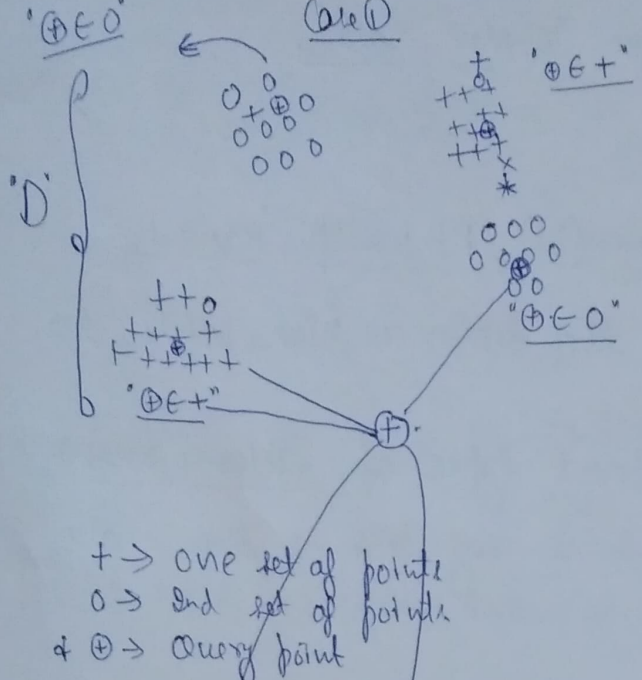
→ "Failure cases of K-NN"

Let's now understand where KNN fails :→ Let's understand it with the help of examples.

Let's take two cases & lets assume "K=3"

Case ①



Now let's assume our query point is away from the rest of the points.

This point (Q-point) is far away from the points in our dataset.

If we take 3 nearest neighbours for this point, we might conclude that that is a point belonging to "+" class

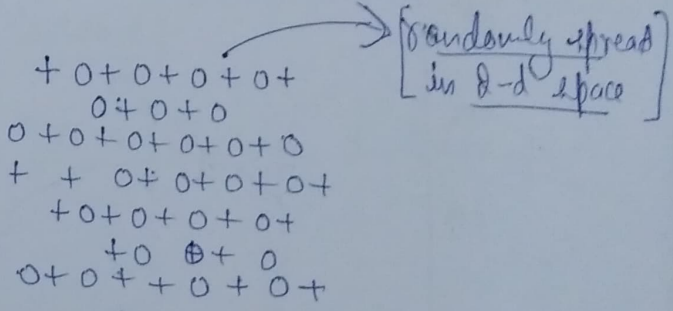
Since the point is really far away from the rest of the points. Does it really make sense to call the query point as either "+" or "O"

It looks little absurd or untrue because we don't have any point really close to our 'query point' "⊕". All the points are very far away from our query point.

So, when we have our query point lying very far away from our dataset, we are not very sure of its class label.

This is one case where 'K-NN' may not work as per our expectations.

Case ② :→ "Jumbled data"



If we take a ["query point"] & apply "K-NN" on it, we will get a class label that is for sure, but most likely that is going to be a wrong prediction, because data is highly jumbled up or intermixed

"P.T.O"

So, if our data is completely random (randomly distributed), there is no useful information at all in it.

KNN miserably fails here, Actually most of the algorithms miserably fail here.

. ——— .

There are the two simplest cases where "K-NN" fails.

→ "Distance Measures"

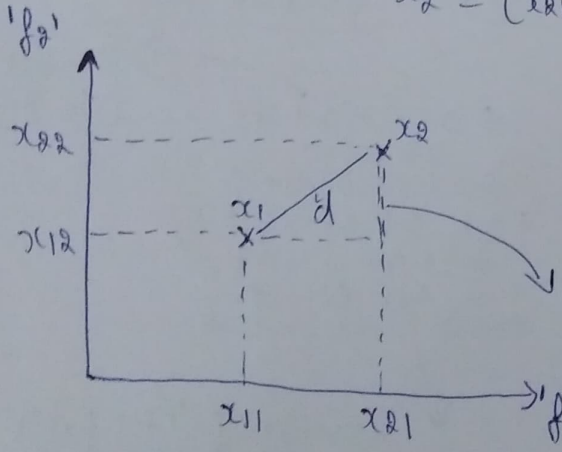
Euclidean distance (L2), Manhattan distance (L1), "Minpowerpi", "Hamming"

In earlier topic we have measured the distance b/w points to get the nearest neighbours.

Now let's look at some different types of distance measures.

Let's have two points "x_1" & "x_2" & let's say both of them are "two-dimensional"

$$x_1 = (x_{11}, x_{12})$$
$$x_2 = (x_{21}, x_{22})$$



'f1' 'f2'
$x_1 = (x_{11}, x_{12})$
$x_2 = (x_{21}, x_{22})$

distance could be measured using simple pythagoras theorem

Simplest understanding of distance is basically the shortest (length of shortest) distance

d = length of the shortest line from "x_1 to x_2"

That's what we typically say when we call distance.

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

↳ length

"Euclidean distance"

↳ It's the distance of shortest distance

Now let's assume that x_1 & x_2 belongs to "d-dimensional" space

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

then distance b/w x_1 & x_2 is the length of the vector

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

↓

"Euclidean distance" :: It is represented as $\|x_1 - x_2\|_2$.

in subscript we put "2"

This is also called as "L_2 norm of a vector"

So, if we have a vector let's say 'x_1' then L_2 norm of the vector is $\|x_1\|_2$ = Euclidean distance of x_1 from origin,

because 'x_1' is a point & a point can be called a vector

$\therefore \|x_1\|_2$ = Euclidean distance of x_1 from origin $= \left(\sum_{i=1}^{d} x_{1i}^2 \right)^{1/2}$

---

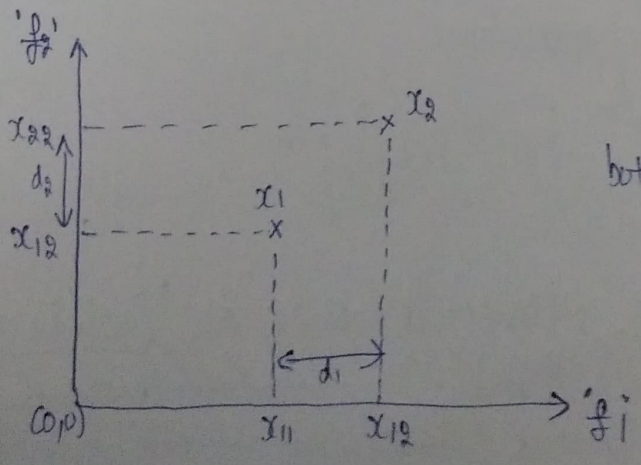
(2) Distance Metric (Manhattan distance)

Manhattan distance b/w two points "x_1 & x_2" in d-dimension

$$\left\{ \sum_{i=1}^{d} |x_{1i} - x_{2i}| \right\}$$

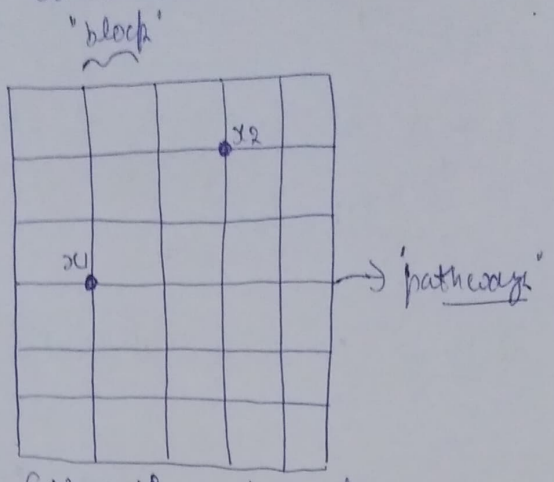
$|\ \ |$ → "Absolute value"

Geometrically, what it means is as shown below :-



both these d_1 & d_2 have to be added to get the manhattan distance b/w "x_1 & x_2".

Sum of these two distances is considered as Manhattan distance.



Now if we have to go from 'x_1' to 'x_2'
we have to cross 2 blocks in "horizontal direction"
& 2 blocks in the "vertical direction" to reach "x_2" from x_1

We can't go diagonally because there are buildings there.

Manhattan distance is, in each direction we take the absolute value.

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

$||x_1 - x_2||_1$ → 'L1 norm' (also called)

Now there is a generalization of "L1-norm" & "L2-norm" called "Lp Norm" (p could be any no. here)

Now for Lp Norm the corresponding distance is called ["Minkowski distance"]

Let's first write the formula, the "minkowski distance" between any two points is written as

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

if $P=2$ → Minkowski distance is same as "Euclidean-distance."
if $P=1$ → " " ——— "Manhattan-distance"

$$\|x_1\|_p = \left(\sum_{i=1}^{d} |x_{1i}|^p\right)^{1/p}$$ → "Lp" norm of a vector "x_1"

for Lp norms 'p cannot be zero" 1/0 is not defined

Note:- "Distances" are always b/w two points & "norms" are always b/w "two vectors". (or a vector)

---

(4) → "Hamming distance" :- It is mostly used in "text-processing". Or when we have a binary vector or a boolean vector.

Imagine we have two points ["x_1"] & ["x_2"] & both of these points are "Boolean vectors".

x_1, x_2 → "boolean vectors" ↷ used for converting text into vector

"x_1" = [0, 1, 1, 0, 1, 0, 0 ; 0 ---]

"x_2" = [1, 0, 1, 0, 1, 0, 1 ---]

Hamming distance b/w two vectors ("x_1" & "x_2") is the number of locations /dimension where the binary vector differ.

"x_1" = [0, 1, 1, 0, 1, 0, 0]

"x_2" = [1, 0, 1, 0, 1, 0, 1]

"Hamming distance" b/w ["x_1" & "x_2" is "3"]

↓ bcoz in three locations they are different

"Hamming distance" is also used when we have "strings"

x_1 = abcadefghik | Hamm dist (x_1, x_2) = "4"

x_2 = acbadegfhik

In Gene sequencing also we use Hamming distance [x_1 = AAGTCTCAG, x_2 = AGATCTCGA]

→ "Cosine similarity & cosine-distance" :-

There is one more distance measure called ["cosine-distance"] & the corresponding similarity is ["cosine-similarity"].

first let's understand the relationship between "similarity & distance".

"x1" & "x2"

"similarity" "distance"

As the distance b/w two points 'x1' & 'x2' increases, this means they are less similar.

↓ (x_1, x_2) ↑ (x_1, x_2)

& As the similarity b/w two points increases, the distance between them decreases.

↑ (x_1, x_2) ↓ (x_1, x_2)

There is the opposite relationship b/w "similarity" & "distance"

Let's first write down the relationship b/w ["cosine similarity"] & ["cosine-distance"].

$$1 - \text{Cos-sim}(x_1, x_2) = \text{Cos-distance}(x_1, x_2)$$

Let's say Cos_similarity lies b/w [-1, 1]

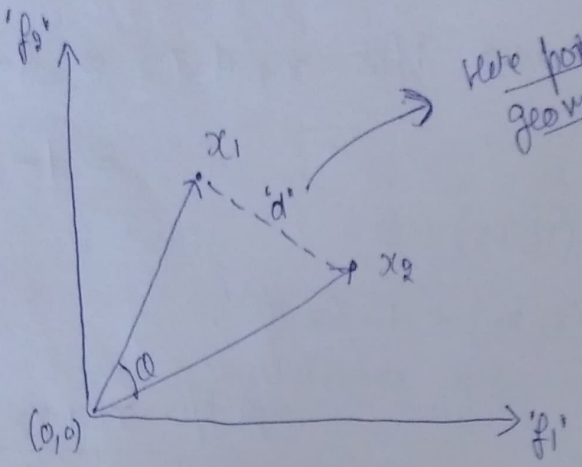
{ Let's say if two points are very similar, Cos_sim = "+1"
& if they are dissimilar, Cos_sim = "-1" }

'+1' means they are perfectly similar.
&
'-1' means they are perfectly dissimilar.

PTO

From this background, now let's understand what is "Cosine-similarity".

Let's say we have two points $(x_1 \& x_2)$ in 2-d space as shown below:



Here points are geometrically farther away.

"d is euclidean distance"

"Cosine distance" says that instead of taking "geometric distance", what if we take angle "θ" as shown above.

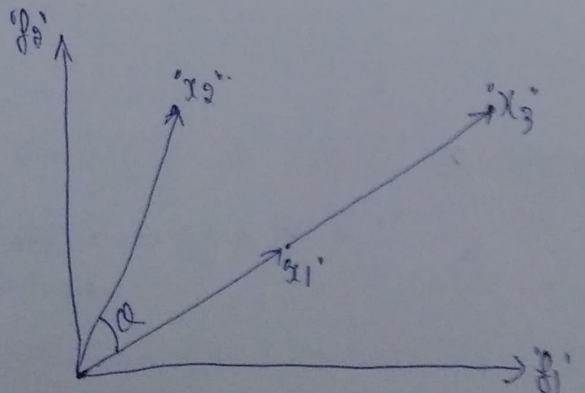
Cos_similarity is ['Cos of angle 'θ']

$$\text{Cos\_sim}_{(x_1, x_2)} = \cos\theta$$

"θ: angle b/w x_1 & x_2".

Here points are angularly farther away.

Let's understand the diff b/w Euclidean distance & Cosine similarity

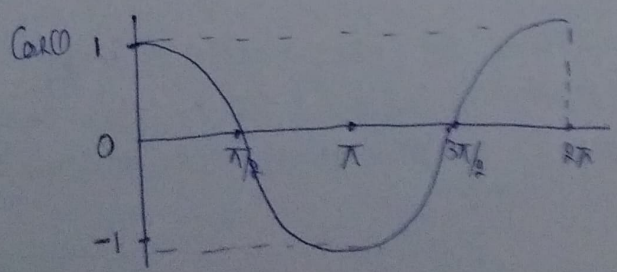


'x1' & 'x3' are in the same direction

$\text{Cos\_sim}(x_1, x_3) = \cos\theta$

$\text{Cos\_sim}(x_1, x_3) = 1$

as $\theta_{x_1, x_3} = 0°$



for "x_1 & x_3" since the angle b/w them is $0°$ ∴

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

but the Cosine distance b/w x_1 & x_3 $= 1 - \text{Cos\_sim}(x_1, x_3)$

$$= 1 - 1 = 0$$

∴ $\text{Cos\_dis}(x_1, x_3) = 0$

while $\text{Cos\_dis}(x_1, x_2) = 1 - \cos\theta$

↓

angle b/w x_1 & x_2.

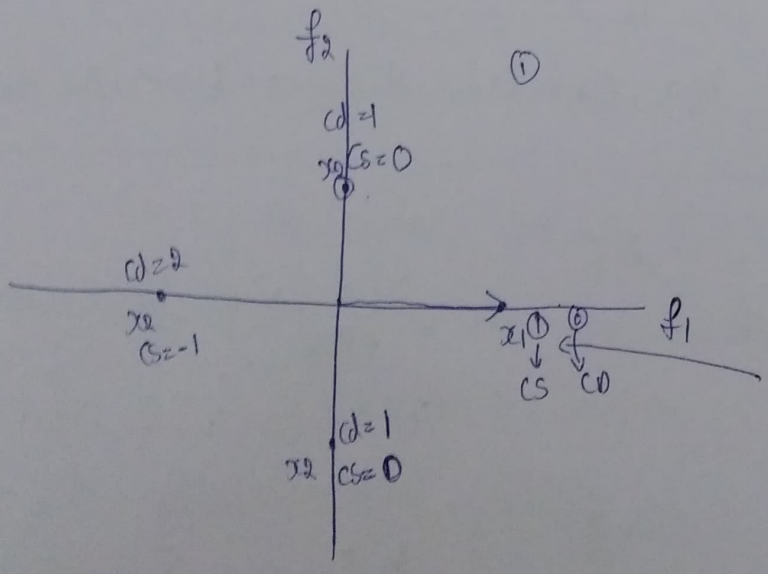
if we look at the "Euclidean distance" b/w "x_1 & x_3", & b/w (x_1 & x_2) we will find out that

$$d_{13} > d_{12}$$

but $\text{cos\_dist}_{13} < \text{cos\_dist}_{12}$

"Cos_distance" uses the angle b/w two points & not the distance as we imagine it geometrically.

Let's understand the properties of "Cosine-similarity" :-



as $\left[\text{cos\_dist}(x_1, x_2) = 1 - \text{cos\_sim}(x_1, x_2)\right]$

$\cos 270° = 0$

if "x_2" is lying with "x_1" then cosine similarity is ①

Imagine a 2d space having two features as shown above, & imagine we have two factors "f_1" & "f_2". Let us assume that there is a vector 'x_1' as shown above. Now imagine any other point (vector) 'x_2' in ① quadrant, as the angle between x_1 & x_2 increases the distance also increases.

if 'x_1' is parallel to feature1 (f_1) & x_2 is lyg in any region as shown previously

angle(x_1, x_2)	dist$_n$(x_1, x_2)	
$0° \to 90$	$0 \to 1$	} Becoz angularly speaking we are
$90° \to 180°$	$1 \to 2$	} going farther & farther away
$180° \to 270°$	$2 \to 1$	→ distance is going to reduce becoz
$270° \to 360°$	$1 \to 0$	we are coming closer back to 'x_1'

This is how the cosine distance works & how exactly cosine similarity works

'Cosine similarity' is nothing but the 'cos' of angle b/w two points

Let's assume 'x_1' & 'x_2' are two vectors.

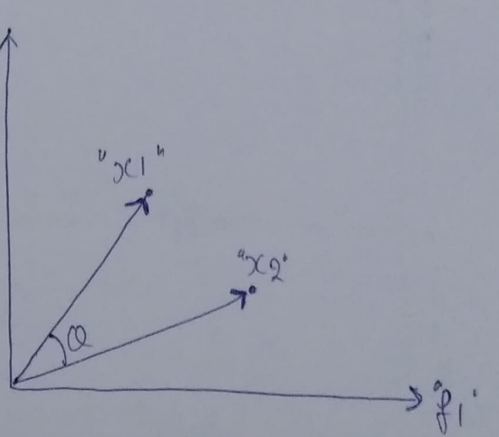
& in linear algebra we have learnt that

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

→ length of $x_1 + x_2$

→ L2 norm of x_2

→ L2 norm of x_1



Now if 'x_1' & 'x_2' are unit vectors, i.e if their length is 1 Unit

then $\boxed{\cos(\theta) = x_1 \cdot x_2}$

'Cos similarity' is nothing but the dot product b/w those two vectors (x_1 & x_2)

→ Relationship b/w "Euclidean distance" & "Cos similarity" :-

If "A" & "B" be two vectors & Euclidean distance is denoted by

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

When 'A' & 'B' are normalized to unit vectors $\|A\|^2 = \|B\|^2 = 1$

$$= 2\left(1 - \cos(A, B)\right) = 2 * \text{cos dist}(A, B)$$

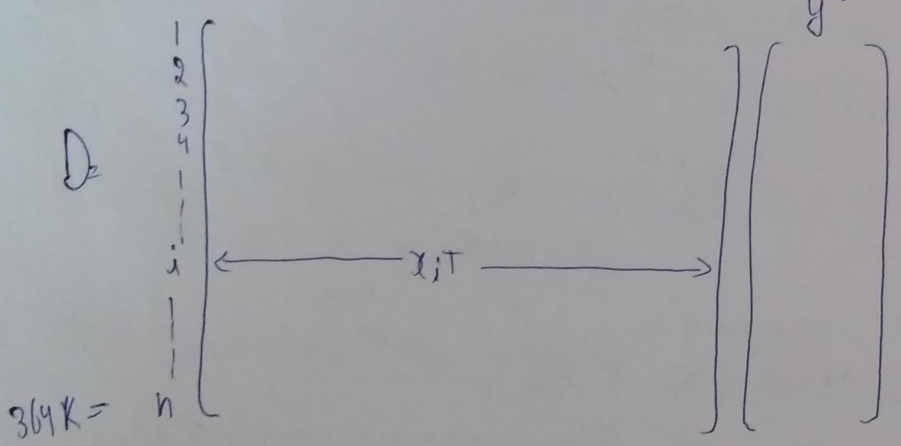
if x_1 & x_2 are unit vector)

$$\left[\text{euc-distance}(x_1, x_2)\right]^2 = 2\underbrace{(1 - \cos(\theta))}_{\text{cos-dist}}$$

$$\therefore \left[\text{Euclidean distance}(x_1, x_2)\right]^2 = \left[2 \cdot \text{cos\_distance}(x_1, x_2)\right]$$

→ 'How to measure effectiveness of K-NN' :

Let's say we have "Amazon food Reviews Dataset" comprising of 'n' reviews & where 'n' is roughly 364K after removing data duplication



Now for each review['i'] we have a vector representation called "x_iT"
& for each review, we also have a corresponding 'yi', where 'yi' ∈ {0, 1}

[0 → -ive review
1 → +ive review]

Now we have a very fundamental question, as how good is 'K-NN' in solving our problem.

Problem:- Given a new review "x_q" what is its polarity?

"polarity" means whether the review is ("+ive") or ("-ive")

This basically means, given ["xq" we want to determine the value of ["yq"). "xq ⟶ yq ?

↳ "yq" is polarity.

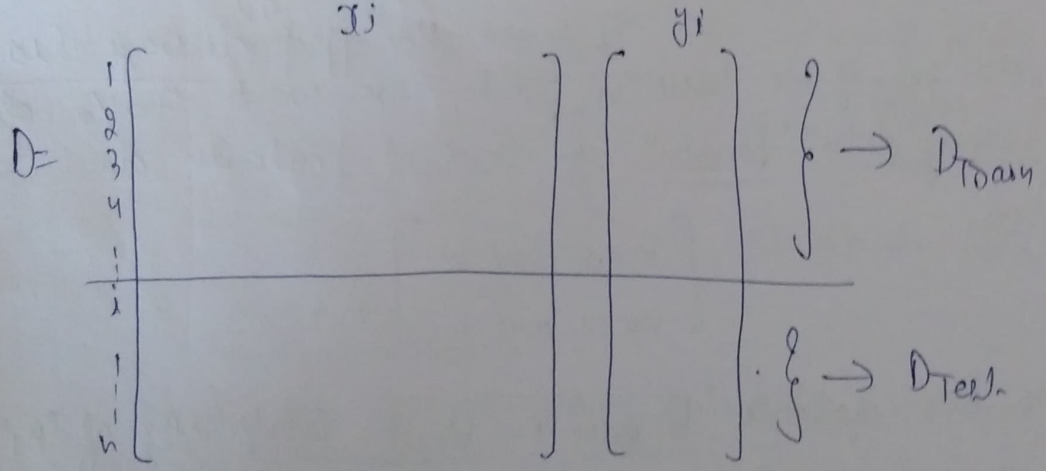
This is our problem, & for every problem, we want to measure the performance of the model we are building, here in our case it is "K-NN"

In KNN, we find the 'K' nearest neighbours & we do a majority vote.

Now let's see how to answer, (How good is KNN?) :-

Let's devise a strategy to answer this question.

We have our huge dataset "D" of 'xi's & yi's'



So, what we do is, we breakup our data into two parts

So, given our big dataset "D" comprising of "n" points, we are going to break it into two datasets.

"Dn" → "DTrain"
"Dn" → "DTest"

such that $D_{Train} \cup D_{Test} = D_n$ & $D_{Train} \cap D_{Test} = \phi$

This means given any point "xi" in "Dn" it is either there in "DTrain" or "DTest" but not both.

∴ for every point "x_i" in "D_n", it either goes to "D_{Train} or D_{Test}" but not to both.

Now the question is how to split the "entire dataset" into two parts "D_{Train}" & "D_{Test}"

"D_n" → "D_{Train}" → n_1

"D_n" → "D_{Test}" → n_2

"$n_1 + n_2 = n$"

One simple way to do that, is to do it randomly.

(It is one of the simplest ways, but not the only way).

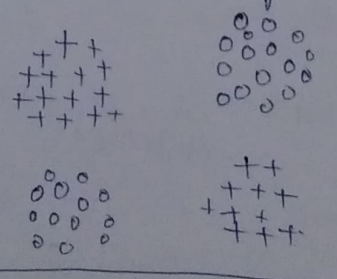
There are other appropriate ways also.

for simplicity, let's say if we have "n" points, we want 70% of points in our "Training data" "D_{Train}" & 30% of points in our "Testing Data" "D_{Test}"

$$\left[\begin{array}{l} n_1 \approx 70\% \, n \\ n_2 \approx 30\% \, n \end{array}\right]$$

Now if we have two datasets "D_{Train}" & "D_{Test}" containing of "n_1" & "n_2" points.

Now we can use "D_{Train}" to train the (K-NN) & then measure, how well it performs, we are going to use the "D_{Test}" data.



D_{Train}

Now each point 'x_i' in $D_{Test} = \{(x_i, y_i)\}_{i=1}^{n_2}$

make it as 'x_q' & find corresponds 'y_q'

→ x_q = pt

Use D_{Train} & K-NN to predict 'y_q'

PTO

```
count=0;
for each pt in D_Test :
    x_q = pt
    Use D_Train & KNN to
    determine y_q.
Now we compare predicted y_q's
    If y_q == y_pt
        Count = Count + 1
```

→ At end of the loop

Count = # of points for which D_{Train} & KNN gave a correct class label.

Now § an ~~say~~ a measure called Accuracy which is equal to

$$\text{Accuracy} = \frac{\text{Count}}{n_q}$$

Count → No. of points for which D_{Train} + K-NN gave a correct class label

n_q → No of points in D_{Test}

"$0 \leq Acc \leq 1$"

if let's say our accuracy is 0.91, it means that 91% of times KNN, with D_{Train} is able to correctly predict y_q from ~~test~~ data set.

Now using all this data, we can conclude that :: K-NN on Amazon food reviews using "D-Train" gives me an accuracy of 91%. (Toy example)

This is how we measure the effectiveness of "K-NN" on a given problem.

→ "Test/Evaluation Time & Space Complexity" :⇒

Given a query point "x_q", how much "time & space" do we need to compute corresponding "y_q".

Let's write a simple "pseudo code".

Input: D_{Train}, K, $x_q \in \mathbb{R}^d$; Output: y_q.

KNNpts = [ ]

for each x_i in D_{Train} :

n — O(d) - Compute $d(x_i, x_q) \rightarrow d_i$

O(1) - Keep the smallest K-distances. → (x_i, y_i, d_i) → Putin KNNpts [ ]

At the end of this loop, we get a list of all the nearest points

Now let's see how much time does it take.

for each dimension we have to do subtraction & square.

Now since 'K' is small so we can optimize it to be a. O(1) fn

∴ Total time complexity of this is O(nd)

O(nd) is the time complexity to find 'K-NN OR nearest neighbours

O(K) — Now since 'K' is small O(K) becomes O(1)

```
Cnt_pos = 0 ; Cnt_neg = 0
for each xi in KNNpts
    if yi == +ve
        Cnt_pos += 1
    else
        Cnt_neg += 1
```

PTO

O(1) { if cnt_pos > cnt_neg
return $y_q = 1$ → +ve
else
$y_q = 0$ → -ve.

So most time taking part in the Algo is, finding the K-nearest neighbours.

Time complexity :- $O(nd) + O(1) + O(1)$

$= O(nd)$

if d is small i.e. "$d << n$"

then Time Complexity $= O(n)$

So, given a single query point, it takes roughly "$O(n)$ time".

This is typically alarming

Lets take our "Amazon food reviews dataset".

Our $n \approx 364K$
$d \approx 100K$ (Bag of words)
≈ 300 (tfidf)

lot of memory

Now imagine, we have to deploy this system.

So, given an ['x_q'] if we have to compute corresponding ['y_q'], what all things we need to store in memory.

RAM/Memory

"$x_q \rightarrow y_q$"

"D Train" → O(nd) → storage

This means it takes large amount of memory.

In ML, we will say what is the amount of memory consumed at runtime or evaluation time.

Time complexity :- $O(nd)$ straightforward

Space Complexity ⇒ space that is needed to evaluate.

to get y_q from x_q

$O(nd)$

→ "Decision surface for K-NN or K-Curves"

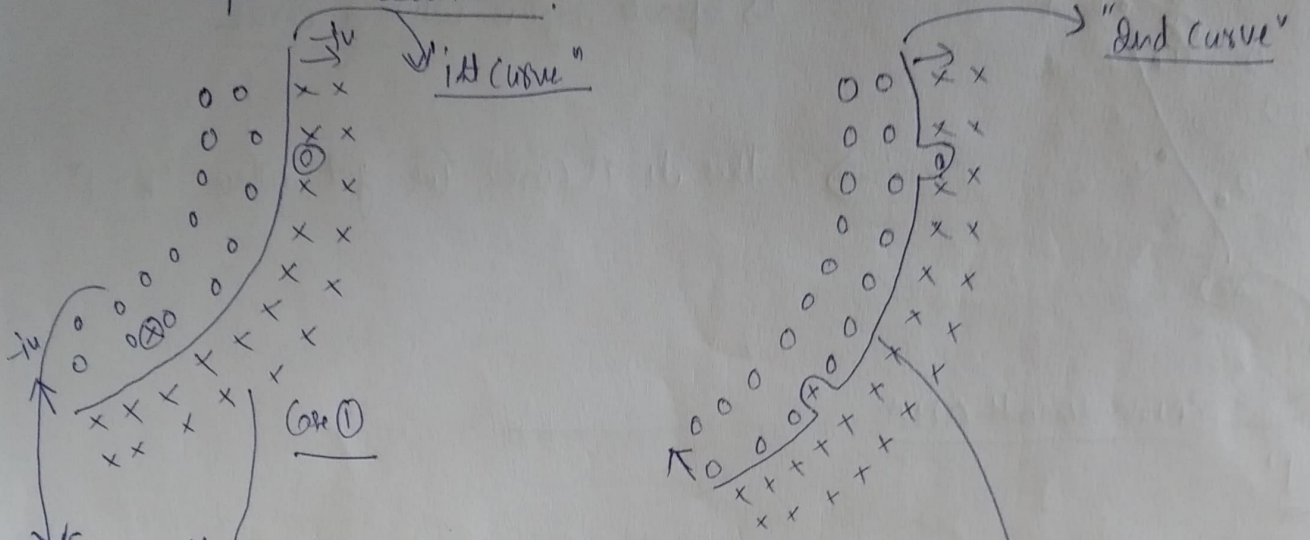
Now let us find out how to pick the right ['K'].

'K' in KNN is often referred to as a "hyperparameter"

Now let's see how to determine the right 'K' & lets find out how things change as 'K' changes.

Let's understand it geometrically.

Let's take two datasets.



Here all are -ive points & in between we have a +ve point.

Here all are +ive points & in between somewhere, we have a -ive point.

If we ask a kid to draw a line to separate these points, he will draw the line/curve as shown above. He will say everything that lies on one side of the curve belongs to one 'set of points' & everything that lies on the opposite side of the curve belongs to other 'set of points'.

There is one other way to draw the curve, because if we consider the curve drawn earlier, the two points are going to be classified incorrectly.

So, for somebody who gives more importance to errors, he will draw the curve in such a way that no point is left behind, & he will say that everything which lies on one side of the curve belongs to one 'set of points' & everything which lies on other side of the curve belongs to the other 'set of points'.

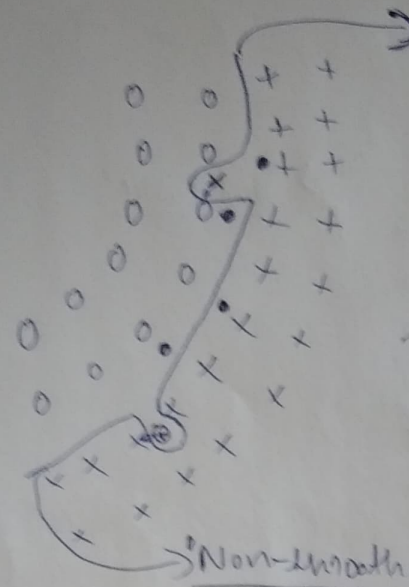
1st curve make two errors & 2nd curve makes no errors at all.

P.T.O

first curve is a smooth curve & not jagged, whereas 2nd curve is having all kinds of "twists" & "turns".

Note:- The curve that separates "+ve" points from "-ve points" is called a decision surface.

The decision surface helps us understand, as what happens when "K" changes.



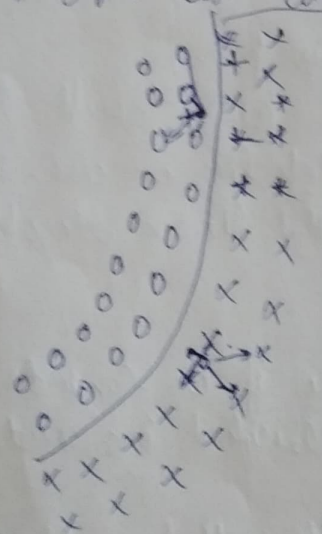
→ 'Decision surface' on the basis of many query points'
↳ (1-NN) (K=1)
↳ Separate regions using a curve.

→ This is a curve we get when "K=1"

'Non-smooth curve'

'K=1'

Let's draw the curve for the same dataset, but with "K=5"
i.e K=5; 'Majority vote'



→ Smooth curve

So, one thing we notice is as "K" increases, the smoothness of the curve increases.

∴ In K-NN, the smoothness of the decision surface increases as K increases.

We say 'K=n' → worst case
'n is the Total no. of points'

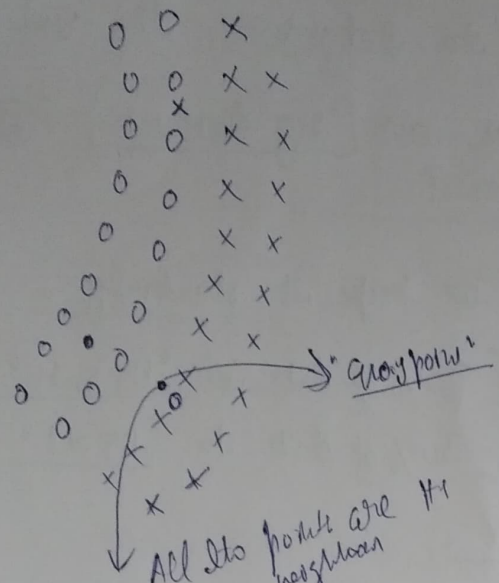
"n" → Total no of points in our dataset

"n_1" points are positive among these "n points"

"n_2" points are negative among these "n points"

"$n_1 + n_2 = n$"

Let's assume that "$n_1 > n_2$"



Now $K = n$

$n_1 =$ +ive

$n_2 =$ −ive

& $n_1 > n_2$

∴ it query point"

Now let $n = 1000$, $n_1 = 600$ & $n_2 = 400$

Now we have to take "1000" nearest points / ∴ all the points will come.

Out of all these points, "600 are +ive" & "400 are −ive"

∴ this point will be called a +ive point.

Now let's take a query point deep inside the +ive point region. Now if we take 1000 nearest neighbours of it. "600" will be +ive & "400" will be −ive ∴ the query point will be mapped to the "+ive-class".

So wherever our query point is, our "1000-NN" will say everything is positive. There is no more a decision surface, everything is +ive. becoz we have more +ive points.

So here we understood geometrically, as what happens when $K = 1, 2, 3, \ldots n$"

As "K" increases our curve gets smoother & when "K" becomes "n", every thing will become one class

→ "Need for cross validation":

We have already seen what happens when 'K' changes, ie when ["K=1"] we get an "overfit function", when ["K=5"], we get a "fit function" & when ["K="n""] we get an "underfit function".

Now the big question here is how to determine the value of 'K'.

We have now two datasets, as our ["big dataset"] "D_n" is broken down into two datasets

"D_n" → "D_{Train}"
"D_n" → "D_{Test}"

& we broke it randomly

[70% of data in "D_{Train}"
30% of data in "D_{Test}"]

One idea to determine 'K' could be

Let's "K=1", we keep training data for training the model. Now using "D_{Test}" we will try to find out the accuracy

↳ [Means No of points that are correctly classified divided by total no of points]

	Train	Accuracy on D_{Test}
K=1	D_{Train}	

for, for "K=1", we take each point in D_{Test} & we try to predict the label.

Remember our D_{Test} also contains pairs like

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

We could make $x_q = x_1$, $x_q = x_2$, ——— $x_q = x_{n_{test}}$ for each of these we are getting a corresponding y value.

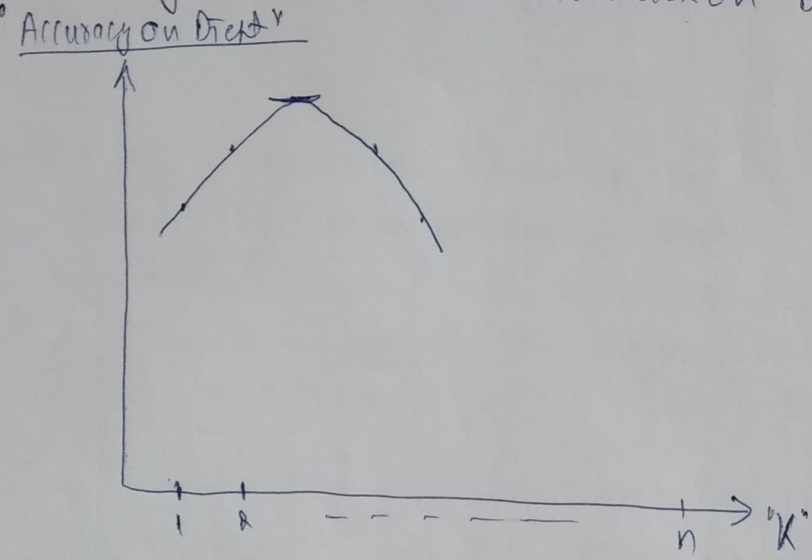
Now for each of the query point, we get a "yq" & now these "yq's are compared with the "y" values that we already have in our D_{Test} dataset.

& if both agree, we say it is correctly classified else misclassified, we have seen this concept, while computing accuracy.

Let's say.

	Train	Acc. on D_{Test}
K=1	D_{Train}	0.78
K=2	"	0.82
K=3	"	0.85

Now let's draw the values, on 'x-axis' we draw the "K-values" & on 'y-axis' we have our accuracies on D_{Test}.



Typically what happens is, as K increases the performance will increase upto some point & then it will start decreasing.

Higher the accuracy the better it is..