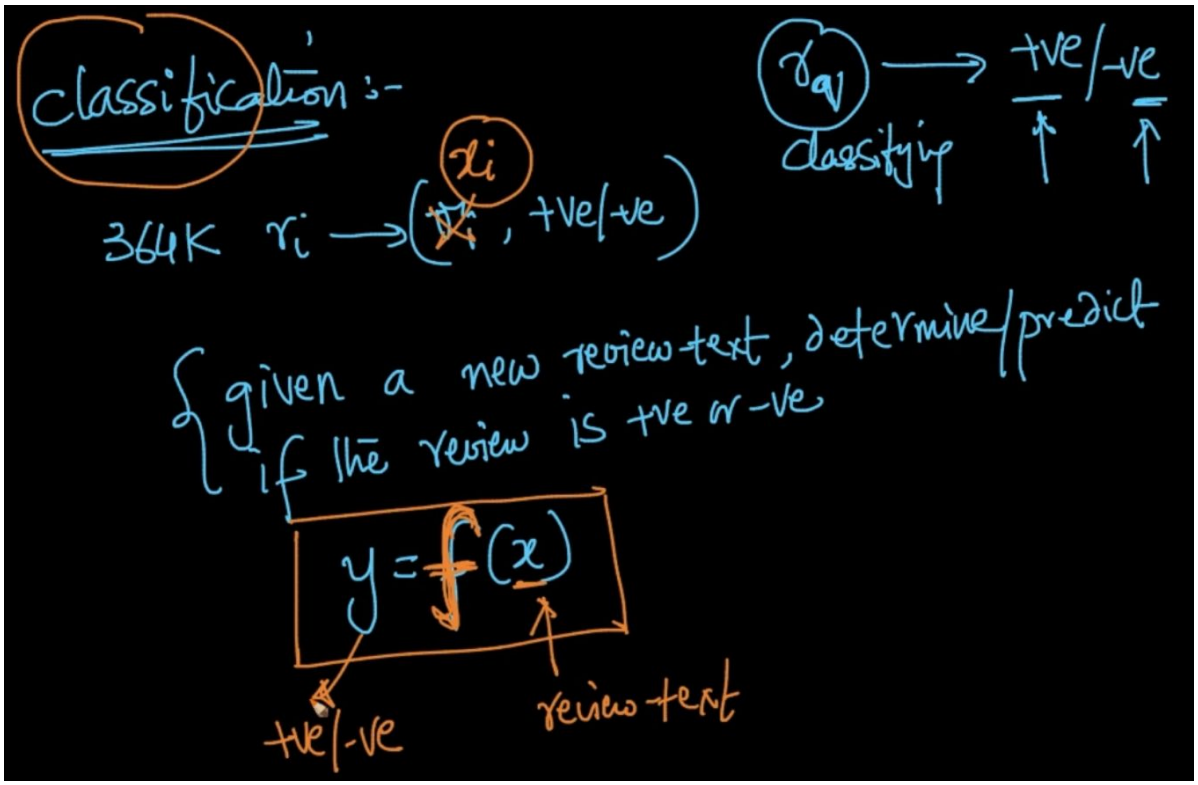
K-Nearest Neighbor

K-Nearest Neighbors(k-NN)

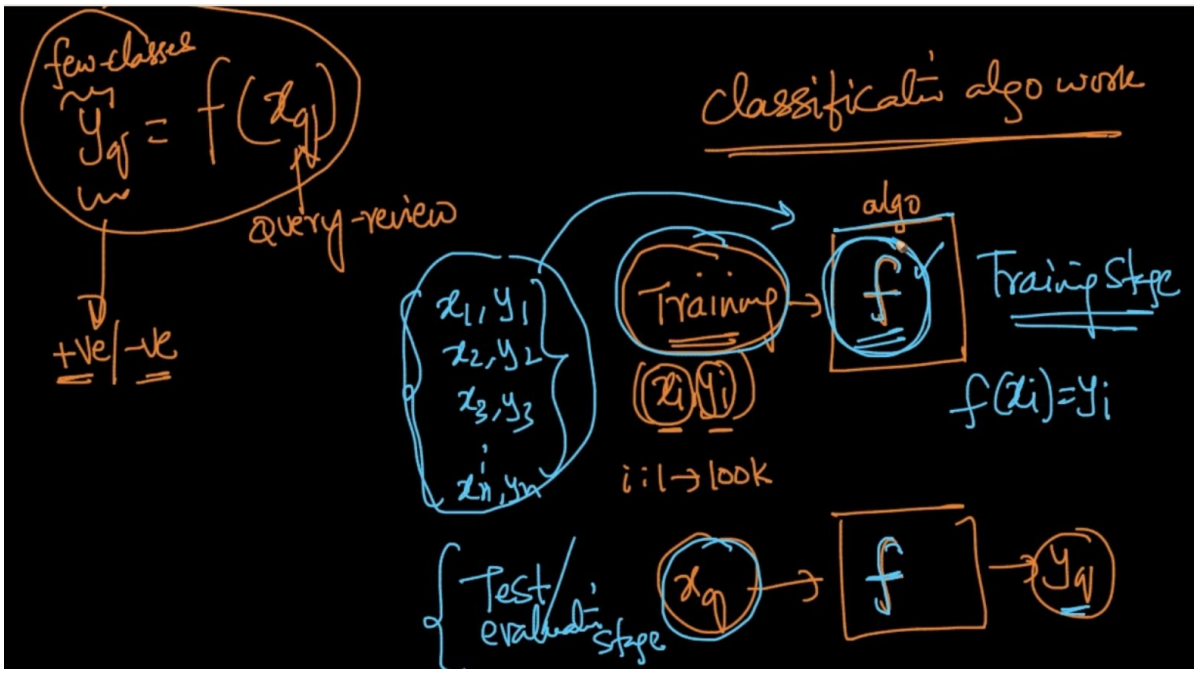
HOW CLASSIFICATION & REGRESSION WORKS?



We are converting our reviews into vectors using techniques like tf-idf, BOW, Word2vec. So if we are given a new review r_q we will determine if our review is +ve/-ve*(label y)

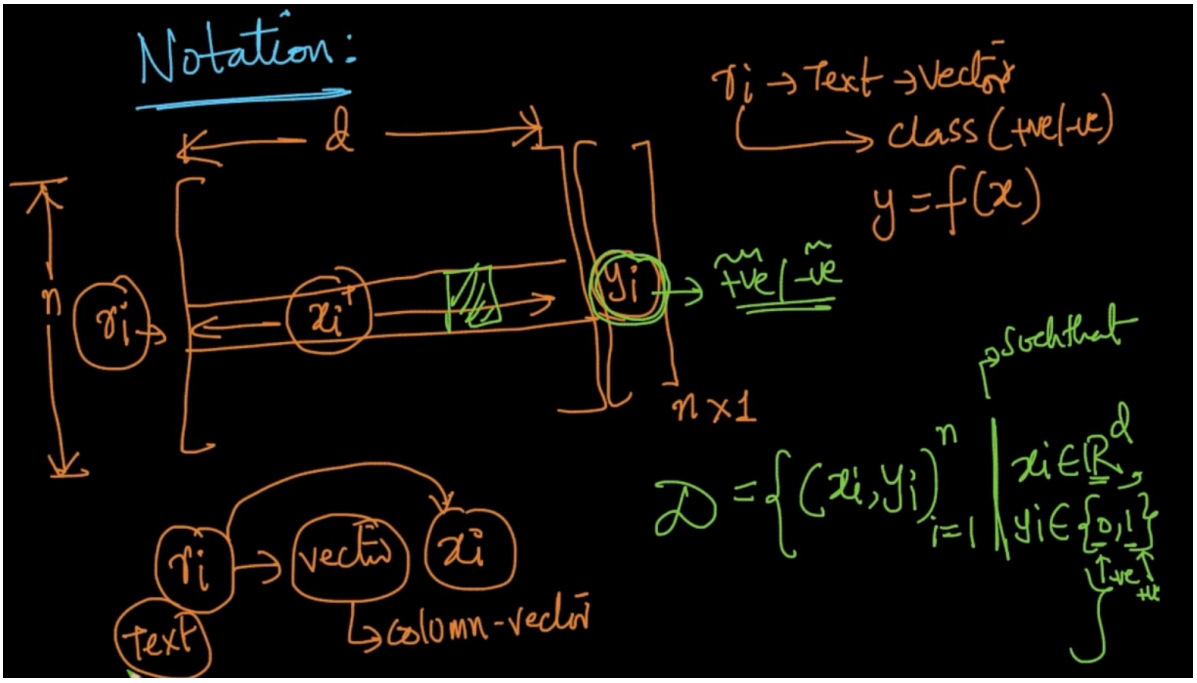
Therefore, our goal is to find the function 'f' applied on converted vector x (review text) such that it gives 'y' (prediction label).

How classification happens?



In our dataset, there are x and y (Train data) and we give that to our classification algorithm and it tries to learn the function f such that y = f(x) to predict the target variable given a new dataset x_q

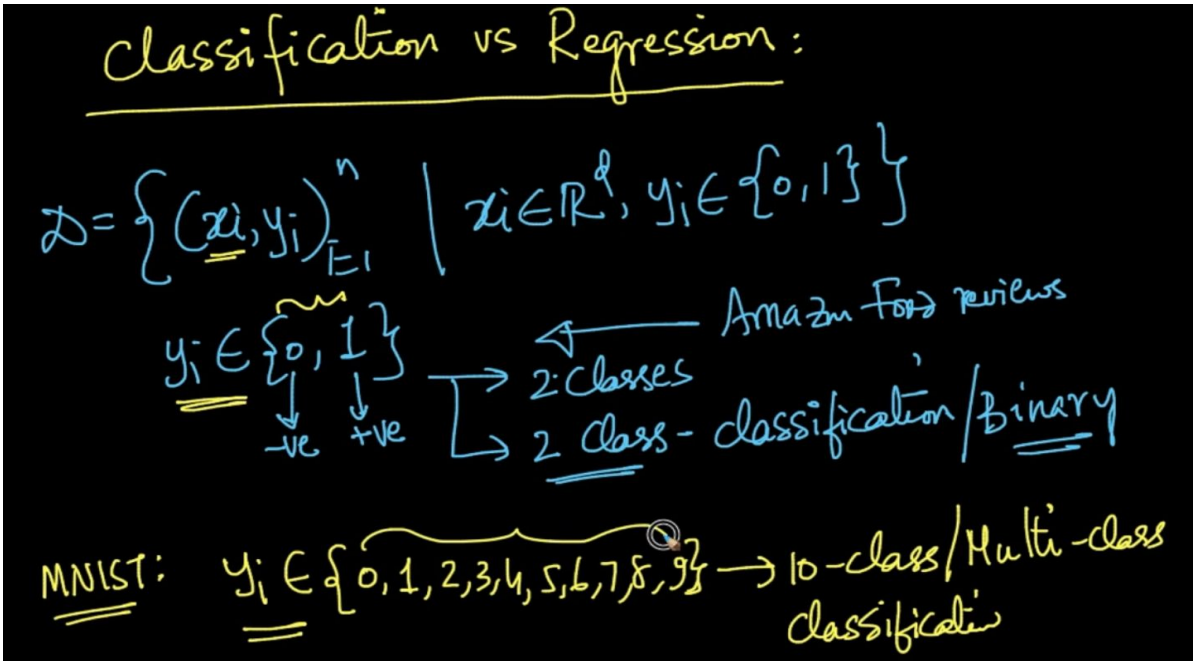
DATA -MATRIX NOTATION



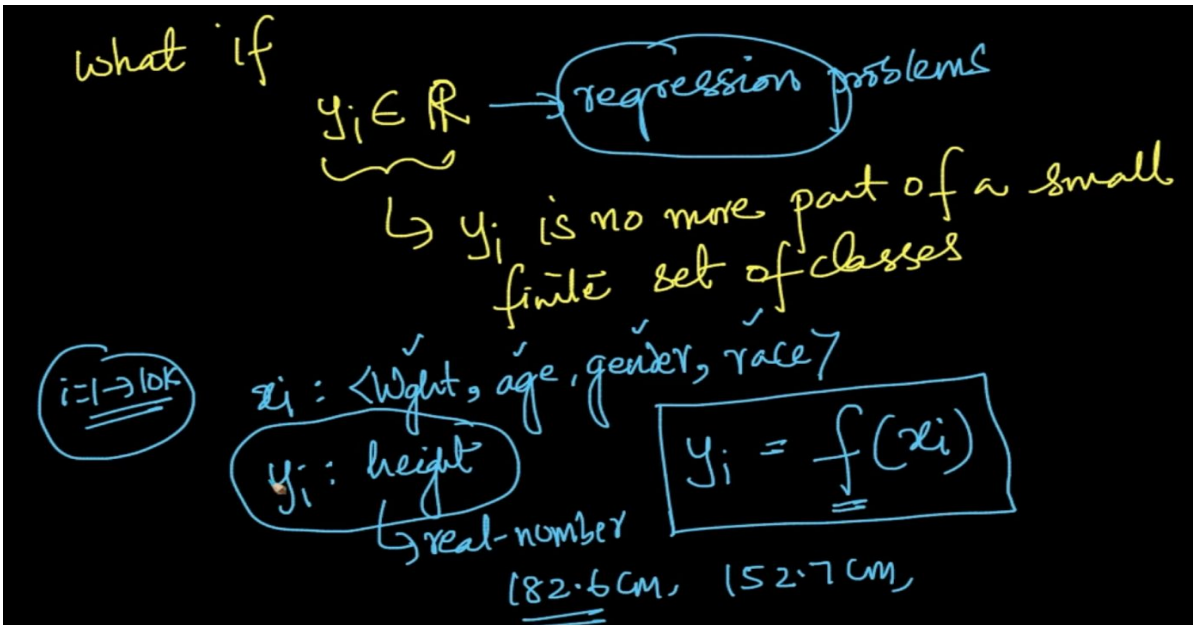
The notation is D = { $(x_i, y_i)_{i=1}^{n}$ | $x_i \in R^d$, $y_i \in \{0,1\}$} which means D is a set of x, y pairs such that x is a d-dimensional vector and y has values (0,1) which is our target label.

We input both our x and their corresponding y into our classification algorithm and our algo learns the function f such that f(x) = y and if we give a new input x_q the function f gives us the value of y . SO our goal is to learn that function.

CLASSIFICATION v/s REGRESSION

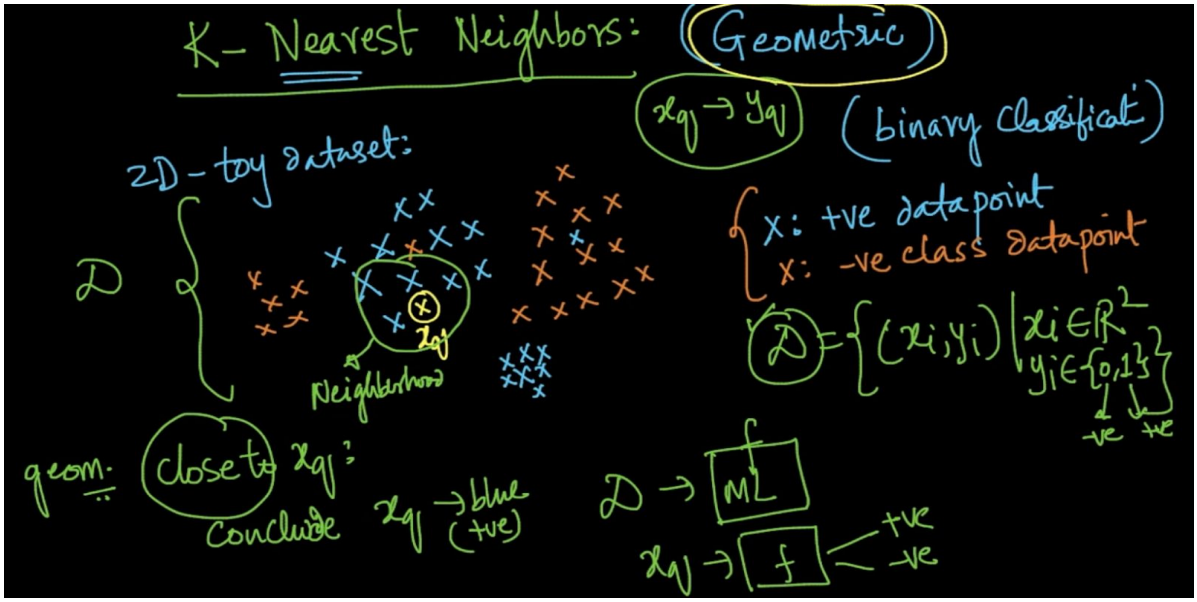


This is our classification where y $\in \{0, 1\}$ (binary classification) or y $\in \{0, 1, 2, ...9\}$ multi class classification. It means y $\in$ finite set of values here

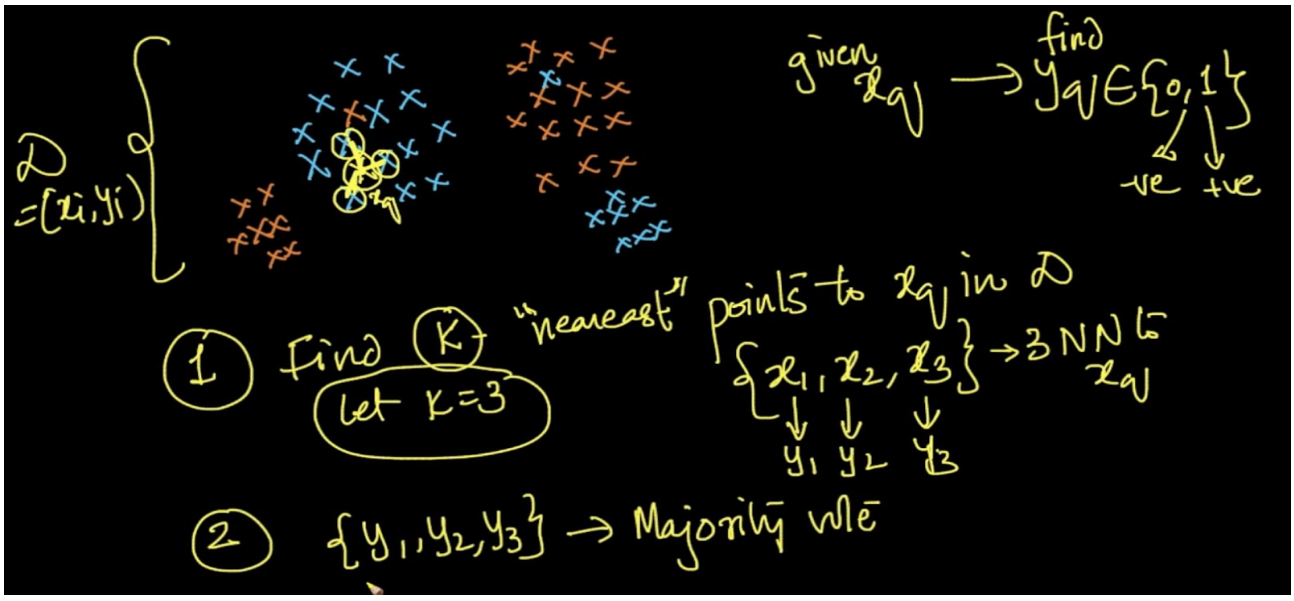


In Regression, y $\in R$ (real number) . It can be seen above where we are predicting height.

K -NEAREST NEIGHBORS INTUITION (KNN)

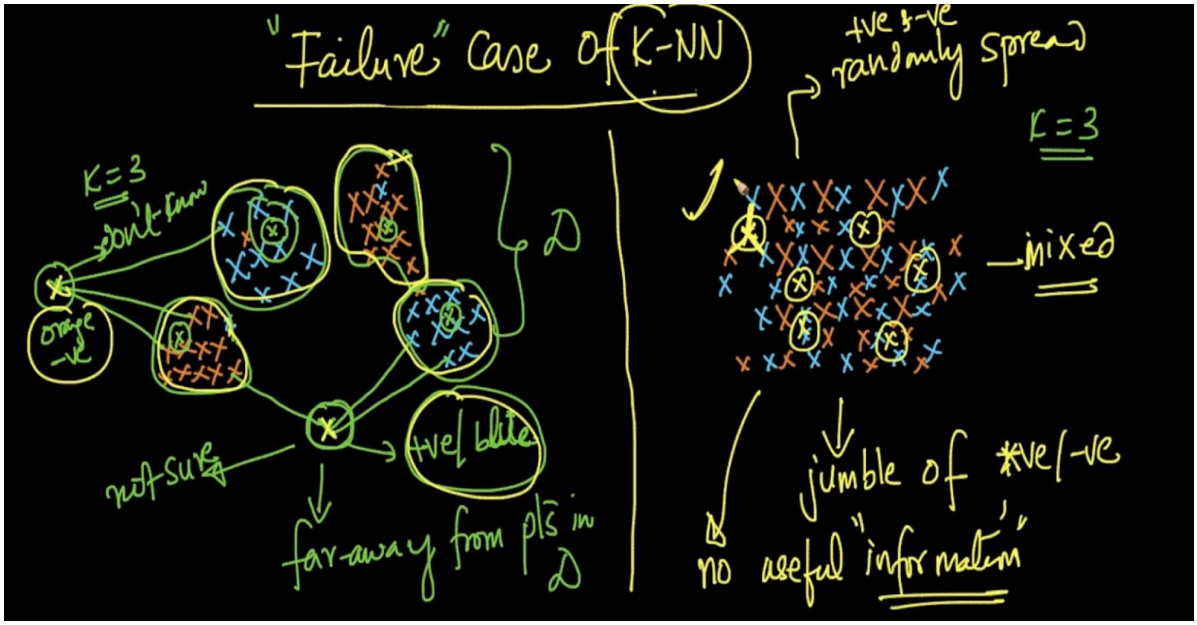


Our 2-D dataset is plotted here as positive and negative (y) and we are giving it a query x_q We need to predict the class of x_q . So we are taking the geometrically closer points to it and then predicting its class. Since closer points are blue x_q has positive class y_q



We select k nearest points to x_q and all the nearest points have their corresponding y. We use that y's and do a majority vote. Ex (y1,y2,y3) = (+,+,+) => y_q = + . Note: **k** should be odd

k-NN FAILURE CASES

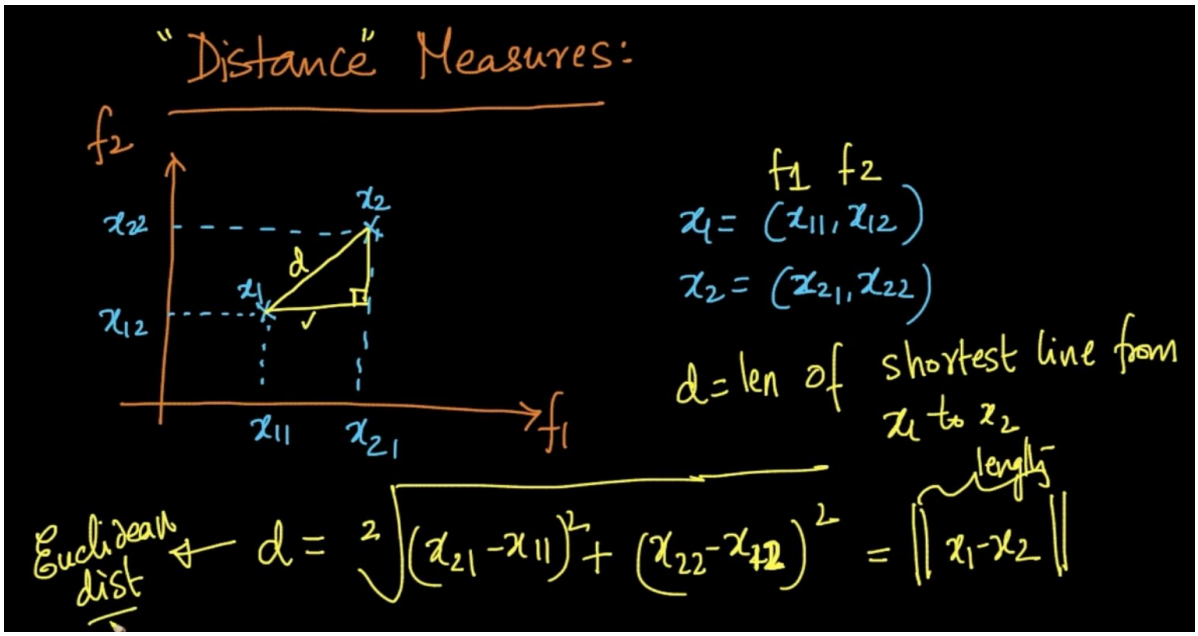


Case 1 (L.H.S) : Here, we can estimate classes if data point appears near to it but what to do if it's far away . As seen above, if the point is far away then we can't be sure of whether it's positive or negative

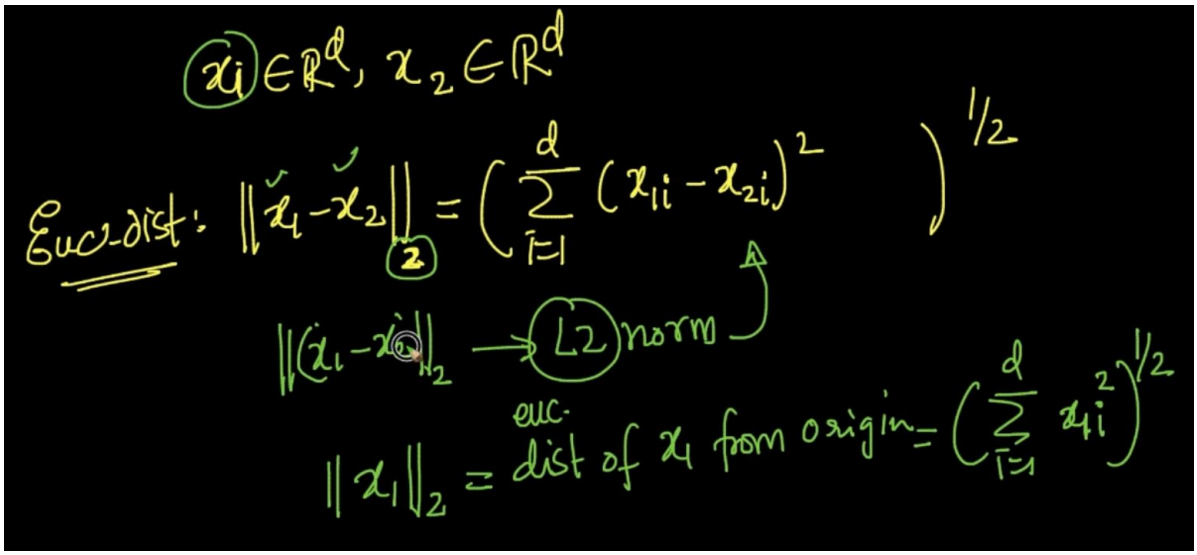
Case 2 (R.H.S) : The data points are randomly spread so if we bring a new data point it'll suck because since it's random we can't get useful information from it

DISTANCE MEASURES

EUCLIDEAN DISTANCE

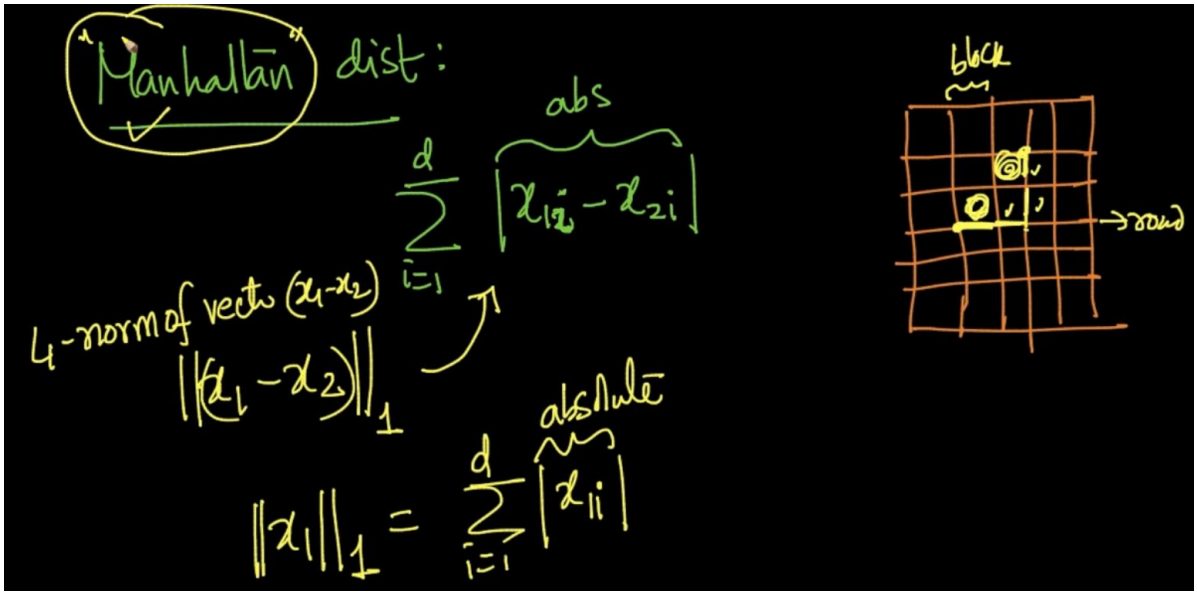


This is Euclidean distance of two points which is basically the shortest distance between two points x1 and x2 represented as ||x1-x2||.

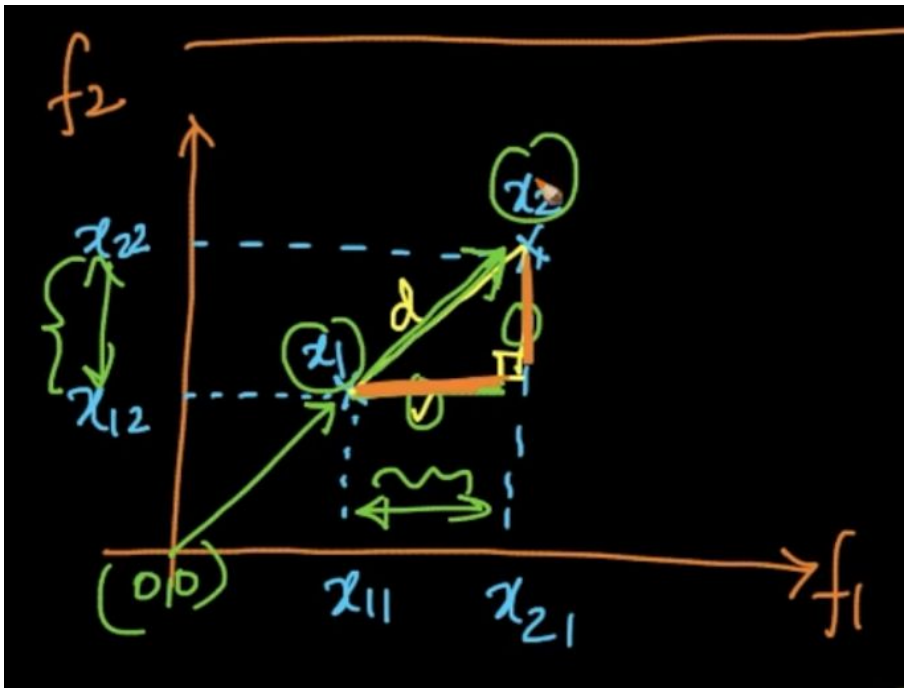


For d-dimensional vectors x1 & x2 represented as $||x1 - x2||_2$ also known as L2-norm

MANHATTAN DISTANCE

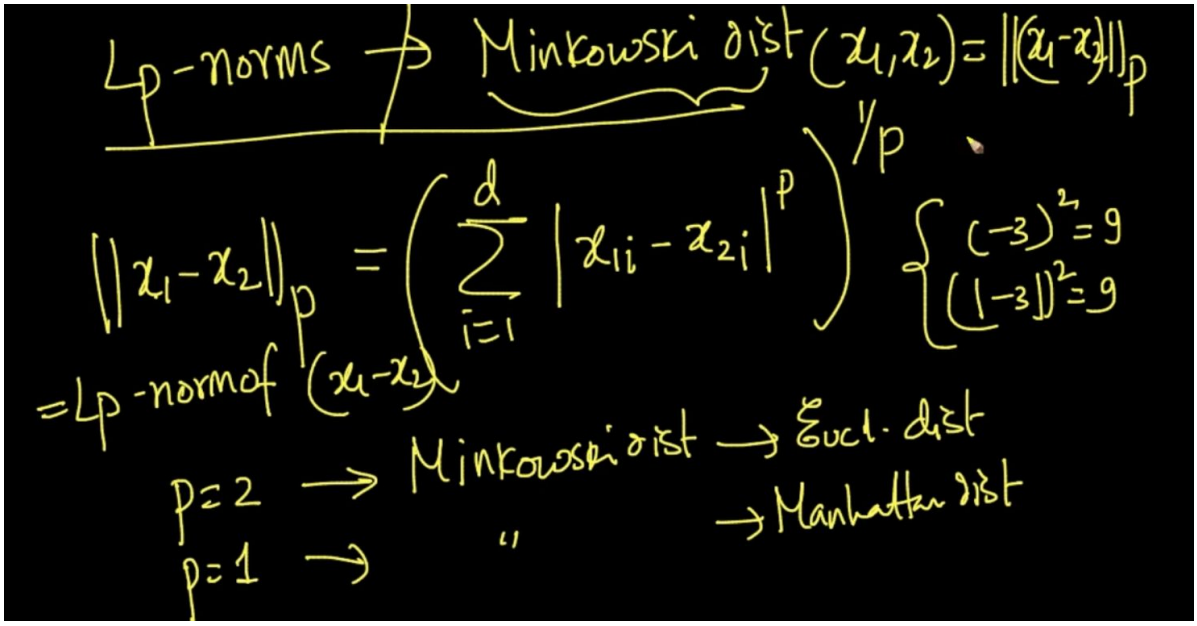


It is nothing but the sum of distance of dimensions. Also known as L1 norm and the value we get is absolute value.



Manhattan distance of $x_1 \& x_2$ is the summation of the distances in orange i.e $((x_{21} - x_{11}) + (x_{22} - x_{12}))$ unlike Euclidean in which we used Square root $\sqrt{(x_{22} - x_{11})^2 + (x_{22} - x_{12})^2}$

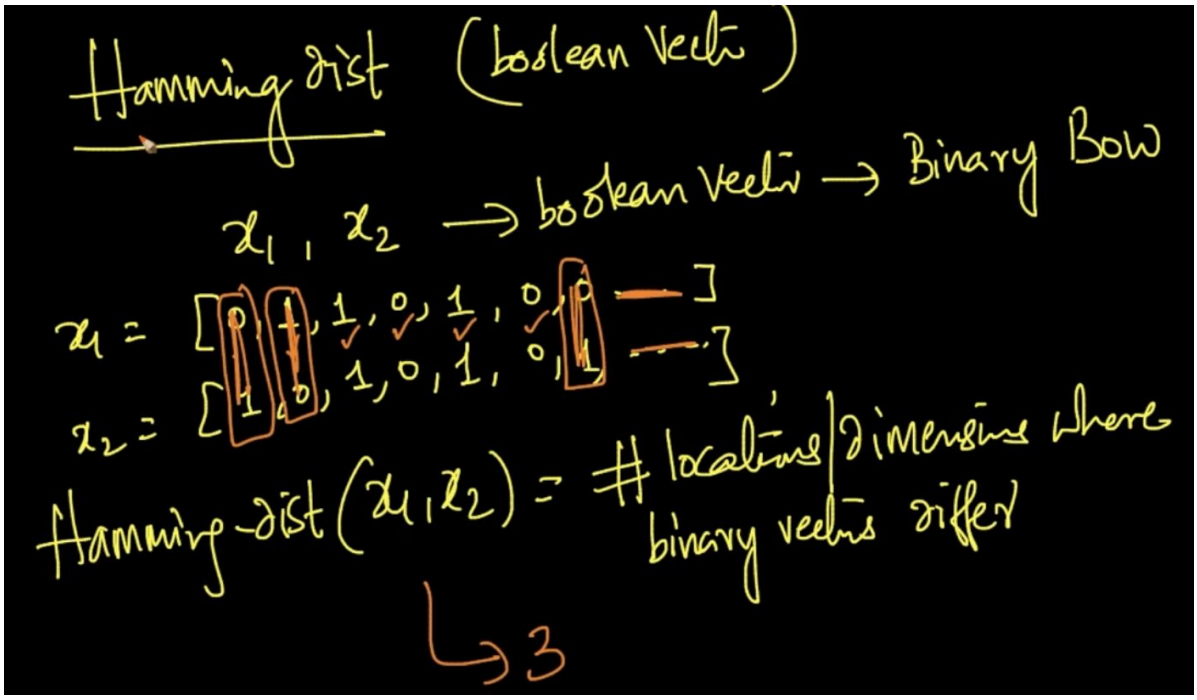
LP NORM -> MINKOWSKI DISTANCE



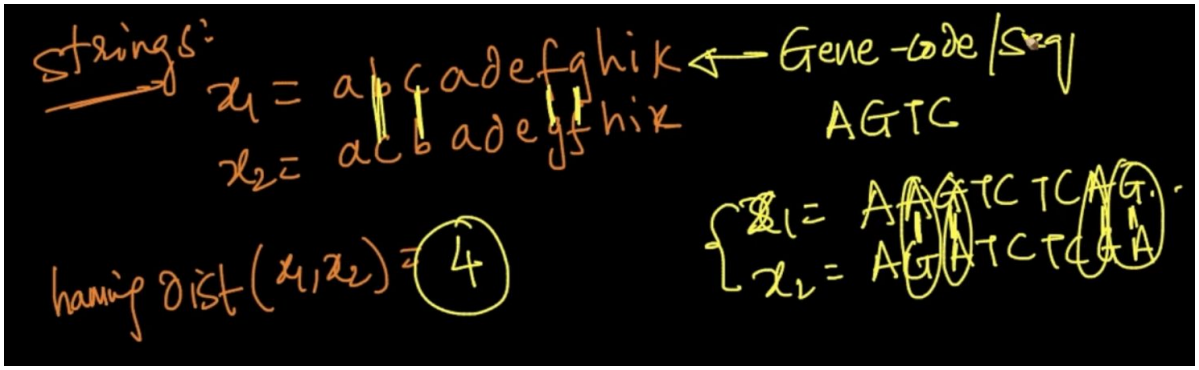
Generalization of L1 or L2 norm. Lp - norm is nothing but Minkowski distance between two points.

Note - Lp norm is of vector you get by subtracting (x1 - x2) and Distance is between two points x1 & x2

HAMMING DISTANCE

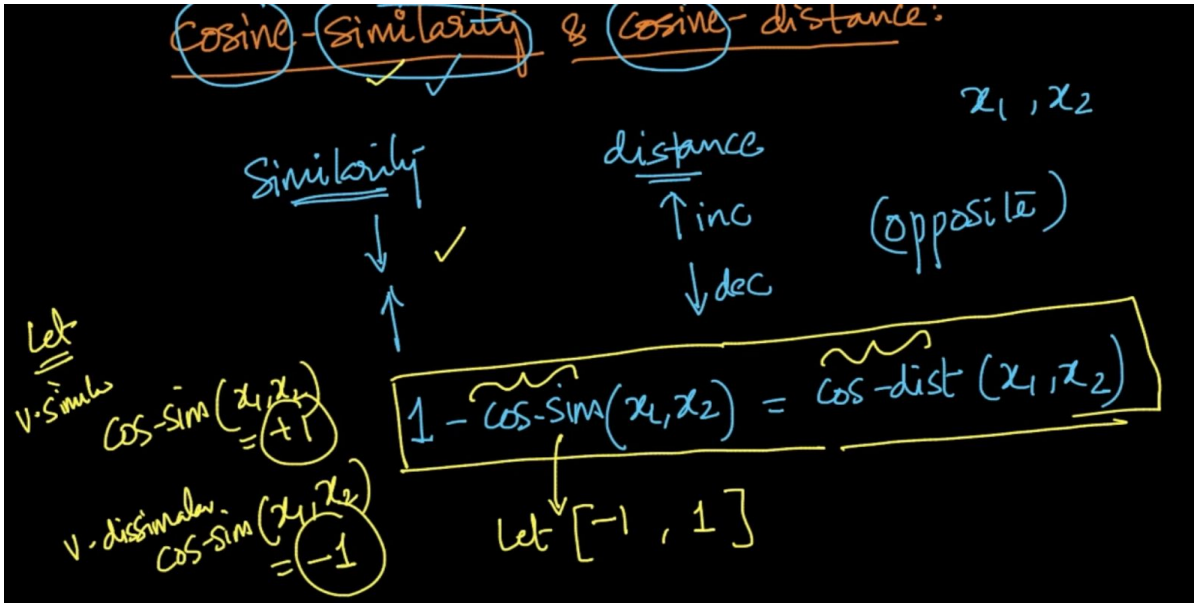


It is used to find the difference in Boolean vectors or Binary BOW. Hamming Distance is no. of locations/dimensions where binary vectors differ . Hamming dist(x1,x2)=3 above

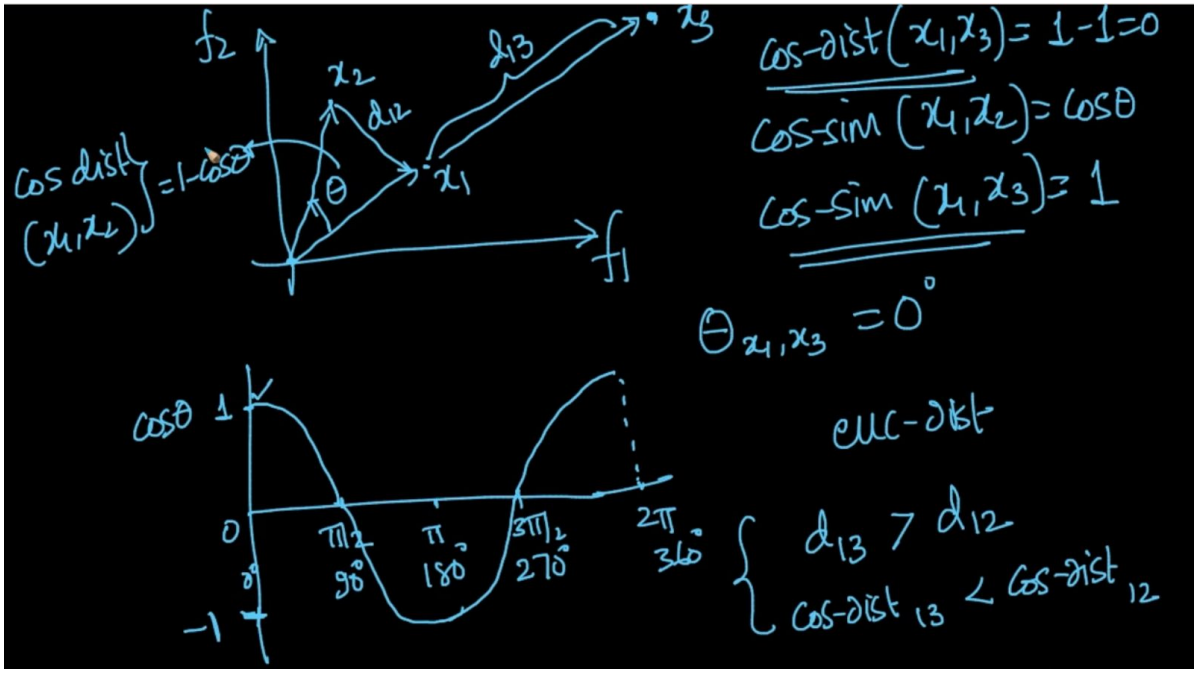


Also used to find difference between two strings . Extensively used in Gene Code as GEne code is basically a String as seen above . Ex : x1(gene code) = AGTCGGTA

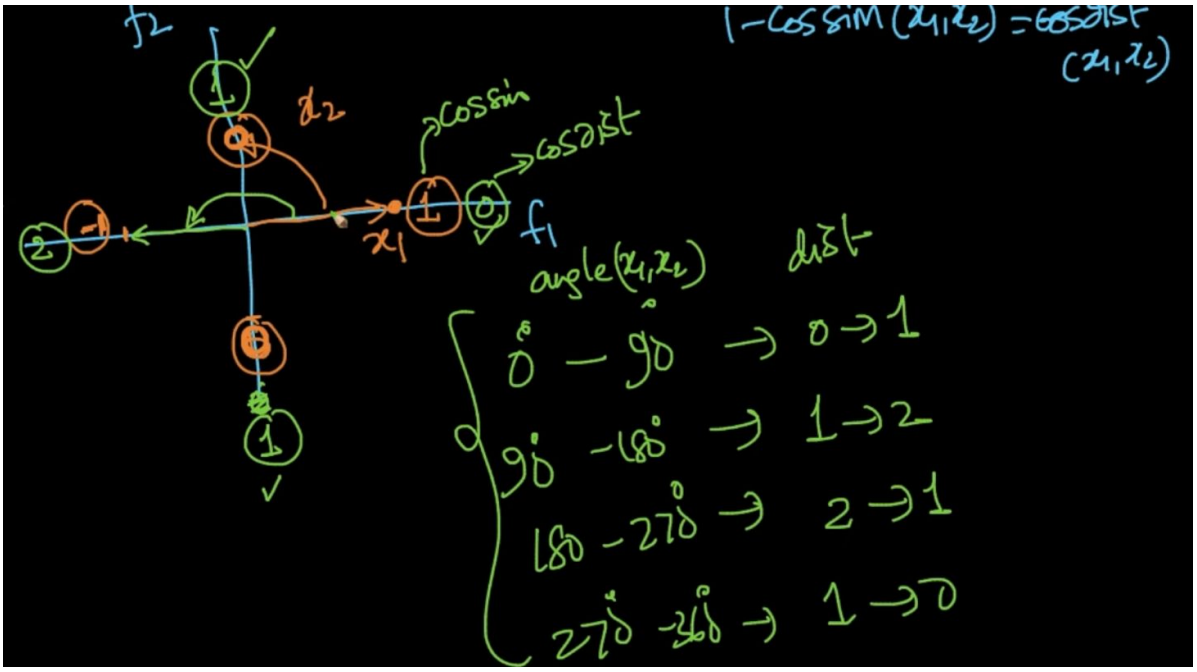
COSINE SIMILARITY



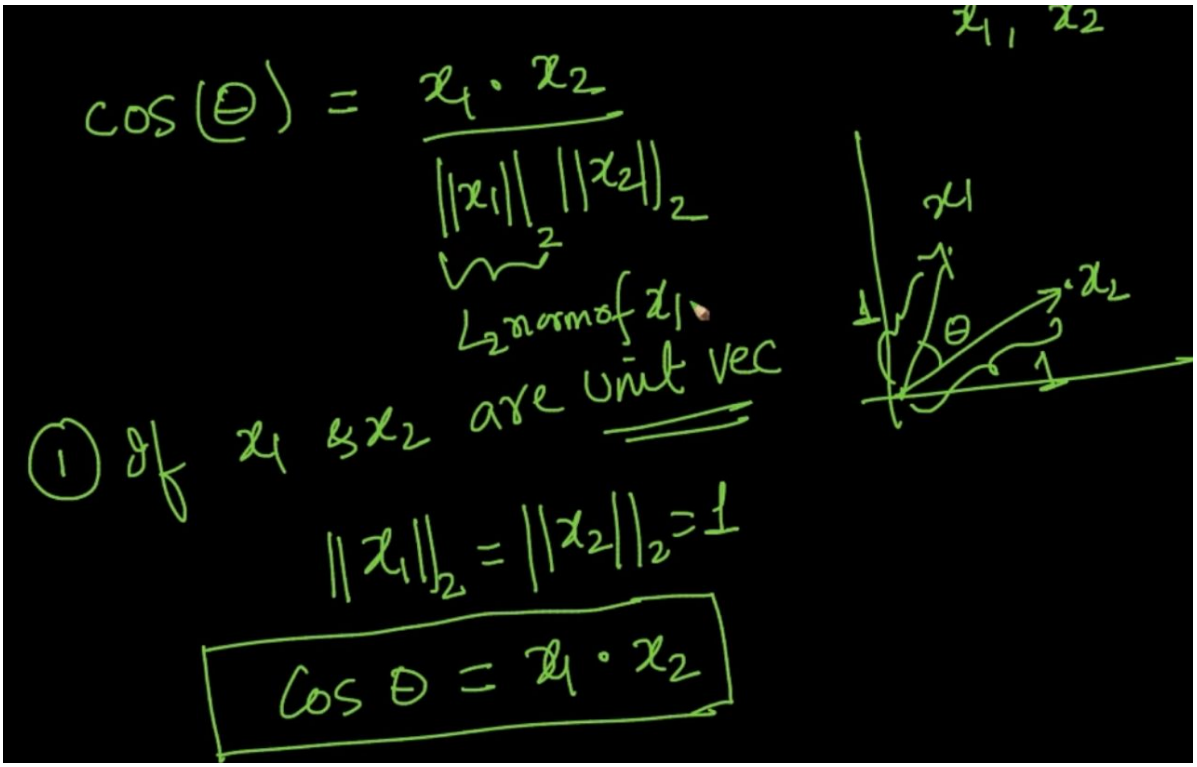
As the distance between 2 points increases similarity decreases and vice-versa. This is how it should be formulated.

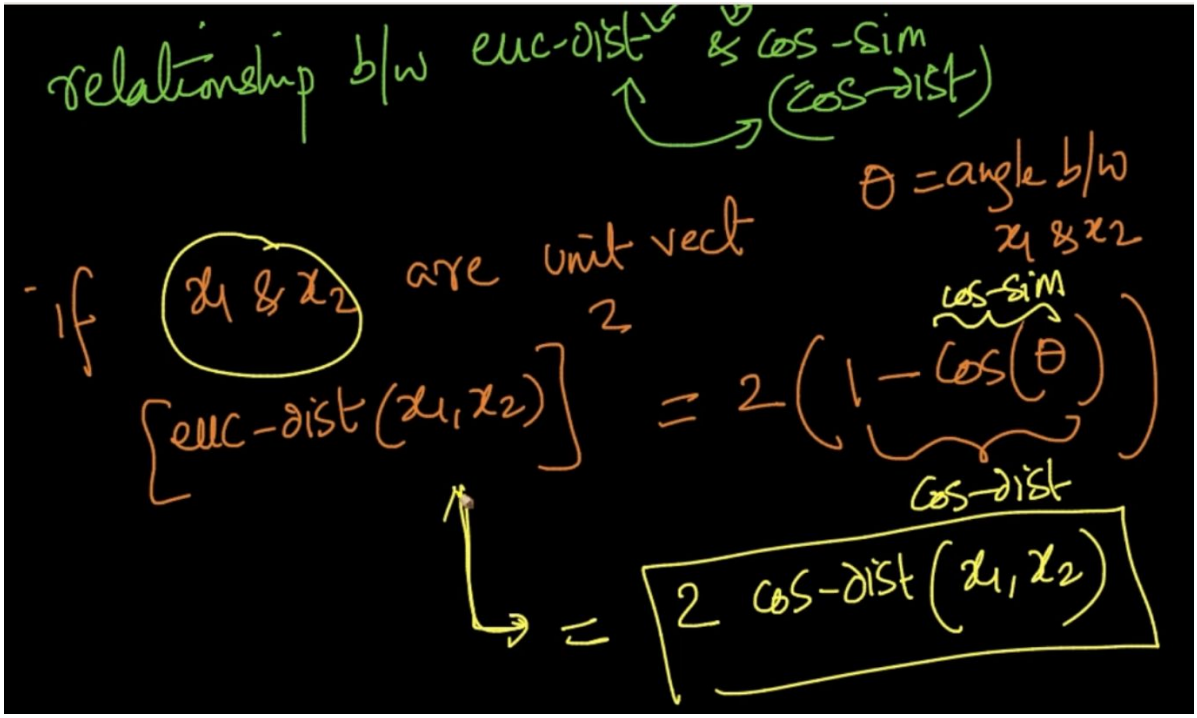


Cosine similarity is dependent on angle between 2 vectors. As $\theta_{x1,x3}$ = 0 cos-sim(x1,x2)=0

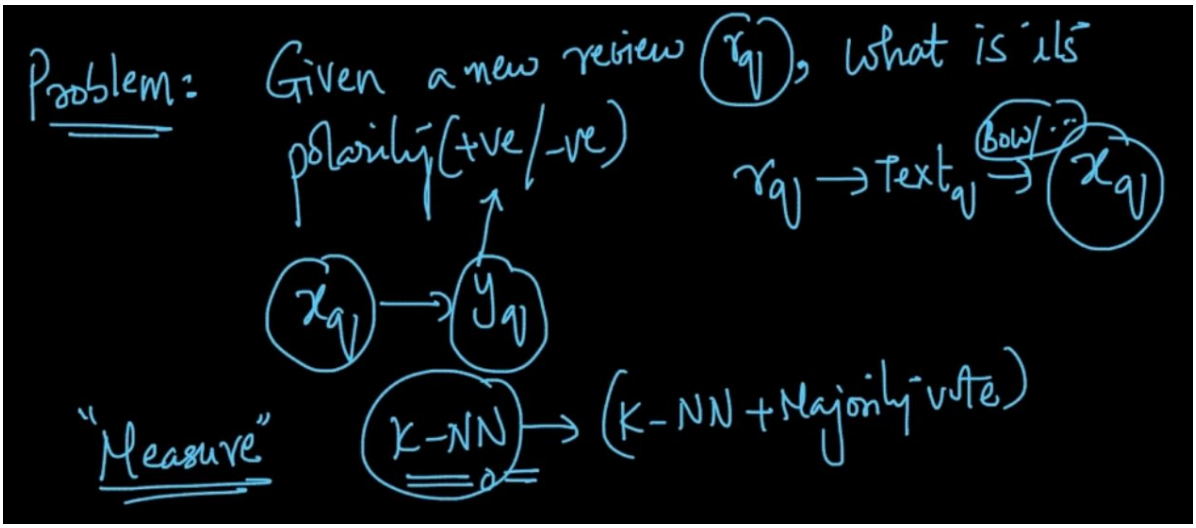


Cosine similarity and cosine distance of two points when angle is changing.

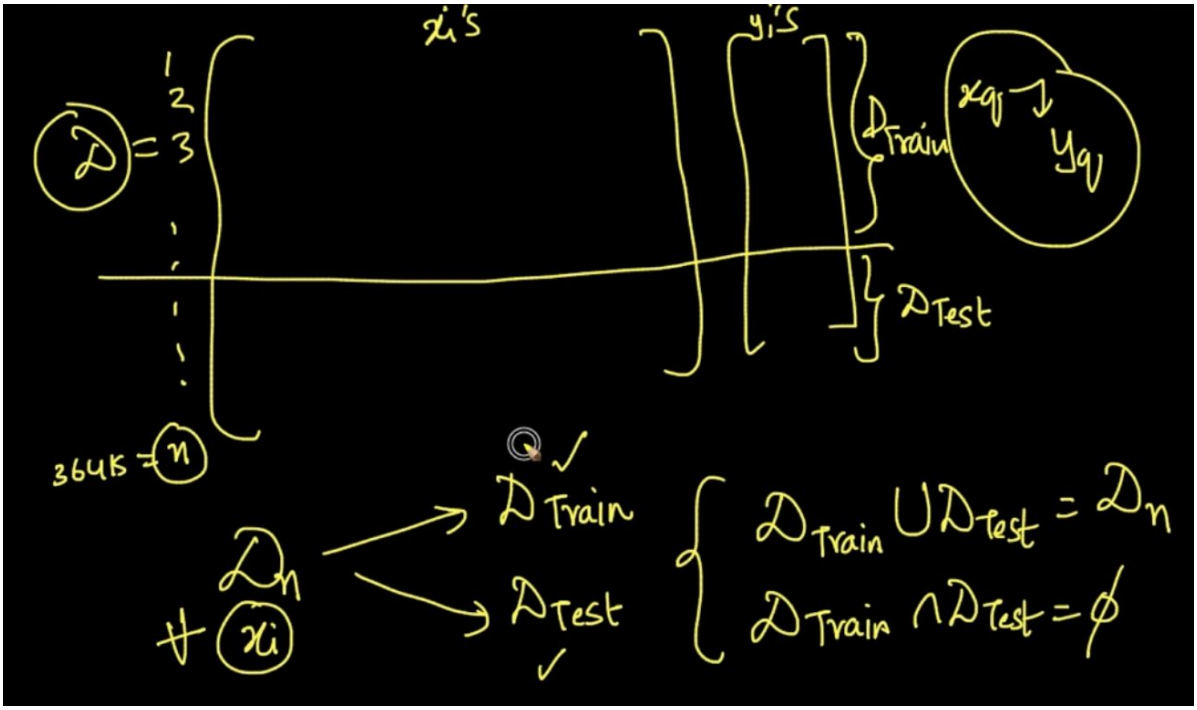



relationship b/w euc-dist & cos-sim
(cos-dist)
- if x_1 & x_2 are unit vect
θ = angle b/w x_1 & x_2
cos-sim
$[\text{euc-dist}(x_1, x_2)]^2 = 2(1 - \cos(\theta))$
cos-dist
$= 2\ \text{cos-dist}(x_1, x_2)$

HOW GOOD “k-NN” IS

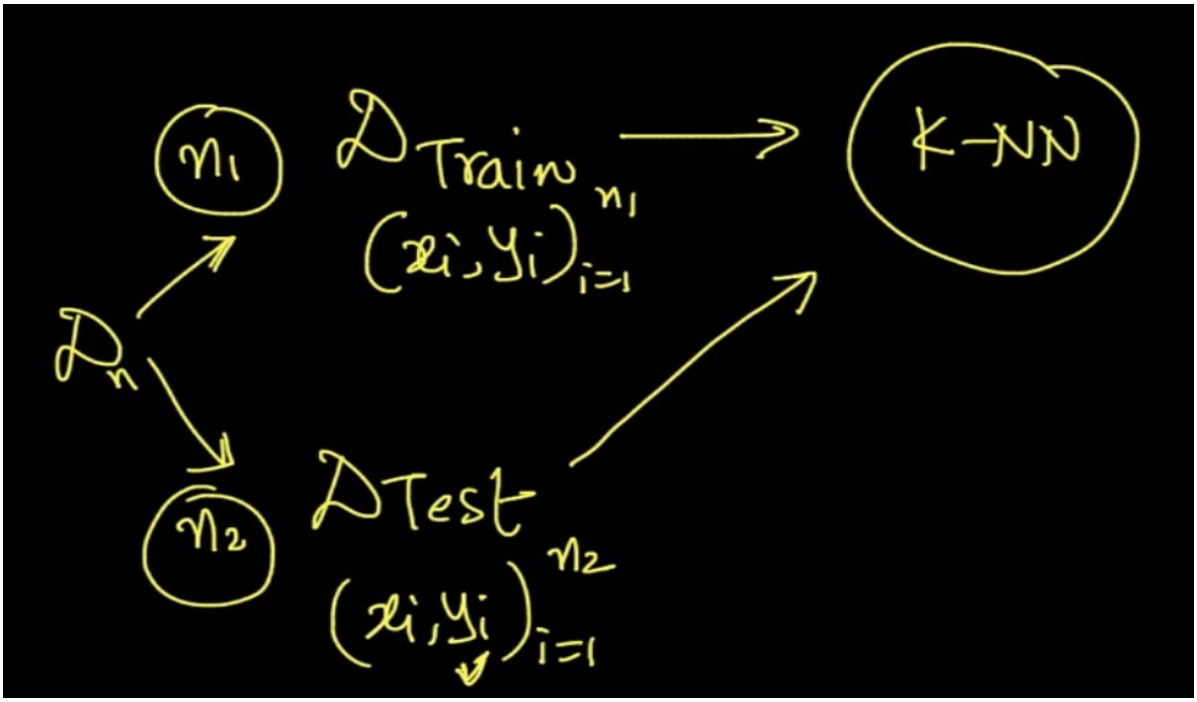


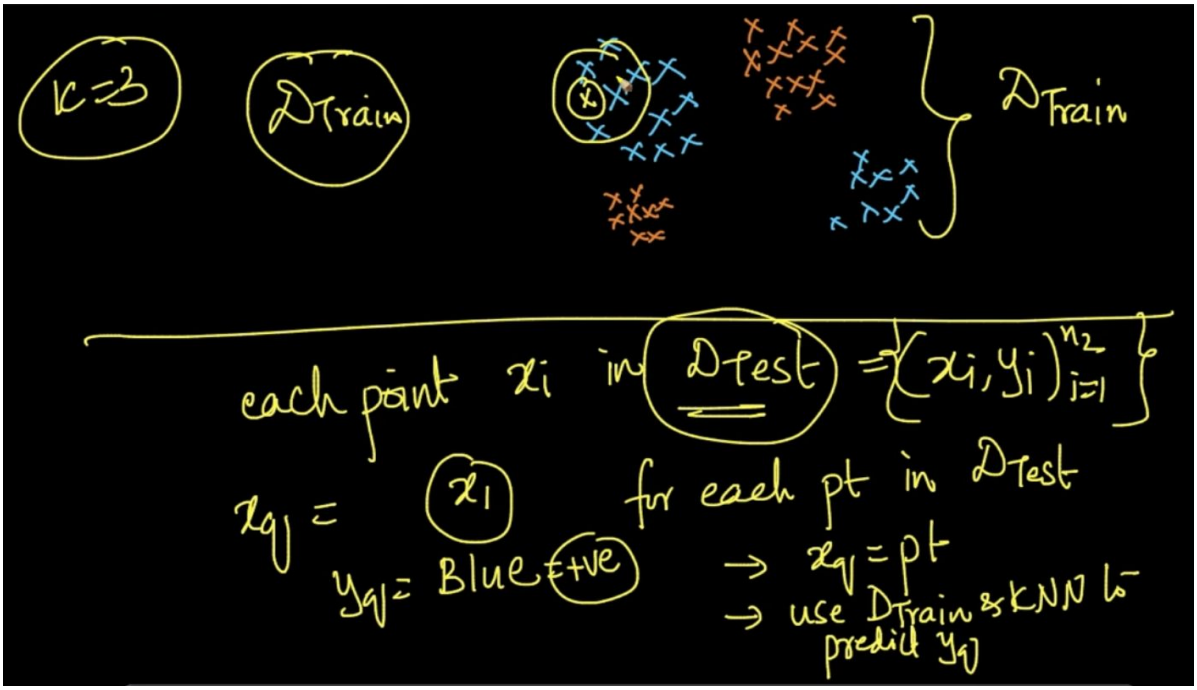
When we get a new review r_q, we use it’s text and convert it into vector x_q



We split our data into two parts: Test and Train. How’s the split done? Randomly

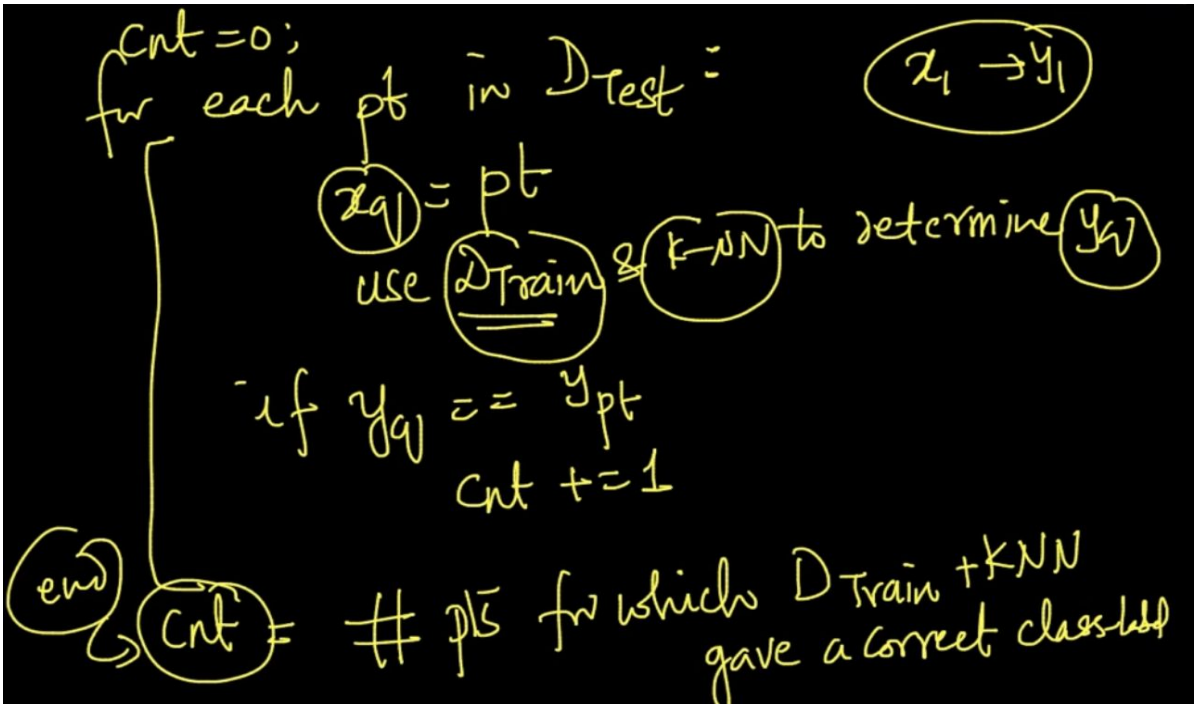
Normally, 70% is in Train and 30% is in Test.



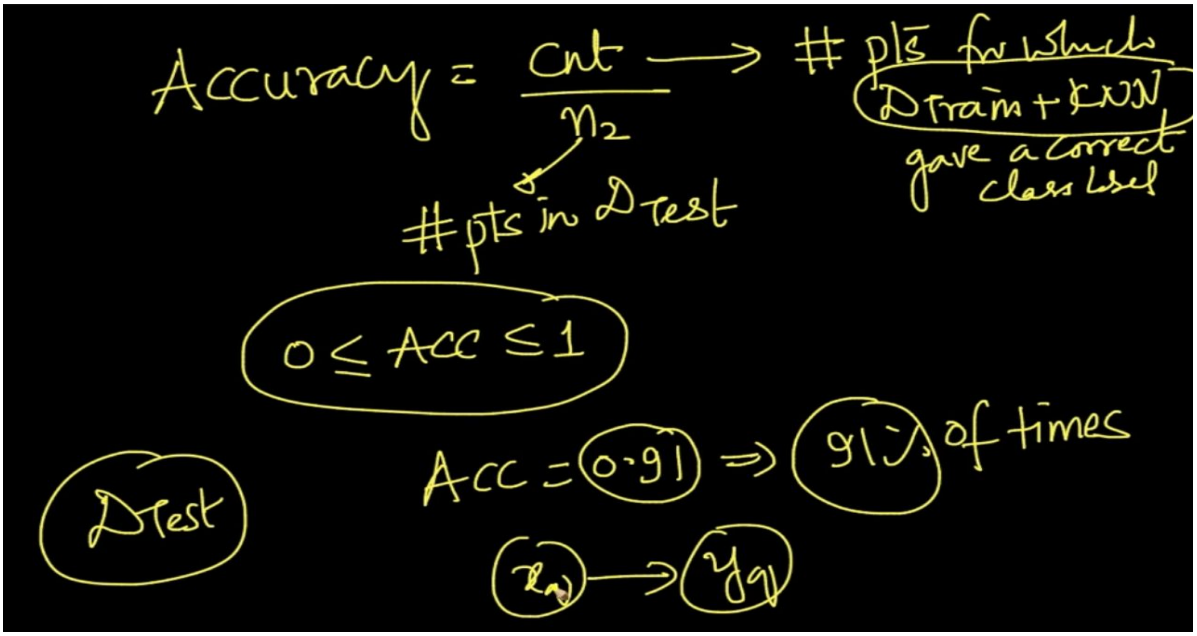


We take each data point from our Test data and use Train data and K-NN to predict y.

As seem above x1 lies near blue points and therefore y = Blue

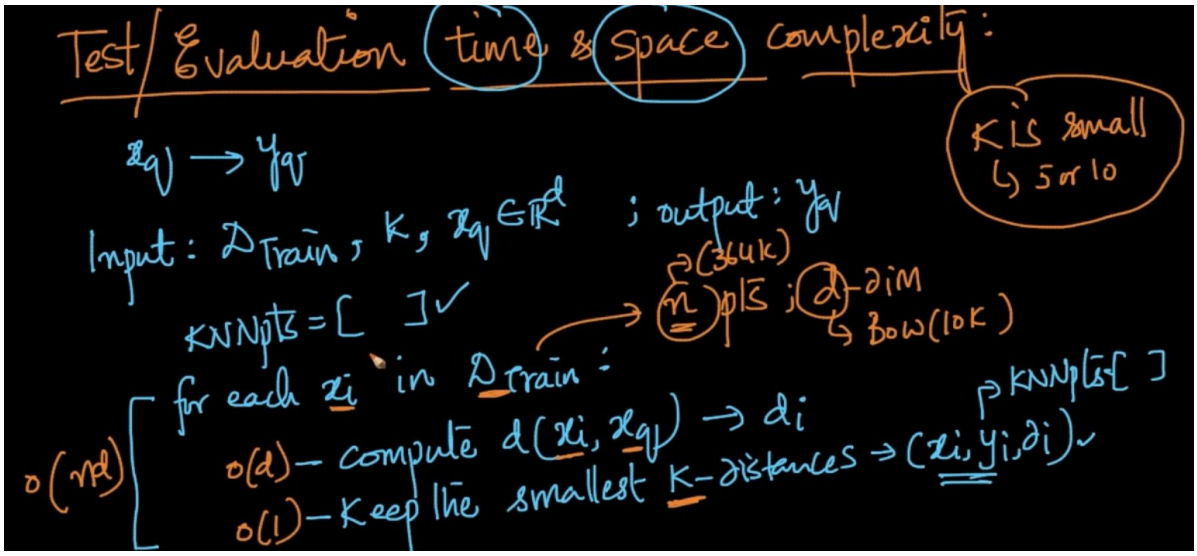


We take point from the test data. Store that in a variable x_q and use Train data and k-NN to determine y_q. We get the count of points for which Train data + k-NN gave correct answer



So accuracy is a measure of how effectively our k-NN is working for our train data.

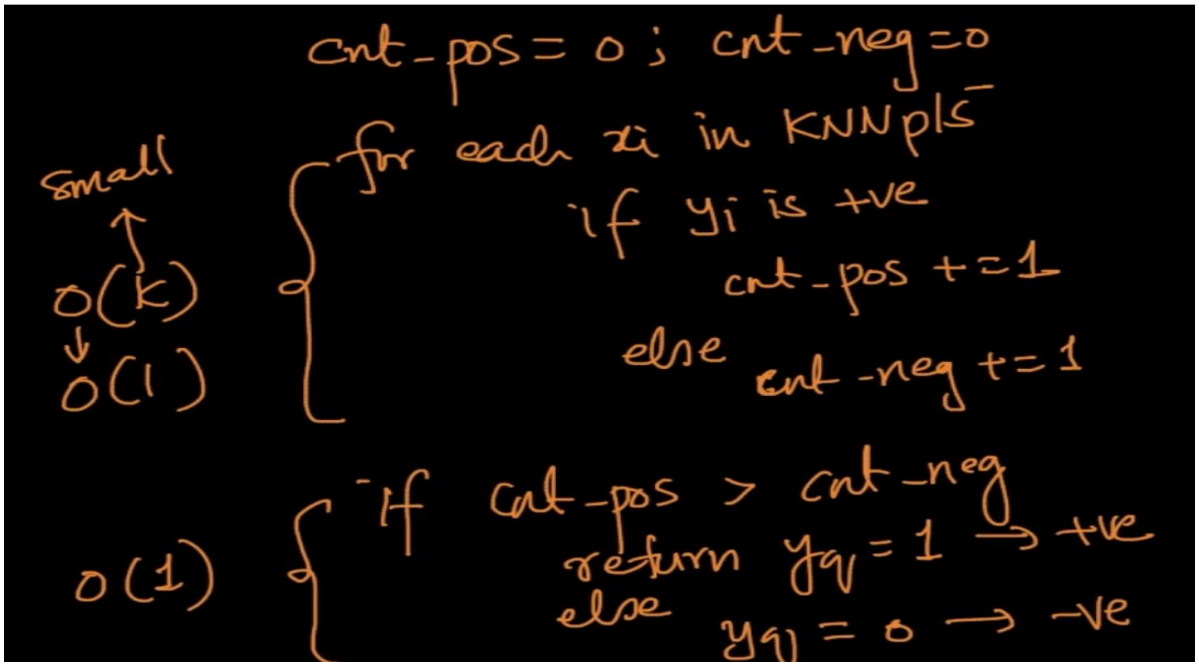
EVALUATION TIME AND SPACE COMPLEXITY



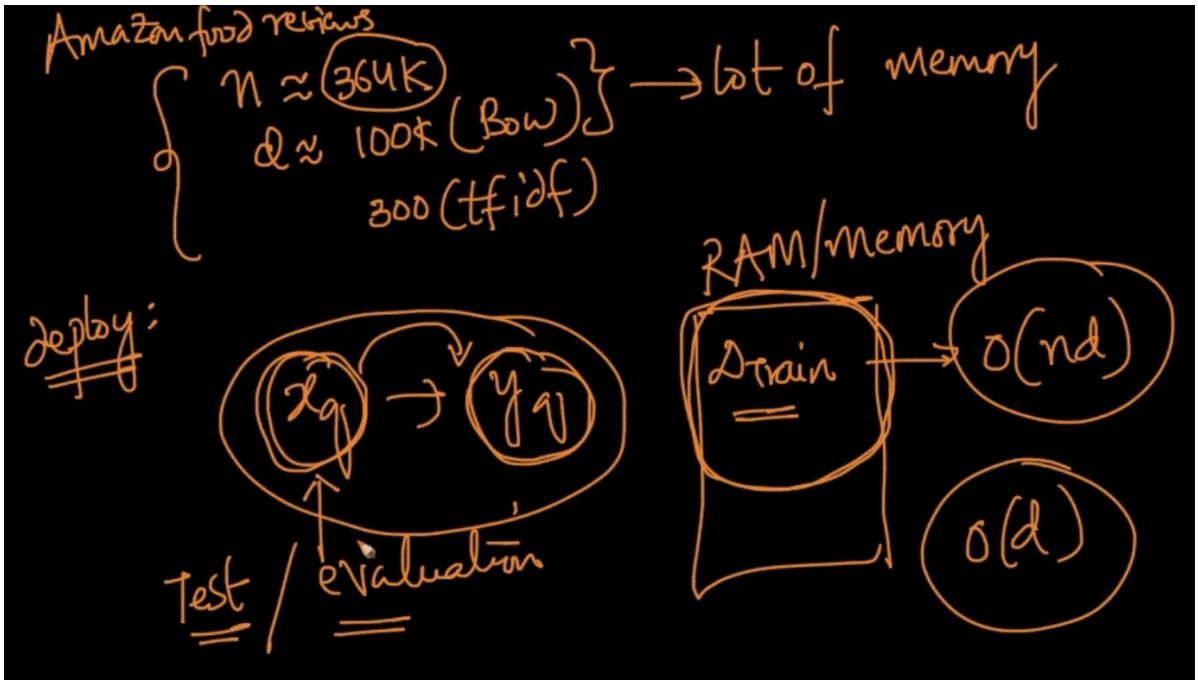
Inputs : Train data, k, new query $x_q \in \mathbb{R}$ with d -dimensions

We get a list of -nearest points after running the above loop . The time complexity is O(n*d)

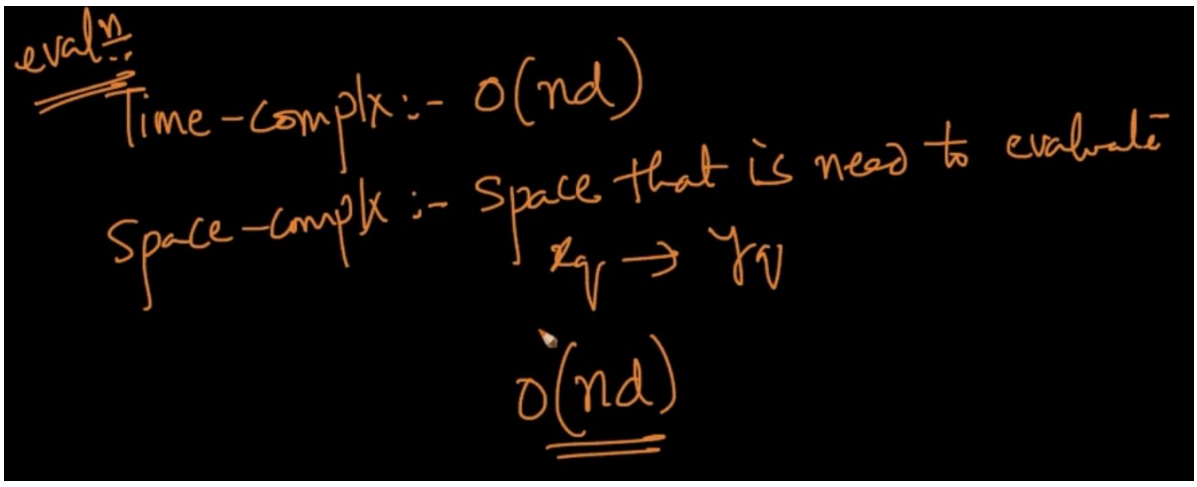
Since each data point have d dimensions and there are total n data points



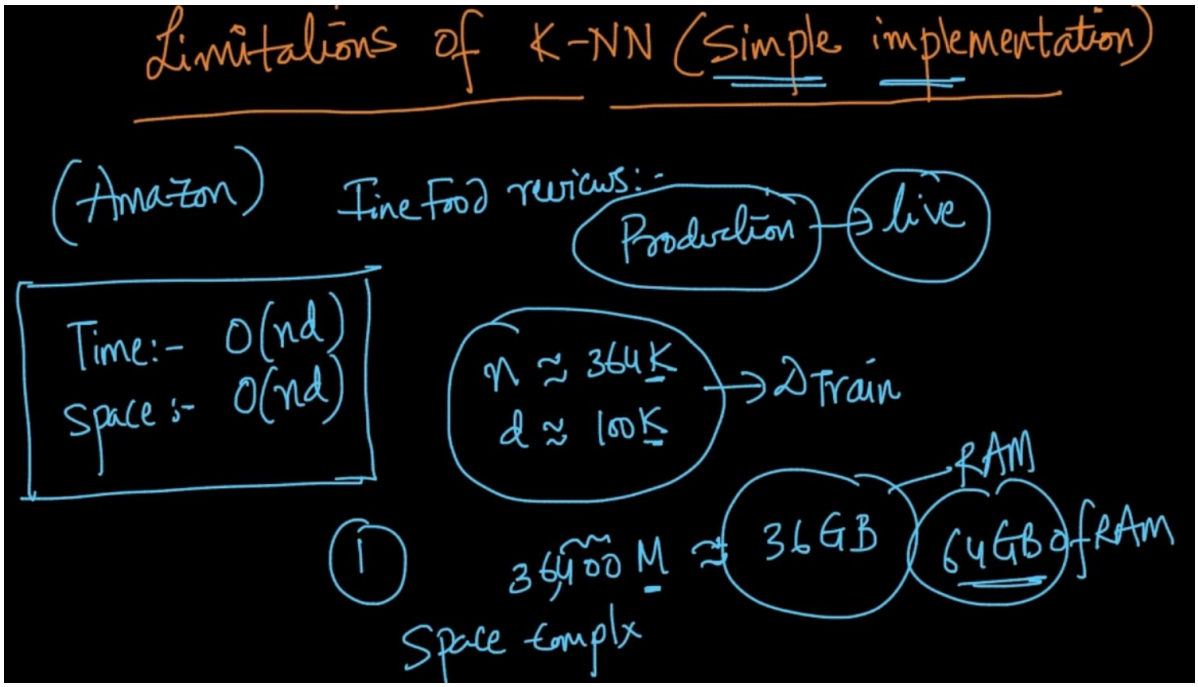
It's time complexity is also small so the only larhe operation is calculating distances O(nd)



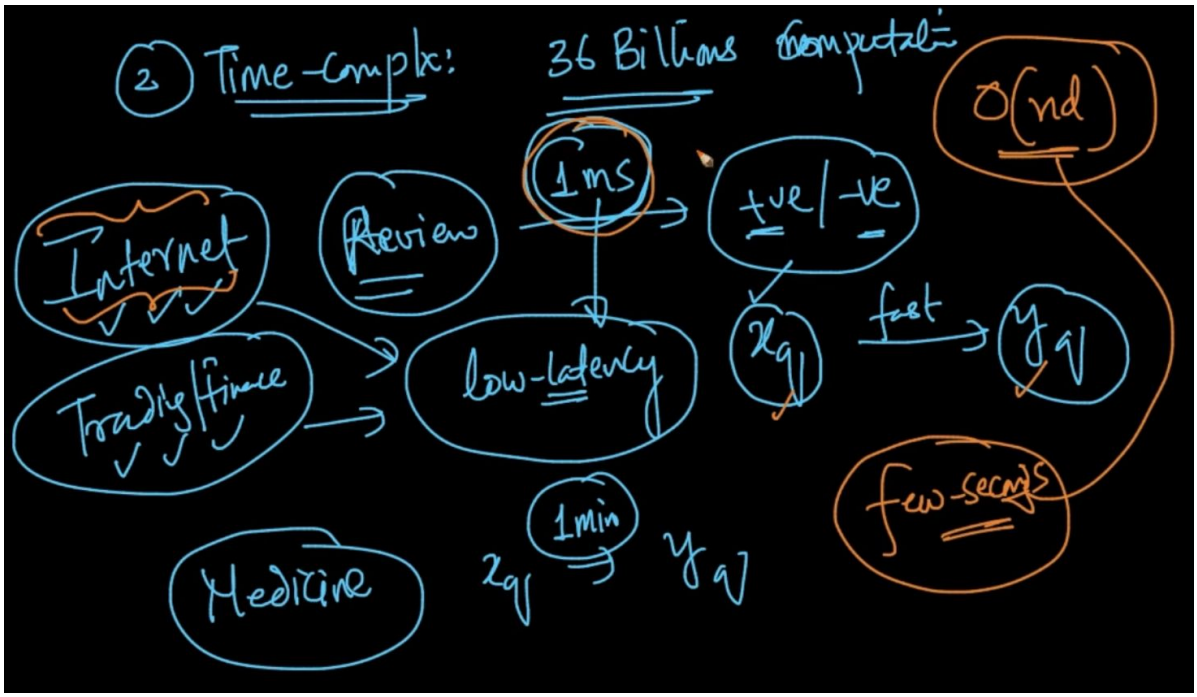
We want y_q given x_q. So, Train data which has O (n*d) is stored in RAM which is a lot at the evaluation time. So O(n*d) is the space complexity as well



LIMITATIONS OF k-NN

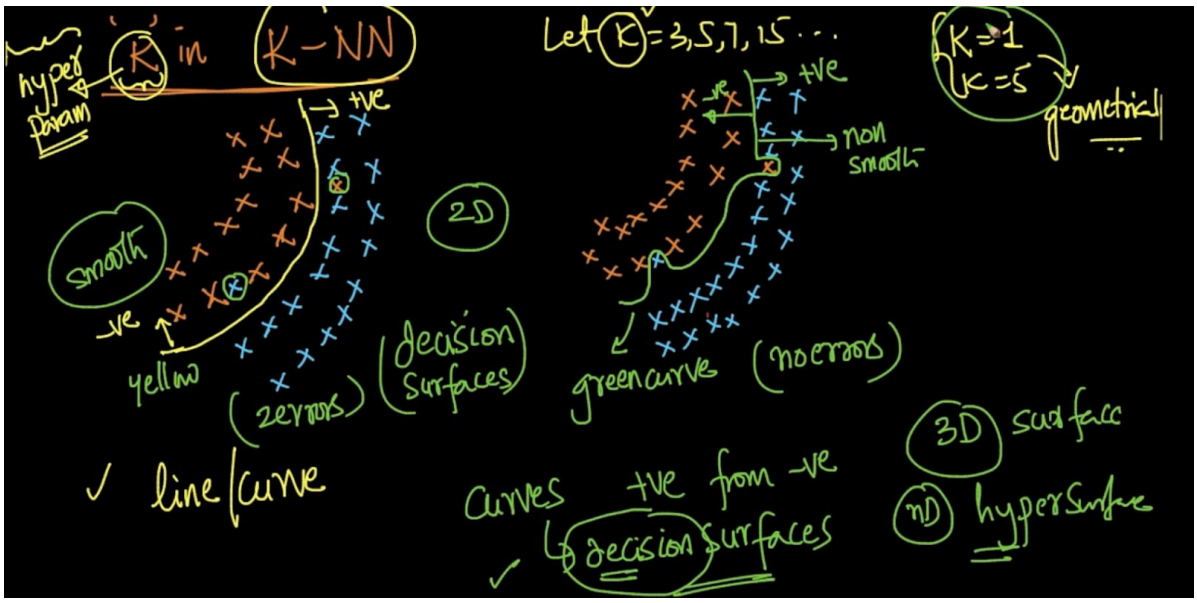


Suppose we want to implement Amazon food reviews using k-NN. Since space complexity of k-NN is O(nd) it'll be very large . So it is a limitation since it requires large hardware.

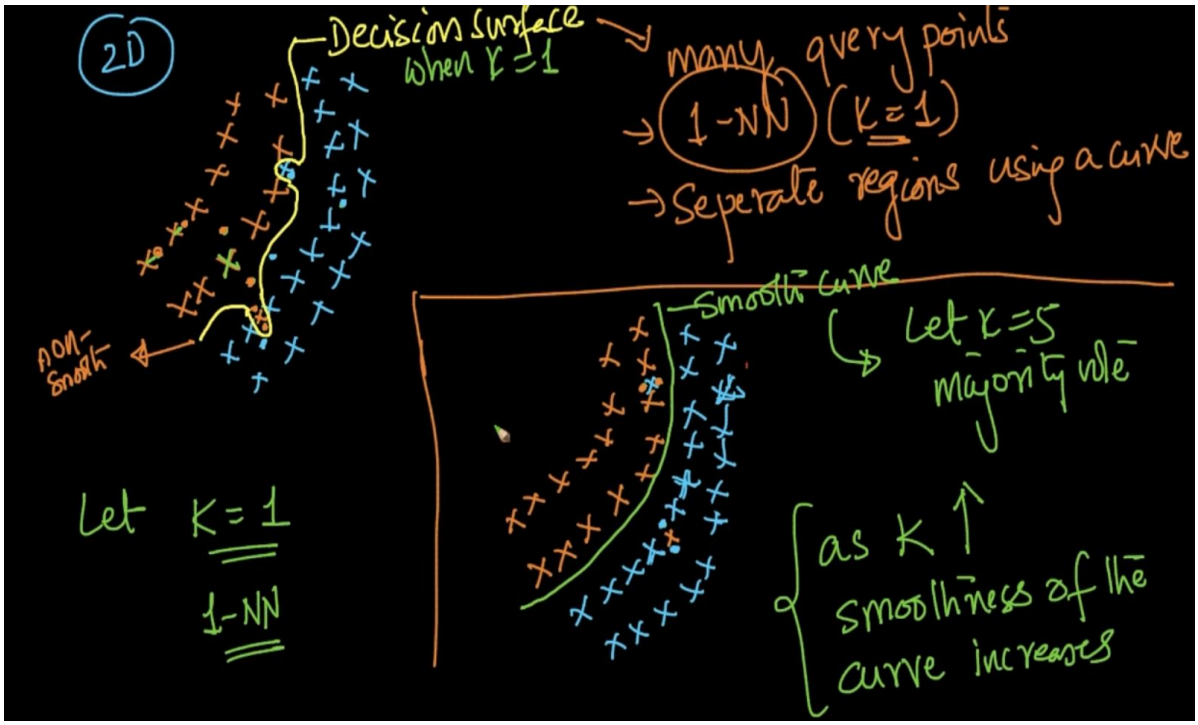


We require a low latency system but with Time complexity O(nd) it gets larger than usual.

DECISION SURFACE FOR k-NN AS ' k ' CHANGES



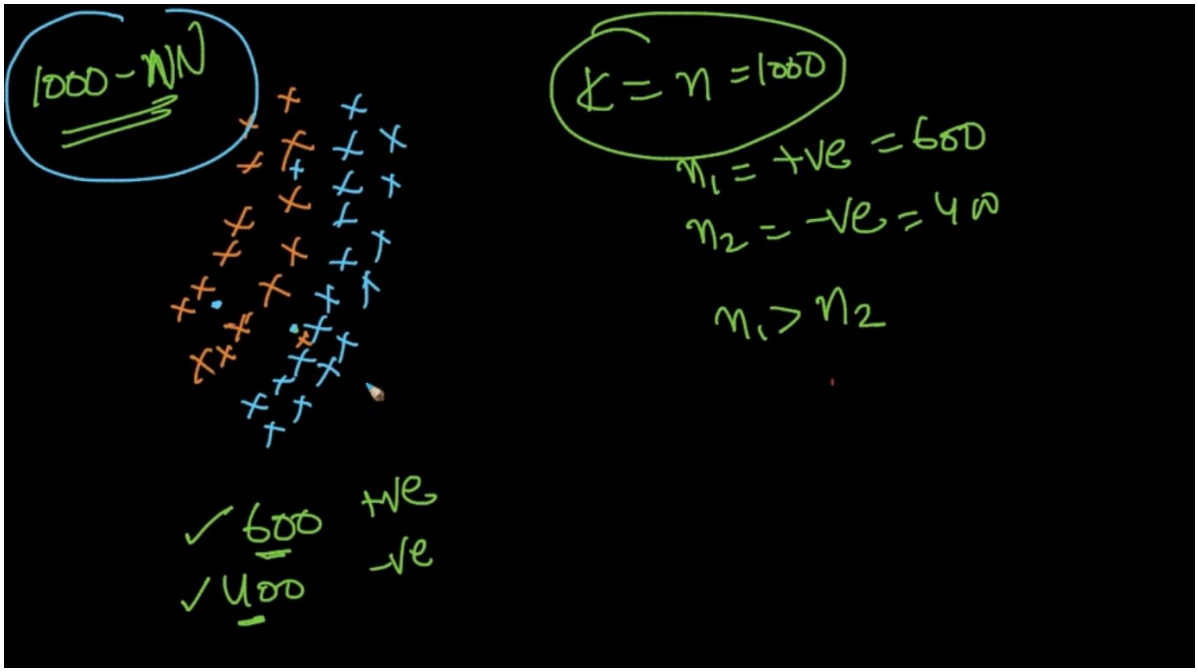
The curve separating the dataset in two parts is decision surface. Let's see how decision surfaces are affected when k changes



For K = 1, if the new query (dotted point) is lying near the orange point which is in the blue Points it'll classify it as Orange.So the decision boundary won't be smooth as seen above

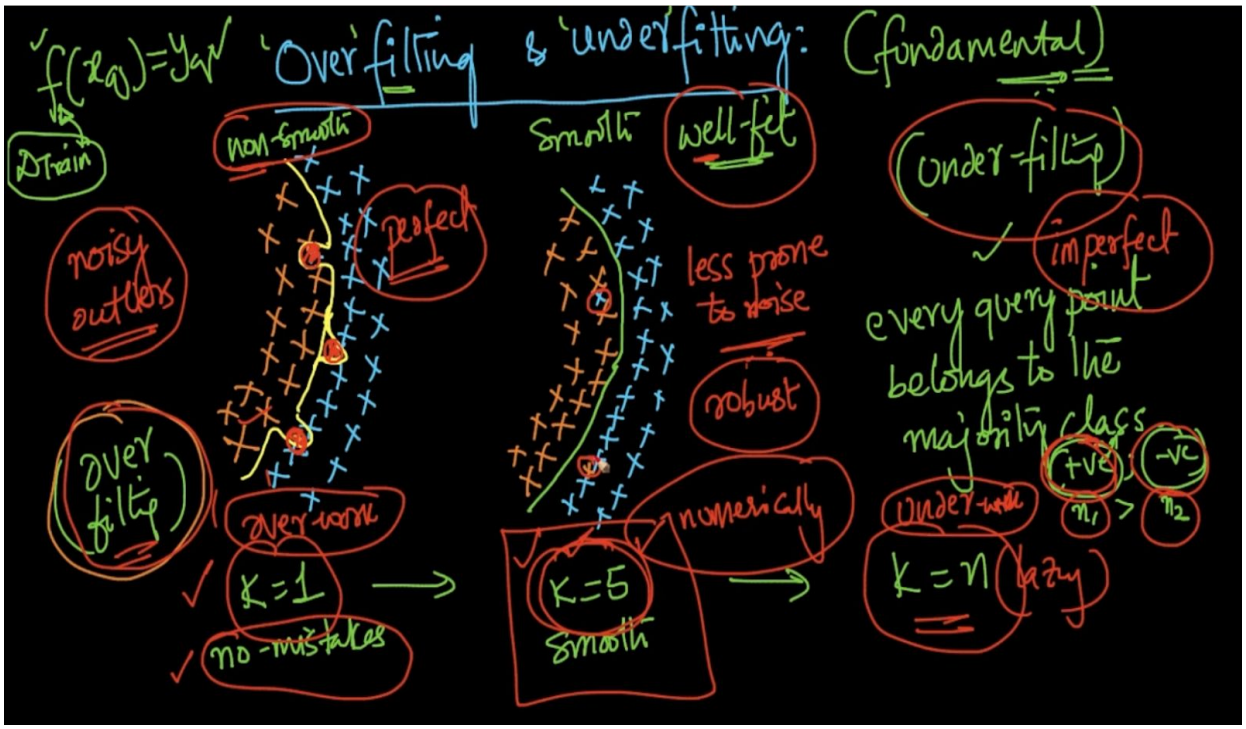
For K = 5, if the same case as above happening it'll label it as Blue since majority vote will be taken . The curve / decision boundary will be smooth .

If K = n, where n => total no. of points and n1 = 600 +ve points(Blue) and n2 = 400 -ve points (Orange)



In the new query point which is lying at the orange but our k-NN will say it's blue because majority is blue . This is a major issue. So if k increases the curve becomes smoother but if k = n then it'll label any new query as the majority label as seen above

OVERFITTING & UNDERFITTING

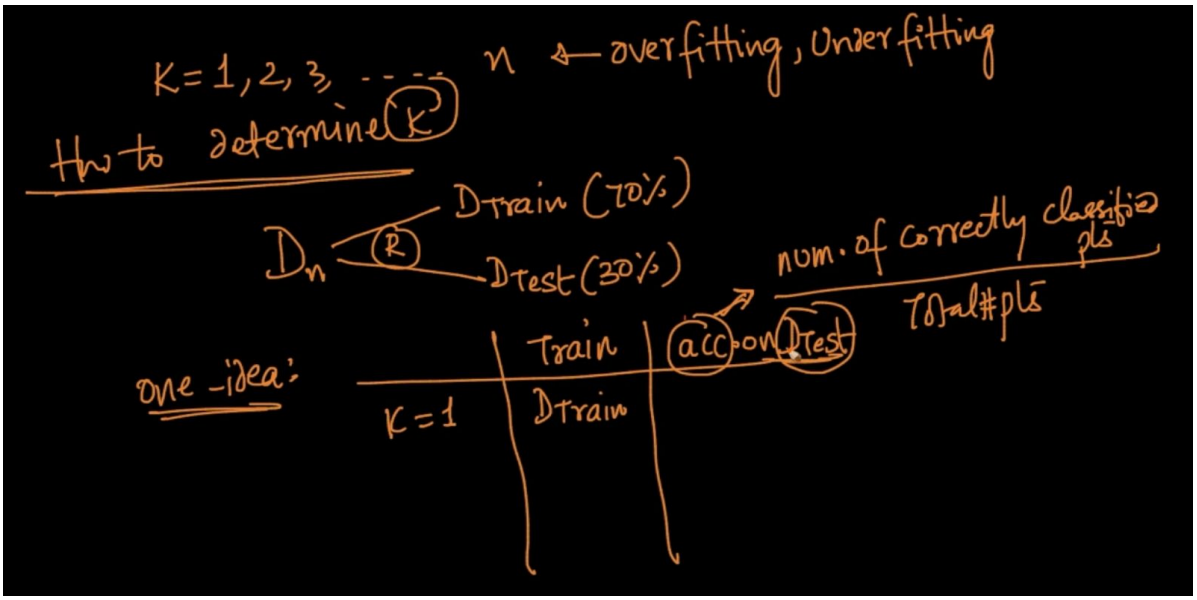


For K = 1, it's trying very hard to make a decision boundary considering all the outliers as seen . It's making a perfect decision tree but non smooth curve. It's overfitting

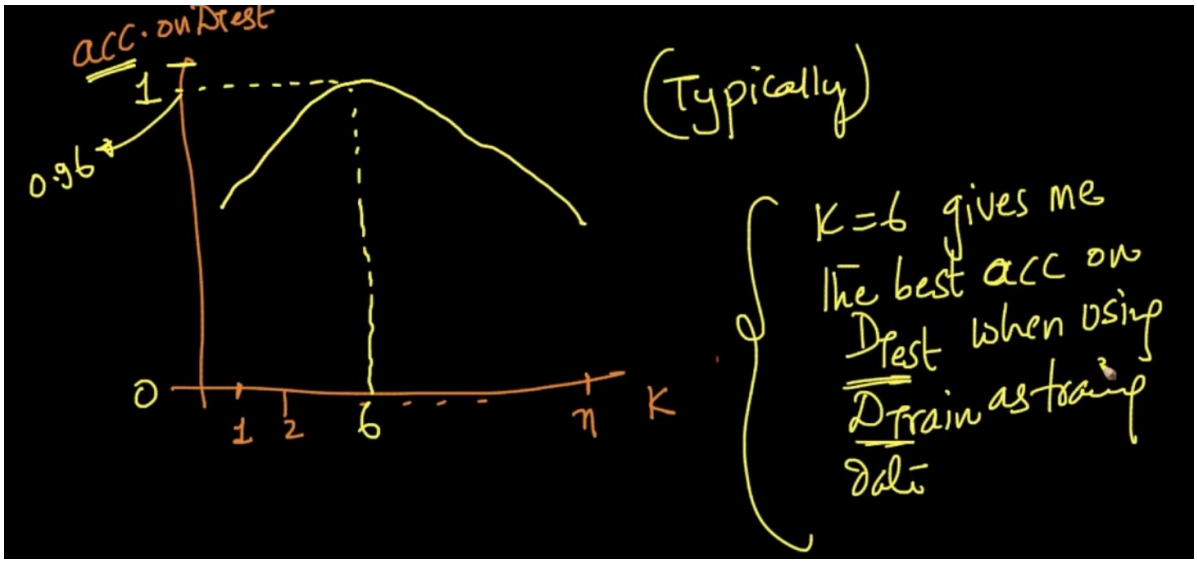
For K = n, it's not trying hard as it's very visible from the above example. It'll classify every query point as the majority class.

For K = 5, it's perfect and smooth . This is what we want in Machine Learning systems

NEED FOR CROSS-VALIDATION

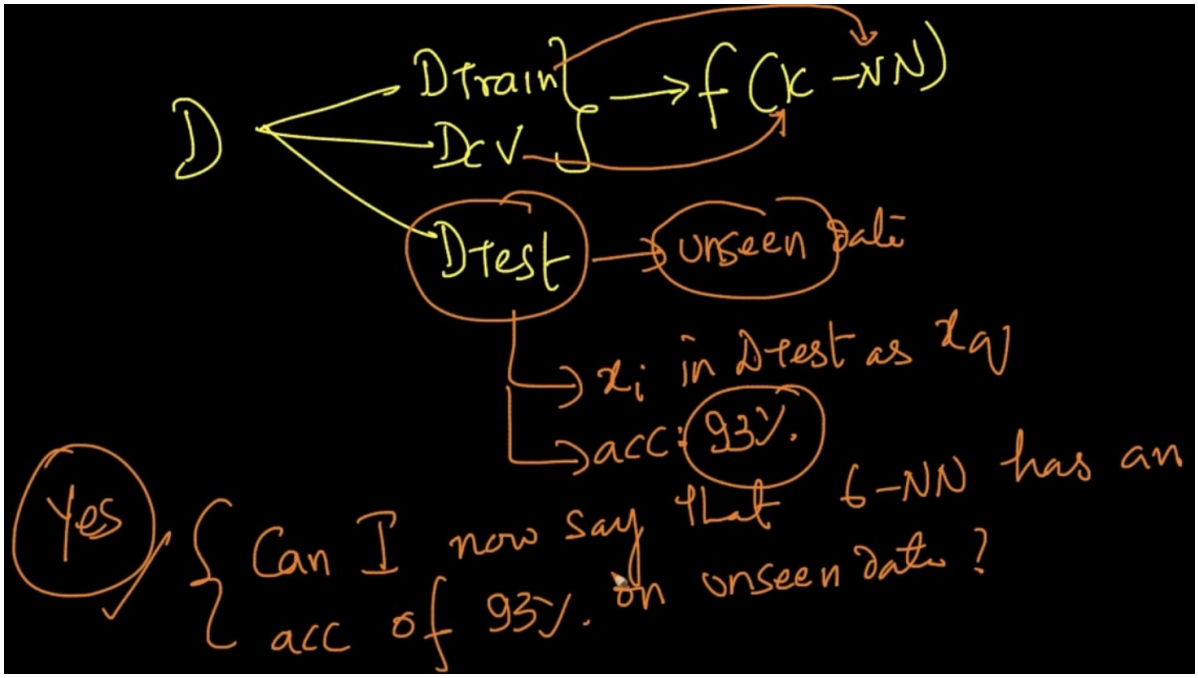


How can we determine 'k'? One idea is to use different K and check accuracy of k-NN on Test data



So, accuracy is checked on Test data with different K and K = 6 gave the largest accuracy

So our Train data is used for Nearest Neighbors and we got K from test data but it should perfectly work on **unseen data.** We predicted 96% on Test data but same thing cannot be said for unseen data.



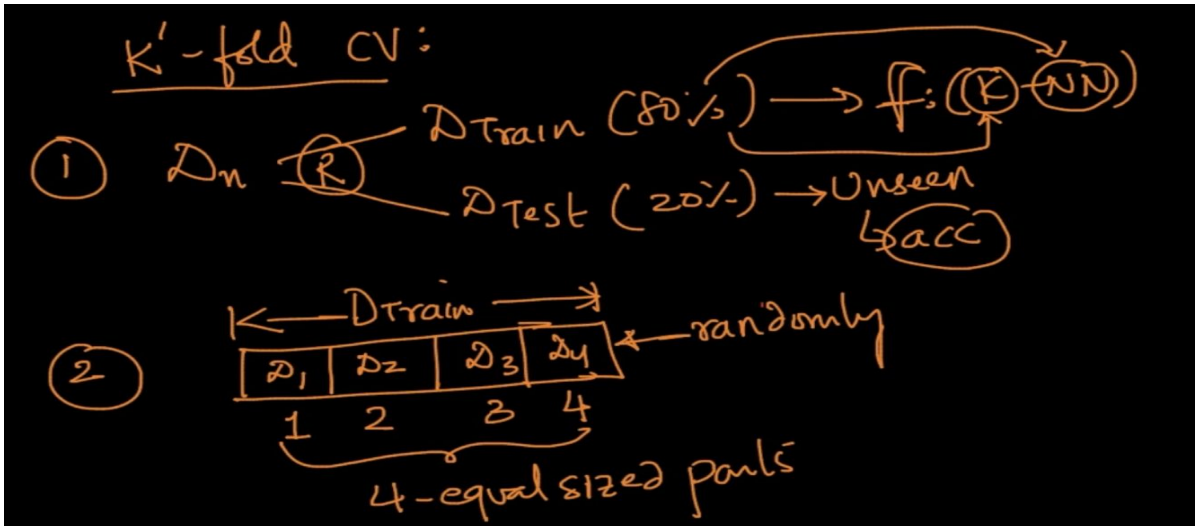
We divided the data into Train and Test and Train is used for Nearest Neighbors and Test is used for getting K but now we are dividing the data into 3 parts.Train,Test and Cross Validation (CV) . Now CV data is used to decide the number of K and then it is tested on Test data (unseen data)

The accuracy we get on the Test data is the accuracy we can use for our prediction.

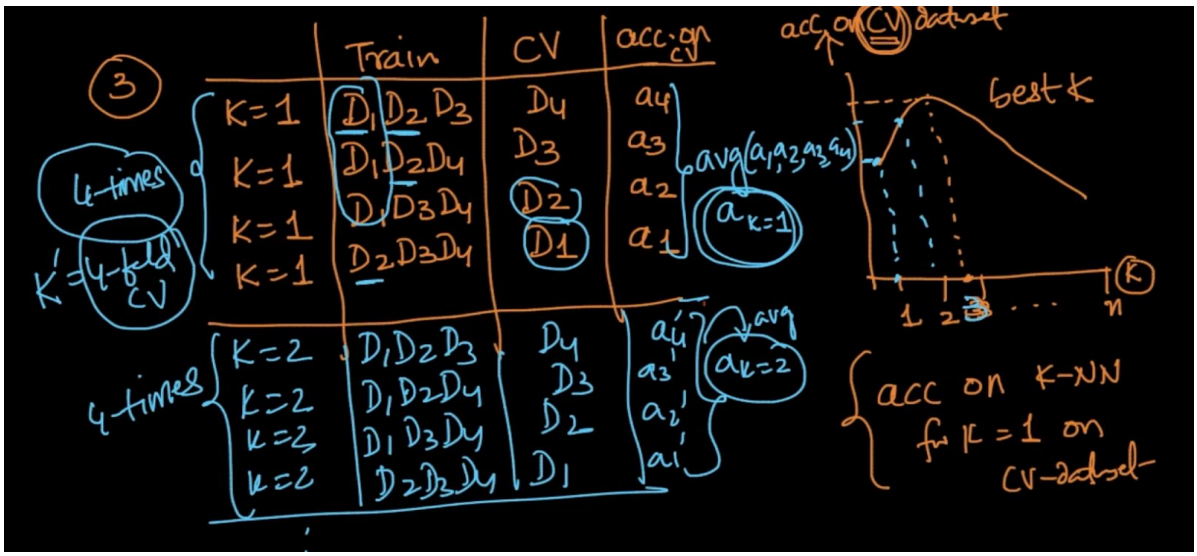
Rule of thumb is 60% is used for Train Data, 20% is used for Test and 20% is used for Cross Validation (CV)

K-FOLD CROSS VALIDATION

We only used 60% of data for Train and 20% for CV and 20% for Test normally but since ML improves with more data we might want to train it for more data. So we use a technique called K-fold cross-validation

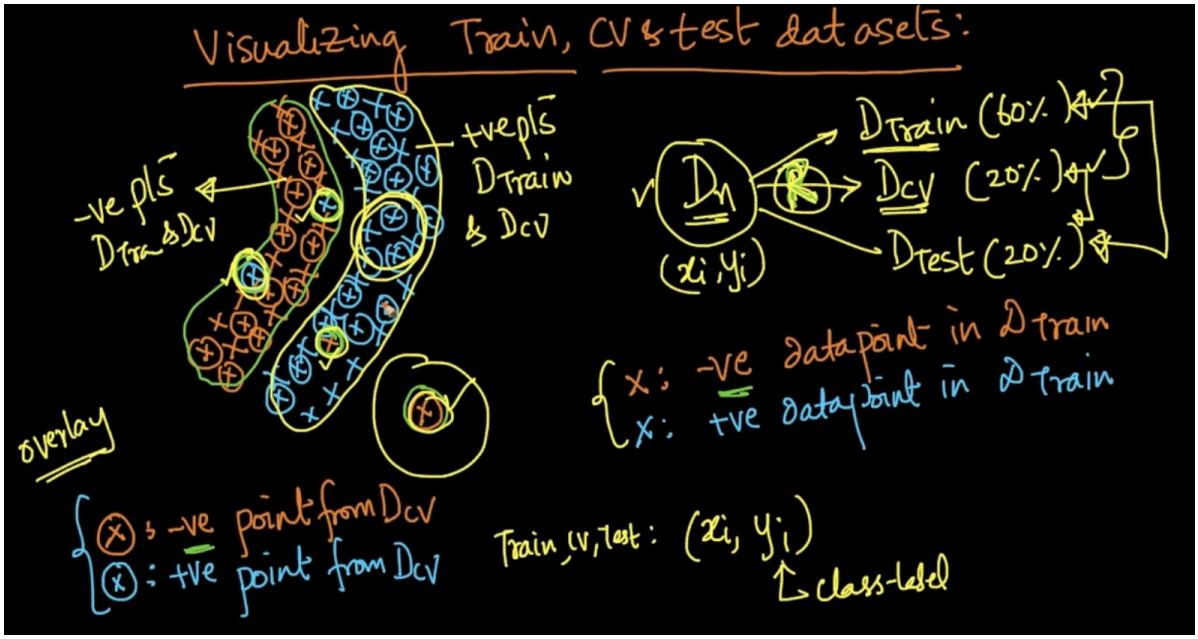


We divide the data into 4 equal sized data after splitting into Train and Test.



So, here we are training it on 3 parts for train and 1 for CV and repeating the procedure 4 times to get accuracy and we get avg of that accuracies. It is 4-fold CV .Here, we are smartly using 80% of the data for Training.Generally , we do **10**-fold CV

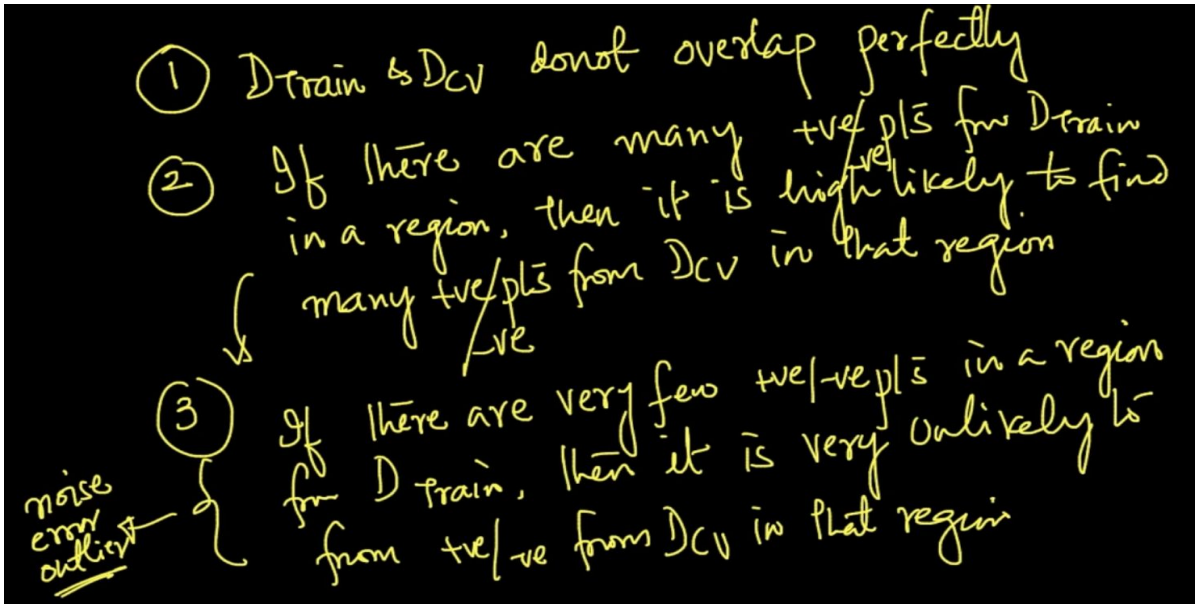
VISUALIZING TRAIN, TEST AND VALIDATION SET

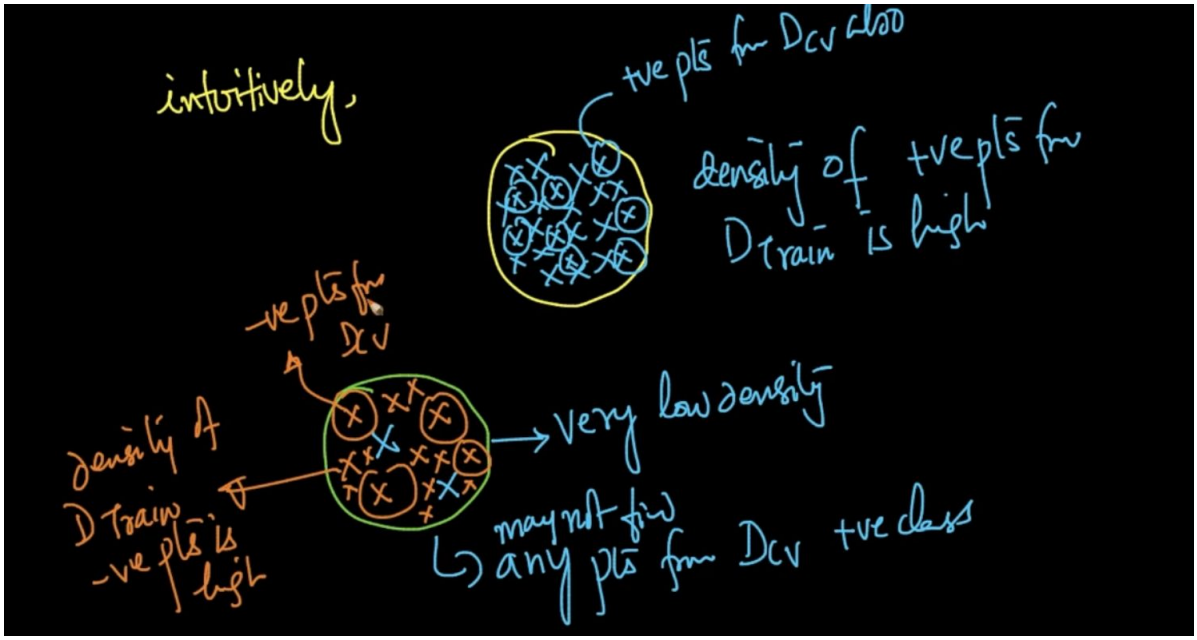


x : -ve datapoint, x : +ve data point in Train . Ⓧ : -ve datapoint from Cross validation

Ⓧ : +ve datapoint from Cross Validation

So the thing is if you're randomly selecting points for Train and CV then they are overlayed as seen i.e more CV datapoints can be found in the corresponding Train data region as -ve CV points are more likely to be found in +ve Train data but there can be points from CV that can appear in region which is not in the Train data region as seen above

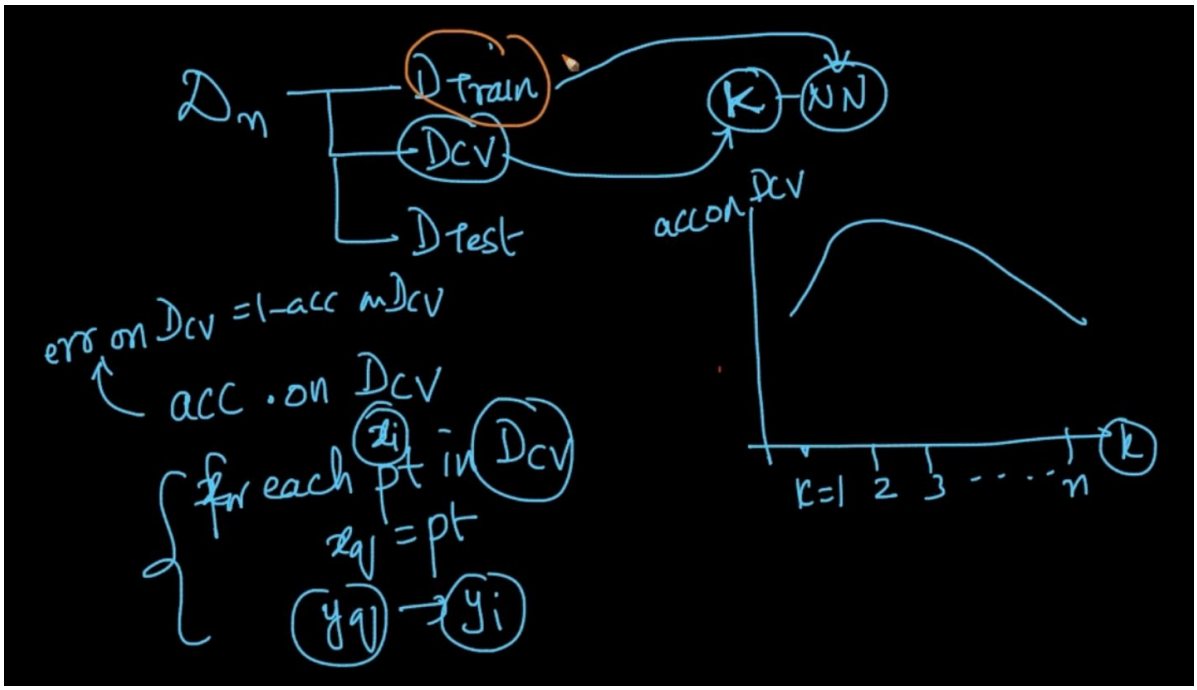




In the upper region +ve points from CV data can be found more but in the below region very low number of +ve data points are there since it's a region of -ve points so the probability of finding +ve CV data points is very unlikely

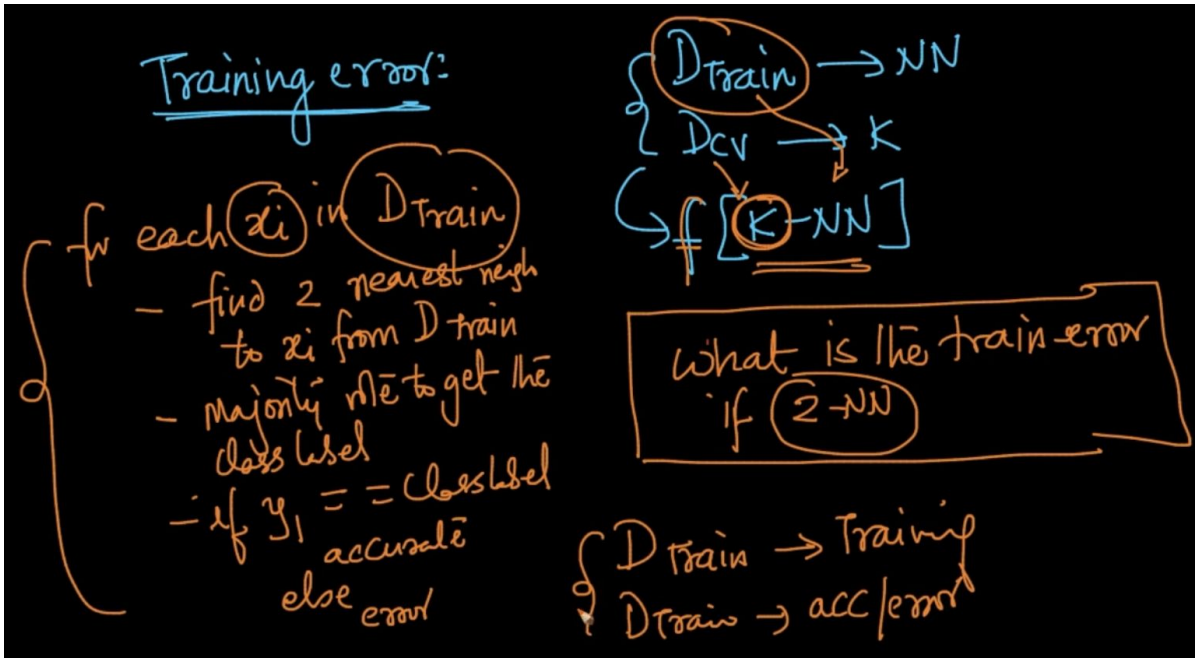
OVERFITTING v/s UNDERFITTING

We've seen that k-Fold Cross Validation is a great strategy to get the best K which is neither under fitted nor overfit but how can we be really sure that we are not overfitting or underfitting?

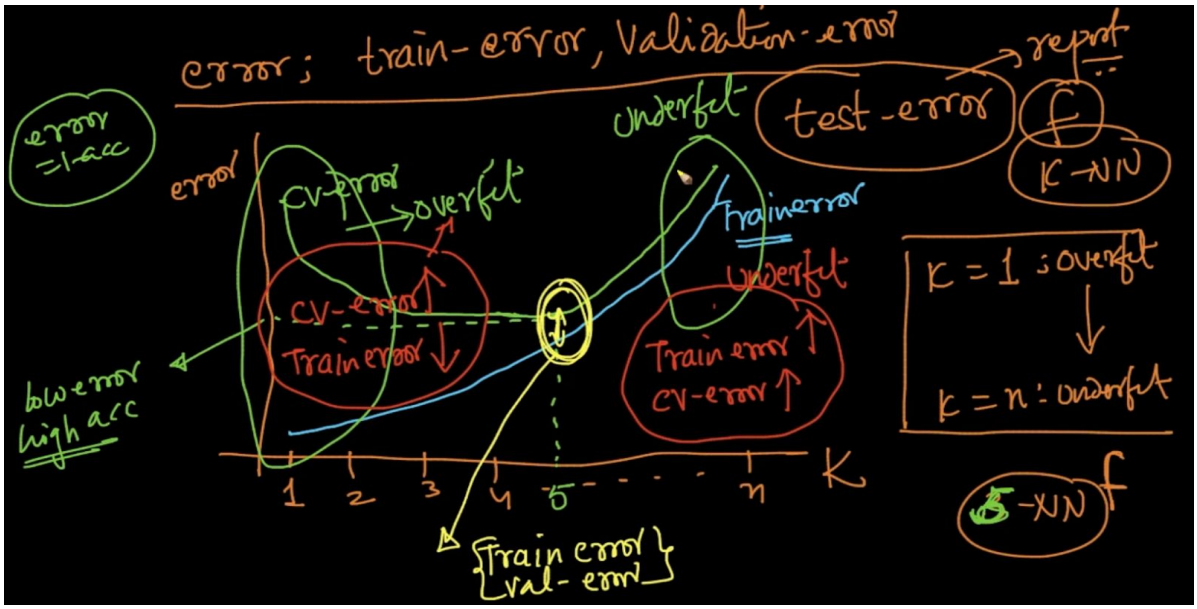


We know how to get the accuracy on Cross Validation data and CV error = 1 - accuracy

For getting accuracy on CV data. For each point x_i on CV data we store that on x_q and we get y_q and see if they match with y_i of x_i. It is trained on Train data. If it matches then accurate otherwise error. This is Cross Validation/Validation error.



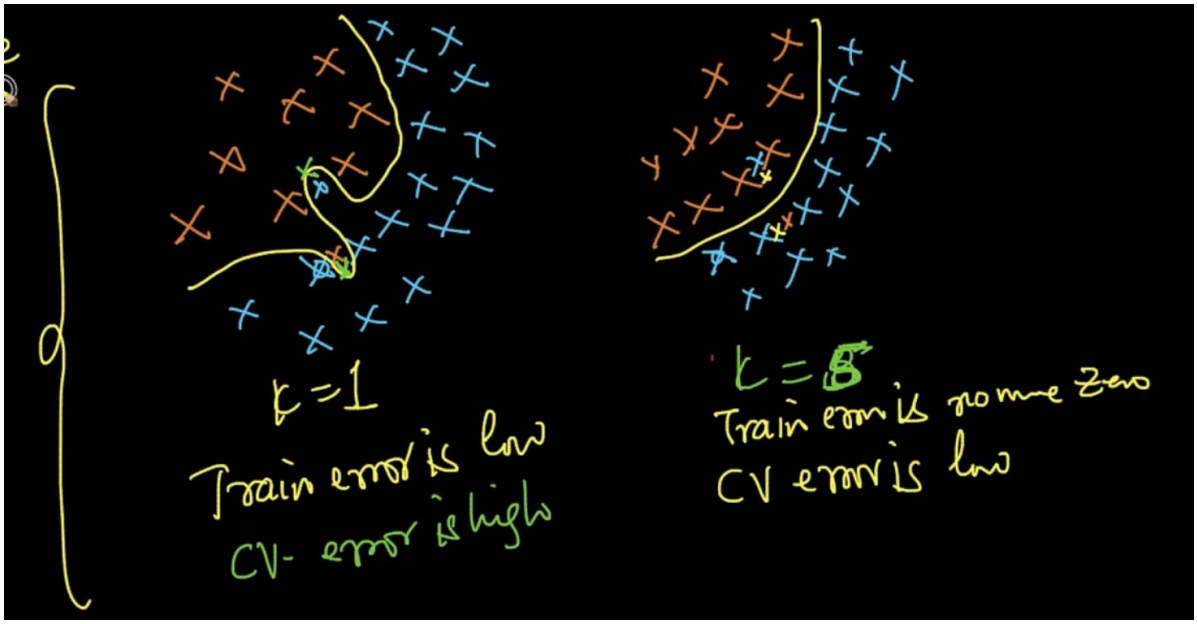
Train error : We get the accuracy/error on Train data by applying k-NN it on Train data



This is error-K graph .

Train error starts from low error and gets high with increase in number of K and CV error's behavior can also be seen . If the Train error is low and CV error is high then overfit.If both are high then Underfit. So we select the tradeoff where the error is at both the gap is min

NOTE : If the Train-error, CV - error ↑ then Underfit and if CV error ↑ , Train error ↓ overfit

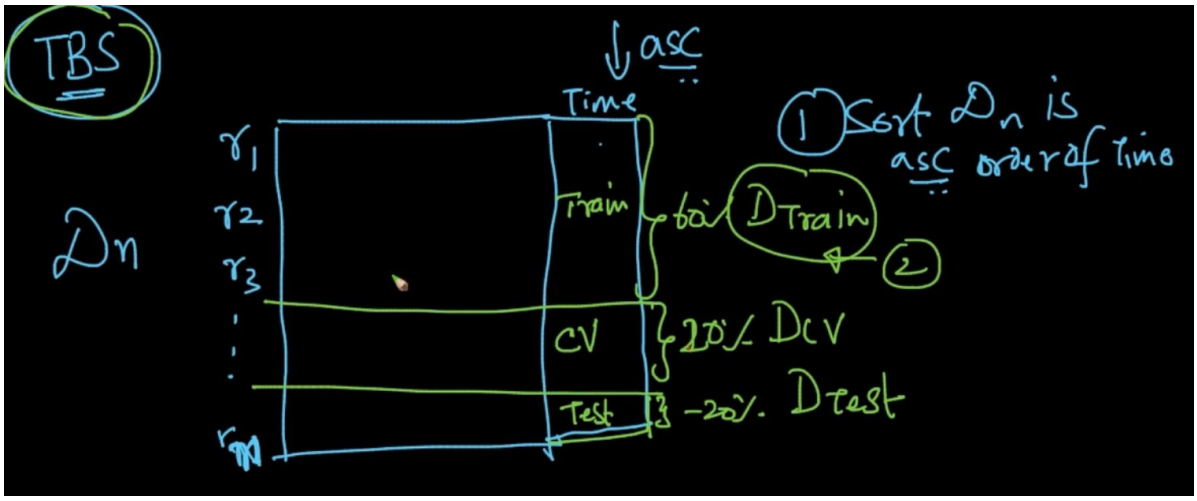


At K = 1, Train error is low because it is over fitting but CV error can be high as because if a point lies near the orange point which is in the blue region then it'll give orange which is error.

At K = 5, it'll work properly .Train error is no more low but CV error is also low

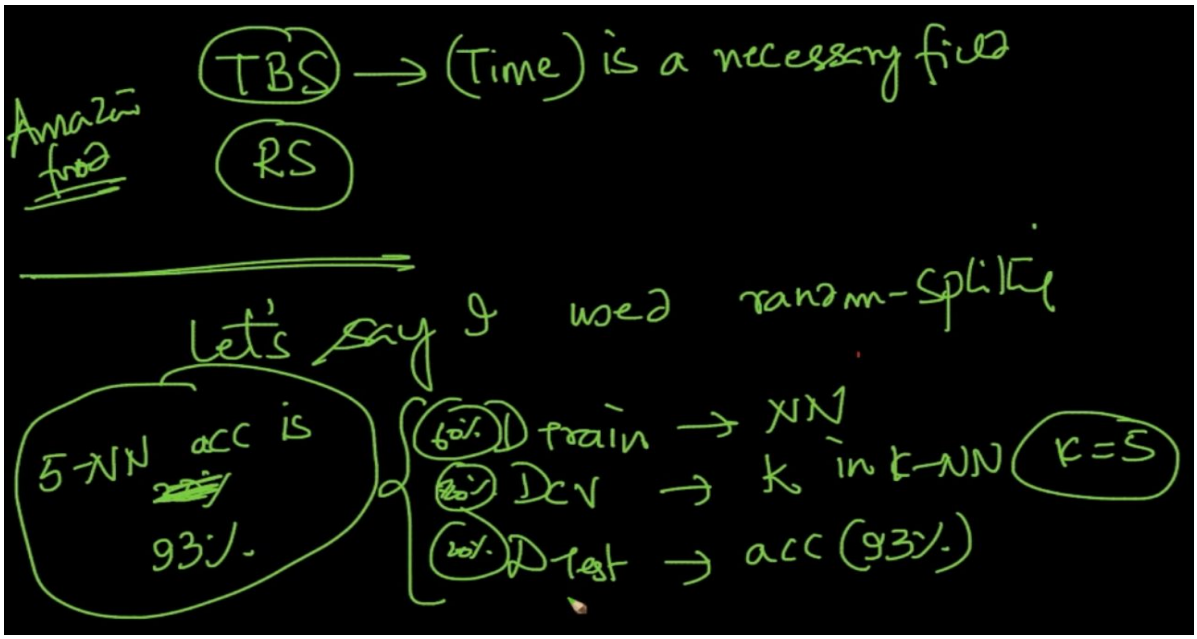
TIME-BASED SPLITTING

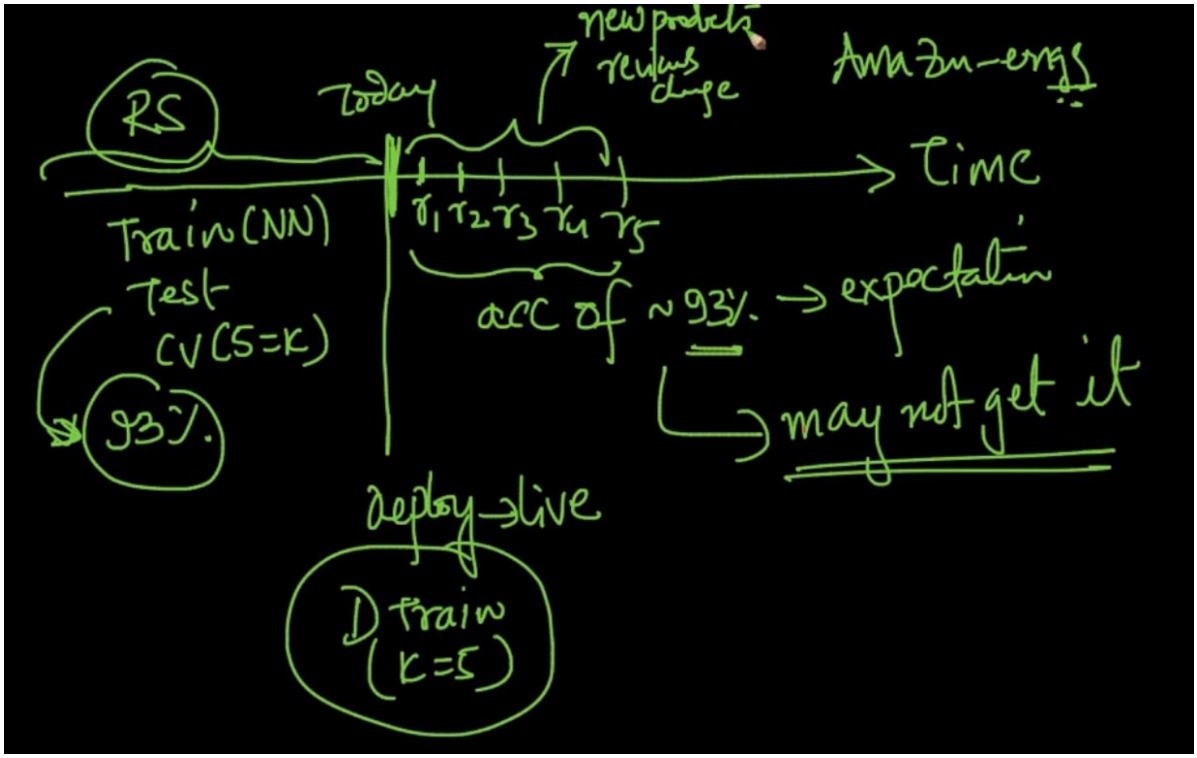
Till now we've only seen random splitting of a dataset in which any data point can go into any split i.e Test, Train or CV but there are other approaches to it as well . One is Time based splitting. If there is timestamp given in our dataset the TBS is used



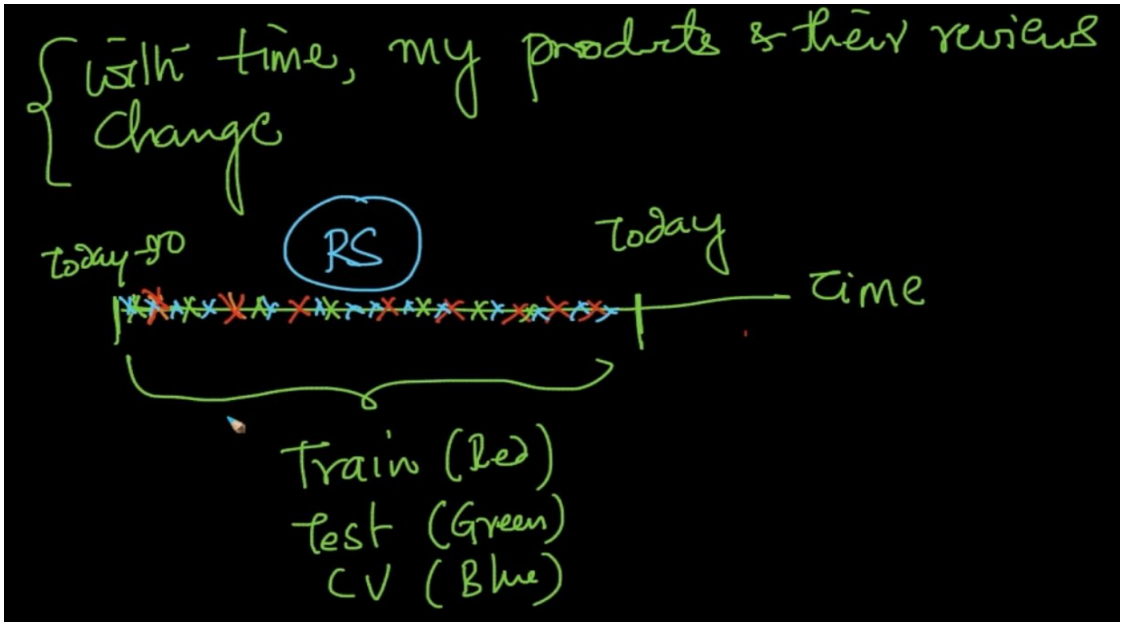
In TBS ,we sort the data with Time and then Train data is 60% of the oldest data, CV data is less old 20% (recent than Train because sorted by time) and Test is the most recent data

Let's see why this is important by using our Amazon Fine Food Reviews

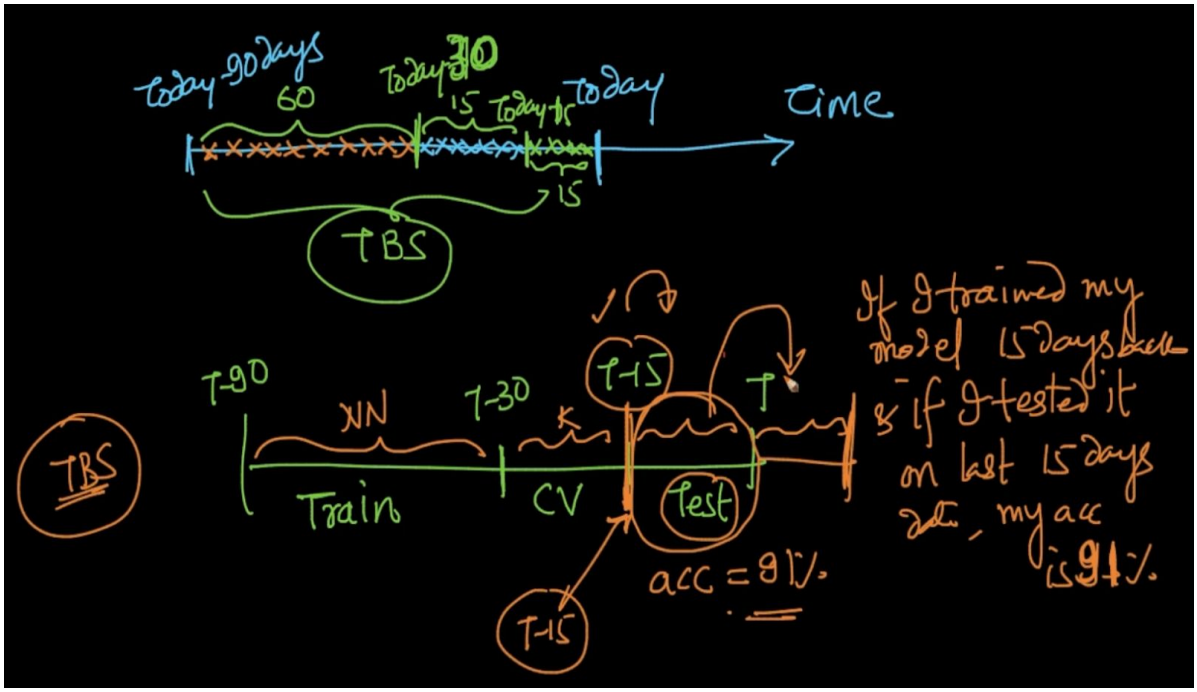




In Random Splitting, if we get 93 % accuracy on Test data till today and if new reviews are getting added we'll be expecting a 93% accuracy on new reviews but we may not get it since reviews get changed overtime obviously



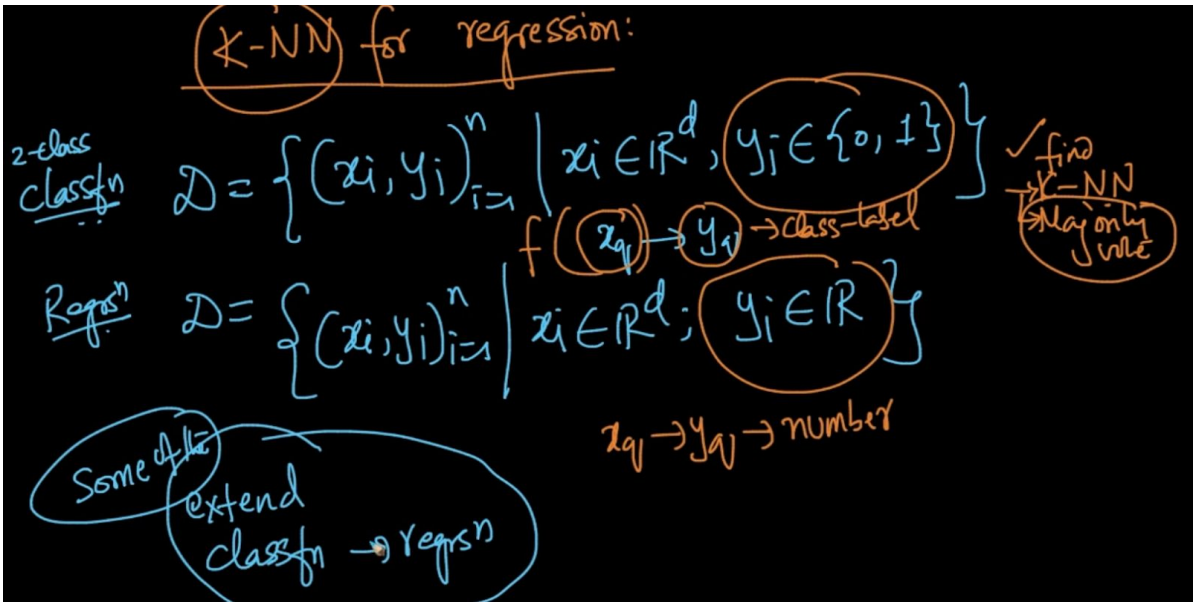
In RS our Train,Test and CV data has data from all the parts



Here we are using the last 90 days data and oldest 60 days data is used for Train, 15 days for CV and the latest 15 days data in Test and if we are getting 91% accuracy on Test then we can say that 91 % accuracy can be expected in future reviews as well.

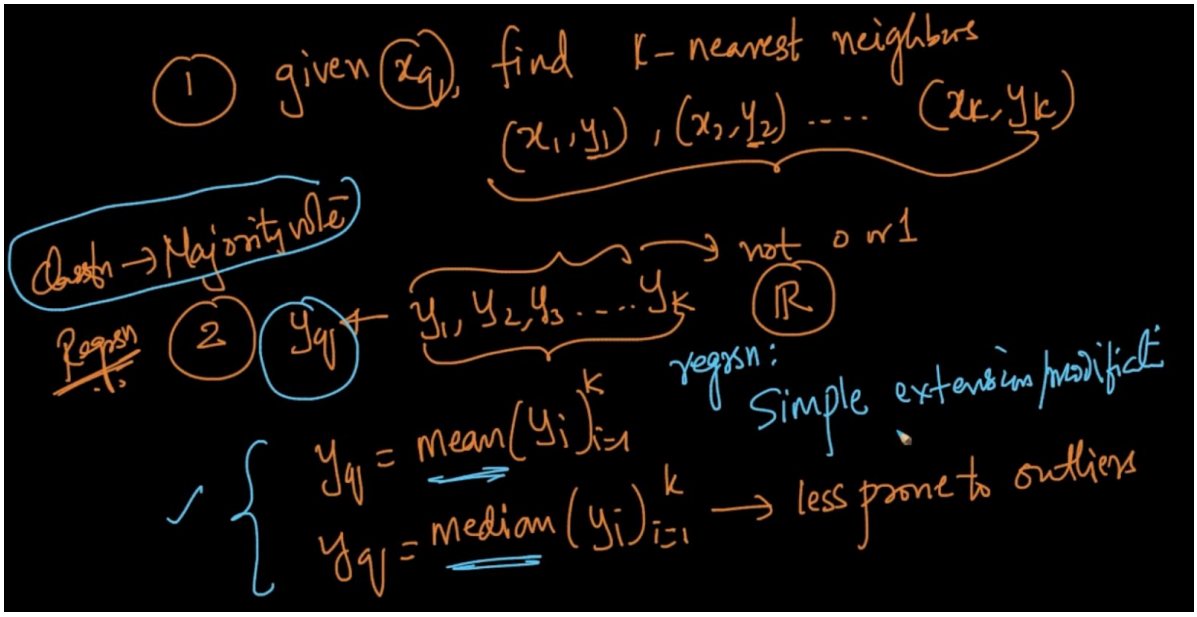
** **NOTE** - Whenever Time is available and if data/ behavior of data changes over time (e.g reviews) the Time based splitting is preferable

k-NN FOR REGRESSION



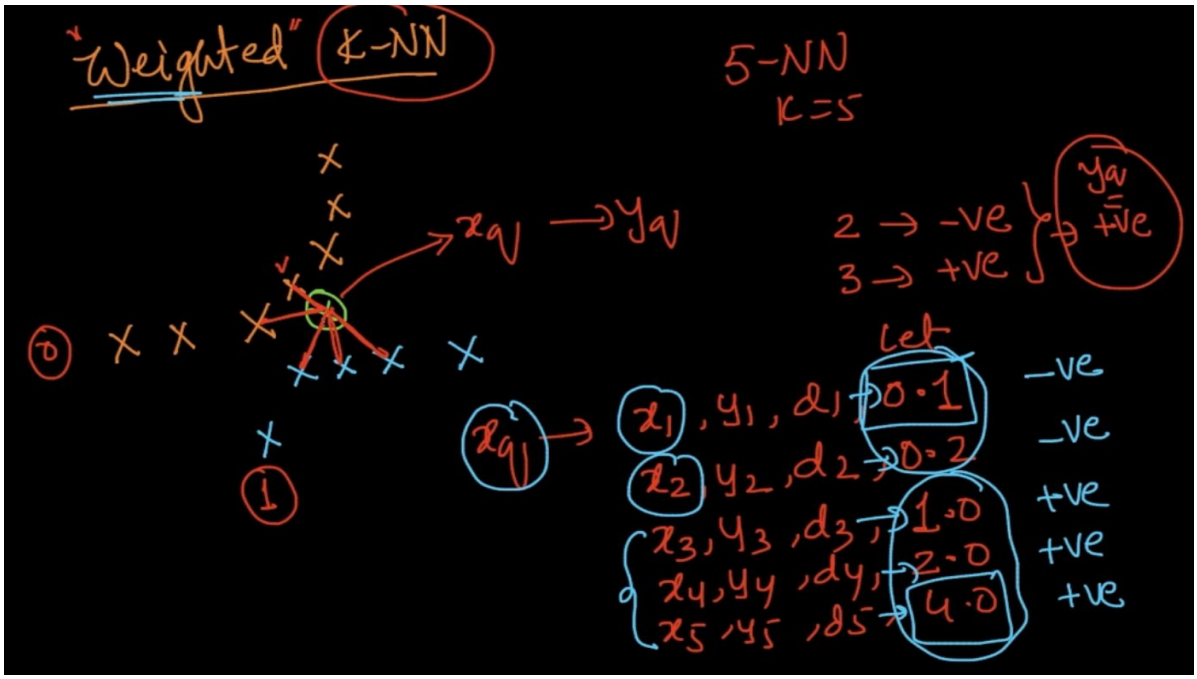
In Classification , y is class - label i.e {0,1} binary class or {0,1,2,3…..9,10} multi class

In Regression , y can be any number

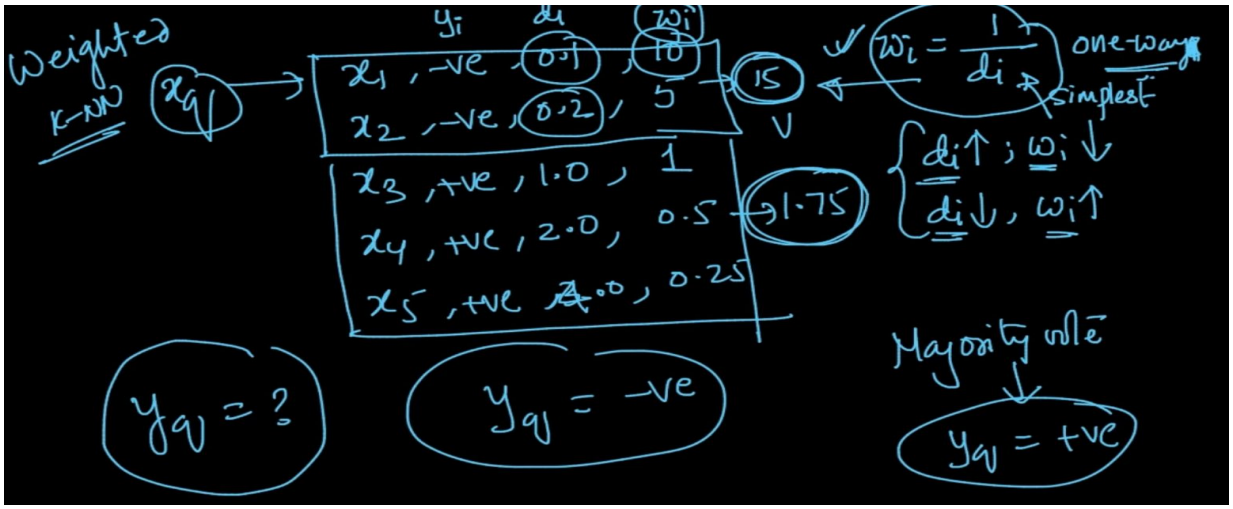


For prediction of y_q in Classification majority vote of all the y's is done but in Regression y_q is predicted by calculating mean or median of y's of all the k-nearest neighbors

WEIGHTED K-NN

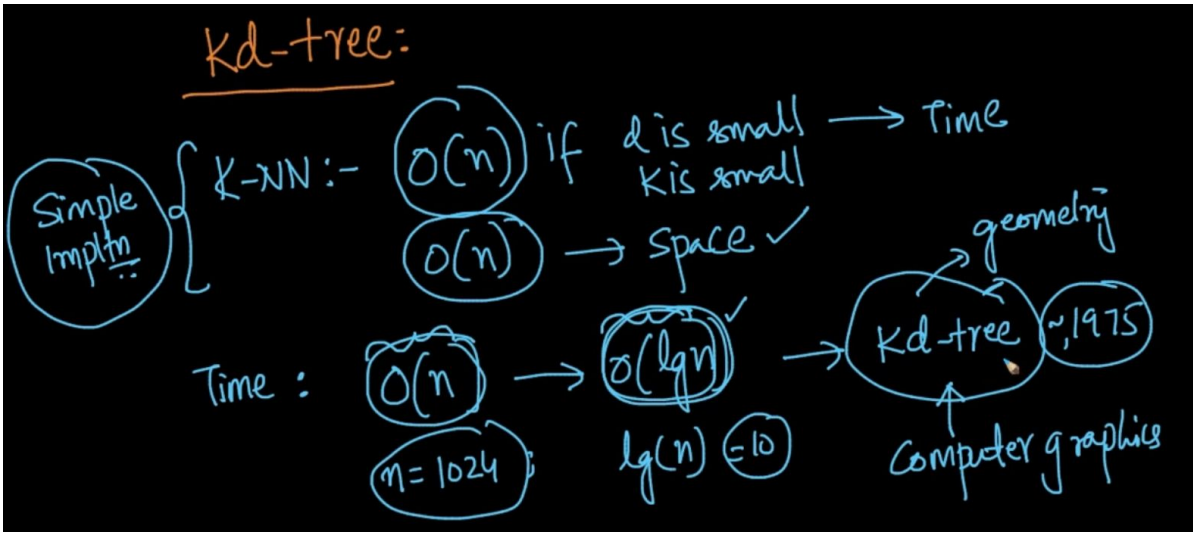


Here, we've a query x_q and it's Nearest Neighbor is calculated and through majority votes it is positive but distances of negative points is less . So shall we not predict Negative label since it's closer.



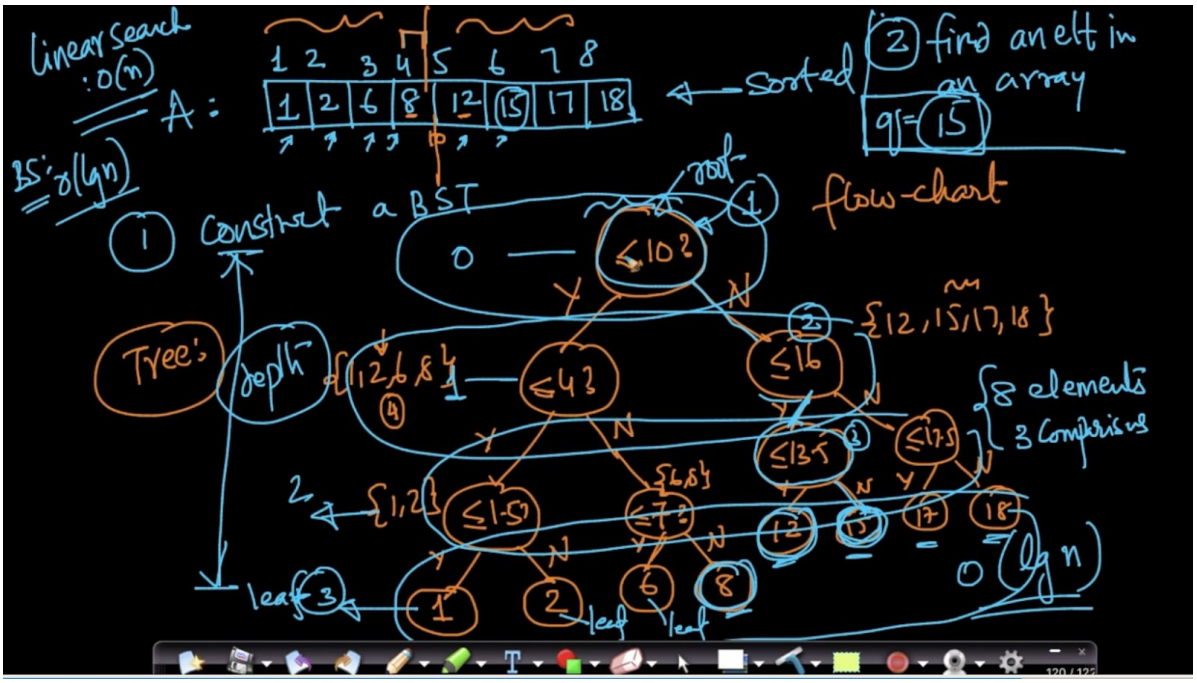
Weights are calculated through the above weight function (w = 1/d) and sum is calculated and wt of negative > wt of positive so y_q = negative

BINARY SEARCH TREE



We want to reduce the time complexity of K-nn from O(n) -> O(lg n) but first we'll learn a simple data structure called Binary Search Tree

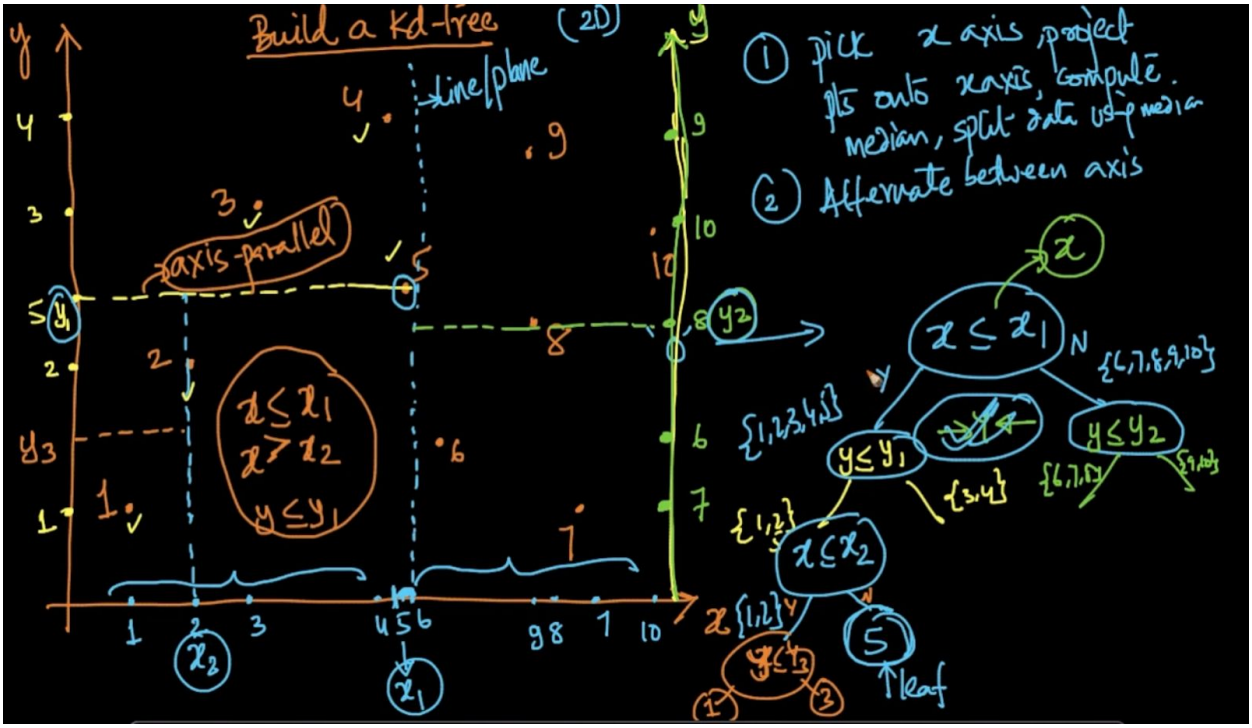
Problem - Given a sorted array find a number is present in array or not?



Just 3 comparisons needed compared to linear search in which Time complexity is O(n)

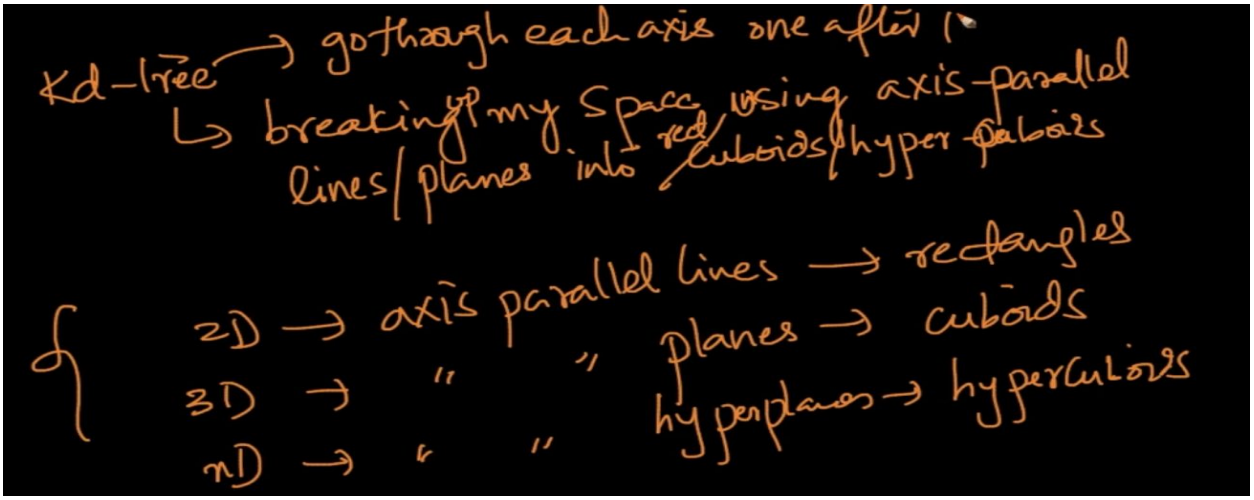
Depth = 3 which is equal to time complexity of BST O(log n)

HOW TO BUILD A KD-TREE

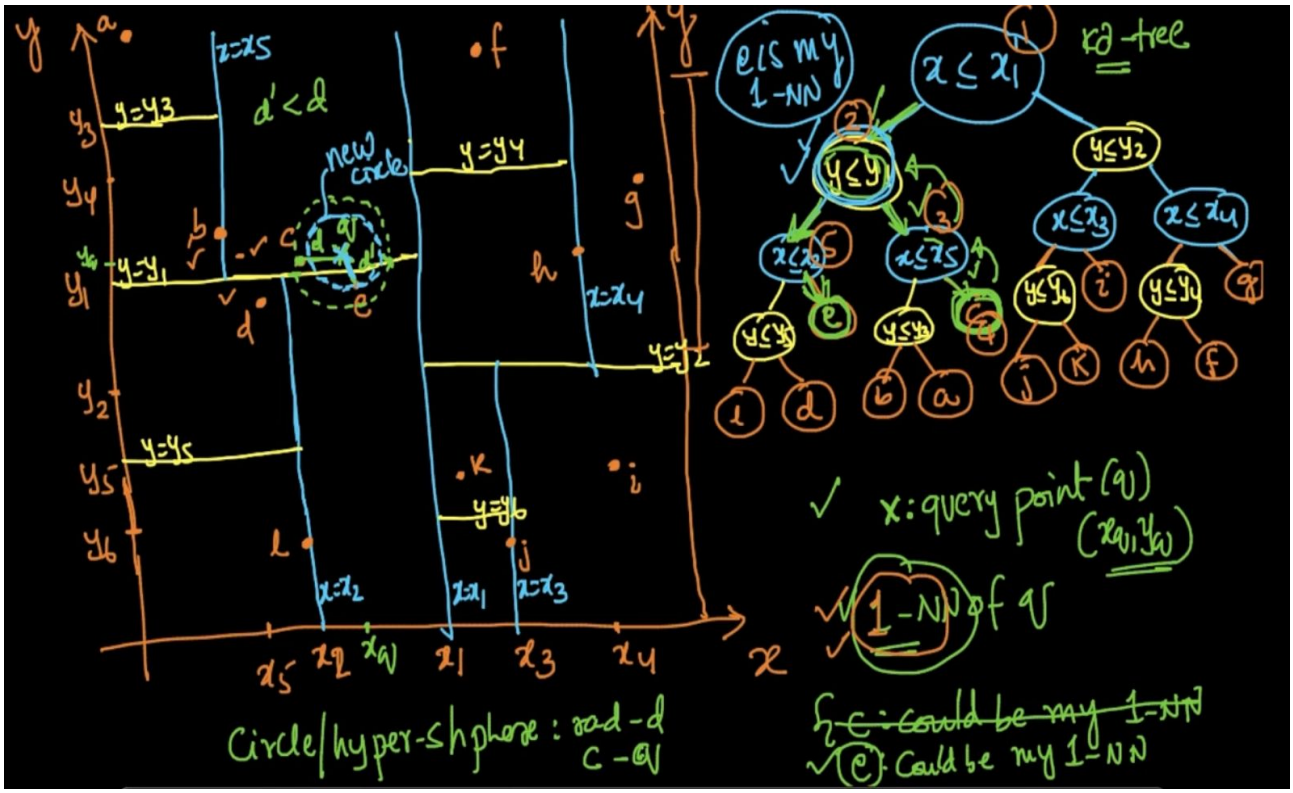


Pick X-axis , project all points into that, get median ,split data into median . Now do the same things with y axis with the split data and now x- axis and repeat that alternatively till you get all the leaves.

What we are doing here us we are breaking our vector spaces using lines called axis-parallel in 2D . Ex : For 5 the conditions are x $\le x_1$, x > x_2 , y $\le y_2$ and x_1, y_2, x_2 are medians of data



FIND NEAREST NEIGHBORS USING KD-TREE



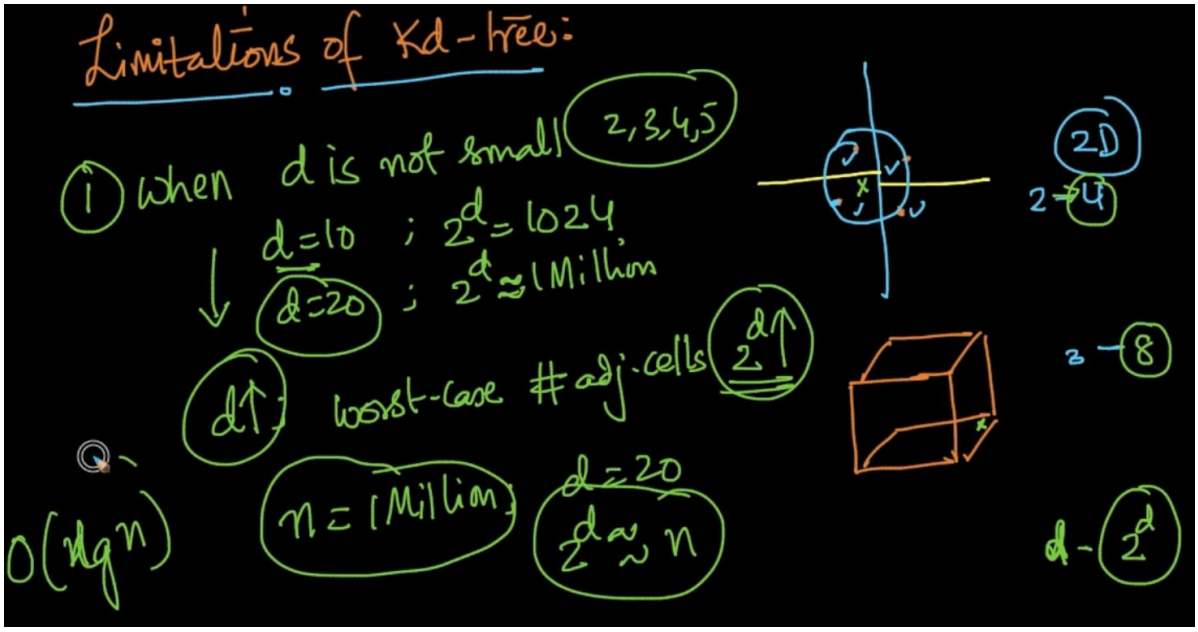
We've x : x_q -> proj (x axis), y_q -> proj (y axis) . For determining it's nearest neighbor we'll use the kd-tree and check the conditions. Here , $x_q \leq$x1 (Yes) -> $y_q \leq y1$ (No) -> $x_q \leq x5$ (No) -> c but it can be seen that **e** is more close to **x** than **c** . So we draw a circle with radius = len(xc) and see if there is any line intercepting . Here it is y1 so we'll backtrack to y1 and see the other condition (x < x2) which is No therefore e is the 1-NN

Time Complexity: No. of comparisons to find NN

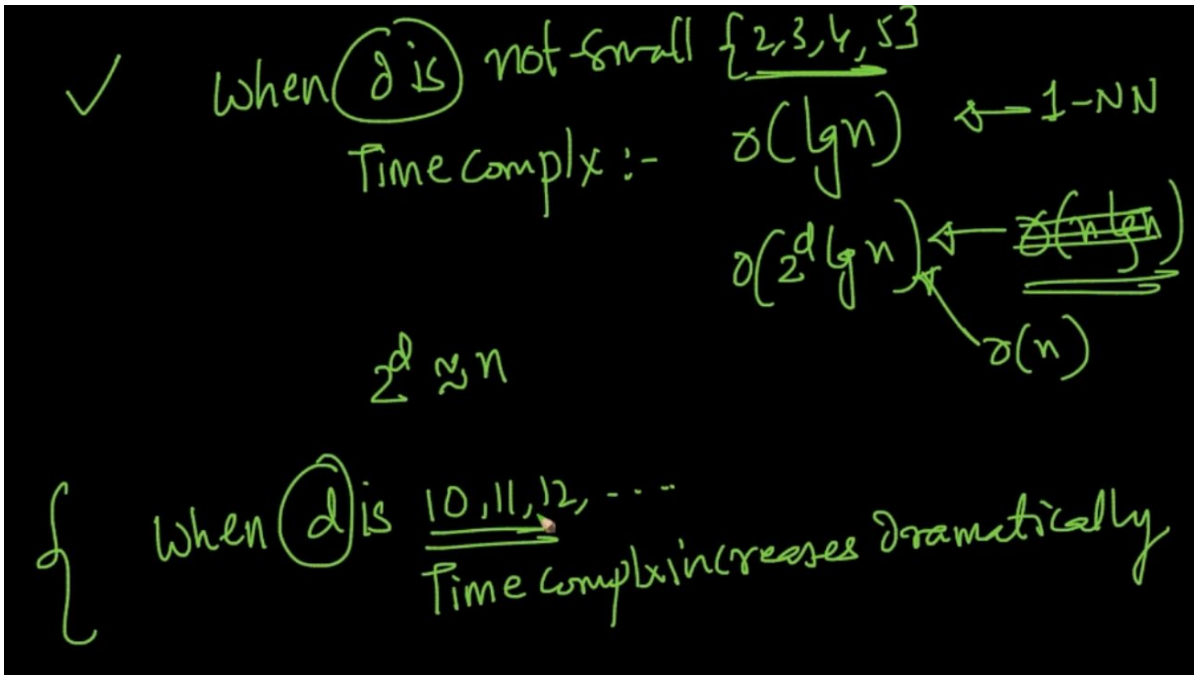
Best case : O(log n)

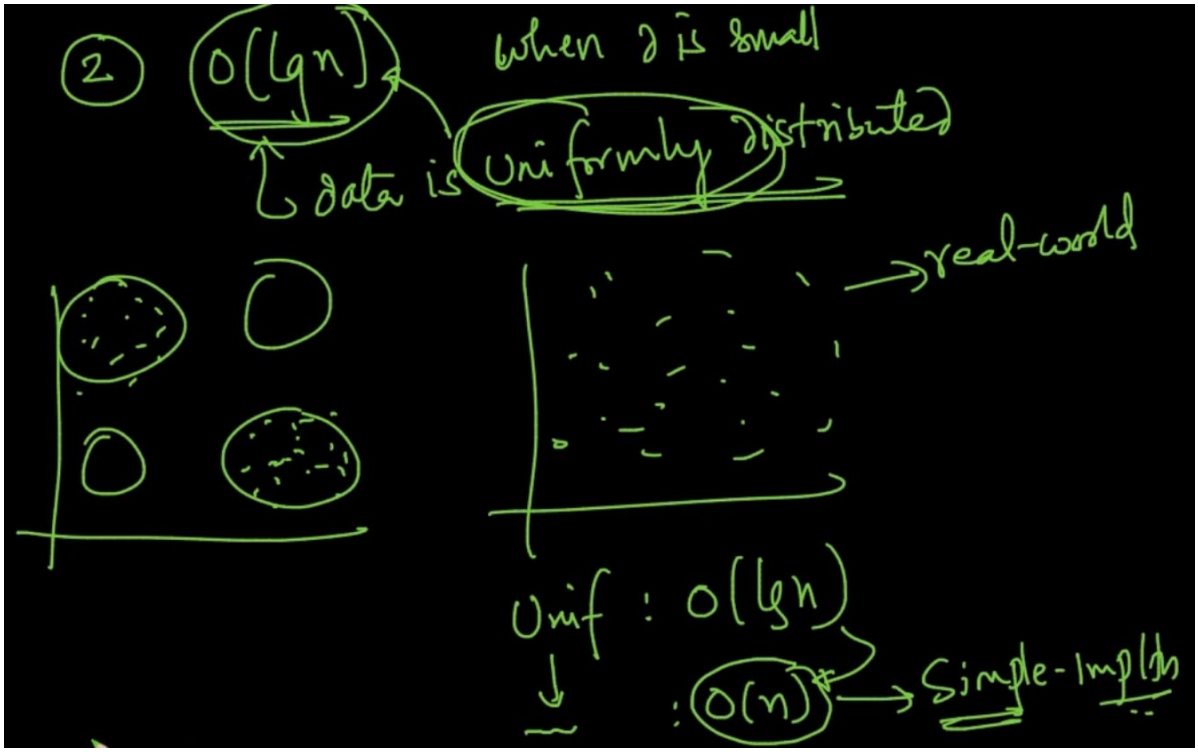
Worst case : O (n)

LIMITATIONS OF KD-TREE



If the data is 2D we've to look at 2^D = 4 adjacent cells as seen above. So as d ↑ in worst case adjacent cells 2^D

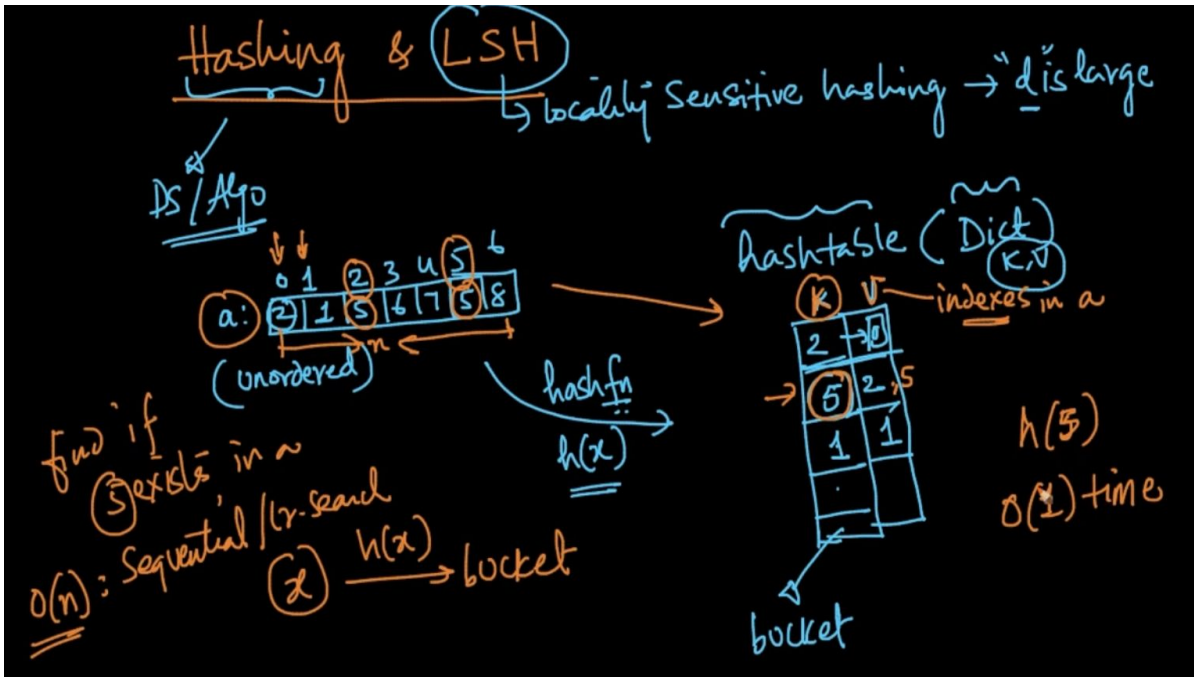




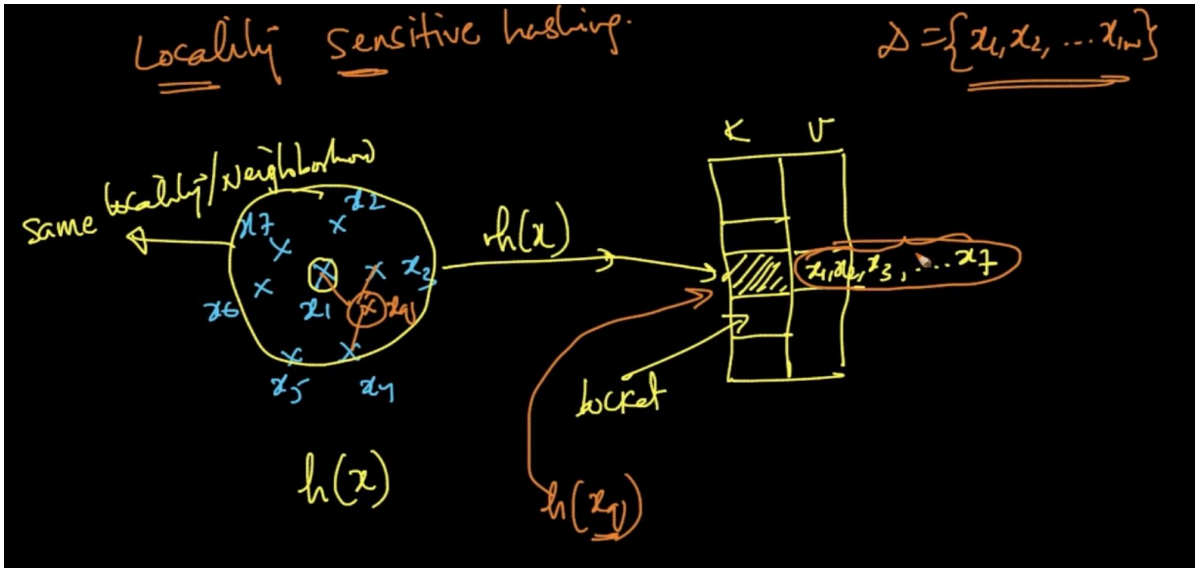
When data is not uniformly distribution our KD -Tree merges towards O(n)

[k-d tree - Wikipedia]

HASHING & LSH (LOCALLY SENSITIVE HASHING)

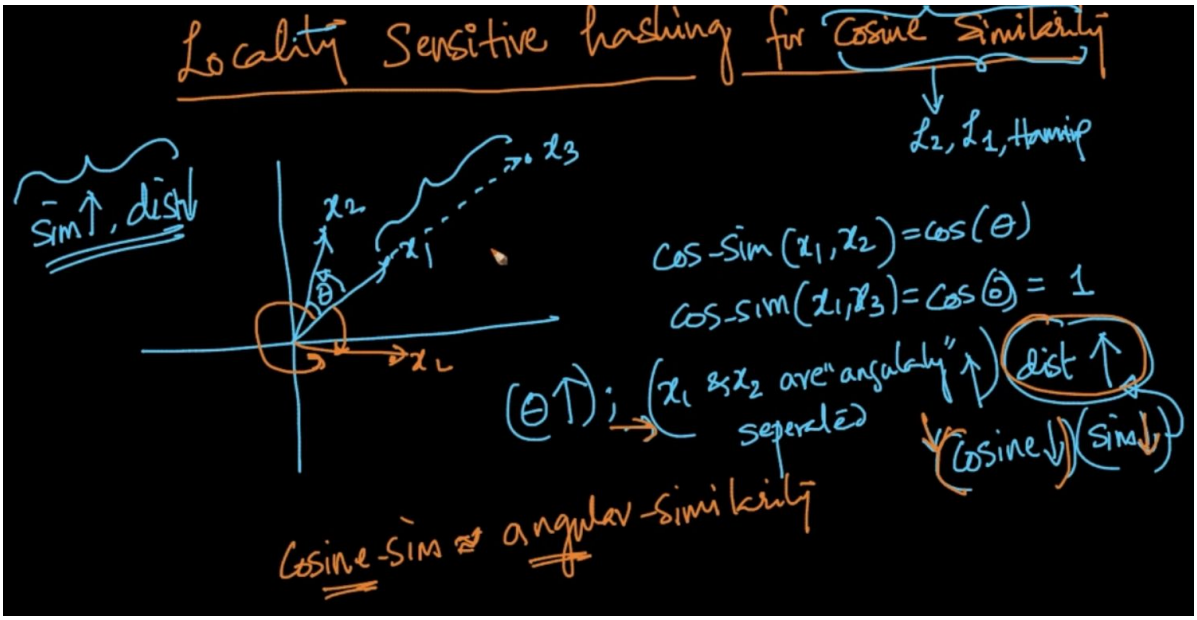


Values in array are stored as a key and the indexes are values . If we need to find one value let's say 5 then we'll apply hash func'n h(5) and we'll get it on O(1) time

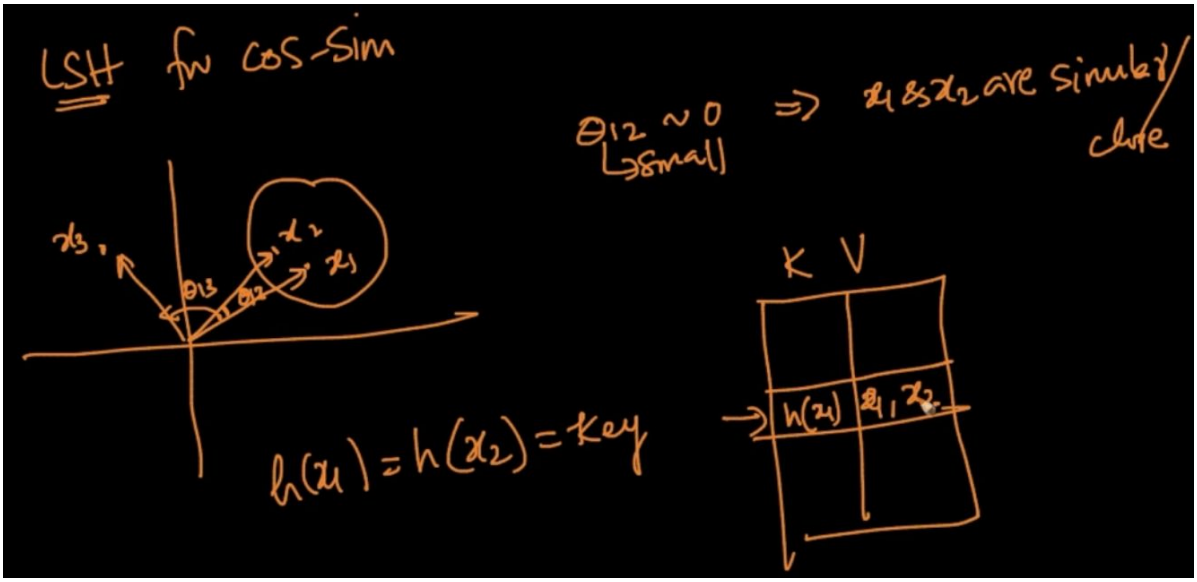


If we've data close as seen we'll apply hash function h(x) such that it goes in the same bucket and if a new point x_q is near them then it'll go to the same bucket

LSH FOR COSINE SIMILARITY

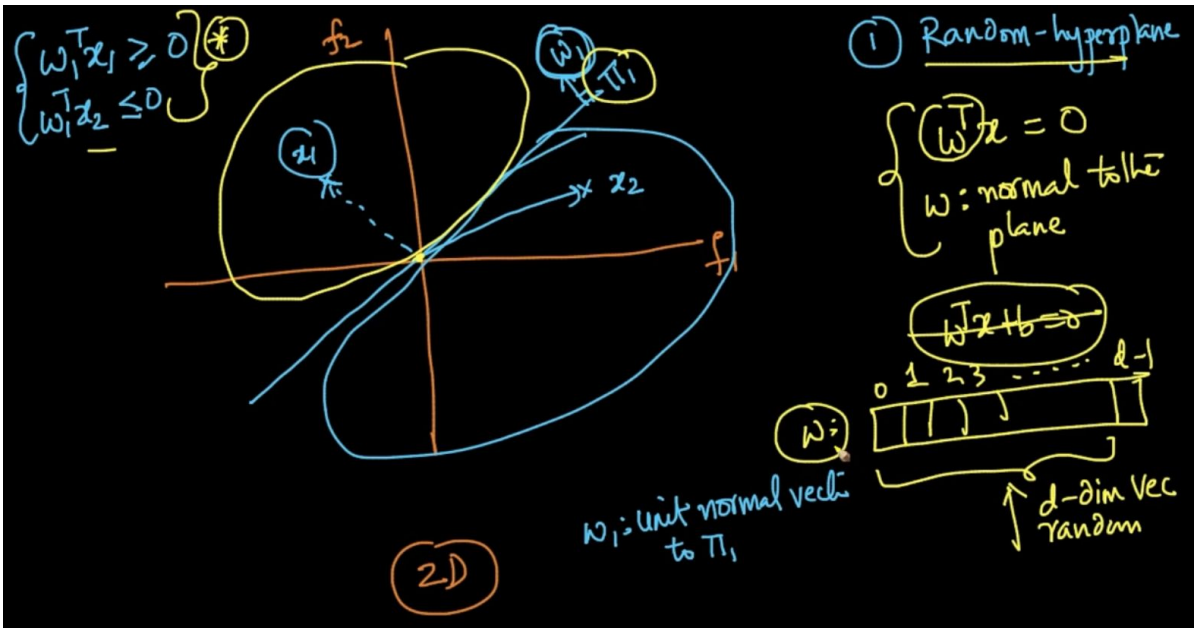


As θ increases cosine similarity decreases and therefore similarity also decreases



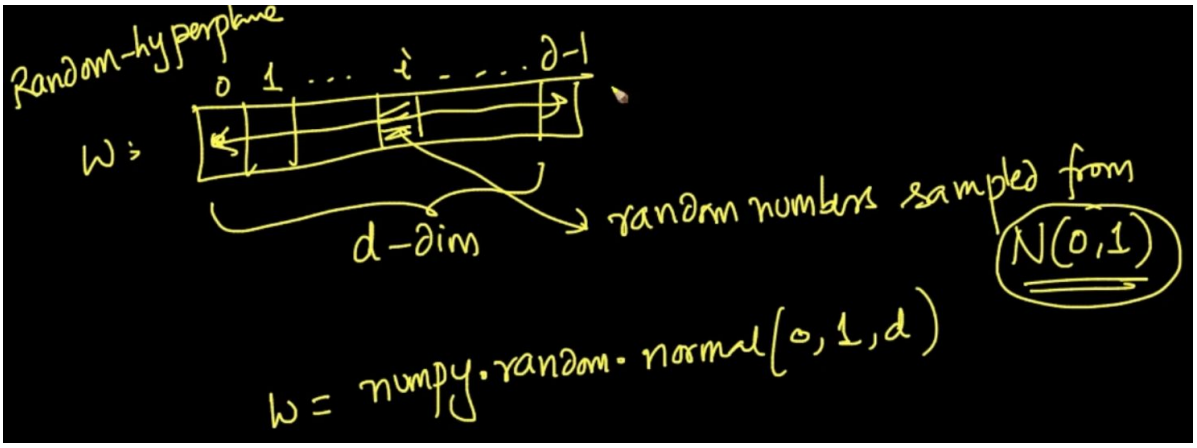
As θ between two data points x1 and x2 is small they are similar therefore in the hash table

These points should go together and the key is h(x1) = h(x2) . So our objective is finding that hash function h(x)



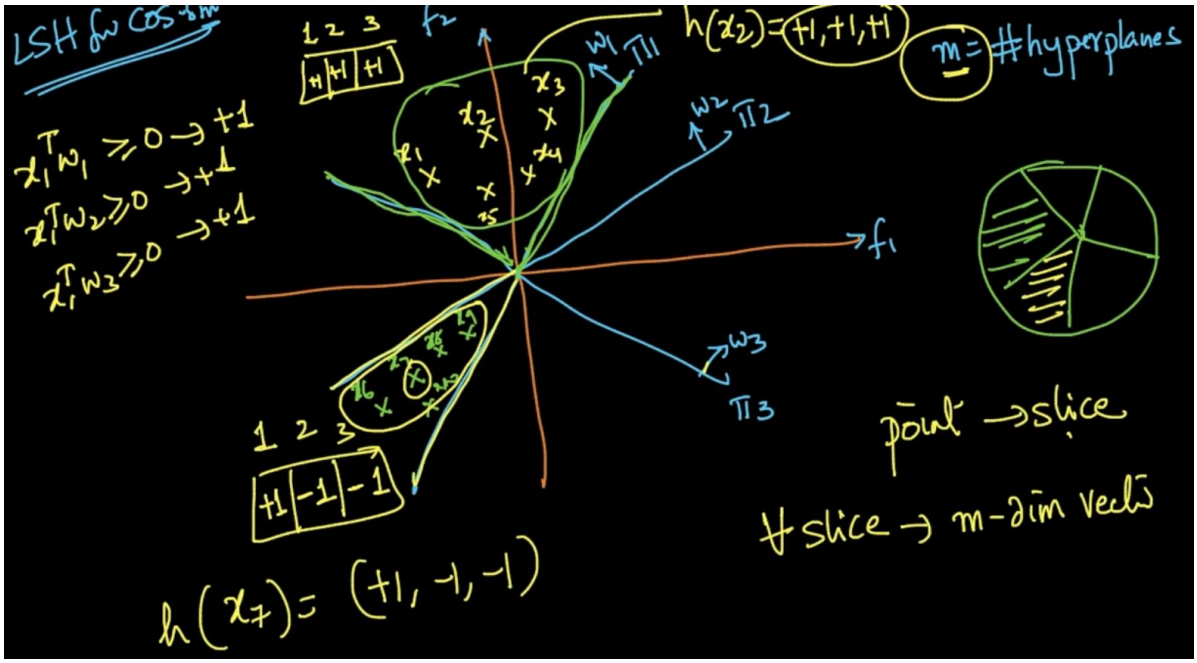
Suppose there's a hyperplane $\pi 1$ and it's normal w_1, We know $w_1^T x_1 \geq 0$ since w_1 is pointing towards x_1 and opposite for x_2.This idea is our crux here.

w is a d-dimensional vector and if we can generate w we can generate it's hyper-plane

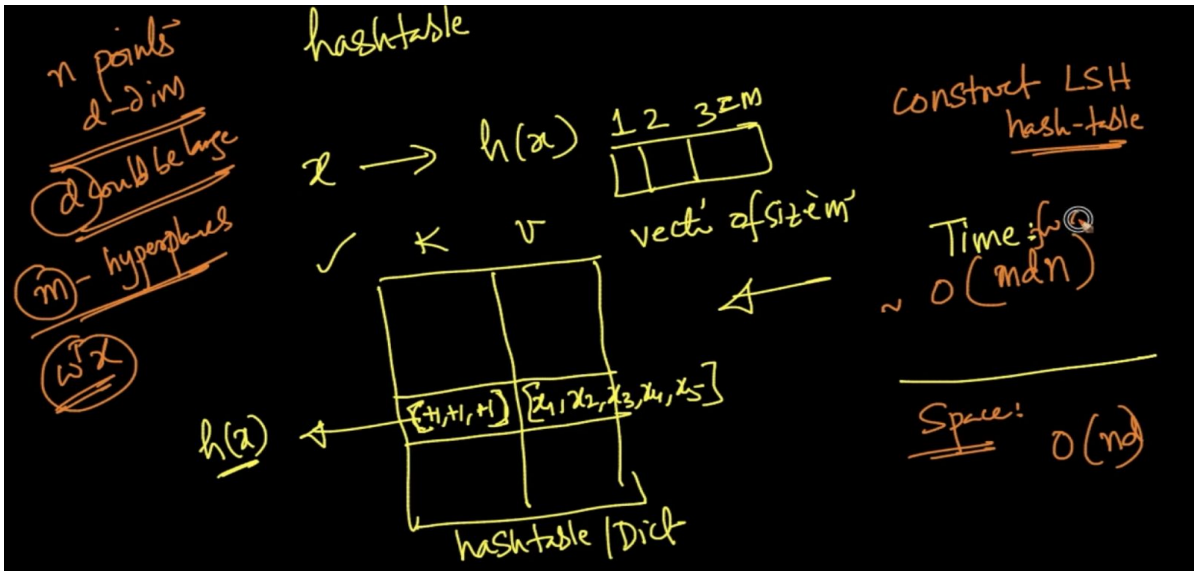


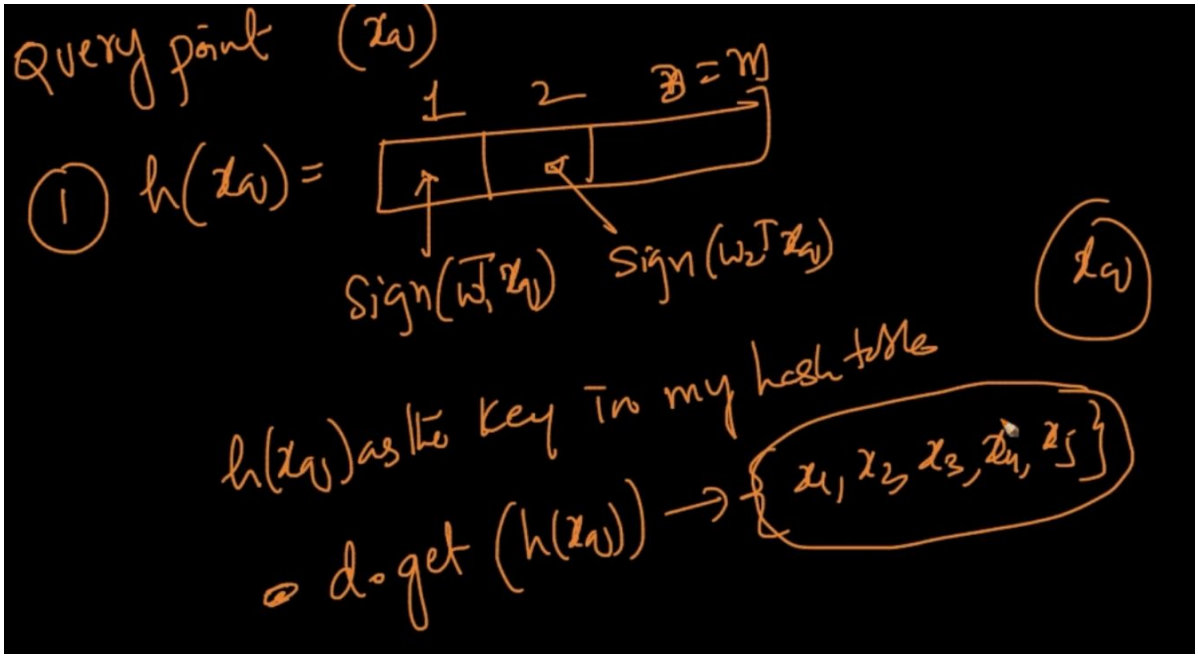
We want to generate a random hyperplane with it's normal w such that the values in w are random numbers with mean 0 and variance 1 and normallly distributed N(0,1)

So, when we are generating random hyperplanes we are uniformly splitting our vector space into different regions

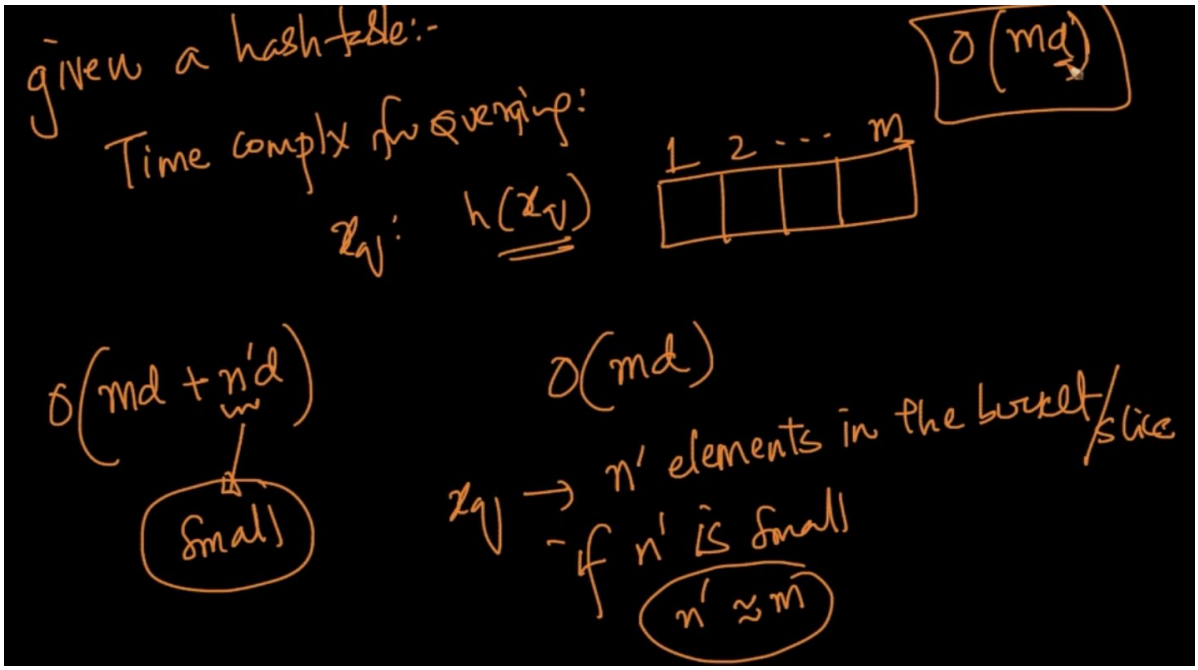


So we are generating m-hyperplanes (m = 3 here) and for all slices m-dimensional is created (3-D vector here since m = 3) . As seen x_1 is on the +ve side of all the planes therefore $x_1^T w \geq 0$ -> +1 for all w1,w2,w3. Therefore, a vector of 3-dimension is [+1,+1,+1] for x_1 and that vector is our h(x). Same, for all the other points .If $x^T w < 0$ then -1 goes to vector. Therefore all the data points with the same vector/h(x) goes into same hashtable



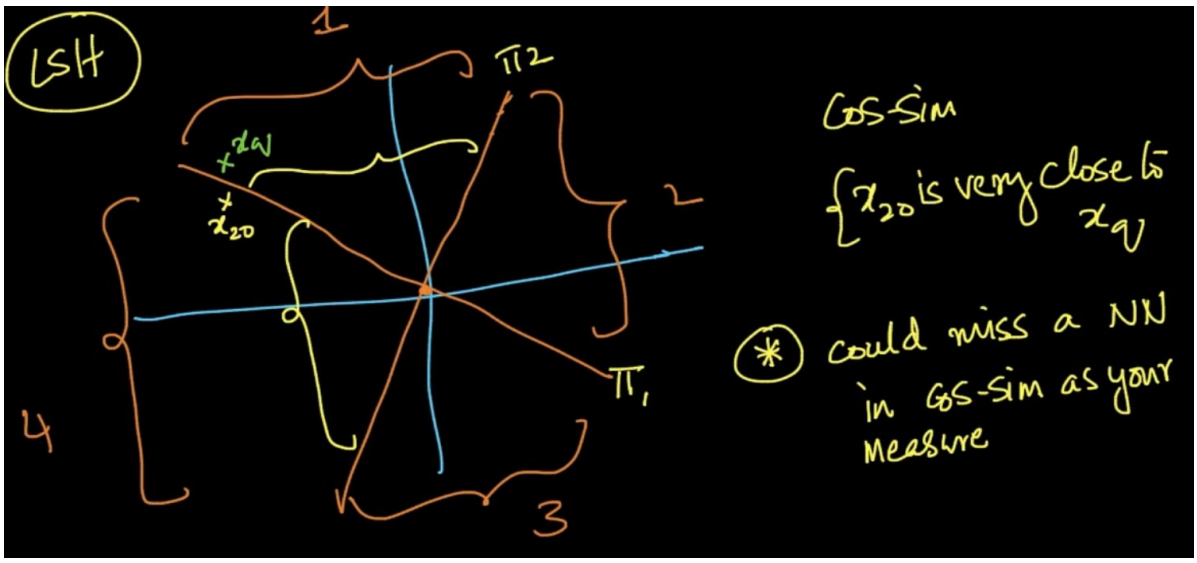


We've a new query x_q. We calculate h(x_q) and get the vector. We get the probabilitistic nearest Neighbors from that here h(x_q) = h(x_1) and so on for x2,x3. Therefore, {x1,x2..x5} Will be it's NN

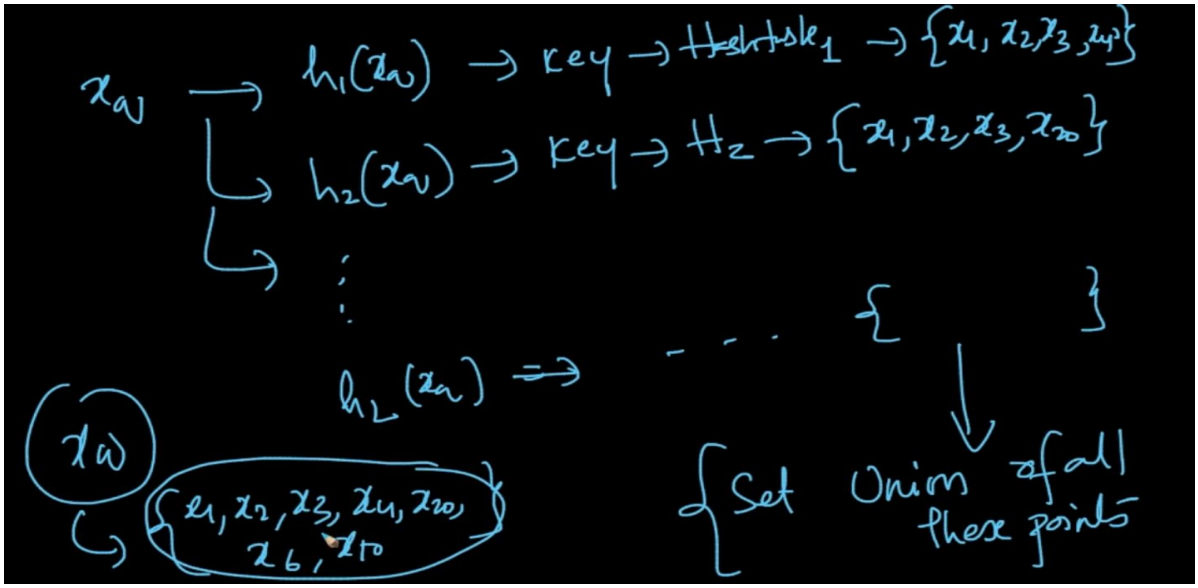


Time complexity for querying is O(md) as m-hyper planes and x_q is d-dimensional.

Typically, m = log(n) therefore time complexity : O(d*log(n))



x_{20} & x_q are nearest Neighbors but since they are in different planes it could be missed



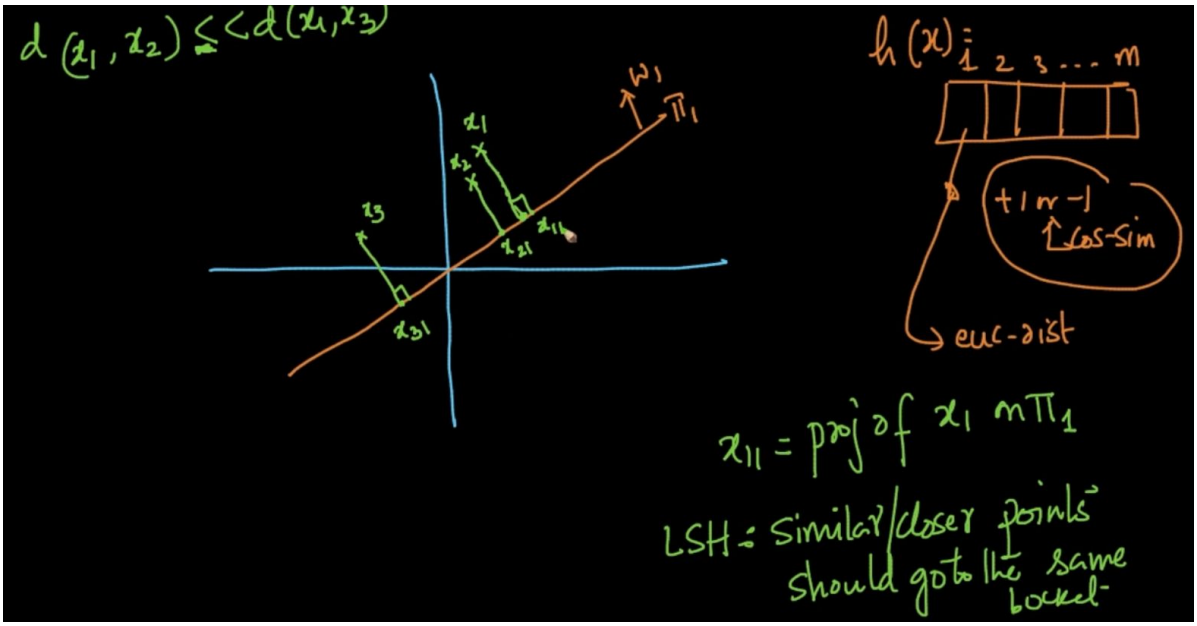
We are generating m - hyperplanes and there will be one hyperplane such that x_q & x_{20} are in same hyperplane. So , for x_q we'll generate different hash tables and some different NN can be given for x_q . We'll take a Union of all these points and from that set NN will be calculated. Total, L- hash tables are created where L is a number.

Time complexity to a query given L hash tables: O(m*d*L) L is small here

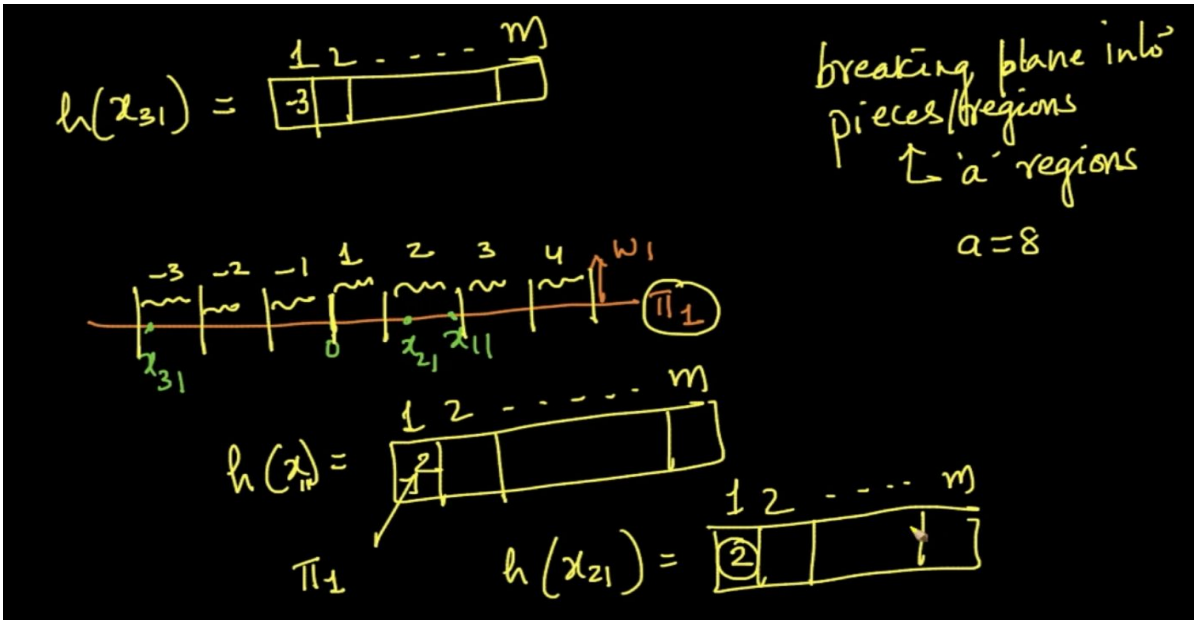
Note : As number of hyperplanes ↑ , no. of slices will also ↑ => no. of pts per slice ↓

Therefore, m should be chosen appropriately, Typically, m = O(log(n))

LSH FOR EUCLIDEAN DISTANCES

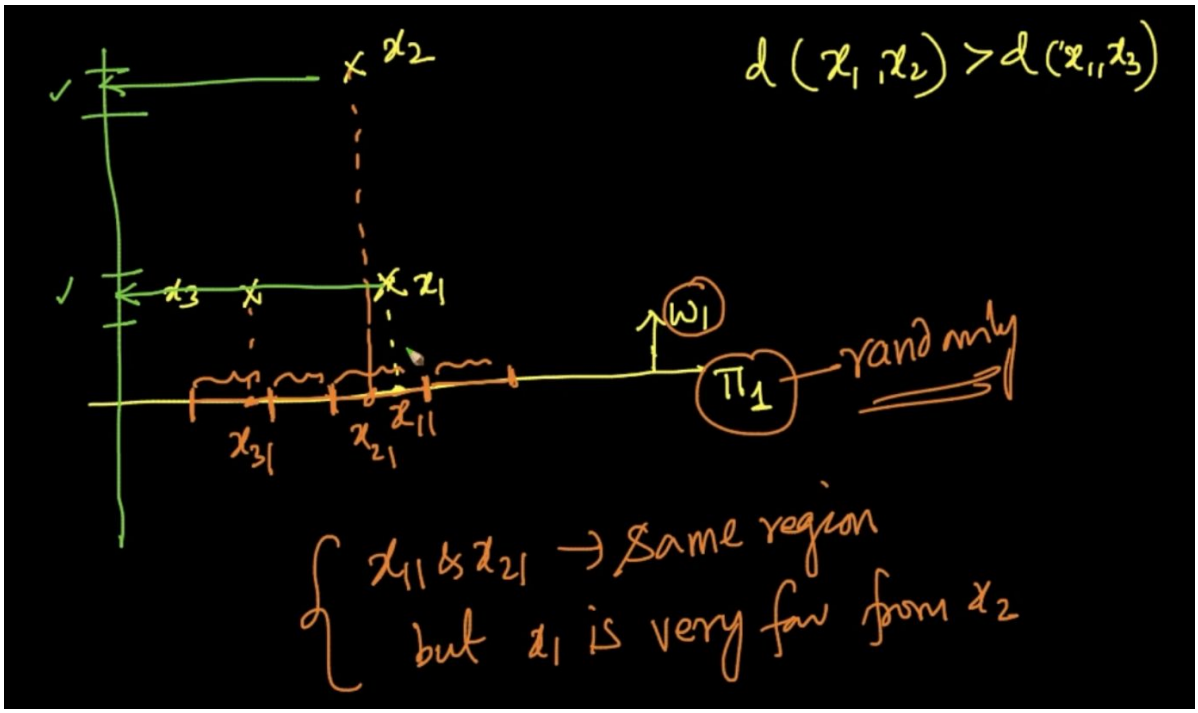


We project data points on hyperplane.



Then after projecting we break plane into regions and give the regions values (here (-3,-2.. 3,4)) . When we calculate h(x) the region value of the plane goes into vector instead of -1 or 1 like LSH for cosine similarity. That's the difference between LSH of cos-sim and euc-dist

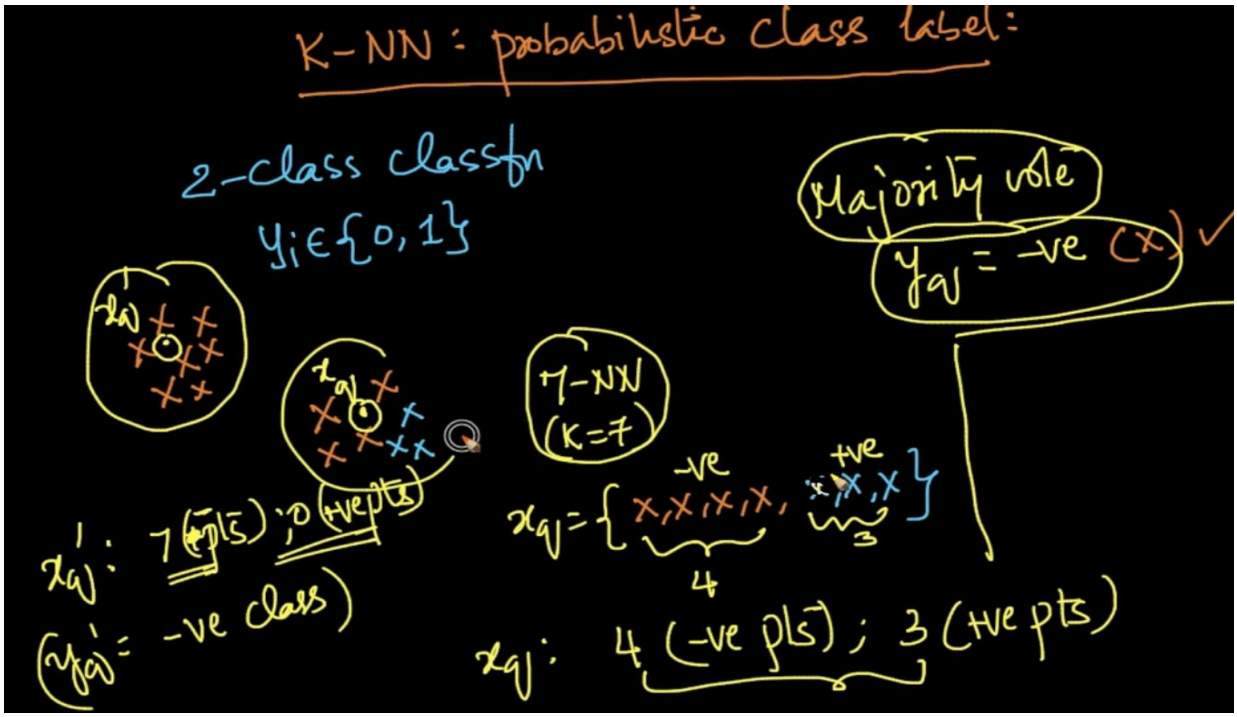
As seen above x_{11} & x_{21} are very near therefore they are in the same region '2' on plane π_1



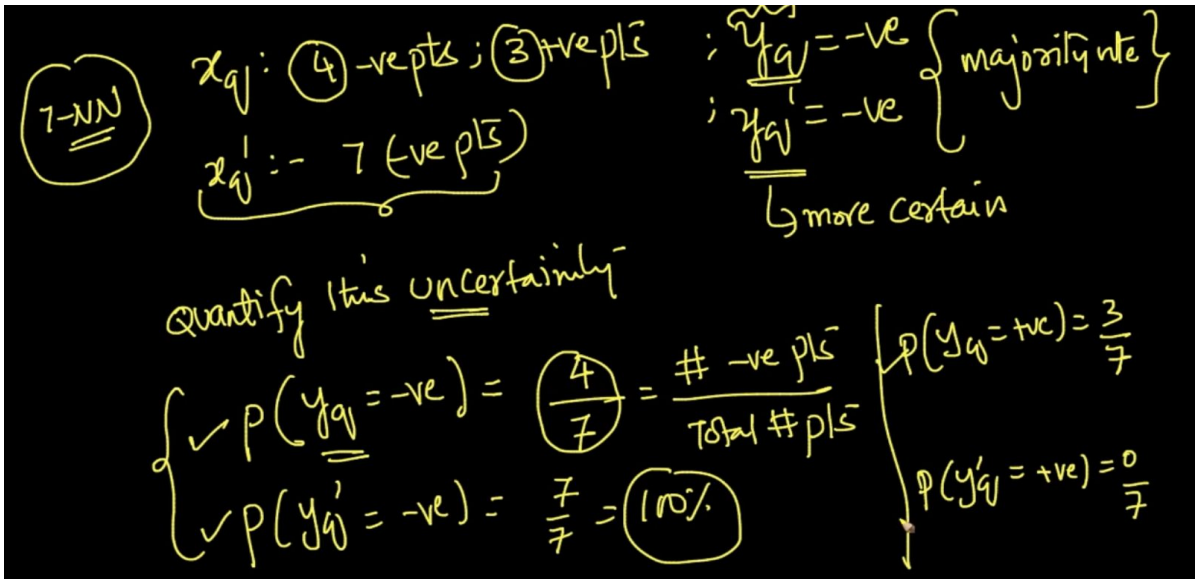
Boundary case : x_1 & x_2 are very distant but when projected on plane π_1 they fall into the same region but in green plane they are into different regions appropriately as it should have been

So, we should be knowing that LSH is randomized and it's not perfect

PROBABILISTIC CLASS LABEL



For k =7 (7-NN) x_q = -ve by majority vote and $x_{q'}$ = -ve too from majority vote but we want to be certain /quantify about how -ve/+ve our query point can be



We are getting the Probability by using the above formula. By probability we can be certain about which class our x_q might belong to.