Notes on KNN :

1) Explain about Supervised Machine Learning?

Classification :

ex:- Given a new review text, determine/predict if the review is +ve or -ve

Concept : $y = f(x)$ ---> finding the function is a classification problem.

$y(q) = f(x_q)$ (x_q -> Query review)

1. Binary classification :-

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

(such that; 0 -ve, 1 +ve)

$f(x_i) = y_i$

2. Multiclass classification :-

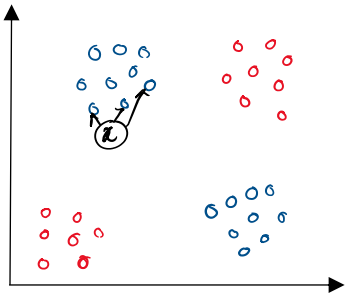
$y_i \in \{0, 1, 2, 3, 4, 5, 6\}$

3. Regression : $y_i \in \mathbb{R}$ ---- Real number

2) Explain about K-Nearest Neighbours?

KNN : k - nearest neighbours

$D = (x_i, y_i)_{i=1}^{n}$; given x_q find $y_q \in \{0, 1\}$



① find k nearest point to x_q in D

Let k = 3 $\{ x_1, x_2, x_3 \}$ 3 nearest neighbours to x_q

$x_1 \downarrow y_1$, $x_2 \downarrow y_2$, $x_3 \downarrow y_3$

Notes on KNN :

② majority vote :-

$\{ y_1, y_2, y_3 \}$ ⟶ majority or voting

+ve +ve −ve

∴ y_q for given x_q will be +ve ..... (2/3 votes)

Note: k - should be odd number to get majority vote.

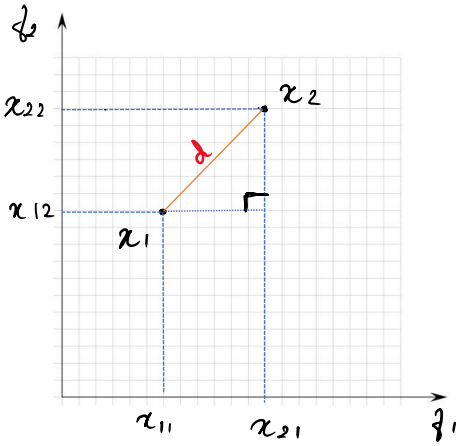
3) Failure cases of KNN?

failure cases :

1. If there is randomness or noise in the data.
2. If (x_q) or test value are far away from the cluster.

4) Define Distance measures: Euclidean(L2) , Manhattan(L1), Minkowski, Hamming?

Distance measure :-



features	f_1	f_2
x_1	x_{11}	x_{12}
x_2	x_{21}	x_{22}

① Euclidean Distance :-

d = length of shortest line from x_1 to x_2

Euclidean Distance = $d = \sqrt{(x_{21} - x_{11})^2 + (x_{22} - x_{12})^2}$

$= \| x_1 - x_2 \|$ $\quad (c^2 = a^2 + b^2)$

$x_1 \in \mathbb{R}^d$; $x_2 \in \mathbb{R}^d$

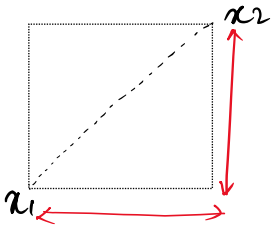
euclidean ⟶ $\| x_1 - x_2 \|$

$= \left(\sum_{i=1}^{d} (x_{1(i)} - x_{2(i)})^2 \right)^{1/2}$

Note : ---> $\| x_1 - x_2 \|_2$ ---- L2 norm

Notes on KNN :

② Manhattan Distance :-



$$\sum_{i=1}^{d} |x_{1i} - x_{2i}|$$ (absolute)

from origin

$$\|x_1\|_1 = \sum_{i=1}^{d} |x_{1i}|$$ (absolute)

L_1 norm = $\|x_1 - x_2\|_1$

③ Minkowski distance :- (L_p -norm)

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

If $p = 2$ minkowski distance = euclidean distance

$p = 1$ minkowski distance = Manhattan distance

Note: * distance are for 2 points

* norms are for vector

ie:- Minkowski → Lp-norm of any vector $(x_1 - x_2)$

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

④ Hamming distance :-

x_1, x_2 → boolean vector ⟶ ex : Boolean BOW

Hamming distance (x_1, x_2) = # locations / dimensions where binary vector differs

ex :- $x_1 = [0, 1, 0, 1, 0, 1, 0 \ldots]$

$x_2 = [1, 0, 0, 1, 0, 1, 1 \ldots]$

Hamming distance $(x_1, x_2) = 3$

ex : Gene sequence : (AGTC) ⟶ long sequences

$x_1 = A\,G\,A\,T\,C\,T\,C\,A\,G$

$x_2 = A\,A\,G\,T\,C\,T\,C\,G\,A$

Hamming distance = 4

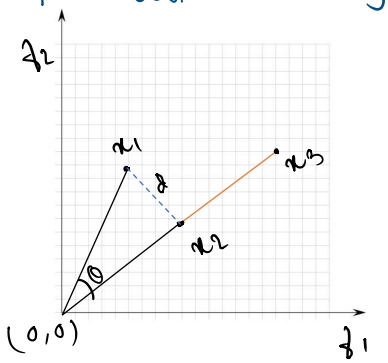
Notes on KNN :

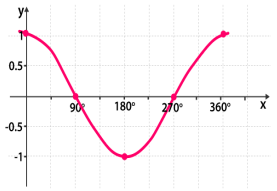
5) What is Cosine Distance & Cosine Similarity?

as Cosine Similarity ↑ Cosine distance ↓

as Cosine Similarity ↓ Cosine distance ↑

1 − Cosine similarity (x_1, x_2) = cosine distance (x_1, x_2)





d = euclidean distance

cos-sim (x_1, x_2) = cos θ

where θ = angle between x_1 & x_2

now lets take x_3 :

cos-sim (x_1, x_2) = cos θ

cos-sim (x_2, x_3) = 1 { because θ = 0, cos θ = cos 0 = 1

cos-dist = 1 − cos-sim

= 1 − 1

∴ cosine dist = 0 --- for x_2 & x_3

Cosine similarity and cosine distance uses angle to measure.

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

If x_1 & x_2 are unit vector then cos θ = $x_1 \cdot x_2$

* Relationship between Cosine Similarity and Euclidean Distance?

∵ cosine dist = (1 − cos θ)

$$d^2 = 2 * (1 - \cos\theta)$$

Square of Euclidean distance = 2 × cosine distance

$$d^2_{(x_1, x_2)} = 2 * \text{cos-dist}(x_1, x_2)$$

Notes on KNN :

6) How to measure the effectiveness of k-NN?

ex:

problem : Given a new review (x_q), what is the polarity (+ve/-ve)

We have = D_n $\because$ $x_i \longrightarrow y_i$

D_n < D_{train} (70%), D_{test} (30%) } Randomly

① for each point in D_{test}

$x_q \longrightarrow$ point

use D_{train} + KNN to detect y_{pt}

② If $y_q = y_{pt}$

count += 1

where, count = # pts which D_{train} + KNN gave correct class label

③ Accuracy

$$\text{Accuracy} = \frac{\text{count}}{n_{test}} \quad \because n_{test} = \#\text{ of pts in } D_{test}$$

$$0 \leq \text{Accuracy} \leq 1$$

#1 Test / Evaluation time & Space complexity

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

Output :- y_q

If k is small

Time & Space complexity = $O(nd)$ $\because$ n = no. of pain in Dataset, d = dimensions

note it is roughly equals to $O(nd)$, if d is small - then $O(n)$

Notes on KNN :

7) Limitations of KNN?

1. Space - time Complexity :

for a example where k is small

Time : O(nd)

Space : O(nd)

Expectation :- x_q fast $\longrightarrow$ y_q { low latency systems }

k-NN $\longrightarrow$ Simple implementation $\longrightarrow$ O(nd) ; O(nd)

space time

ex: $n \simeq 364k$ ↓ ↓

$d \simeq 100k$ 36 GB ram 36 Billion Computation

which is not acceptable

(2) Decision Surface for K-NN changes as k changes

'k' $\longrightarrow$ hyperparameter

* Curves / lines that separates +ve & -ve points are called decision surfaces.

In K-NN ; The Smoothness of decision Surface increases as K - increases

case 1: If k = 1

There is high chance that based on the nearest neighbour the pt will be missclassified

case 2: If k = 5

Majority rule will play a role in the classification

case 3: If k = n or k is very large

If k is large then algorithm will be biased towards majority class.

eg n_1 = +ve n_2 = -ve where $n_1 > n_2$ $n = 1000 = k$

600 400

Notes on KNN :

8) How to handle Overfitting and Underfitting in KNN?

Overfit	wellfit	underfit
$k = 1$	$k = 5$	$k = n$
• no mistake	• some mistakes	• every point belongs to majority class
• non smooth	• smooth	• Doesn't cares
• prone to noise	• less prone to noise	

Overfitting: function or Algorithm tries to overtrain to not make any mistake on training data (includes noise or outlier)

underfitting: Algorithm or function underworks on training data and will not care about mistakes.

9) Need for Cross validation?

Objective :- Future unseen point $x_q \xrightarrow[\text{accuracy}]{} y_q$

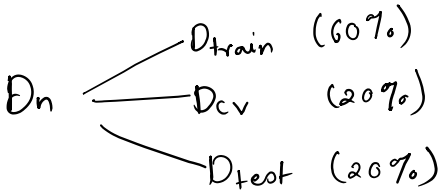
Generalization :- if it performs better on unseen data.

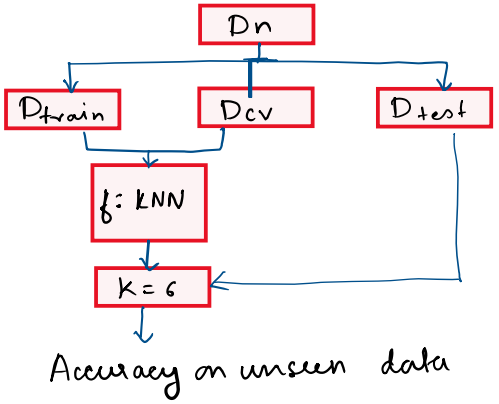
Easier :-

1. we split D_n (total dataset) in two parts
   - (a) D_{test}
   - (b) D_{train}
2. we used D_{train} to train the data
3. we used D_{test} to test the data, based on the accuracy we decided the value of $K = 6$ or hyperparameter.

But :- Our model has seen both train and test data, so we are not sure that our function/Model will generalize better on future unseen data

Cross - validation :-

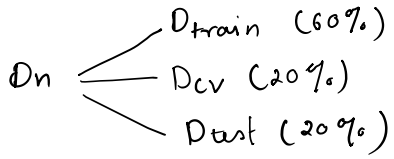




Notes on KNN :

10) What is K-fold cross validation?

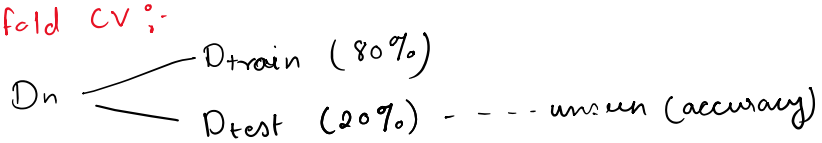
As seen above :

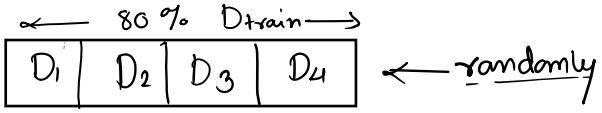


Problem : Only 60% of the data is used for training

* More training data = the better model

k-fold CV :-

① 

② 

③

Fold	Train	CV	accuracy
K1	D1, D2, D3	D4	A4
K2	D1, D2, D4	D3	A3
K3	D1, D3, D4	D2	A2
K4	D2, D3, D4	D1	A1

k = 4 fold validation

avg = A_k

* Accuracy (Avg accuracy using 4 fold CV) → Dtest → Accuracy of 3NN on Dtest (3NN)

* k in k-fold ≠ k in KNN

* Time it takes to compute optimal k in KNN increases by k-fold times if we use k-fold CV

Advantages :-

① Dtrain & Dcv donot overlap perfectly when randomly sampled.

② If there are many +ve points from Dtrain in a region, then there is high-likely to find many negative (-ve) points from Dcv in that region

③ If there are very few points (+ve) in a region from Dtrain, then it is very unlikely to find (+ve) points from Dcv in that region.

Notes on KNN :

Question :- How can we be really sure that we are not underfitting or overfitting ?

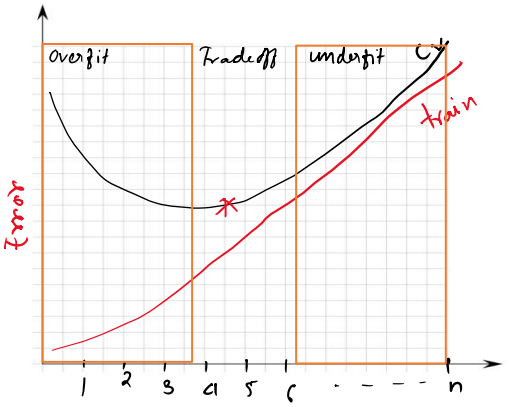
* Accuracy = $\frac{\#\text{ correctly classified pts}}{\text{Total no. of points}}$

* Error = 1 − Accuracy

* Now we can calculate Train error ; validation error
  D_{train} , D_{CV} , D_{test}

If Train Error ↑, CV Error ↑ } underfit

If Train Error ↓, CV Error ↑ } overfit



11) What is Time based splitting?

Whenever time is available and if thing/behaviour and data changes overtime, then we should use TBS (time based splitting)

TBS is alternate to Random Splitting :-

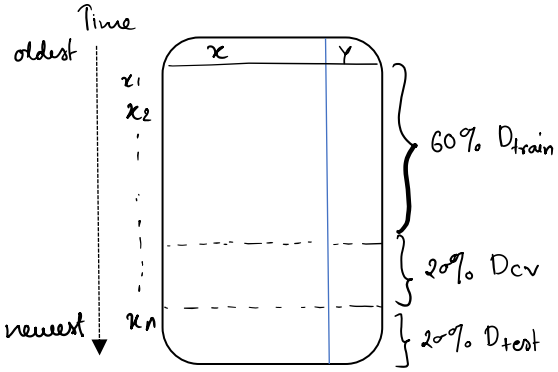
① Sort Dn in ascending order of time

② Split data in D_{Train}, D_{CV}, D_{test} based on time

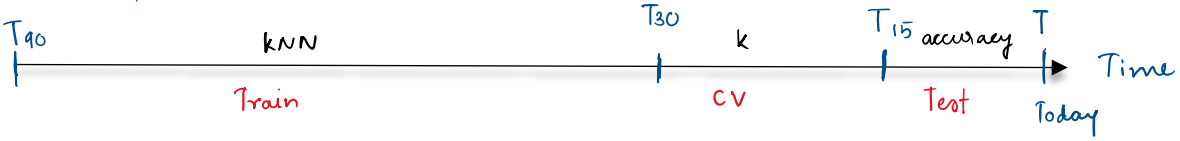
D_{train} ⟶ oldest

D_{CV} ⟶ old

D_{test} ⟶ newest



Example :



Notes on KNN :

12) Explain k-NN for regression?

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

for regression :-

1.] given x_q, find K- nearest neighbour

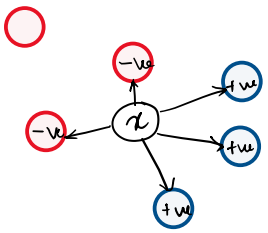
(x_1, y_1), (x_2, y_2), (x_3, y_3) - - - - (x_k, y_k)

2.] $y_q \leftarrow y_1, y_2, y_3 - - - y_k$

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

y_q = median $(y_i)_{i=1}^{k}$ $\leftarrow$ less prone to outlier

13) What is Weighted k-NN?



by majority vote

3 +ve , 2 -ve

y_q = +ve

by weighted K-NN

w_i (+ve) = 1.75

w_i (-ve) = 1.5

y_q = -ve

As seen in the example it is unfair to decide class just based upon majority vote

∴ weighted KNN

weighted KNN gives more weight to the nearest point and less weight to farthest neighbour

$w_i = \frac{1}{d_i}$ where d_i is distance of the neighbour

$d_i \uparrow$; $w_i \downarrow$

$d_i \downarrow$; $w_i \uparrow$

neighbour	class	d_i distance	w_i weight	
x_1	-ve	0.1	10	} 15
x_2	-ve	0.2	5	
x_3	+ve	1.0	1	
x_4	+ve	1.0	0.5	} 1.75
x_5	+ve	4.0	0.25	

Notes on KNN :

14) How to build a kd-tree?

why ?

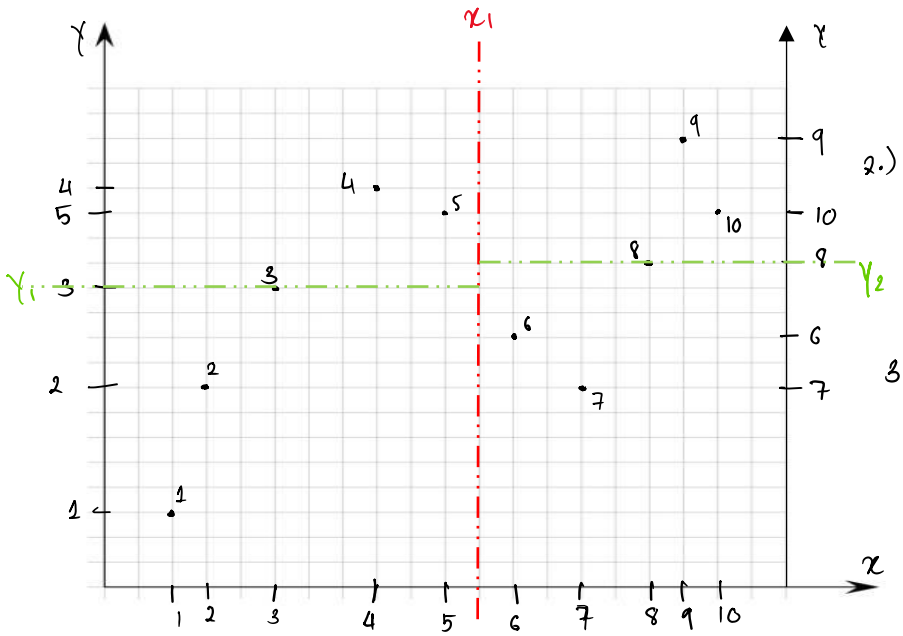
As we that time and space complexity of a simplified KNN is roughly O(nd)

Can we somehow bring down time and space complexity from O(nd) to O(log n) ?

using concept of Binary Search Tree (BST) algorithm

* such a model so called as a kd-tree

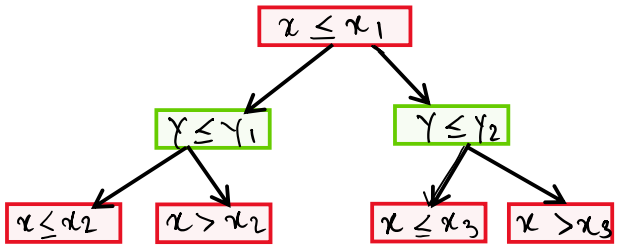
kd - tree :-



1.) pick x-axis, project points on x-axis, compute median and split using median

2.) pick y-axis, project points on y-axis, compute median and split using median

3.) Continue splitting alternate axis till we reach leaf node



Kd-tree :- breaking up spaces using axis-parallel lines/planes into rectangle, cube or cuboid.

Notes on KNN :

Limitations of Kd-tree :

① when point is near boundary we need to look at the adjoining region

2D = 4 regions

3D = 8 regions

$d = 2^d$

② when d is not small

$d = 10$; $2^d = 1024$

$d = 20$; $2^d \approx$ million

$d \uparrow$: worst case # adjoining cell $2^d \uparrow$

$n =$ 1 million $\longrightarrow 2^d \approx n$

no longer $(\lg n)$

③ when d is not small $\{2, 3, 4, 5\}$

Time complexity : $O(\lg n)$ 1-NN

$O(2^d \lg n)$ ← as d increases

Disadvantages :-

① when d is 10, 11, 12 - - - -

The time complexity increases dramatically

② $O(\lg n)$ ← when d is small and data is uniformly distributed

But in real world data is not uniformly distributed

uniform = $O(\lg n)$ →

simple implementation = $O(n)$ ←

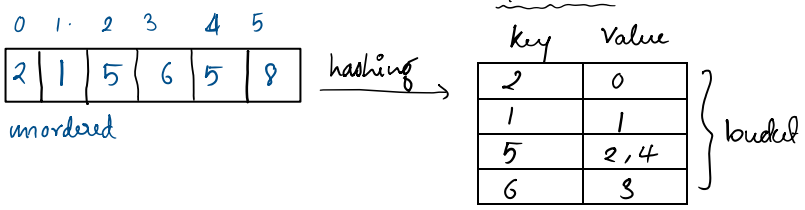
Extensions of Kd-tree :-

1. implicit
2. min/max Kd tree
3. Relaxed Kd tree
4. Ntropy
5. Quad tree
6. Ball tree
7. Oct-tree
8. R-tree
9. vantage point tree

Notes on KNN :

15) What is Locality sensitive Hashing (LSH)? (Hashing vs LSH?

Hashing :-



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

* Hashing = O(1) ; k-NN (sequential) = O(n)

Locality Sensitivity Hashing :-

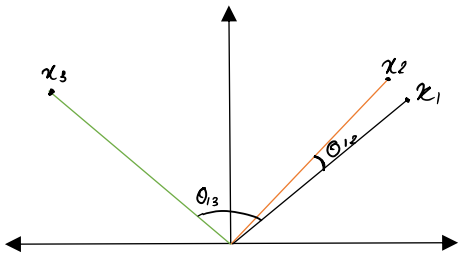
(1.) it will create a hashfunction and will store all the nearest points in just one bucket.

(2) Given the new query point (xq) it will find nearest point and hashfunction will lead us to the bucket with all the nearest value.

16) LSH for cosine similarity?

LSH for cosine similarity :-

Cosine similarity = Angular similarity



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

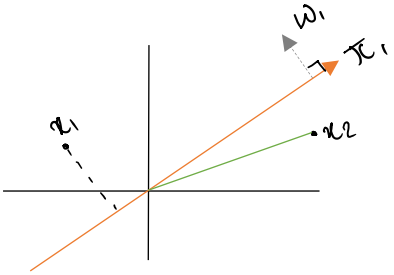
θ_{13} is large $\therefore x_1$ & x_3 are very dissimilar

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

key	value
$h(x_1)$	x_1, x_2

LSH (Randomized Algorithm)

it says → "I will not always give correct answer but I will always give correct answer with high probability."



$w^T x_1 \geq 0$ (+ve)

$w^T x_2 \leq 0$ (-ve)

$w_1 \rightarrow$ unit normal vector to plane π_1

1. Random Hyperplane

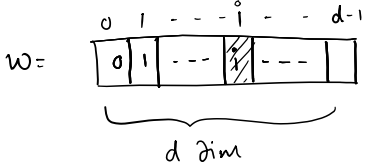
$w^T x = 0$

w : normal to the plane

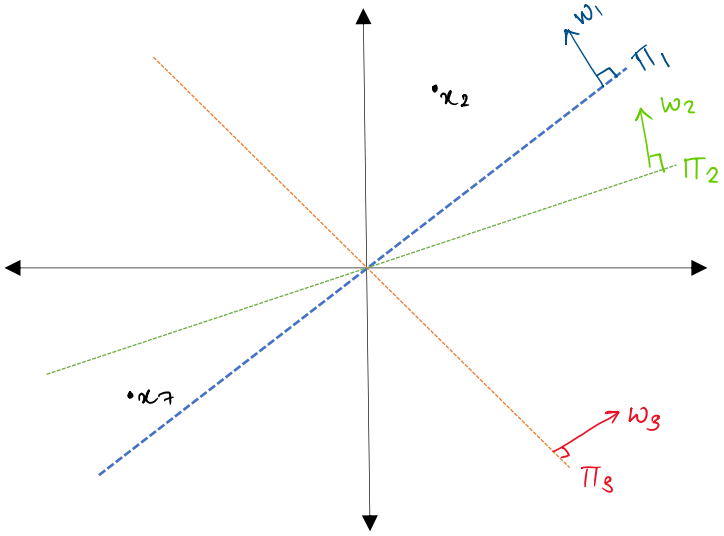
$\because w^T x + b = 0$ (so as passing through origin)

Notes on KNN :

Random numbers sampled from normal distribution ; where mean = 0 & variance = 1 $(N(0,1))$

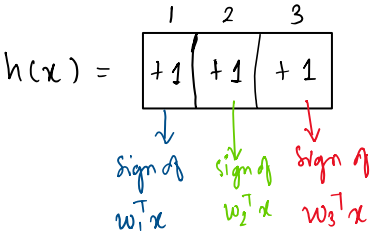


>> w = numpy.random.normal(0, 1, d)



m = # hyperplanes

∀ slice ⟶ m-dim vector

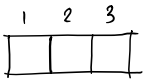


ex: $h(x_2) = +1, +1, +1$

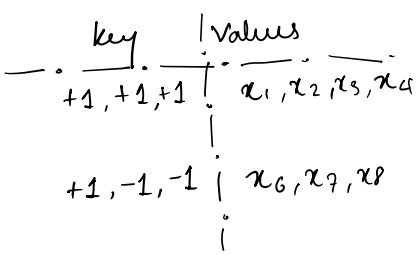
$h(x_7) = +1, -1, -1$

Hash table :-

$x \rightarrow h(x)$



vector of size 'm' (here 3)



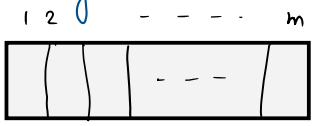
Time and Space LSH :-

Space : O(nd) - - → every point has to be stored

Time : O(mdn) ⟶ time to compute hash table

Time Complexity of Quering

x_q : $h(x_q)$



O(md)

x_q ⟶ n' element in the bucket/slice O(md + n'd)

If n' is small then $n' \approx m$ O(md)

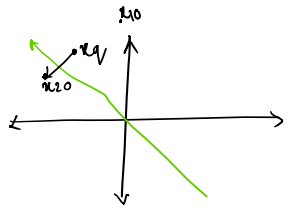
Typically $m \approx \log(n)$

Time :- $O(d \times \log(n))$

Notes on KNN :

problems with LSH :

You could miss the nearest neighbour in a cosine-similarity as your measure



x_q and x_2 will go in different bucket of hashtable and we will miss.

Fix : Create multiple sets of hyperplane and repeat multiple times

$x_q \rightarrow h_1(x_q) \rightarrow key \rightarrow$ Hashtable 1 $\dashrightarrow \{x_1, x_2, x_3, x_4\}$

$\rightarrow h_2(x_q) \rightarrow key \rightarrow$ Hashtable 2 $\dashrightarrow \{x_1, x_2, x_3, x_4\}$

$\rightarrow h_l(x_q) \rightarrow key \rightarrow$ Hashtable l $\dashrightarrow \{x_4, x_8, x_q\}$

Set of union of all points

$x_q \rightarrow \{x_1, x_2, x_3, x_4, x_8, x_q\}$

Now we can compute cosine similarity for nearest neighbours)

Time Complexity : $O(mdL)$ $\quad m = O(\log n)$

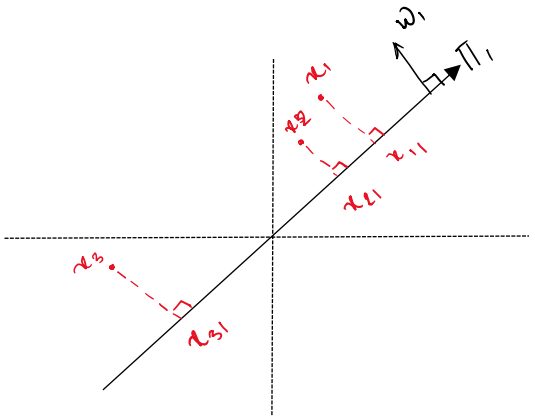
As no. hyperplane increases ; slices ↑ ; # of points per slice ↓

∴ we should choose 'm' wisely.

17) LSH for Euclidean distance?

$x_{11} \rightarrow$ projection of x_1 on Π_1

$x_{21} \rightarrow$ projection of x_2 on Π_1



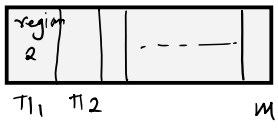
$d(x_1, x_2) << (x_1, x_3)$

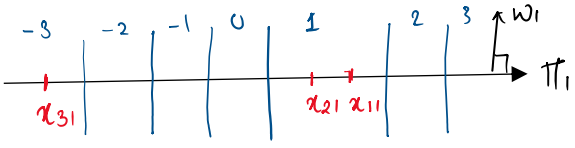
LSH → similar or closer points should go to the same bucket

Breaking plain into pieces/regions

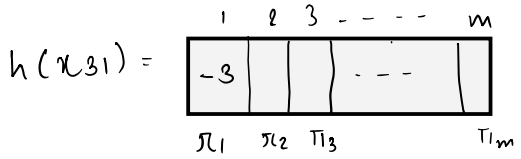
hyperplanes = m

$h(x_{11}) =$

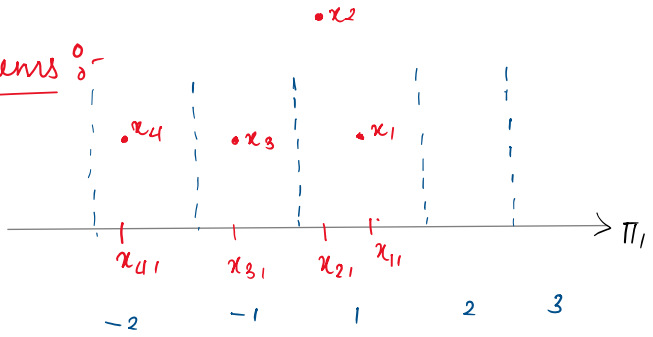




Notes on KNN :



Problems :-



x_{11} & x_{21} are in same region (1)

but x_1 is very far away from x_2

$d(x_3, x_4) << d(x_2, x_1)$

Solution : create multiple and random hyperplane

Note : LSH is not a perfect but probabilistic / randomized Algorithm.

KNN : Probabilistic Class Labels

example: 2 class classification

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

x_q : 4 -ve pts ; 3 +ve pts $(y_q = -ve)$ ← majority vote

x_q' : 7 (-ve pts) ; $(y_q' = -ve)$

Quantify the uncertainty

$$P(y_q = -ve) = 4/7 = \frac{\#\text{ -ve pts}}{\#\text{ total pts}}$$

$$P(y_q' = -ve) = 7/7 = 100\%$$

Similarity

$$P(y_q = +ve) = 3/7$$

$$P(y_q = +ve) = 0/7$$

Probabilistic class label

$$x_q \longrightarrow y_q \longrightarrow P(y_q = +ve), \; P(y_q = -ve)$$

} certainty