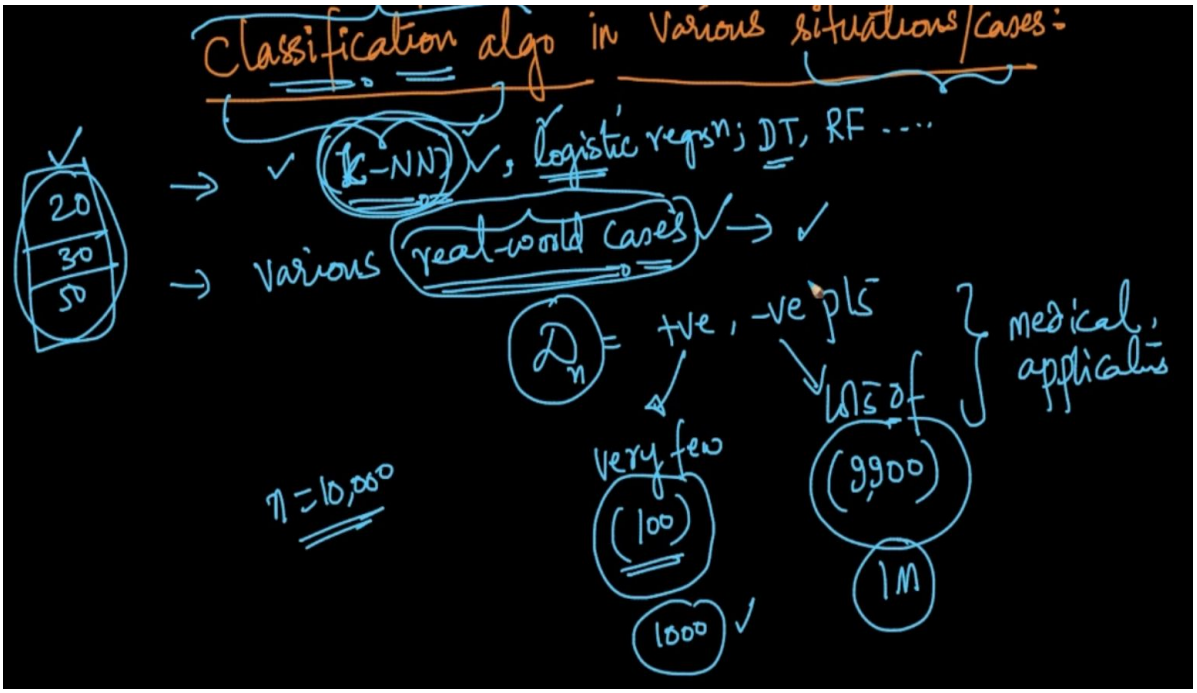
CLASSIFICATION ALGORITHM IN VARIOUS CASES

CLASSIFICATION ALGORITHM IN VARIOUS CASES

INTRODUCTION

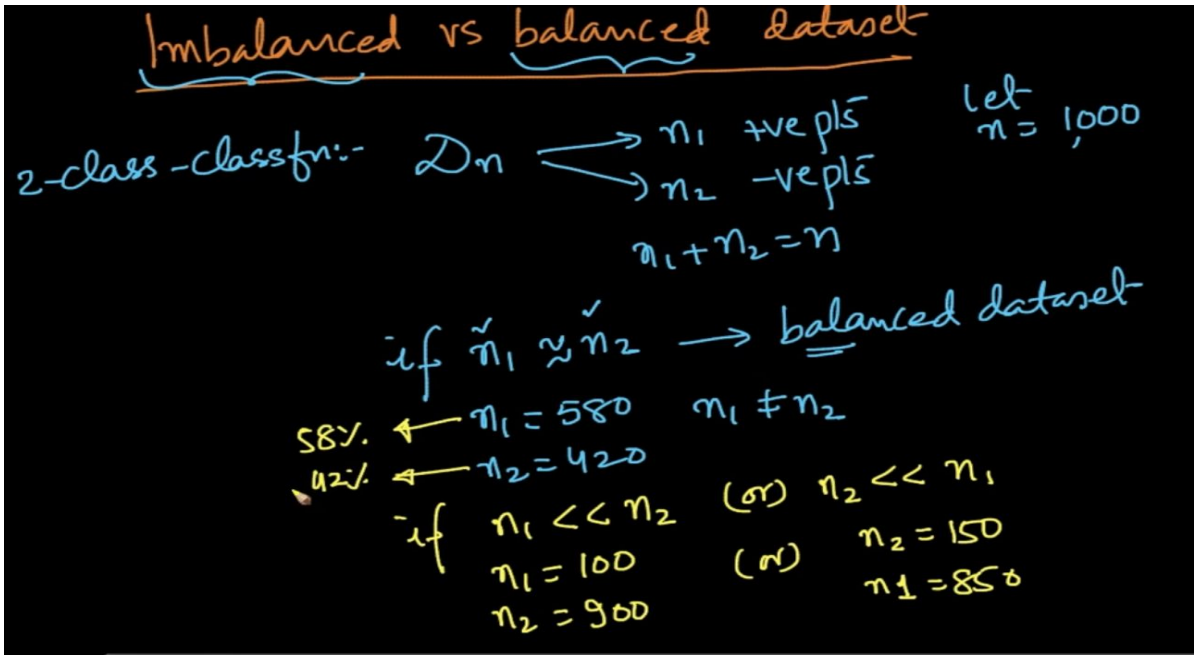


We'll know various classification Algorithms like k-NN, Logistic Regression, SVM etc but if we don't know how to apply that knowledge in different cases then it's not worth it.

Here, suppose we've medical dataset. In that there are only a few which are diagnosed with diseases from the total dataset. So we need to know how to modify our algorithm for this case

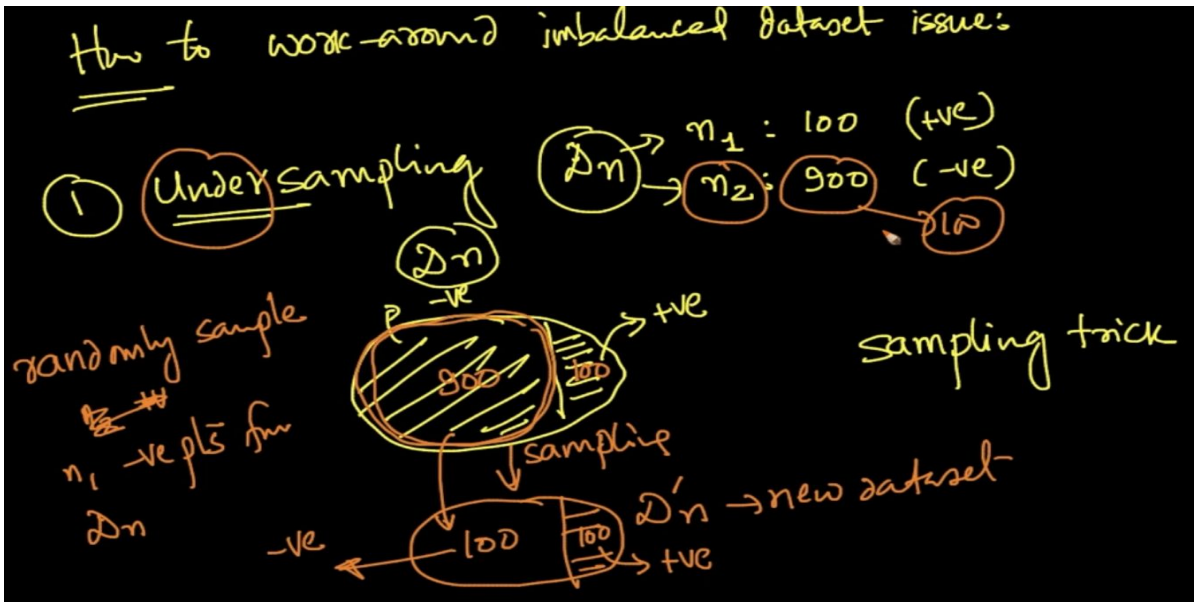
Knowing how to apply Algorithms in various cases is far more important then knowing the number of tricks/Methods you know

IMBALANCED v/s BALANCED DATASET

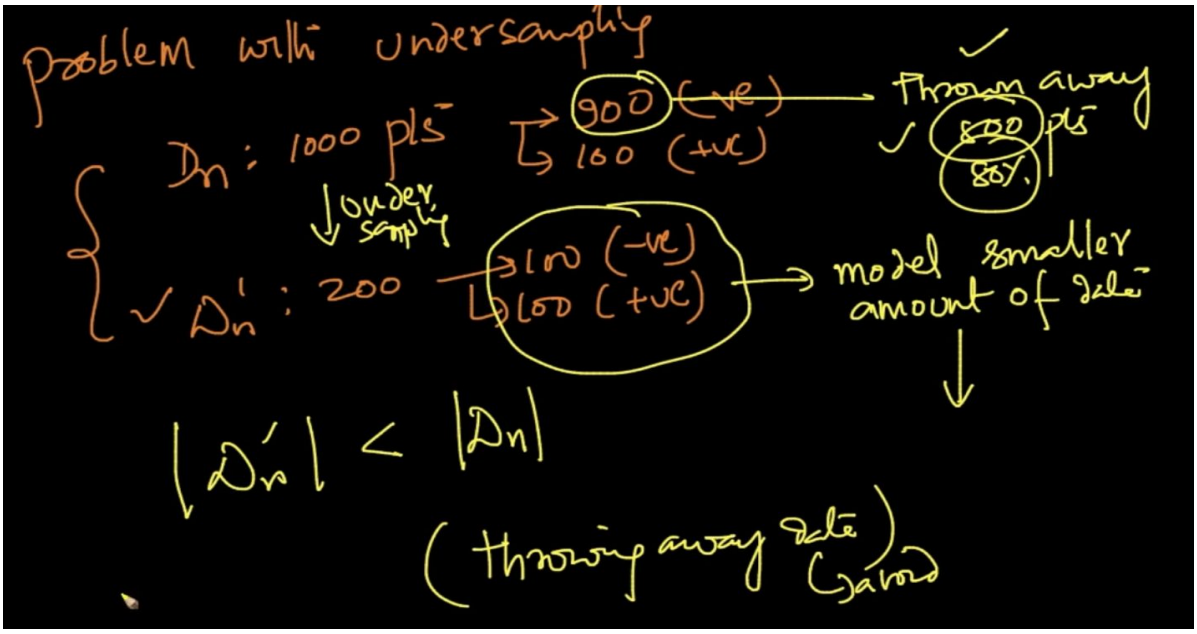


If there's a huge difference number of different class labels then it's imbalanced dataset

Here, n = 1000 , n1 = 900 (-ve pts) , n2 = 100 (+ve pts) so it's an imbalanced dataset

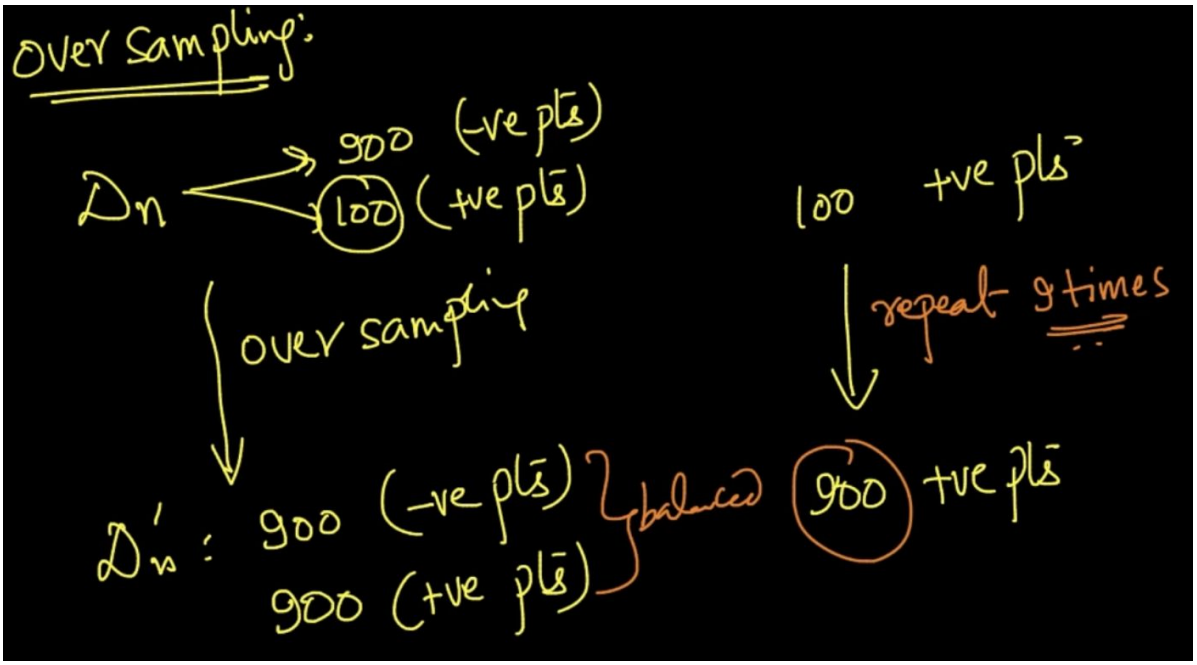


This is undersampling (1 method to handle imbalanced data) . We keep the minority class (n1) as it is and randomly take points as same as n1 (n2 is 900 we take 100 pts from that)

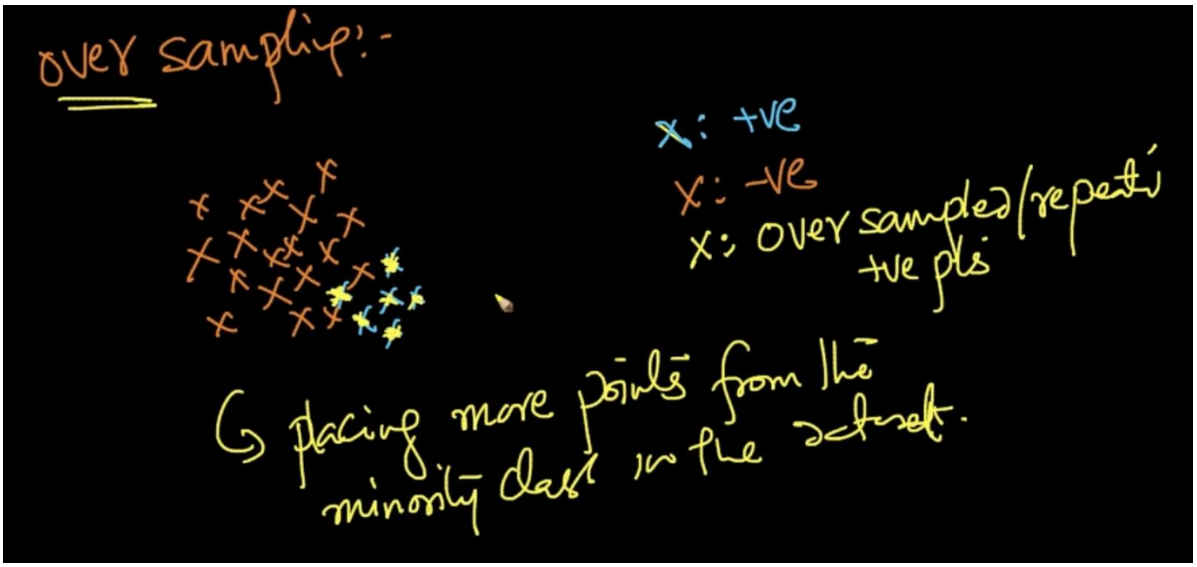


Problem with undersampling is we are throwing away data (80% here) which should not happen

OVERSAMPLING

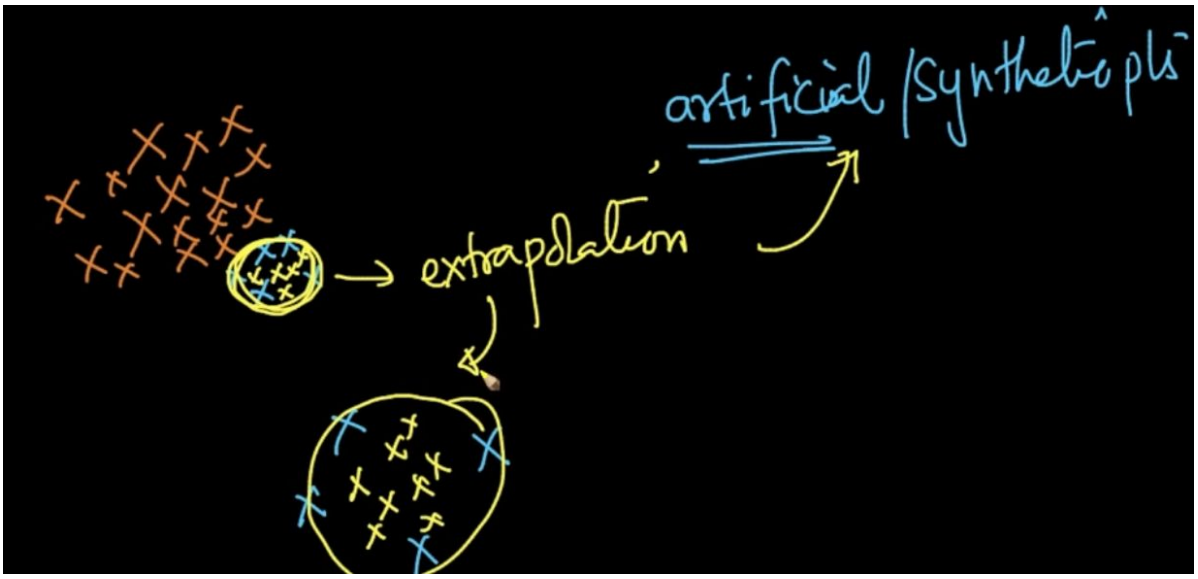


In this method, the minority class is repeated to level with majority class. Here, n1 = 100 So we repeat it 9 times to balance it with majority class (n2 = 900). Now both balanced



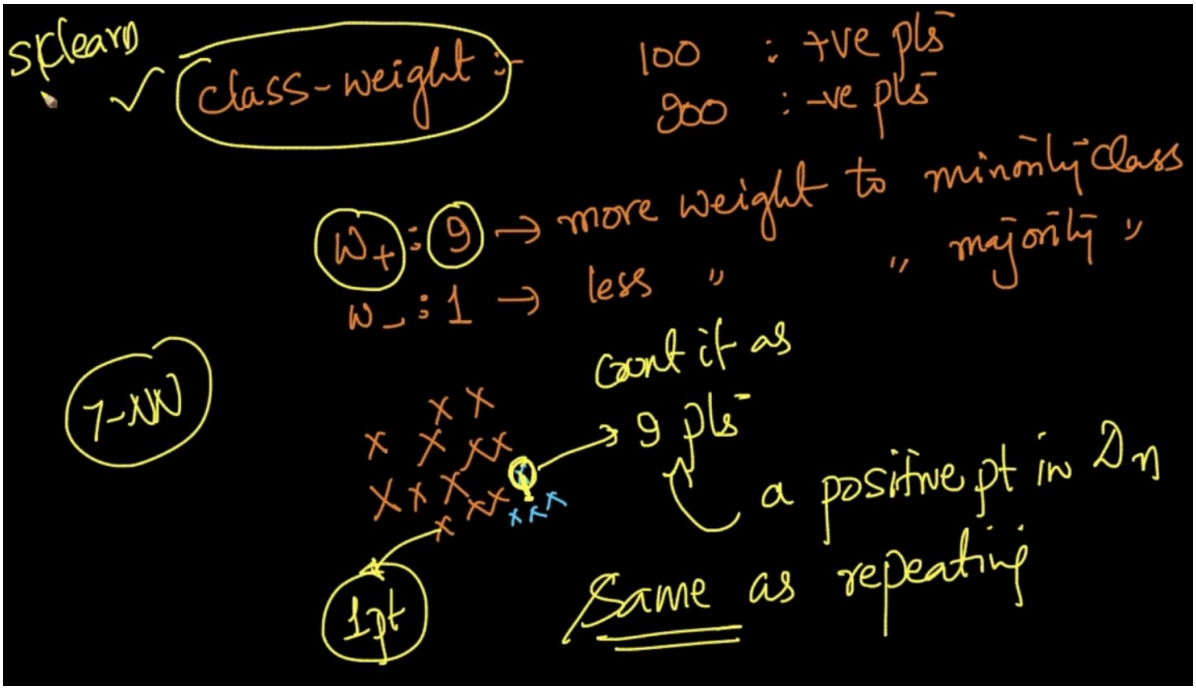
Here in the minority class, we are placing the same points 9 times on top of minority class points

EXTRAPOLATION/ ARTIFICIAL POINTS CREATION



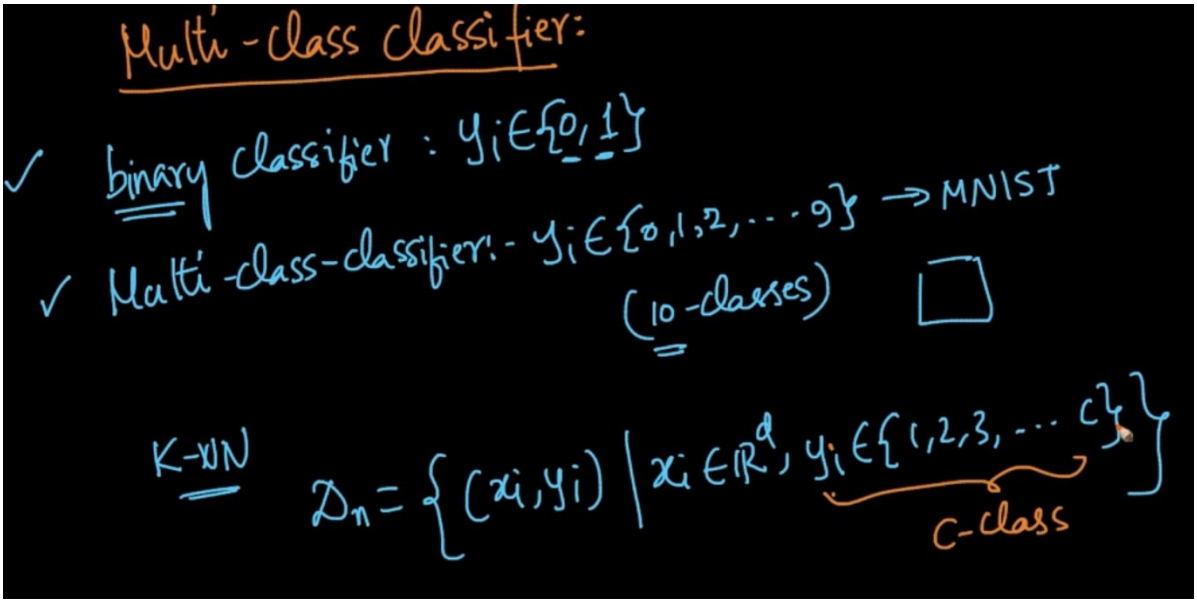
For the minority class region we are artificially creating points on it to balance the dataset

CLASS WEIGHT

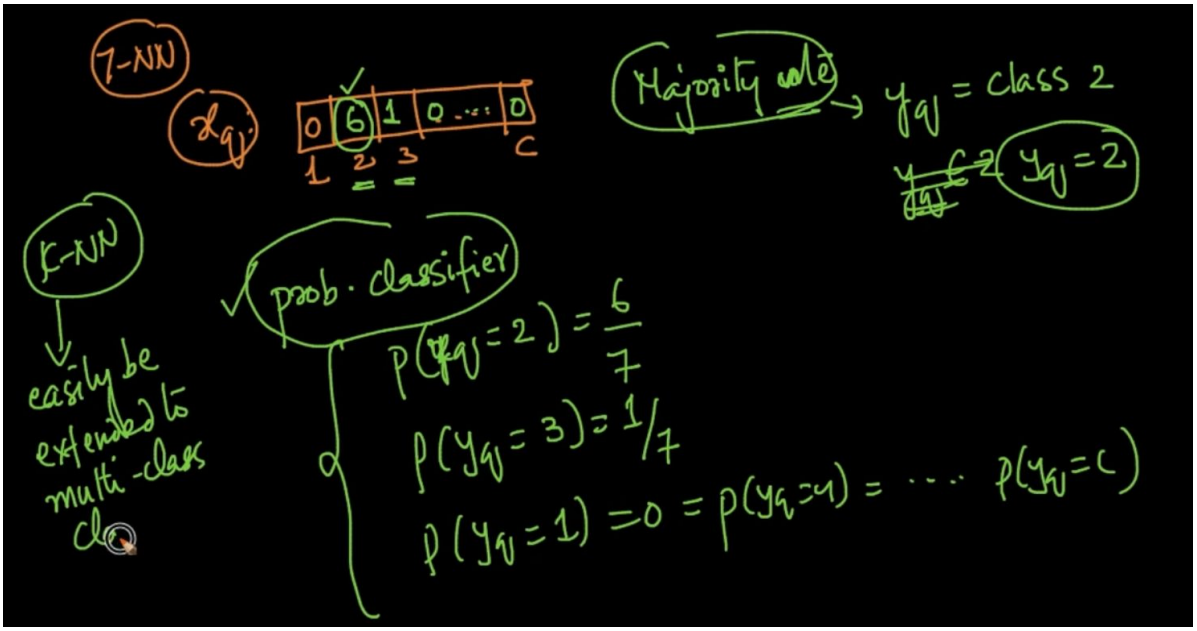


More weight is being given to minority class . So, if we are using a pt from minority class We count it as 9 point. So if we are using 7-NN and a point is near it all the nearest neighbors will be from that point. Same as repeating

MULTI CLASS CLASSIFICATION

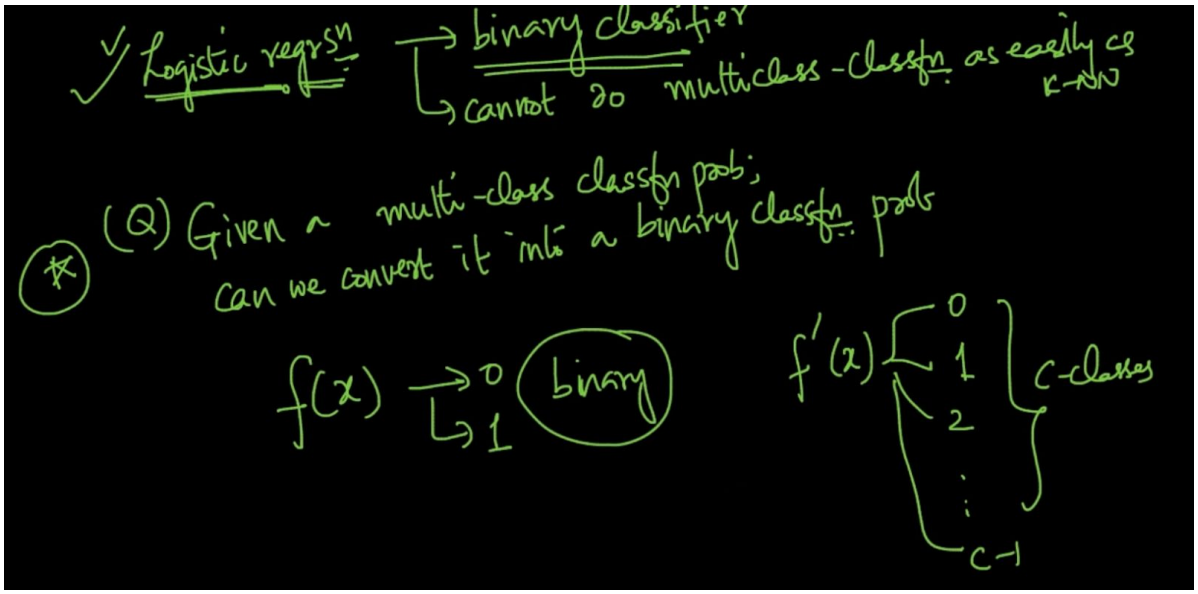


There's a c number of classes in our dataset. How does k-NN work in that

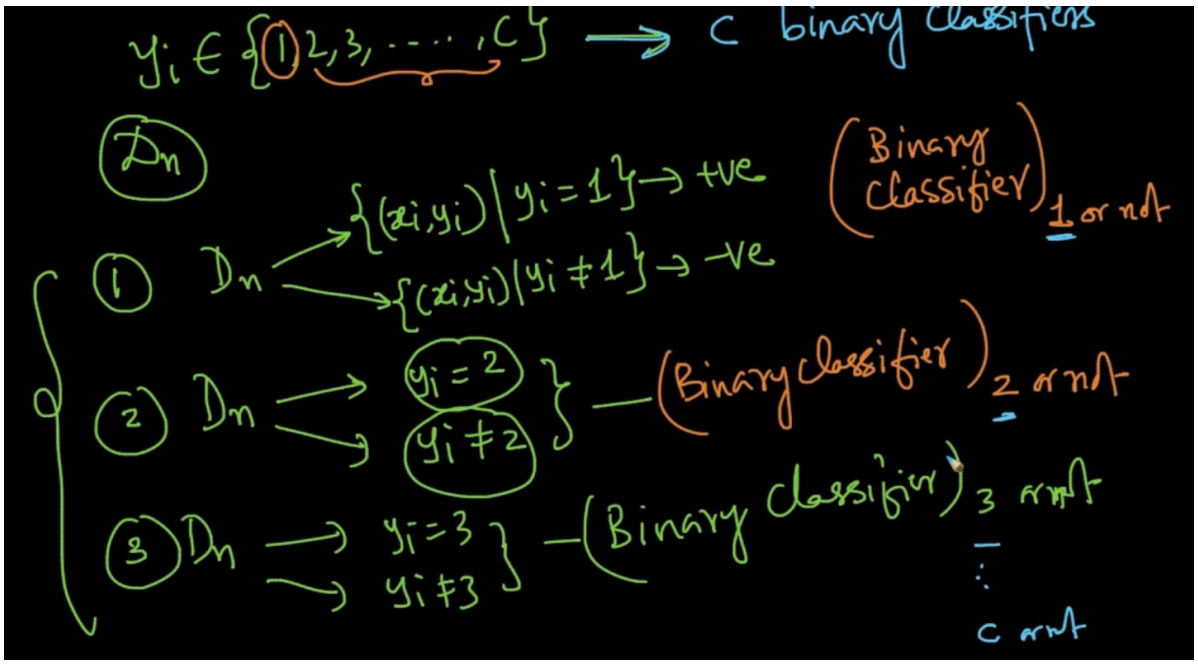


If we've x_q and running 7-NN and we are getting 6 values in class 2 and 1 in class 3

Then by majority vote y_q = 2 or if probability classifier then P(x_q = 2) = 6/7. Same can be done for other classes as well. So k-NN can easily be extended to multi-class classification

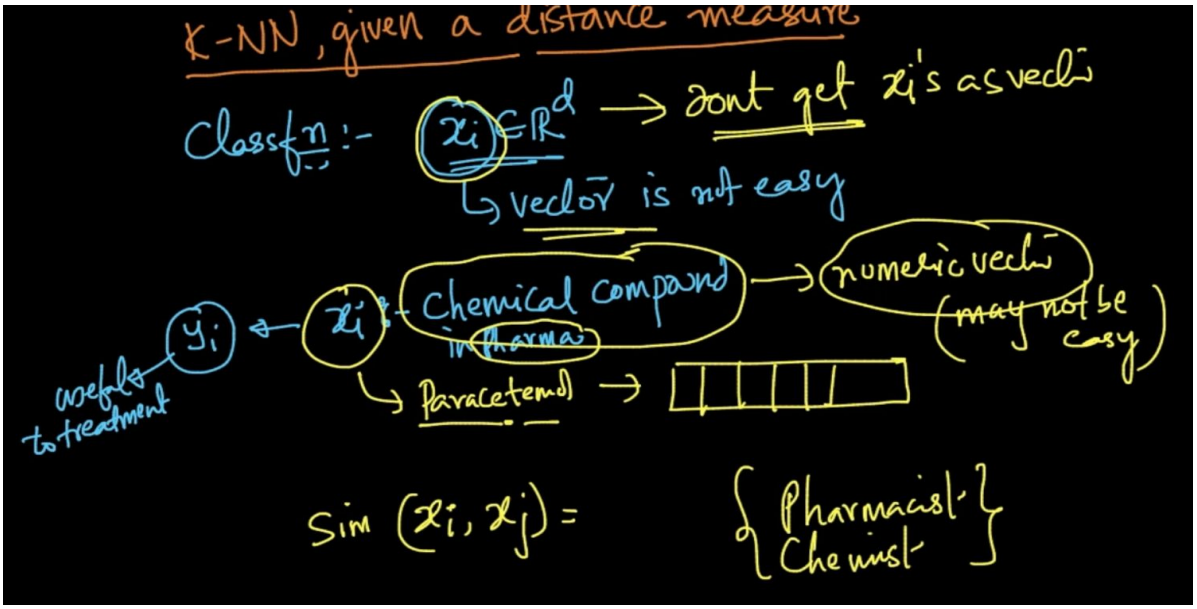


We can't do multi class classification on Logistic regression . So can we do anything to convert multiclass to a binary class f'(x) -> f(x)

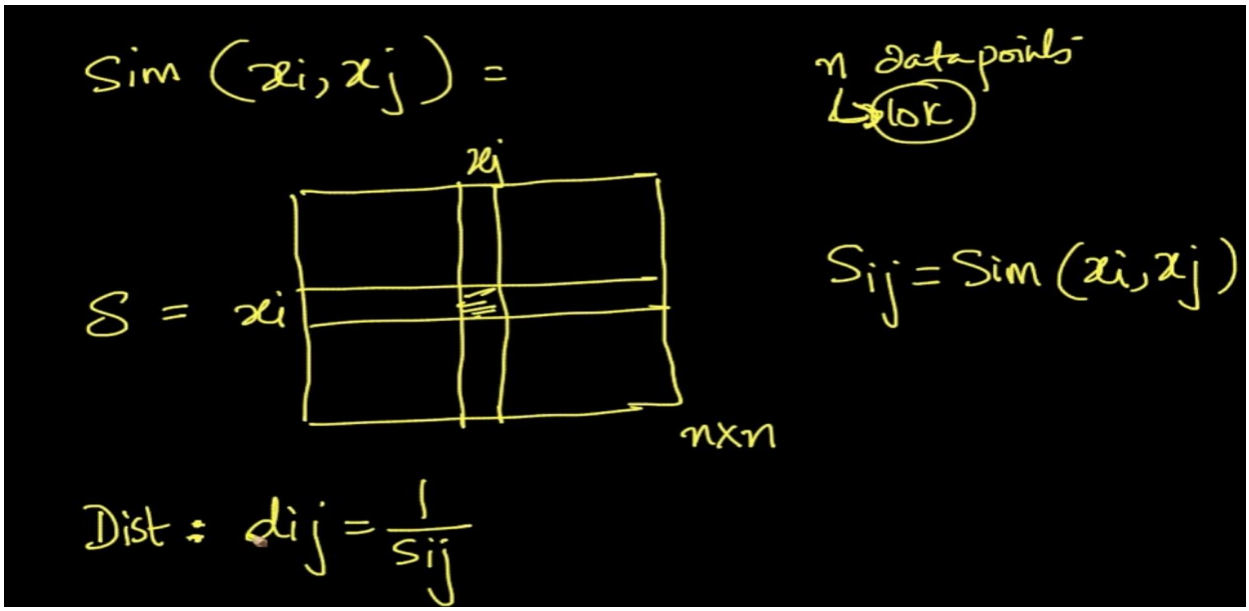


Here we are converting multi class into binary classification by breaking the dataset into 2 parts . For e.g : y_i ∈ {1,2,3,4, ...c} . D_n is broken into two parts 1st is y_i = 1 ,2nd is $y_i \neq 1$ Same for other classes . This is called 1 vs Rest method. We are training c- models here as No. of classes are c

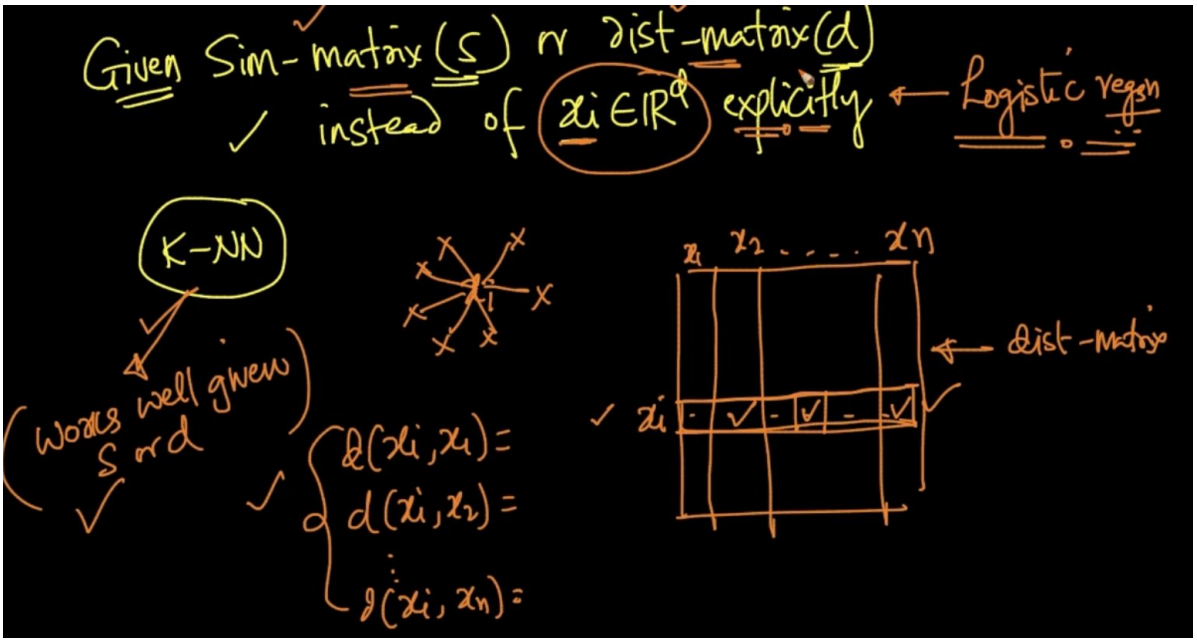
k-NN, GIVEN A DISTANCE MATRIX OR SIMILARITY MATRIX



Sometimes , x cannot be converted into a numeric vector. For e.g a chemical compound but similarity between 2 chemical compounds can be calculated

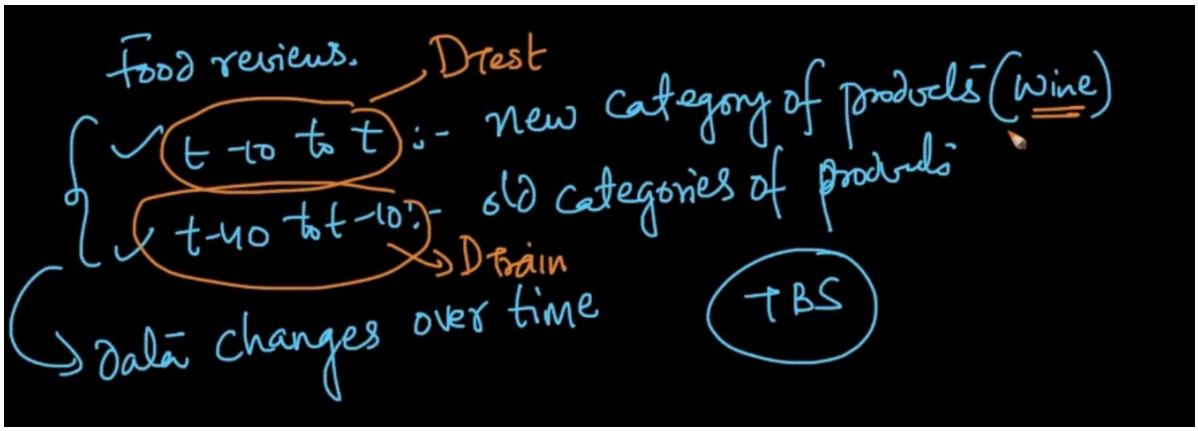


A similarity matrix S of all the chemical compound is created. S_{ij} tells us about the similarity between two chemicals x_i, x_j . Distance matrix can also be created .

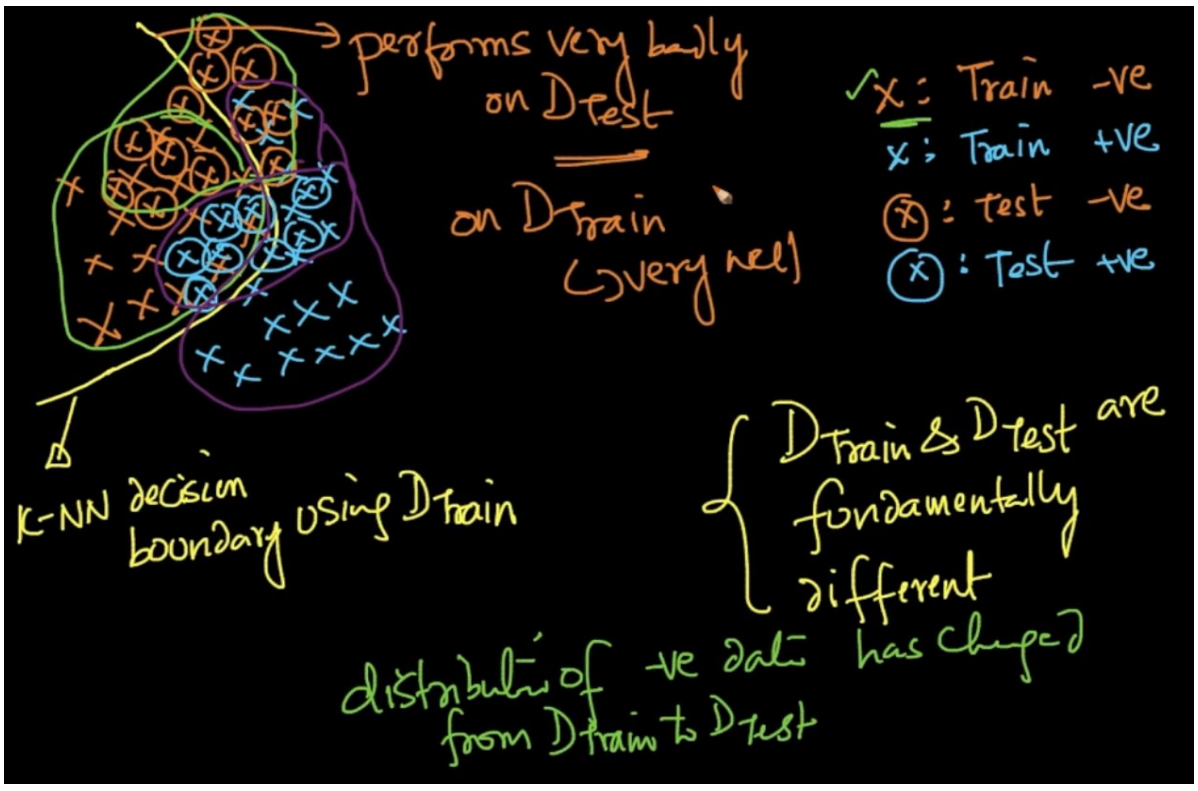


Given distance matrix for all x . We get the 3 nearest neighbors if k=3 by using distance matrix . So, k-NN can work awesomely with Similarity matrix or distance matrix

TRAIN AND TEST SET DIFFERENCES

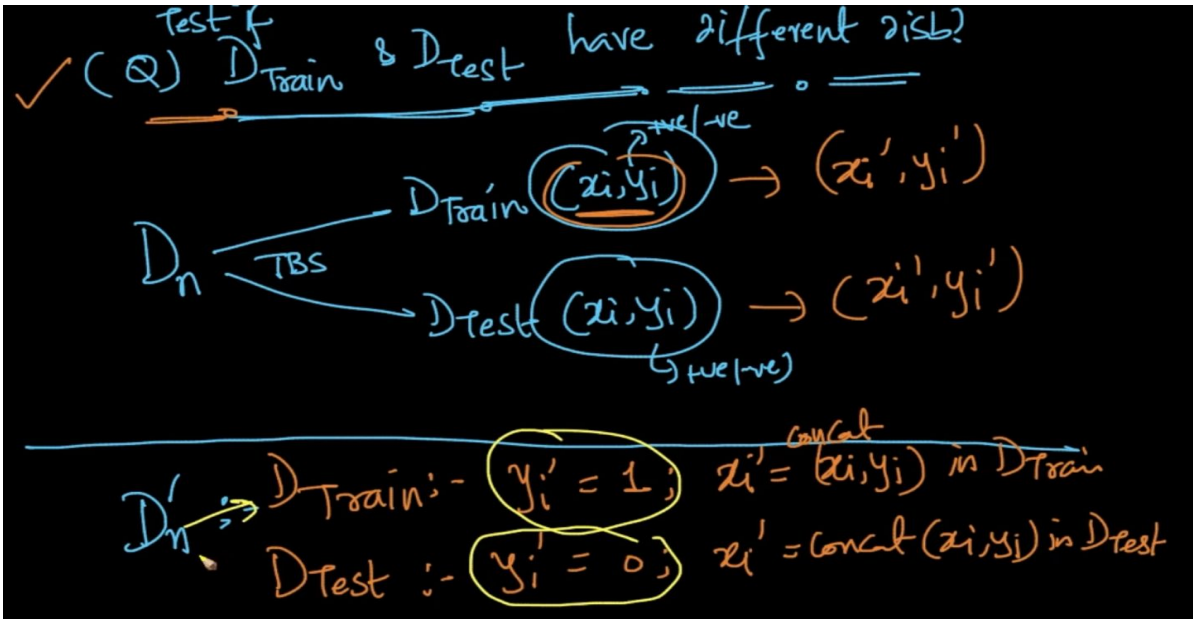


In TBS data can be changed overtime as seen i.e new products can be added or old products removed in Amazon Food Reviews Dataset

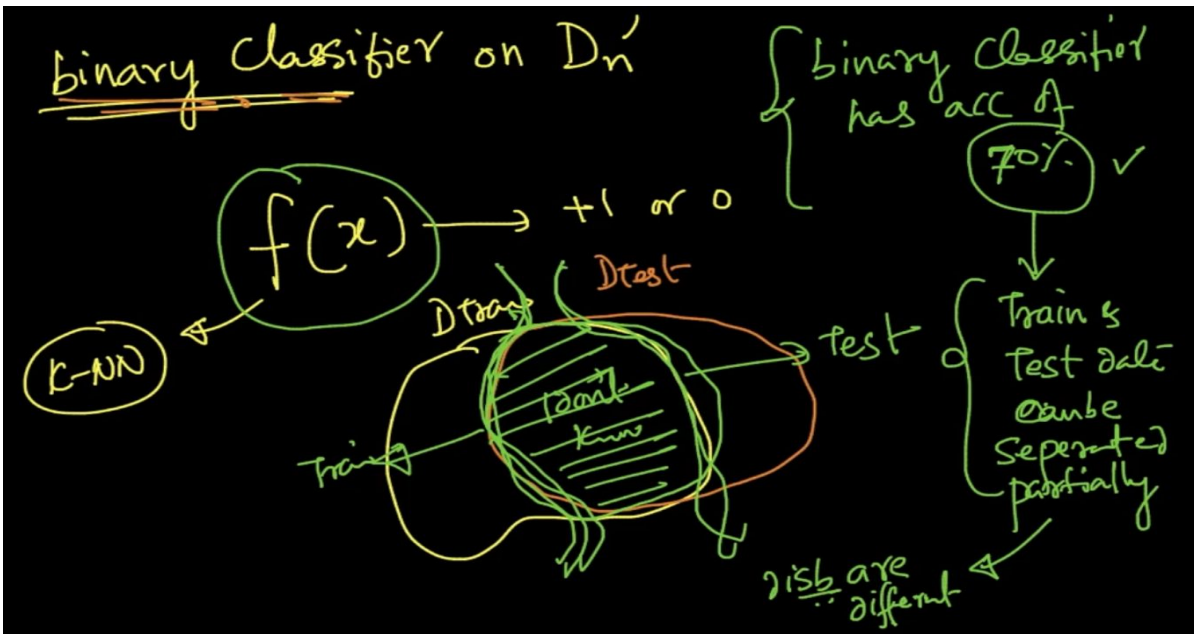


Our decision boundary from k-NN performs very well on Train data but with a change of data overtime our distributions can change . So k-NN will perform badly in Test data.So our CV error will be low but Test error is very high.

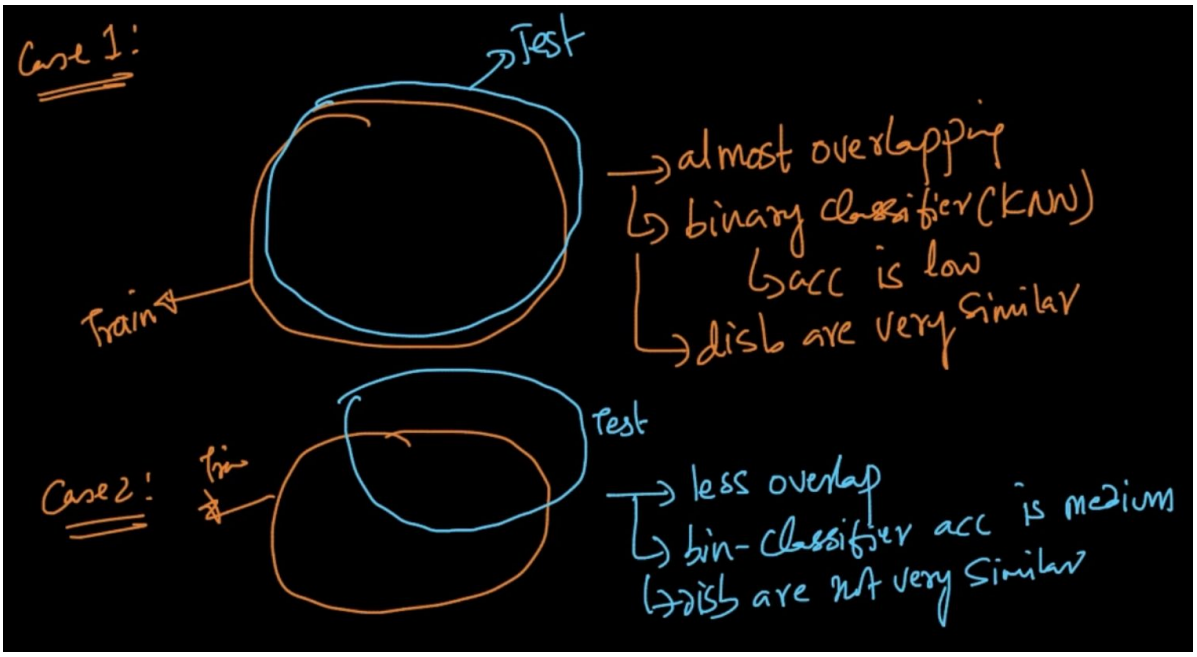
So how to determine if data is changing overtime ?



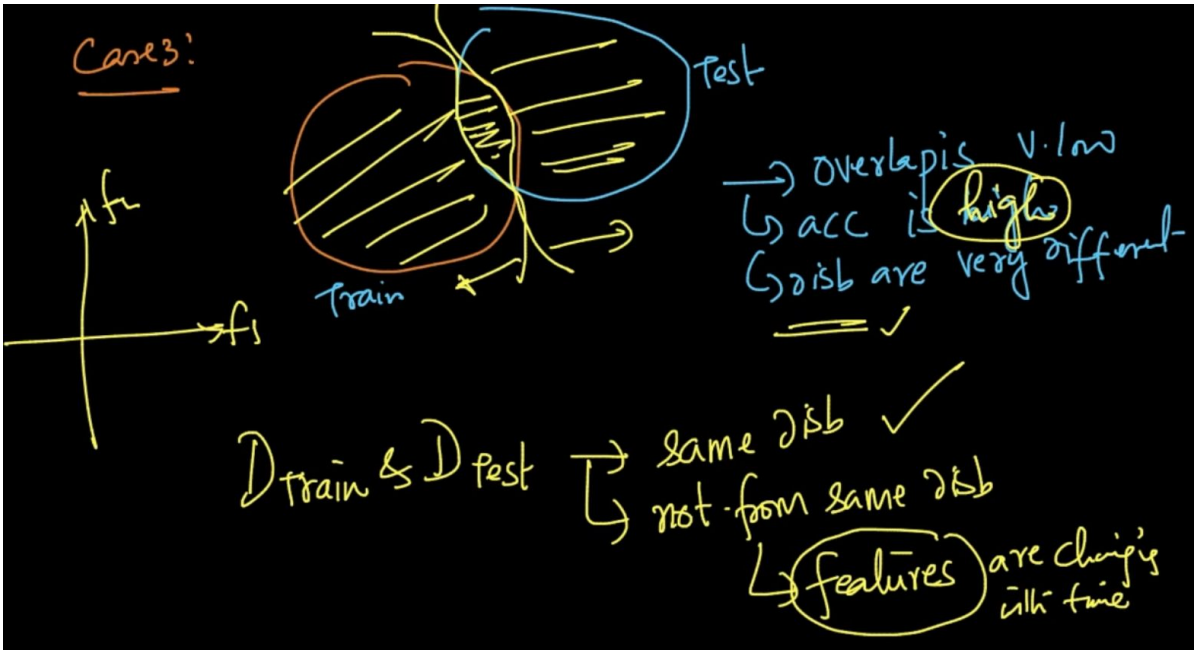
So , we make a new dataset D'_n and Train data $x_i^{'}$ = concat (x_i, y_i) of previous Train data D_{train} and $y'_i = 1$. Same for test data except $y_i^{'}$ = 0



So we are doing binary classification on D'_n and check accuracy. Here it is 70% means it Train and Test data can be separated partially which implies distributions are different. Since the overlap isn't much binary classification performs good here

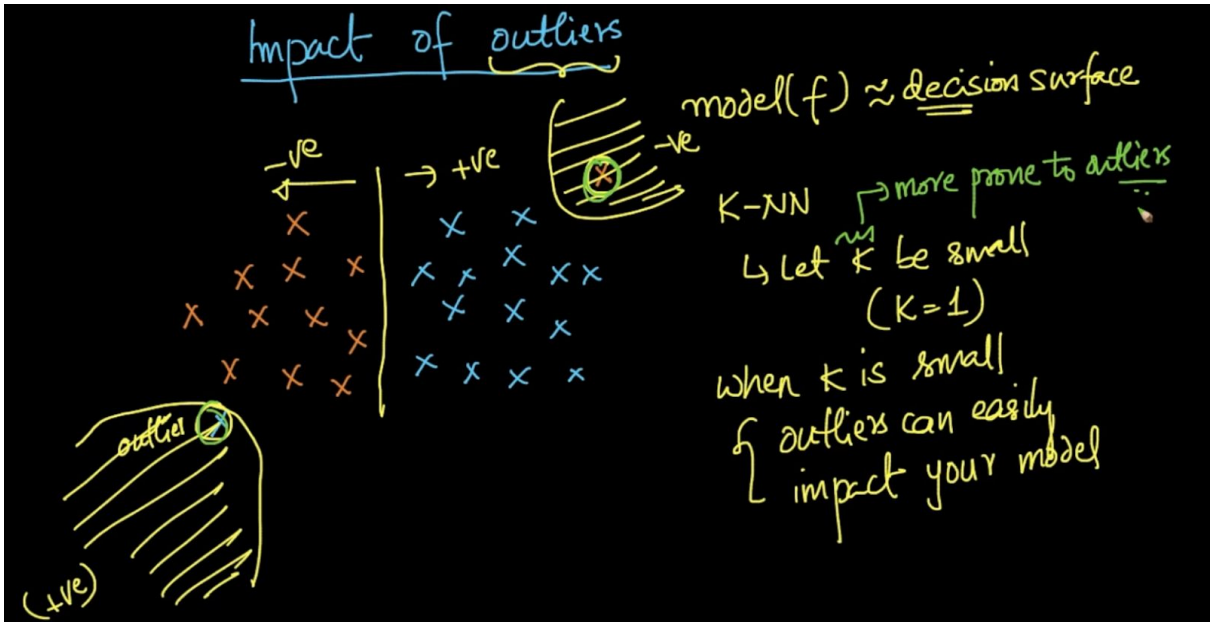


Case 1: Since they are overlapping they've the same distributions therefore they can't be seperated by binary classification i.e it;s accuracy will be low

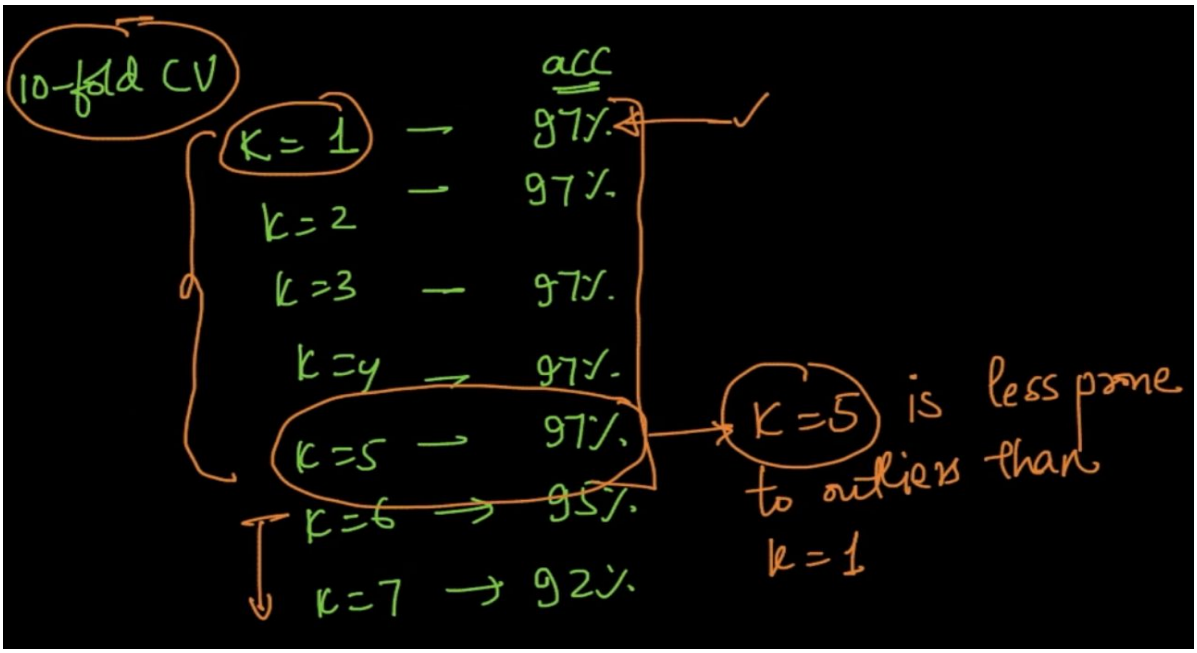


Case 3: Overlap is very low so binary classifier's accuracy will be very high . It means that features are changing over time or they are temporally stable. To tackle this we build/change/design new features

IMPACT OF OUTLIERS

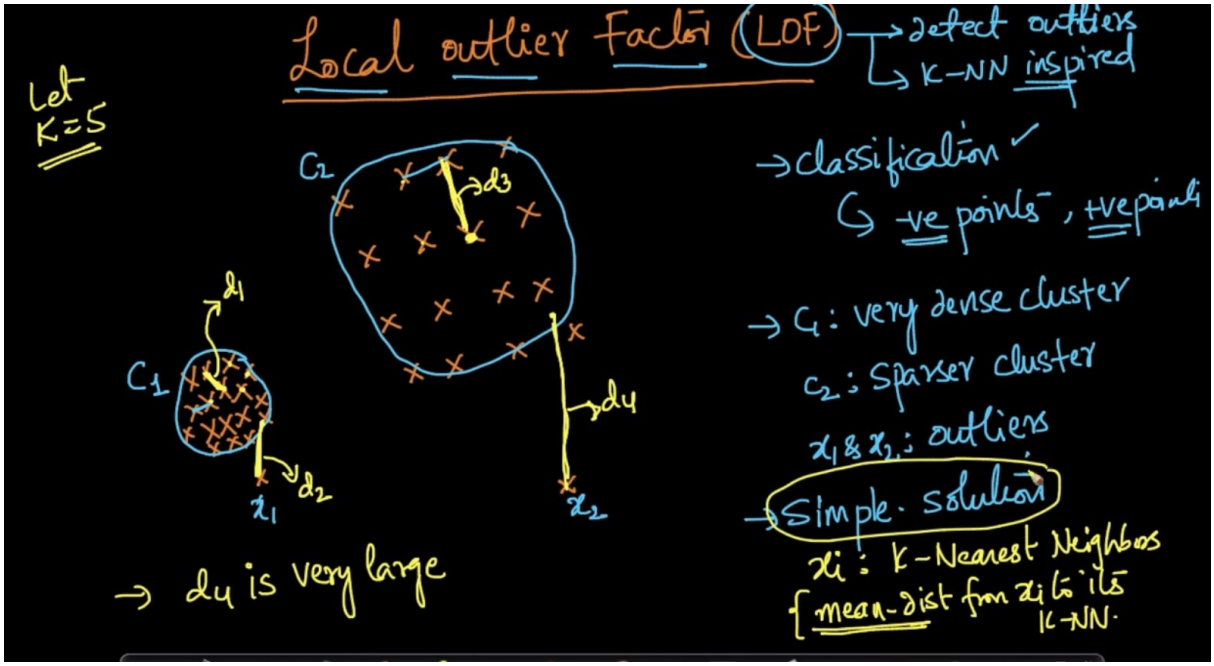


If k=1 and outlier is present then it'll mark all the query point with it's class label which is wrong

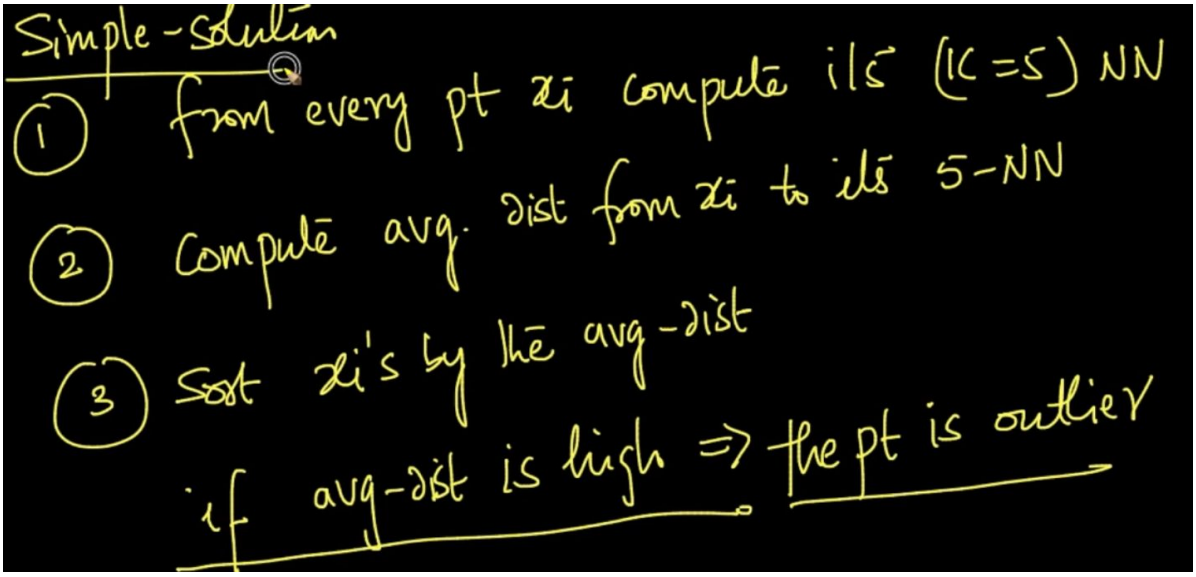


In k-Fold CV if we are getting same accuracies on different k's we'll choose a high k since they are less prone to errors . Here, k = 5

LOCAL OUTLIER FACTOR (MEAN DISTANCE TO KNN)

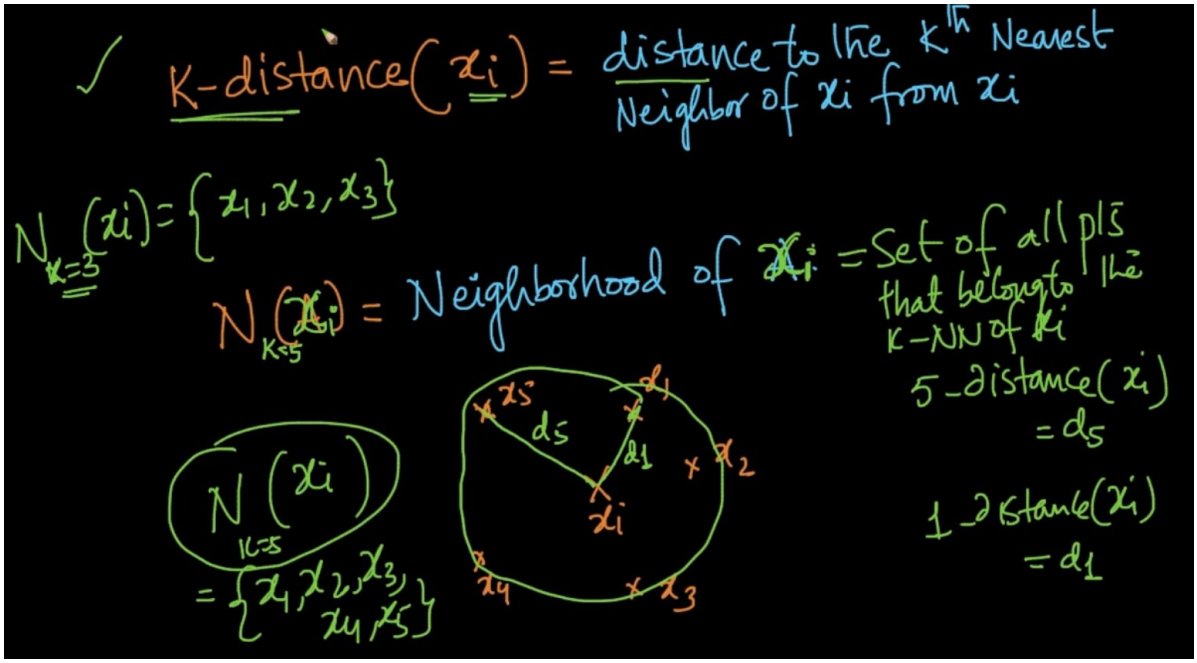


Mean distances of all data points $x_i's$ are calculated .

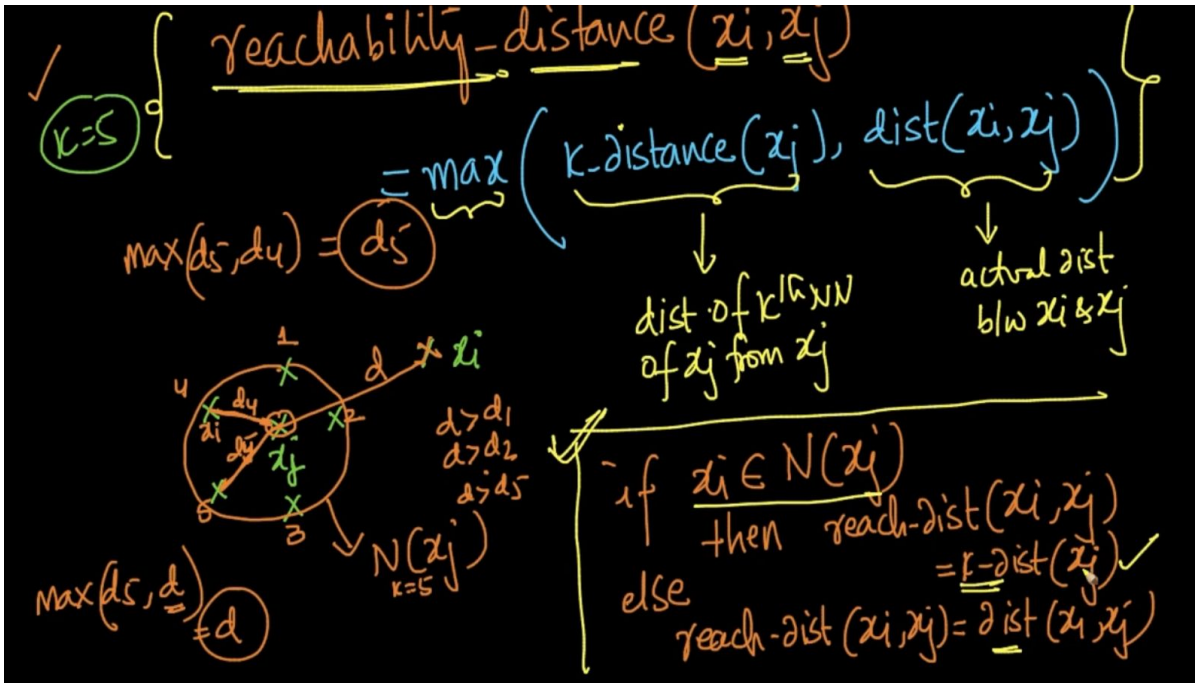


So, x_2 & x_4 are outliers but with this solution only x_2 will be considered an outlier as d2 ≃ d3 means if we remove outlier x_1 with mean dist d2 then all the other points i.e of sparse cluster c2 has to be removed

k - DISTANCE & REACHABILITY DISTANCE

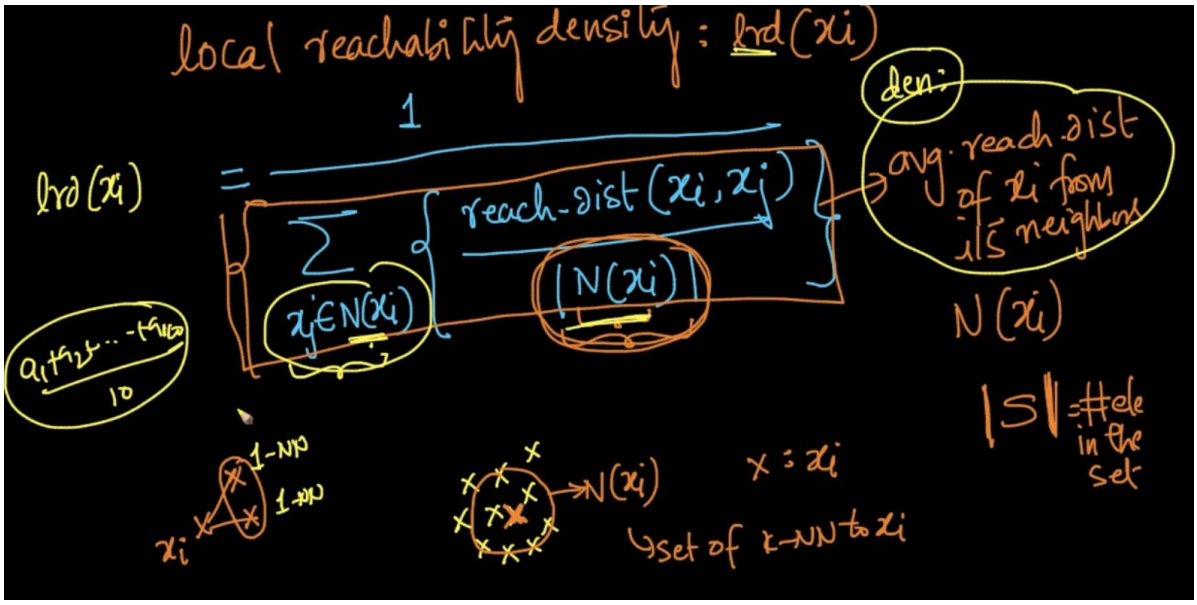


5-Distance (d5) > 1-distance (d1). Neighborhood is also explained pretty well

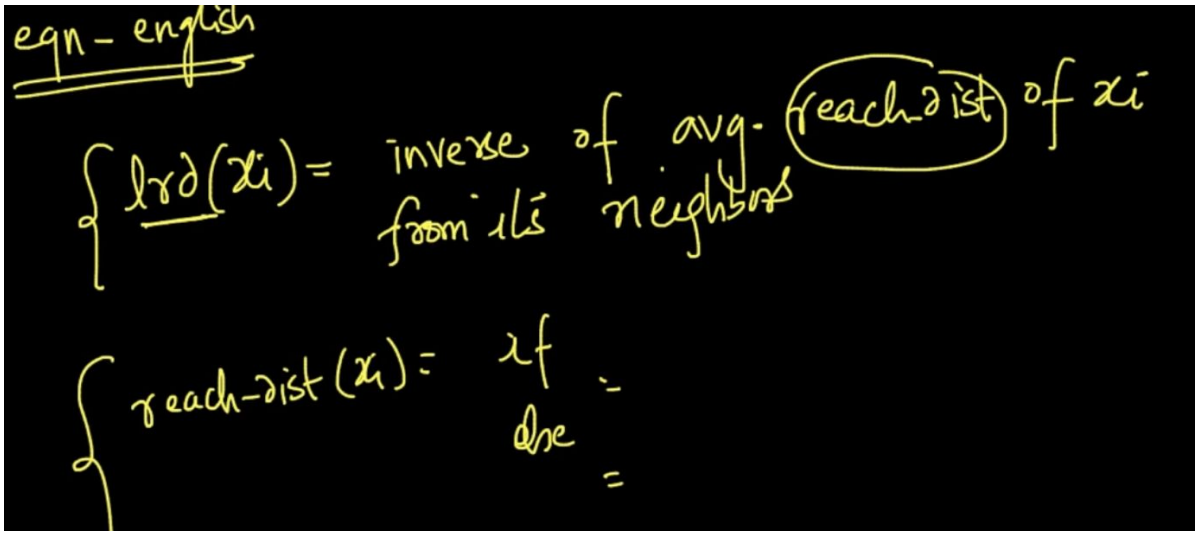


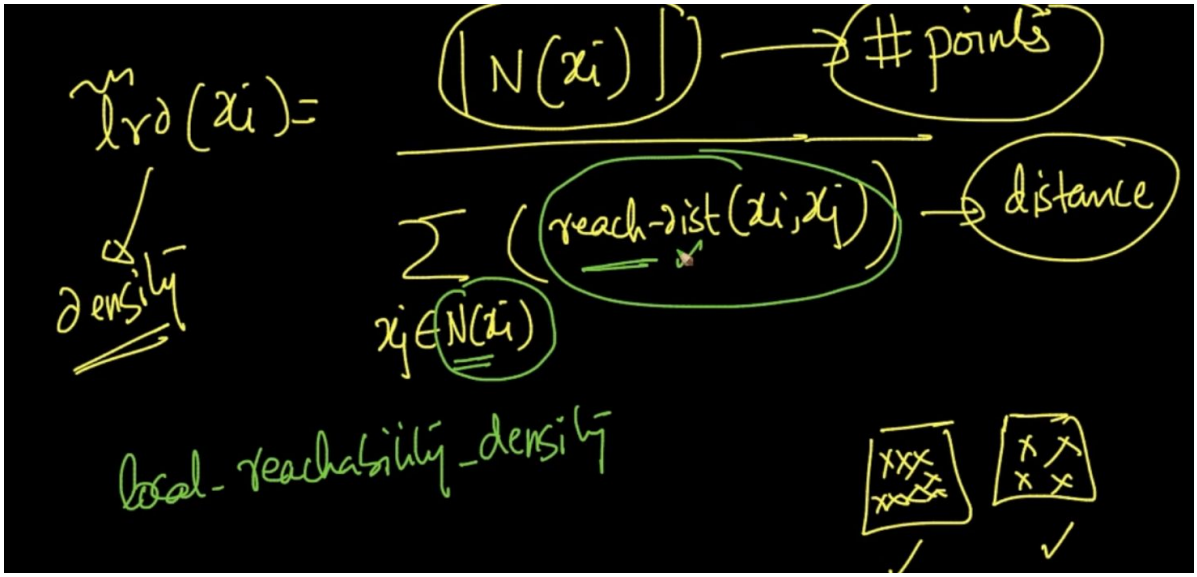
Check the if - else condition to understand it

LOCAL REACHABILITY DENSITY



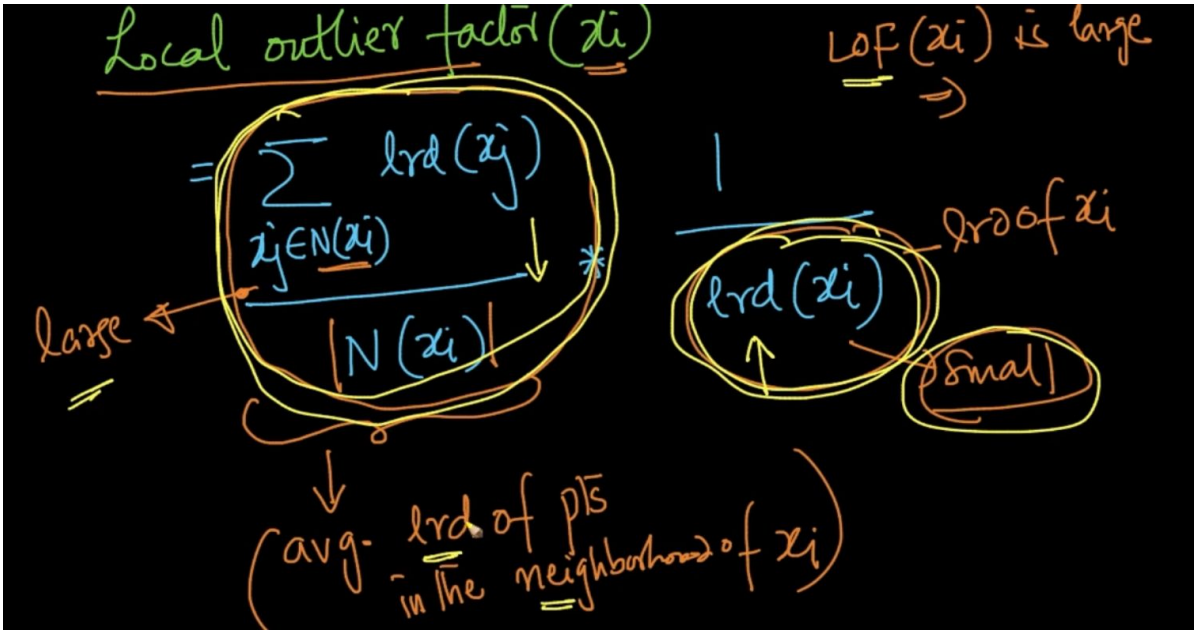
Lrd(x_i) is inverse of summation of reachability distance of all the points belonging to the neighborhood of a point x_i divided by |N(x_i)| i.e number of points in Neighborhood which is basically k but note that two points can also have the same NN .



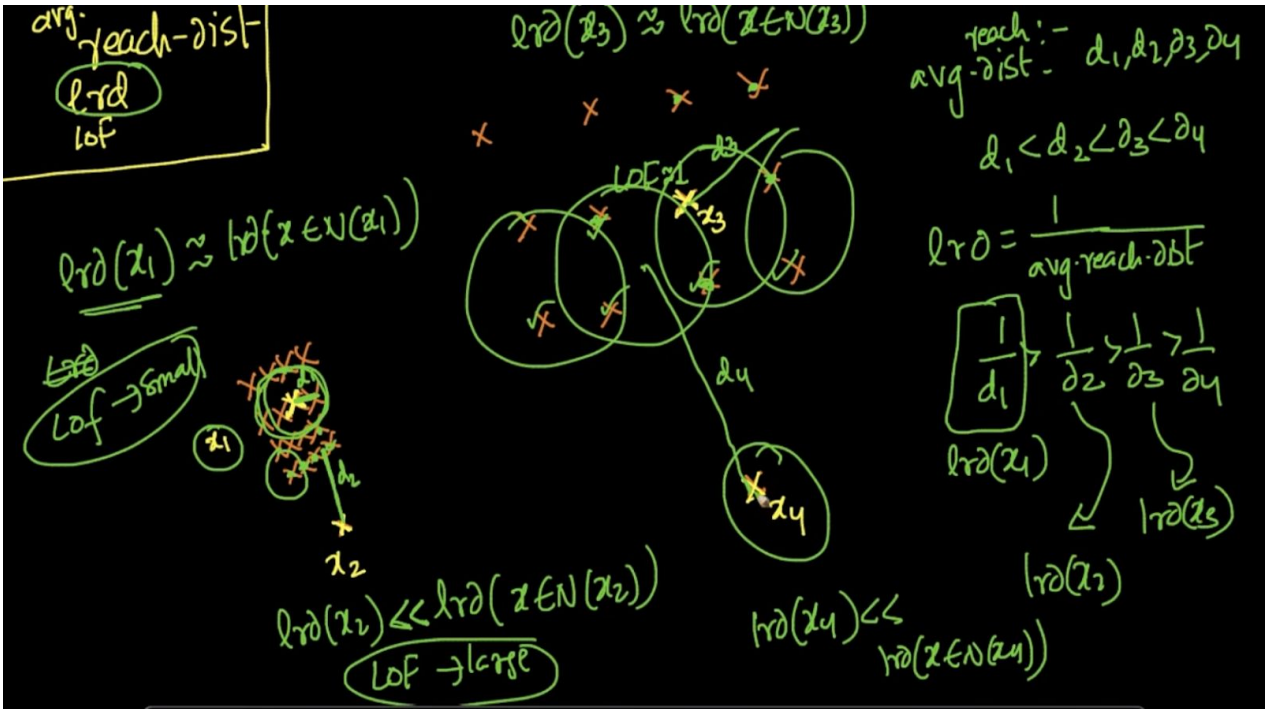


lrd(x_i) is basically a density as it is No. of points/ Summation of reachability distance Of all the neighborhood points

LOCAL OUTLIER FACTOR



LOF(x_i) is large if the right lrd is small and left part is large. It simply means that density of points around x_i is small but density of it's nearest neighbors is huge

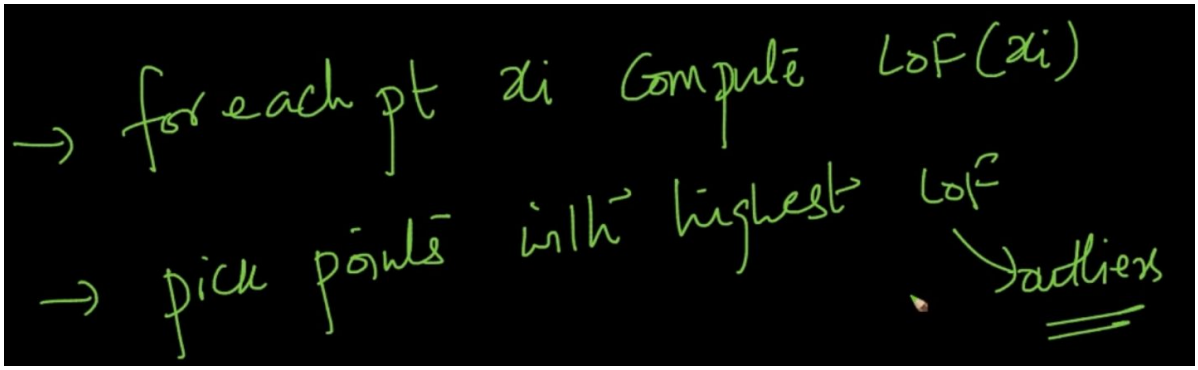


Avg reach distance is calculated but it ain't enough and so is lrd calculation. So LOF is used.

For x_1 :Density is great so lrd is also great and lrd(x_1) $\approx$ lrd (x $\in$ N(x_1)) since they are very close and when we apply that to LOF formula above LOF(x_1) will be small

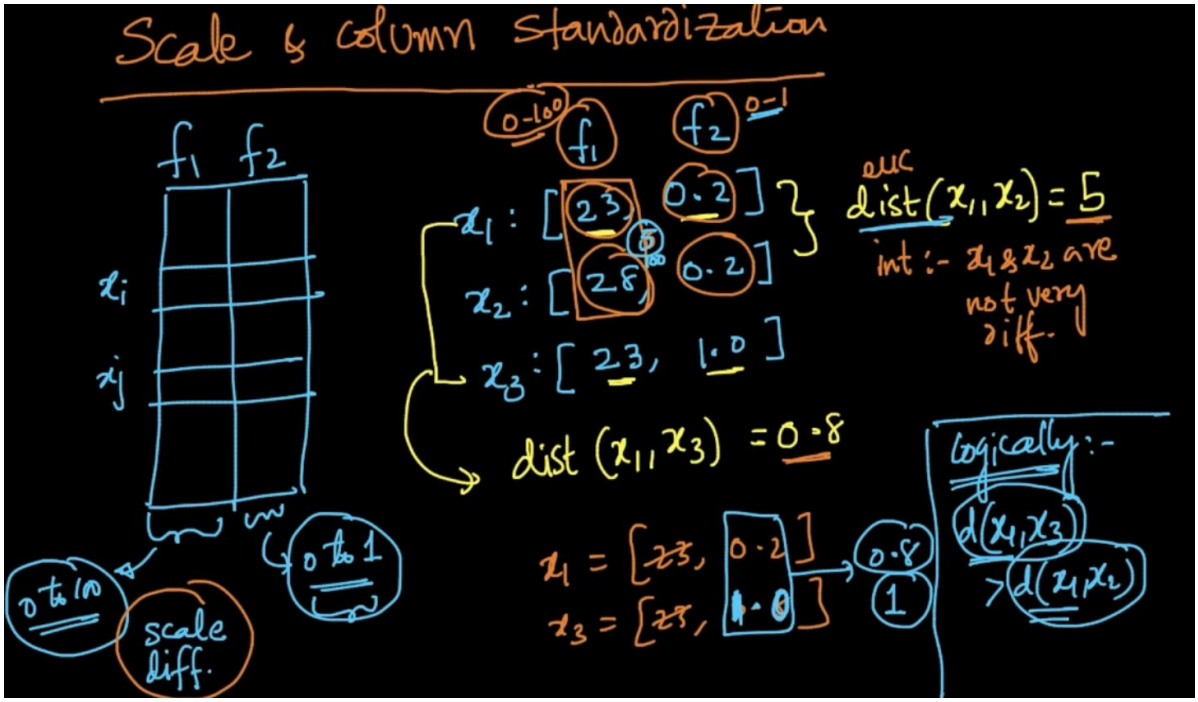
For x_2 : Density is none and therefore lrd(x_2) << lrd (N (x_2)) so LOF(x_2) is large.

Same concept for other points x_3 & x_4. So, we can observe that if LOF is large then it's an outlier



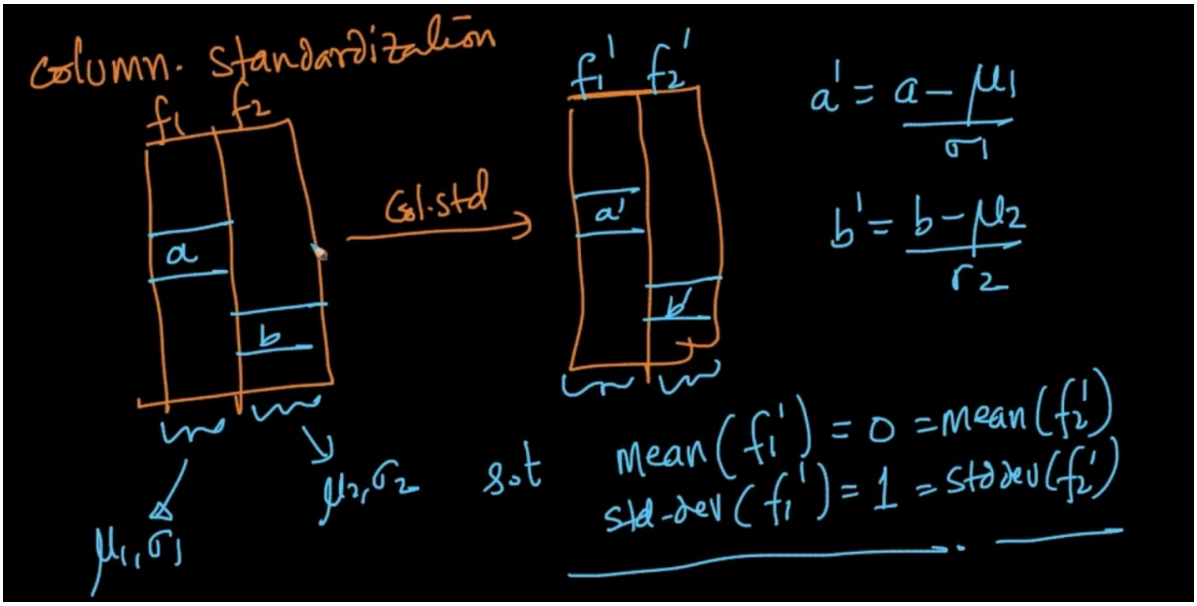
After doing it remove some percentage of outliers.

IMPACT OF COLUMN & SCALE STANDARDIZATION

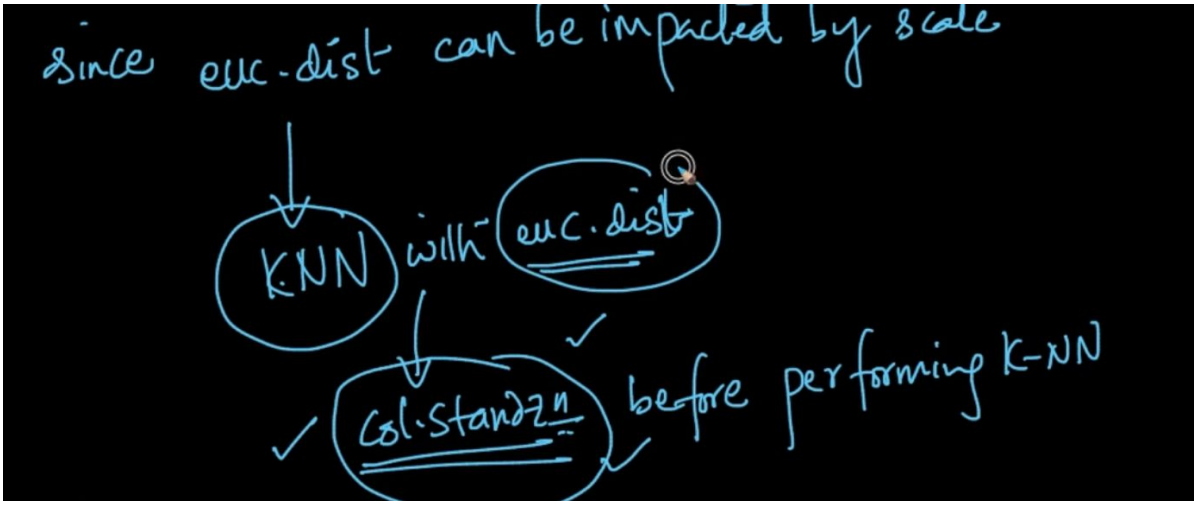


We've 2 features f_1, f_2 and their Euclidean distances are calculated. euc-dist(x_1, x_2) = 5 and euc-dist(x_1, x_3) = 0.8 but distance of x_1, x_2 is coming from f_1 scaling from 0-100 .So distance 5 isn't much of a difference because it is just 5% of variability but distance of x_1, x_3 is coming from f_2 which is scaling from 0-1 so distance 0.8 denotes 80% variability so logically d(x_1, x_3) > d(x_1, x_2) .So Euclidean distance is impacted by scales

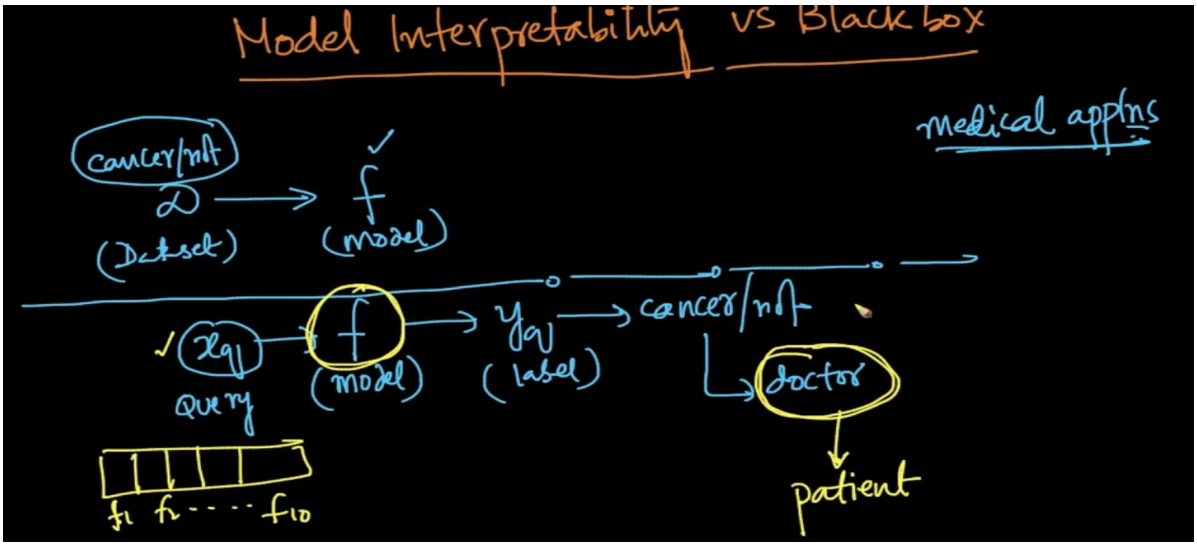
So how do we tackle this?



So we bring the features to the same scale by making their mean 0 and std-dev 1

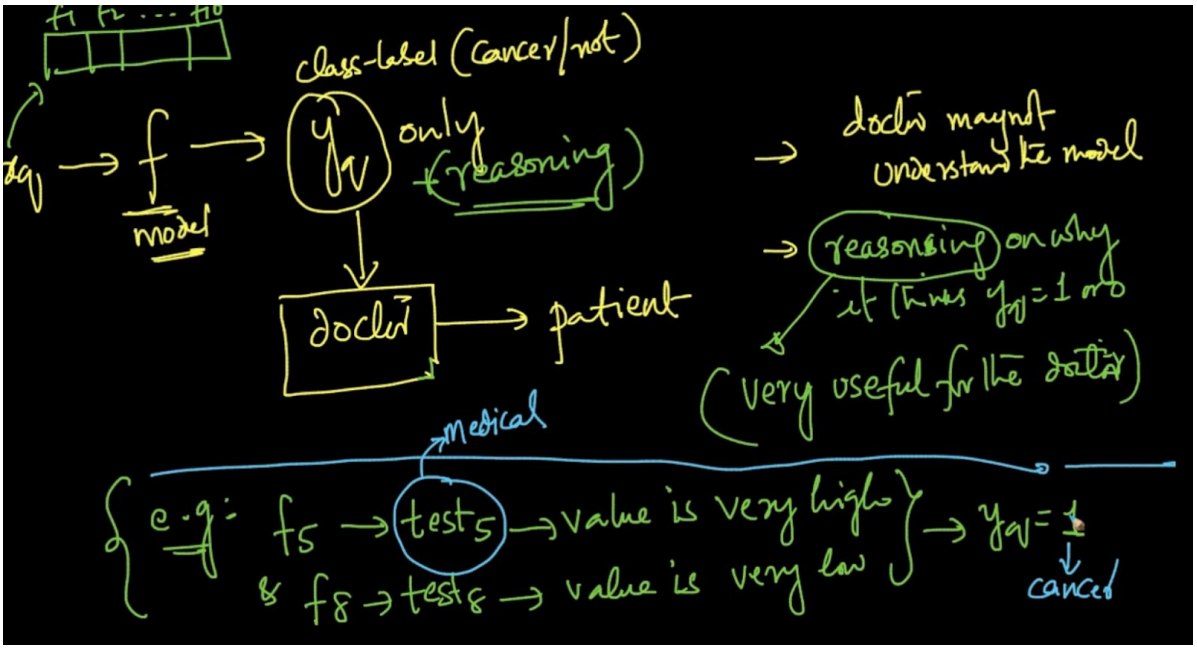


INTERPRETABILITY



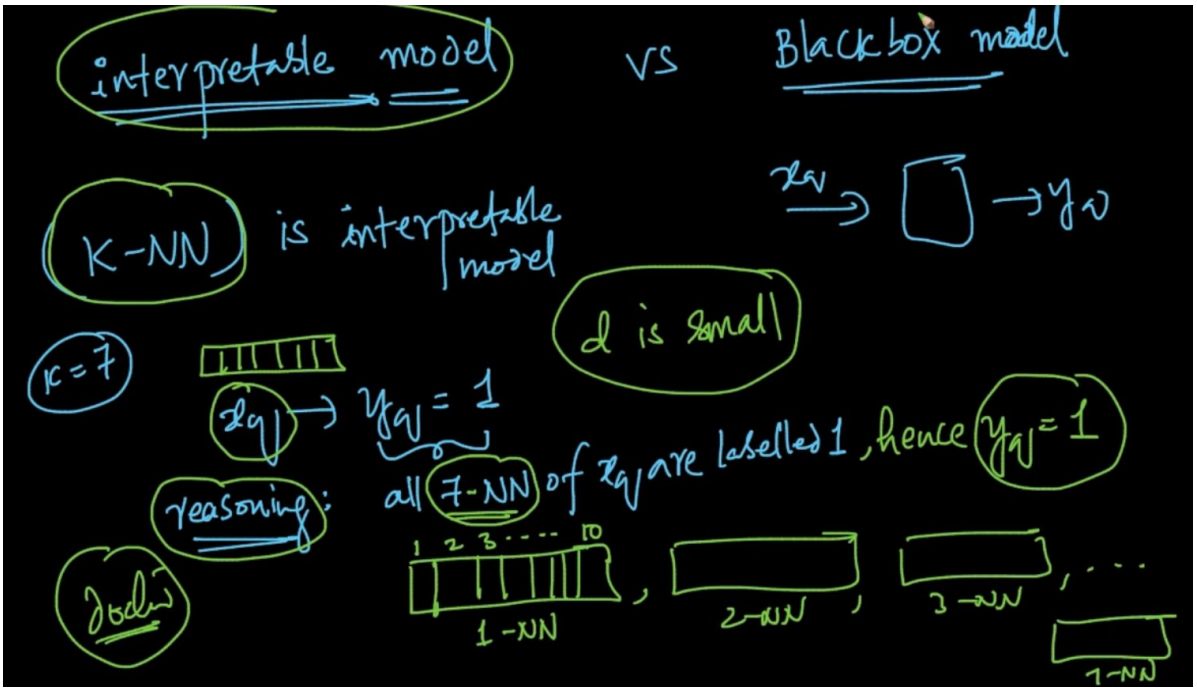
Suppose we've model for cancer diagnosis and we train it to predict cancer in a patient.

If our model just gives prediction label y_q only the Doctor can't be sure because he isn't an ML Engineer so he can't be sure with just y_q



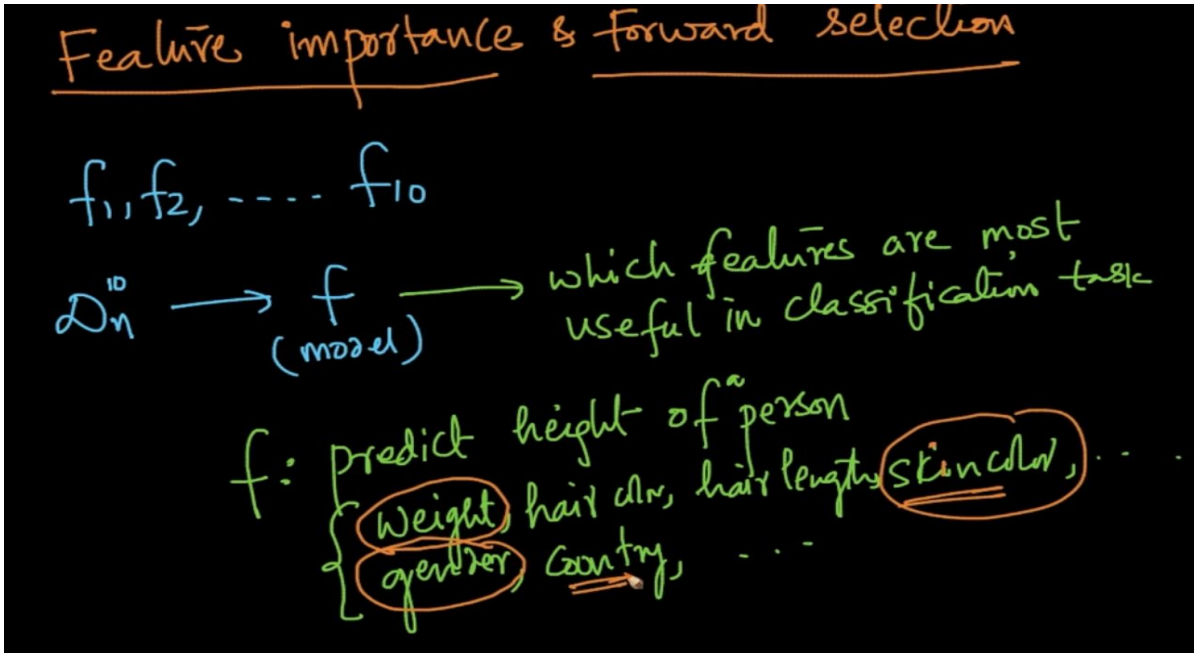
So if the model also give reasoning like above where some medical tests which are features of dataset have some values like high/low that's why y_q = 1 i.e cancer

If model can give explanation like this it's called interpretable model

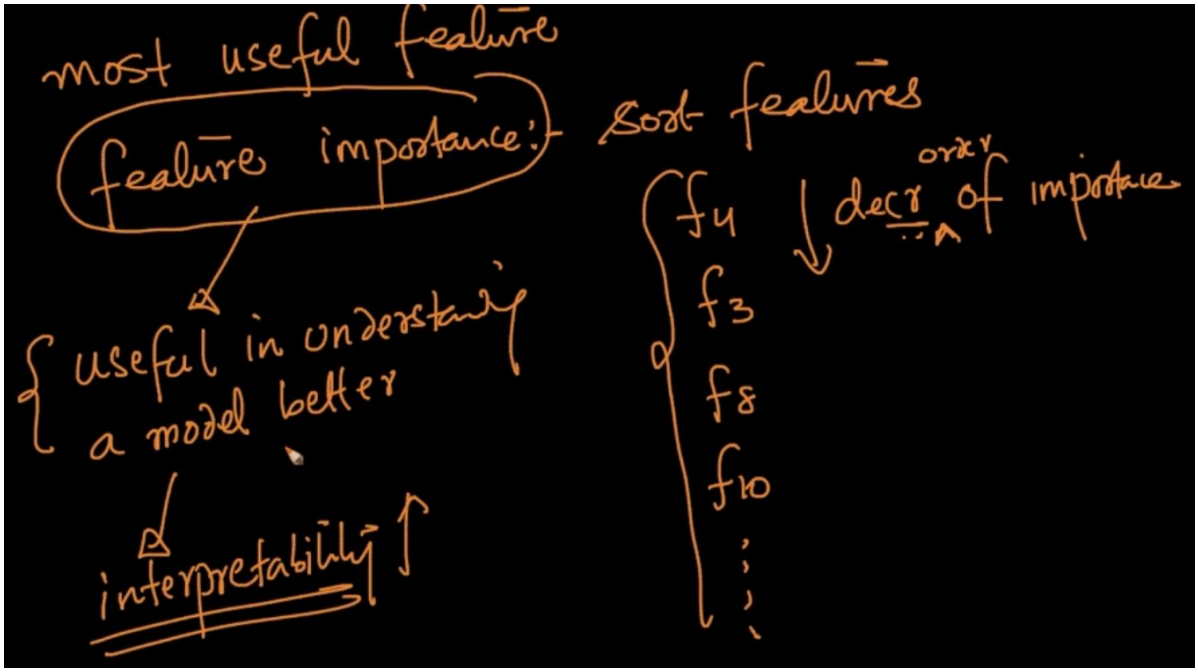


If we are running k-NN with 7-NN and we are getting $y_q = 1$ then we can interpret the model because our NN have medical tests as dimensions but d is low and our model is giving reasoning that all the neighbors have very similar results on tests and they all have cancer i.e $y_q = 1$. So, the doctor can understand our model because d is also small .

FEATURE IMPORTANCE AND FORWARD FEATURE SELECTION



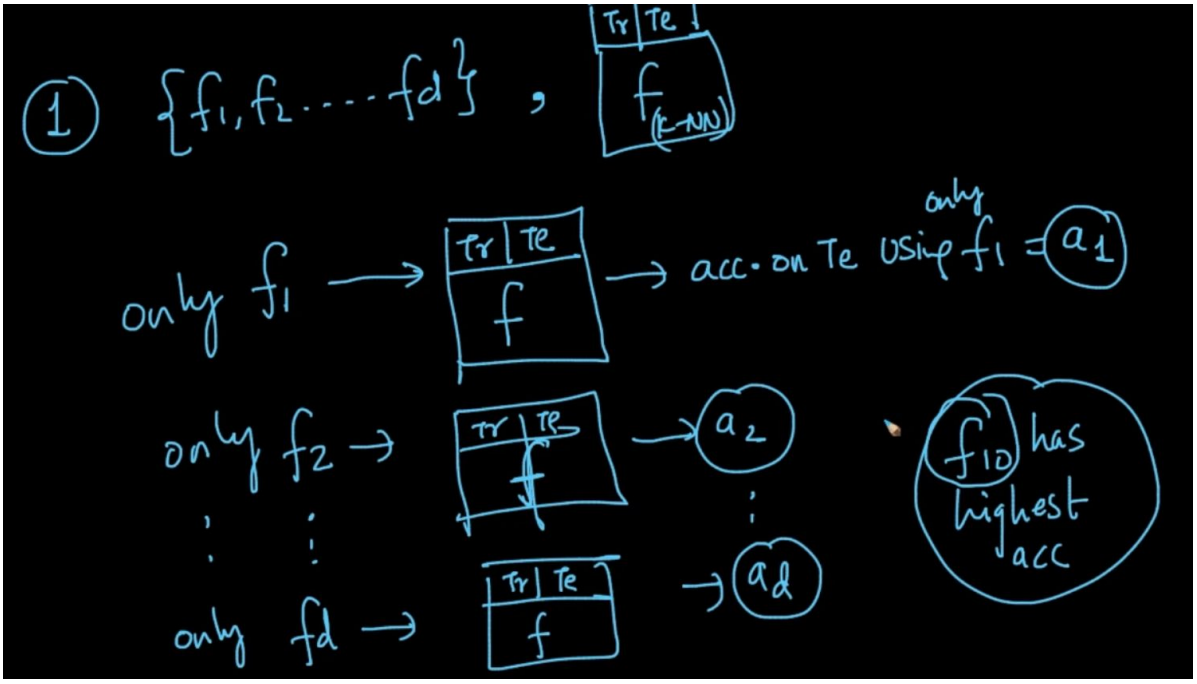
For predicting heights we need to get the most important features



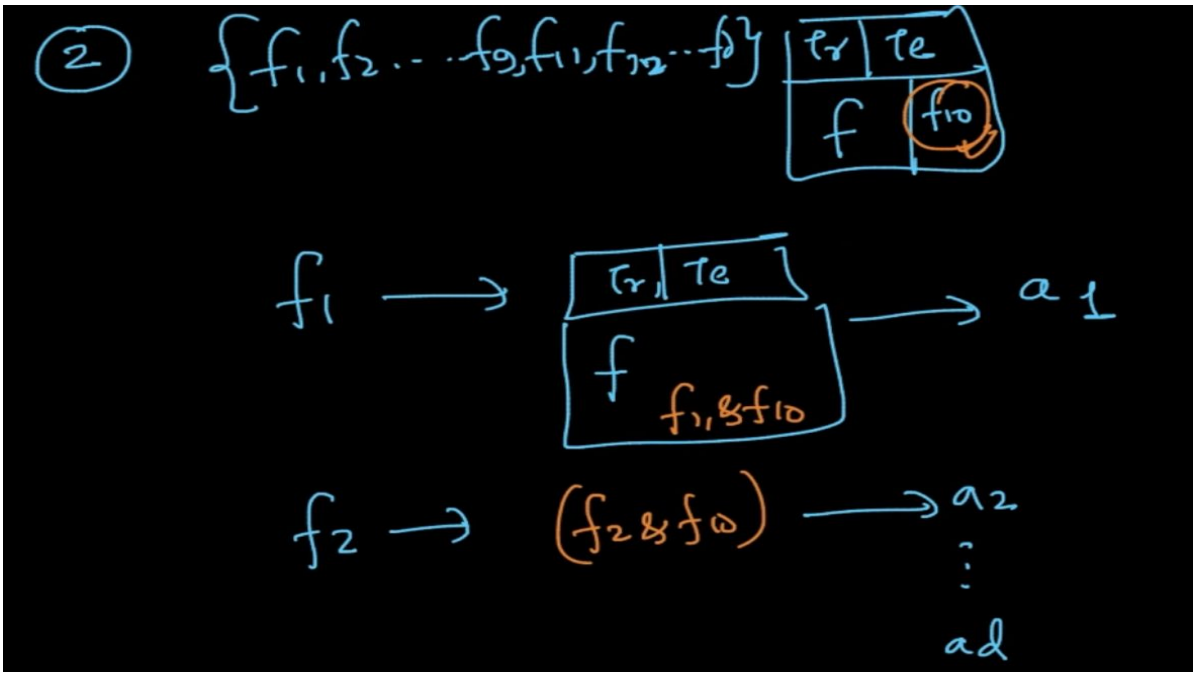
If we know the most important features it means that we can understand model better which means more interpretability

So a method called forward feature selection is used to pick the top 100 dimensions

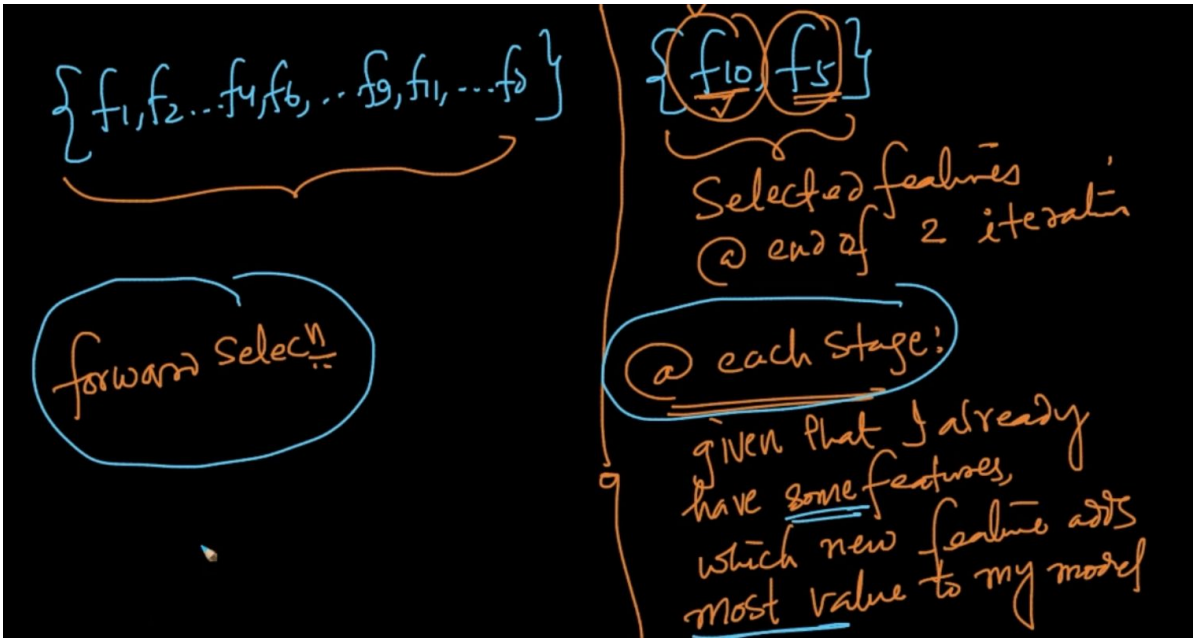
Note - PCA, tsne are dimensionality reduction techniques but they maintain the distance instead of giving us the best features



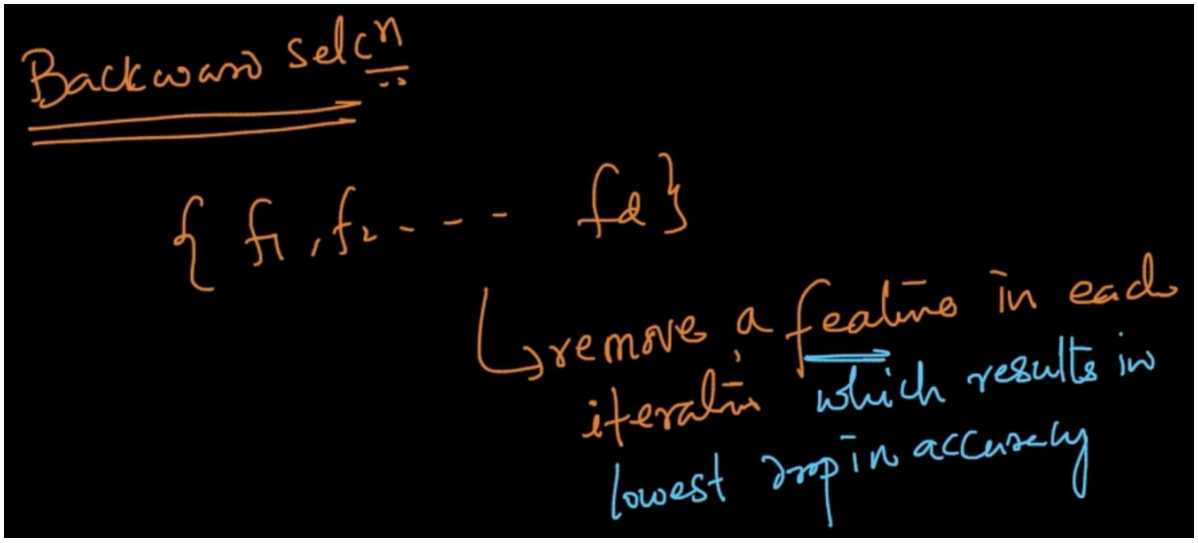
We've d-features . We take each feature till we test our model on every feature with their corresponding Train and test data and check accuracy. Suppose feature f_{10} has the highest accuracy then we'll select that



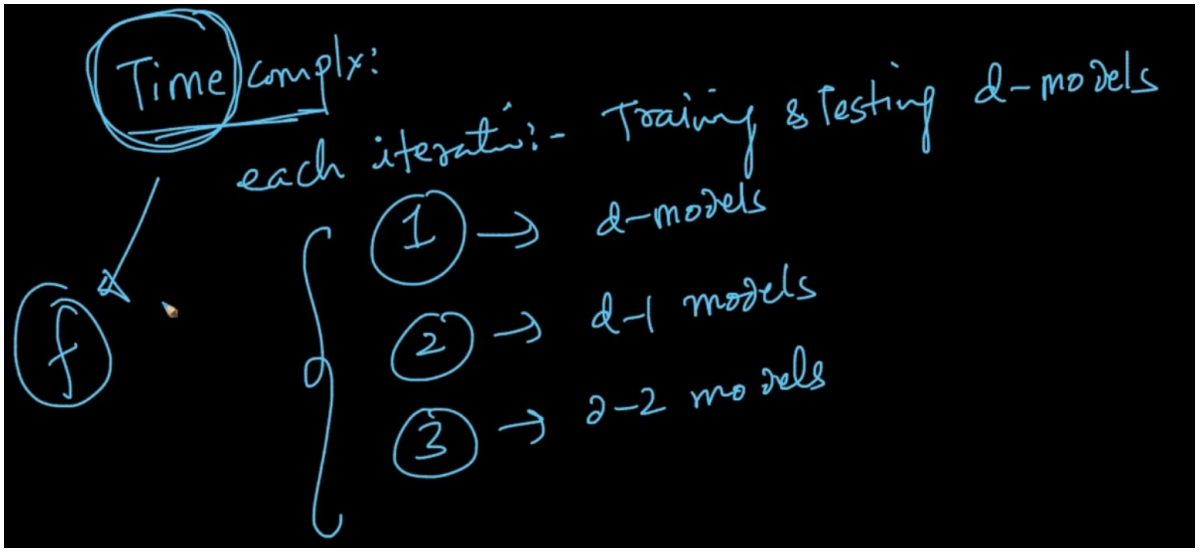
Feature f_{10} is the most important feature. Now, we'll select it and along with it train our models and get accuracies taking both the features . Now let's say f_5 has got the highest accuracy. Therefore our features f_5 ,f_{10} are our selected features



So we keep repeating this till we get our number of desired features. Let's say we wanted our top 100 features from 1000 features

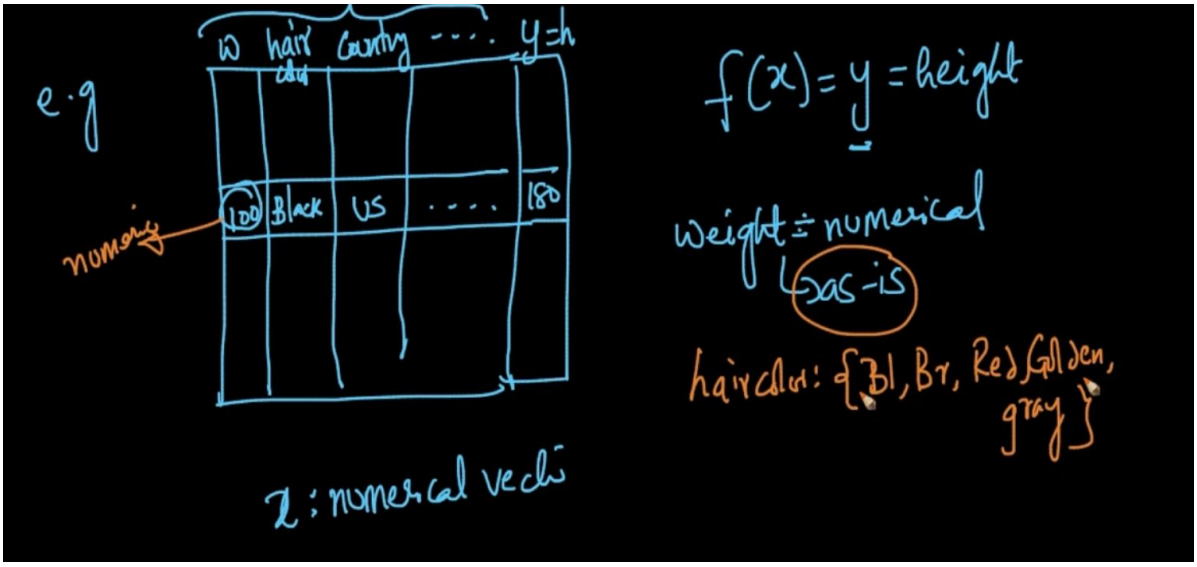


In Backward selection we remove a feature in each iteration which results in lowest drop of accuracy

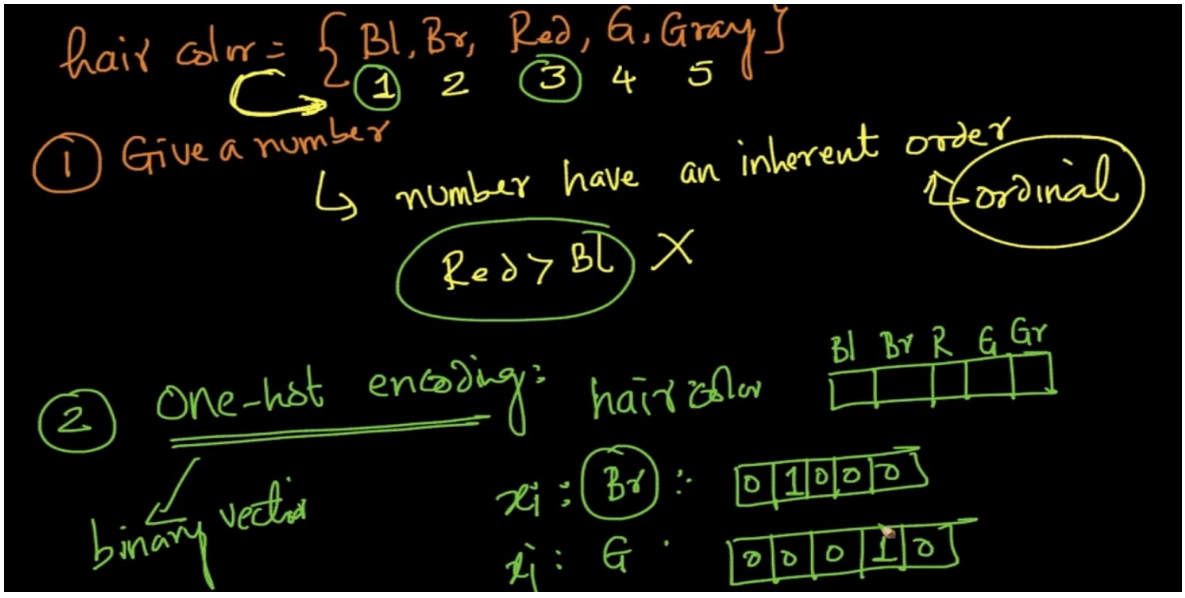


Time complexity is very large obviously but it can be done in any model or algorithm

HANDLING CATEGORICAL AND NUMERICAL VALUES

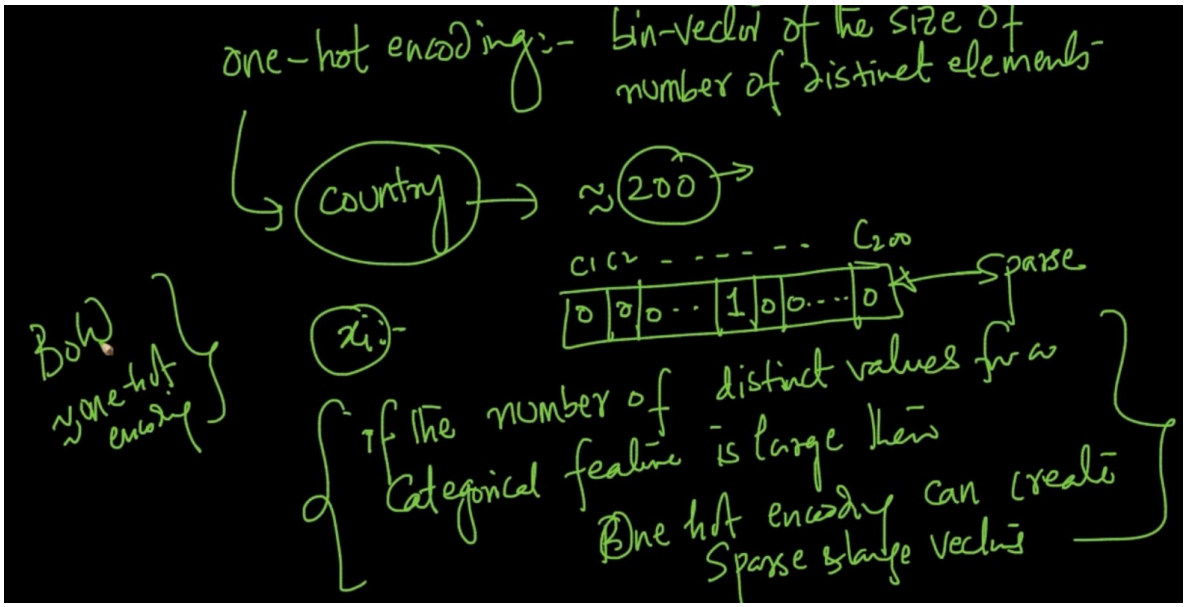


We are trying to predict heights of people and it's features X are {Weight. Hair color, country} . So, weight is numeric but haircolor isn't : {Black, Brown, Red,....} it's a categorical variable but we want our X should be a numerical vector so hair color should be converted to numeric instead of categorical.

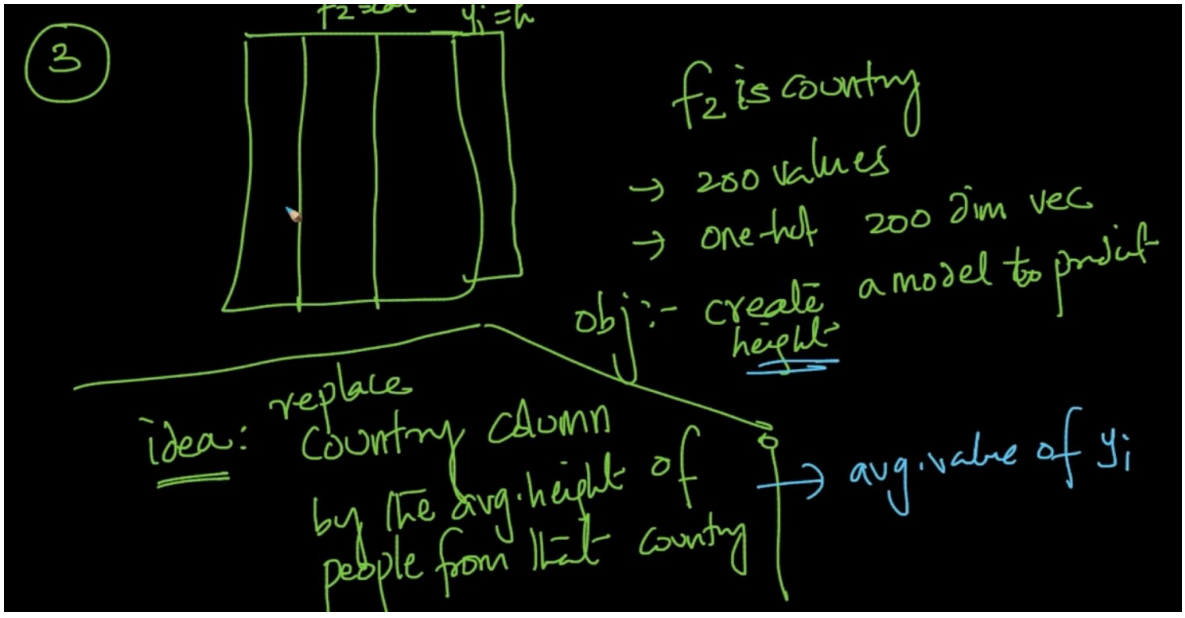


1) We are giving them a number but it'll create an inherent order i.e if Red => 3 and Black => 1 which will indicate Red > Bl

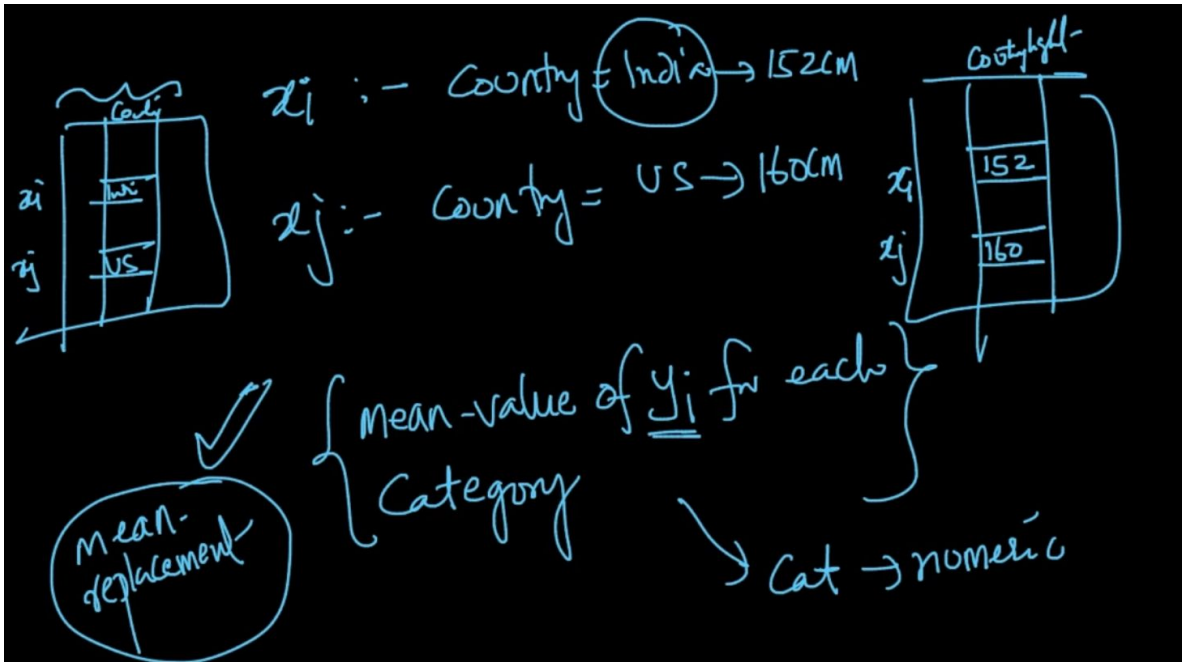
2) So we use one hot coding in which we create a d dimensional vector where d is number of distinct categories like color of hairs in haircolor feature in which we feed binary values Here, no inherent order like Red > Black



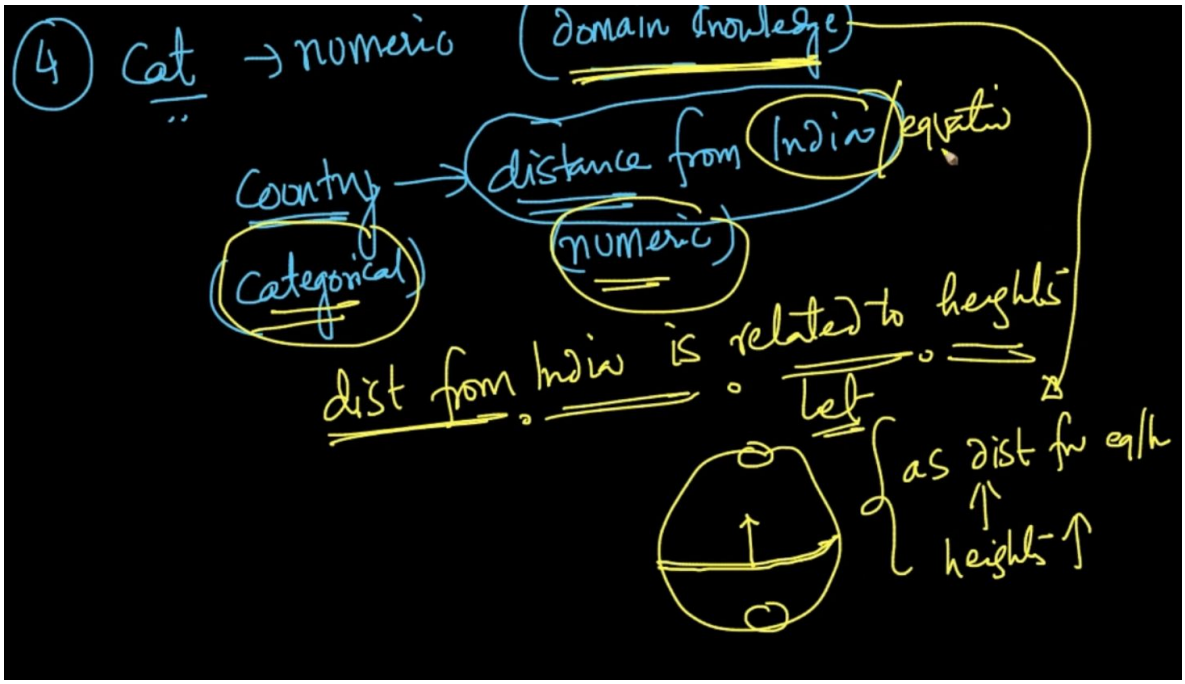
But there's a problem with one hot encoding i.e if the number of distinct values or d is large for ex : like countries where countries = 200 sparse & large vectors can be created



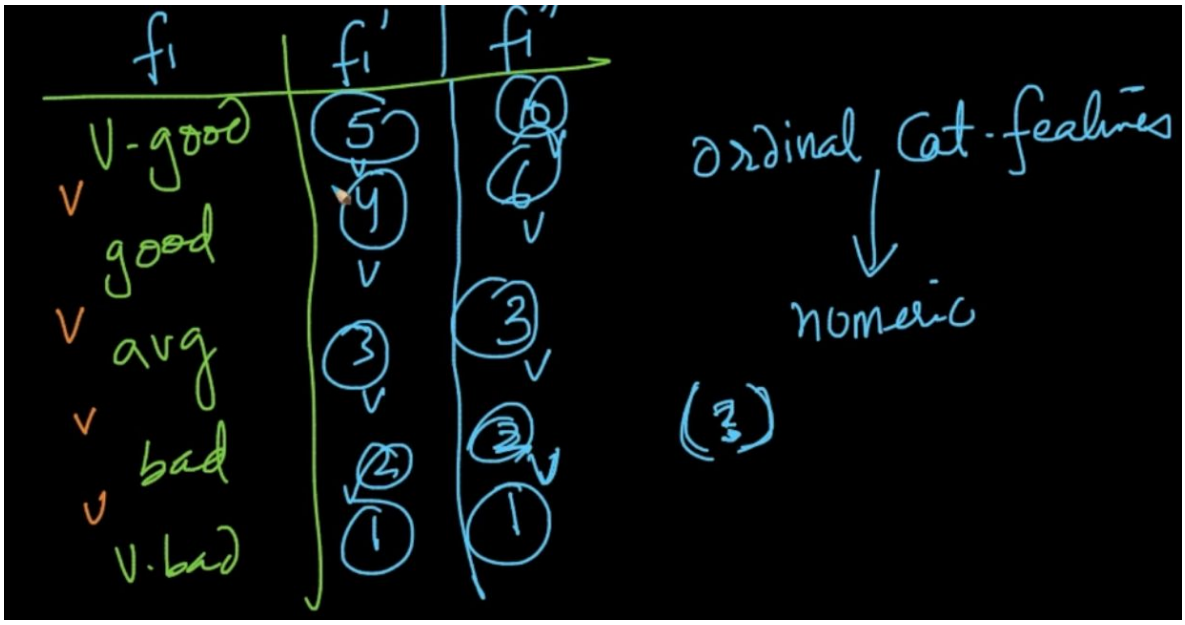
Suppose we've country feature . We'll replace it with average country height but note that it is very objective specific i.e related on predicted values



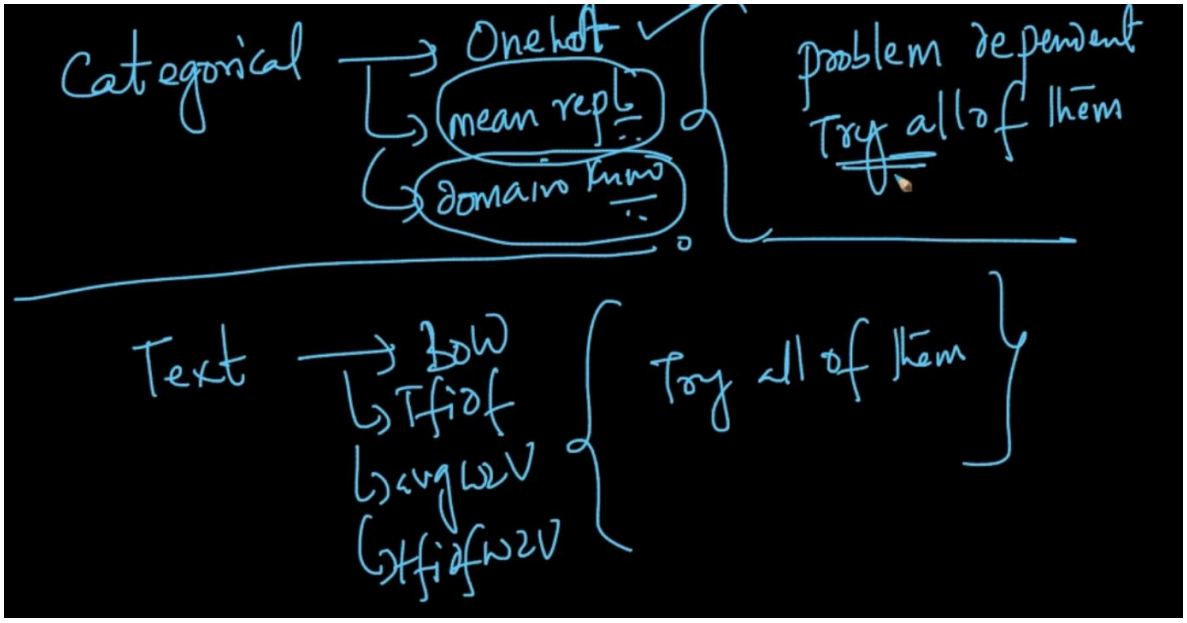
Replacing categorical variable Country with average/mean of y_i for each category i.e Mean (y_i) for India is 152 cm so Country Height is 152 for Indians



Domain knowledge : If an anthropologist says that distance from India is related to heights as distance from India ↑ heights ↑. So this is another method of converting categorical to numerical

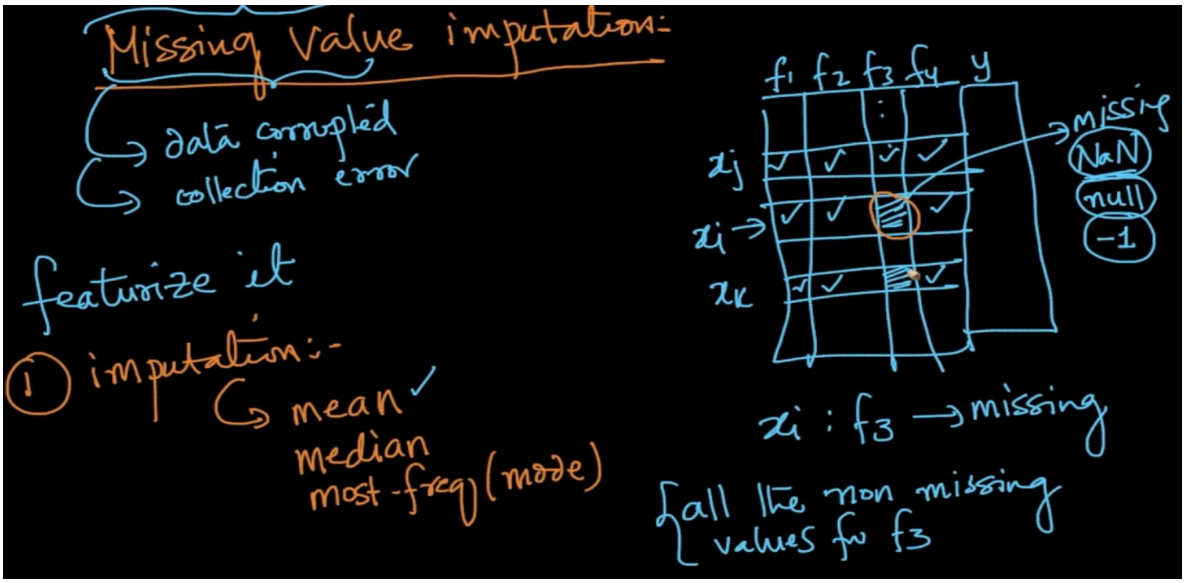


Ordinal categorical features like {V.Good, Good, avg, bad, v. bad} they can be converted into numeric by giving them numeric value based on their severity like 5 for V.Good and 1 for V.Bad but we there's no proper answer on how to give numeric values to them we can also give 10 => V.Good and 1 => V.Bad



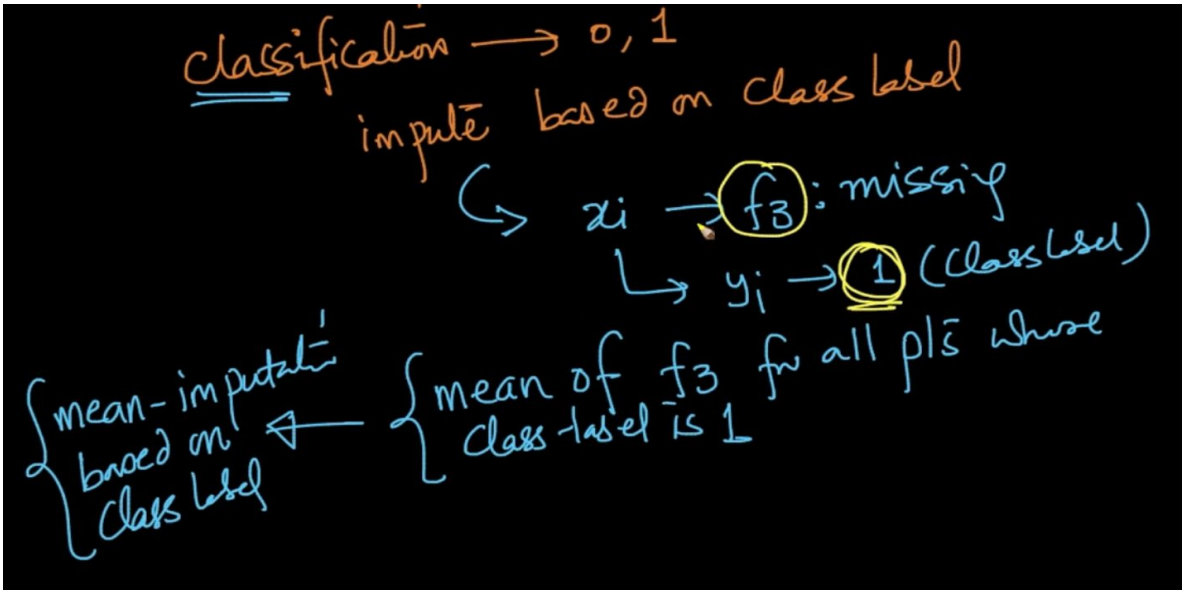
So these are the methods for dealing with Categorical but there's no better because they are very problem dependent. So what do you do? You try all of them

HANDLING MISSING VALUES BY IMPUTATION

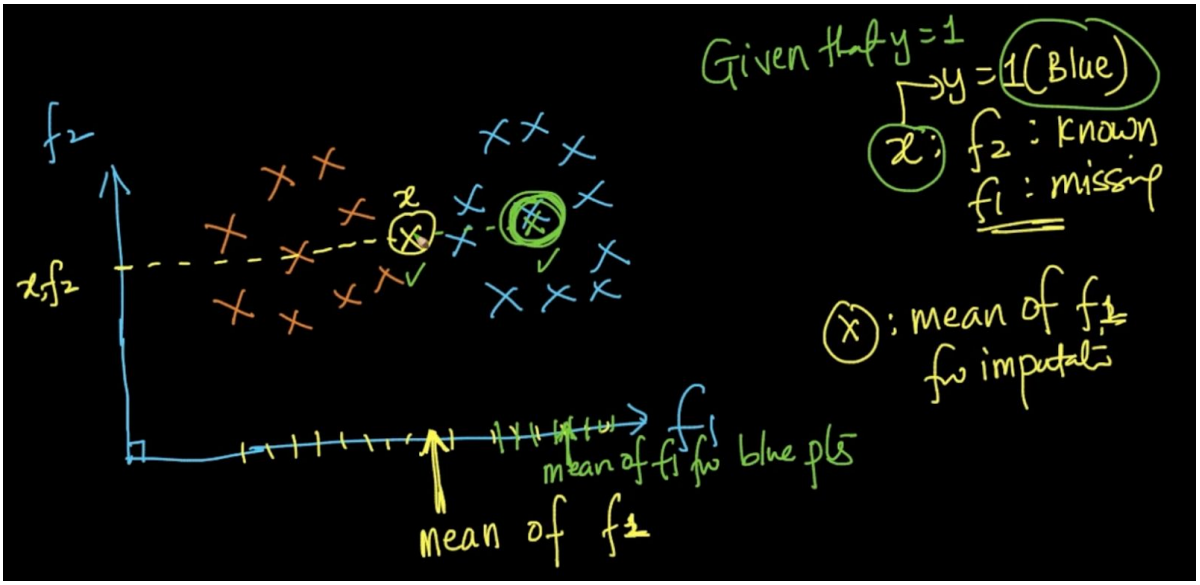


Sometimes, in some features values can be missing or NaN. To deal with it we do imputation with the methods listed here

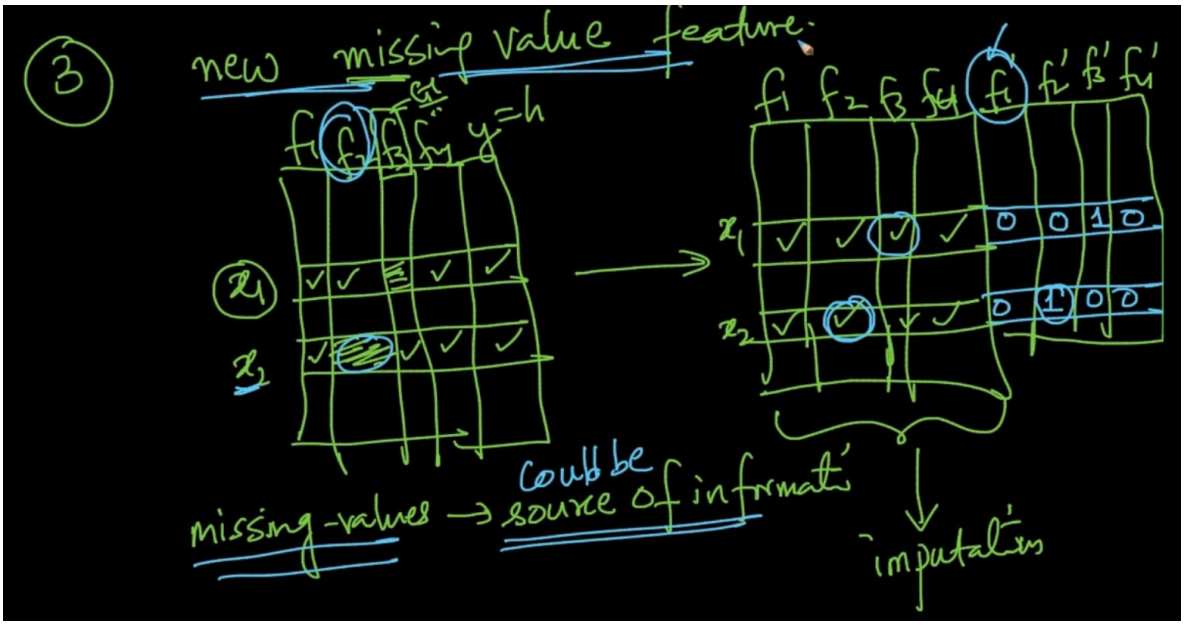
In our missing feature f_3 of x_i we can put the mean of all non missing values for f_3



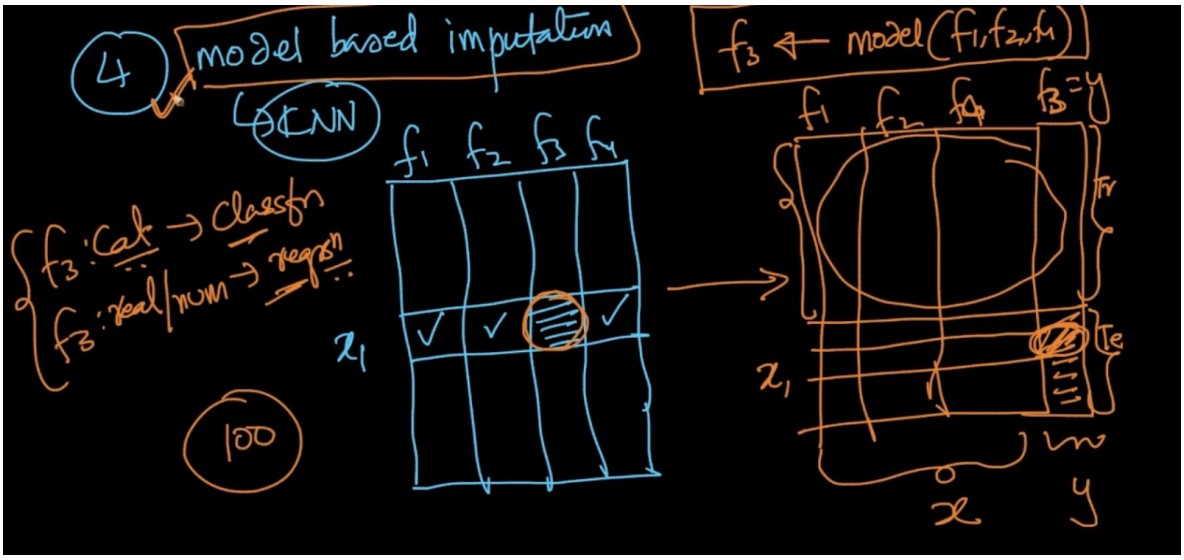
In classification , for f_3 of x_i whose class label $y_i = 1$ instead of taking mean of all the pts of f_3 for missing values we can just get the Mean of f_3 for points where y = 1



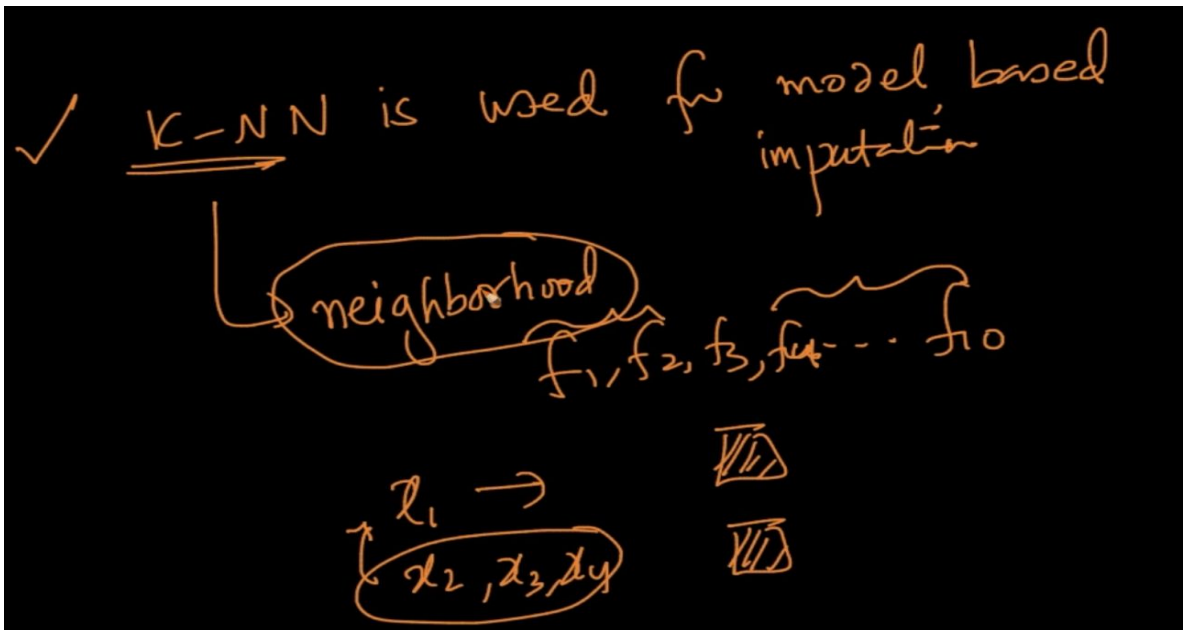
Here we want x whose feature f_1 is missing but y = 1. If we are taking the mean of all f_1 it'll lie in the yellow region but if we take the mean of f_1 where y = 1 then it'll be at a much better place as it seems because it's y = 1 as well



We add new features by adding a binary vector which is giving us info about the missing feature . Sometimes missing values could be the source of info

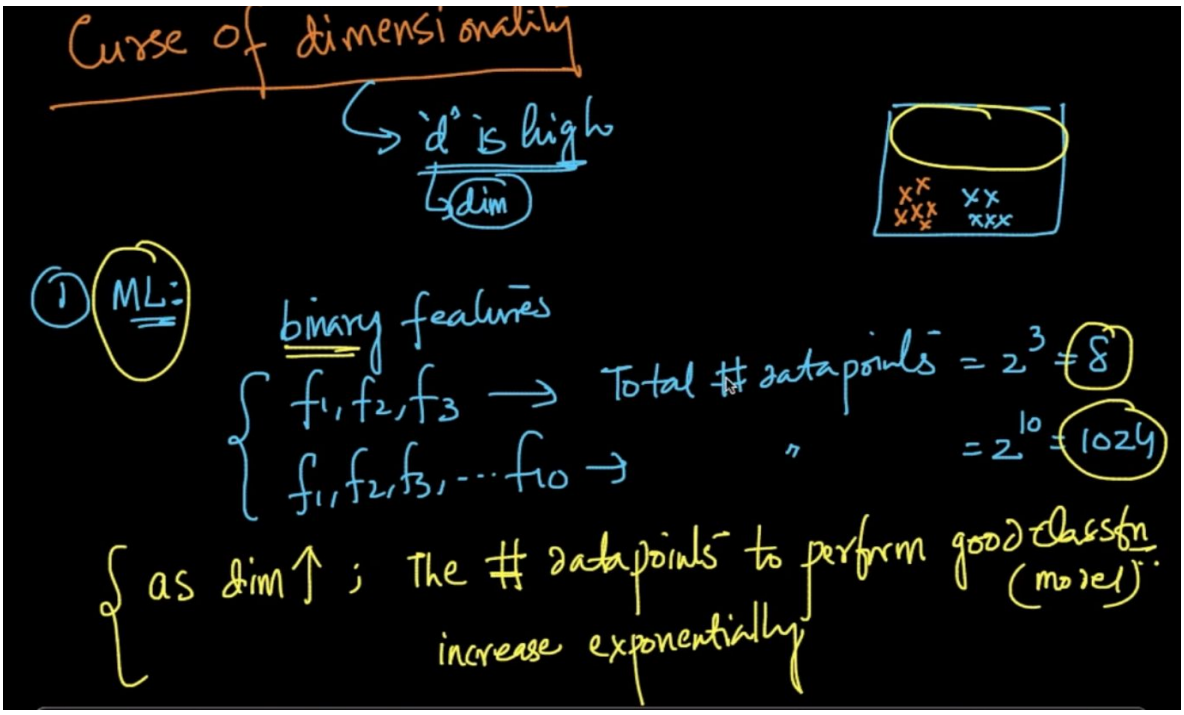


In model based imputation, e.g if f_3 has missing values . Then we can make it as a class label and use the other features to predict them. As you can see f_3 has been used as class label and in that all the missing values are in Test so we can predict them based on their Train data which has values of f_3

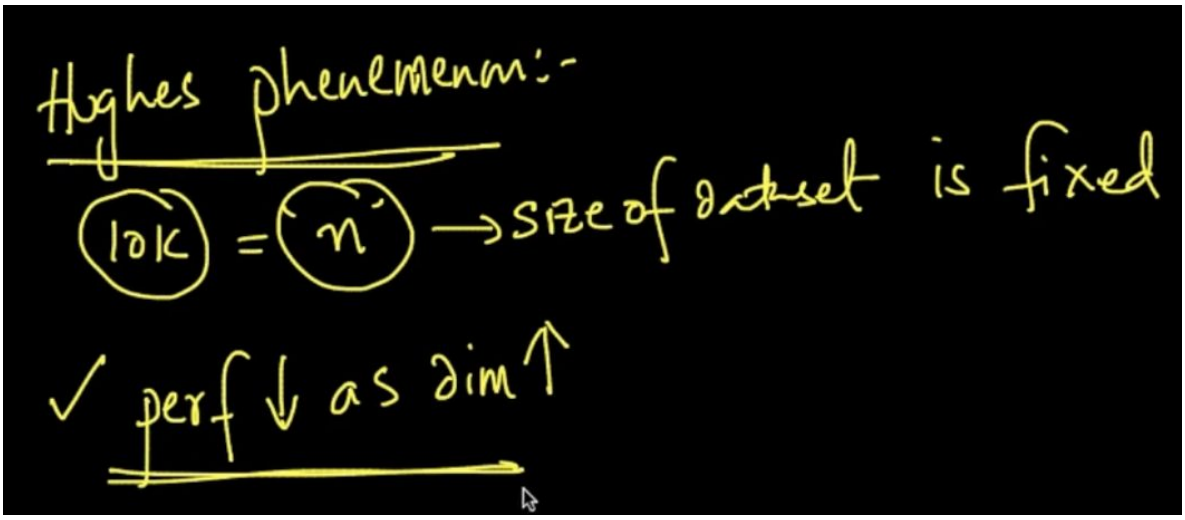


k-NN is used for model based imputation.Data point x_1 has missing value in feature f_3 But x_2, x_3, x_4 are very similar to x_1 in other features so we can predict the missing value of f_3 as well

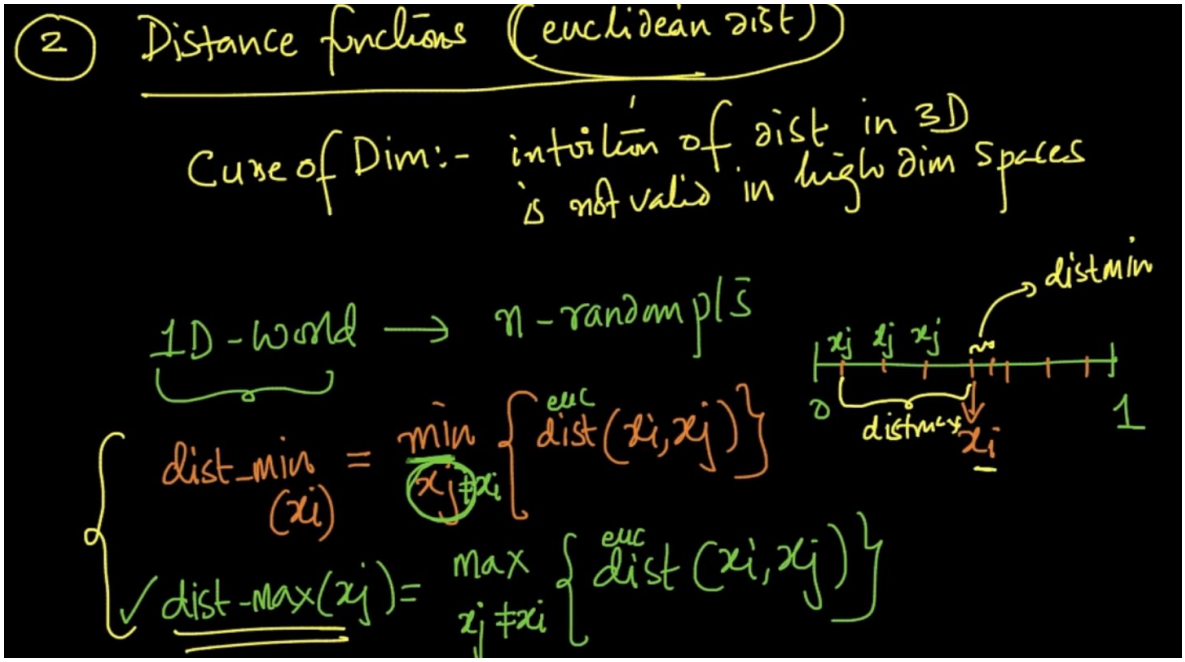
CURSE OF DIMENSIONALITY



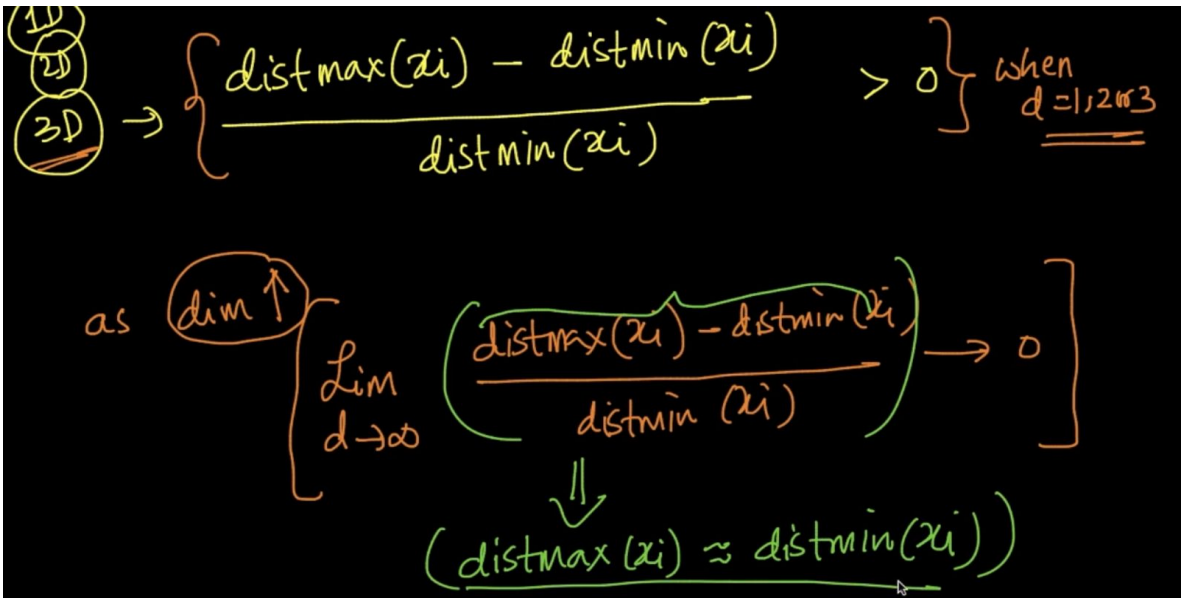
Suppose we've three binary features/dimensions it'll require 2^3 = 8 datapoints and if 10 features $2^{10} = 1024$ so as dimensions increase the number of datapoints to perform good classification increases exponentially



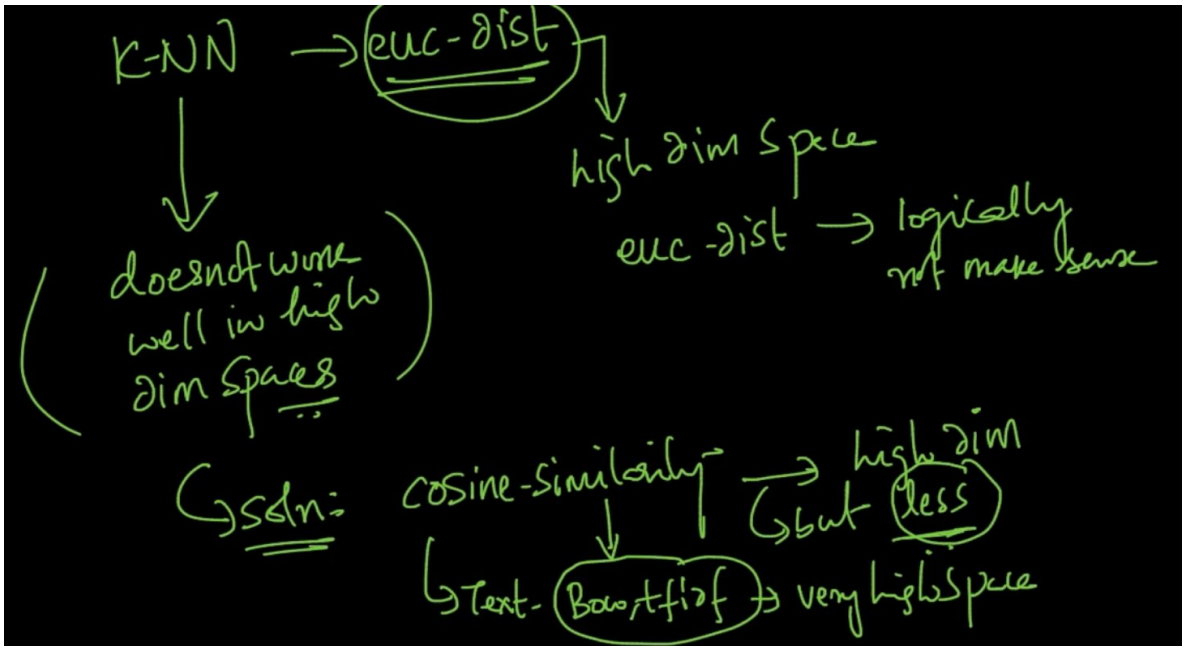
Hughes phenomenon : Our performances ↓ as dimensions ↑ if your size of dataset is fixed



Here we've defined two terms 1) Dist_min (x_i) which gives us the minimum distant point or closest point from all points from x_i.2) Dist_max(x_i) which gives us the maximum distant point or farthest point from all points from x_i



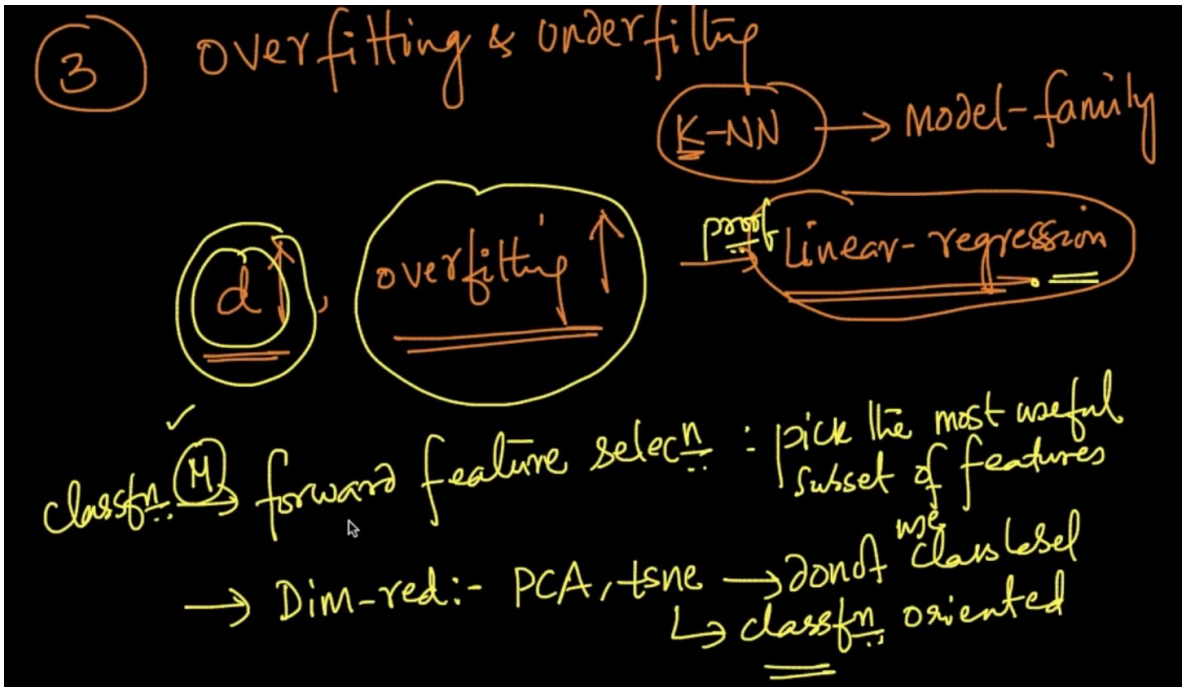
As dimensionality d increases the above ration tends to become 0 which means Distmax ≈ Distmin which is an alarming thing as it means every pair of points are equidistant from each other



As distances don't make sense in high dimensions as Distmax ≈ Distmin ,k-NN don't work together very well so to tackle it we use Cosine similarity as it doesn't get affected by Curse of dimensionality and that's why we use it for text as iy has very high dimensions,

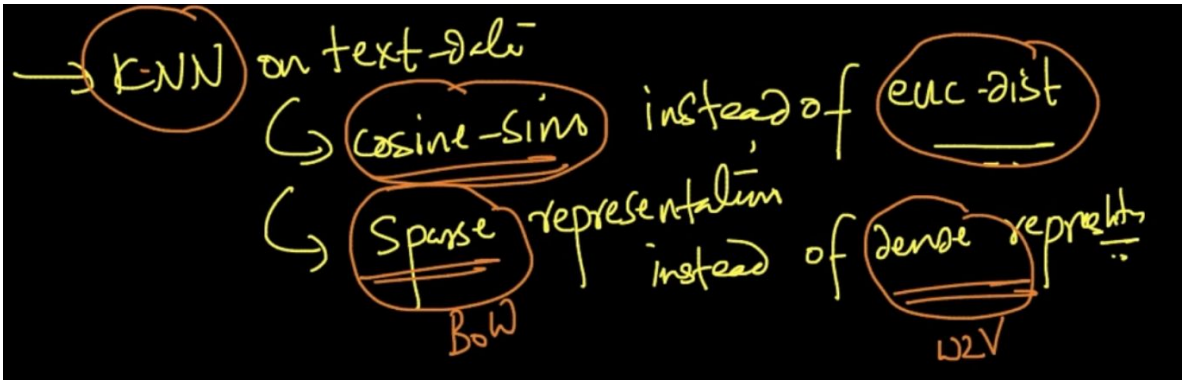
BOW, tf, idf are used since they are sparse

As dimensions become high & sparse the data get's non uniform as 90% of data is 0 so impact of Curse of dimensions become lower as whole intuition of it is dependent on random spread of data



As dimensionality increases overfitting also increases so we use forward feaure selection mostly to select the most important features hereby reducing dimensionality

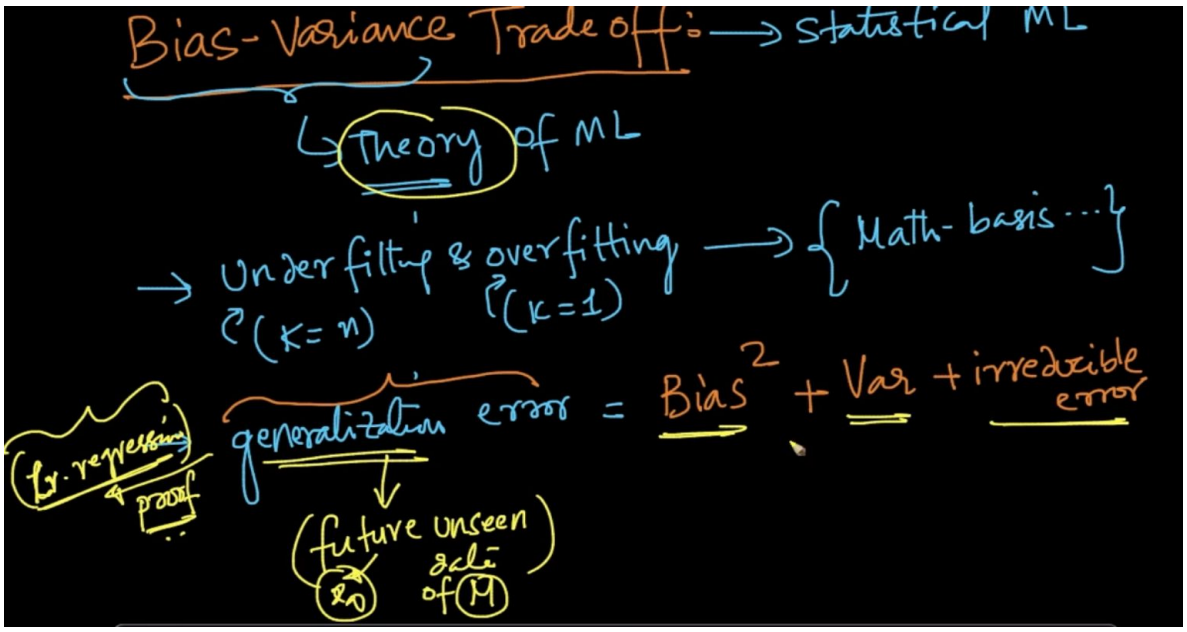
Dimensionality reduction : PCA, t-SNE doesn't use class label in reducing the dimensions it is only related to maintain the spread /distance of points so not used for classification



Techniques mentioned above are used as impact of Curse of dimensionality is lower on them

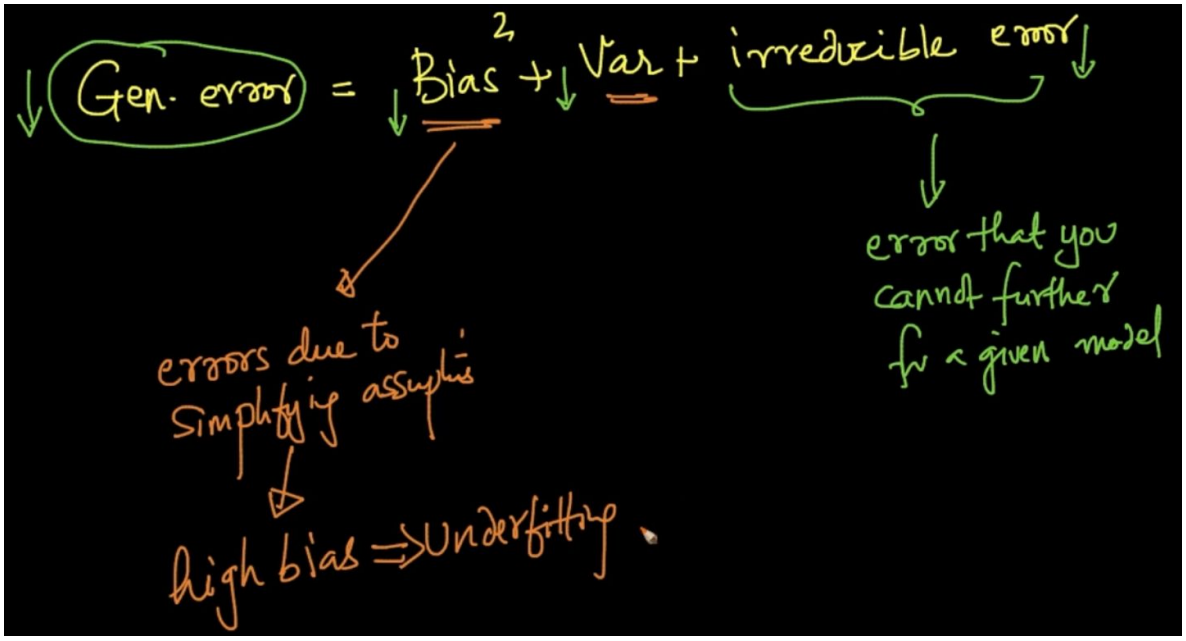
[Curse of dimensionality - Wikipedia]

BIAS VARIANCE TRADEOFF



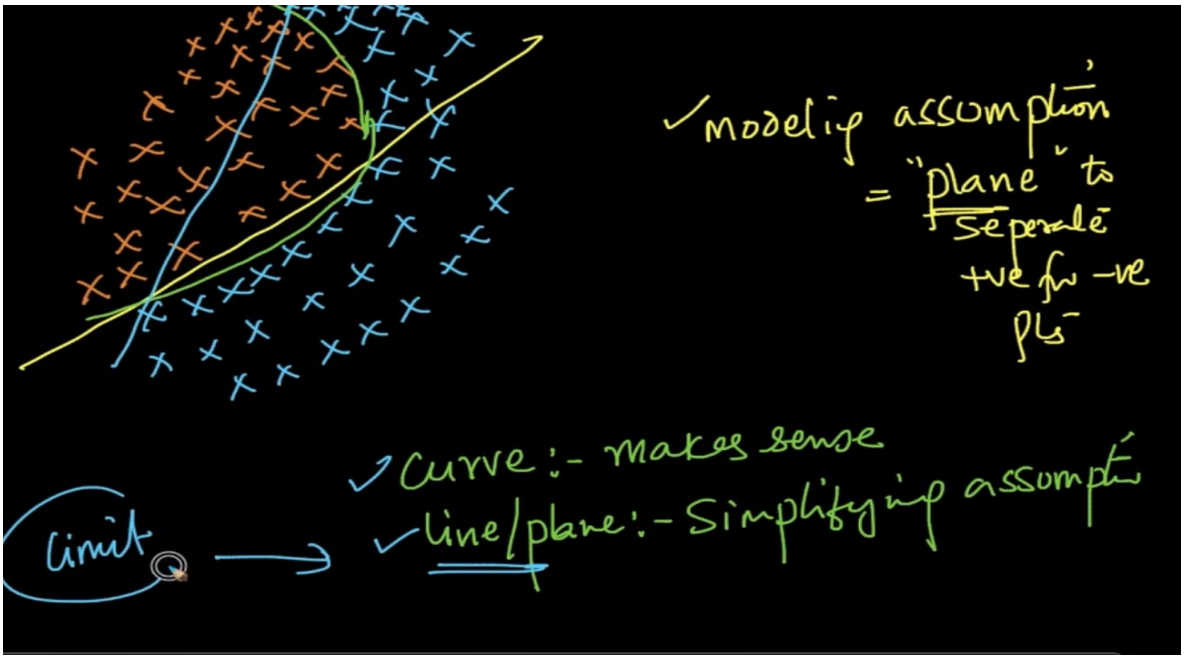
We understood the concept of underfitting & overfitting but now we are doing a Mathematical analysis of it. The answer of how to do it is Bias-Variance Tradeoff.

Generalization error = $Bias^2$ + *Variance + irreducible error* . It is that error which occurs on



unseen data

BIAS

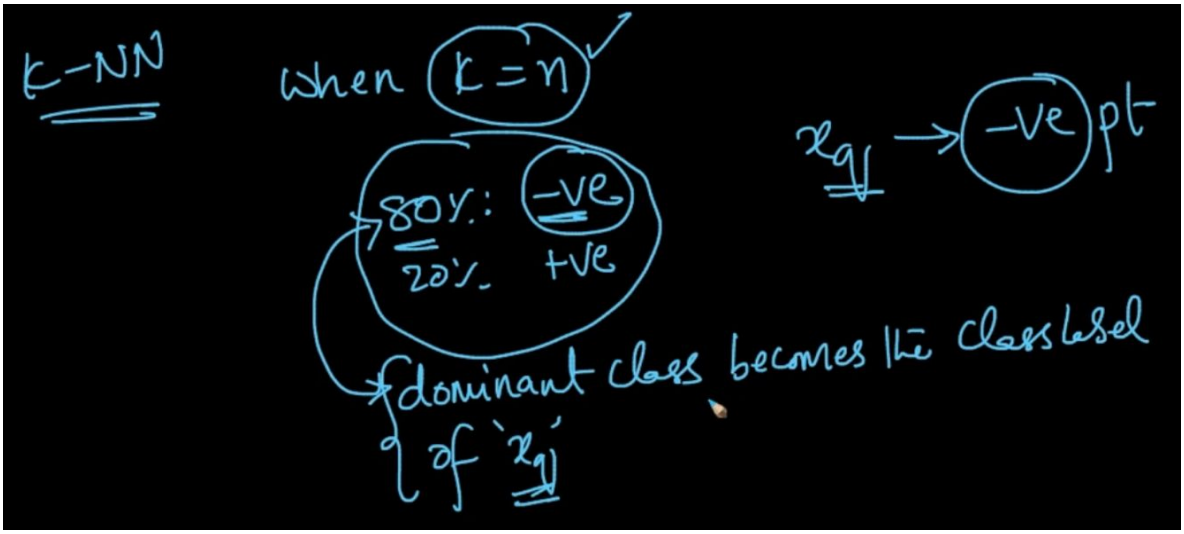


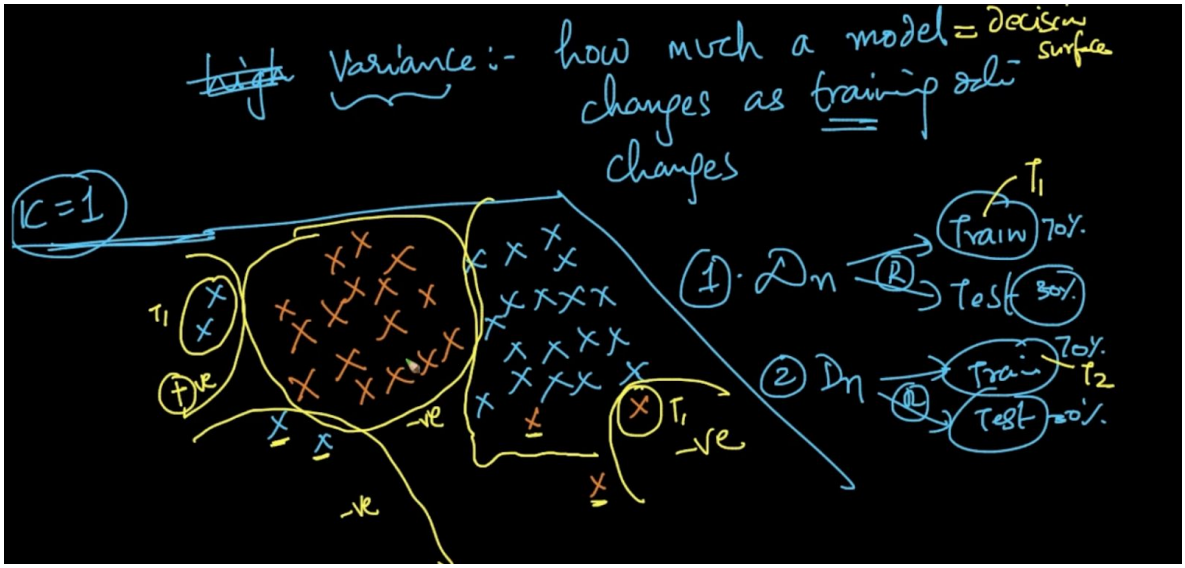
Bias error is error due to simplifying assumption .Suppose we are separating that data with a plane / line instead of curve (which can separate the data better) we are simplifying it and it leads to underfitting

In k-NN where k = n , if we make the dominant class as class label of every new query x_q we are doing a simplifying assumption which leads to high bias => underfitting

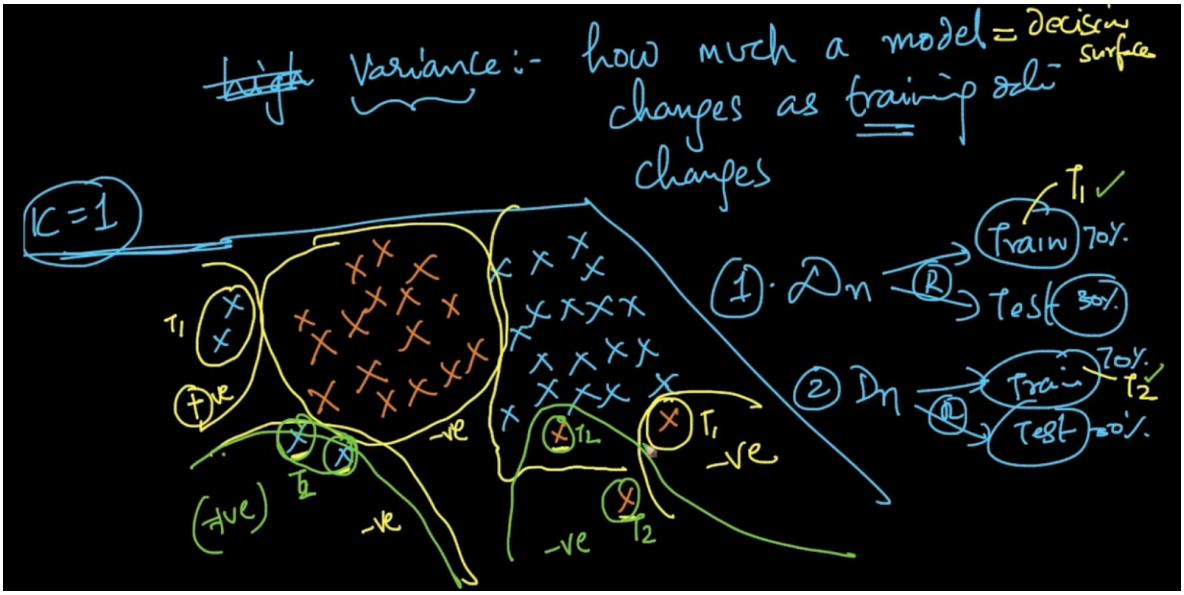
So whenever we've high bias we are underfitting as we are generalizing assumptions

VARIANCE

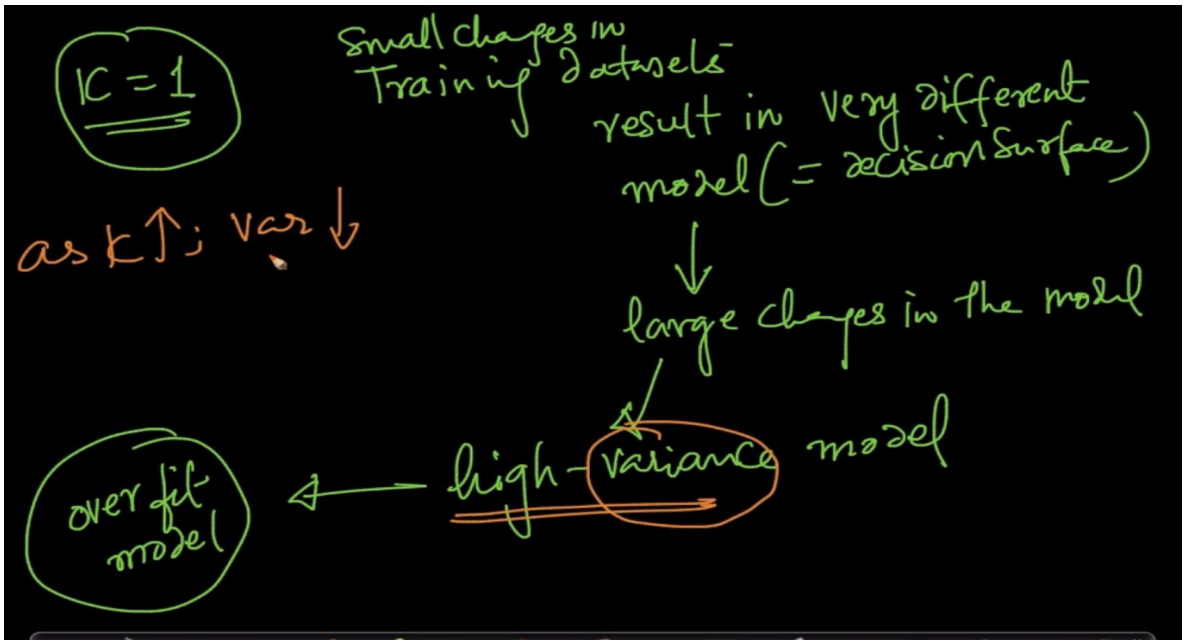




For k = 1 , this is the decision surface with class labels on first random Data splitting where Train data T_1

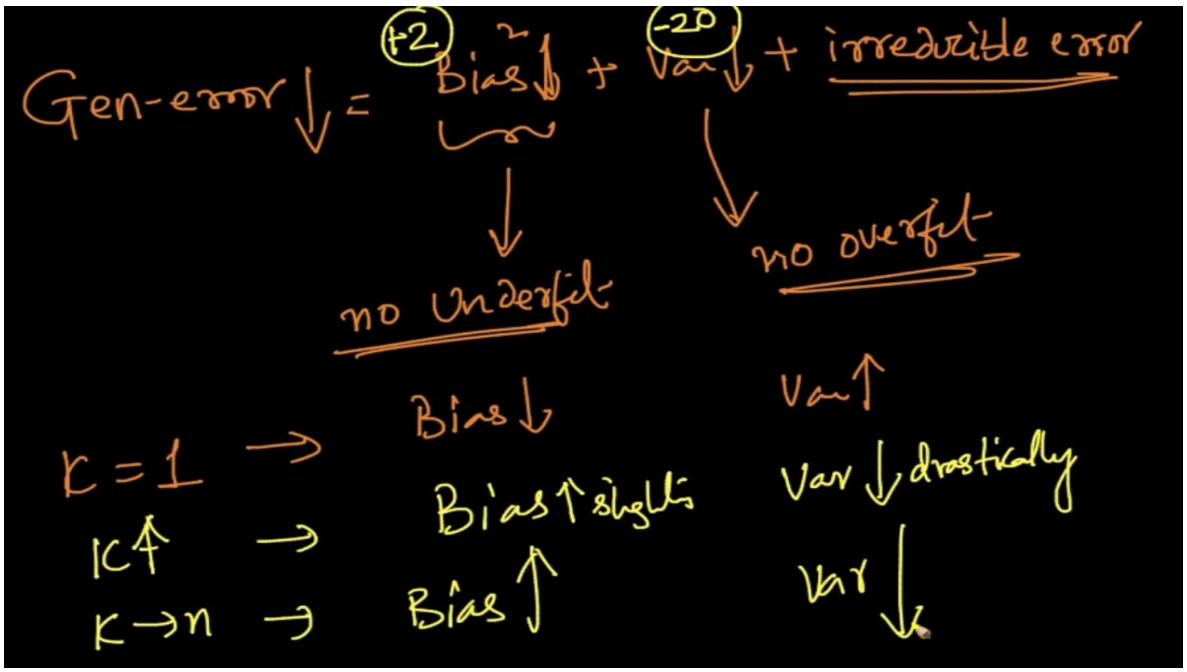


As we do the 2nd Random split where Train data T_2 .Here, the green surface where it was first -ve in T_1 becomes +ve in T_2 . So, it's a variance

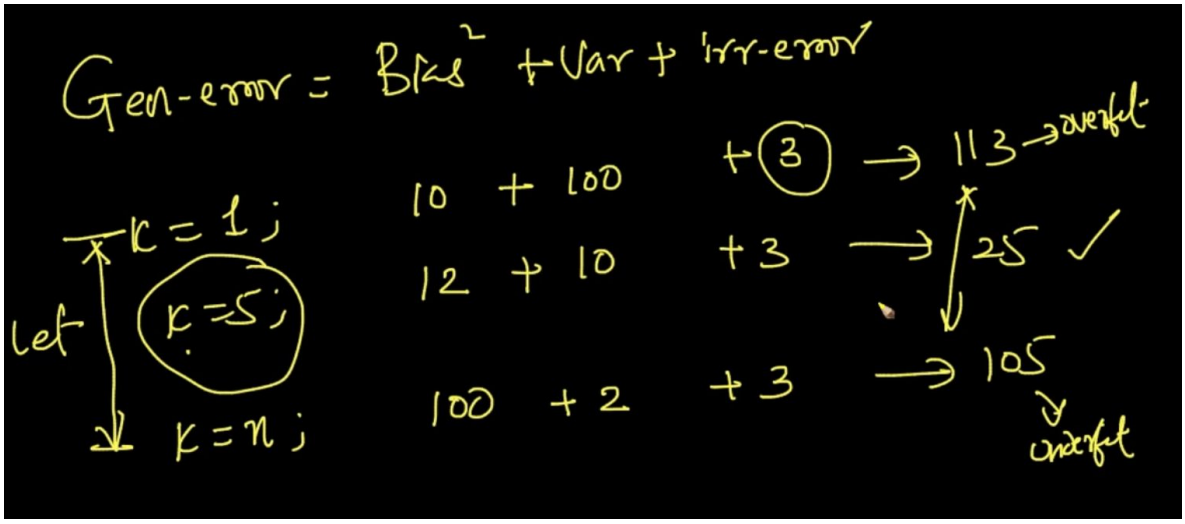


Small change in Training dataset result in very different model (= decision surface)

This change in decision surface/model is large which can lead to high variance which leads to overfit. As k ↑ ; var ↓

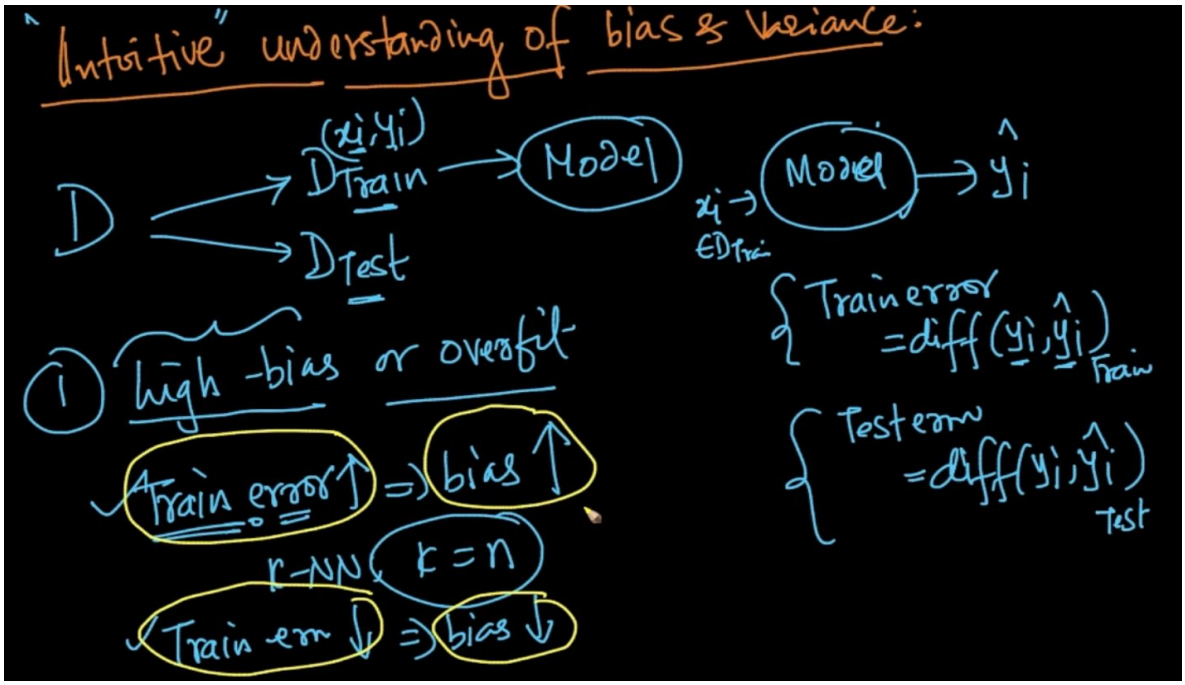


So we want to balance bias error and variance error as seen that they vary on the values of their ‘k’



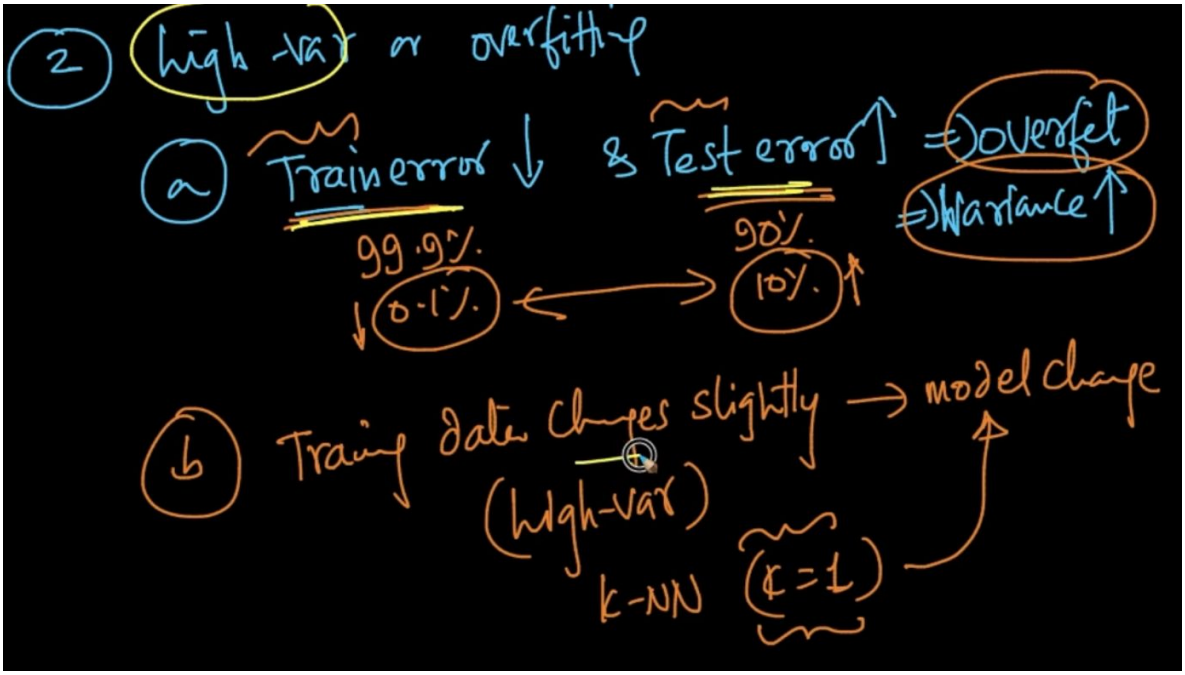
Here at k = 5 , Generalization error is the lowest so we should be looking for finding the right balance

INTUITIVE UNDERSTANDING OF BIAS AND VARIANCE



Our model is given x_i and model gives $\hat{y}_i$ and $\hat{y}_i$ difference is tested with y_i which gives Train error and Test error.

High bias/ Underfit : A way to determine it is if Train error is high then bias is also high



High Variance / Overfit : if our Train error is low but Test error is high then we say that our model is overfit or has high variance