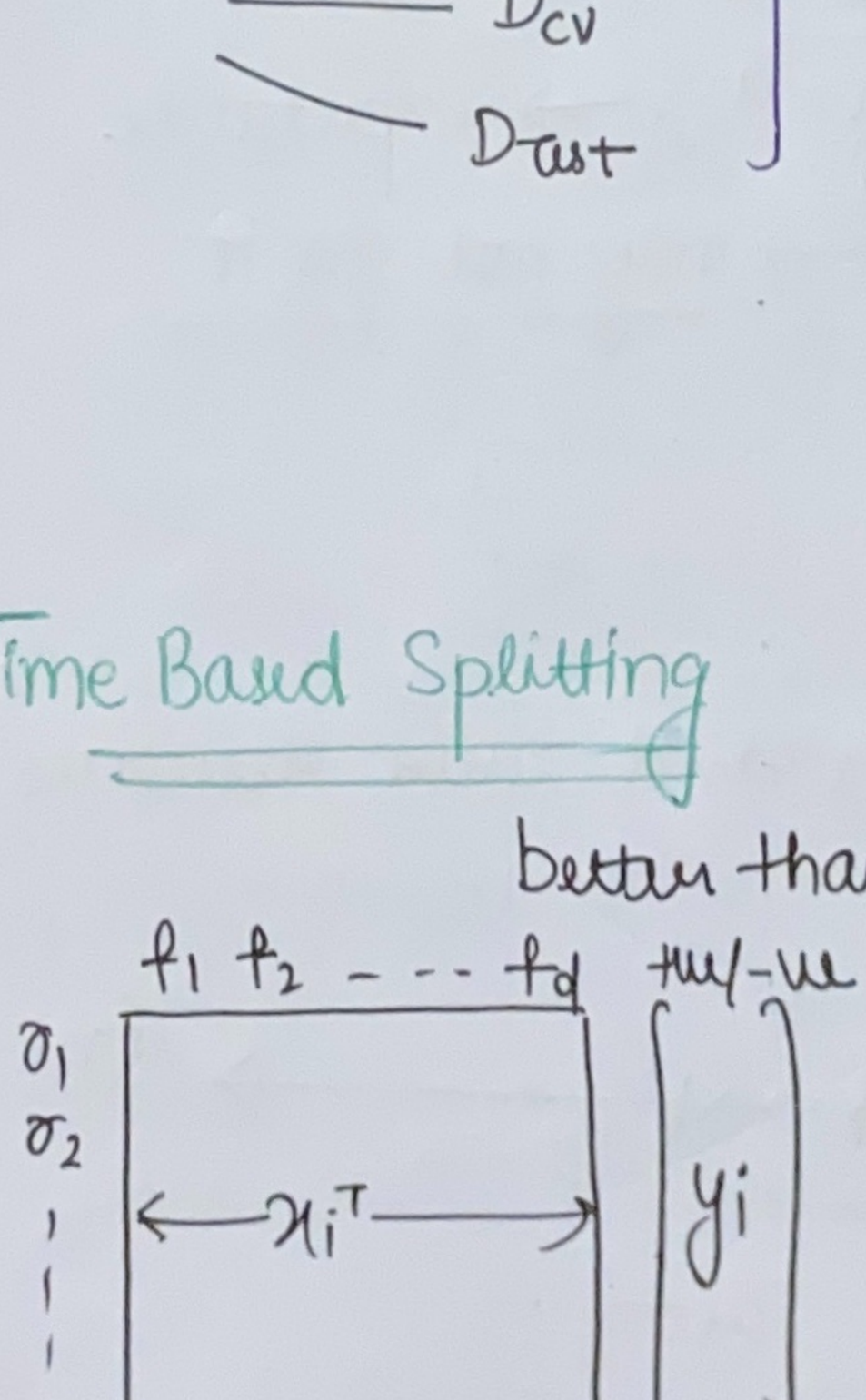


✓ Time Based Splitting

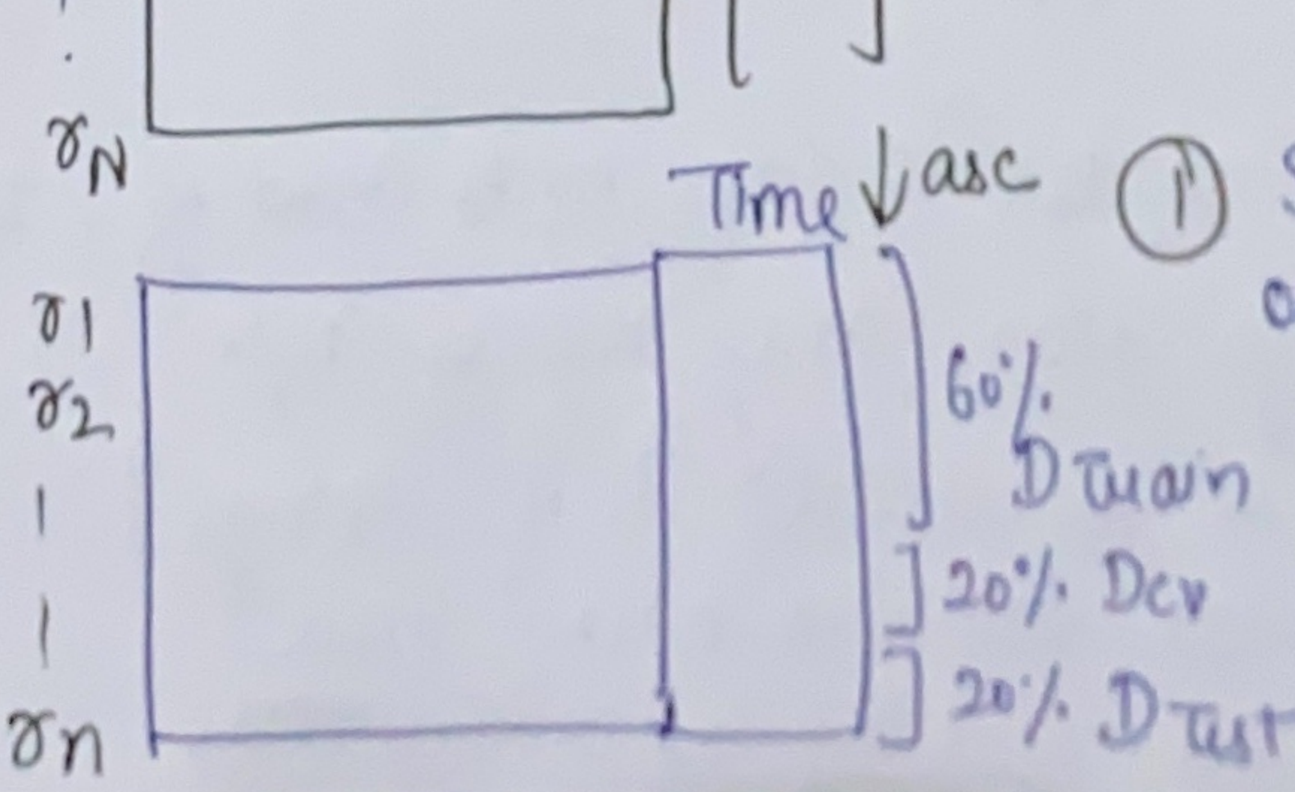
↗ depends on case

better than random-splitting

(D)



(TBS)



① Sort Dn in asc order of Time

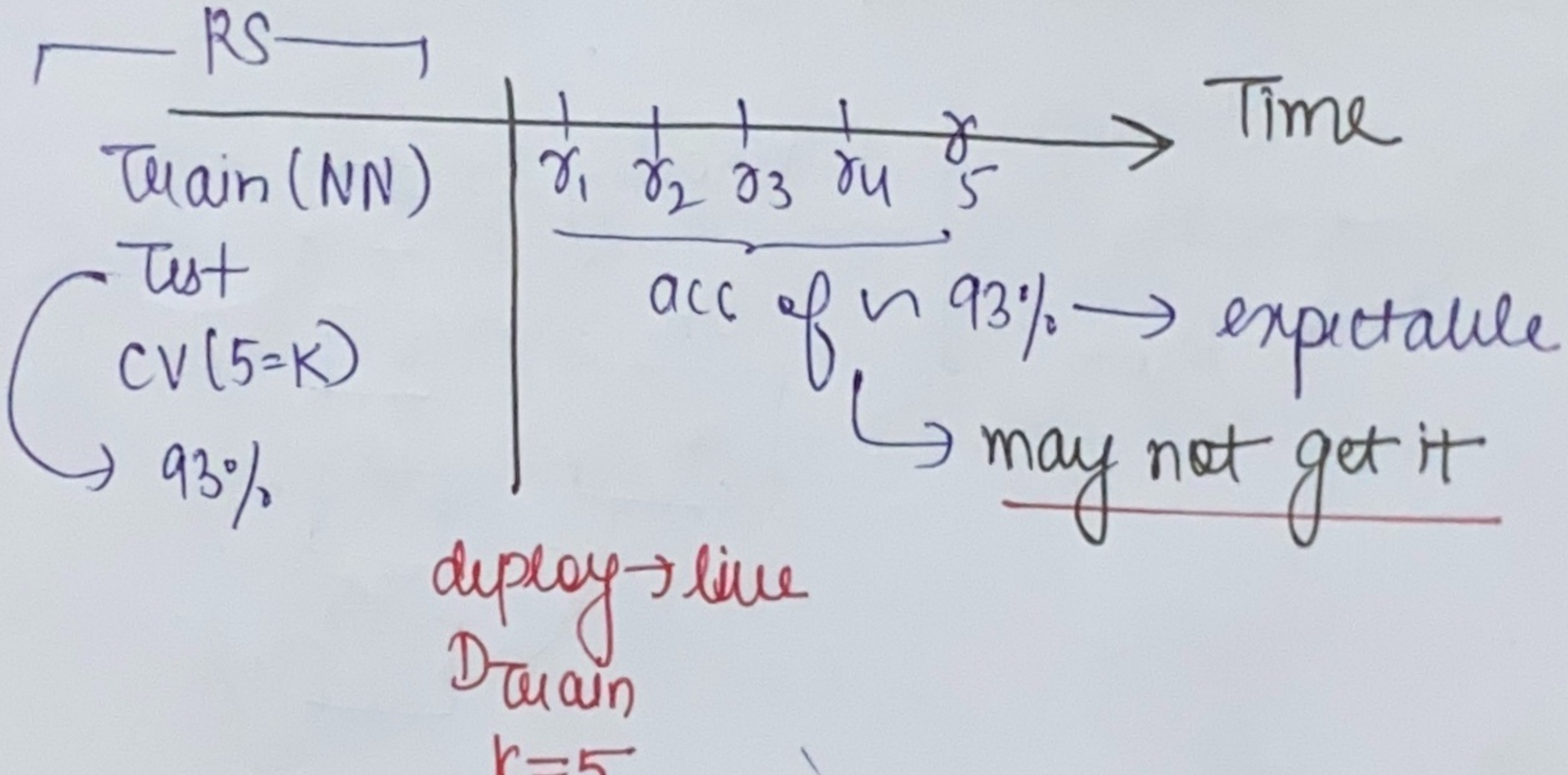
Amazon food review — TBS ⟶ (Time) is necessary

but say I use random splitting

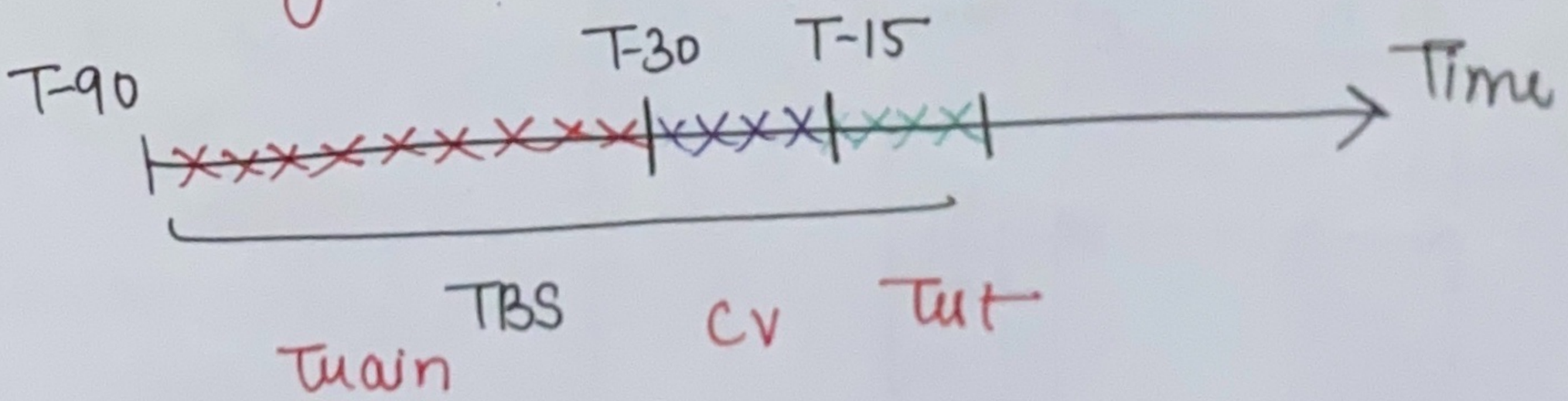
(60%) D_{train} ⟶ NN

(20%) D_{CV} ⟶ k in k-NN

(20%) D_{Test} ⟶ acc (93%)



with time, my products and their reviews change



If I trained my model 15 days back and if I tested it on last 15 days data, my acc is 91%.

→ whenever time is available & if things / behaviour on data changes over time then time based spitting is preferable.

K-NN for regression

classification
2(class)

$$D = \{(x_i, y_i)_{i=1}^{n} \mid x_i \in R^d,\ y_i \in \{0,1\}\}$$

$x_q \rightarrow y_q$

Regression

$$D = \{(x_i, y_i)_{i=1}^{n} \mid x_i \in R^d,\ y_i \in R\}$$

$x_q \rightarrow y_q \rightarrow$ number

① given x_q, find K-nearest neighbour

$(x_1, y_1), (x_2, y_2) \text{ ---- } (x_k, y_k)$

② $\underbrace{y_1, y_2, y_3 \text{ ---- } y_k}_{\text{real values}} \rightarrow y_q$

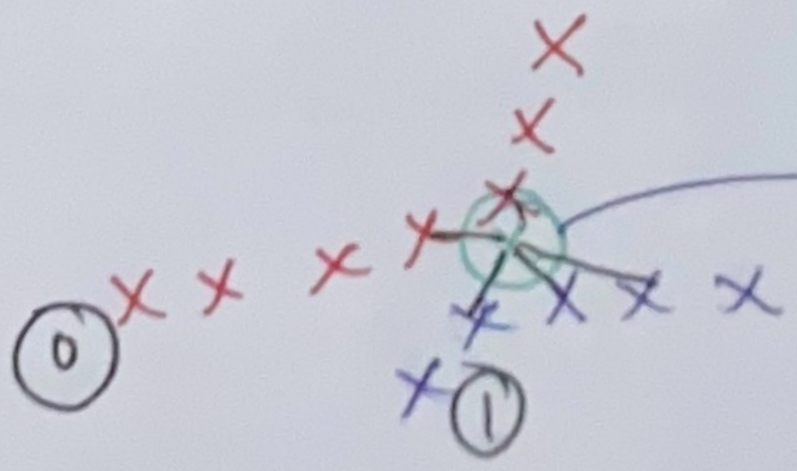
$y_q = \text{mean } (y_i)_{i=1}^{k}$

$y_q = \text{median } (y_i)_{i=1}^{k} \rightarrow$ less prone to outliers

extend

calasf $\rightarrow$ regression

Weighted K-NN



$x_q \rightarrow y_q$

k=5

2 → -ve
3 → +ve
] $y_q \rightarrow$ +ve

weighted K-NN

$x_q \longrightarrow$

				w_i	
x_1	y_1	d_1	(0.1) −ve	(10)	15
x_2	y_2	d_2	(0.2) −ve	5	
x_3	y_3	d_3	(1.0) +ve	1	1.75
x_4	y_4	d_4	(2.0) +ve	0.5	
x_5	y_5	d_5	(4.0) +ve	0.25	

$w_i = \frac{1}{d_i}$ [Simplest weight function]

$d_i \uparrow, w_i \downarrow$

$d_i \downarrow, w_i \uparrow$

$y_q = -ve$ as $15 > 1.75$

Voronoi Diagram (k-NN (k=1))

Voronoi diagram is partitioning of a plane into regions based on distance to points in a specific subset of the plane. That set of points (seeds, sites or generators) is specified beforehand and for each seed there is a corresponding region consisting of all points closer to that seed than to any region. These regions are called voronoi cells.

Kd-tree

Simple Implem
- K-NN: O(n) if d is small → time
  - k is small
  - O(n) → space

Time: O(n) → O(lgn) → kd-tree ∽1975

n=1024 lg(n)=10

↑ Computer graphics geometry

Binary Search Tree (BST)

→ binary search (Python)

prob: given a sorted array, find a number is present in array or not. (q=15)

1	2	3	4	5	6	7	8
1	2	6	8	12	15	17	18

A: ← Sorted

avg

① Construct a BST

Tree:

flow chart

root ← ≤10?

Y → {1,2,6,8} → ≤4

Y → {1,2} → ≤1.5

Y → 1 (leaf)

N → 2

N → ≤7?

Y → 6

N → 8

N → {12,15,17,18} → ≤16

Y → ≤13.5

Y → 12

N → 15

N → ≤17

Y → 17

N → 18

only 3 Comparisons

Linear Search: no of comparisons of $O(n)$

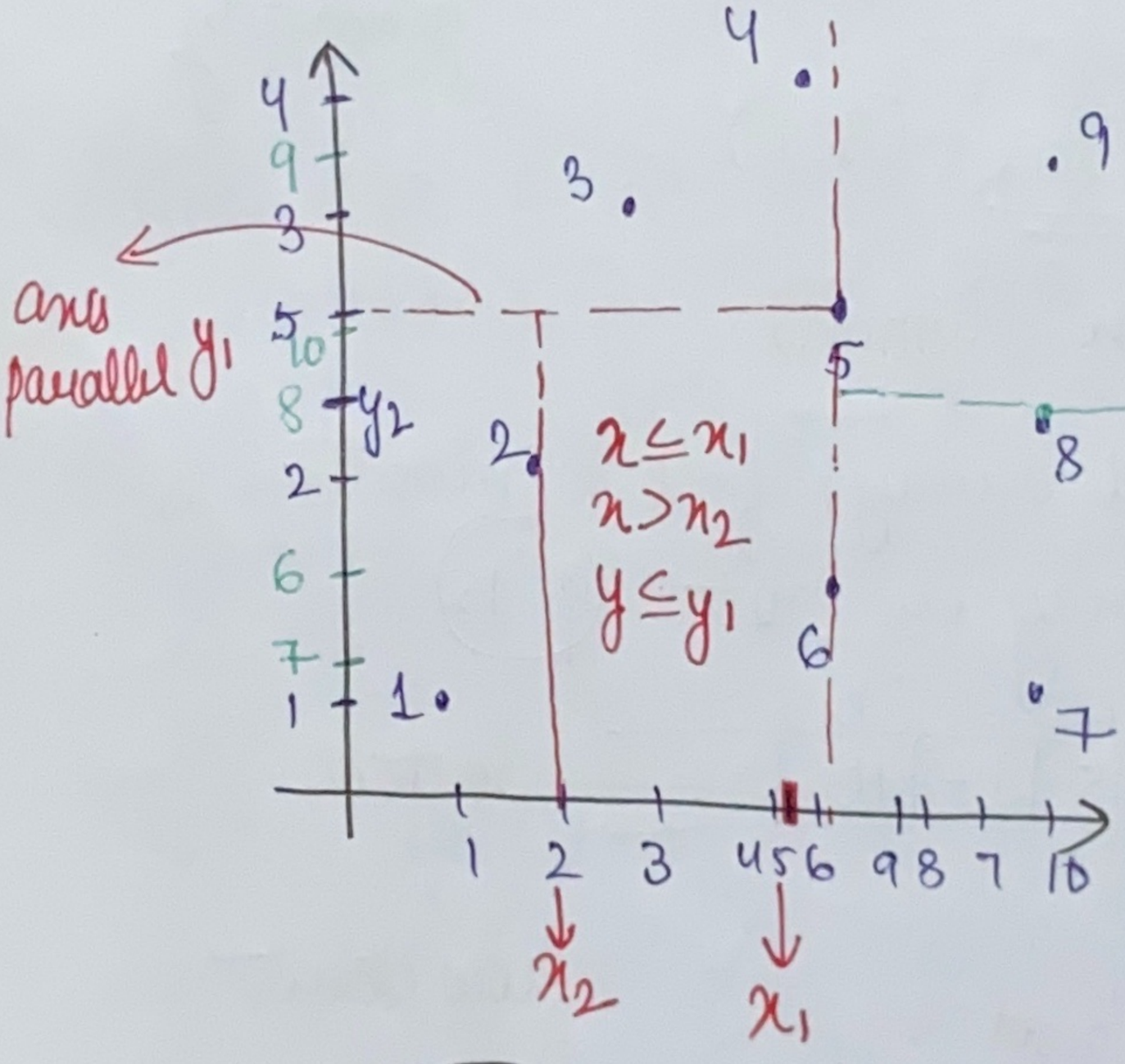
Binary Search : $O(\log n)$

Time complexity to search is $O(\log n)$

↓

depth of the tree

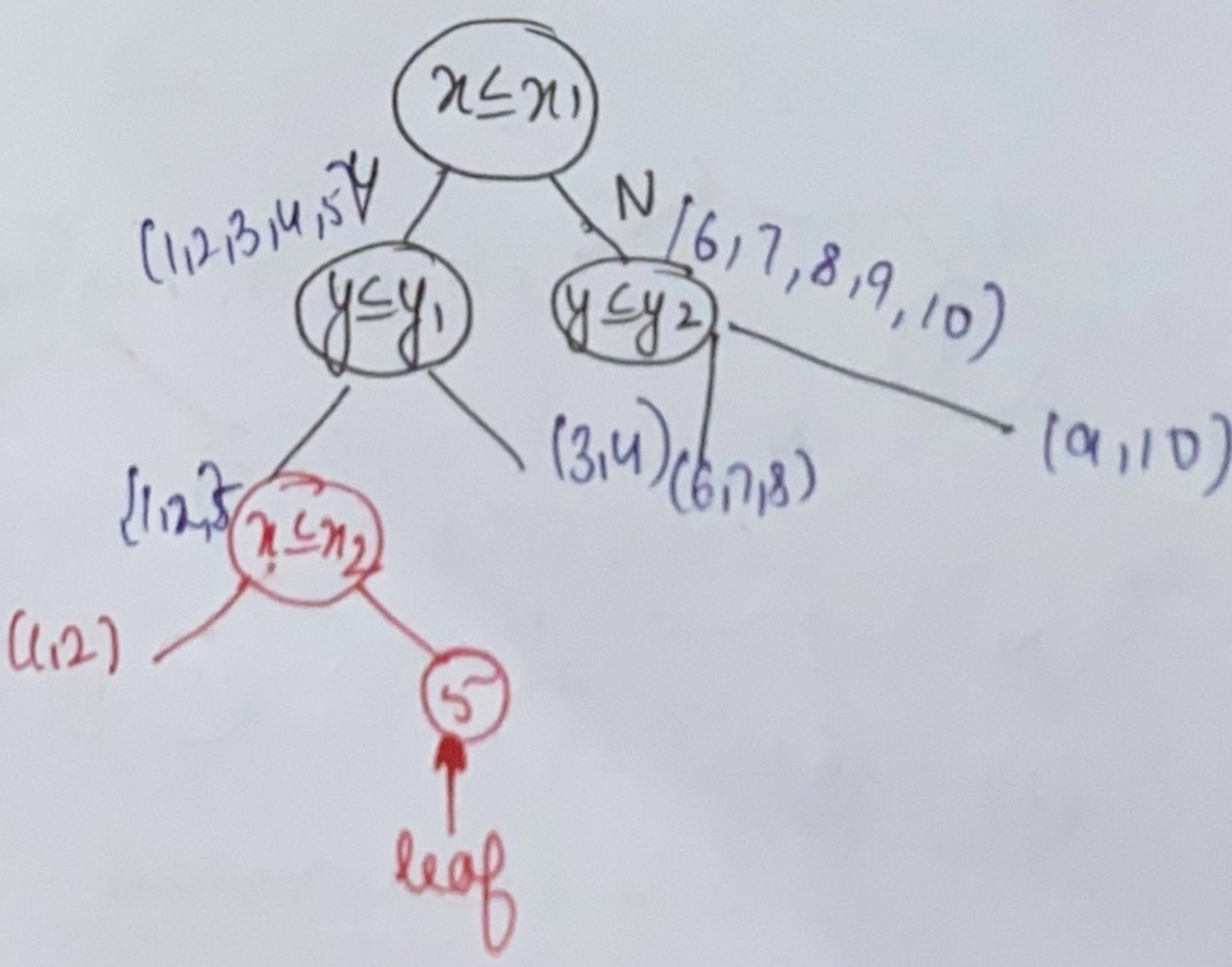
Build a Kd-Tree (2D)



① pick x axis, project pts on x axis and compute median

② Split data using median

③ Alternate b/w axis



Kd tree

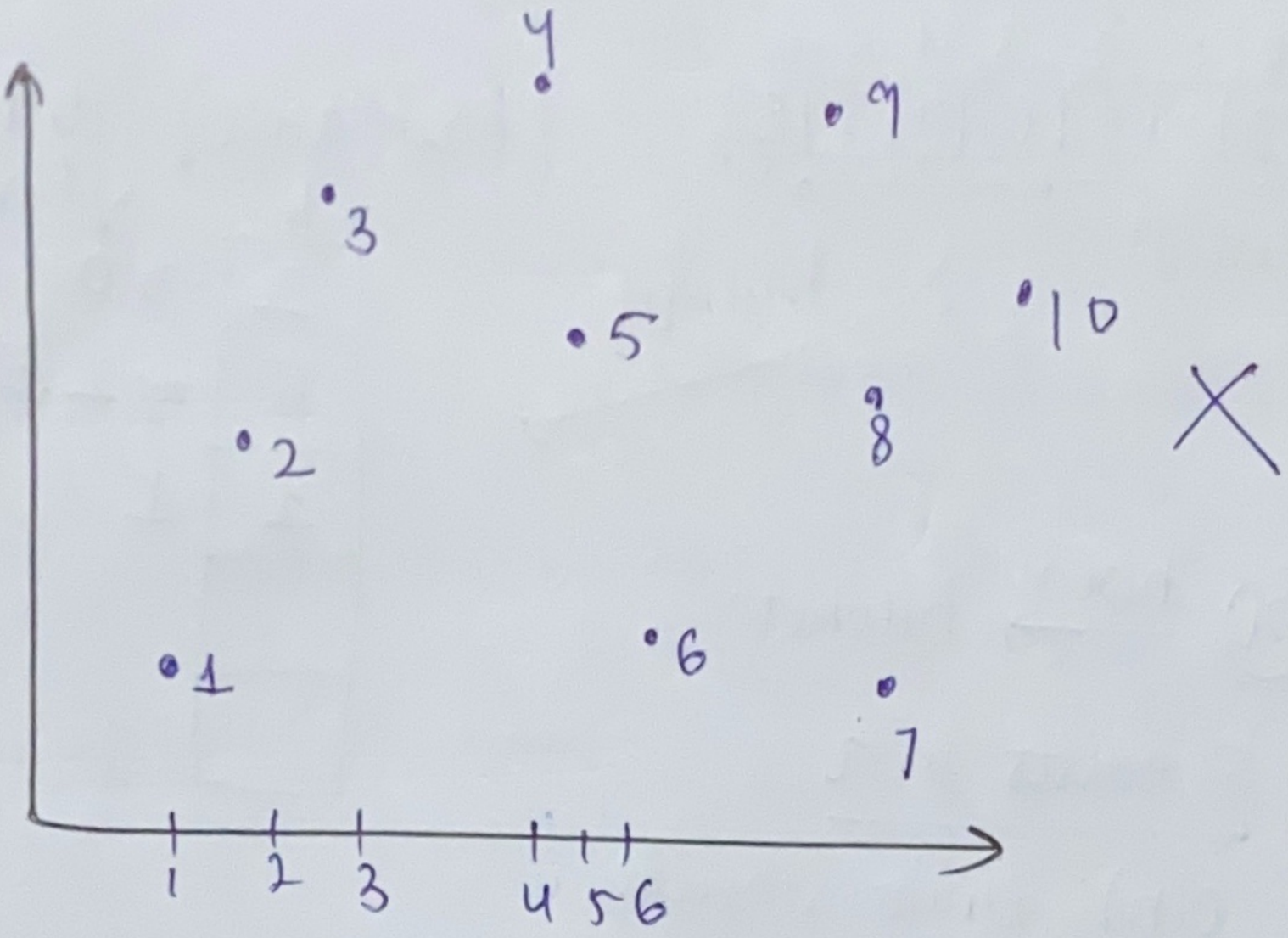
↓

breaking my space using axis parallel lines/planes into cyboids/hypercuboids

2D → axis parallel lines → rectangles

3D → " " planes → cuboids

nD → " " hyperplanes → hypercuboids



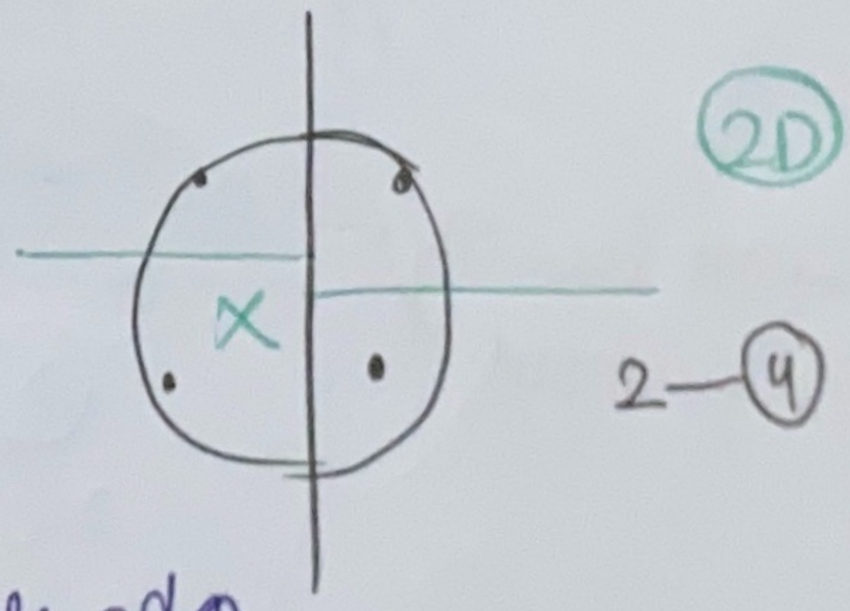
Limitation of Kd tree

① When d is not small

d=10 2^d = 1024

d=20 2^d ≈ 1M

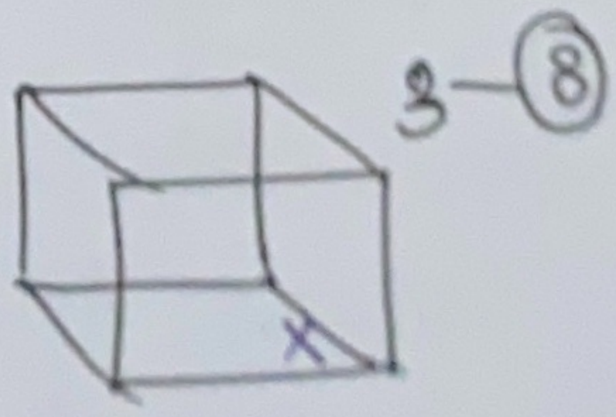
d↑; (worst-case) # adj cells 2^d↑



When d is not small {2,3,4,5}

Time complex: O(lgn) ← 1-NN

O(2^d lgn) ← ~~O(nlgn)~~



② When d is small

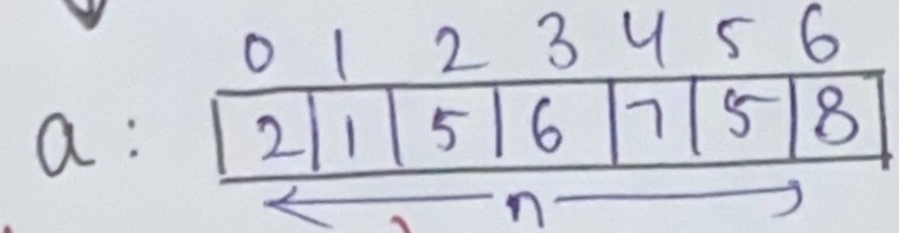
O(lgn)

↳ only if data is uniformly distributed!

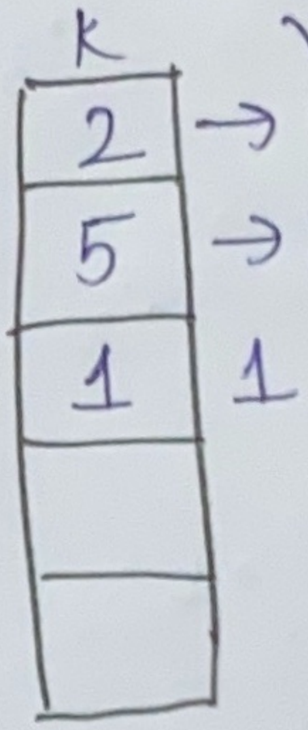
d → 2^d

Hashing & LSH

↳ locality Sensitive Hashing → (d large)



[hashtable] (dict)



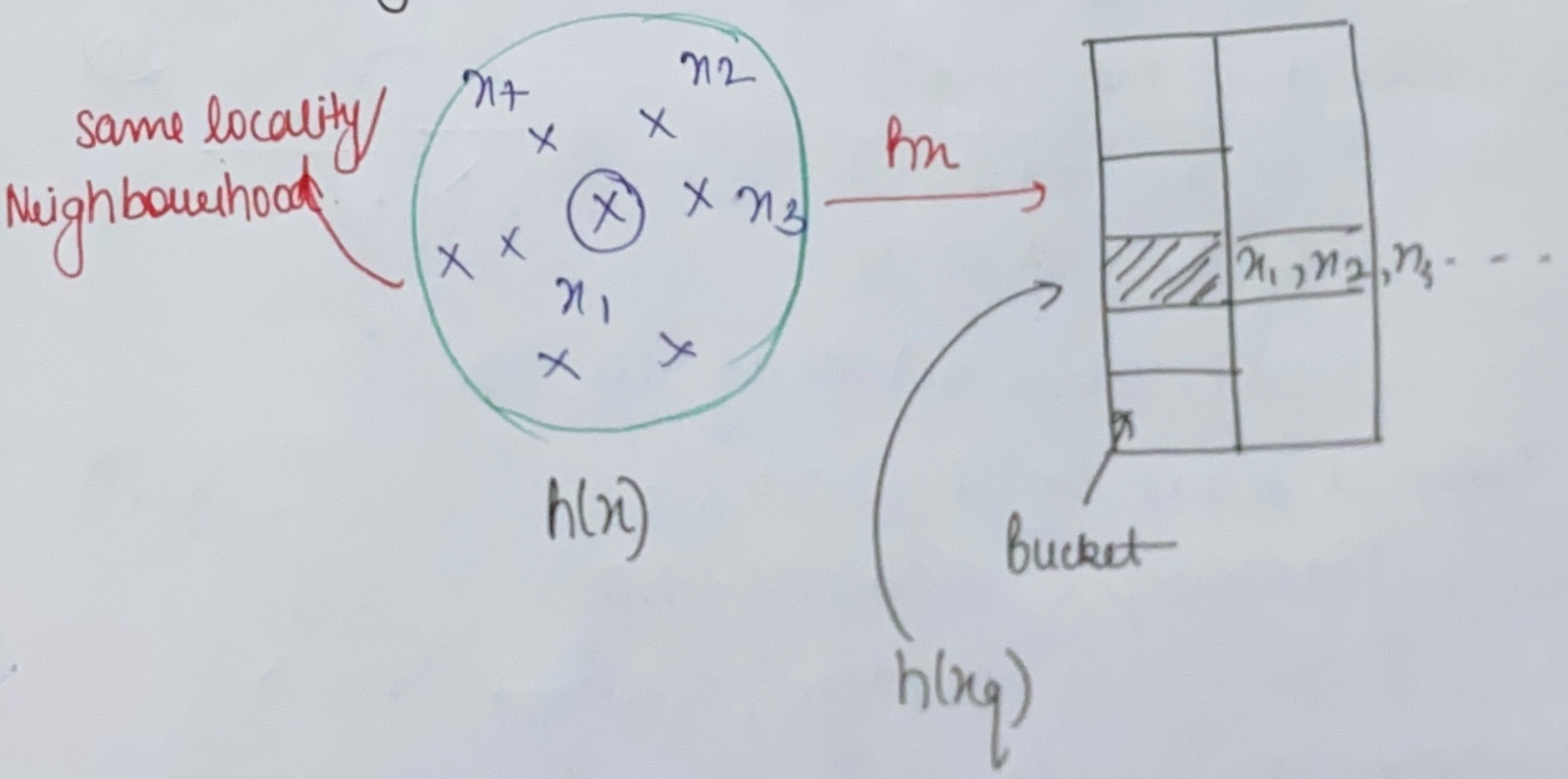
hash-fn

(x) $\xrightarrow{h(x)}$ bucket

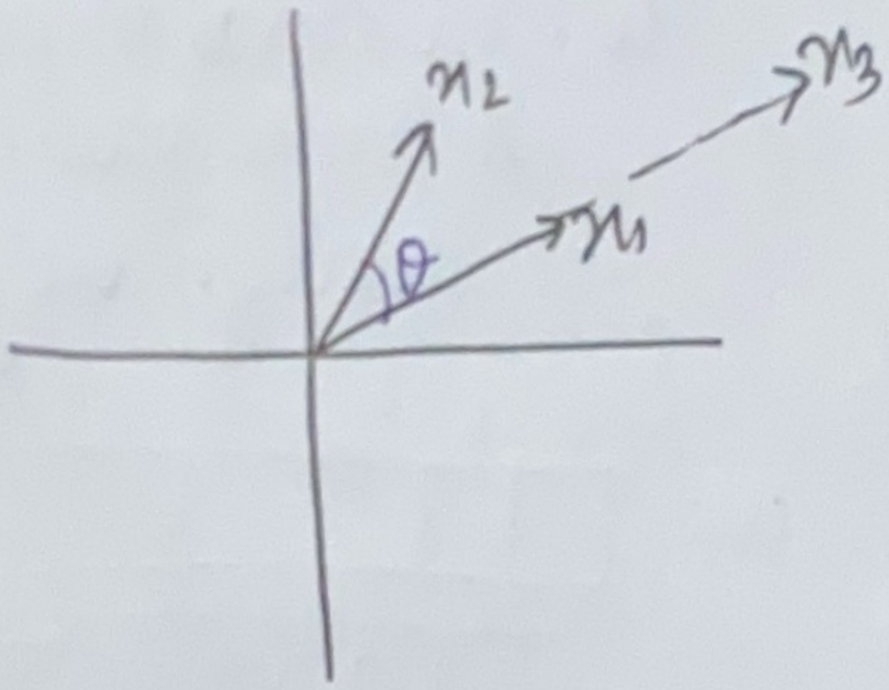
find if 5 exists in a

O(n): linear search

Locality Sensitive Hashing



Locality Sensitive Hashing for Cosine Similarity



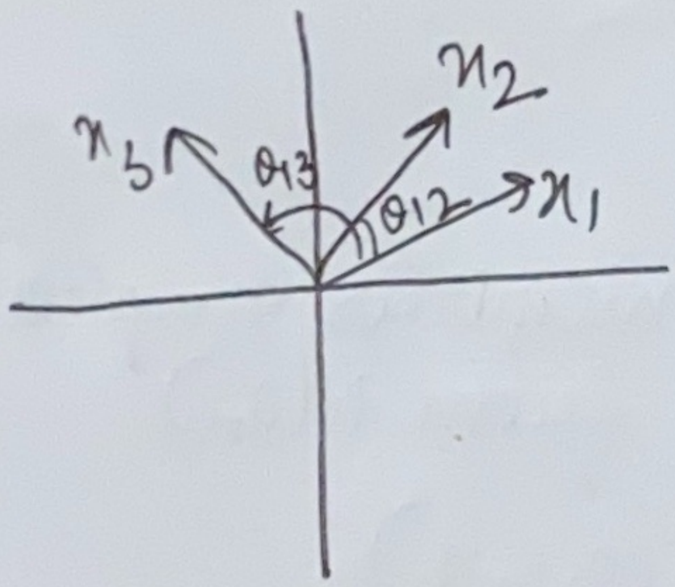
$\text{CosSim}(x_1, x_2) = \cos\theta$

$\text{Cos Sim}\ (x_1, x_3) = \cos(0) = 1$

$\theta \uparrow$; x_1 & x_2 are angularly separated [dis ↑ Cosine ↓ Sim ↓]

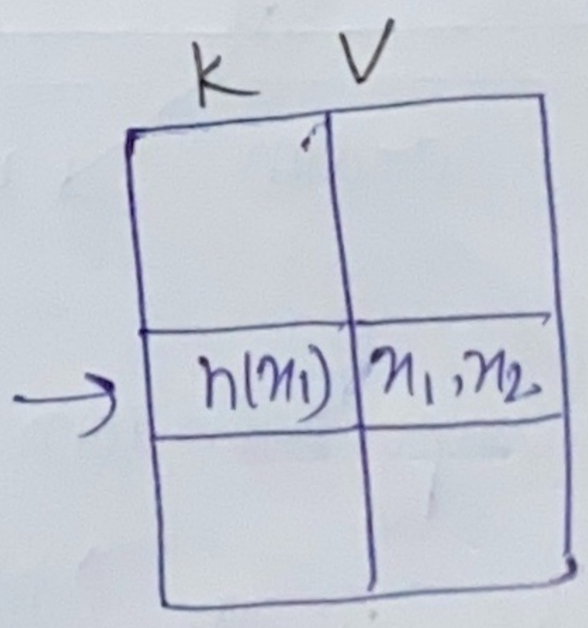
Cosine-Sim ⇆ angular similarity

LSH for Cos-Sim



$h(x_1) = h(x_2) =$ key

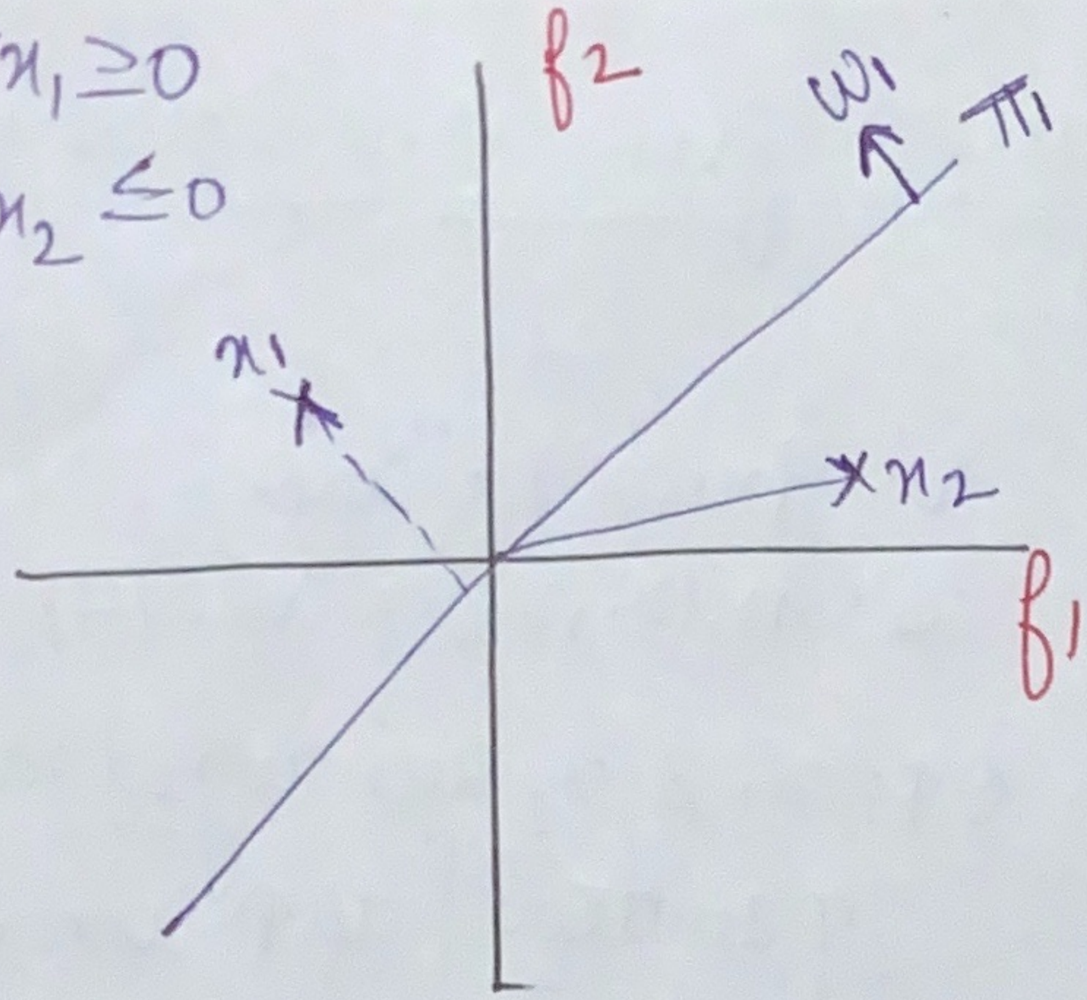
$\theta_{12} \downarrow \theta \Rightarrow x_1$ & x_2 are similar
↳ small



LSH: Randomized algorithm

↳ I will not always give correct ans

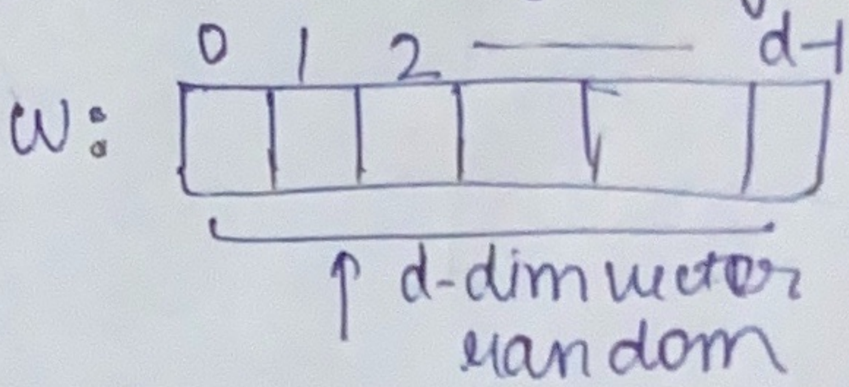
↳ prob. I will give correct ans with high probability



① Random-hyperplane

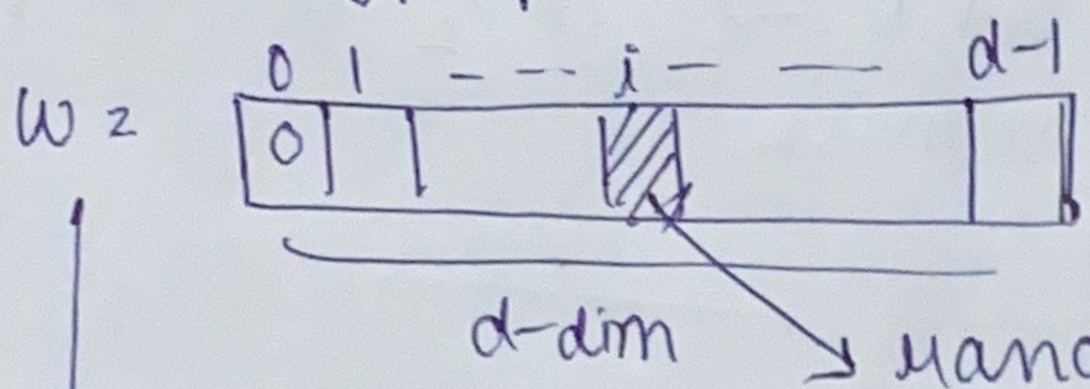
$w^T x = 0$

w: normal to plane & plane passing through origin

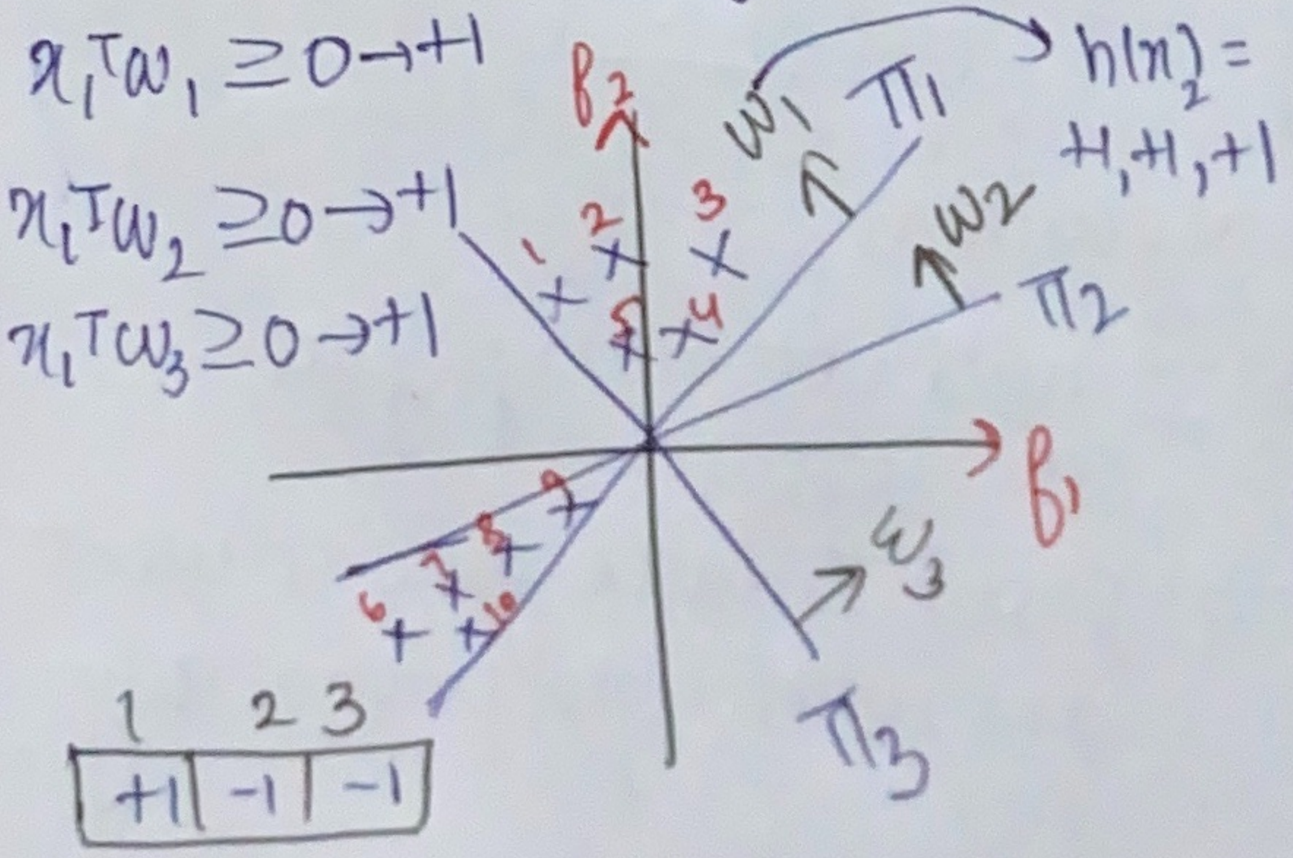


(2D) w_i: unit normal vector to π_1

Random-hyperplane

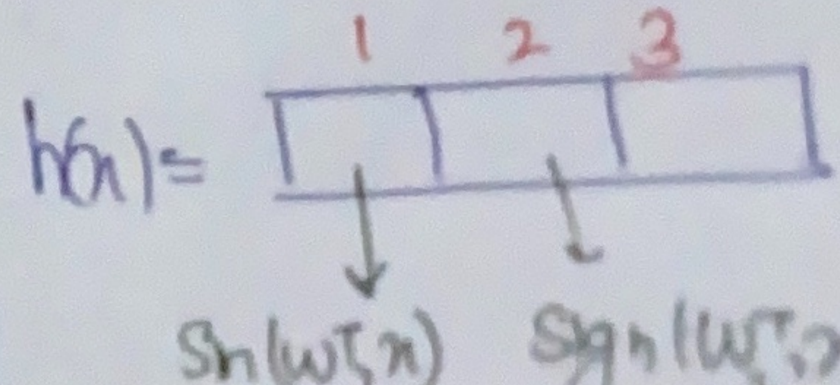


w_2 = numpy.random.normal(0,1,d)



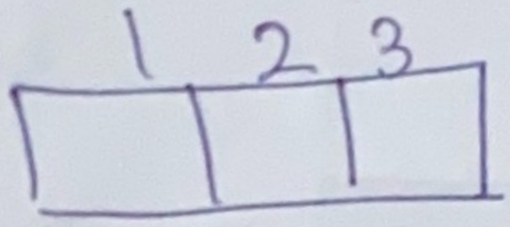
n = ## hyperplanes

Point → Slide

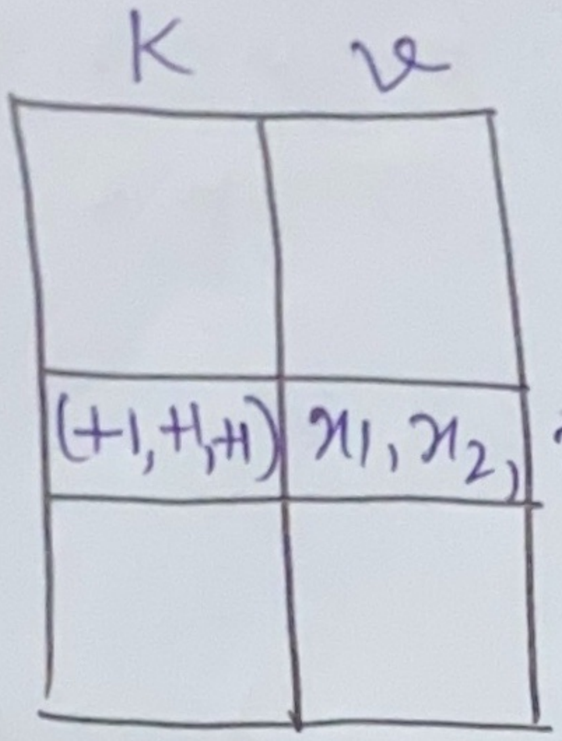


hash-table

$x \rightarrow h(x)$



vector of size m



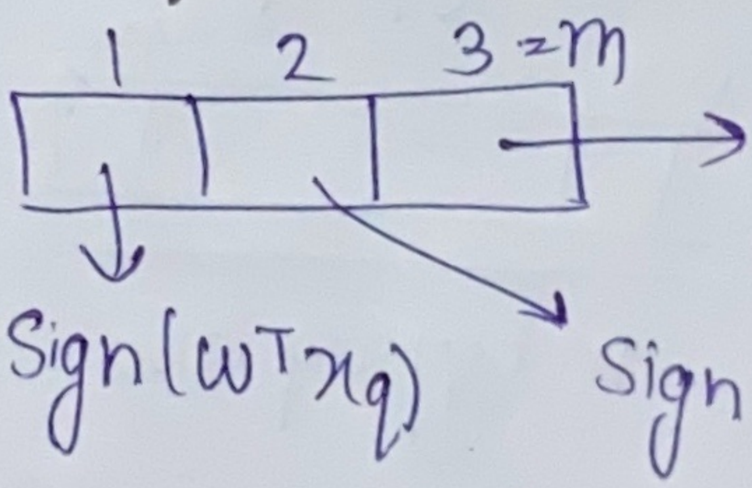
Time: $O(md)$

$\sim O(mdn)$

Space: $O(nd)$

Query point (x_q)

① $h(x) =$



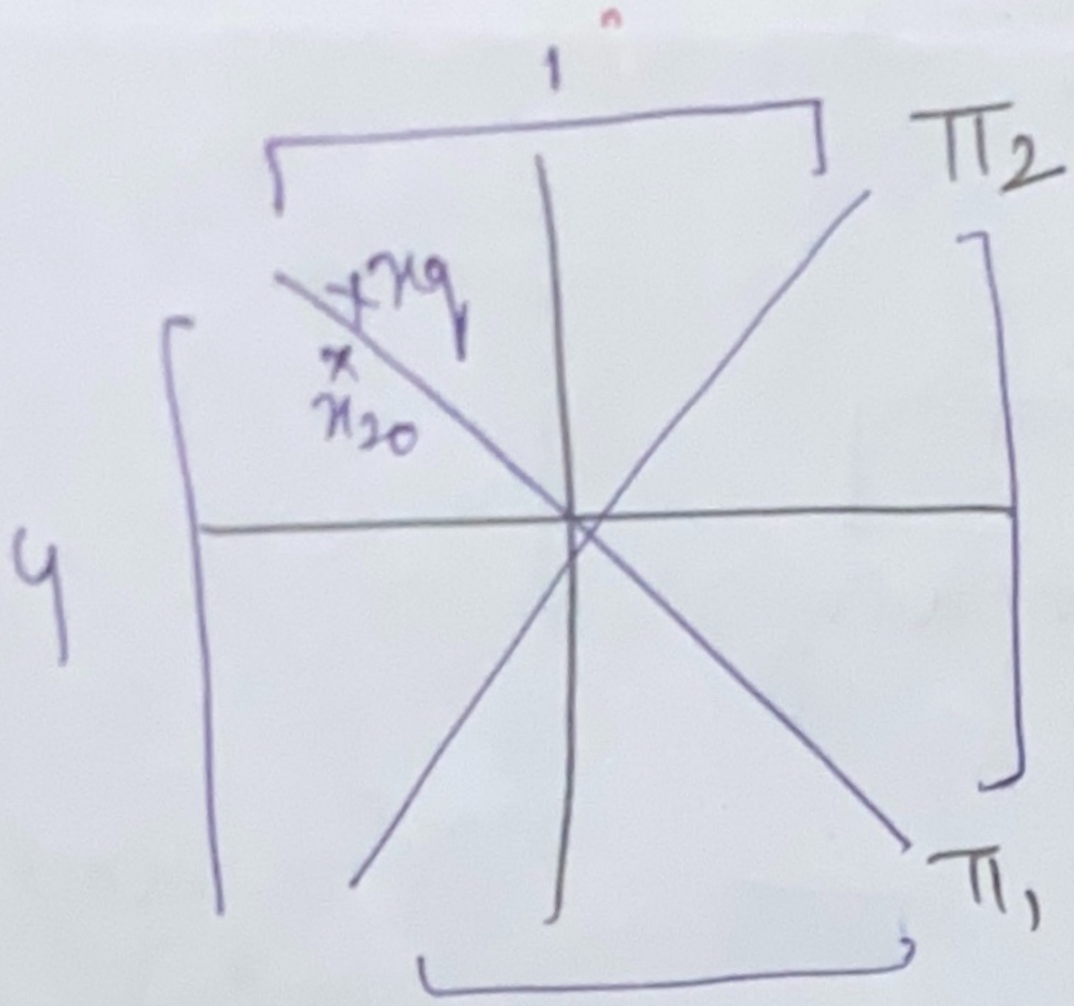
$h(x_q)$ as key in hash-table

d.get($h(x_q)$) $\rightarrow \{x_1, x_2, x_3, x_4, x_5\}$

Typically # hyperplanes

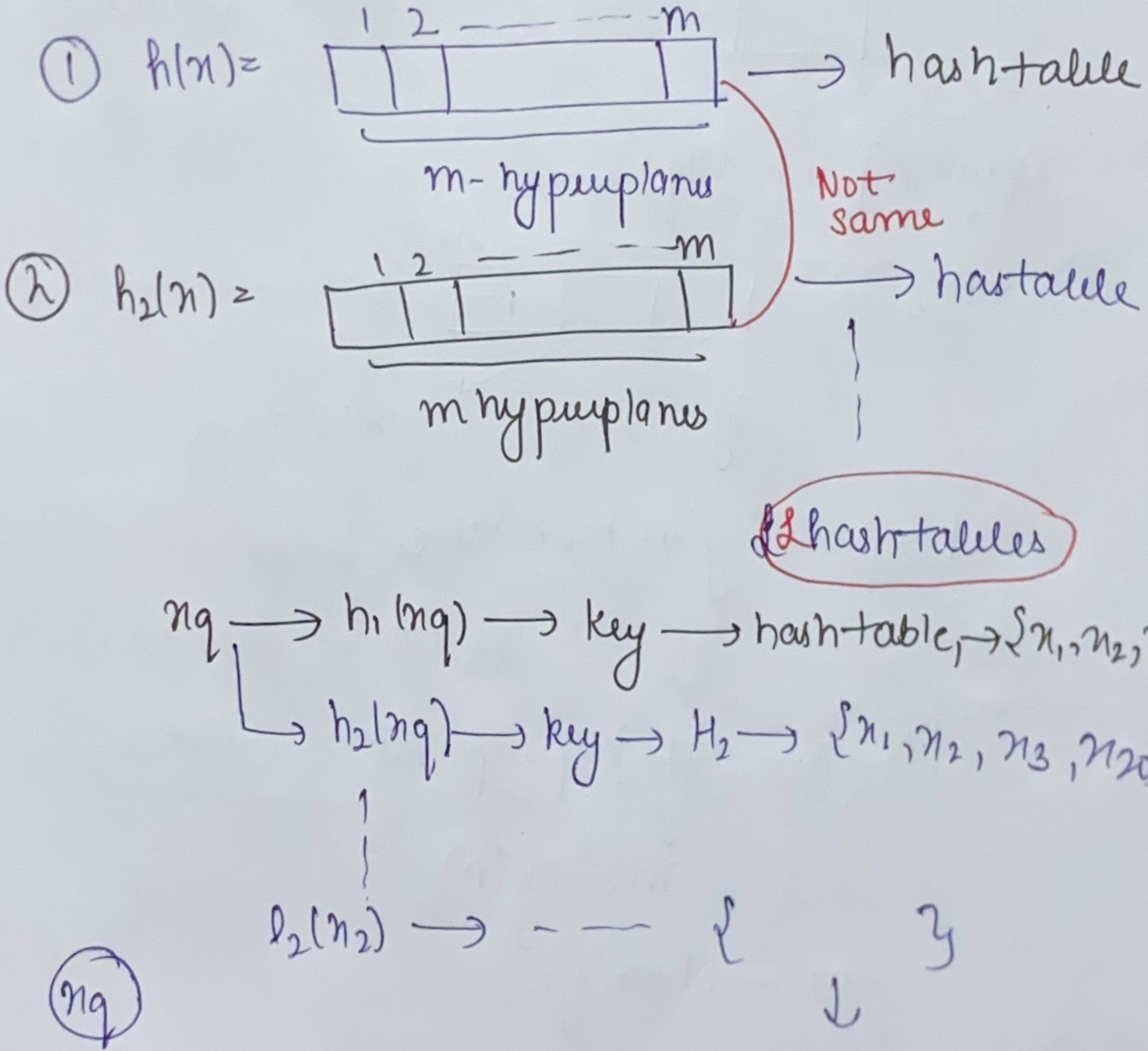
$m \approx \lg(n)$

Time :- $O(d * \lg n)$



Cos Sim
(x_{20} very close to x_q)

* Could miss a NN in Cos Sim as your measure



Time complex to query given a hashtables

$\downarrow$

$O(md\ell)$

$\uparrow$

{ as no of hyperplane $\uparrow$

slices $\uparrow$

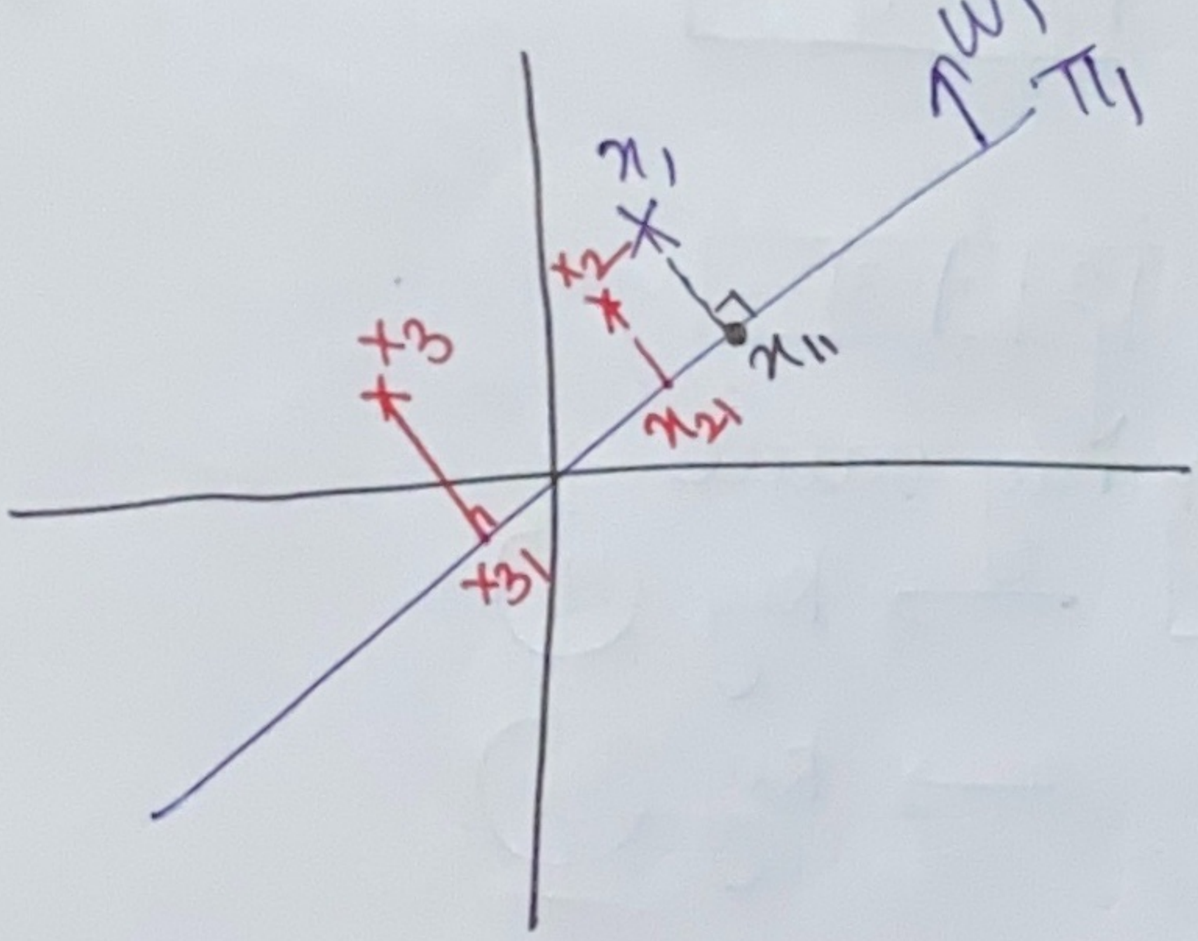
pts per slice $\downarrow$ }

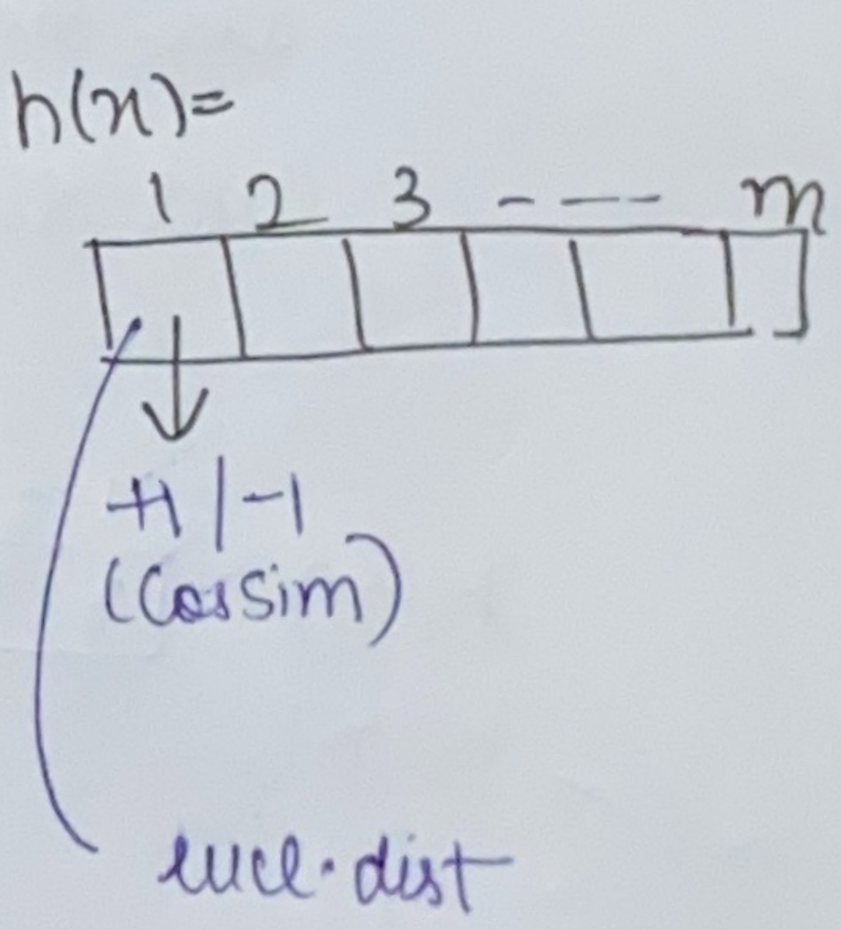
LSH for Euclidean distance

$\longrightarrow$ LSH : Cos. Sim

$\longrightarrow$ eucl dist : Simple extension

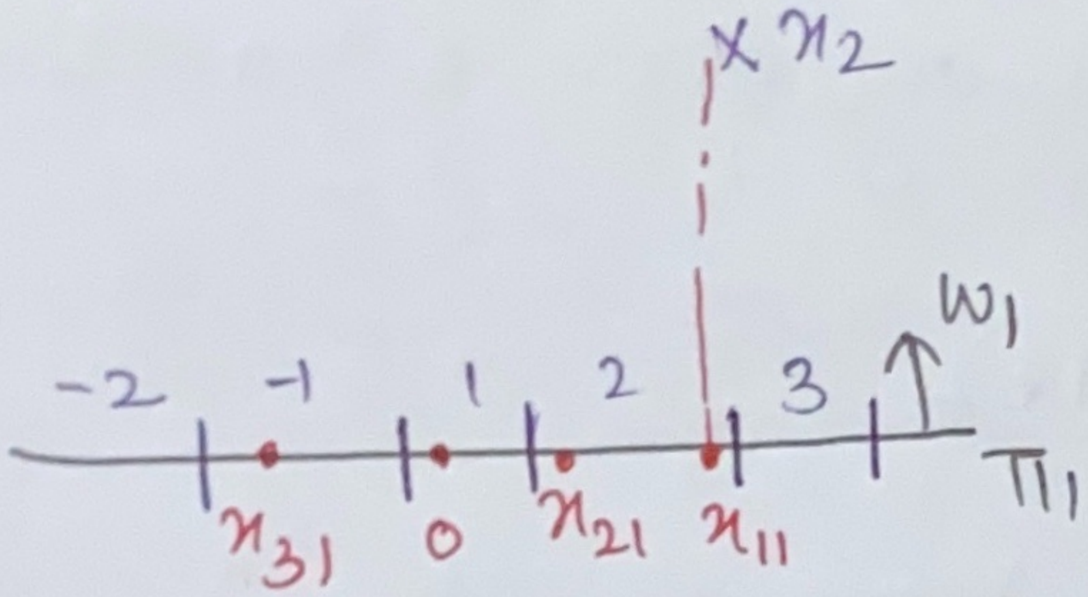
$d(x_1, x_2) << d(x_1, x_3)$





x_{11} = proj of x_1 on π_1,

LSH : Similar / closer points
should go to same pocket

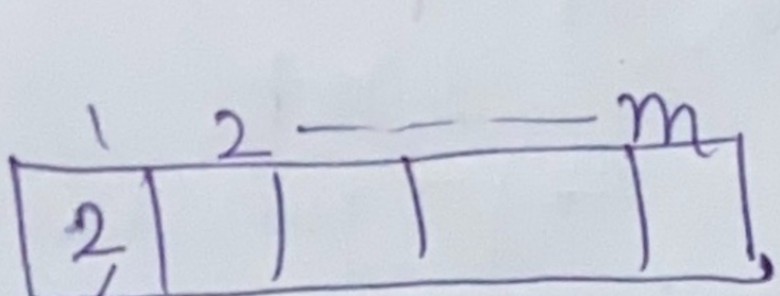


Breaking plane into pieces / regions

↑ 'a' regions

a = 8

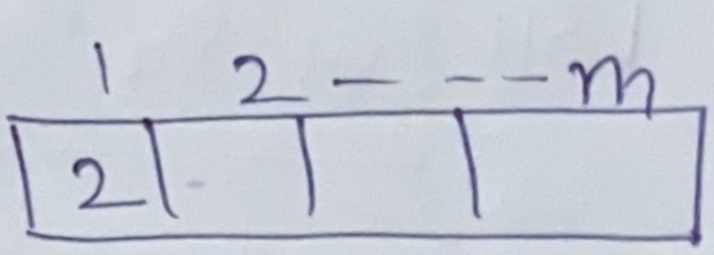
$h(x) =$



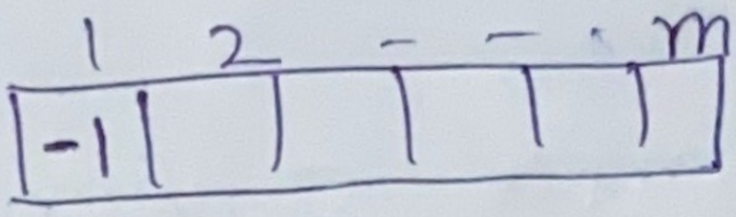
π_1

{ x_{11} & x_{21} → same region but x_1 is very far from x_2

$h(x_{21}) =$

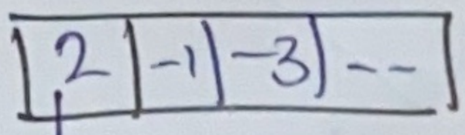


$h(x_{31}) =$



Cos- Sim LSH: | +1 | -1 | - - |

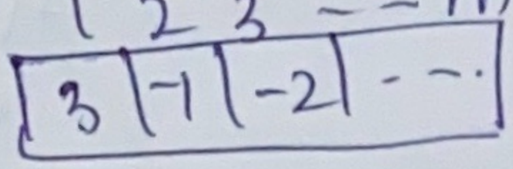
Euc dist →



any value

1 2 3 - - m

$h_1(x)$:



→ key H_1

→ key H_2

x — $h_2(x)$ ⋮ $h_L(x)$

LSH is not perfect

↳ probabilistic / randomized algo

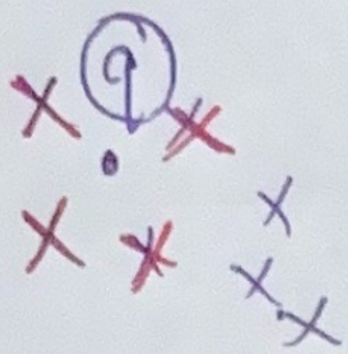
K-NN: Probabilistic Class Label

2-class classfn

$y_i \in \{0, 1\}$

Majority Vote

$y_q = (-ve)$ (x)



$n_q = \{x, x, x, x, x, x, x\}$

n_q: 4 (-ve pts) : 3 (+ve pts)

$$P(x_q = -ve) = \frac{4}{7} = \frac{\#\text{-ve pts}}{\text{Total \# pts}}$$

⇒ Probabilistic Output (class label)

↳ how certain we are