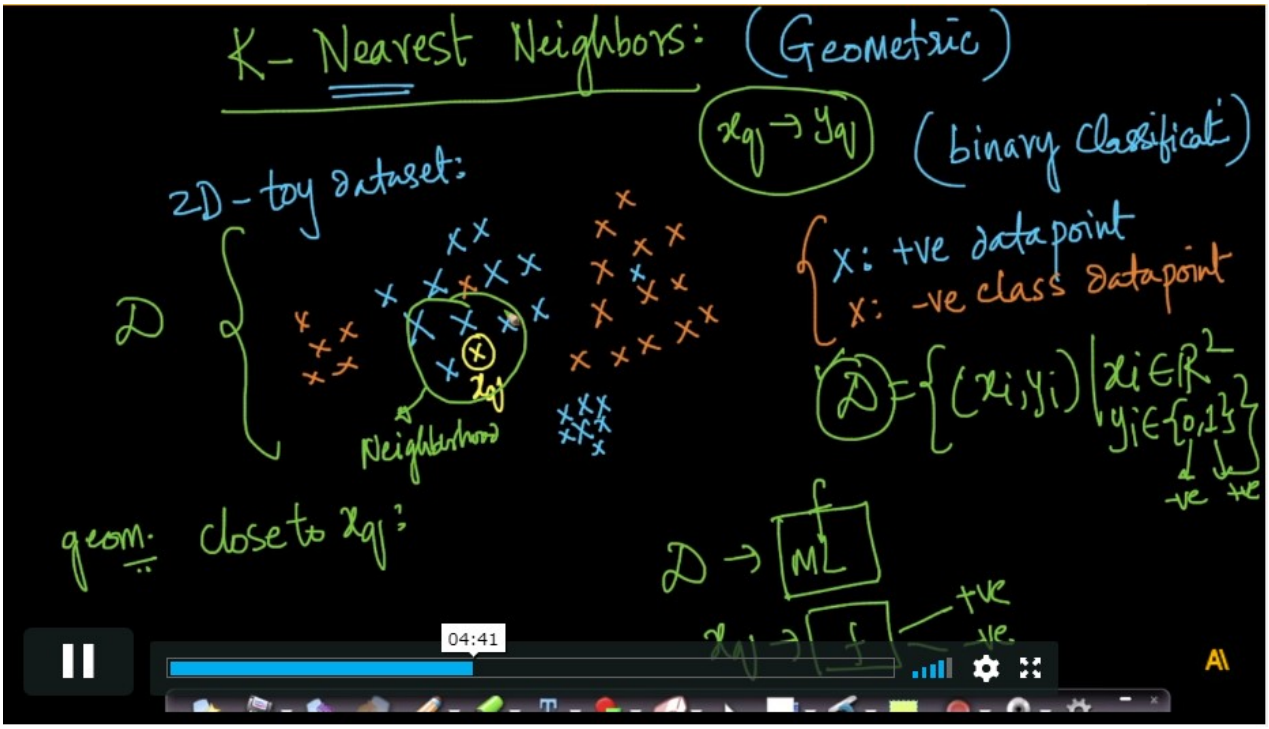
Notes

K – Nearest Neighbors

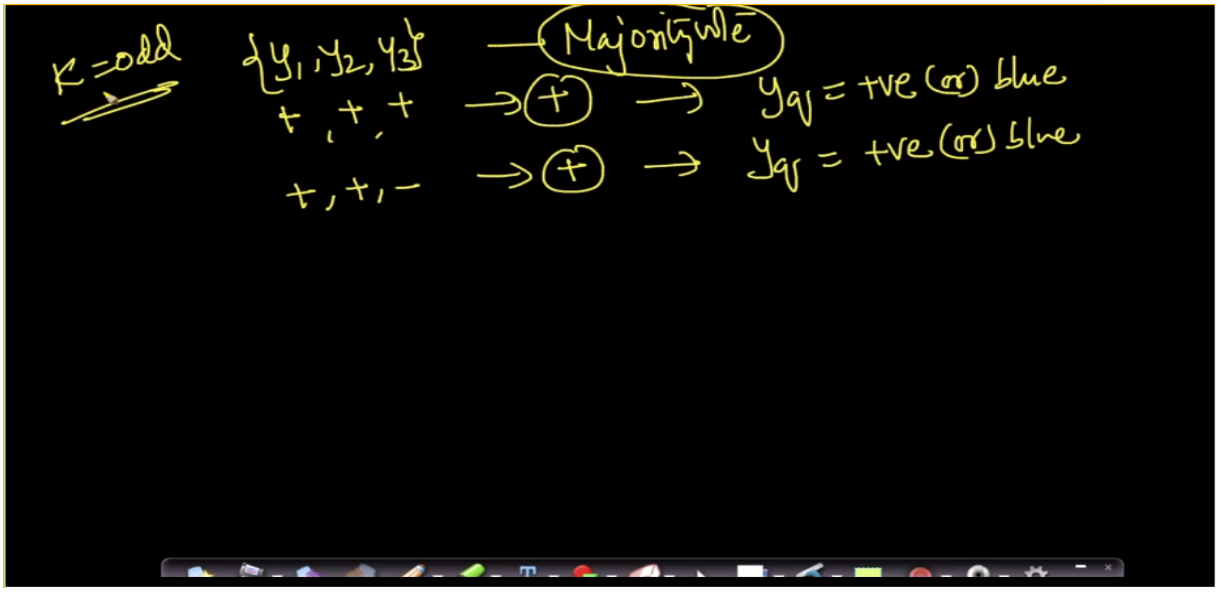
Given the data set.

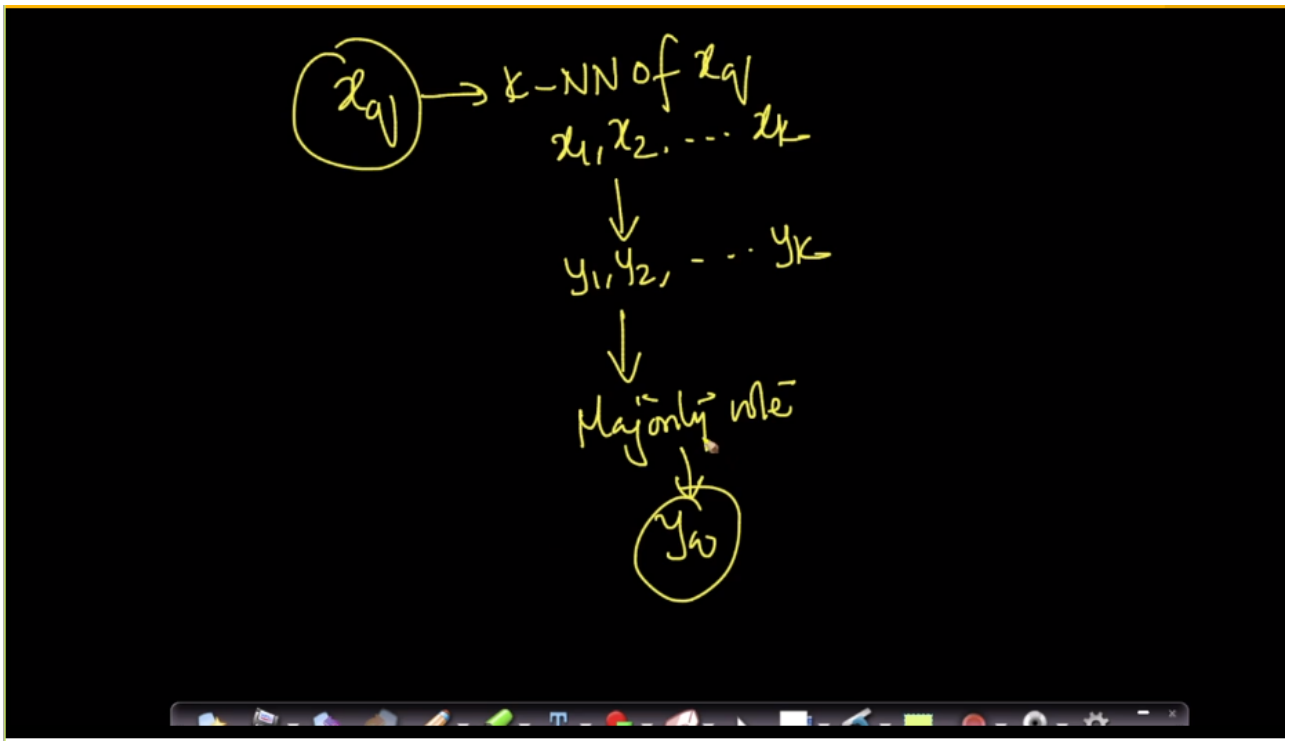


Because the query point lies near to the blue point then the point considered to be the blue class, KNN at a high level.

Working:

- Find the K nearest points of the query point.
- It does majority vote of the K nearest points.

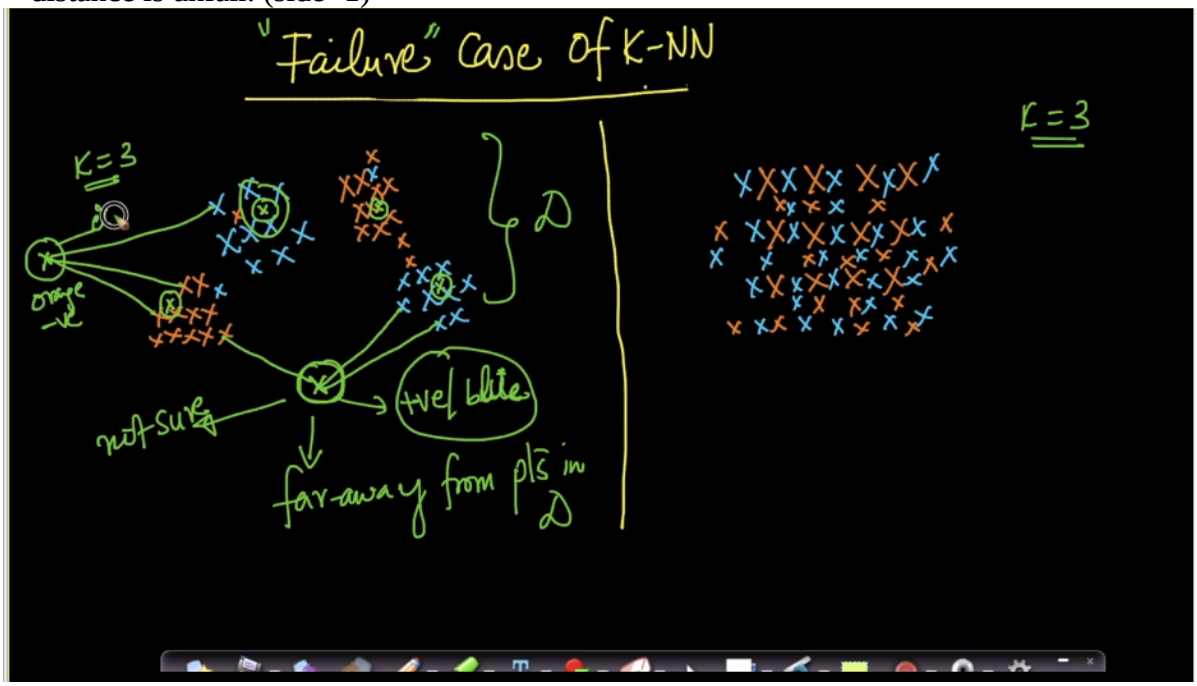




Failure cases of KNN:

Case - 1(clustered)

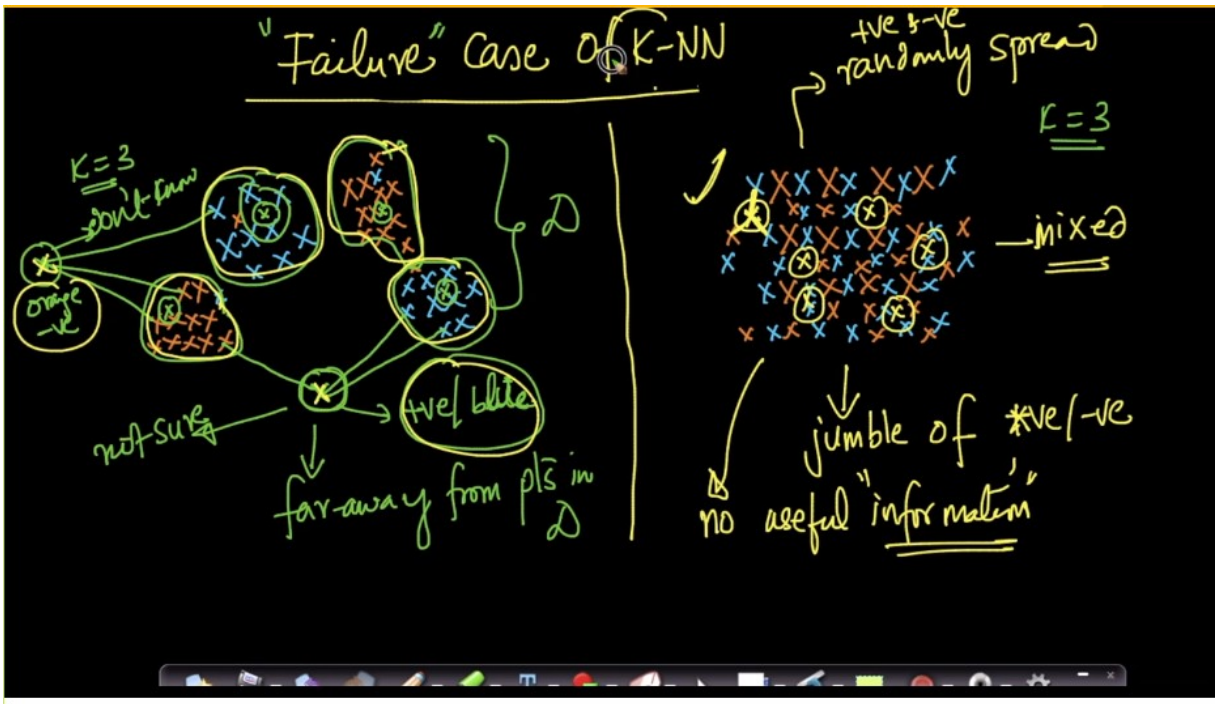
- For the very far away query points deciding the class of the query point by computing the distance is unfair. (side -1)



It is better to say not sure of the point.

Case – 2(randomly spread)

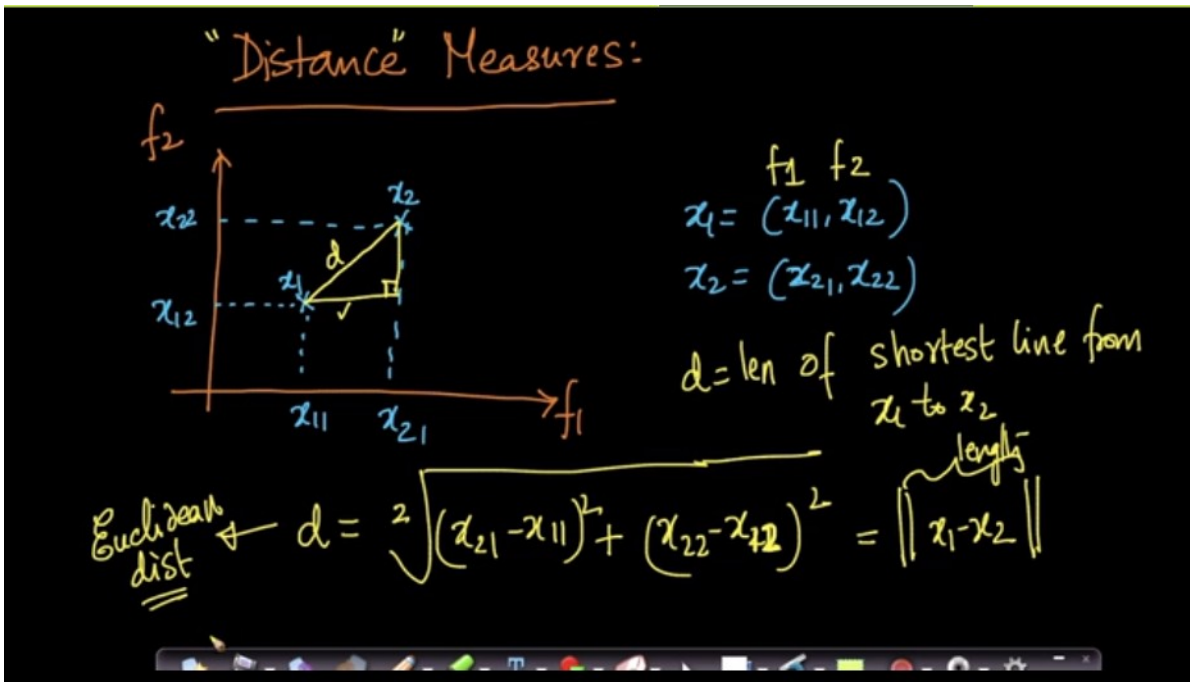
- No useful information in the Jumble data.



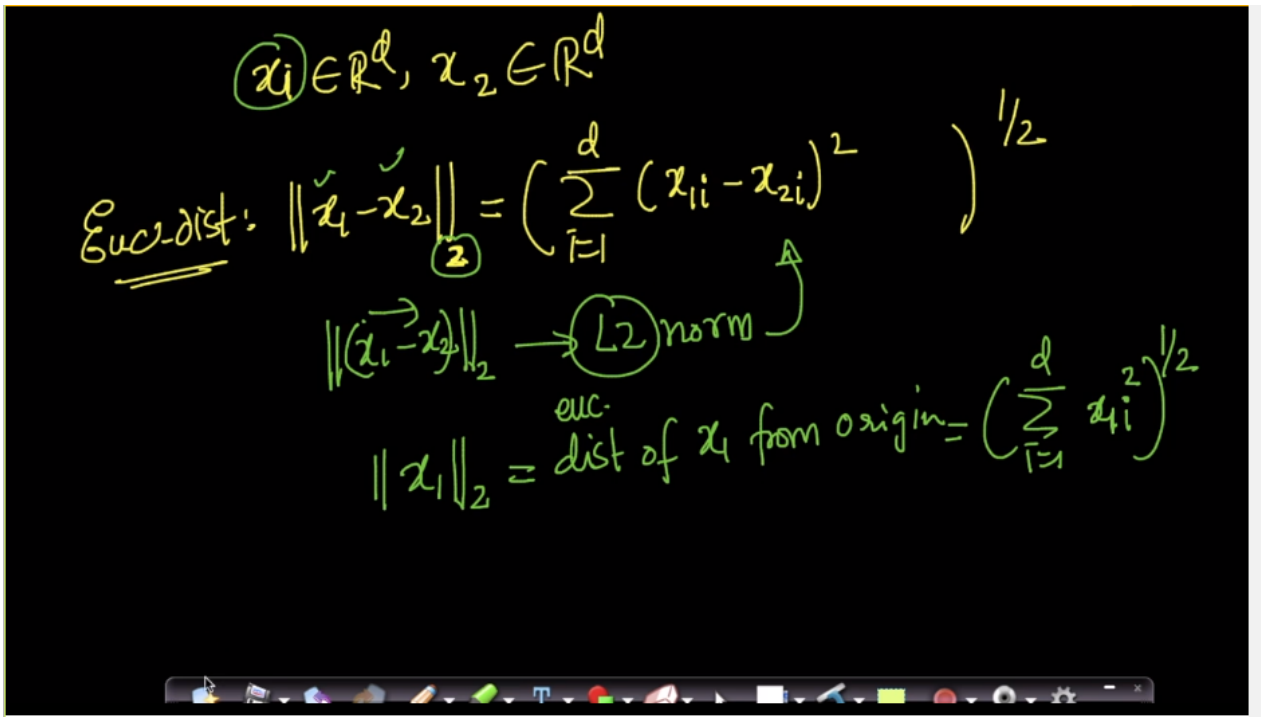
Distance Metrics:

Euclidean(L2) distance:

- d is the length of the shortest line from x1 to x2.



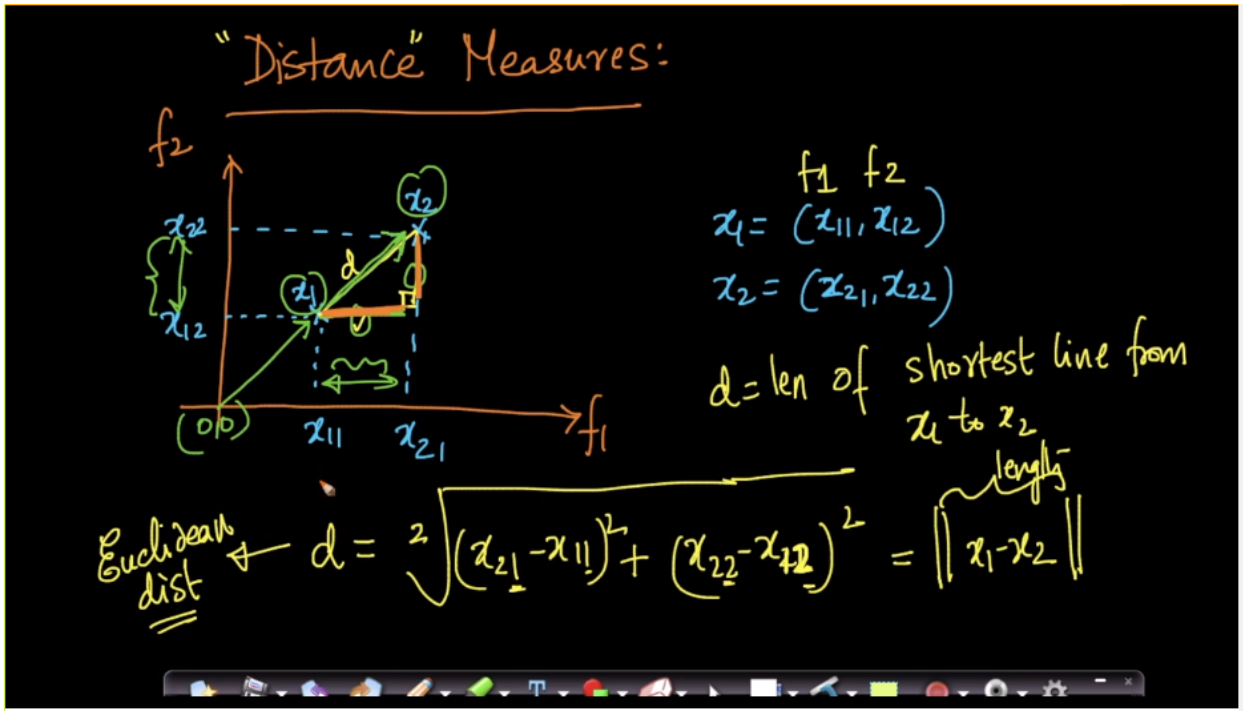
For d dimensional:



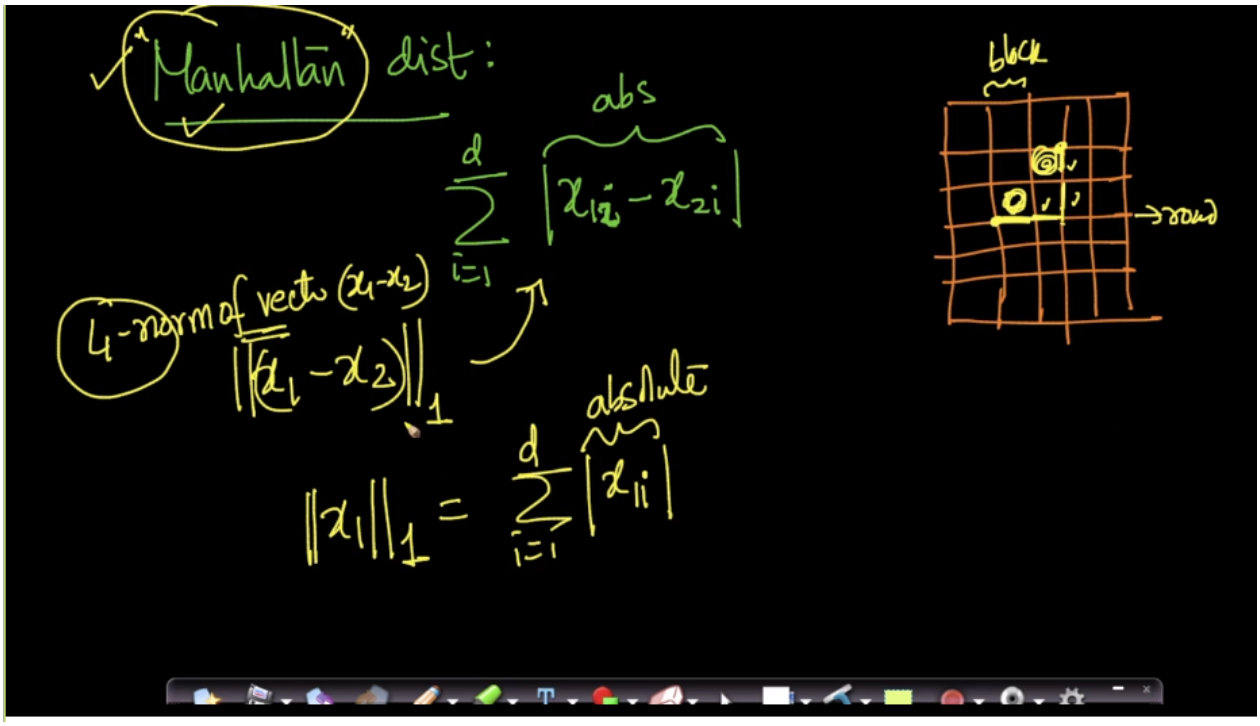
Euclidean distance is also called as the L2 norm in machine learning.

Manhattan distance:

- The sum of the distance along the perpendicular is called the manhattan distance.
- The sum orange coloured distance bars in the figure is called Manhattan distance.

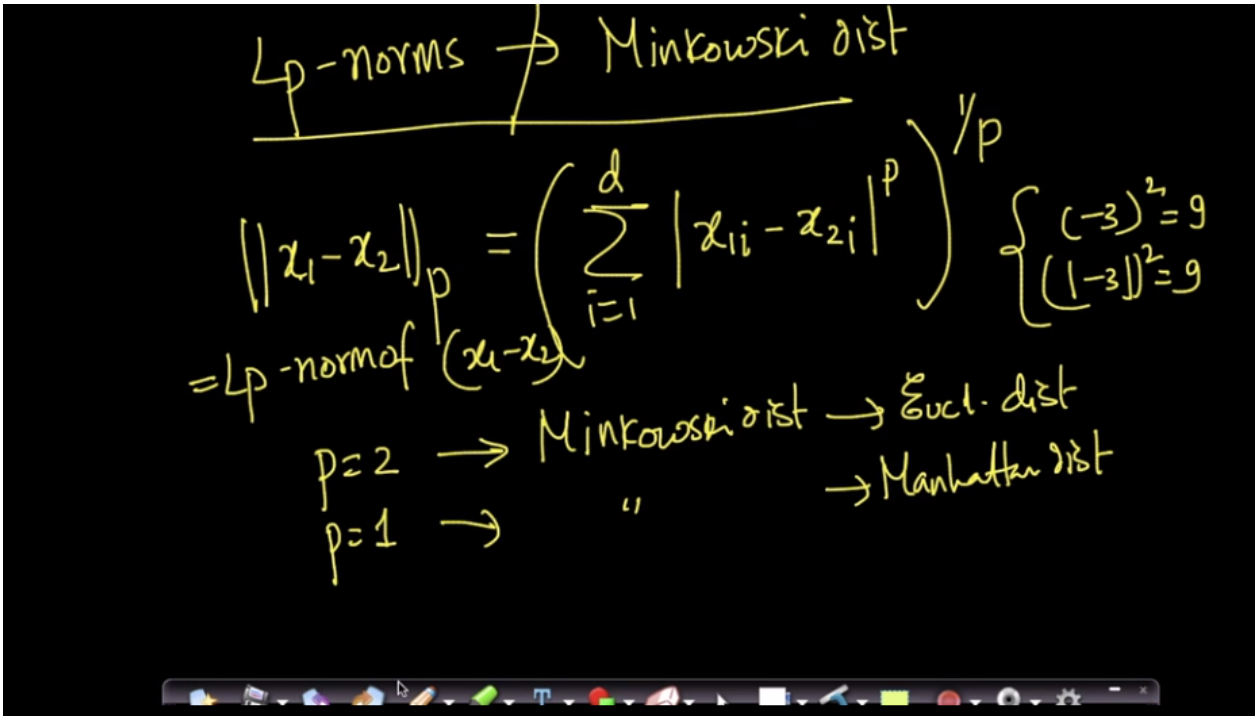


- Absolute value of the distance between the points is called Manhattan distance.
- This is also called as the L1 norm.

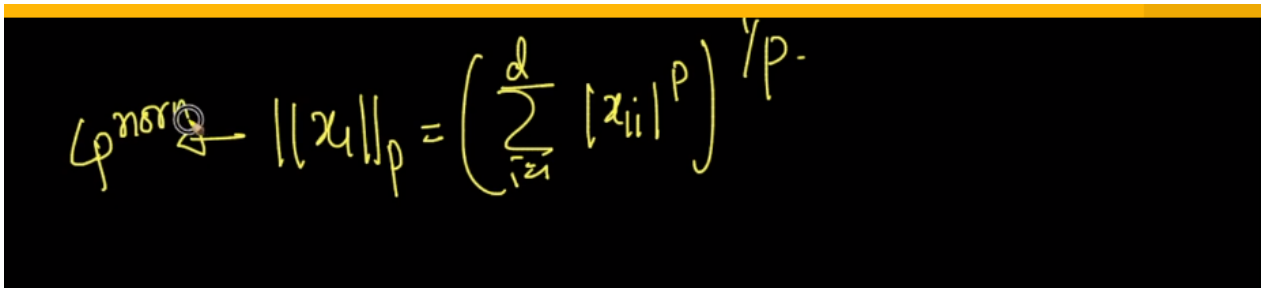


Lp norm or Malinowski distance:

- It is the power raised to “p”. when p = 1 distance is called the manhattan distance.
- For p=2 eucledian distance.

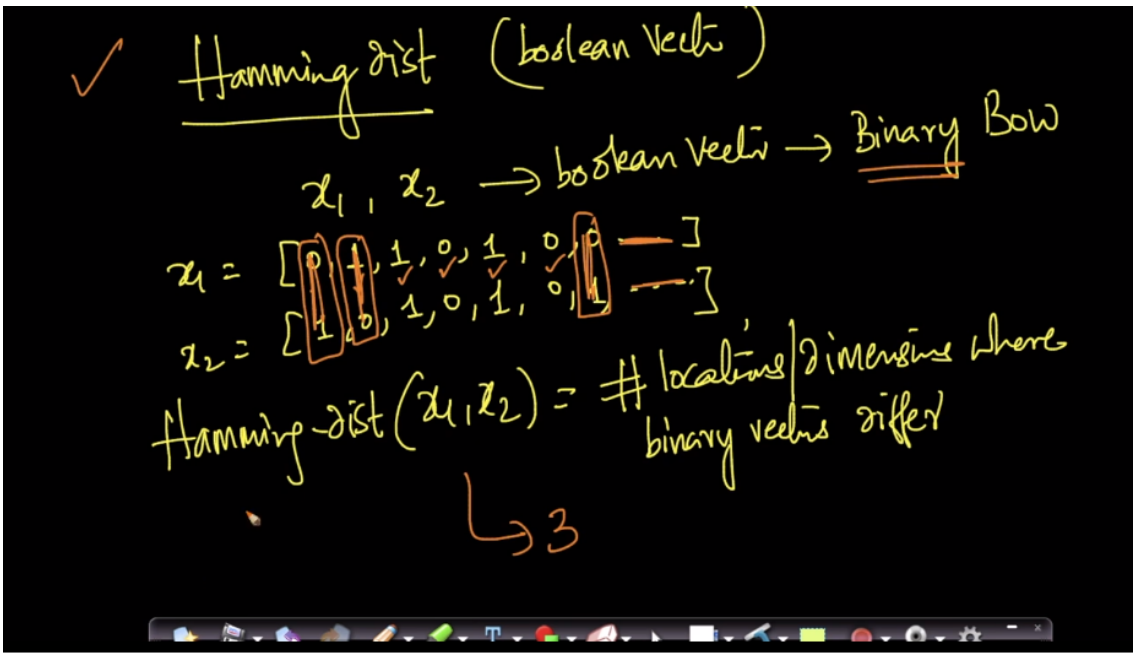


Lp norm of the vector:



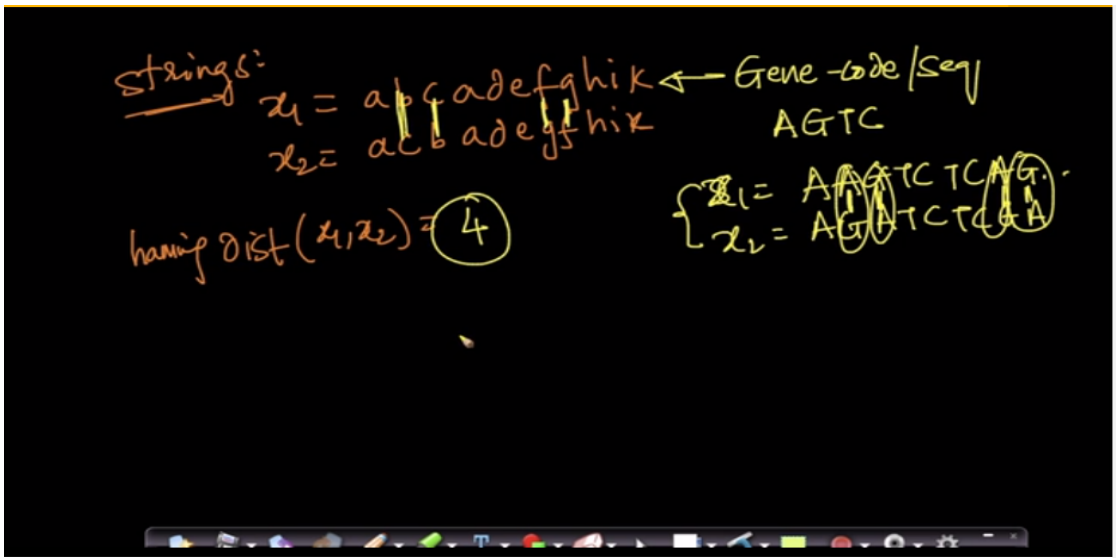
Hamming distance:

- It is used in text processing and for the binary vectors.
- Number of Locations where the binary vectors differ.



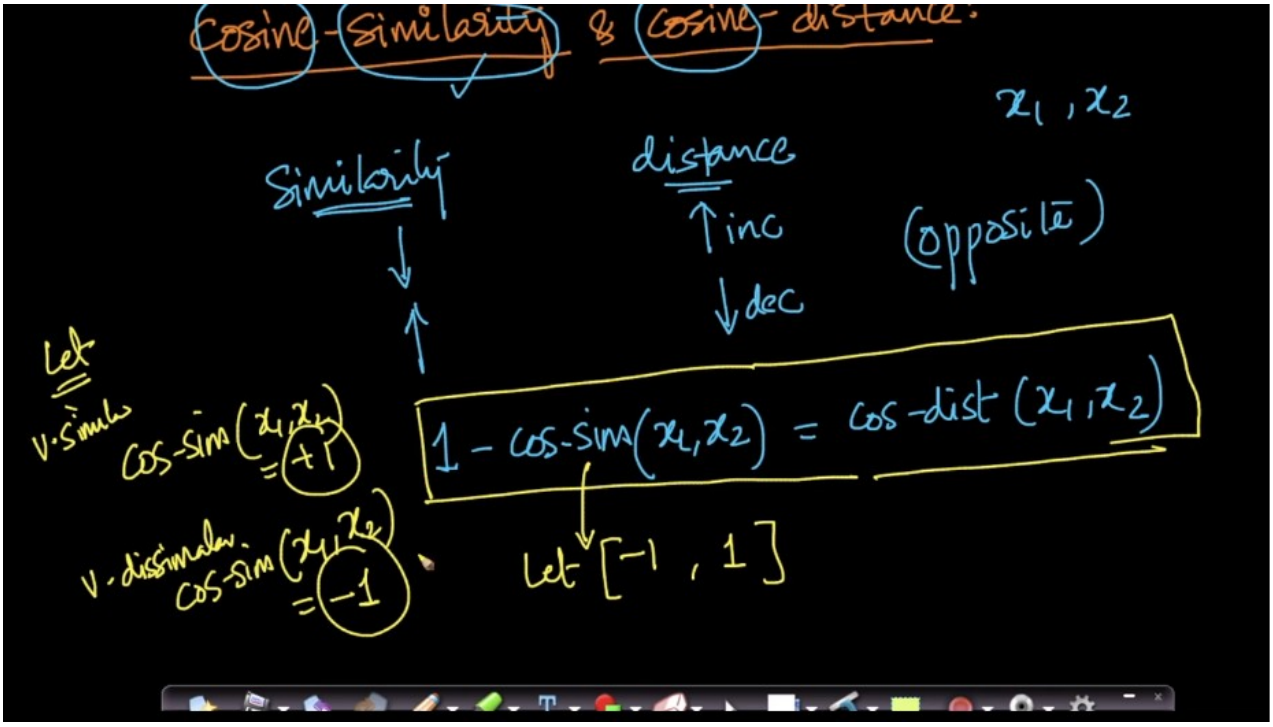
Hamming distances are also used in the strings.

- For gene codes we can use Hamming distances.



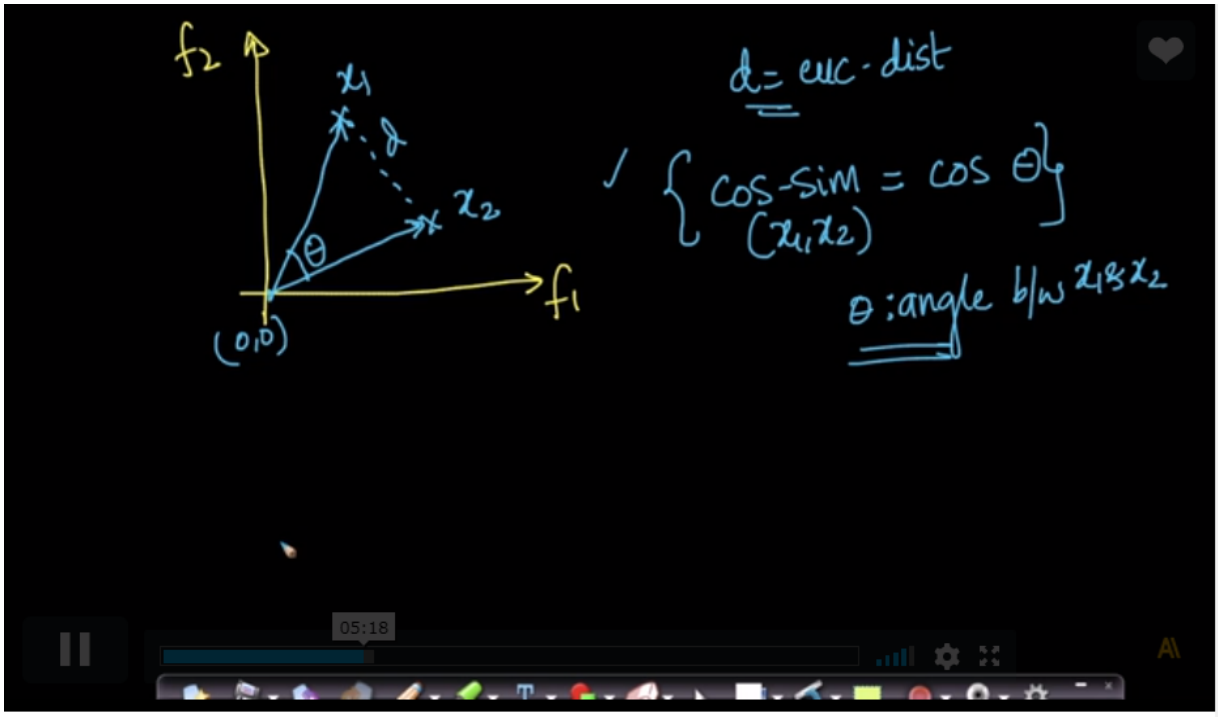
Cosine Distance and Cosine Similarity

- Relationship between similarity and distance.

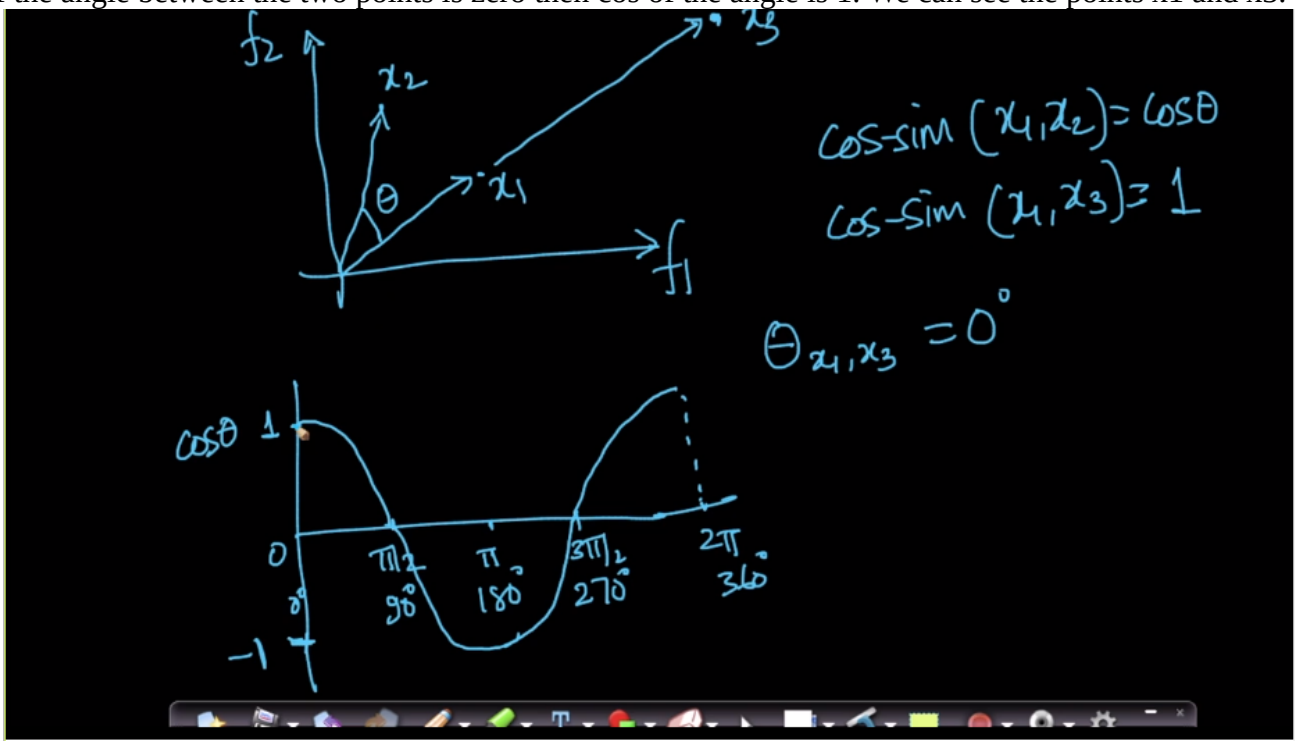


Cosine similarity:

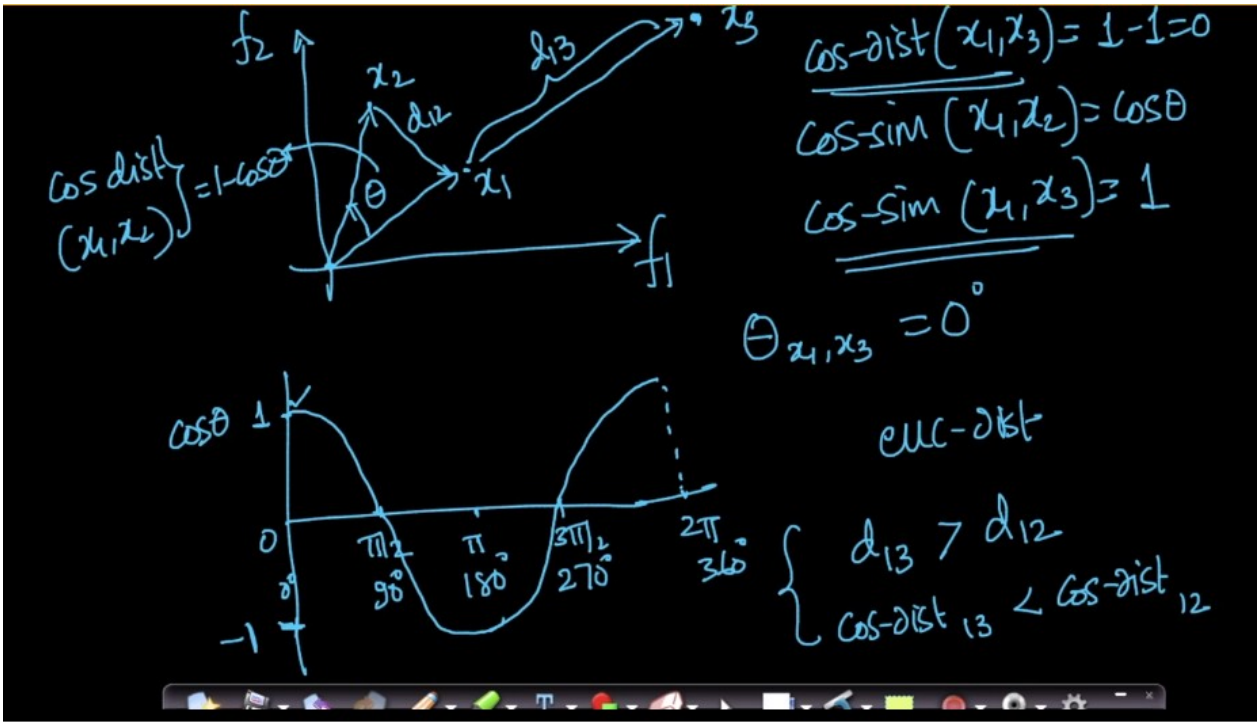
- cosine similarity is the cos(theta) where theta is the angle between the two points.



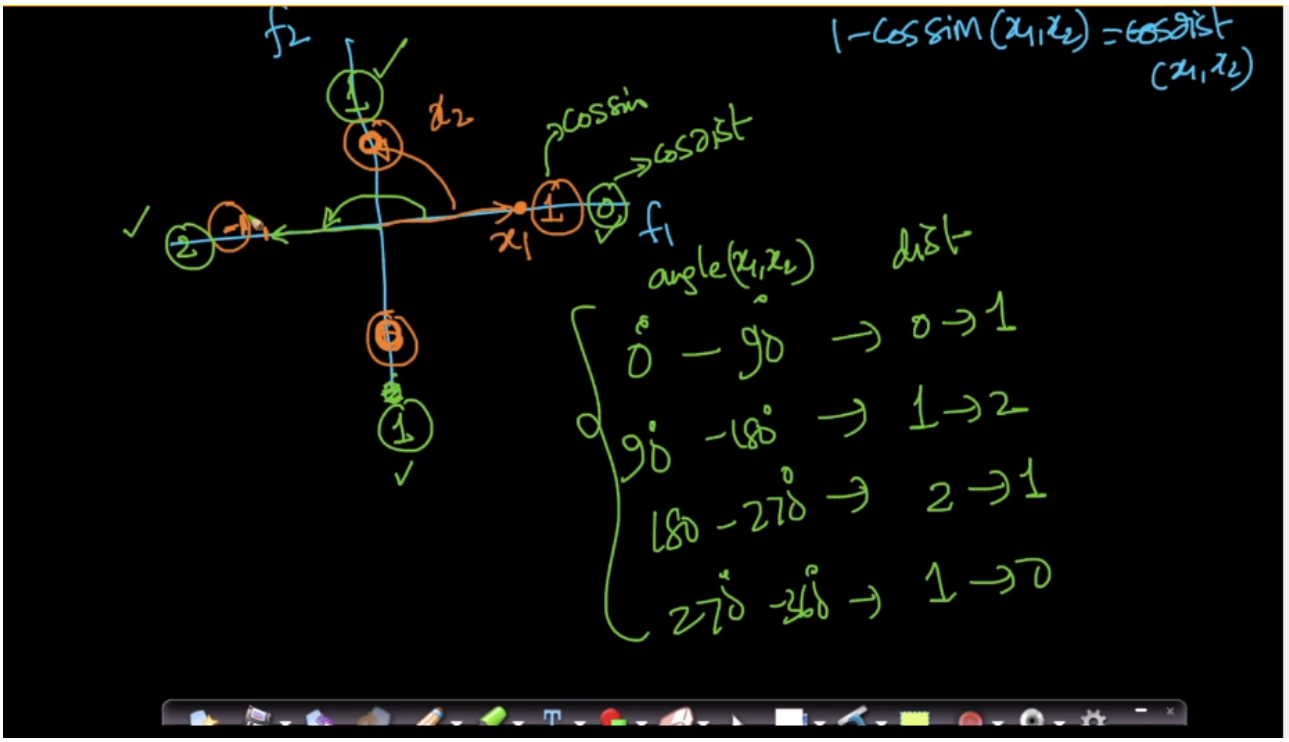
If the angle between the two points is zero then cos of the angle is 1. We can see the points x1 and x3.



- The cosine distance between x1 and x3 is 0 if the angle is also zero.

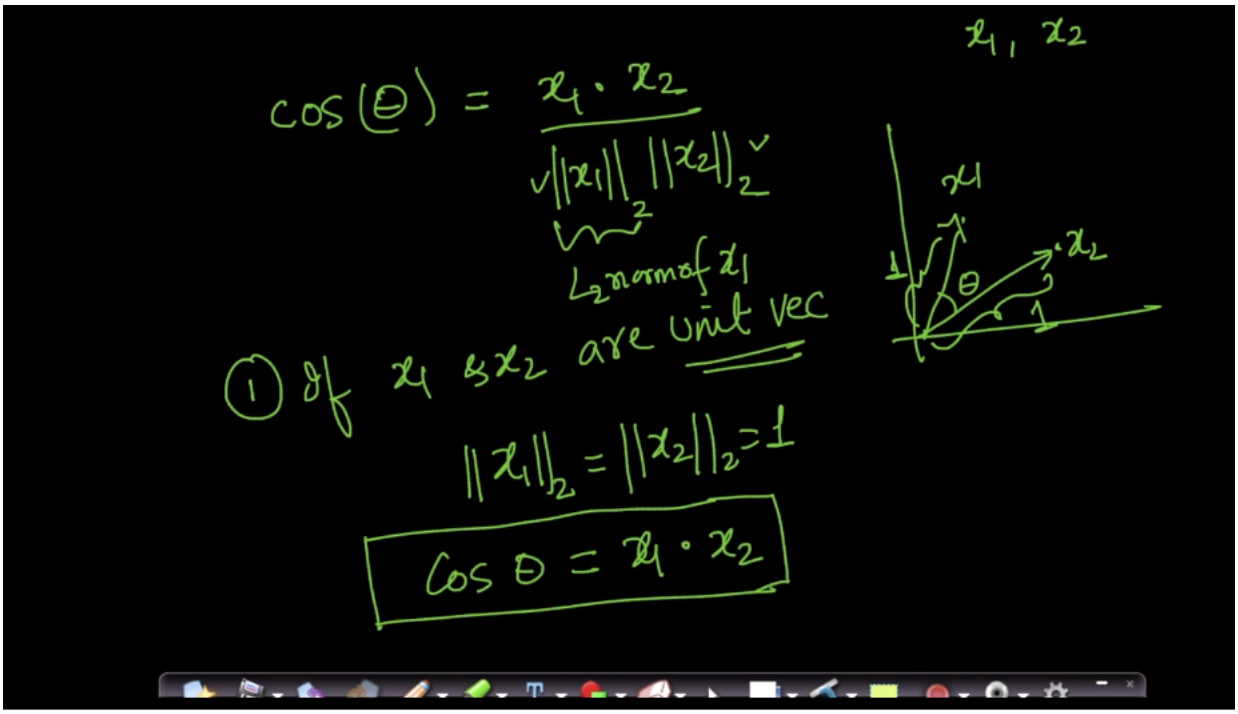


Various situations of cosine similarity and cosine distance.

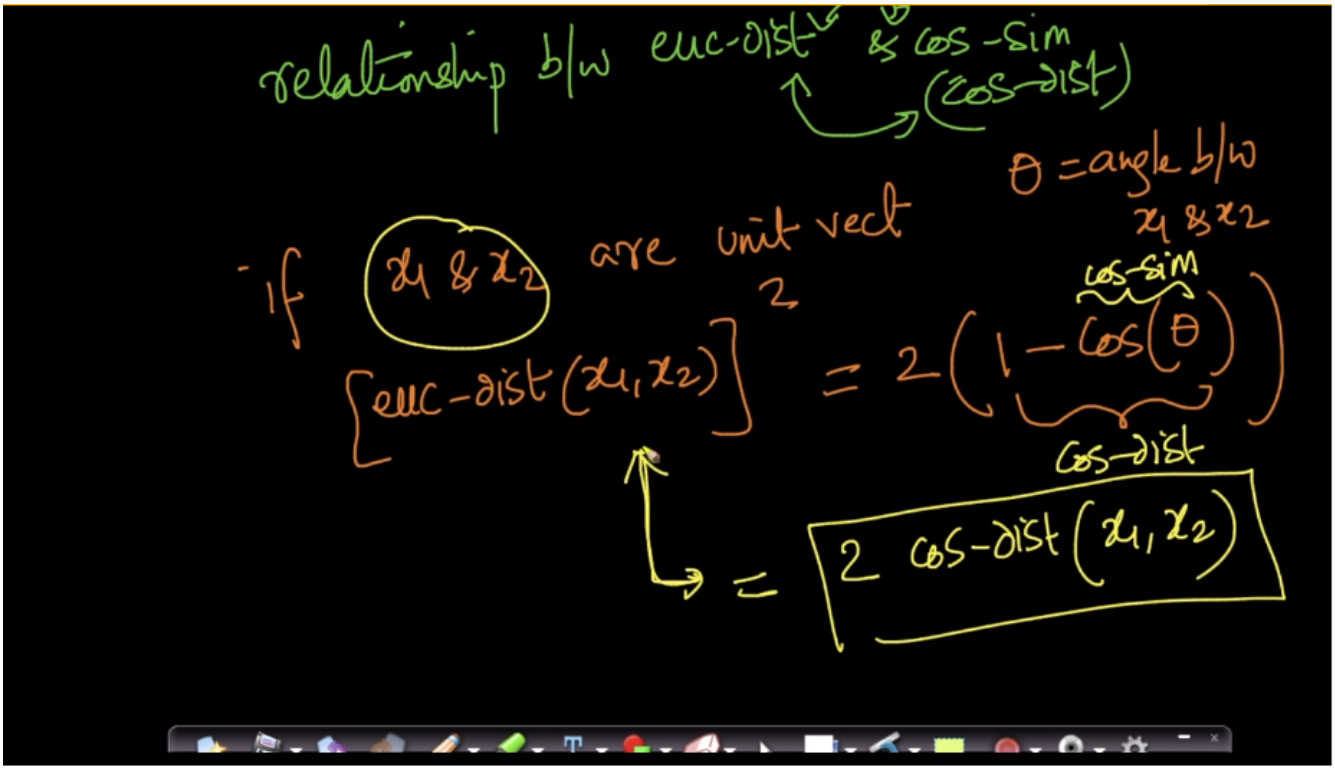


Formula for cosine similarity:

- If x1 and x2 are two unit vectors then the cosine similarity is dot product of the two data points x1 and x2.



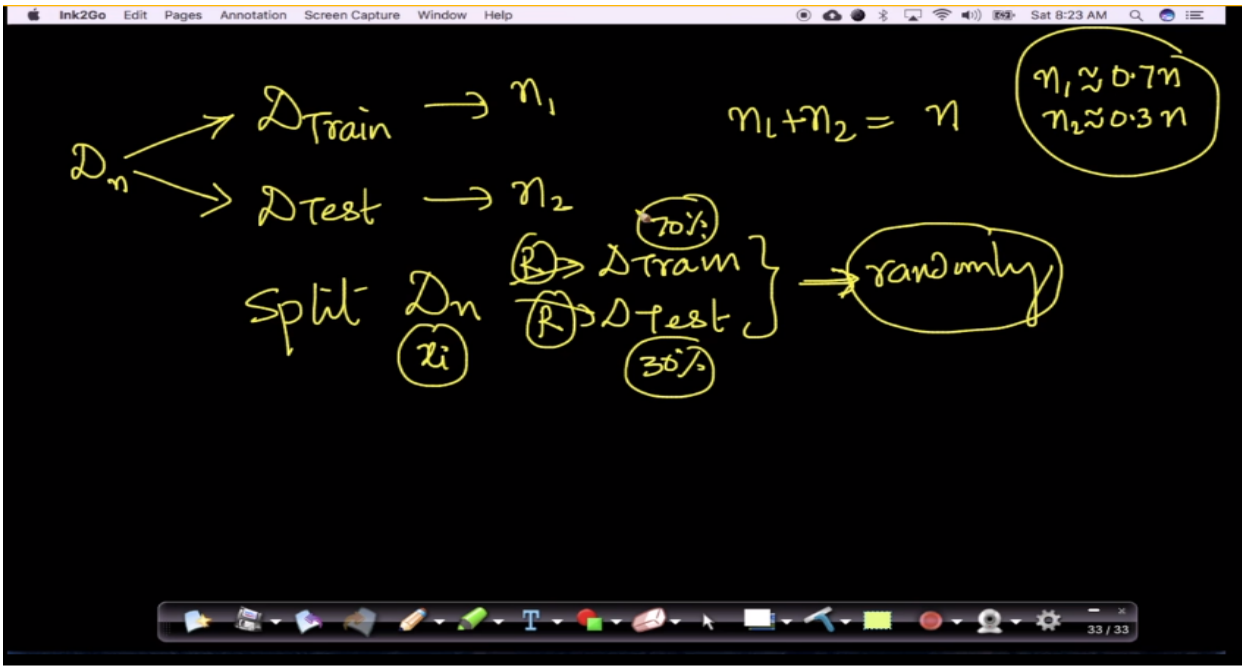
Relationship between euclidean distance and cosine similarity:



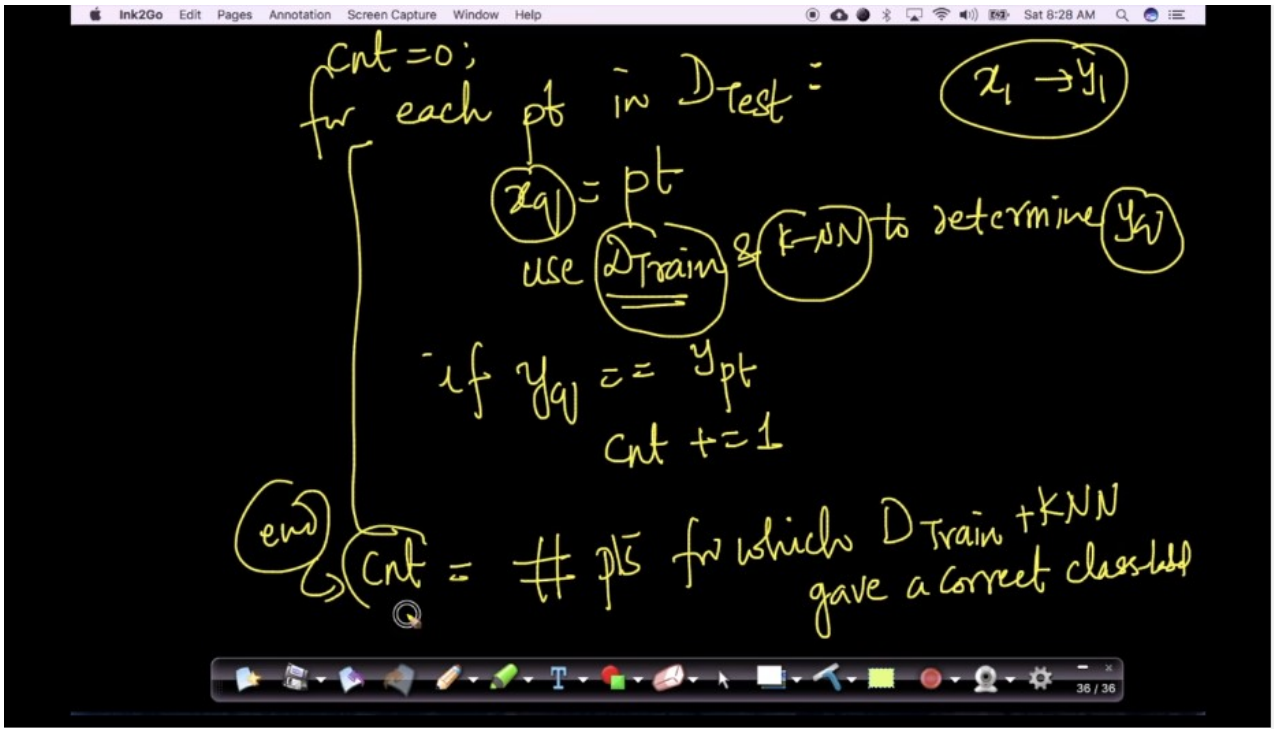
The above formulation is true if x1 and x2 are unit vectors.

Measuring the effectiveness of k-NN:

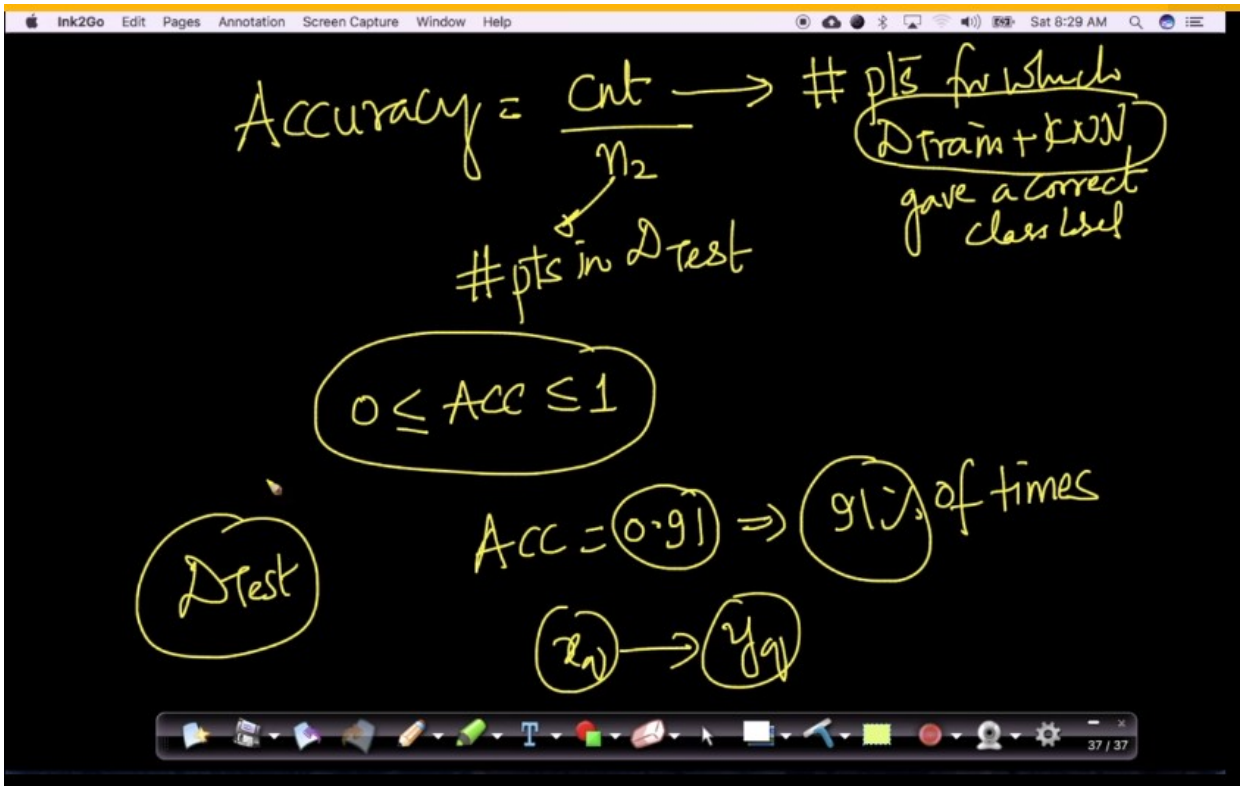
- The data set must be split into train and test datasets, the splitting must be done randomly.



Implementing the KNN on test data points one at a time.

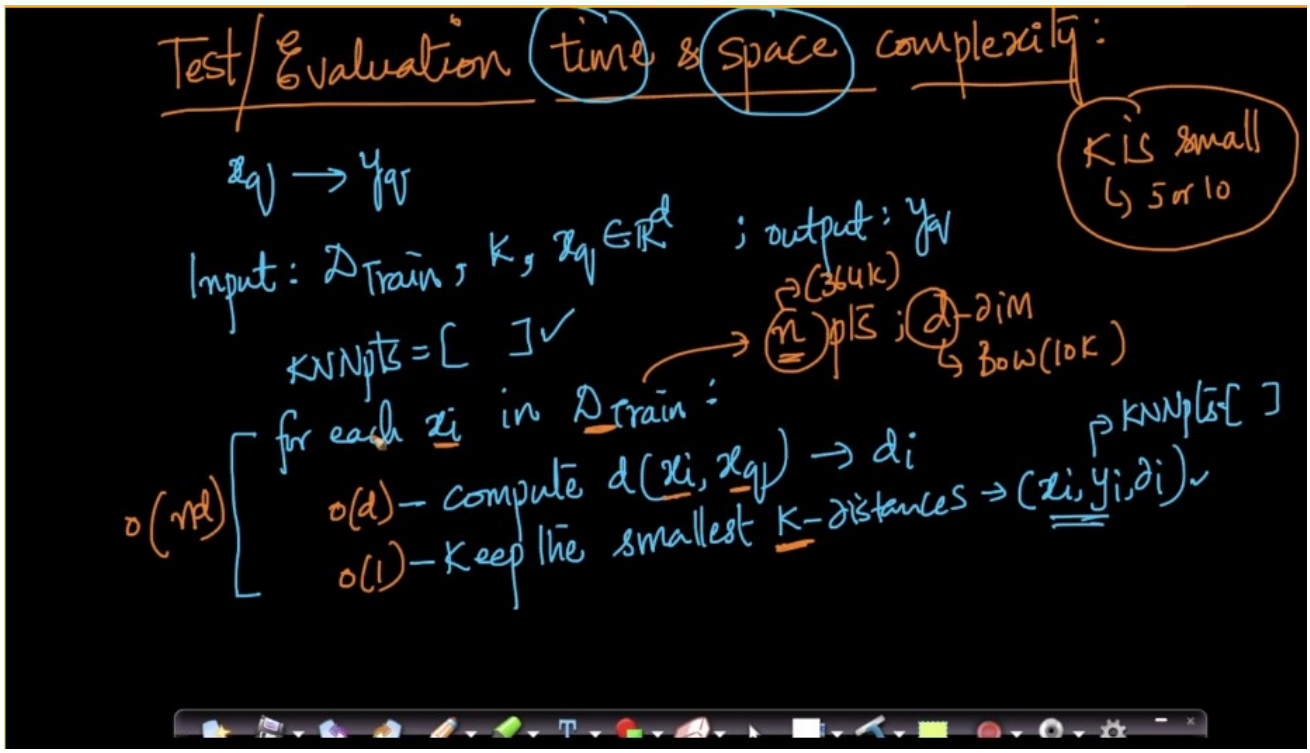


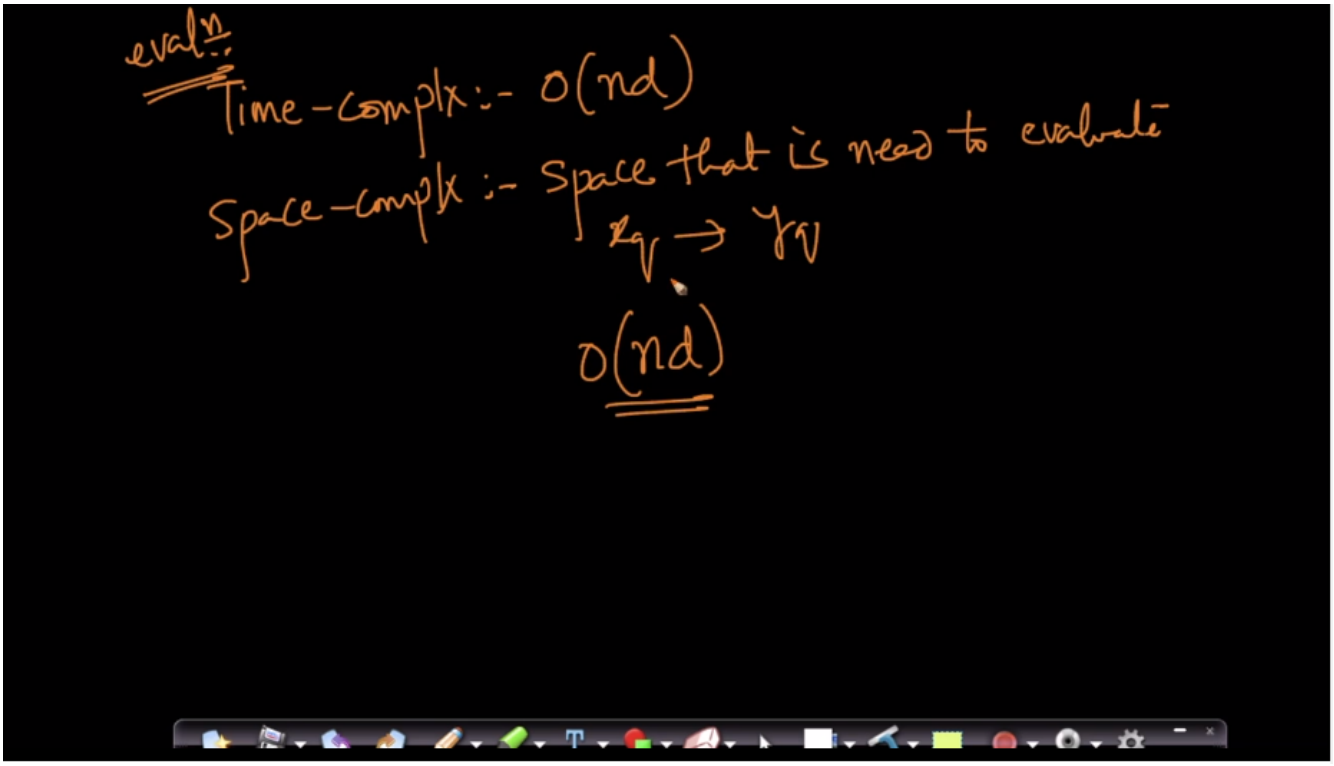
- Accuracy measure of KNN



We can conclude the accuracy.

Time and space complexity of KNN:

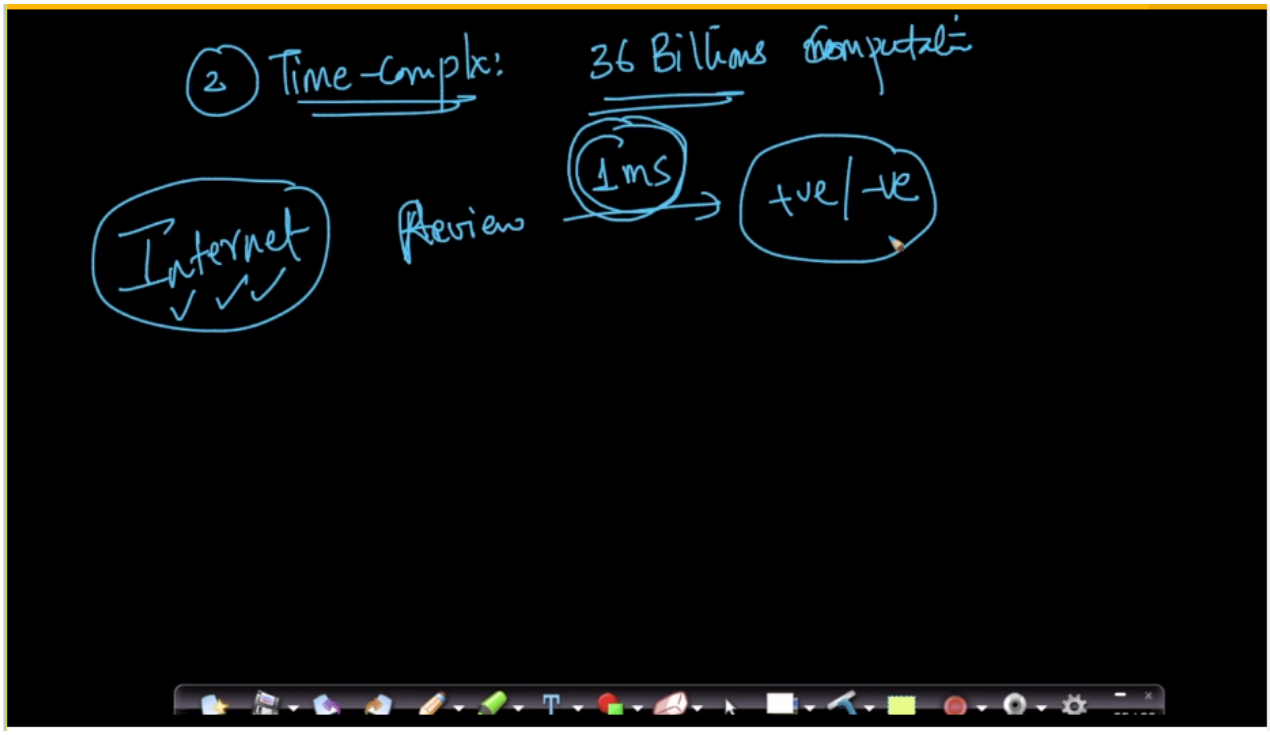




The space and time complexity is O(nd) for KNN which is not at all efficient.

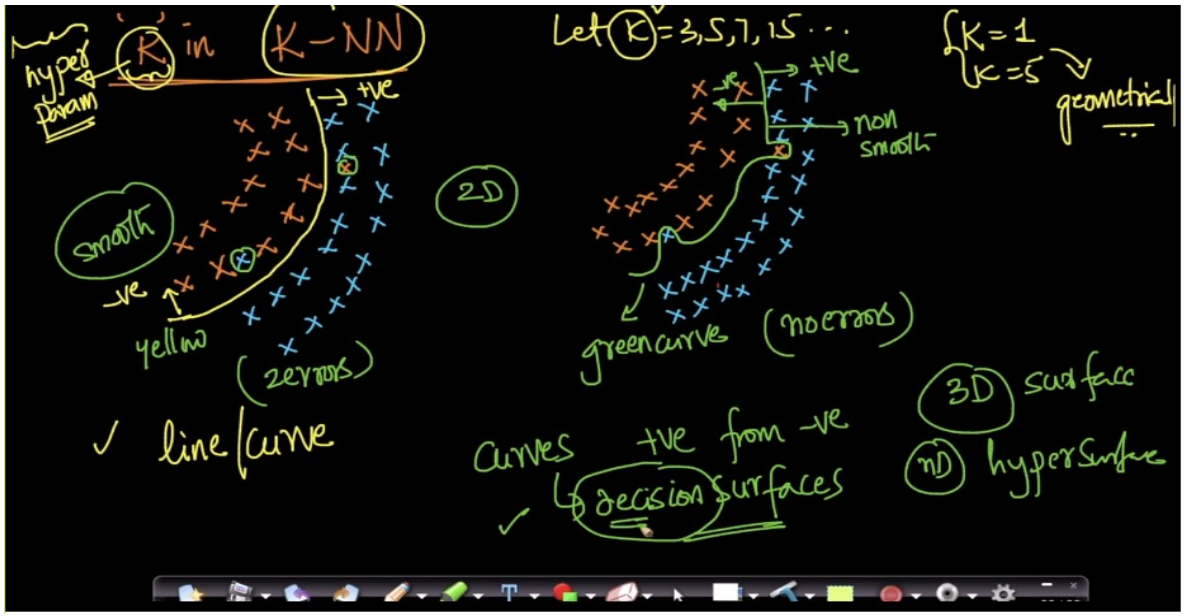
KNN Limitations:

- The algorithm occupies lot of memory.



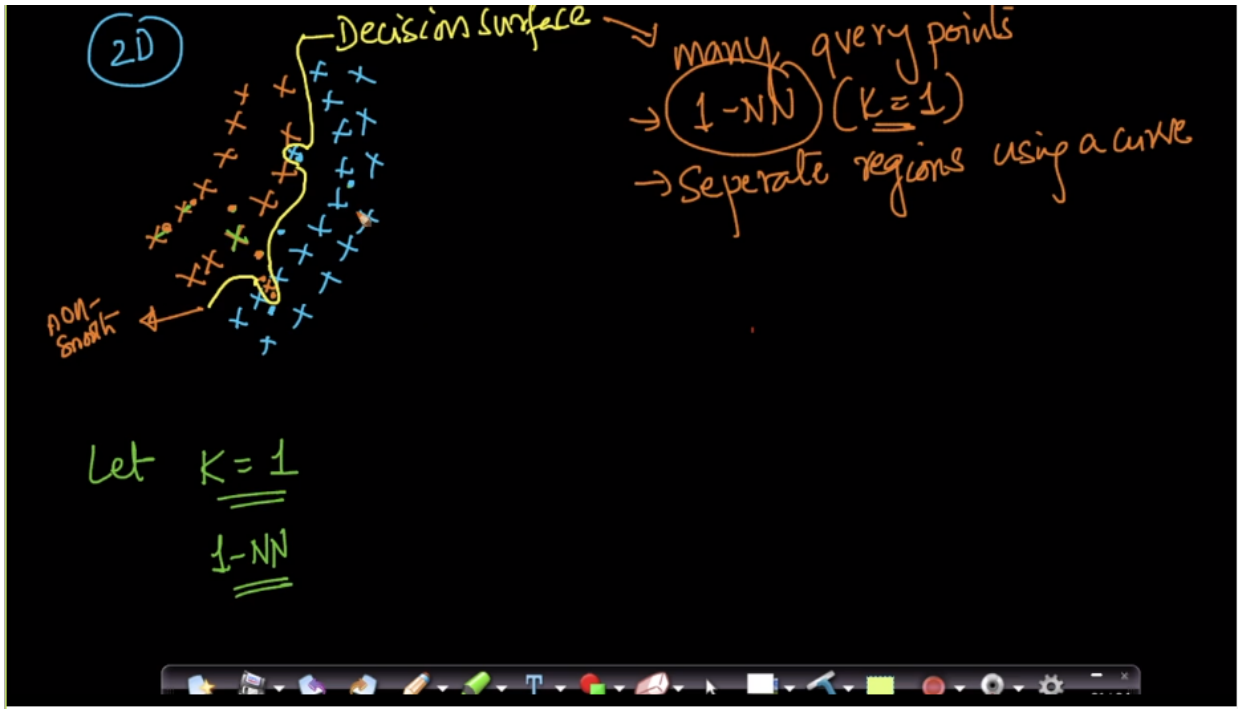
Decision surface for kNN as K changes:

- K is referred to as hyper parameter.
- Yellow curve makes errors and green curve makes no errors.
- These curves are called decision curves.



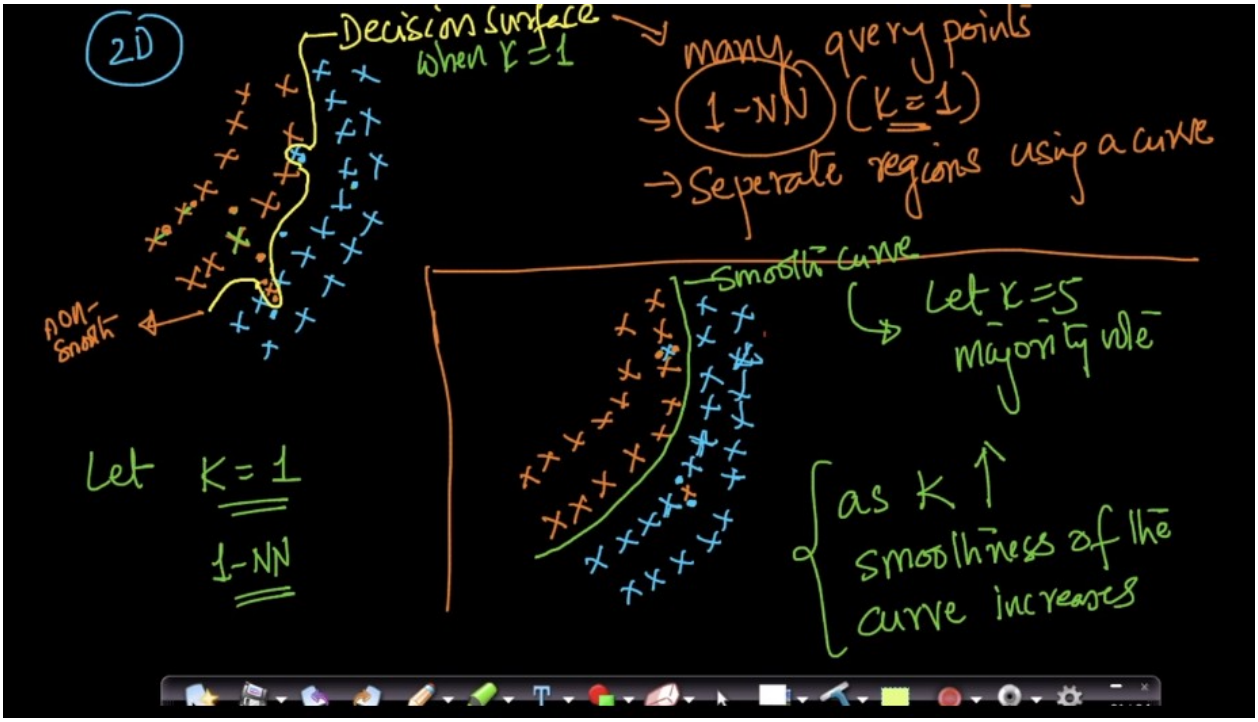
For k = 1

- The decision surface is drawn using many query points with k = 1.



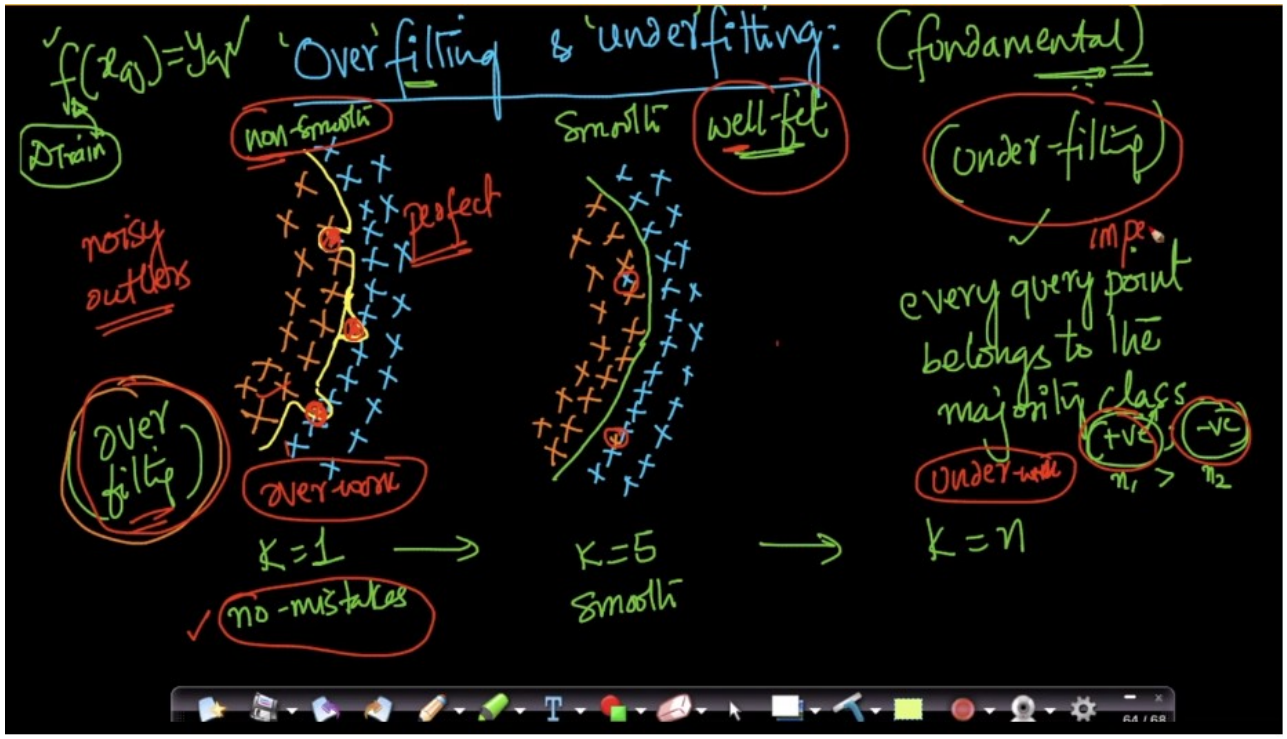
For k = 5

- As k increases the smoothness of the curve increases.



- For k = n
- The majority of the two classes is given as class label to the query point.

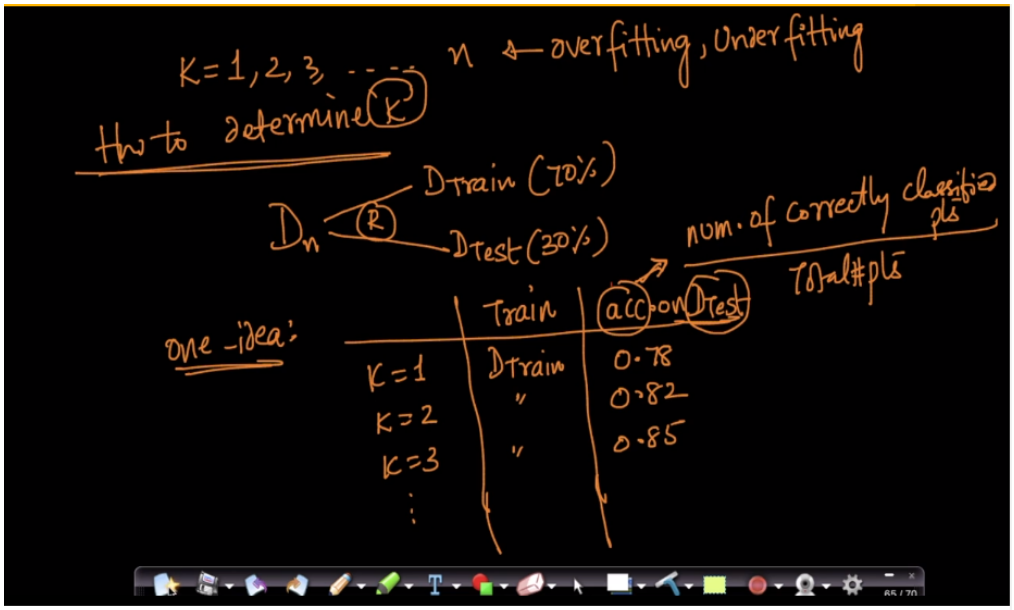
Over-fitting and Under-fitting



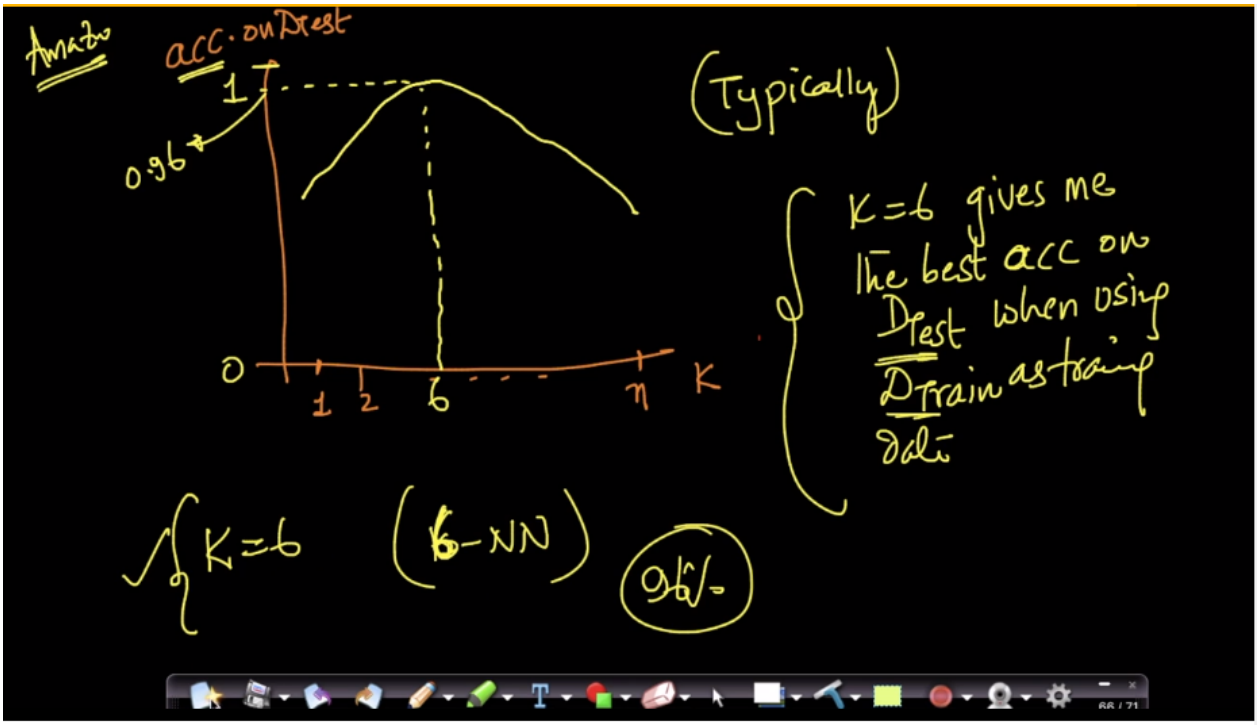
- The right value of K is useful for differentiating the outlier and random data point.

- The middle decision surface is considered as well fit K value.

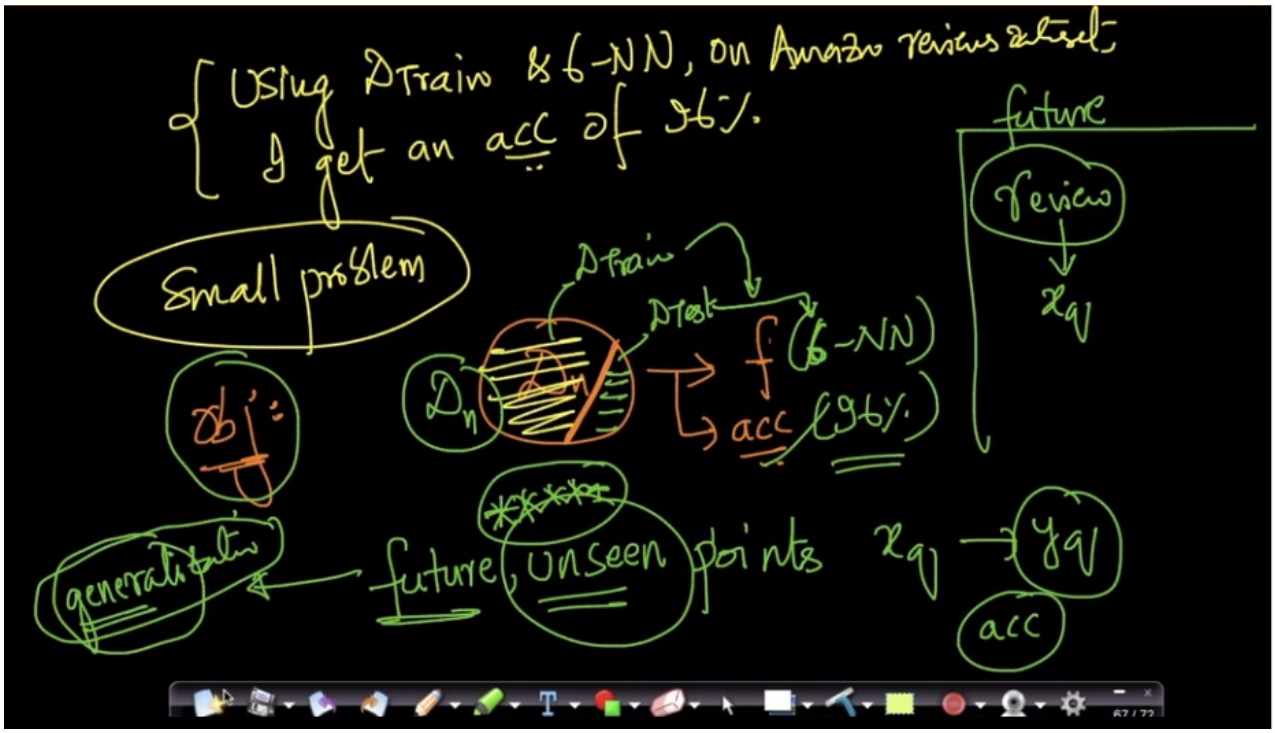
Need for cross validation



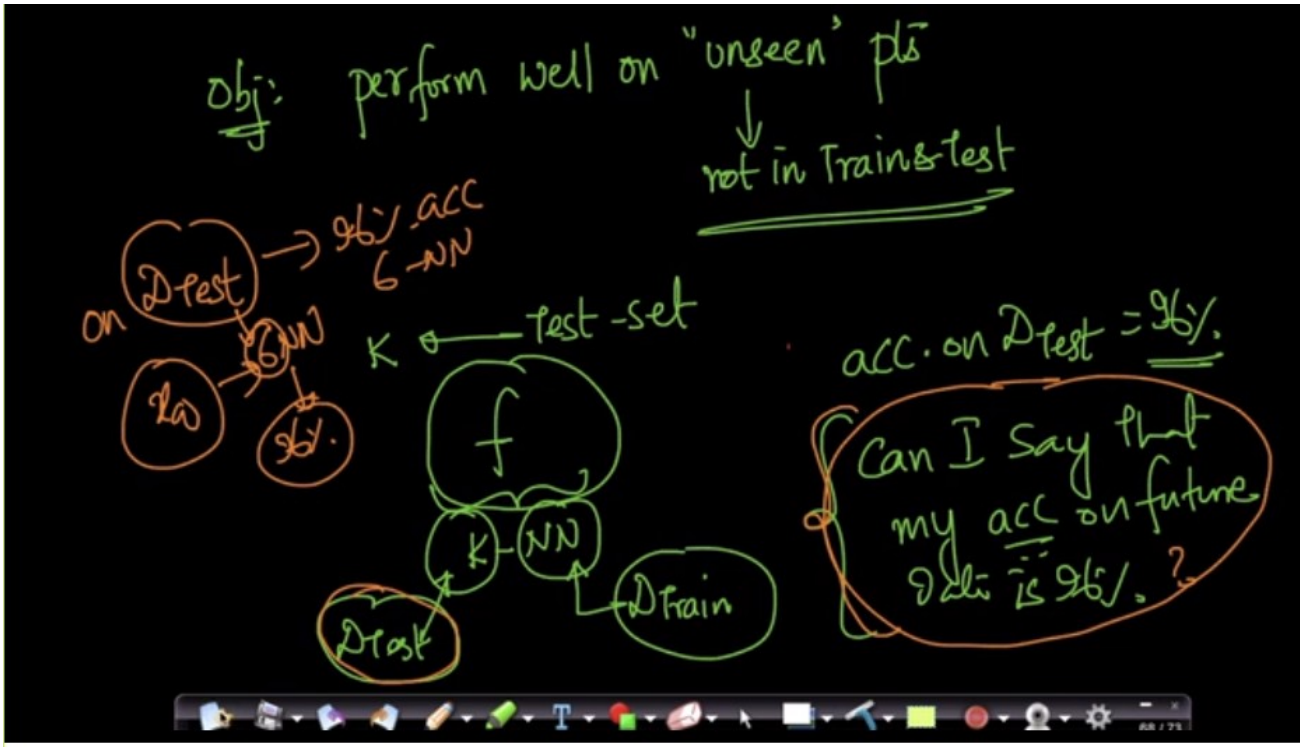
- plot Accuracy vs K values on the test data will give the right K value.



The value K=6 is choosen on the performance of the test data set only.

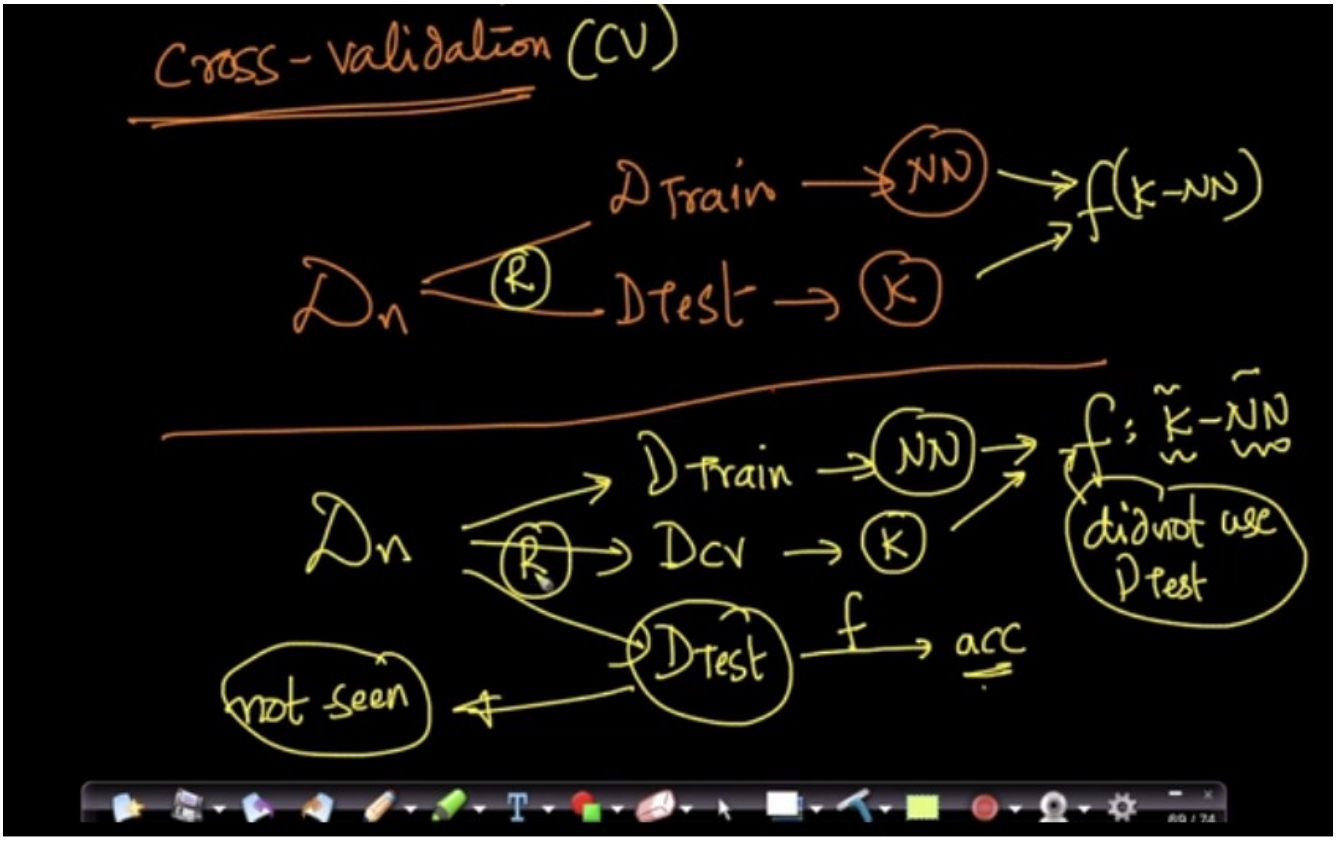


Our objective is to perform well on the **unseen** points.

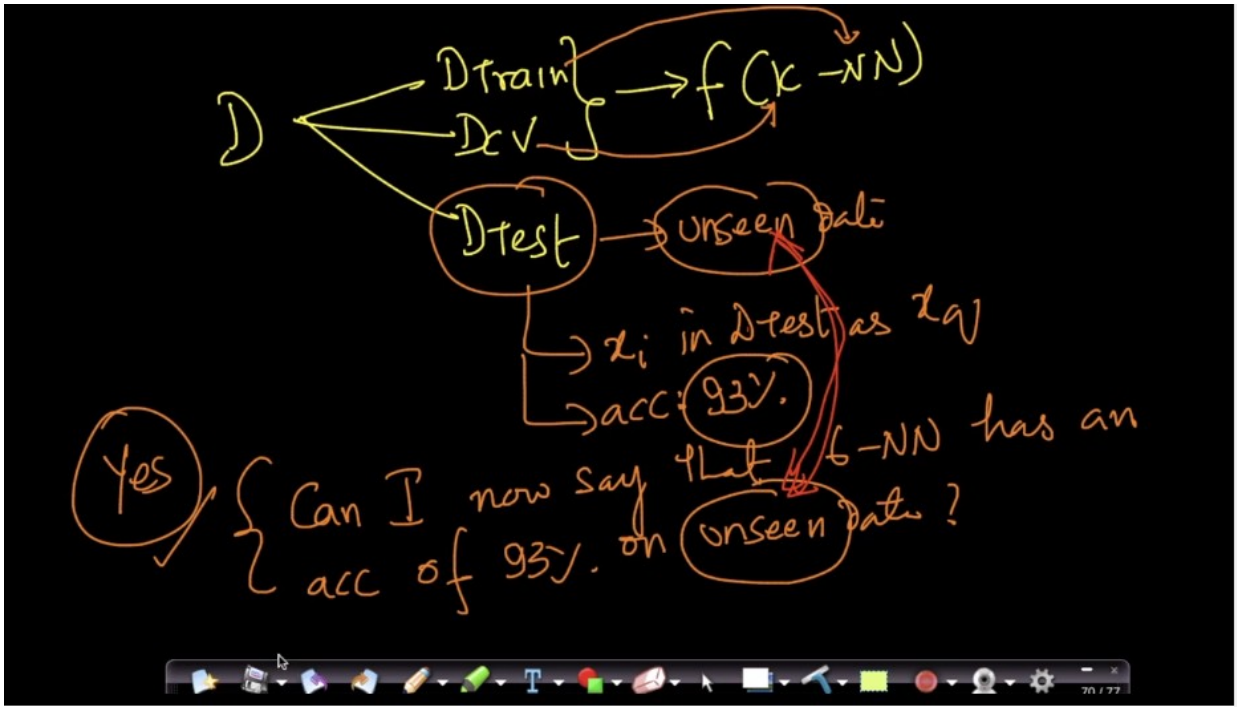


For calculation of the accuracy we only used the test data set. **We cannot say that our model will perform with the same accuracy on the future unseen data during deployment.**

Because of this problem we will use a technique called Cross Validation.



During the training process value of K can be found using Cross validation and test data is treated as unseen data.

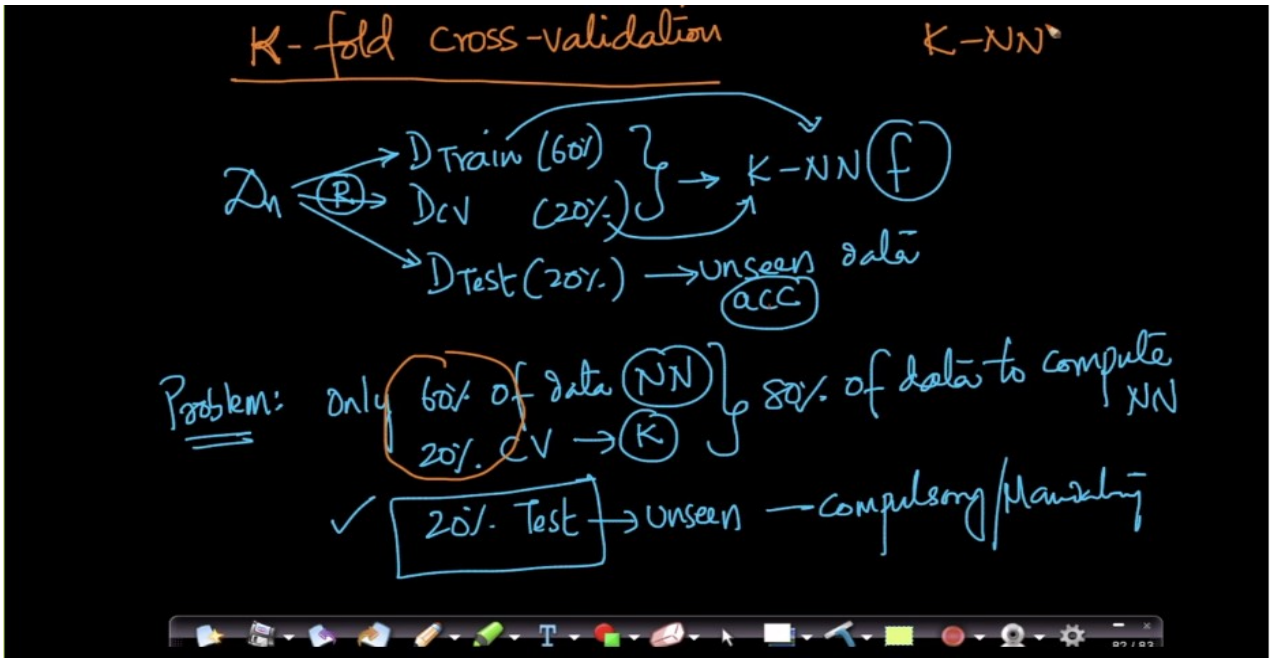


The breaking of the data into **Train, Dev and Test** is done randomly.

We obtain the nearest neighbors from the training data and right K value using the dev data set and test data set is unseen data. The accuracy on the unseen data set is known as generalization accuracy.

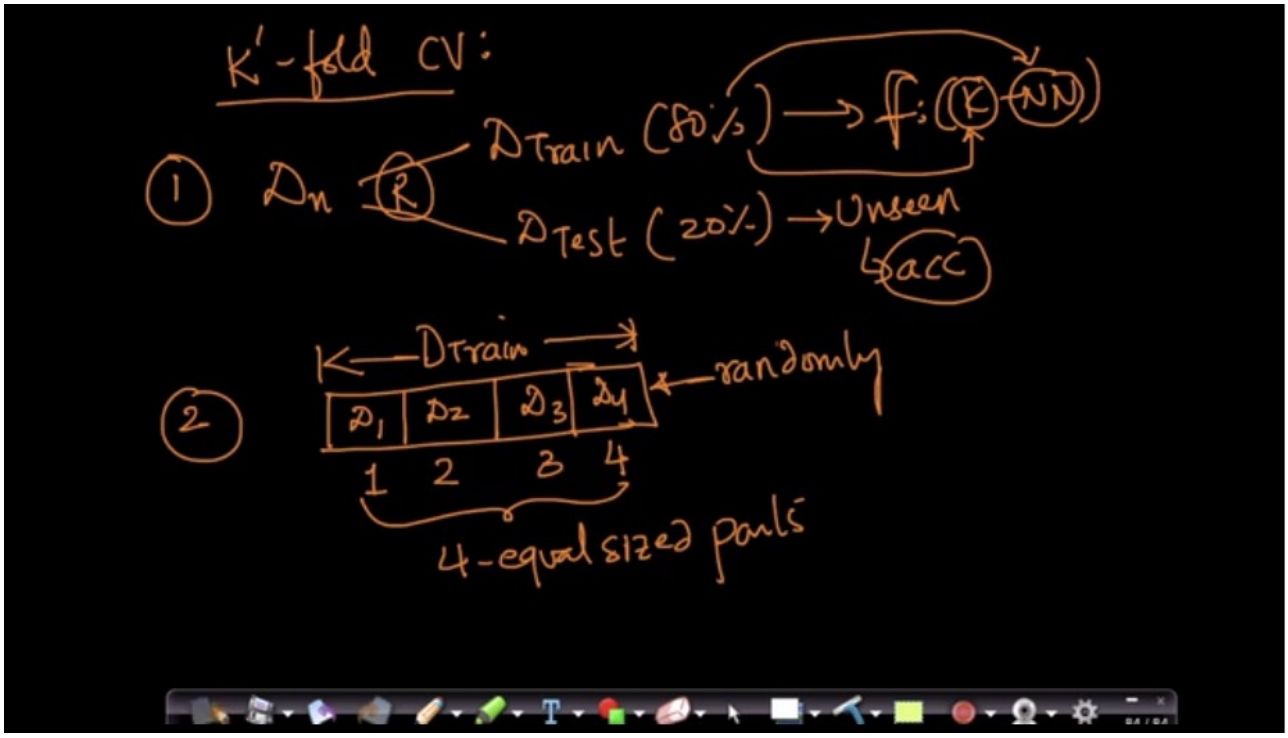
K – Fold Cross Validation:

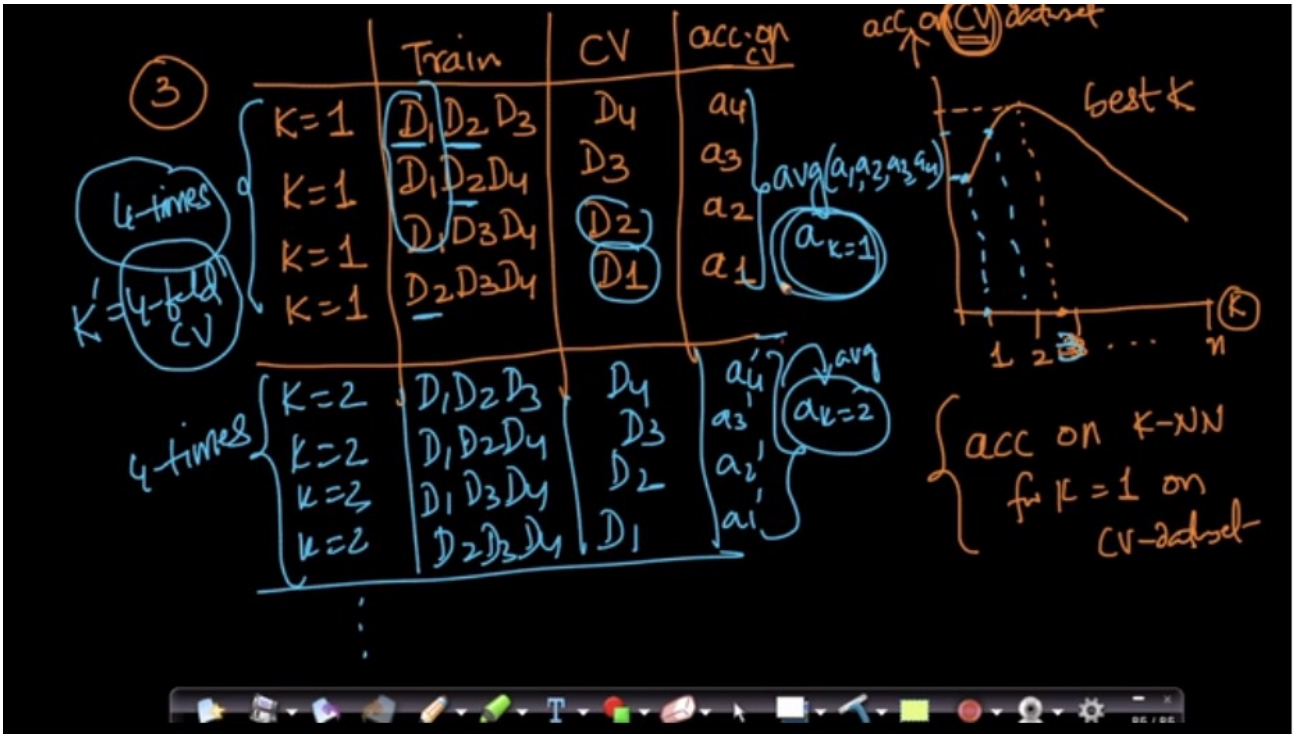
- In this technique we use the maximum possible training data for finding the correct K value.



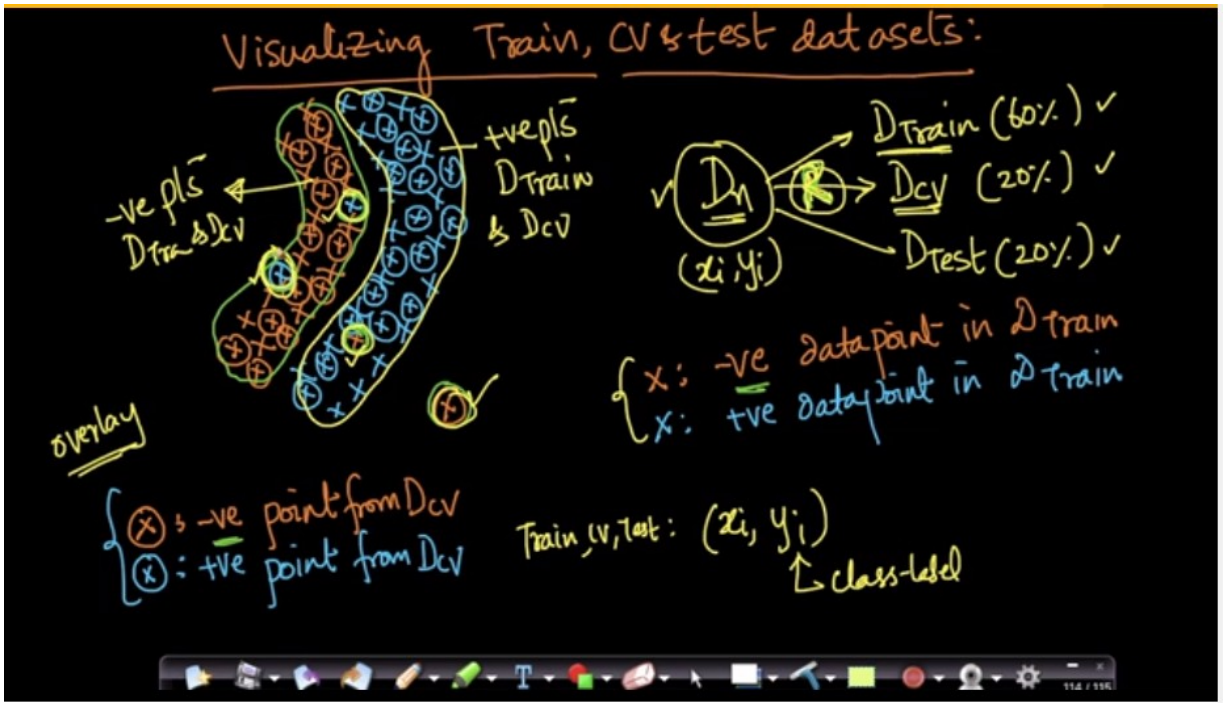
- Using the 80 percent of the data for determining the right K value is called K fold cross validation.

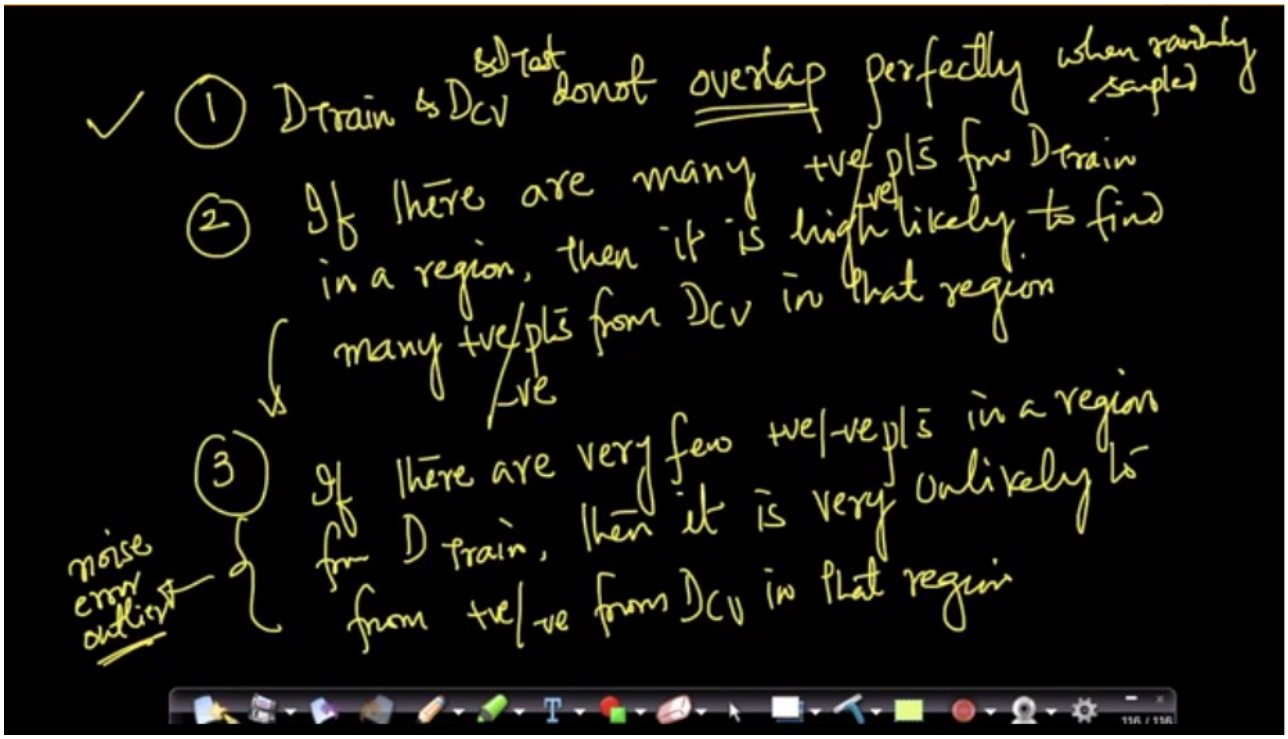
Different steps in K- fold cross validation





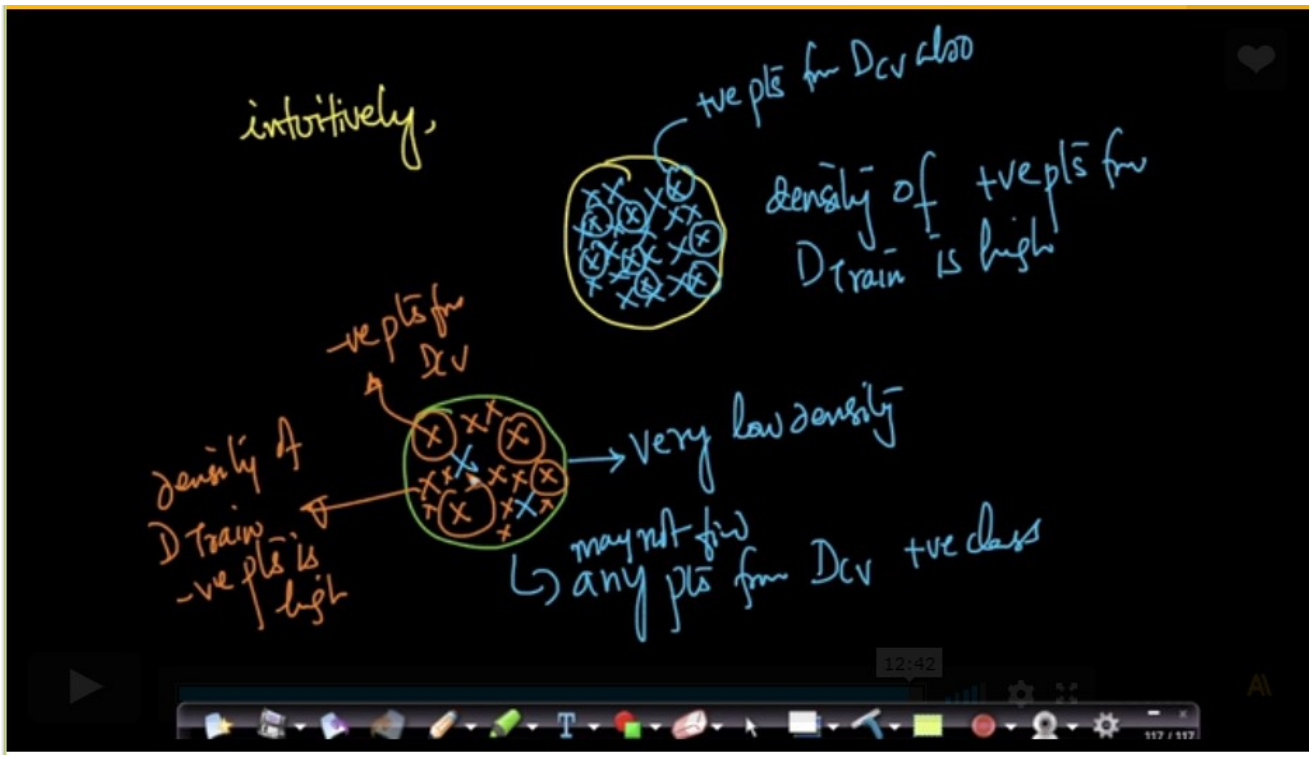
Visualizing Train, CV and Test Data sets:



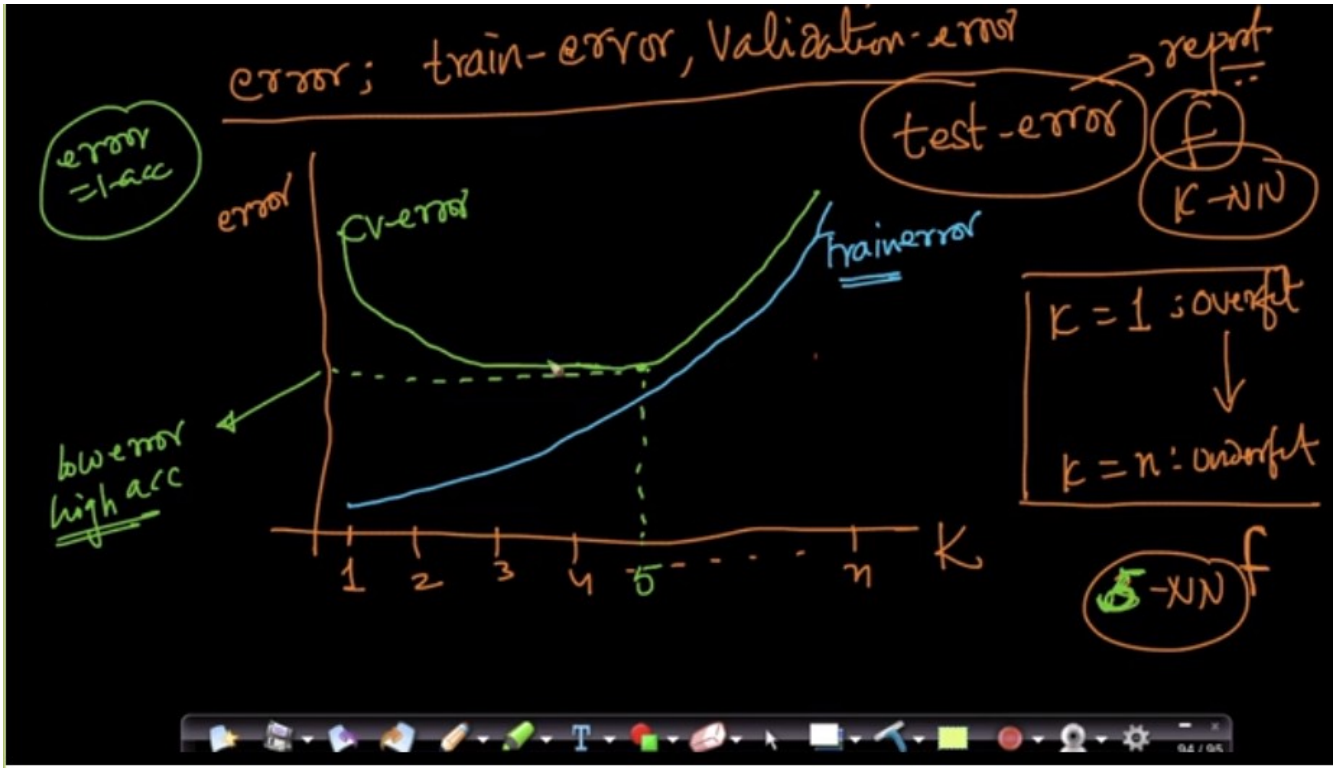


Intuitively we can think that if the density of the positive points is more in dtrain then there will be more points in dcv also.

If the density of the dtrain is low then the probability of finding the points from the dcv is also less of may not be seen. We can also find the negative points also.

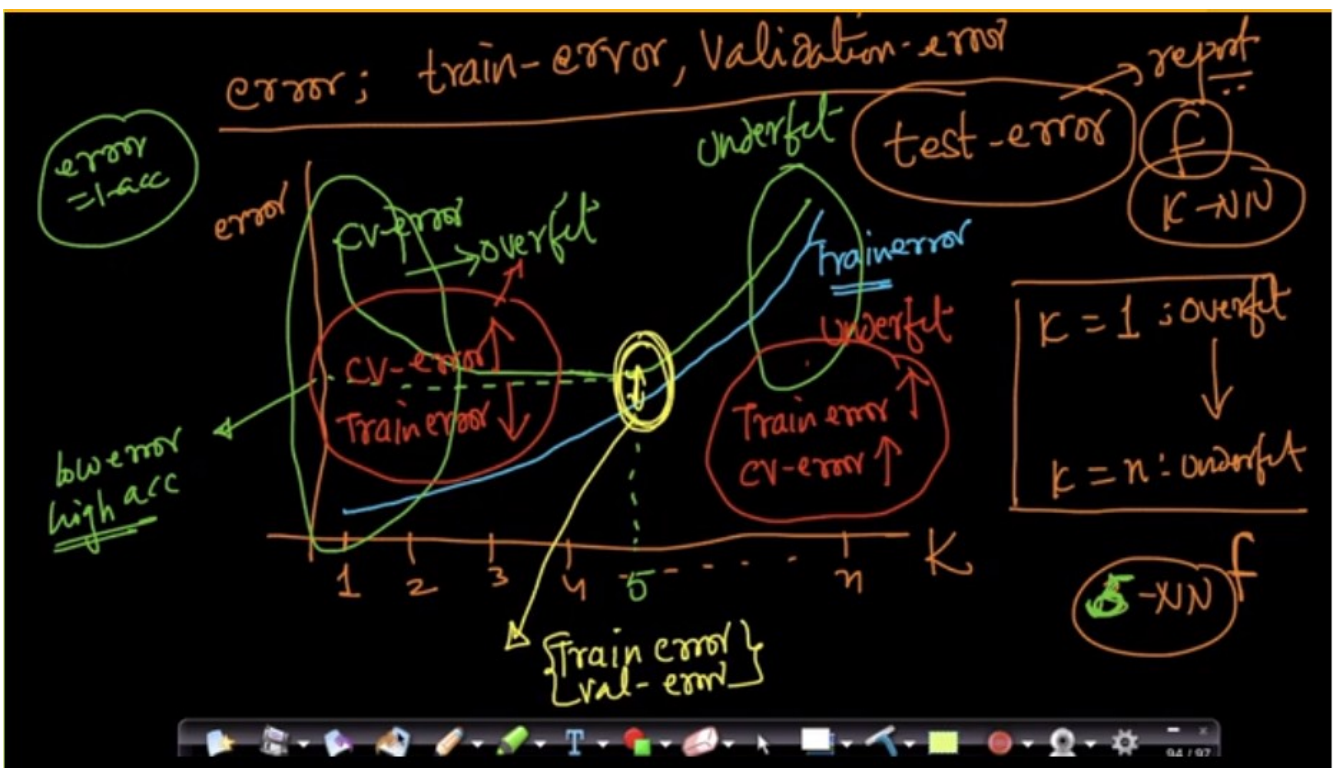


Relationship between train-error, validation-error.

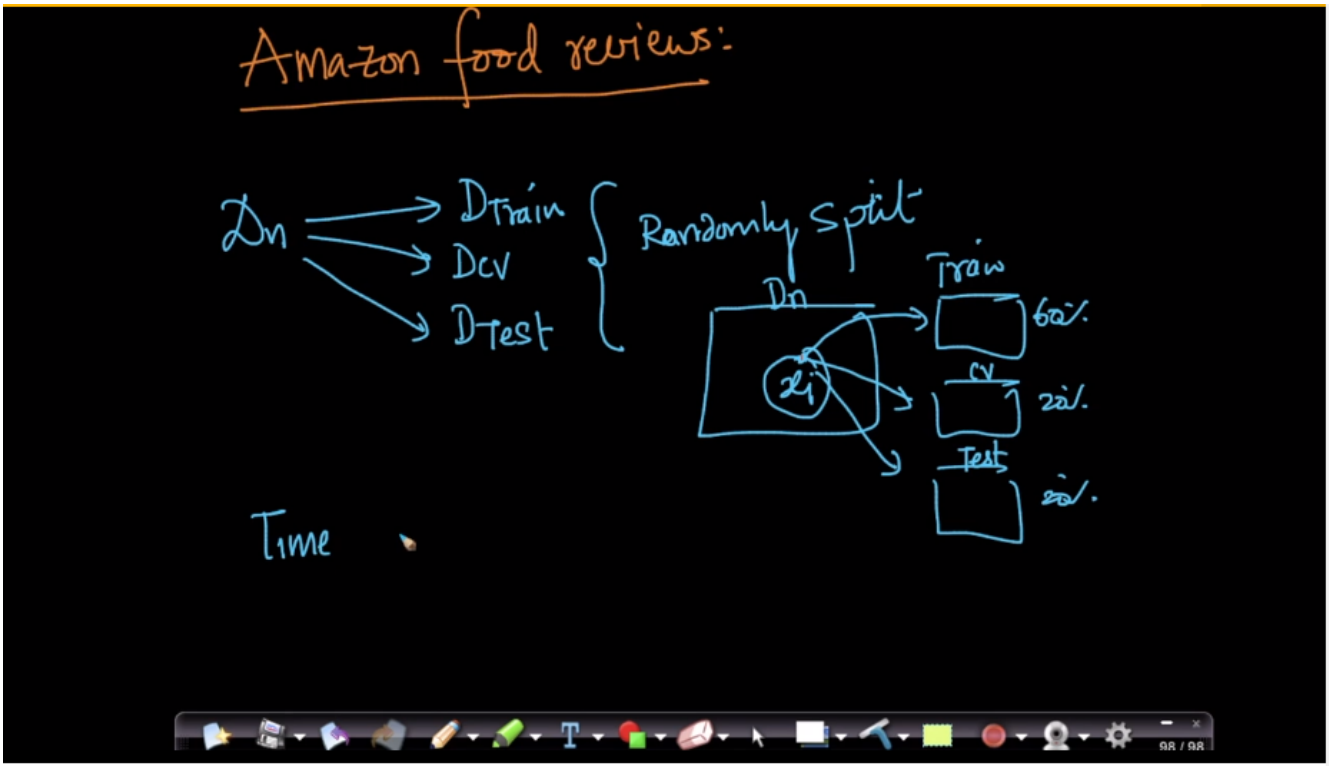


Train error is high
Val-error is high
— Underfit

Train error is low
Val-error is high
— overfit

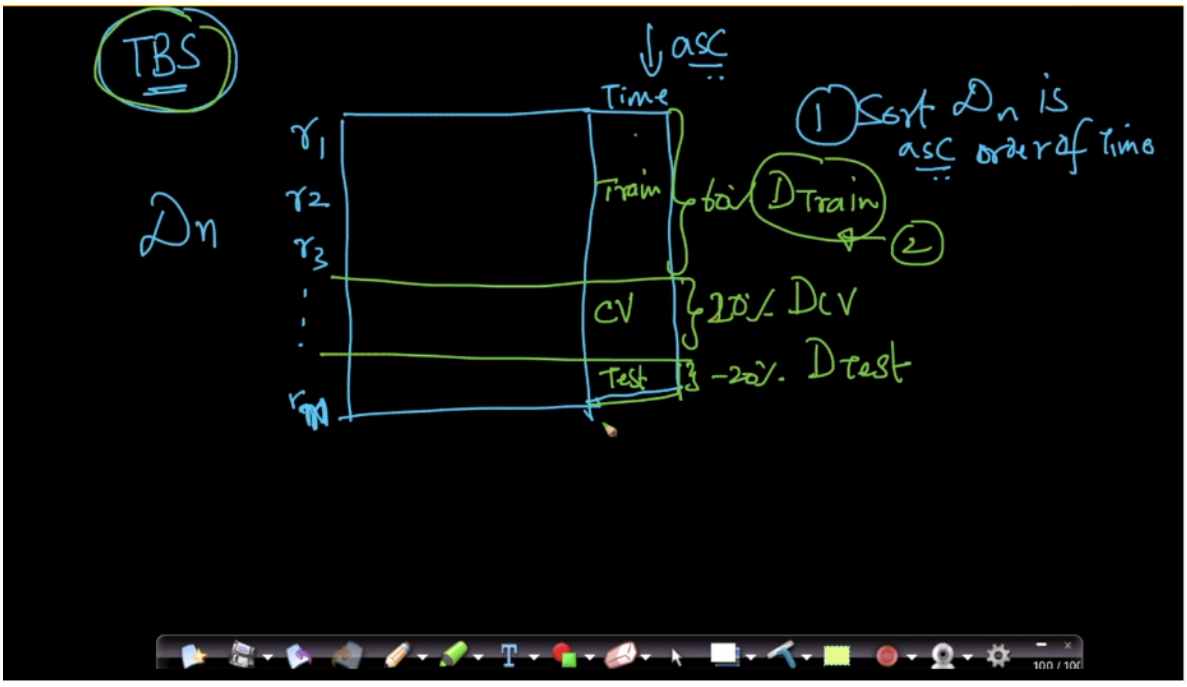


Time Based Splitting

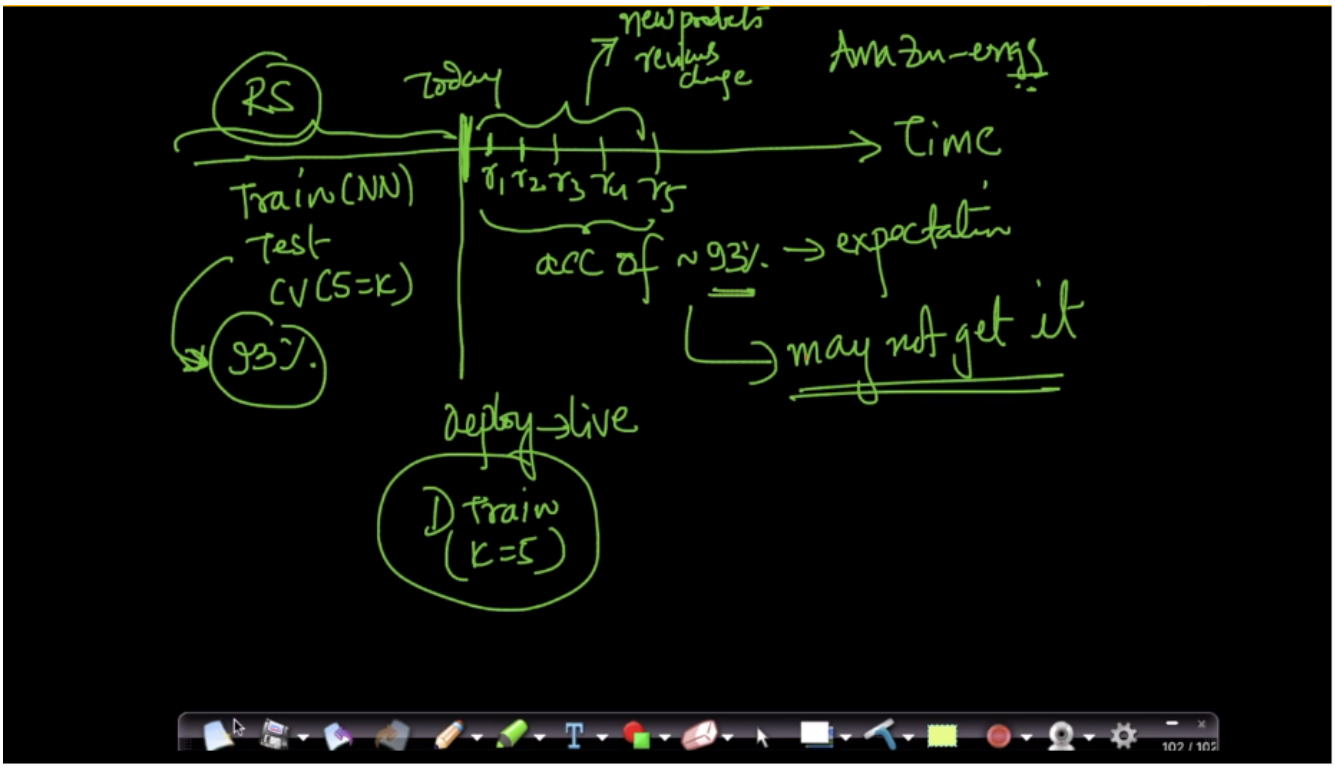


Here the train, cv and test data sets have the 60, 20 and 20 probabilities respectively of a random data point to be in the three of the splits.

Time based splitting the data :-

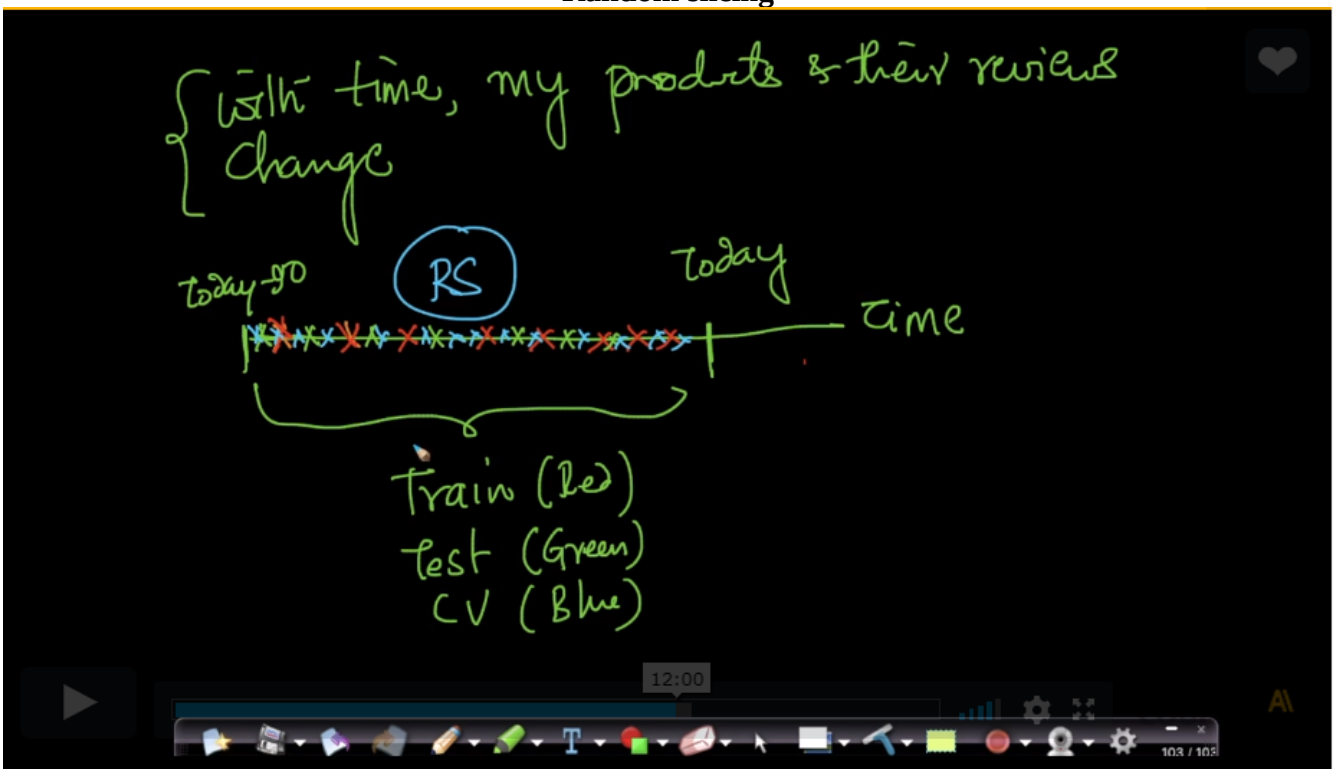


With time things change like product reviews and so on .. so the model does not perform well in case of random splitting.



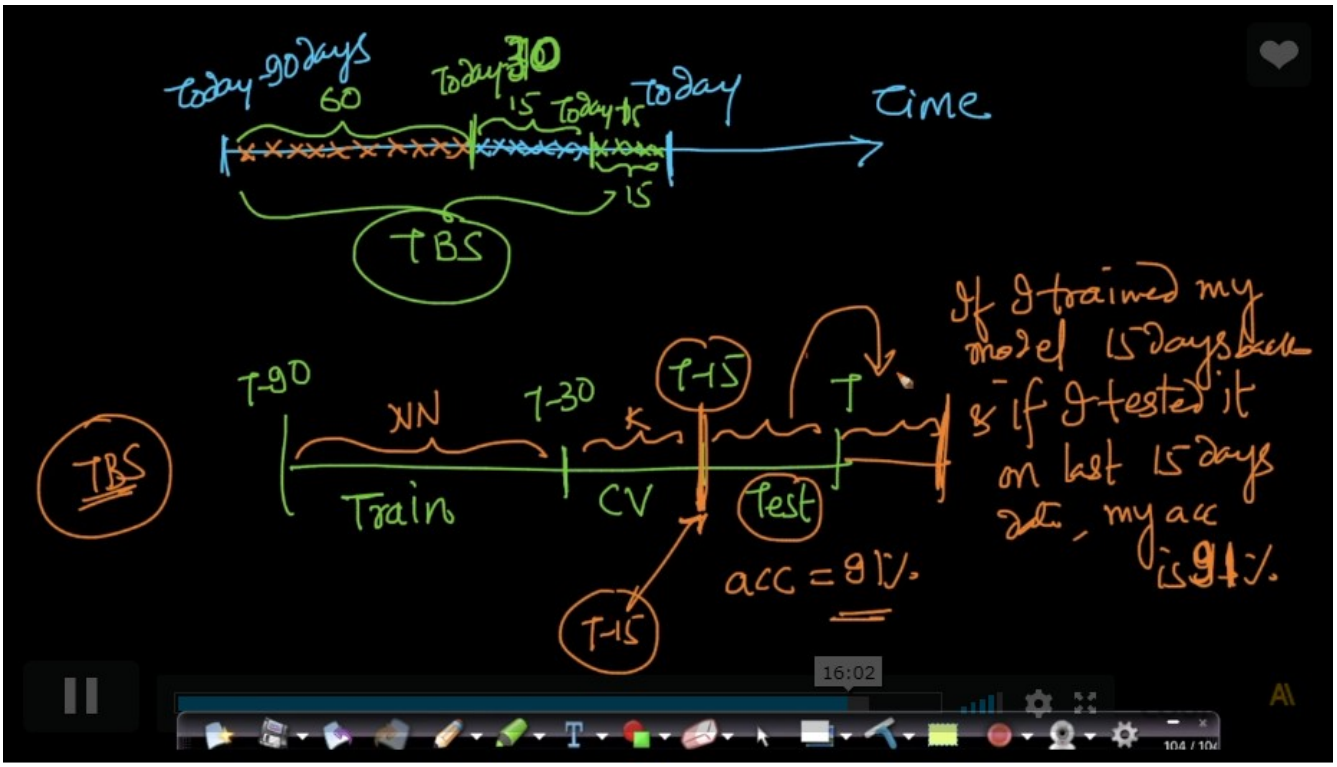
Hence, random splitting the dataset is not the efficient method.

Random slicing

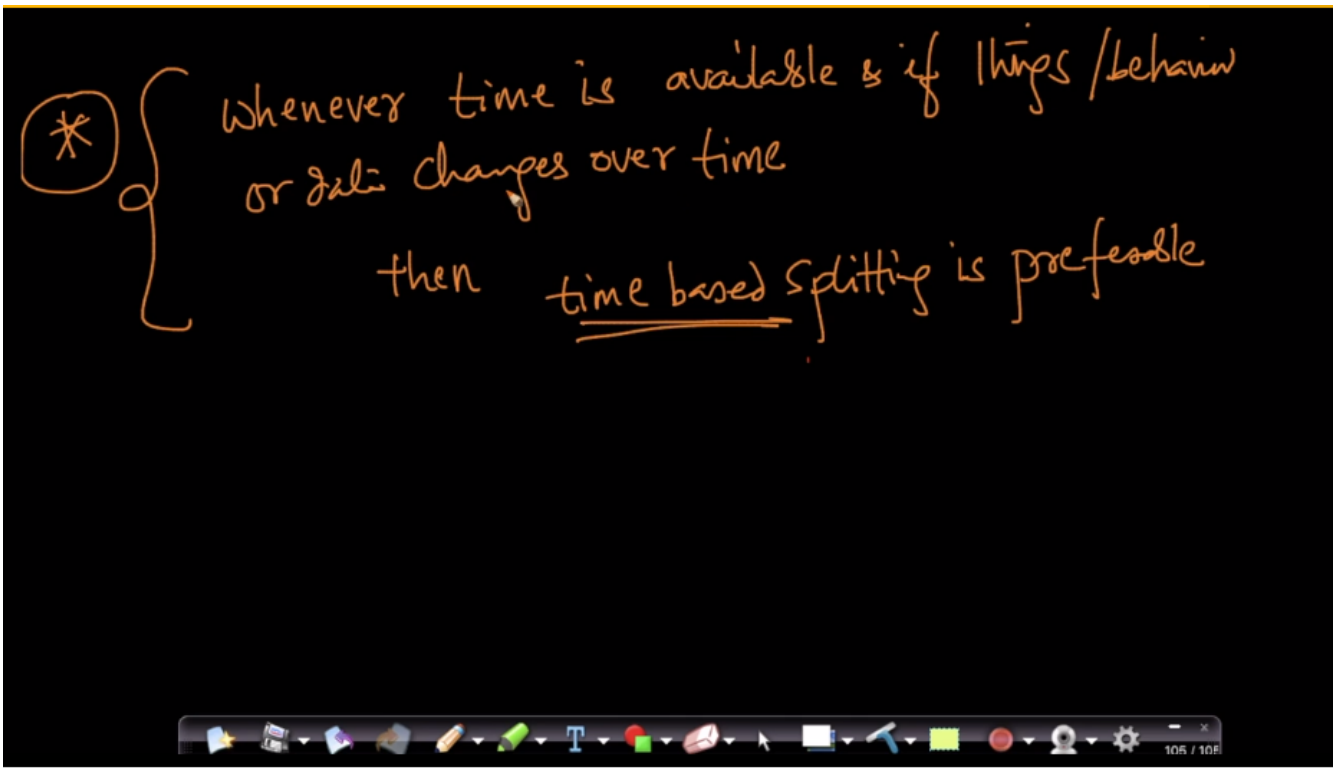


The train, dev and test data come from all over the place.

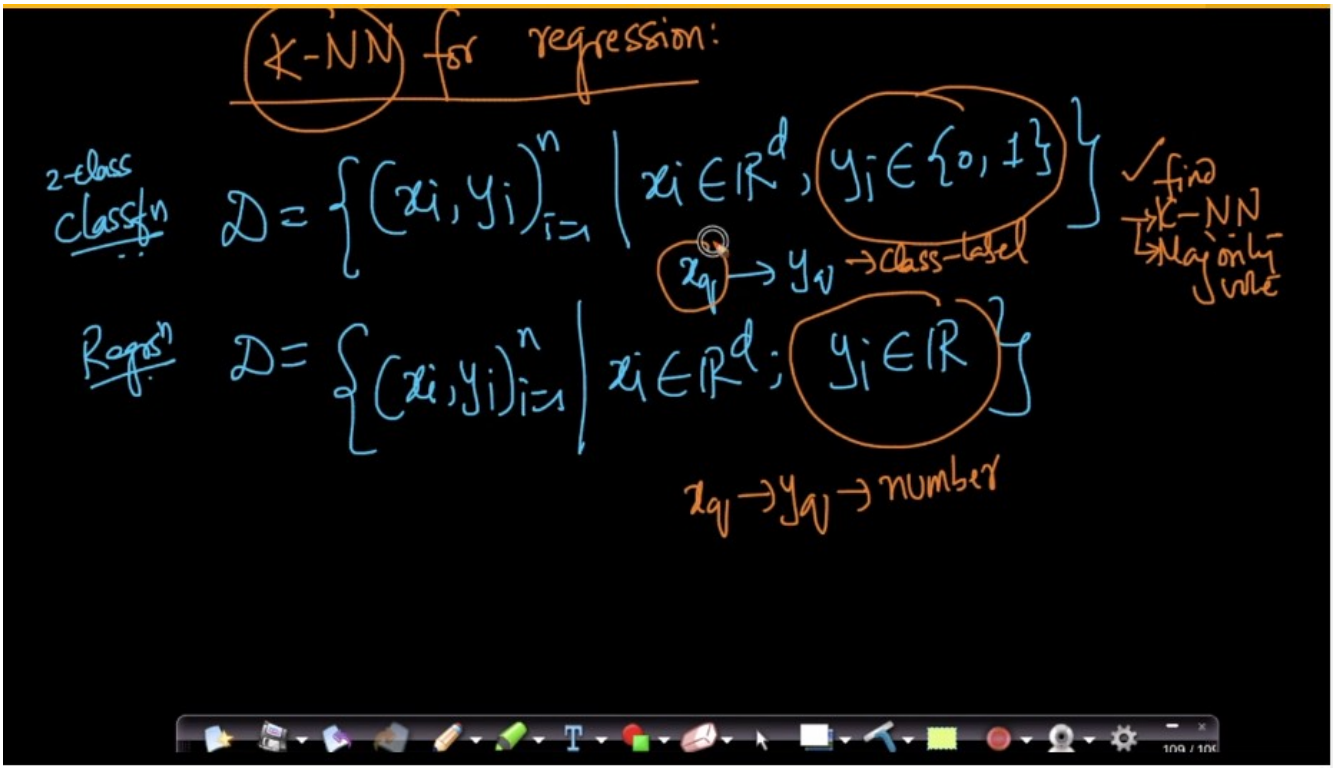
Time – based splitting



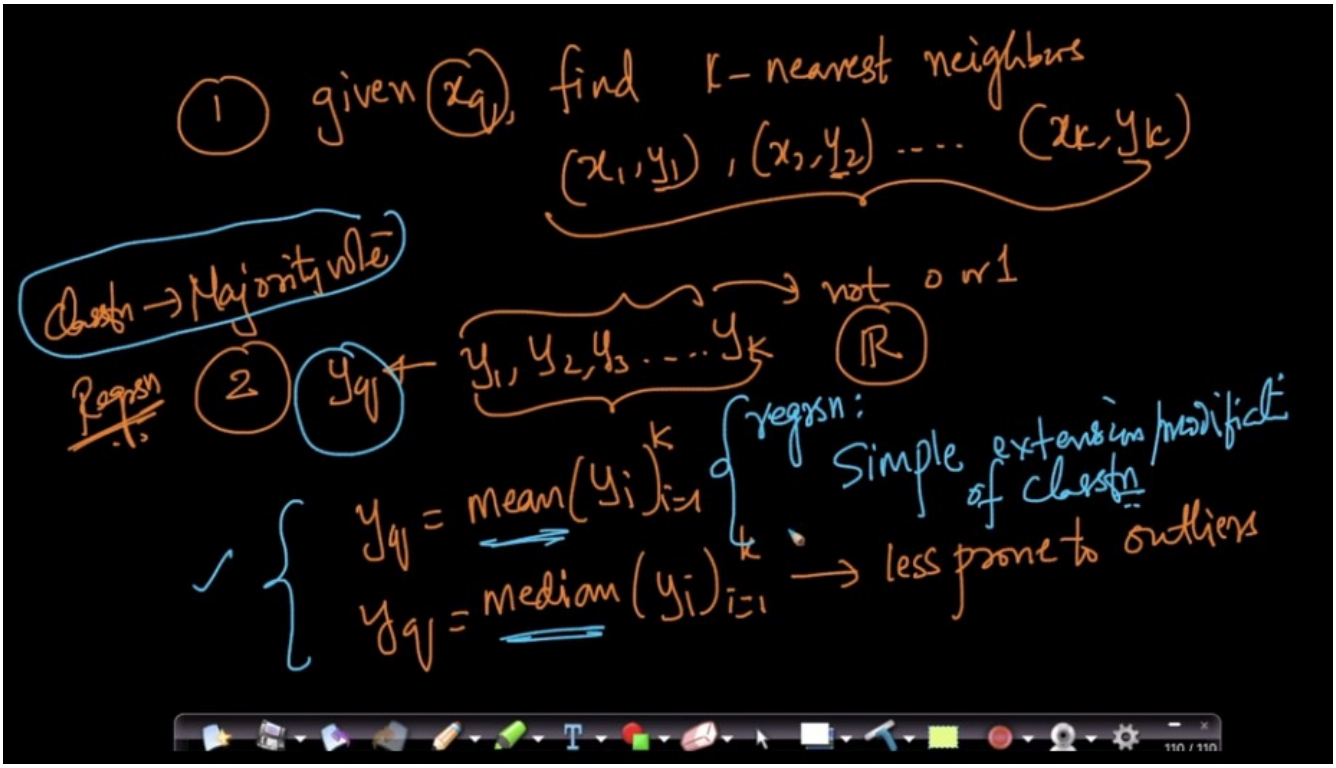
Here the test data comes from the sequential data.



KNN for regression:

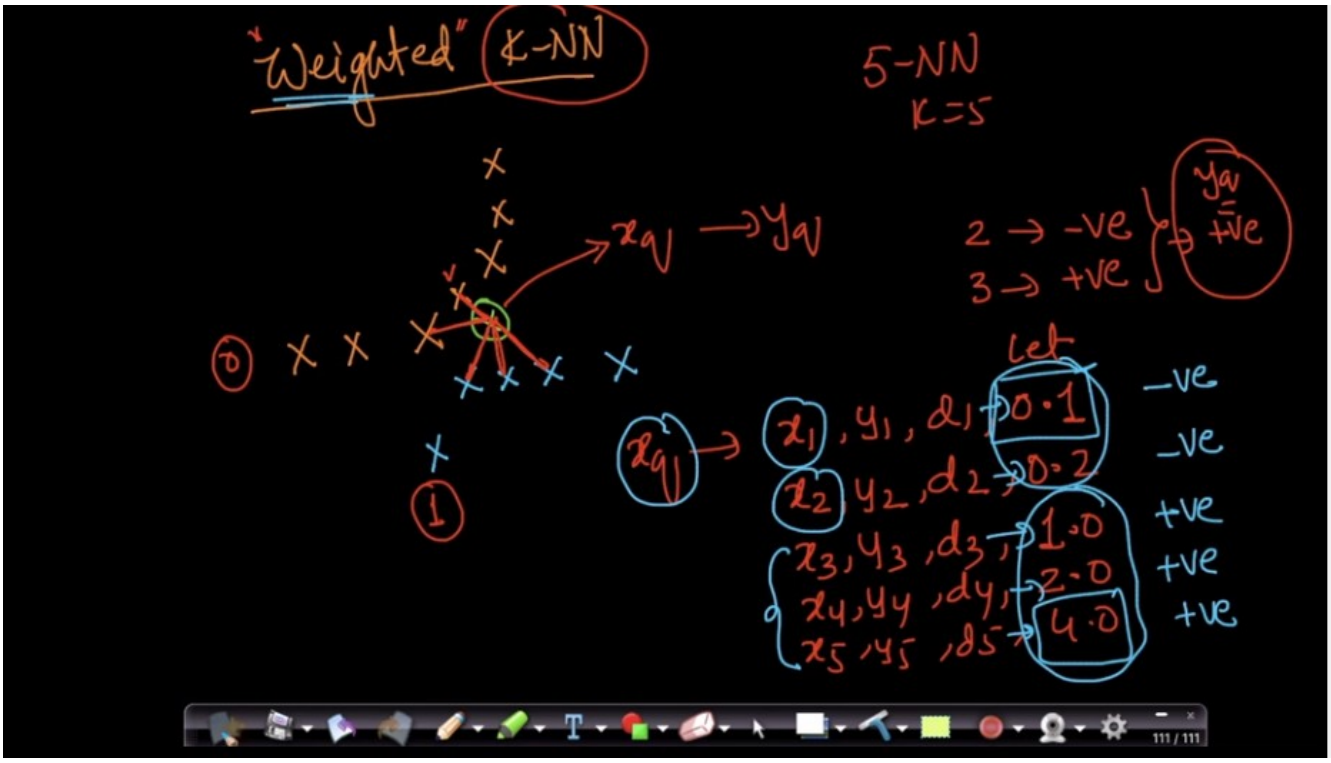


KNN can be used as regression by taking the mean/median of the nearest points of the query point.

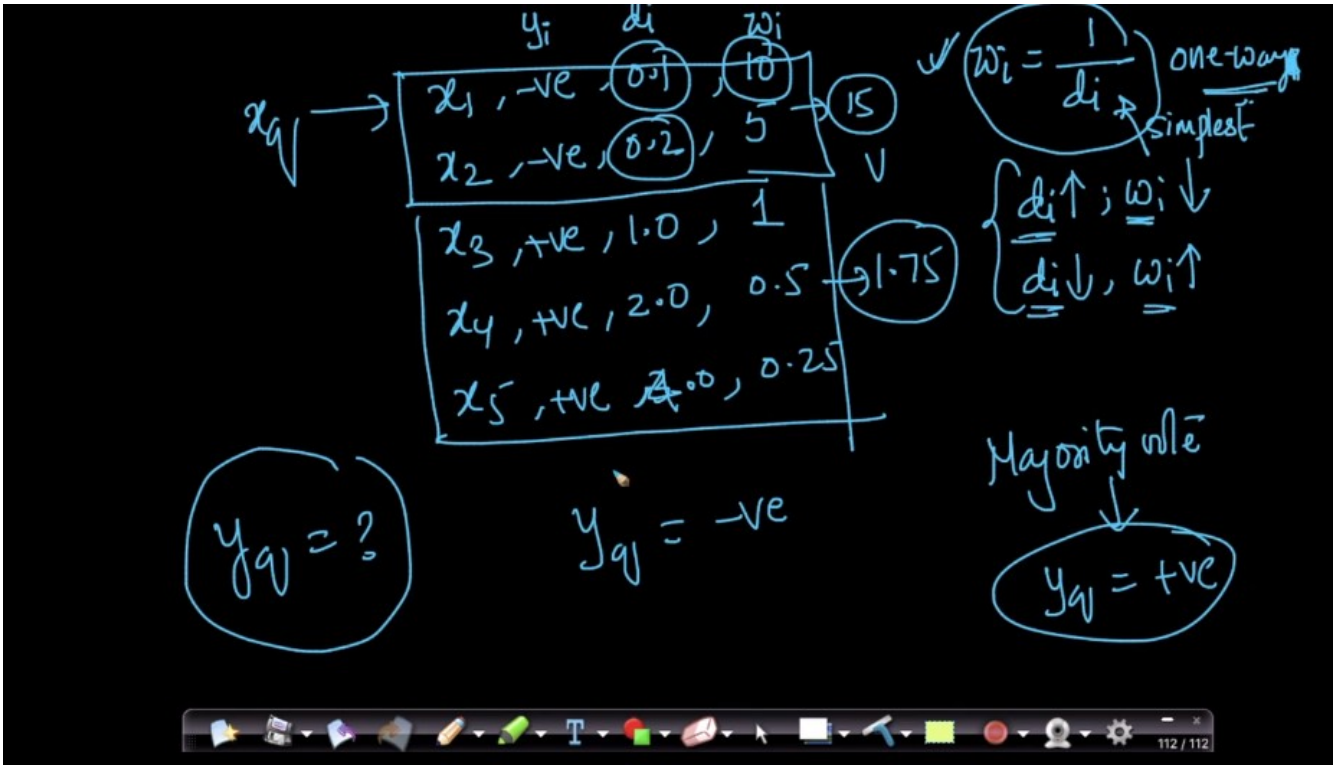


Weighted KNN:

In weighted KNN, we will give the weight to each of the nearest neighbor.

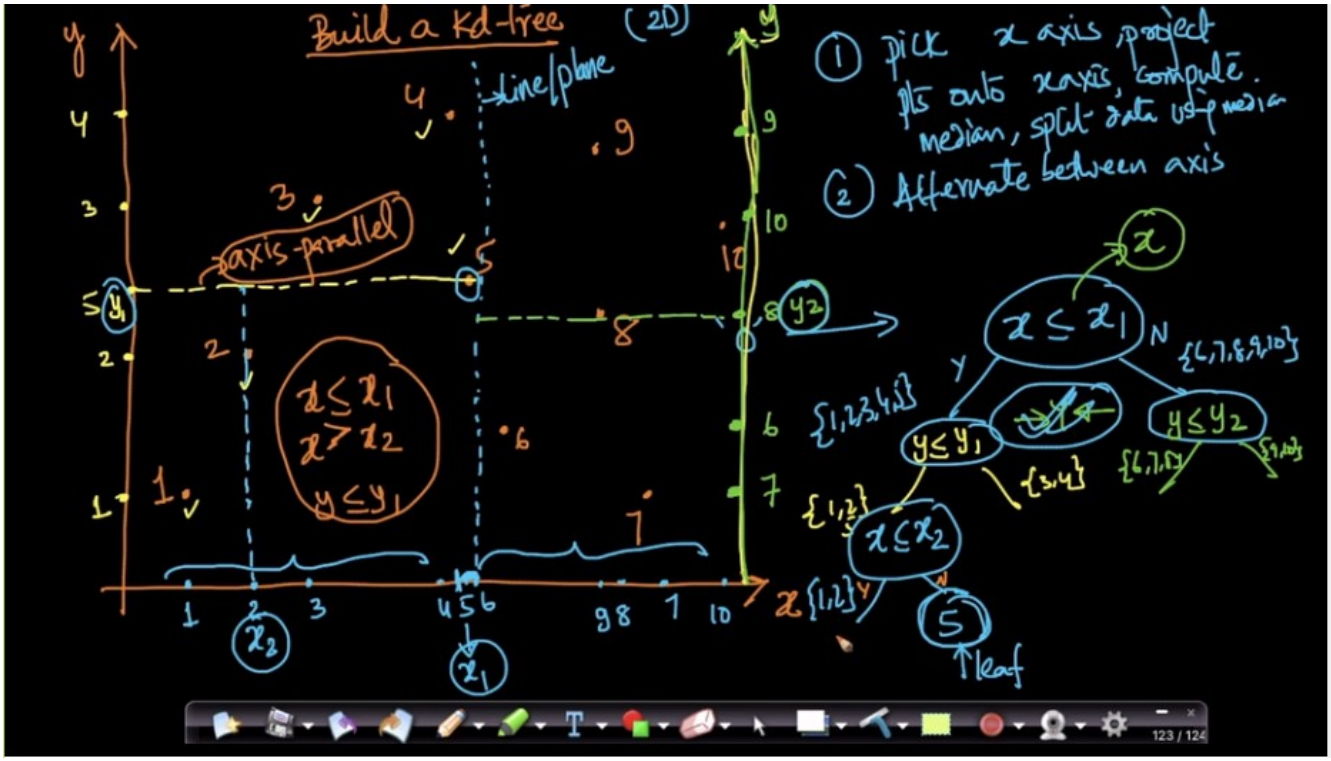


The majority weight is taken into consideration for making the classification. The highest weight class is considered as the conclusion.

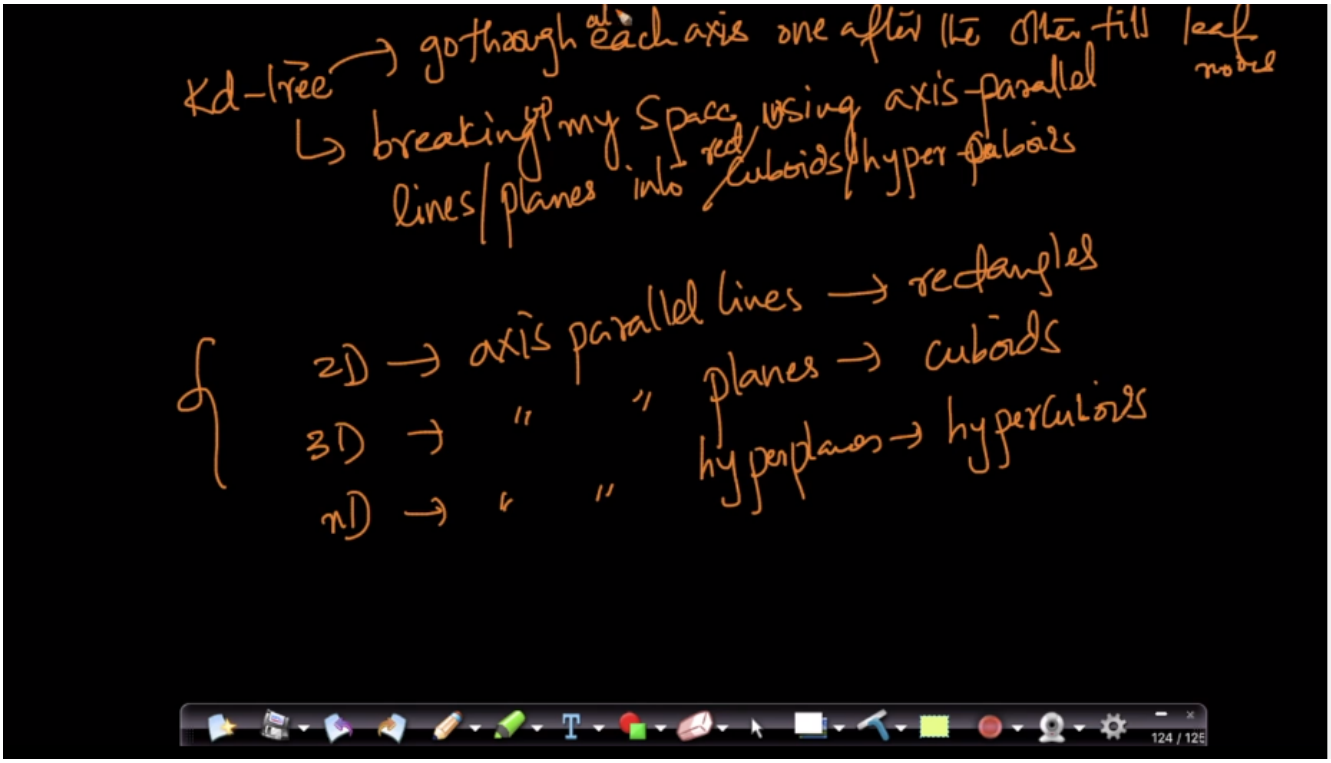


How to build a kd – tree:

Divide the data into x and y axis at the median. Alternating x and y axes simultaneously.

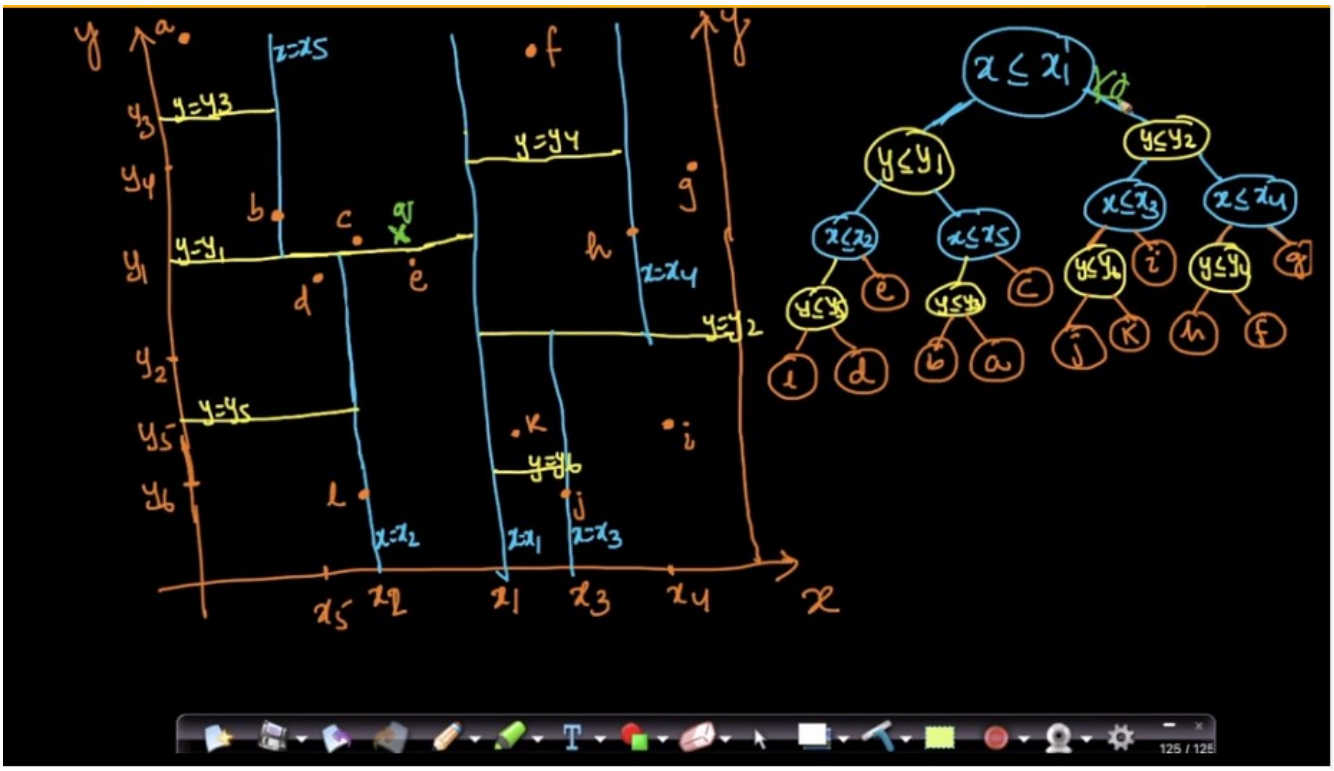


Using the kd – Tree method the 2D data is broken into simpler spaces by axis – parallel lines/ planes.



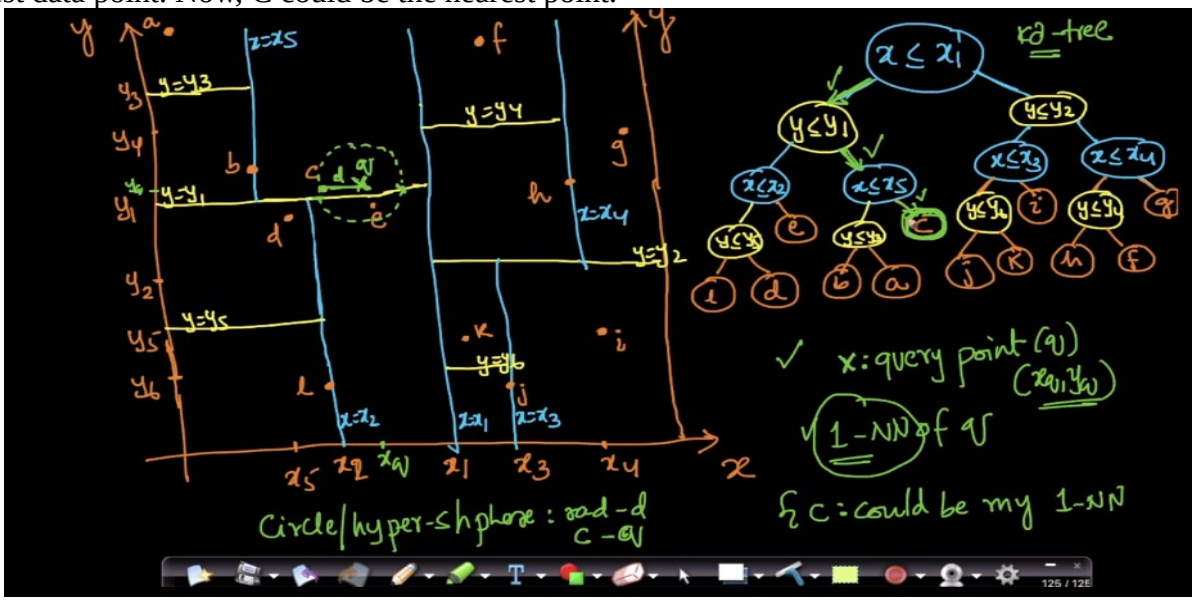
Finding nearest neighbors using kd-Tree.

Kd – Tree implemented graph.

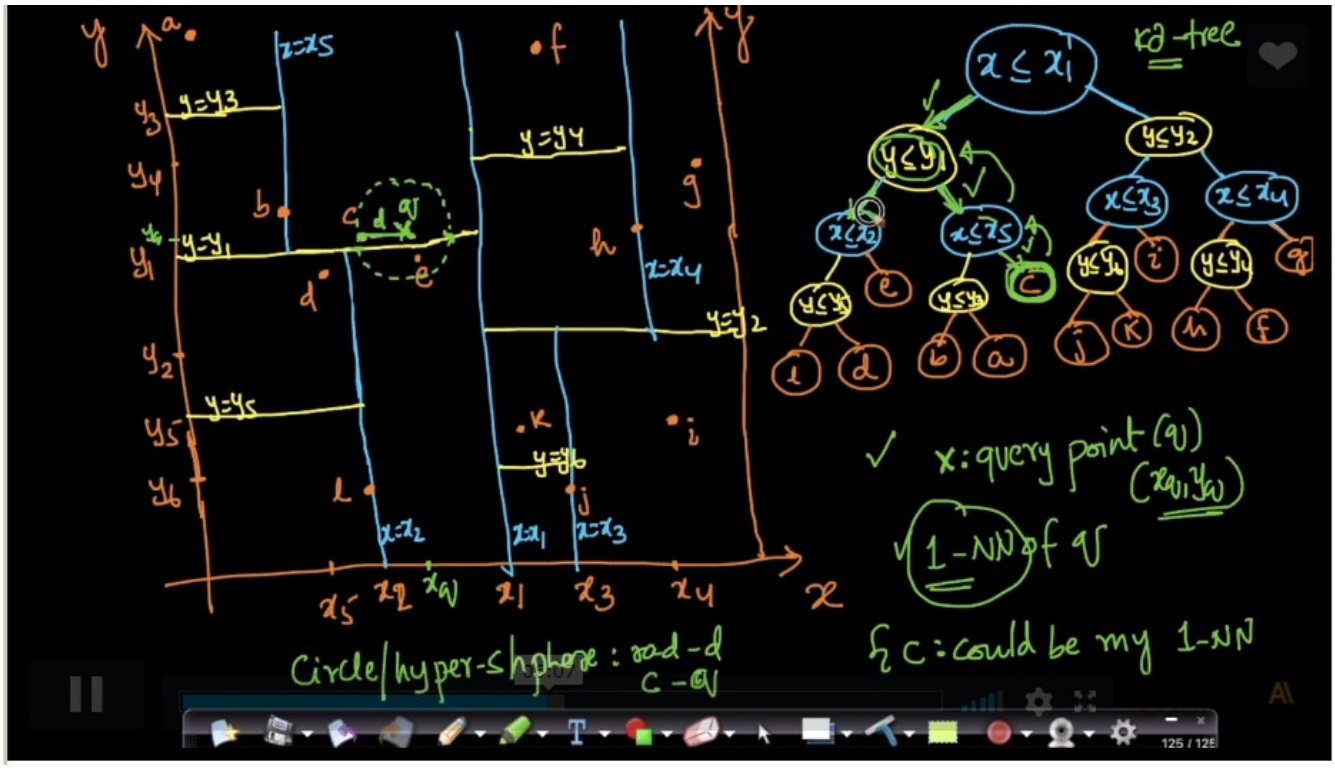


Given xq and yq as the query point, traverse the kd tree.

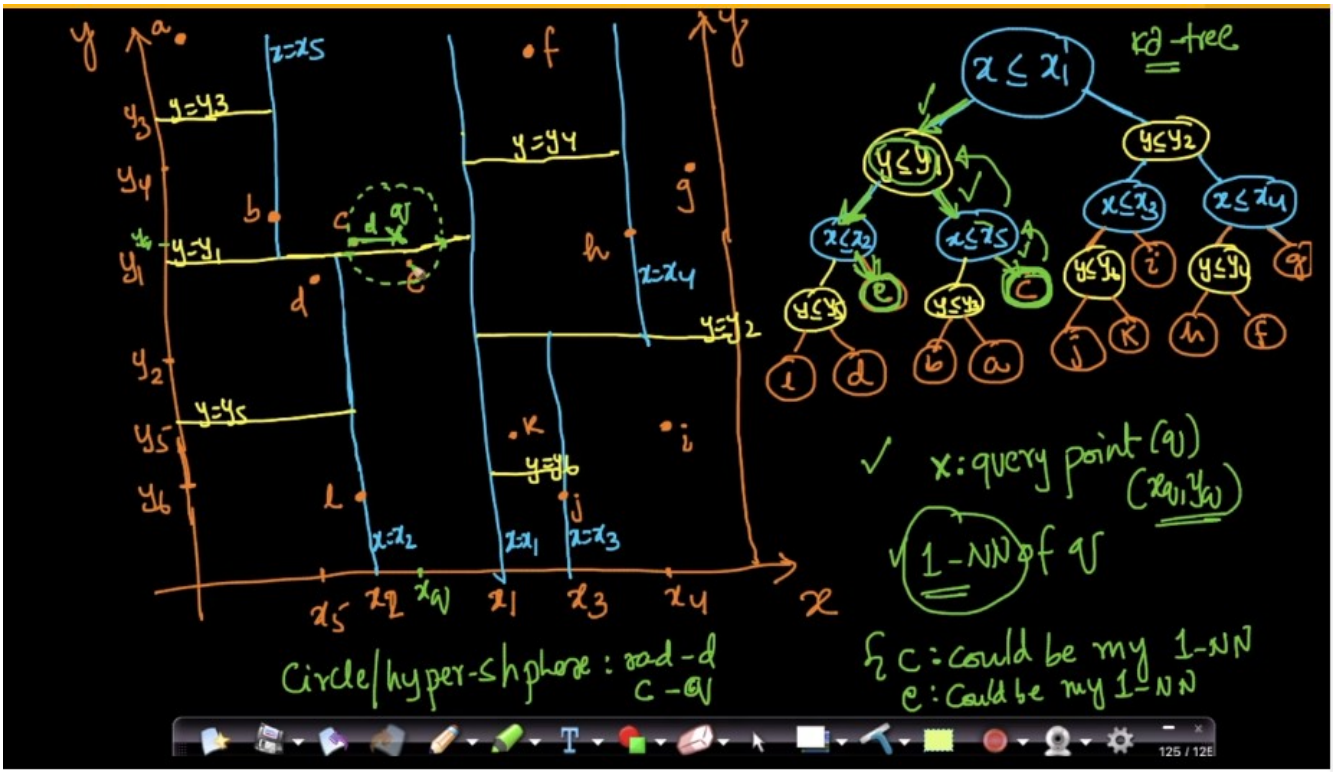
By traversing the tree with the query point coordinates, we reach the block containing the data point. We draw the circle with the radius and if there is a data point in the radius that point is treated as the nearest data point. Now, C could be the nearest point.



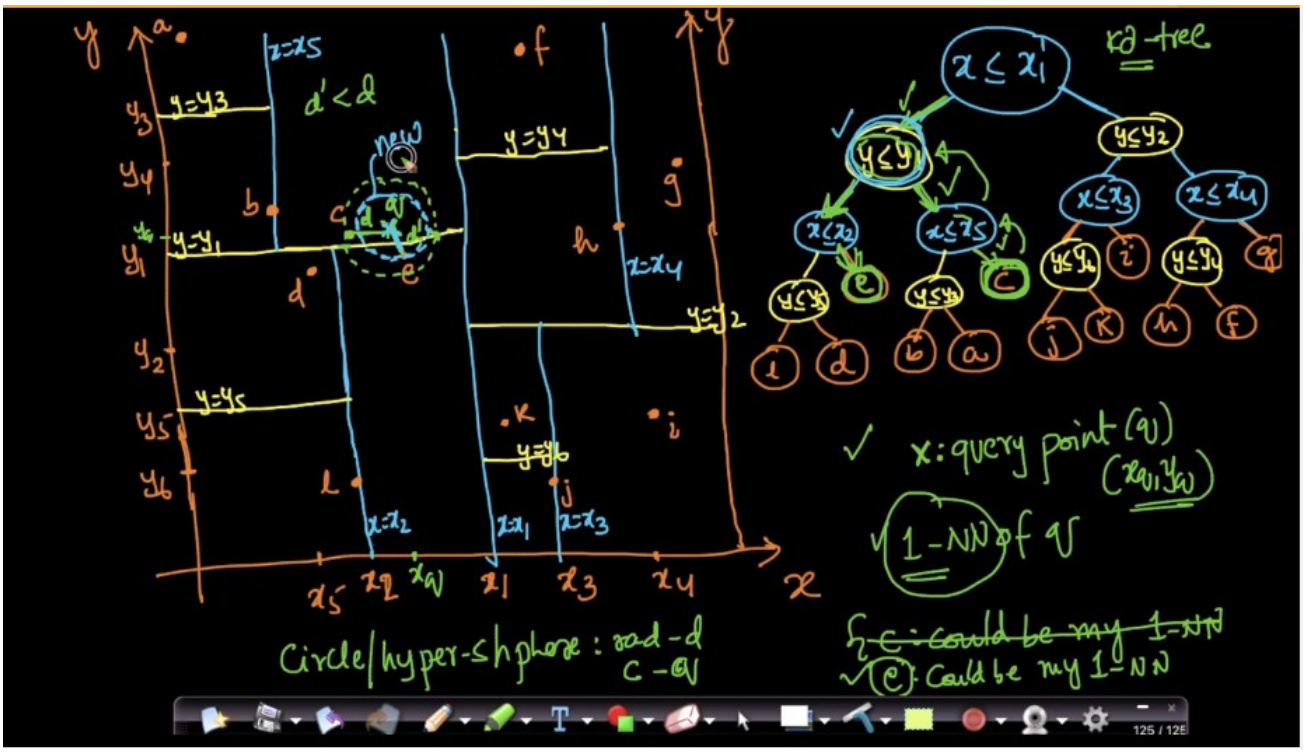
The circle intersects the line y = y1 the point is backtracked to the node where y<=y1.



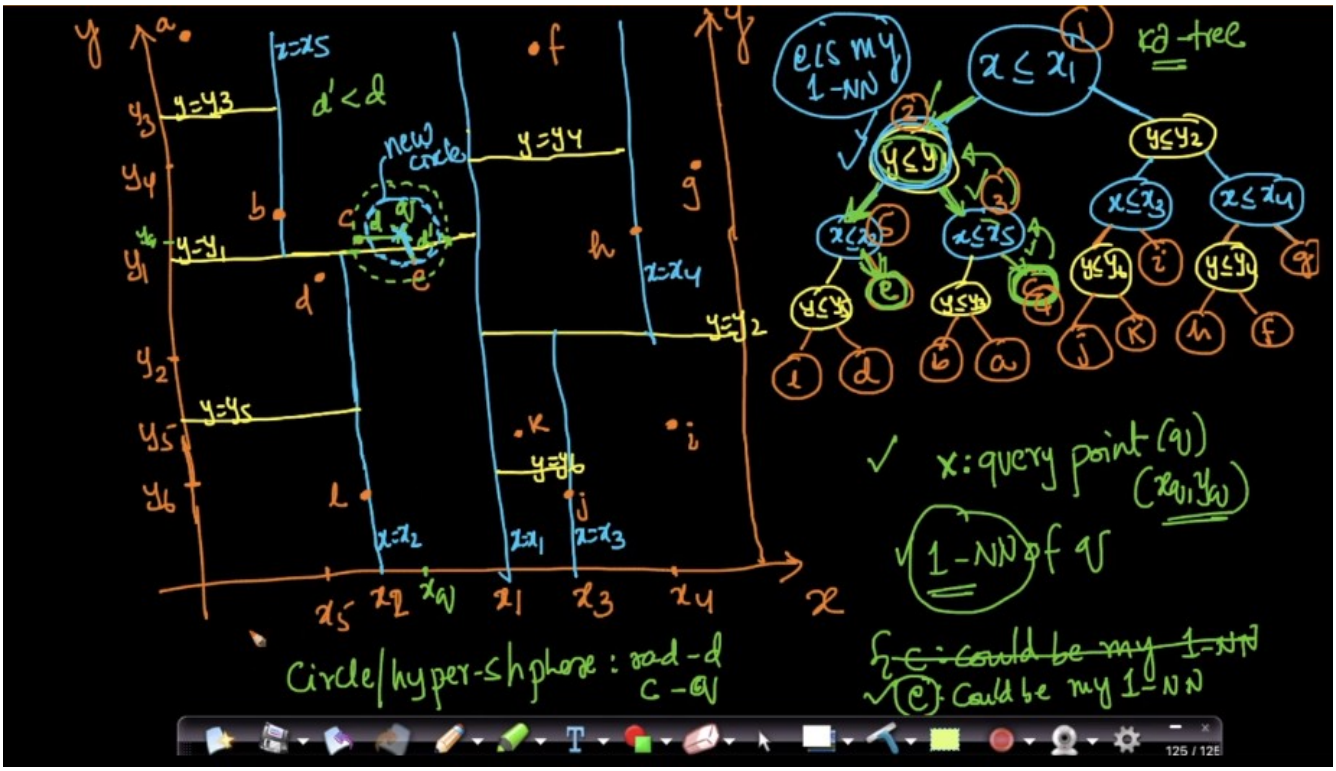
Now xq is checked with the node condition on other side of back tracked node junction.



Now point e can also be the nearest point.

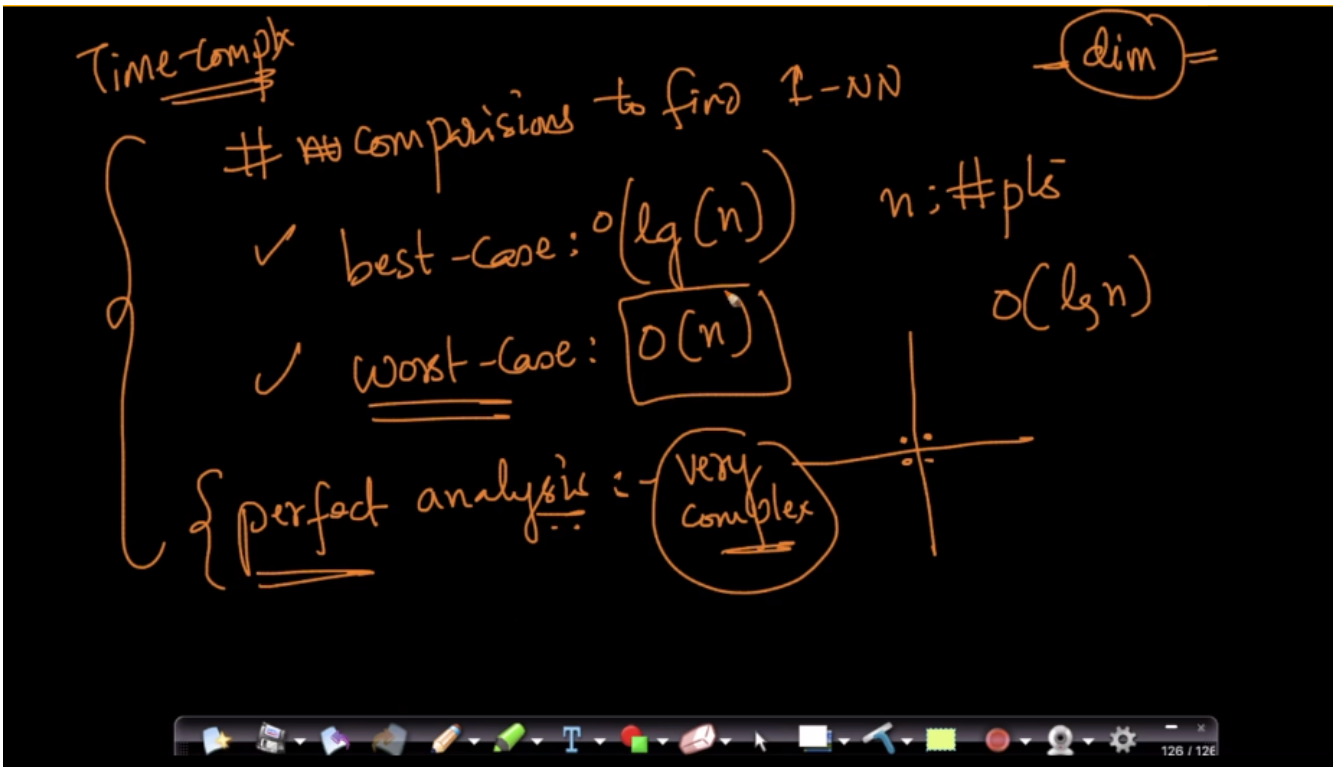


Now there is no other line intersecting the line that is already been intersected.



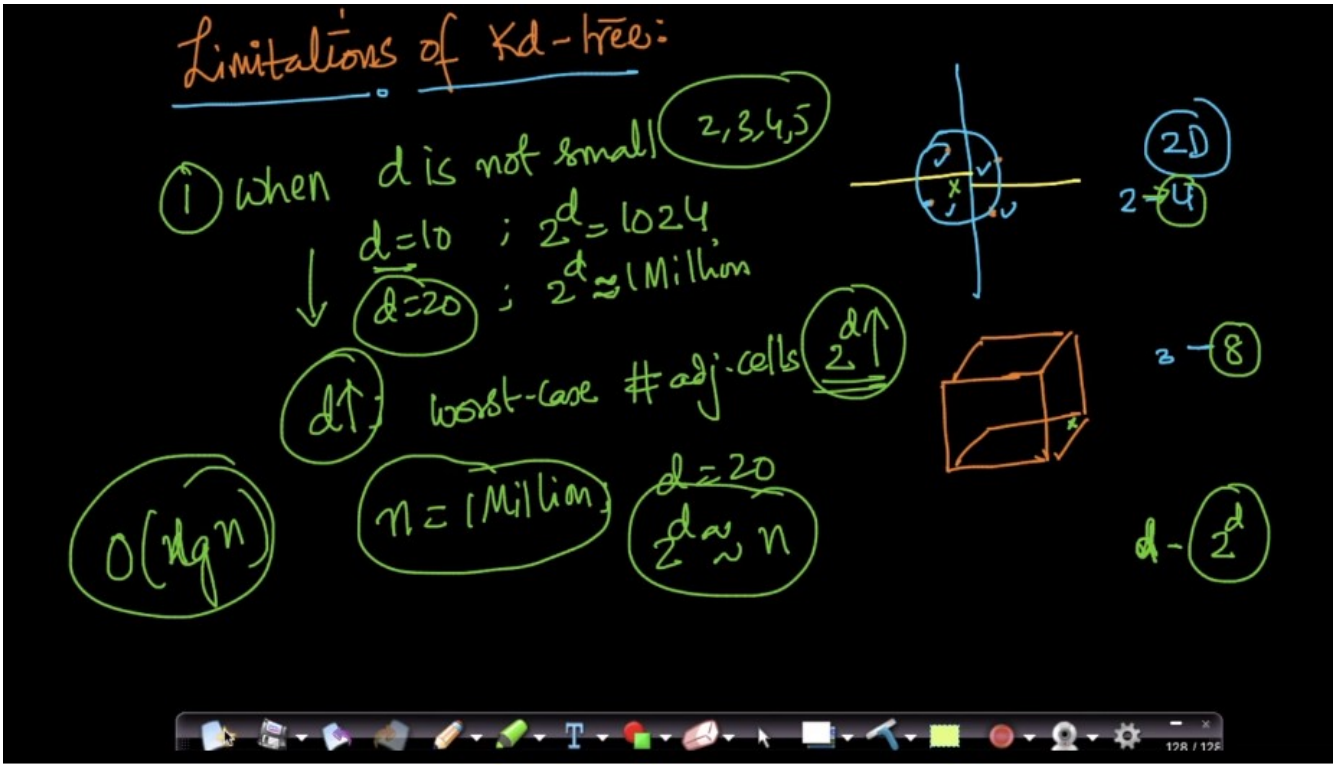
Here are the number of comparisons that are made.

Complexity:



Limitations of Kd tree:

As the dimensions increases the number of comparisons that are made will increase.



As the dimensions increase kd-tree is useless.

Kd – tree does not used machine learning extensively but it was invented for computer graphics.

Extensions:

- There are several extensions to KD-tree like ball-tree etc..

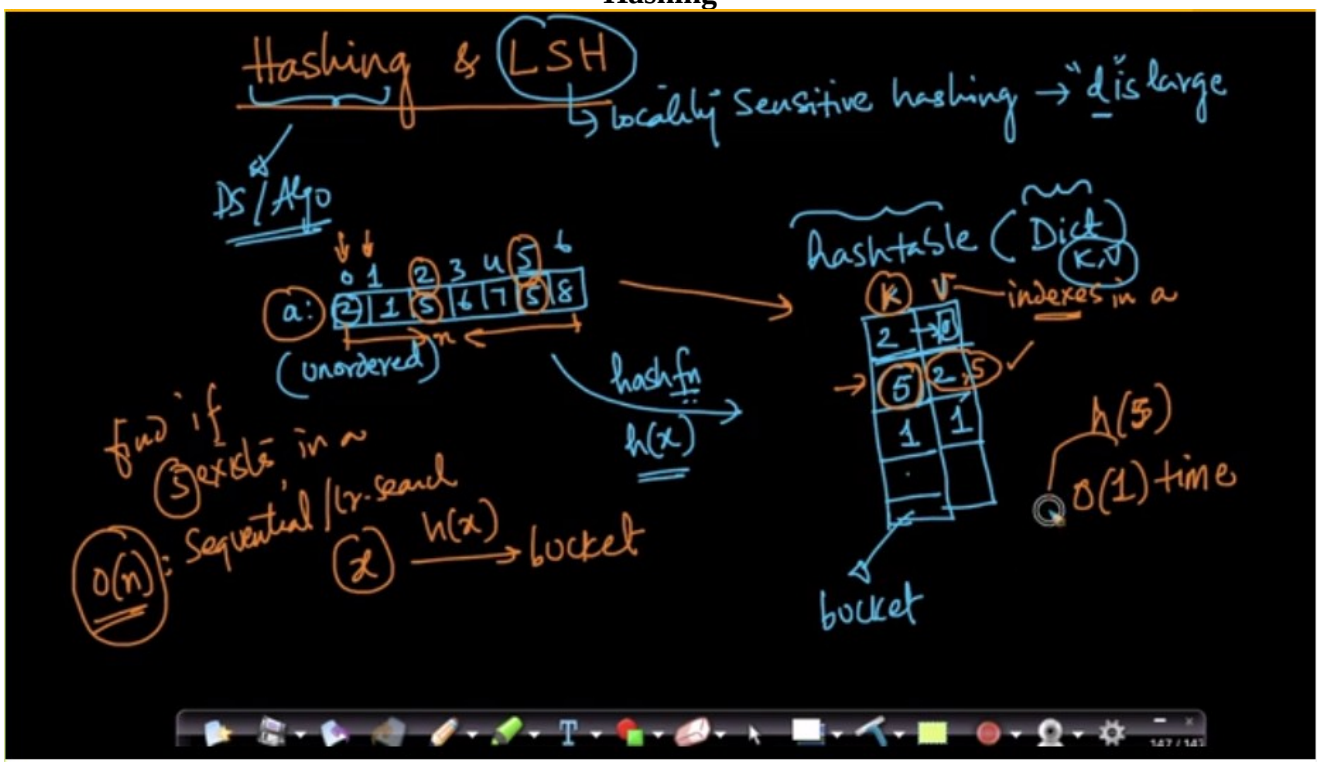
Hashing vs. LSH

- LSH is the powerful technique, it is called **Locality Sensitive Hashing.**

- It works for even for “d” is large.
- A hash table in python is called as dictionary.

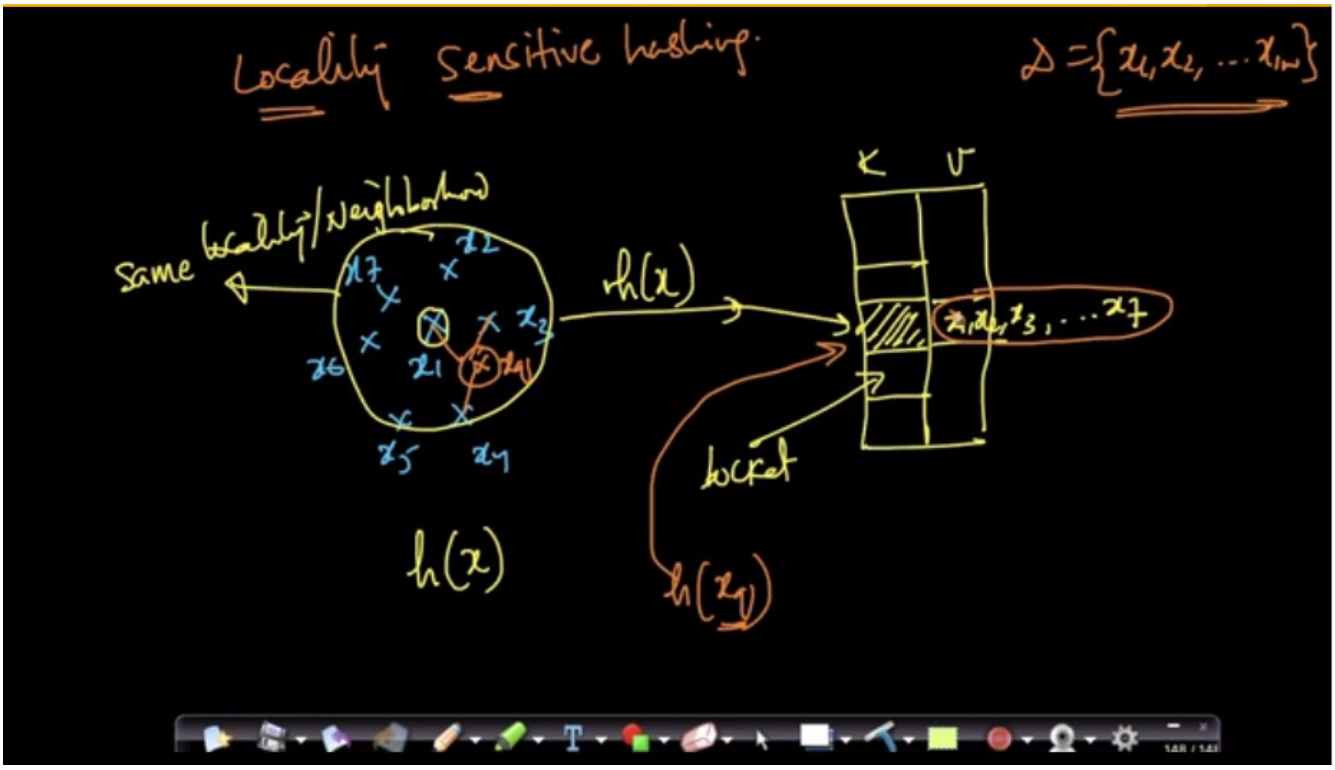
For every value a hash function is applied and stored in the form of buckets with (key, value) pairs.
If the hash function for two numbers are same then the value is stored in the form of linked list for the value returned by the **hash function.**

Hashing



Locality sensitive hashing computes the hash function for the neighbors and stores in the same bucket. By doing this we can obtain all the values using the hash function.

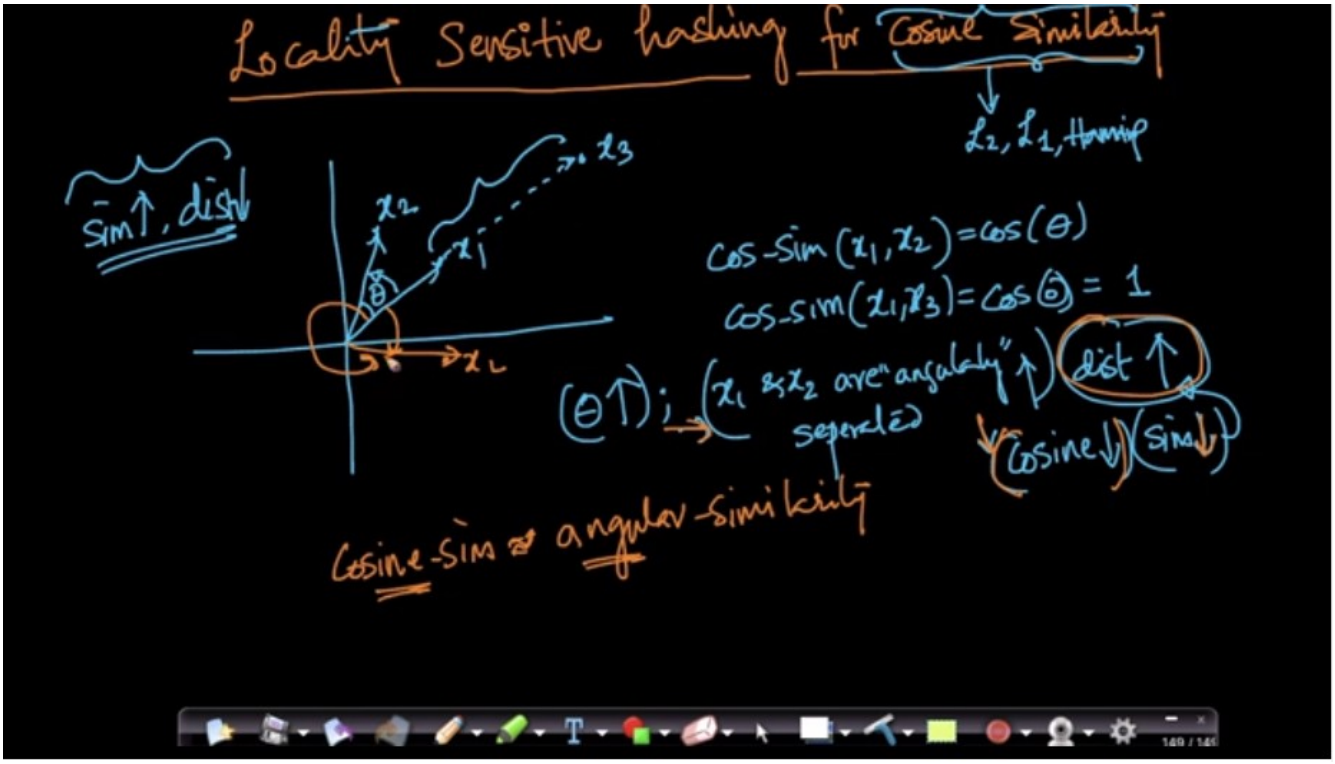
It works brilliantly well for the large dimensions.



The nearest points of the **query** point is searched in very less amount of space.

LSH for cosine similarity

- Cosine similarity for two points is the cos(angle between the points).
- As **theta** increases x1 and x2 are more angularly separated.

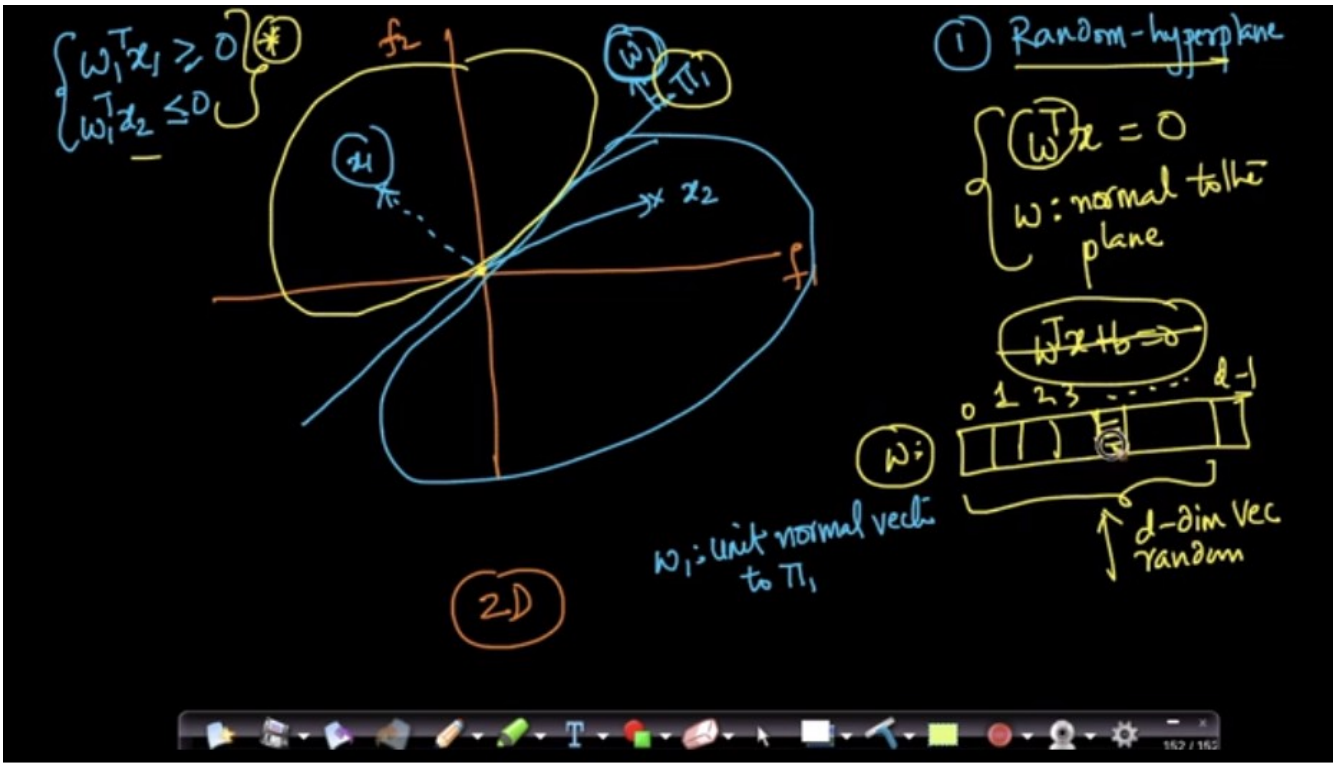


Working:

- The points of less angular seperation are stored in the same hash function value.
- LSH is a randomized algorithm.

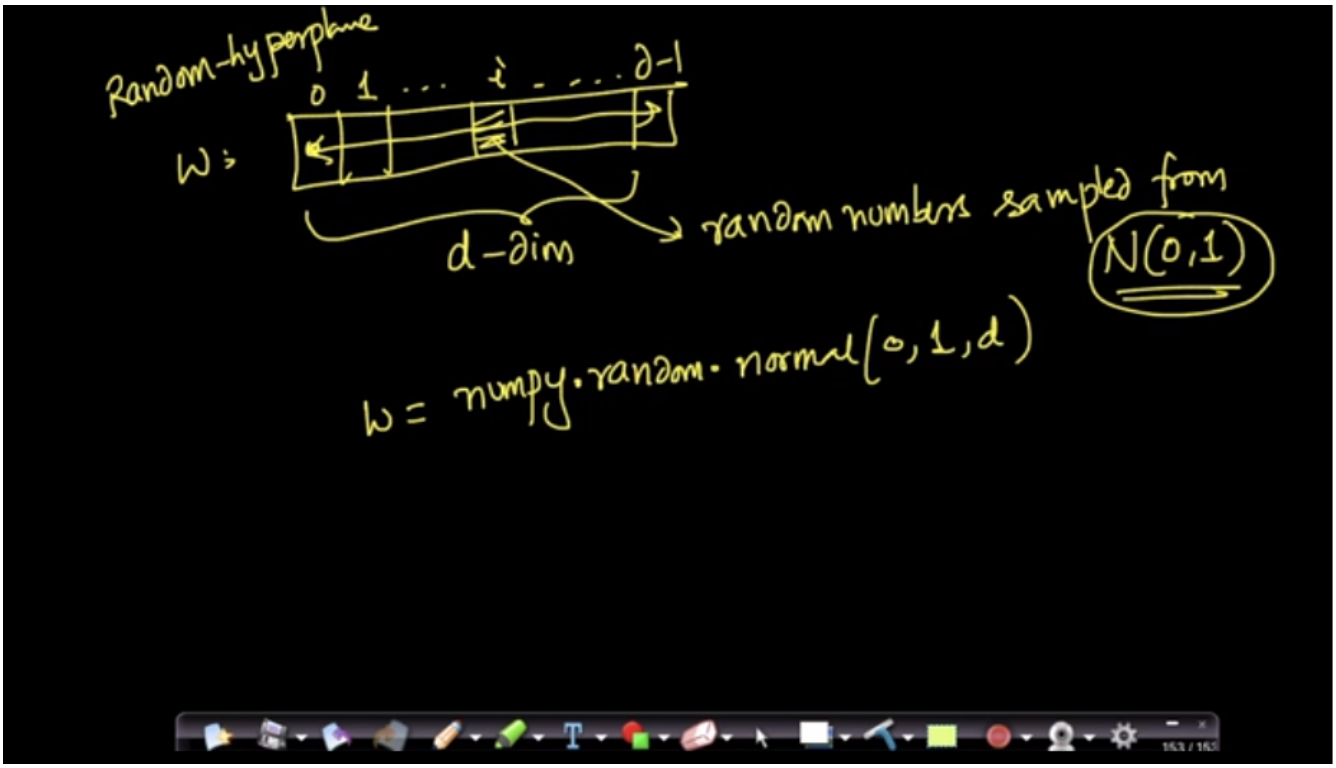
LSH divides the feature space with the hyper planes.

The hyper planes are generated randomly.



The point **x1** is along the direction of the hyper plane and **x2** is the point below the hyper plane.

The vector W is the randomly generated hyper plane of **d dimensions.**



The random hyper planes are generated using the normal distribution of zero mean and 1 standard deviation using numpy.

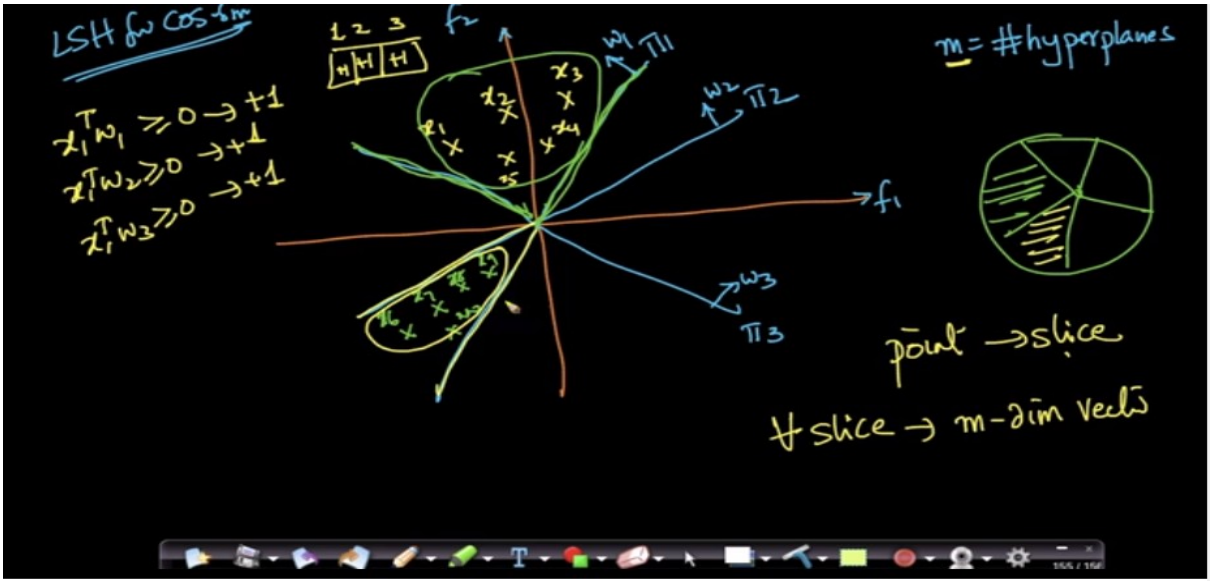
Here are the random hyper planes.

The hyper planes breaks the feature space into several parts like slices of Pizza.

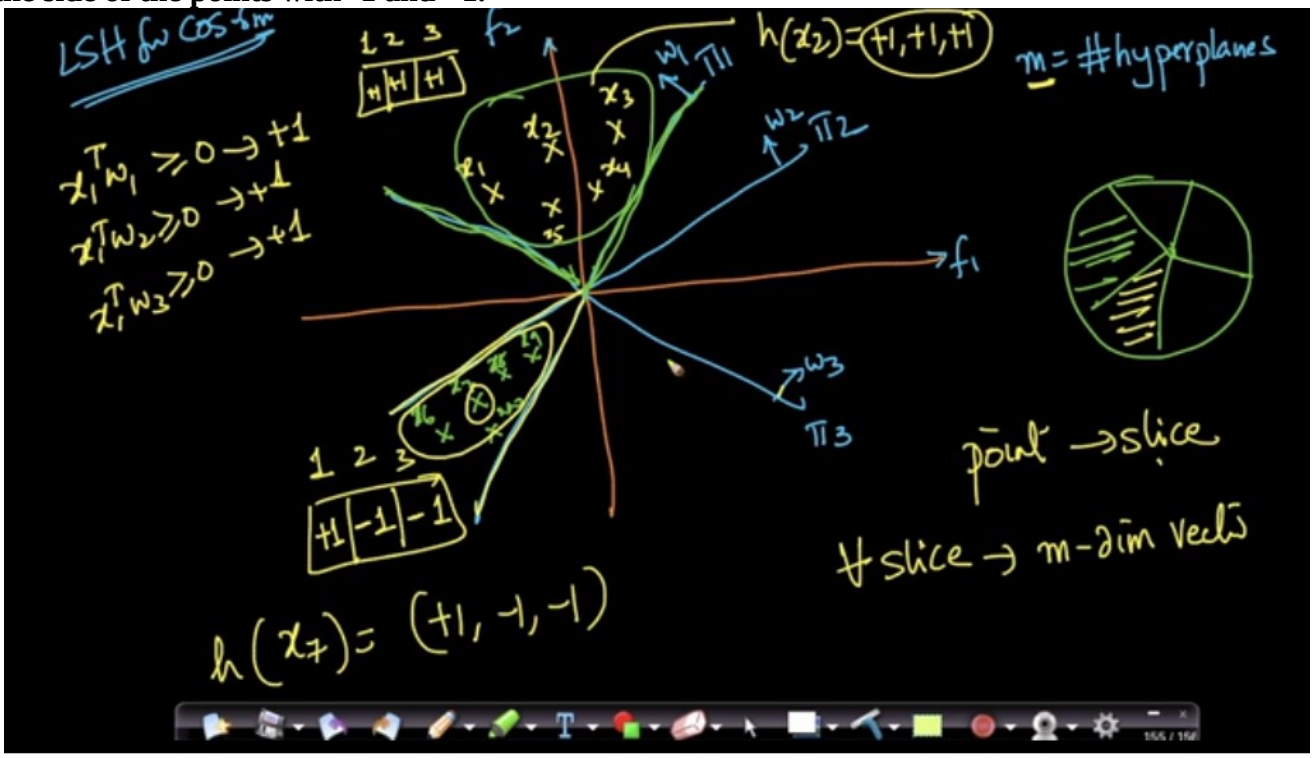
Each point is assigned to a slice.

If we multiply all the values between the slice the points will be on the same side of the normal.

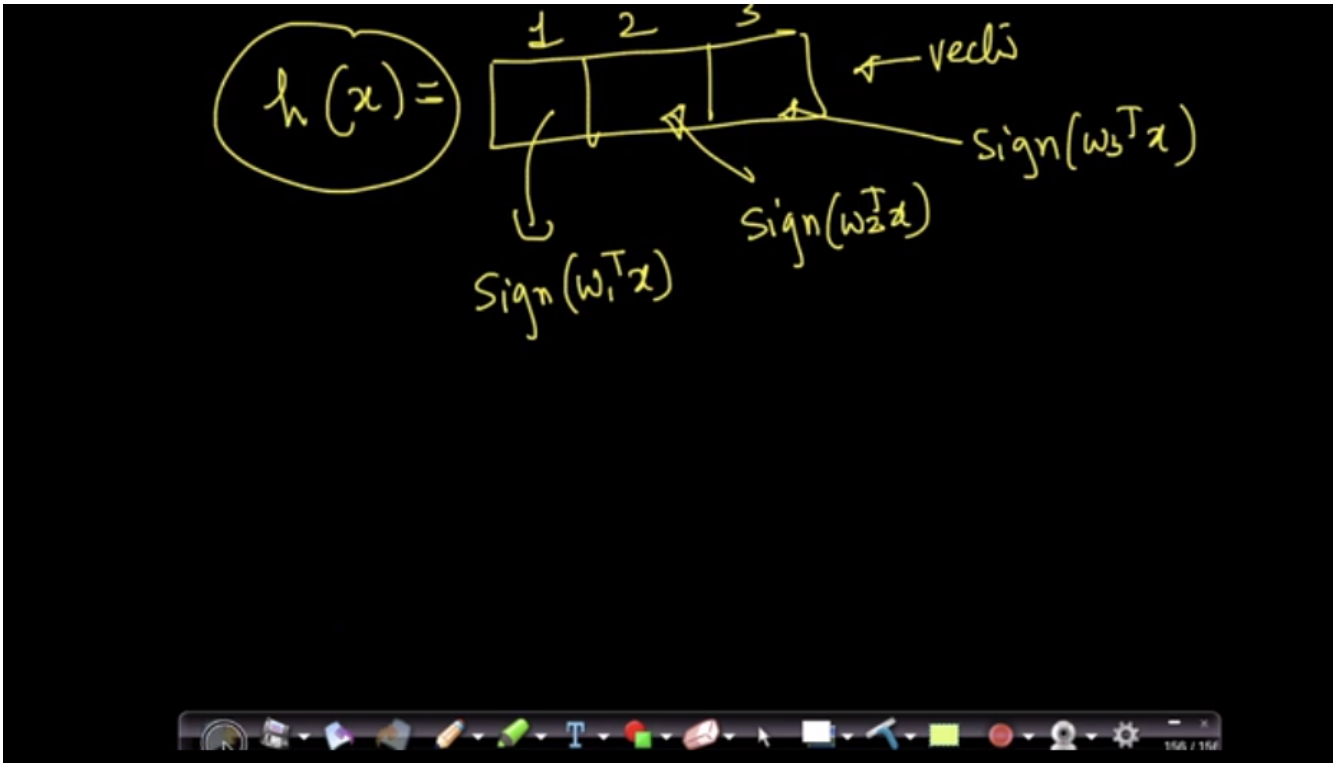
For every slice we make a m – dimensional vector. Where m is the number of hyper planes.



For each point there direction is computed, each point has the m- dimensional vector denoting the side of the points with -1 and +1.



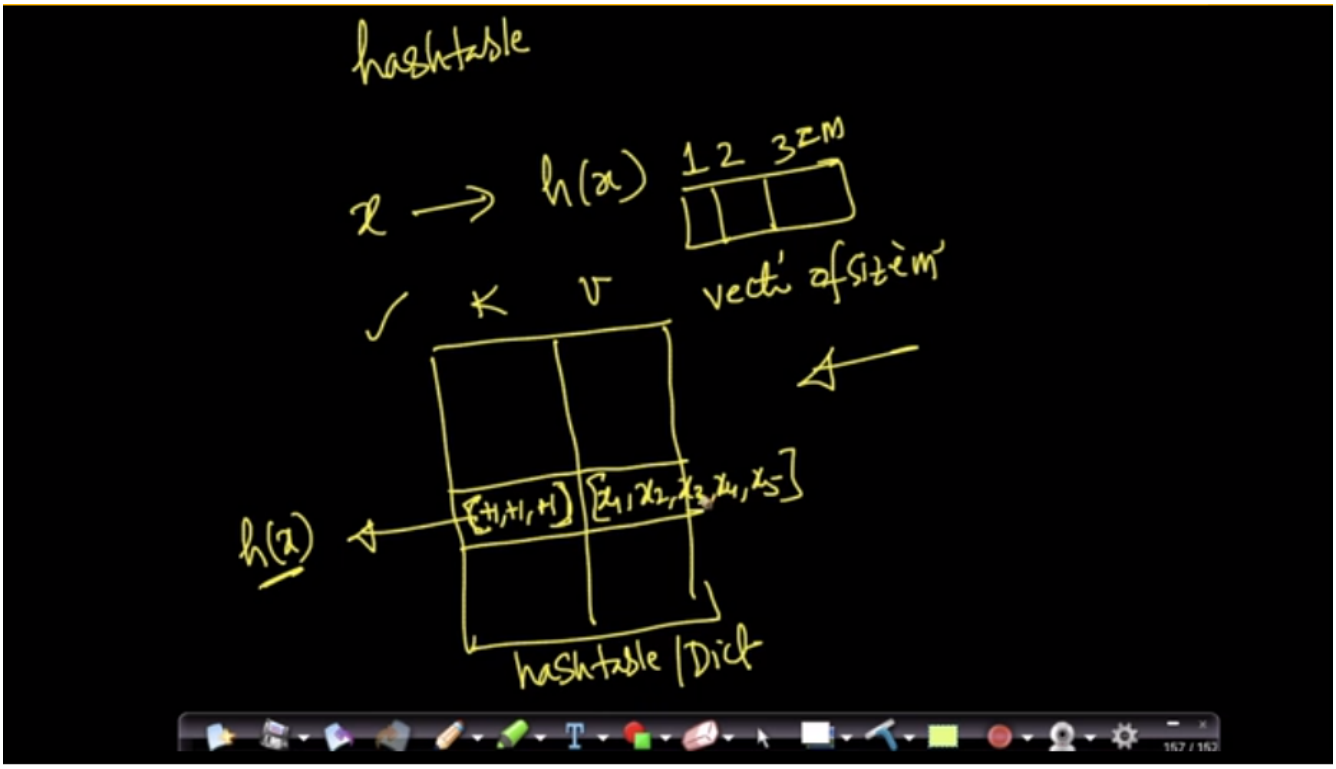
We are using m different hyper planes for calculating the direction of the point.



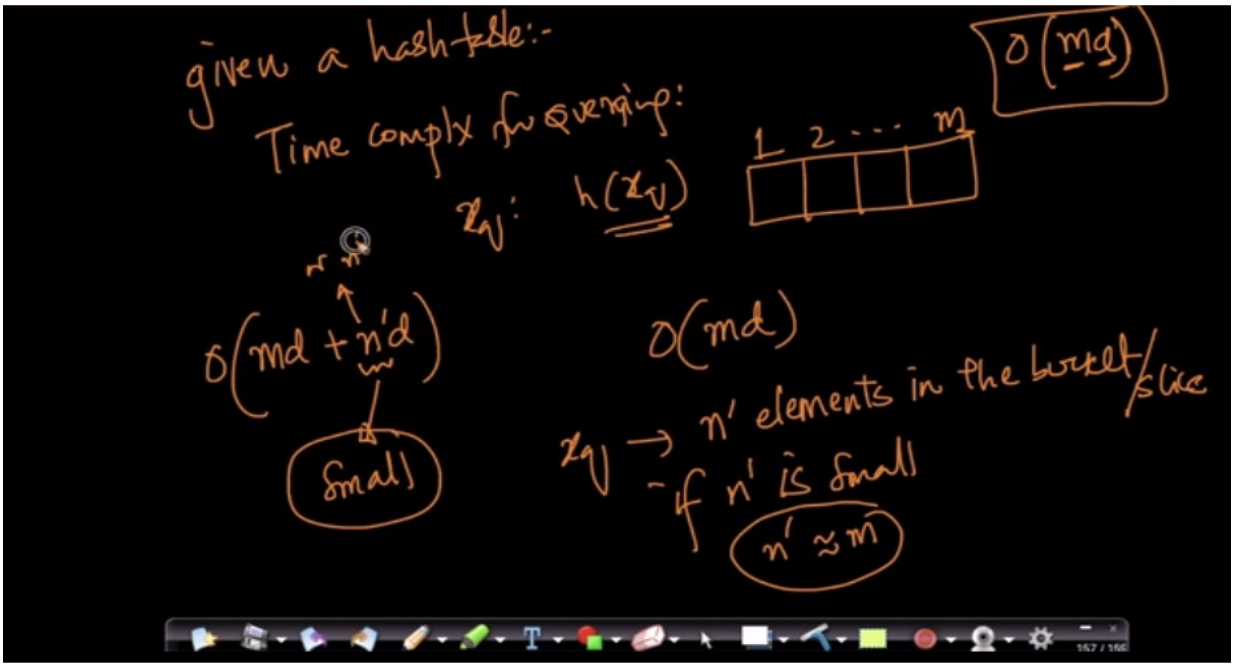
Now we will construct hash table:

The hash function is computed for every separation done by the “m” hyper planes.

The computed vector for each of the point is made as a key for hash table.



The points x1, x2, x3, x4 and x5 are taken into one bucket for the hash value.

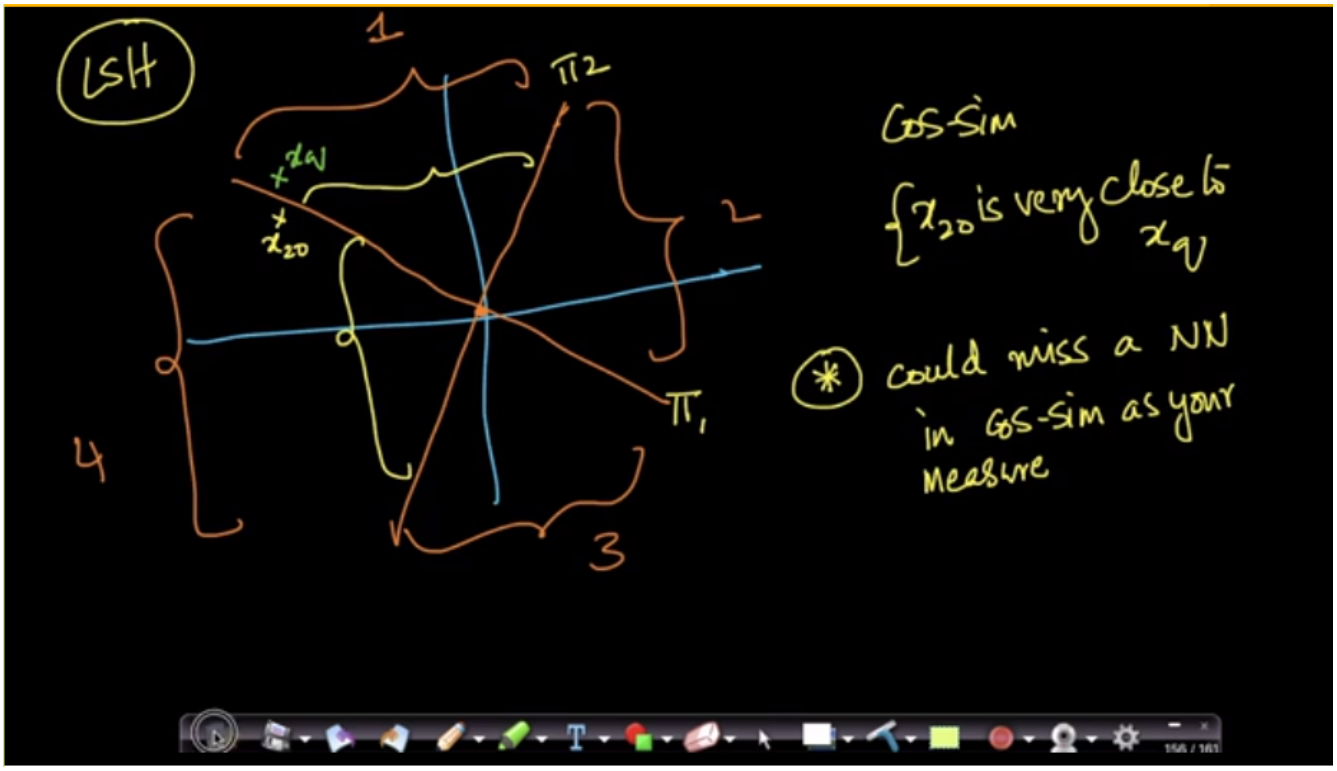


In application, the m is set to log(n).

where ‘m’ is the number of hyper planes.

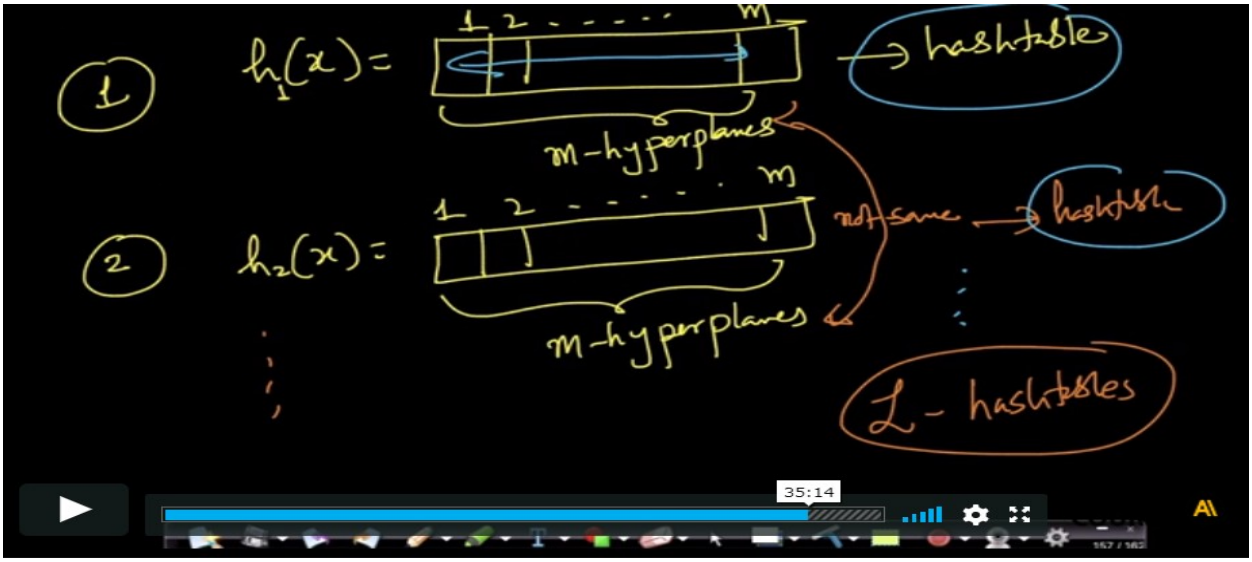
Edge case of LSH for cosine similarity.

One can miss a nearest neighbor if nearest point in the other slice.



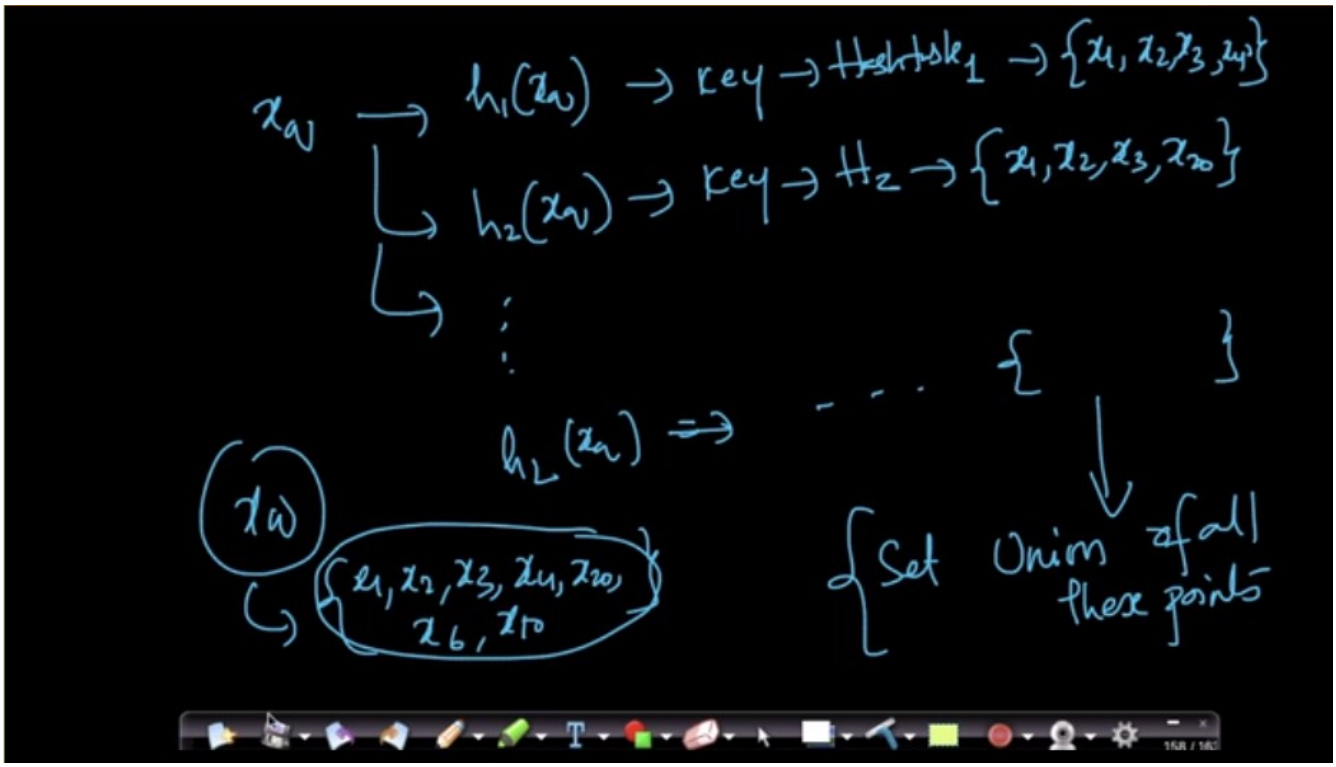
Here is the modification for the hyper planes to overcome the above problem.

We will compute several hash functions with many hyper planes with different values of planes.



Computing values for different hash tables for a single query point.

We will take the set union of all the points and we take the nearest points from the union and report the result.



The final time complexity to query given L hash tables.

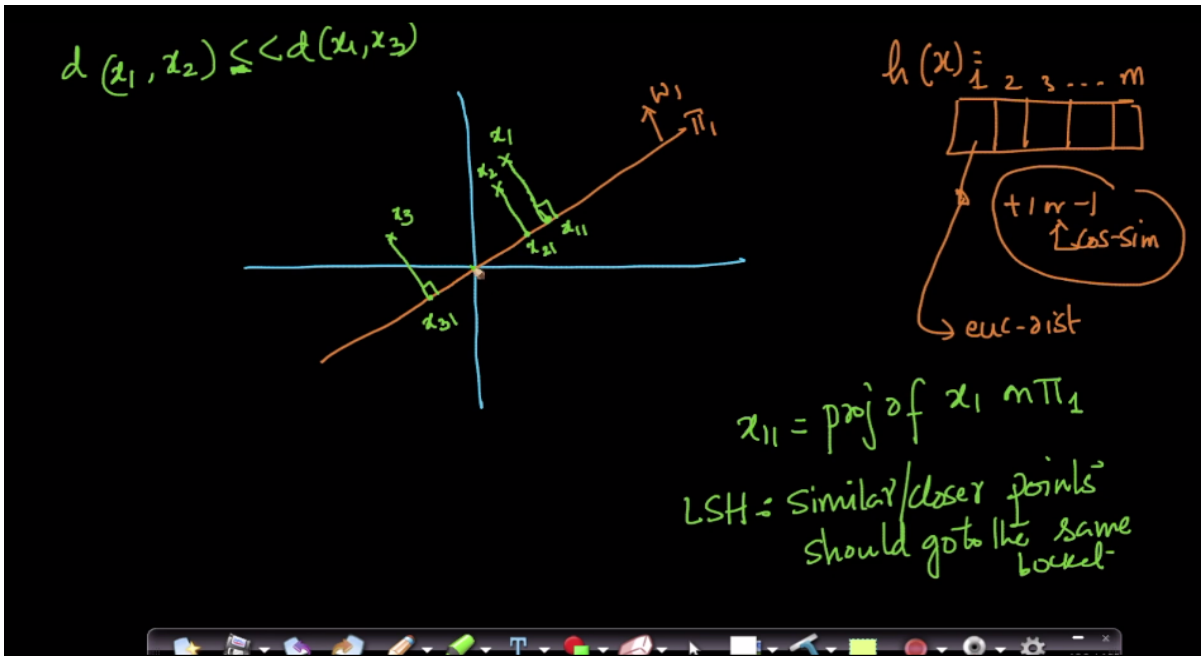
There is a chance of every point belongs to the each slice and there will be no nearest points. To overcome this situation we use **number of hyper planes(m) are roughly chosen to be O(log(n)). Where "n" is the number of points.**

LSH for euclidean distance:

LSH for euclidean distance is a simple extension to the **cosine similarity.**

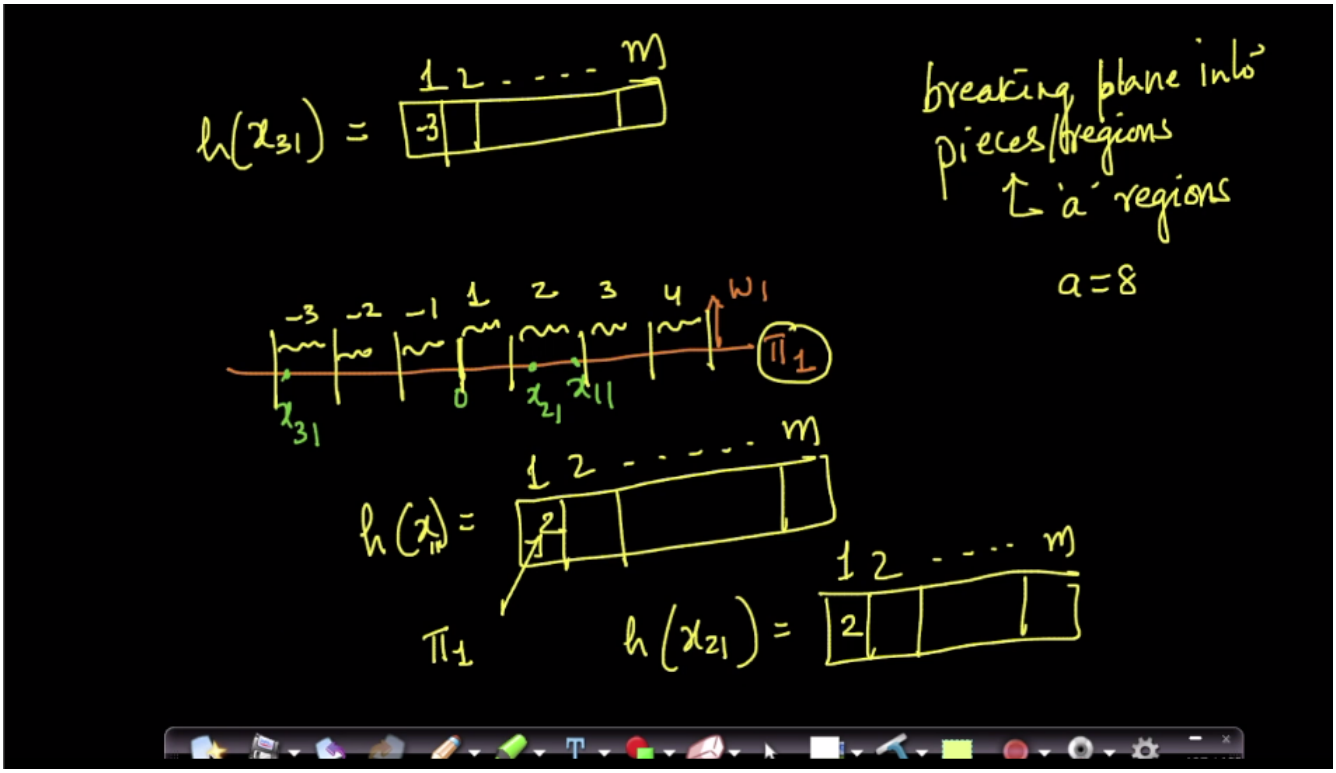
Similar or same points should lie at the near distance.

Step – 1: The points are projected onto the plane.



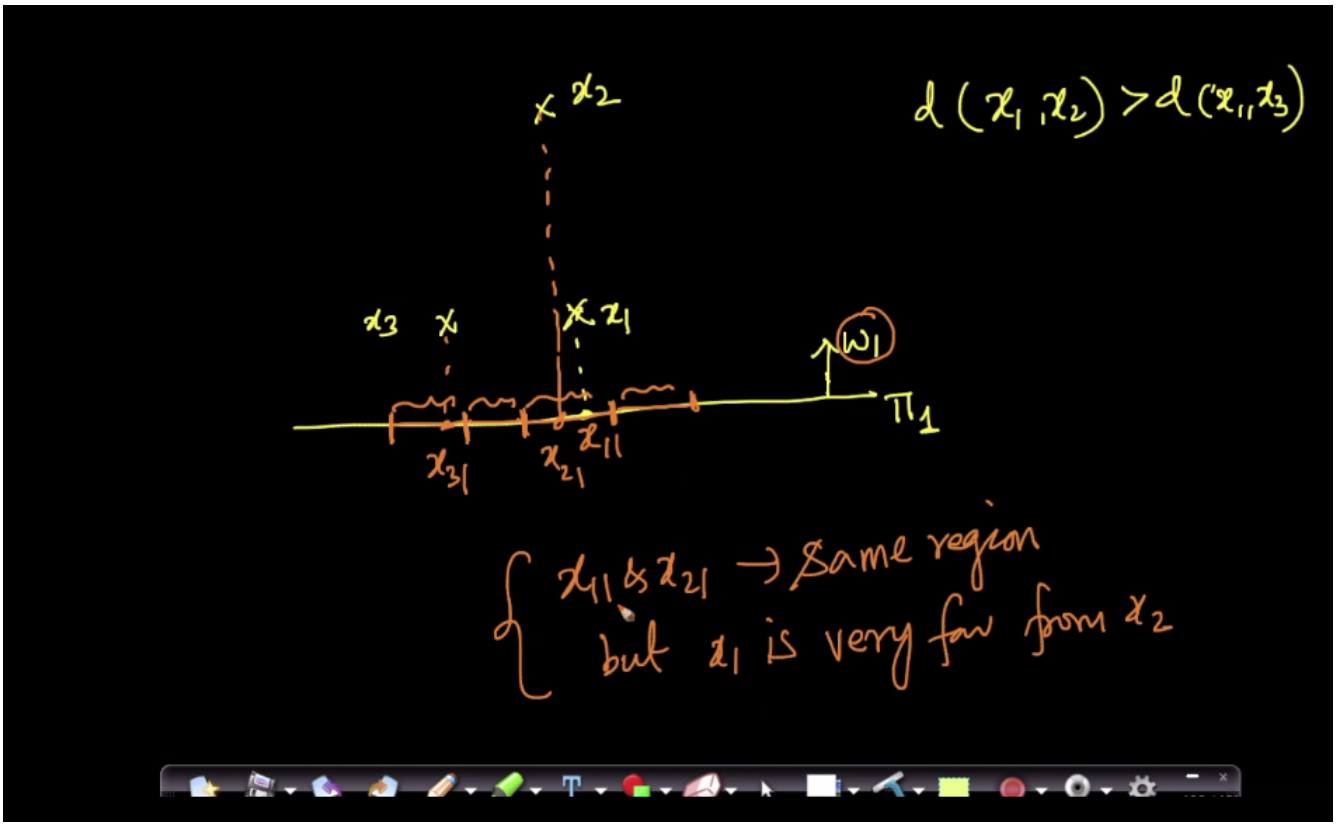
Then the plane is made into ‘a’ regions.

Step – 2: Each region is given some index value.



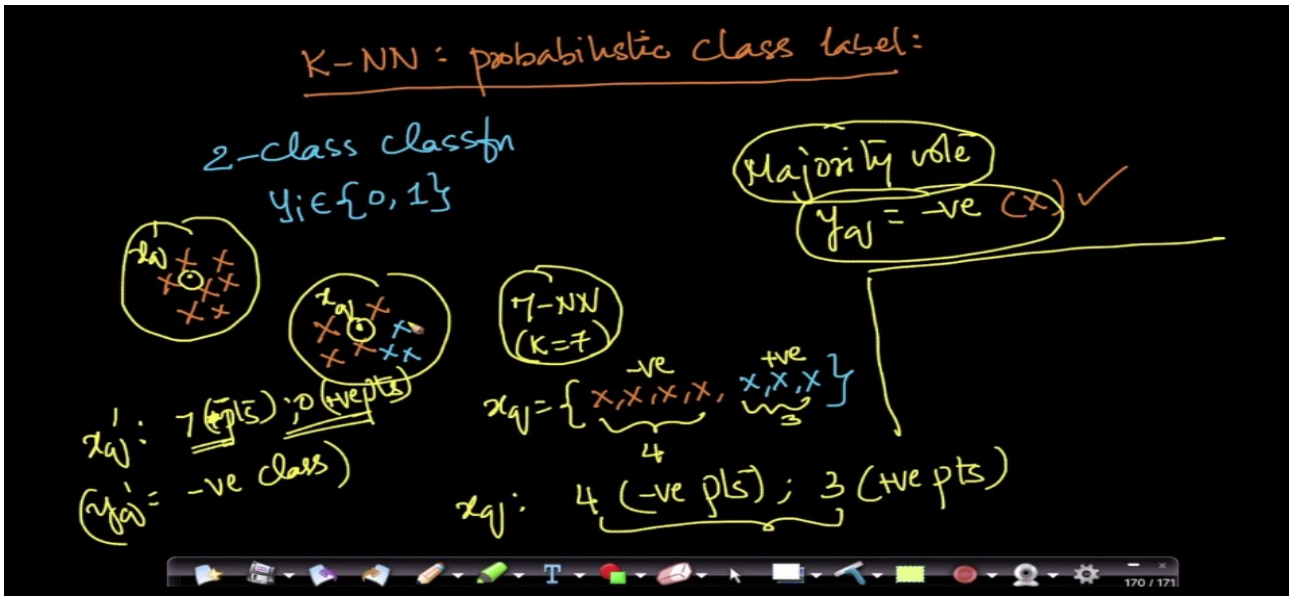
If two points have the same region there distances will be closer.

Edge case:



Even the points x1 and x2 are far from each other, they sometimes tend to fall in the same region.

LSH is not perfect.
It gives the probabilistic approach.
Probabilistic class label: In two class classification, The probability of the class is computed instead of computing the majority for concluding the class.



Defining the probability for the class will be more useful, quantifying the un certainity.