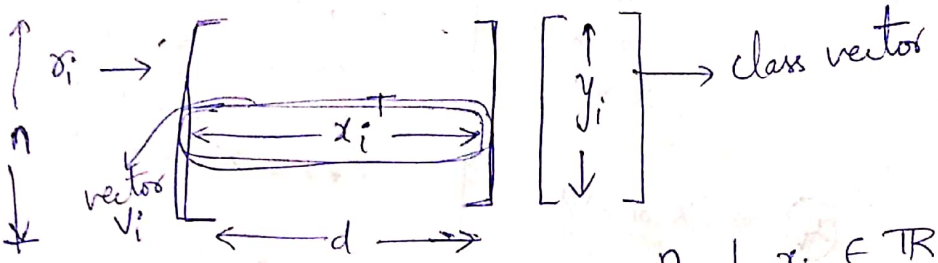
K- Nearest Neighbours

Notation in classification algorithms

Let us consider Amazon Fine food review dataset

$r_i \longrightarrow$ (vector) $\longrightarrow x_i$

$\hookrightarrow$ Column-vector



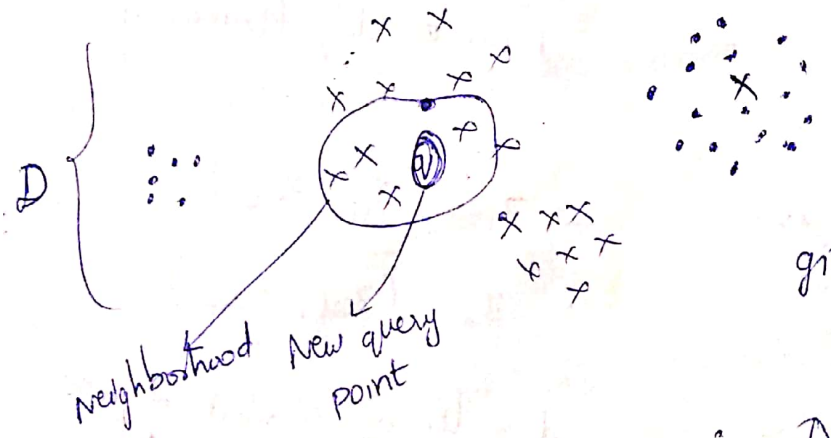
$$\text{Dataset} = \mathcal{D} = \{(x_i, y_i)_{i=1}^{n} \mid x_i \in \mathbb{R}^d,\ y_i \in \{0, 1\}\}$$

such that

$0 \rightarrow$ -ve, $1 \rightarrow$ +ve

K-NN geometric intuition with a toy example

→ 2-D toy dataset



given $x_q \rightarrow$ find y_q

① Find K-nearest points to x_q in $\mathcal{D}$. Let $K = 3$

$\Rightarrow \{x_1, x_2, x_3\} \rightarrow$ 3 NN to x_q

② Take class labels for all these points

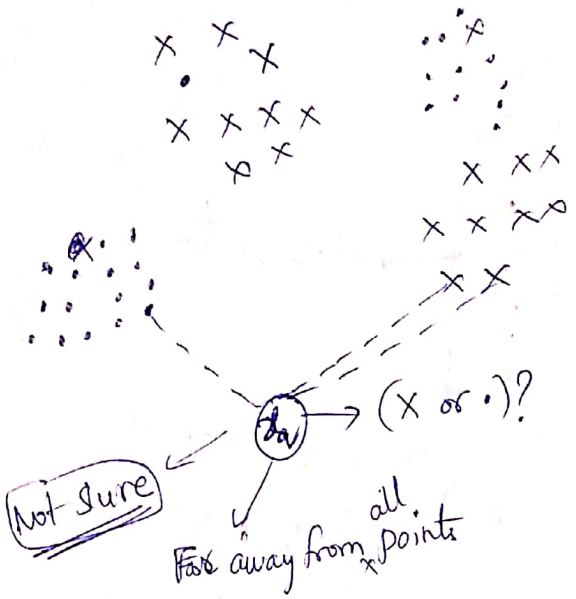
$\{y_1, y_2, y_3\} \longrightarrow$ Take majority vote

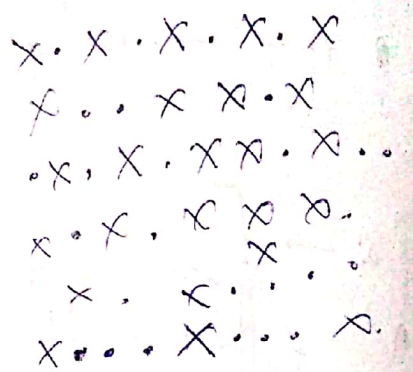
Ex $+, +, - \Rightarrow y_q = +ve$

Scanned by CamScanner

Note: It's always good to take K = odd.

Failure Cases of KNN





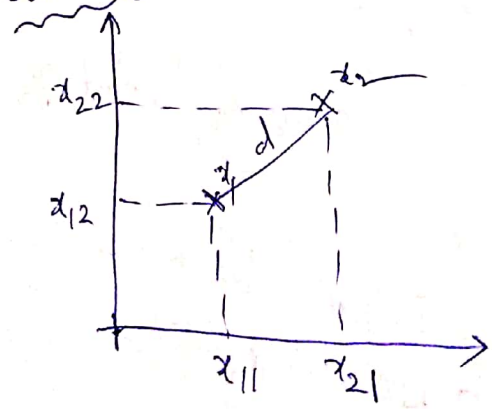
jumble of +ve/-ve
(Randomly Spread)

→ when query point is far away from all the points, then we cannot surely say about the class to which it belongs.

Distance Measures: Euclidean (L2), Manhattan (L1)

Minkowski, Hamming

Euclidean distance



$x_1 = (x_{11}, x_{12})$

$x_2 = (x_{21}, x_{22})$

d = len of shortest of line from x_1 to x_2

$$d = \sqrt{(x_{21}-x_{11})^2 + (x_{22}-x_{12})^2} = \|x_1 - x_2\|$$

EUCLIDEAN DISTANE

Scanned by CamScanner

For d-dimensional points, ie $x_1 \in \mathbb{R}^d$, $x_2 \in \mathbb{R}^d$

then

$$\|x_1 - x_2\|_2 = \sqrt{\sum_{i=1}^{d} (x_{1i} - x_{2i})^2}$$

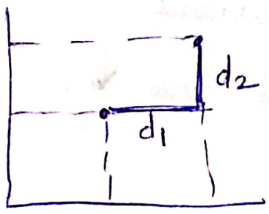
Note: Euclidean distance is represented as

$\|x_1 - x_2\|_2 \longrightarrow$ L2 Norm of a vector $(x_1 - x_2)$

$\|x_1\|_2$ = Euc distance of x_1 from origin.

Manhattan distance

$$d_{manhattan} = \sum_{i=1}^{d} |x_{1i} - x_{2i}|$$



$d_{manhattan} = d_1 + d_2$

Manhattan distance is represented as

$\|x_1 - x_2\|_1 \longrightarrow$ L1 norm of vector $(x_1 - x_2)$

Minkowski distance (L-p norms)

$$\|x_1 - x_2\|_p = \left(\sum_{i=1}^{d} |x_{1i} - x_{2i}|^p\right)^{1/p}$$

$p = 2 \Rightarrow$ Minkowski dist = Euclidean distance

$p = 1 \Rightarrow$ Minkowski dist = Manhattan distance

$\longrightarrow$ L-p norm of vector $(x_1 - x_2)$

Scanned by CamScanner

→ "distance" is for two points
"norms" are for vector

Hamming distance

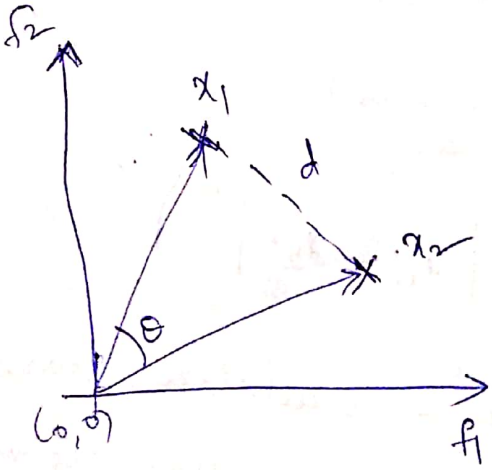
→ used in text processing
x_1, x_2 → boolean vectors

Hamming-dist (x_1, x_2) = No. of locations/dimensions where binary vectors differ

Cosine distance & Cosine Similarity

→ Similarity v/s distance } As distance ↑, Similarity ↓
As distance ↓, Similarity ↑

→ $\boxed{1 - \text{cosine-Similarity}(x_1, x_2) = \text{Cosine-dist}(x_1, x_2)}$



d = euc distance

Cos-Sim(x_1, x_2)
$= \cos\theta$

→ If points are farther away, then angle 'θ' is more ⇒ cosθ is less ⇒ less similar.

$$\boxed{\cos(\theta) = \frac{x_1 \cdot x_2}{\|x_1\|_2 \, \|x_2\|_2}}$$

L2 norm of x_1

→ Relationship b/w euclidean dist & Cosine Similarity

$$\|A-B\|^2 = (A-B)^T(A-B)$$
$$= \|A\|^2 + \|B\|^2 - 2A^TB$$

when A & B are normalized to unit length
then $\|A\|^2 = \|B\|^2 = 1$

$$\Rightarrow \|A-B\|^2 = 2 - 2A^TB = 2(1 - A^TB)$$
$$= 2(1 - A\cdot B) = 2(1 - \cos(A, B))$$

$$\boxed{\|A-B\|^2 = 2(1 - \underbrace{\cos(\theta)}_{\text{cosine-similarity}})}$$

How to measure the effectiveness of K-NN

$D_n \rightarrow D_{train}(n_1)$, $D_n \rightarrow D_{test}(n_2)$

$D_{train} \cup D_{test} = D_n$

$D_{train} \cap D_{test} = \phi$

$n_1 + n_2 = n$

$$\text{Accuracy} = \frac{\text{No of points for which KNN gave correct class label}}{\text{Total No of points in } D_{Test}\ (n_2)}$$

Test/Evaluation time and Space Complexity

Input : D_{train}, k, $x_q \in \mathbb{R}^d$

Output : $y_q \in \{0, 1\}$

Scanned by CamScanner

pseudcode

KNNpts = [ ]

for each x_i in D_{train} : → n pts ; d-dim

n times
- Compute $d(x_i, x_q) \rightarrow d_i \rightarrow O(d)$
- Keep the smallest k-distances → (x_i, y_i, d_i) → KNNpts[ ]

$\hookrightarrow O(1)$

$\rightarrow O(nd)$

count_pos = 0 ; count_neg = 0

for each x_i in KNNpts

if y_i is +ve

count_pos += 1

else count_neg += 1

if count_pos > count_neg

return $y_q = 1$

else return $y_q = 0$

NOTE: There is no training in KNN

Time complexity : $O(nd)$

If $d << n \Rightarrow$ ~~O(d)~~ $O(n)$

$O(Knd)$

Space Complexity : $O(nd)$

KNN Limitations

→ For simple implementation of K_NN, the Space Complexity is $O(nd)$, & the time complexity is $O(nd)$. Due to this, it is not extensively used

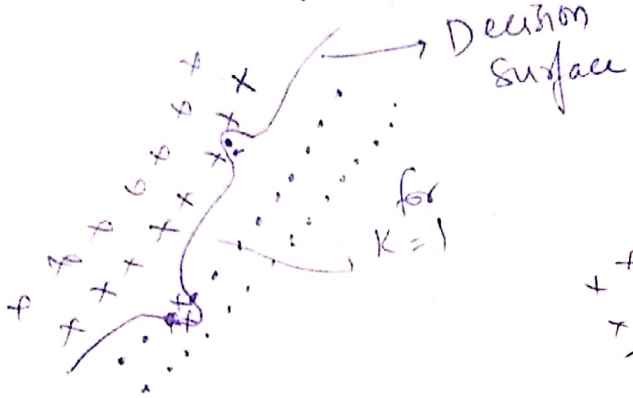
→ Kd-Tree & LSH implementation of K-NN improve time and space complexity

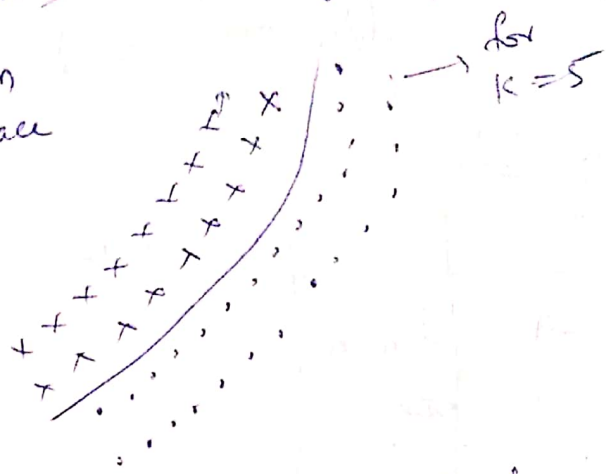
Scanned by CamScanner

Decision Surface for KNN as K Changes

K → hyper parameter

Decision Surfaces } Seperates ~~classes~~ different classes



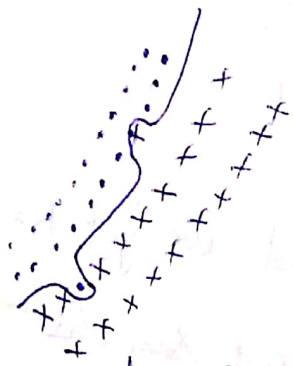


→ In K-NN, the Smoothness of the decision Surface increases as K-increases.

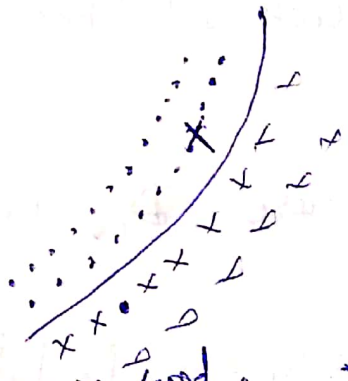
Note max value of K = n = no of points

Overfitting and Underfitting

"fitting" the function to the data



(overfitting)



(Good ~~Under~~fitting)

Scanned by CamScanner

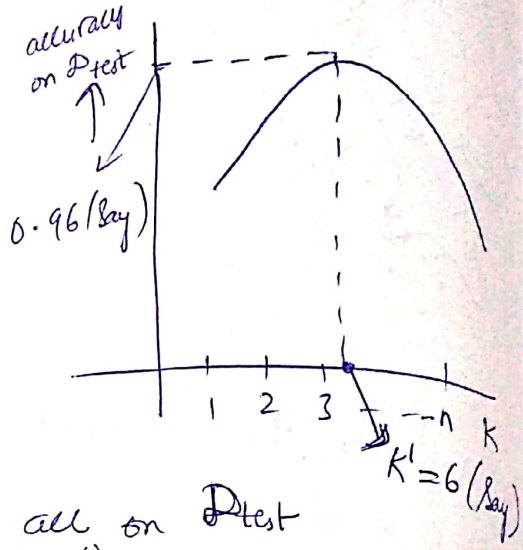
Need for cross validation

How to determine K?

D_n → D_{train} (70%), D_{test} (30%)

one-idea:

	Data	Accuracy on D_{test}
K = 1	D_{train}	0.78
K = 2	"	0.82
⋮	⋮	⋮



K = K' gives me the best acc on D_{test} (0.96)

NOTE: "GENERALIZATION" } · The function should work on future/unseen data

For cross validation D_n → D_{train}, D_{cv}, D_{test}.

→ we use D_{cv} to compute "K"

Note: If 'K' is calculated using D_{test} and we get K=6 for which accuracy is 0.96. Then we cannot say that the accuracy for future data will be 0.96.

So we use another data called (cross validation) data (D_{cv}) which we use to compute the value of K. Say, if we get K = 6 for which accuracy = 0.96 then we can say that accuracy for future data (D_{test}) for K=6 will also be 0.96

K-fold cross validation

$D_n \rightarrow$ D_{train} (60%), D_{cv} (20%), D_{test} (20%)

Problem is } D_{train} = 60% only.

Note: More the training data, better the algorithm is,

→ Is there any way of combining training and cross validation?

Soln is K-fold cross validation

K-fold CV

① $D_n \rightarrow$ D_{train} (80%) → f(K-NN)

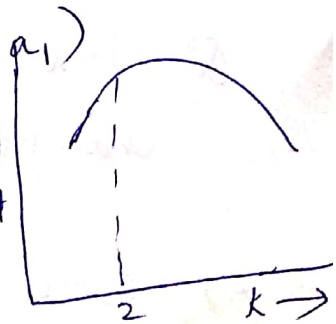
D_{test} (20%) – Unseen.

② D_{train} ← Randomly

D_1	D_2	D_3	D_4
1	2	3	4

4- equal sized parts

objective is to get the following plot



③

		Train	CV	acc on CV
K=1	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
4-fold Cross Validation	K=2	$D_1 D_2 D_3$	D_4	a_4
	K=2	$D_1 D_2 D_4$	D_3	a_3
	K=2	$D_1 D_3 D_4$	D_2	a_2
	K=2	$D_2 D_3 D_4$	D_1	a_1

Take avg (a_4, a_3, a_2, a_1) gives accuracy on dataset acc at K=1

Take avg to get accuracy at K=2

Scanned by CamScanner

(K') - fold CV

→ what is the right K' ? ★★★

Rule of thumb : 10 - fold CV

→ · If we K' - fold CV, then the time complexity of k-NN increases by K' times

→ It's a one time effort

Visualizing train, validation and test datasets

Points to remember:

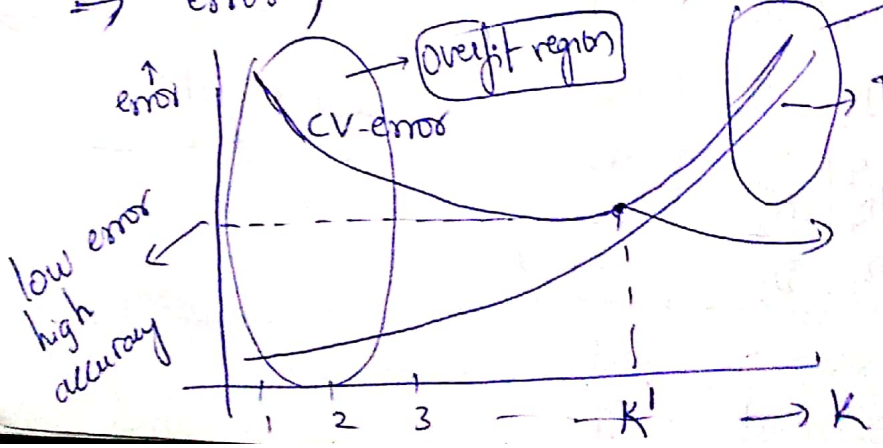
→ If there are many +ve pts from D_{train} in a region, then it is high likely that ~~+ve pt~~ to find +ve pt from D_{cv} in that region.

How to determine overfitting and underfitting

K-fold CV → best K } neither overfit nor underfit

Q) How can we be sure that we are not underfitting or overfitting ?

→ error ; train-error ; validation error



Scanned by CamScanner

low error => high accuracy

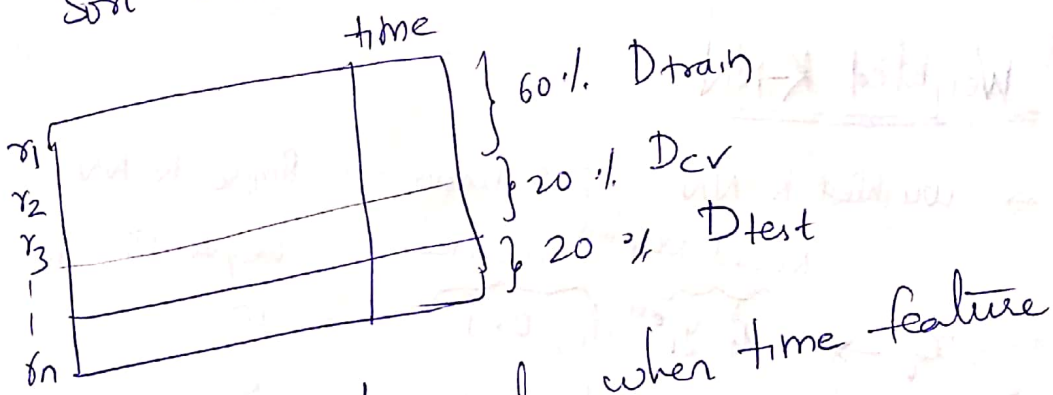
→ Train error is high } underfit
validation error is high

Train error is low } overfit
validation error is high

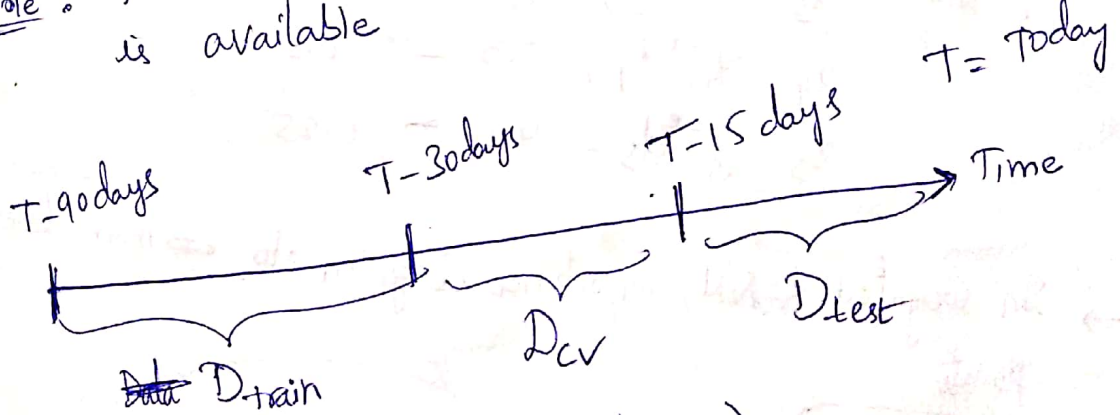
Time based Splitting (TBS)

Ex. For Amazon food Reviews, time based splitting is better than random splitting

step 1: Sort D_n in ascending order of time



Note: TBS can be done only when time feature is available



(Time based slicing)

Note- whenever time is available & things/behavior or data changes over time then TBS is preferable.

Scanned by CamScanner

K-NN for regression

① Given x_q, find K-nearest neighbors

$(x_1, y_1), (x_2, y_2) \text{ - - - - } (x_K, y_K)$

② $y_q \leftarrow y_1, y_2, y_3 \text{ - - - } y_K \quad \in \mathbb{R}$

$\Rightarrow y_q = \text{mean}\,(y_i)_{i=1}^{K}$

(or)

$y_q = \text{median}\,(y_i)_{i=1}^{K}$

Note: Some of the classification algo can be extended to regression (not always). K-NN is one among them.

Weighted K-NN

→ weighted K-NN : Extension to Simple K-NN

	Nearest neighbors	distance		weights $w_i = \frac{1}{d_i}$
$x_q \rightarrow$	x_1, y_1 +ve	d_1, 0.1	—	10
	x_2, y_2 -ve	d_2, 0.2	—	5
Consider (5-NN)	x_3, y_3 +ve	d_3, 1.0	—	1
	x_4, y_4 +ve	d_4, 2.0	—	0.5
	x_5, y_5 +ve	d_5, 4.0	—	0.25

→ In weighted K-NN, importance is given to ~~to~~ more ~~least~~ nearest points

→ $\boxed{w_i = \frac{1}{d_i}}$ is one of the weighing schemes

~~→ Sum of weig~~

Scanned by CamScanner

→ Sum of weights of all -ve points = $10+5 = 15 = W_N$

" " " " " +ve points = $1+0.5+0.25 = 1.75$
$= W_P$

Since $W_N > W_P$,
the point x_q can be ~~classi~~ declared as -ve point

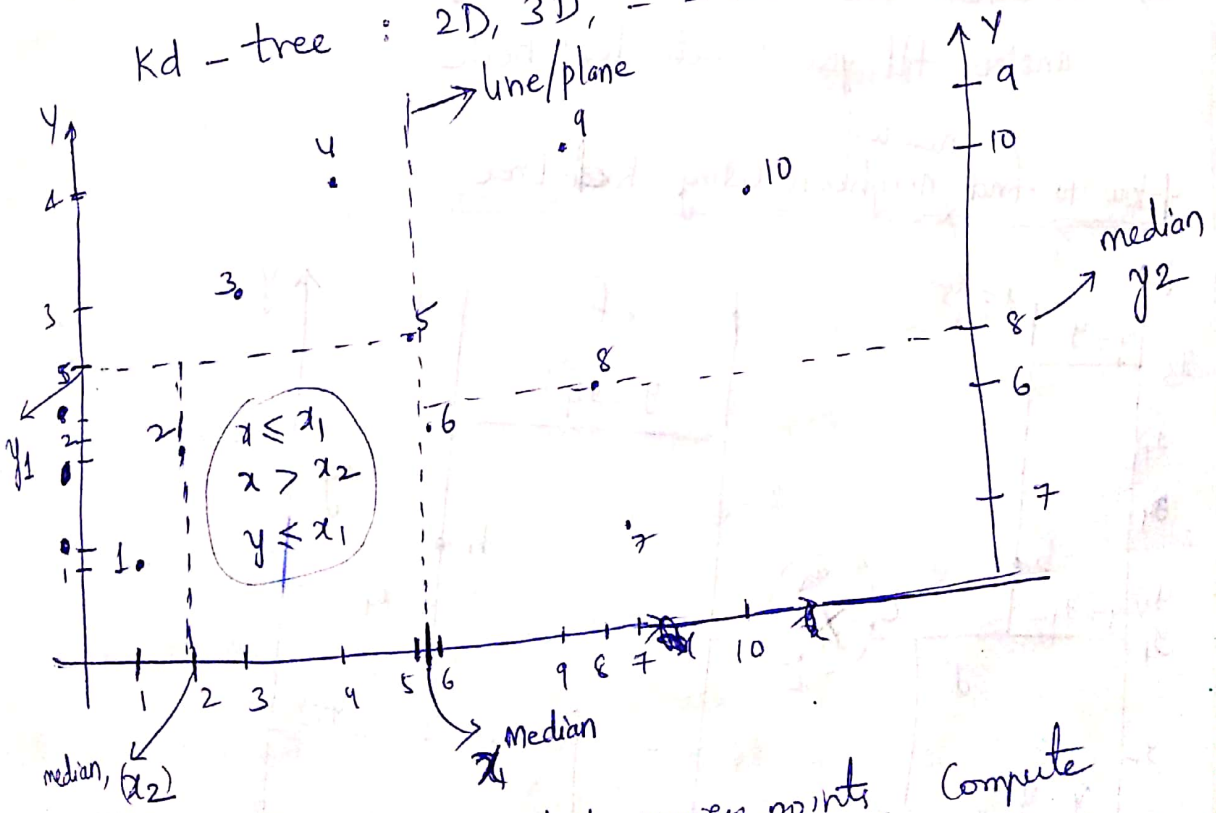
Voronoi diagram

~~Binary Search Tree~~ How to build a Kd-tree from K-NN

→ Kd-tree } Time Complexity of $O(n)$ to $O(\lg n)$

Binary Search Tree (BST) : $\underbrace{\text{Sort numbers}}_{\text{Scalars "1D"}}$ → Tree

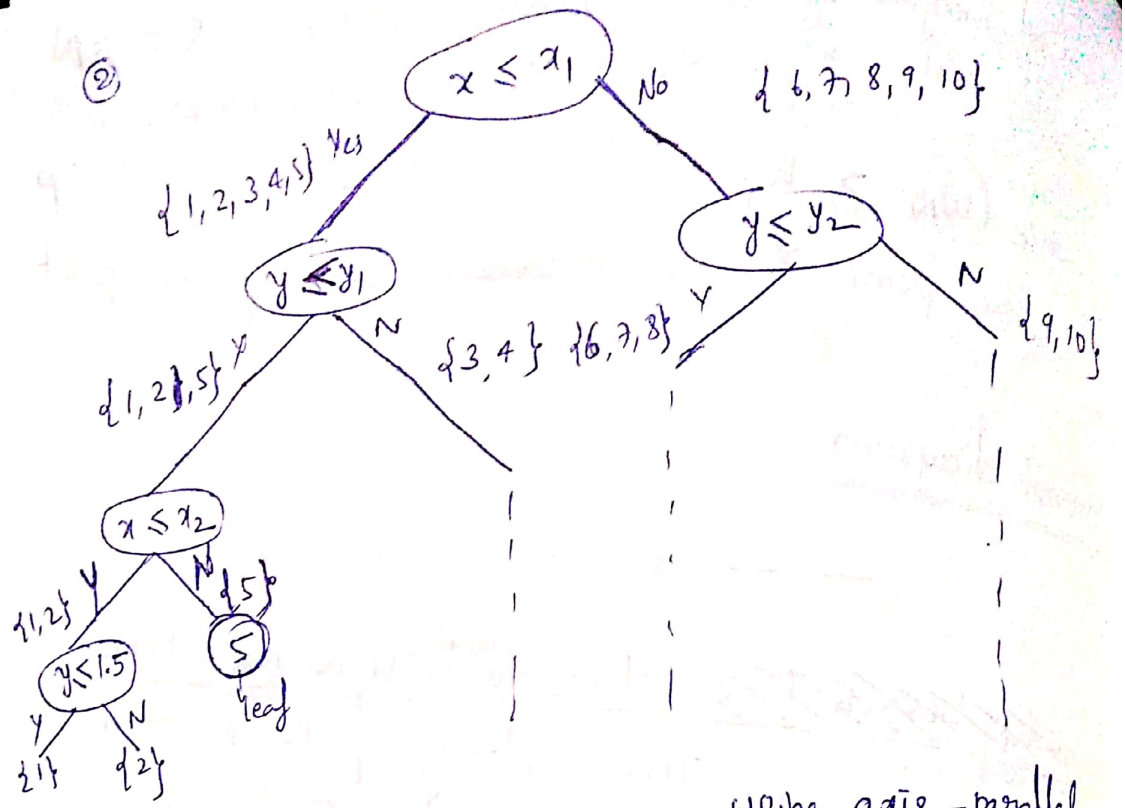
Kd-tree : 2D, 3D, - - - - - nD



① pick x-axis projected ~~proje~~ points, Compute median. (x_1)

Scanned by CamScanner

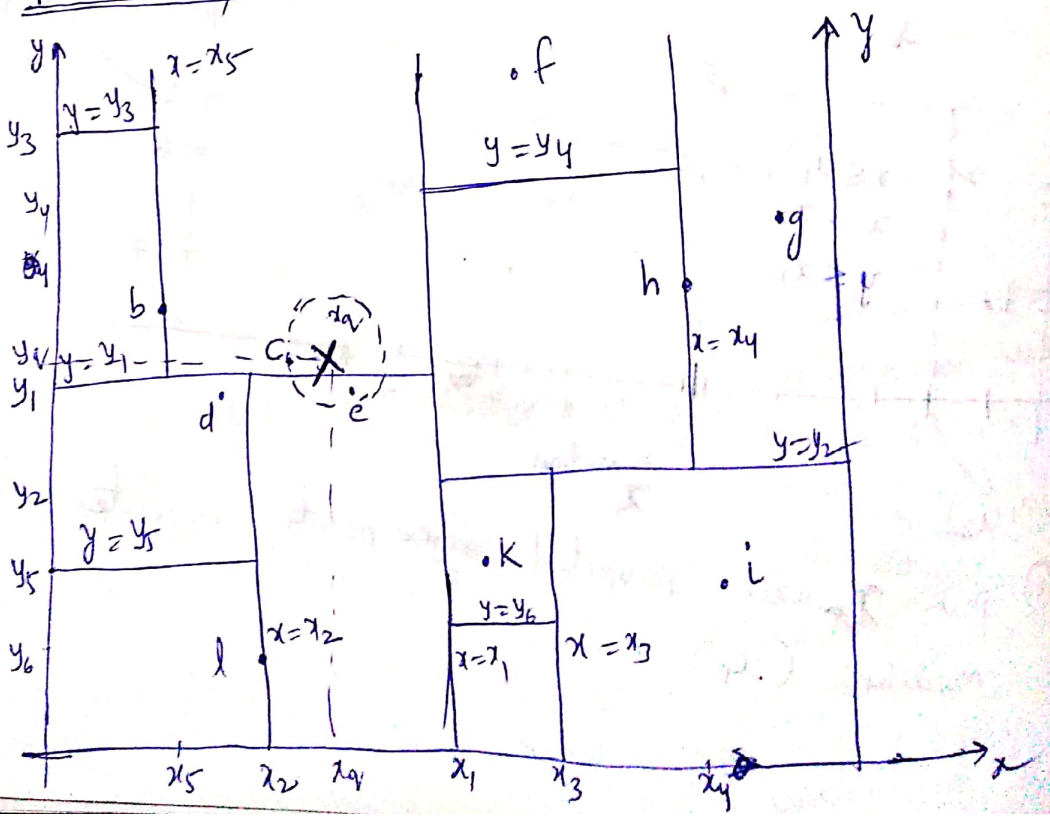
②



→ Kd-tree } Breaking the space using axis-parallel lines/planes into rectangles / cuboids / hypercuboids

→ Alternate b/w ~~axis~~ axes ie go through axis one after another till you reach leaf node.

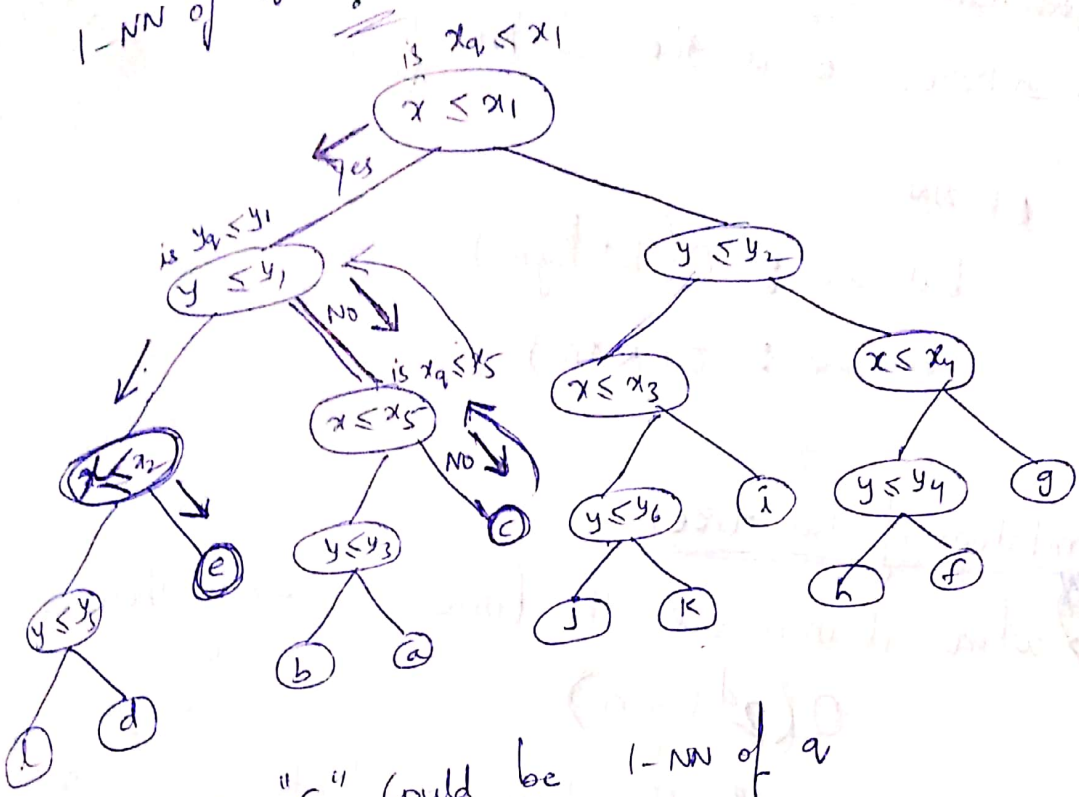
How to find nearest neighbors using Kd-tree



Scanned by CamScanner

x : query point (q) (x_q, y_q)

1-NN of q ?



∴ "c" could be 1-NN of q

~~The can~~

let distance b/w c & q be "d"

→ Draw a circle / hypersphere with radius = d and centre as query point.

→ If there is any other point inside this circle, then that point is closer than c to q

→ Circle is intersecting $y = y_1$. Track back till you get $y \le y_1$. Since the circle is intersecting $y = y_1$, there is a possibility that the nearest neighbor can be on other side of the tree ~~where~~.

Once we reach $y \le y_1$, traverse the other side from there.

→ "e" also could be nearest neighbour.

~~Draw a~~ Distance b/w q & e = d'

Draw a circle with radius = d' & centre at q

~~Ch~~ Check if new circle intersect ~~or~~ any other line ?

Scanned by CamScanner

→ If it doesn't intersect any other line other than what we have traversed from, then that is the nearest neighbour

→ So here, "e" is the 1-NN of q

→ K-NN

best case : $O(K * \log n)$

worst case : $O(K * n)$

Limitations of Kd-tree:

1) when "d" increases, the time complexity increases

$= O(2^d \log n)$

2) when d is small, time complexity $= O(\log n)$

only when data is uniformly distributed

Curse of dimensionality

Kd-tree : → It is not actually intended for ML

It is instead developed for Computer graphics (2D & 3D)

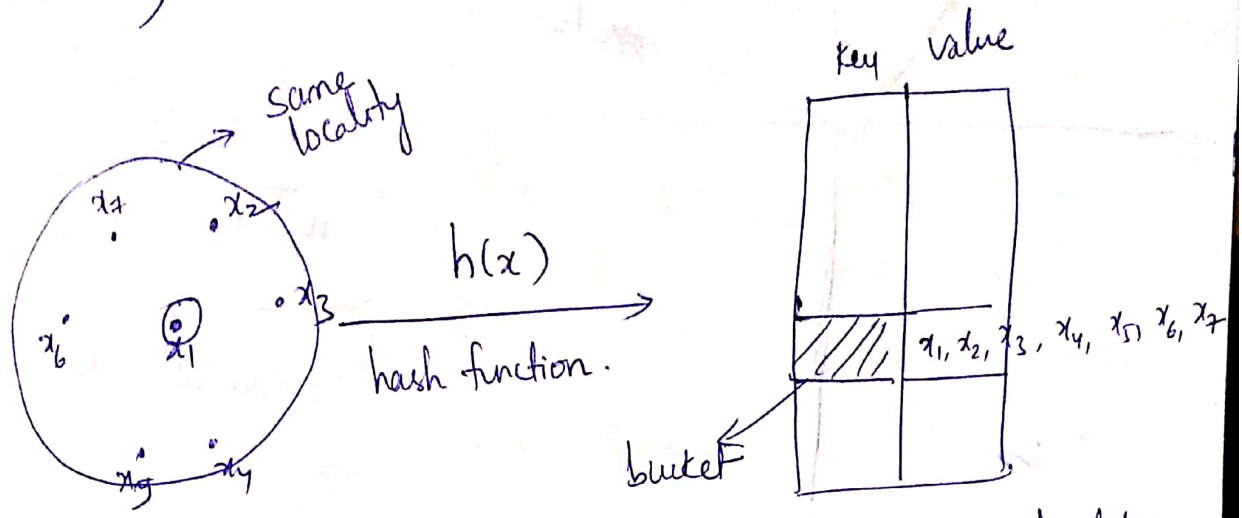
Extensions

— Variations of Kd-trees —

Scanned by CamScanner

Hashing v/s Locality Sensitive Hashing (LSH)

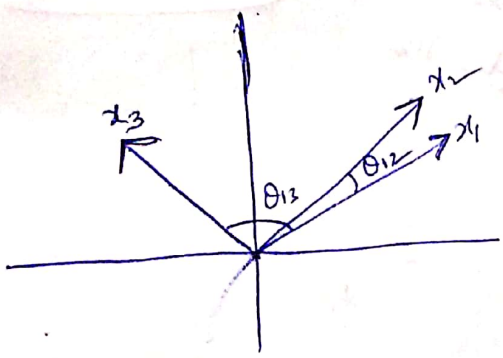
→ LSH } → powerful technique than Kd-tree to find ~~K-NN~~ K-NN



on applying hash-function, all points in the same locality should go into same bucket

→ unlike kd-tree, the LSH work well for even large dimensions.

LSH for Cosine Similarity



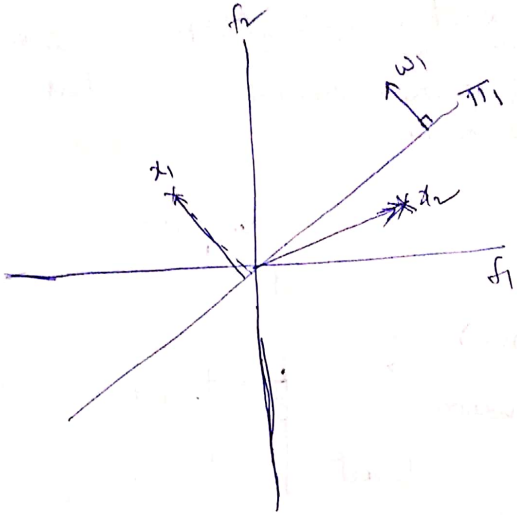
$\theta_{12} \approx 0 \Rightarrow x_1$ & x_2 are similar

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

value $= \{x_1, x_2\}$

→ LSH is (randomized) algorithm } will not always give the exact answer but gives high probabilistic answer

Scanned by CamScanner



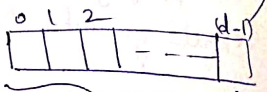
π_1 : } Random hyperplane

w_1 : } unit normal vector to π_1

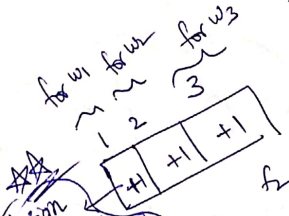
$$\boxed{\begin{aligned} w_1^T x_1 &\geq 0 \\ w_1^T x_2 &< 0 \end{aligned}}$$

→ core idea for LSH

Random hyperplane { $w^T x = 0$ (or) $w^T x + b = 0$ | w : d-dim vector (random)

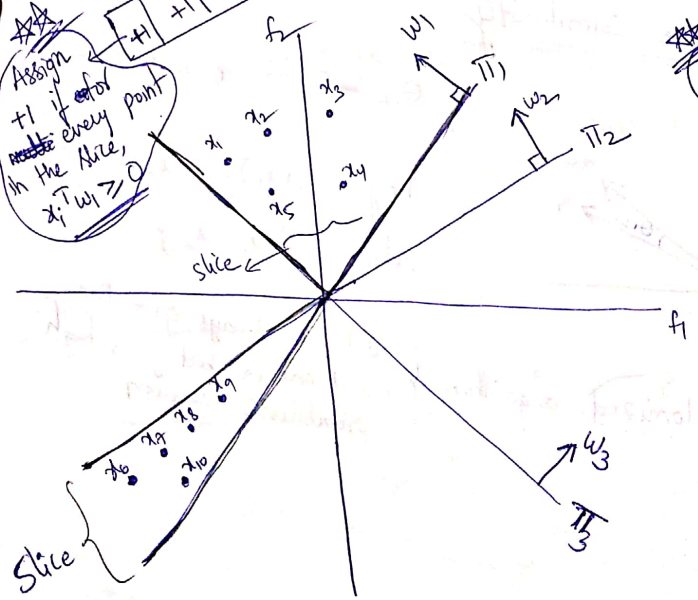


Random numbers sampled from $N(0, 1)$



Assign +1 if for every point in the slice, $x_i^T w_1 \geq 0$

m = # hyperplanes



→ Every point belongs to one of the slices

Scanned by CamScanner

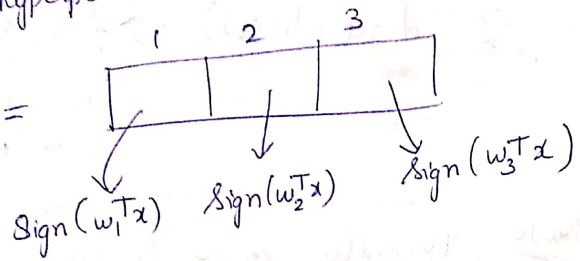
→ $x_1^T w_1 \geq 0 \Rightarrow$ Assign $+1$

$x_1^T w_2 \geq 0 \Rightarrow$ " $+1$

$x_1^T w_3 \geq 0 \Rightarrow$ " $+1$

→ For every slice, create m-dim vector where m = No. of hyperplanes

Hash function $h(x)$ =

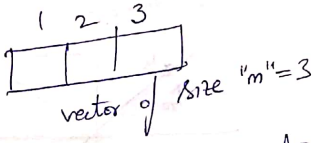


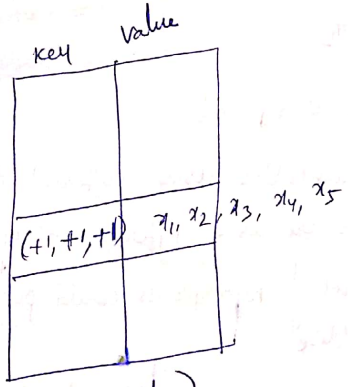
$h(x_2) = (+1, +1, +1)$

$h(x_7) = (+1, -1, -1)$

→ hashtable $\xrightarrow{\text{Compute}}$ $h(x)$

x (point)





(hashtable)

whenever two points have the same hash value i.e $h(x)$ then they are put into same bucket

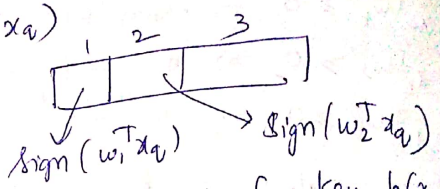
Time to compute hashtable : $O(mdn)$

Space for hashtable : $O(nd)$

Scanned by CamScanner

→ ① query point (x_q)

② $h(x_q) =$

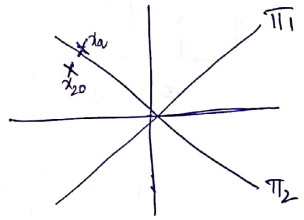


③ Get the value for hashtable for key $h(x_q)$

Ex d.get $(h(x_q)) \rightarrow \{\underbrace{x_1, x_2, x_3, x_4, x_5}_{\text{Nearest neighbours will be among these}}\}$

NOTE: $\boxed{\text{Typically } m \simeq \log(n)}$ } Good practice

→ Problem with LSH } Could miss a nearest neighbour



Improvement ~~to~~ to soln for above prob

→ Compute multiple hashtables using different sets of hyperplanes. Here we are using different set of hyperplanes because any one set of hyperplanes could put missing neighbors in same slice

→ x_q → $h_1(x_q)$ results $\{x_1, x_2, x_3, x_4\}$

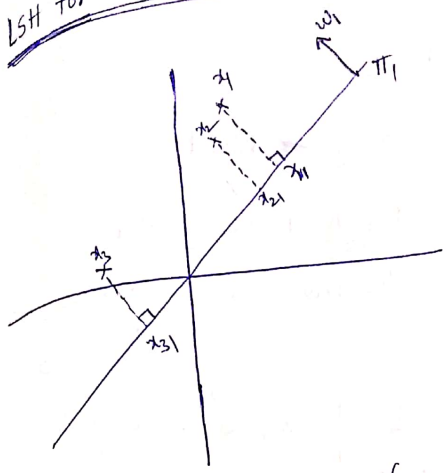
→ $h_2(x_q)$ results $\{x_1, x_2, x_3, x_{20}\}$

⋮

We take union of all these sets and find the nearest neighbors from the resulting set

Scanned by CamScanner

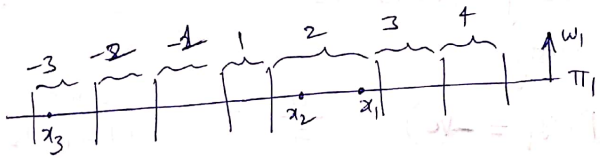
LSH for euclidean distance



x_{11} = proj of x_1 on π_1

Objective of LSH
↳ Similar/closer points should go to the same bucket

→ Breaking plane into pieces/regions (say 'a' regions)



$h(x_{11})$ =

1	2	3		m
2			- - - -	

Corresponding to plane π_1

$h(x_{31})$ =

1	2	3		m
-3			- - - -	

→

x → $h_1(x)$ → [1 | 2 | - - - | m] : Key in H_1

x → $h_2(x)$ → [1 | 2 | - - - | m] : Key in H_2

x → $h_L(x)$ → [1 | 2 | - - | m] : Key in H_L

"L" hash functions

Scanned by CamScanner

NOTE:- LSH gives just the probabilistic answer

Probabilistic class label

Let us ~~the~~ Consider the following ~~ds~~ dataset

For $K=7$ ie 7-NN

the NN $= \{ \underbrace{X, X, X, X}_{4}, \underbrace{\Delta, \Delta, \Delta}_{3} \}$

Since the diff b/w Counts of X, Δ is very small, it is unfair to say that $y_q = -ve\ (X)$

Instead, we give probabilistic ~~label~~ value to quantify the uncertainity.

$$P(y_q = -ve) = \frac{4}{7} = \frac{\#\ -ve\ points}{Total\ \#points}$$

Scanned by CamScanner