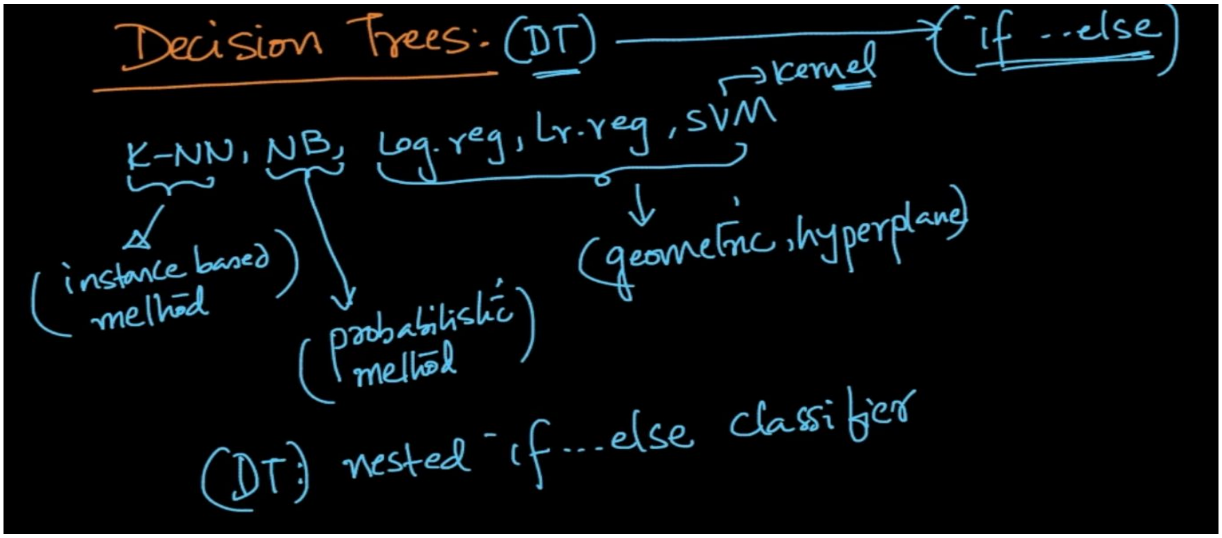
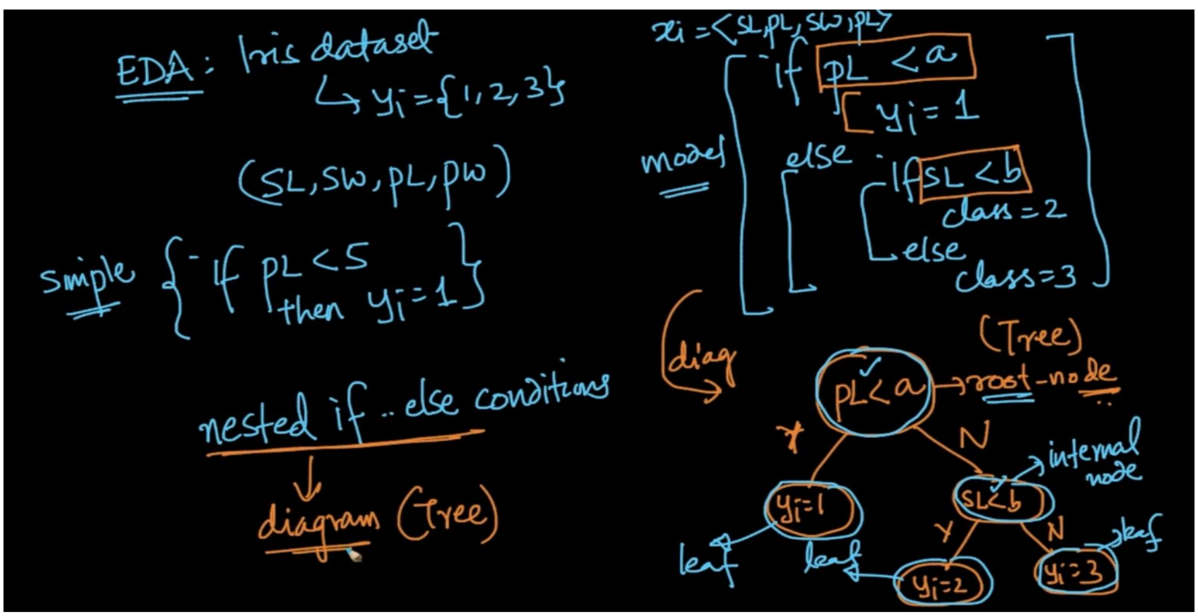
# DECISION TREE

## GEOMETRIC INTUITION

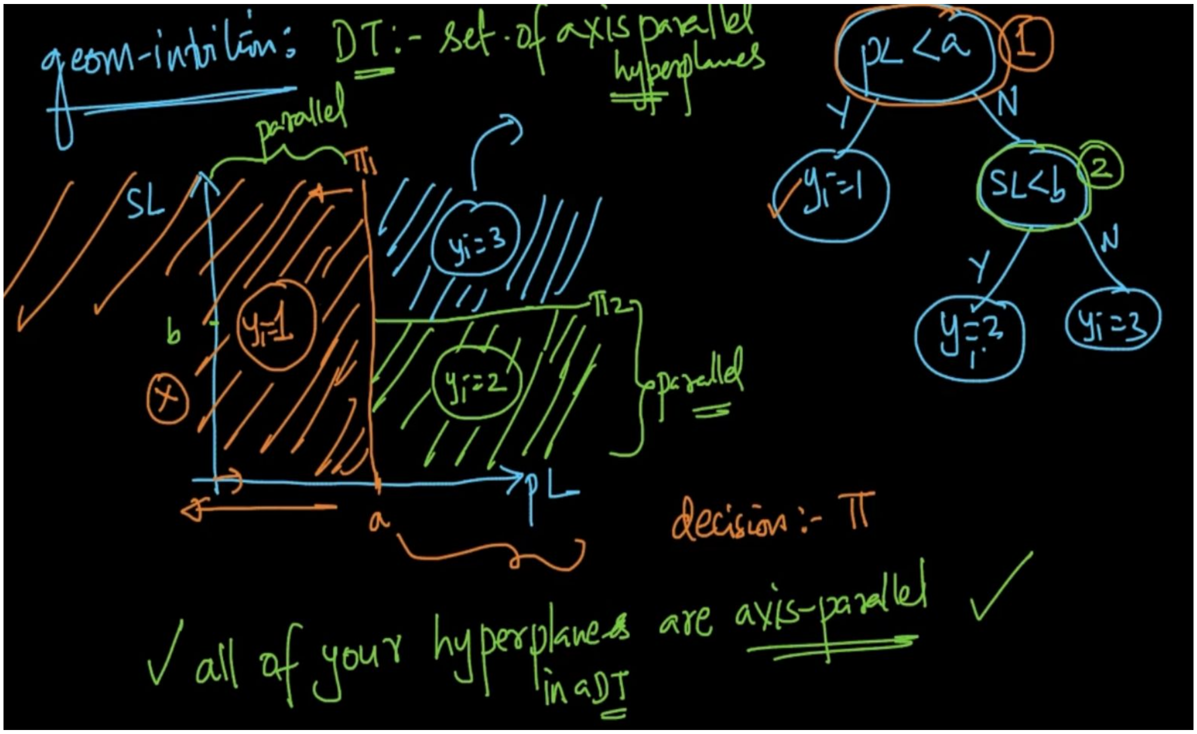



We'd discussed about which type of methods the classifiers have .

Decision Tree, it's a nested if …. Else classifier simply

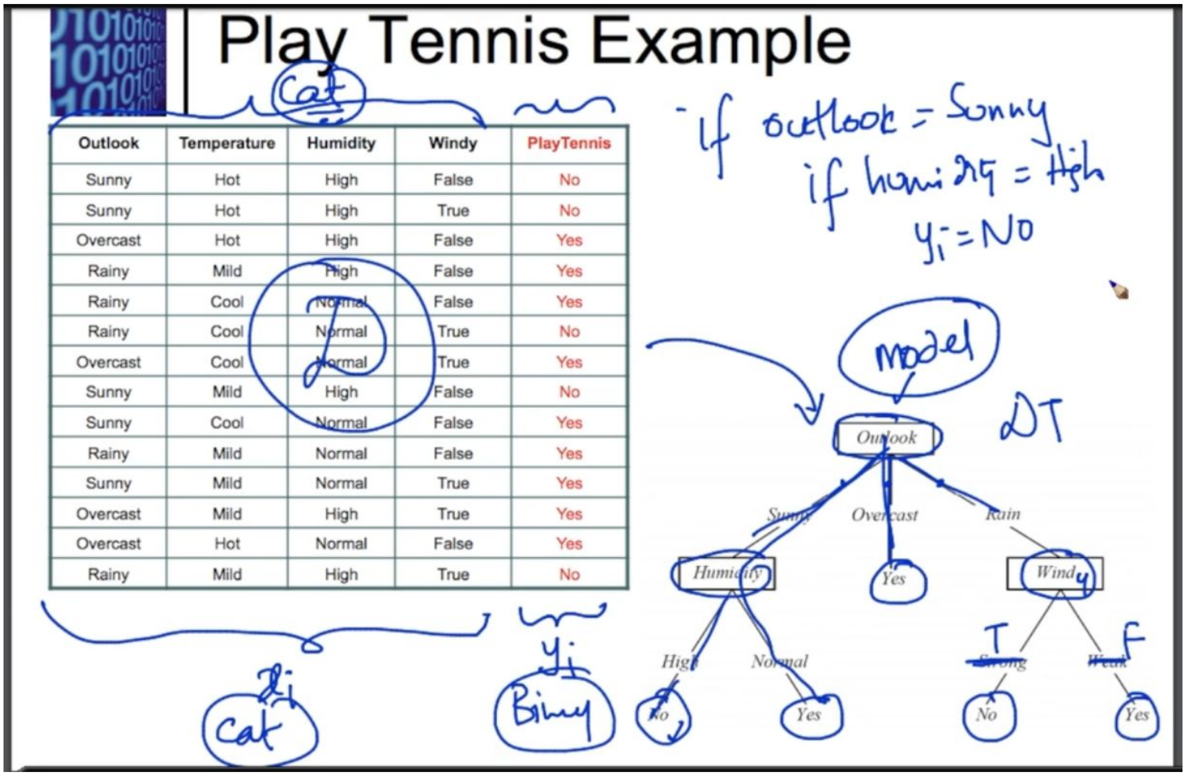



We've a simple nested if-else model for Iris dataset. We are taking the first condition as a root node and from that start building our Decision Tree which is basically diagramatic representation of Nested if-else model. In DT all the non leaf nodes take a decision and leaf nodes tells us about the class label



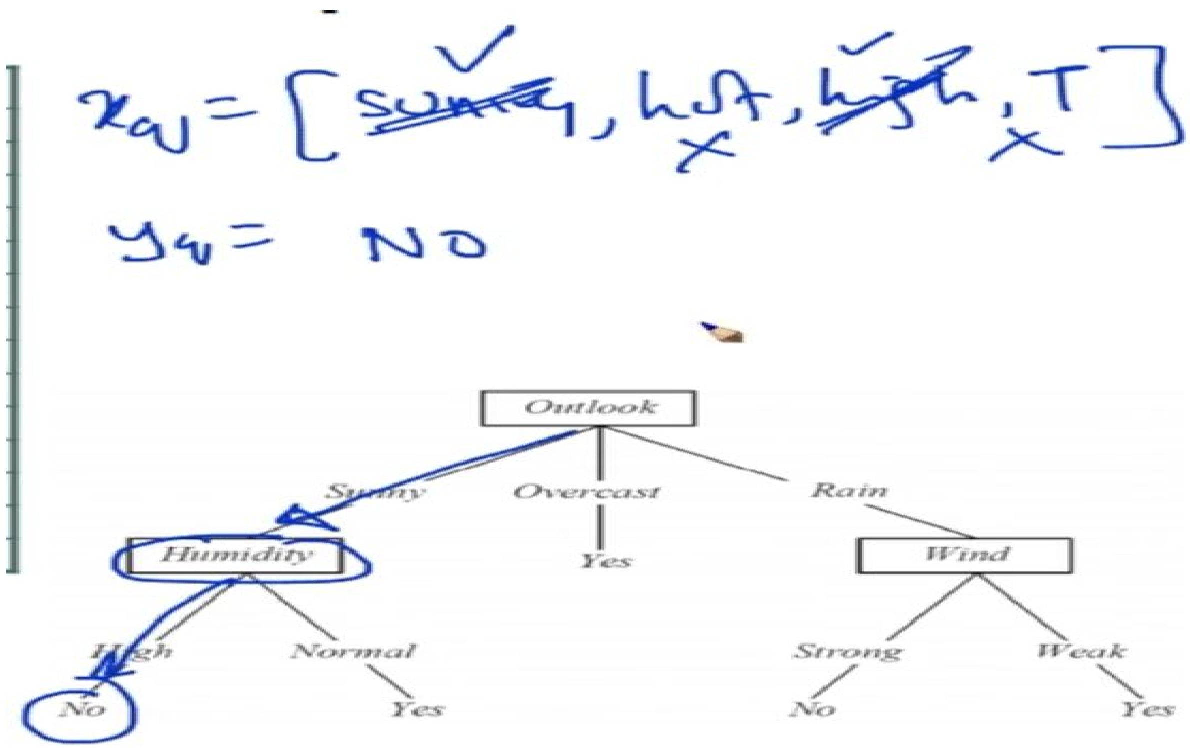

We've 2 features in our DT, PL and SL. So we plot a PL-SL graph and in that with every decision we take we draw a hyper-plane (Eg: PL < a then draw a hyperplane $\pi_1$ on 'a'. We are satisfying the condition that all PL < a => $y_i$ = 1) . Same for other decisions. For every decision we've a hyperplane and all the hyperplanes are axis-parallel i.e hyperplanes will always be parallel to axis.

## SAMPLE DECISION TREE




| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Sunny | Hot | High | False | No |
| Sunny | Hot | High | True | No |
| Overcast | Hot | High | False | Yes |
| Rainy | Mild | High | False | Yes |
| Rainy | Cool | Normal | False | Yes |
| Rainy | Cool | Normal | True | No |
| Overcast | Cool | Normal | True | Yes |
| Sunny | Mild | High | False | No |
| Sunny | Cool | Normal | False | Yes |
| Rainy | Mild | Normal | False | Yes |
| Sunny | Mild | Normal | True | Yes |
| Overcast | Mild | High | True | Yes |
| Overcast | Hot | Normal | False | Yes |
| Rainy | Mild | High | True | No |

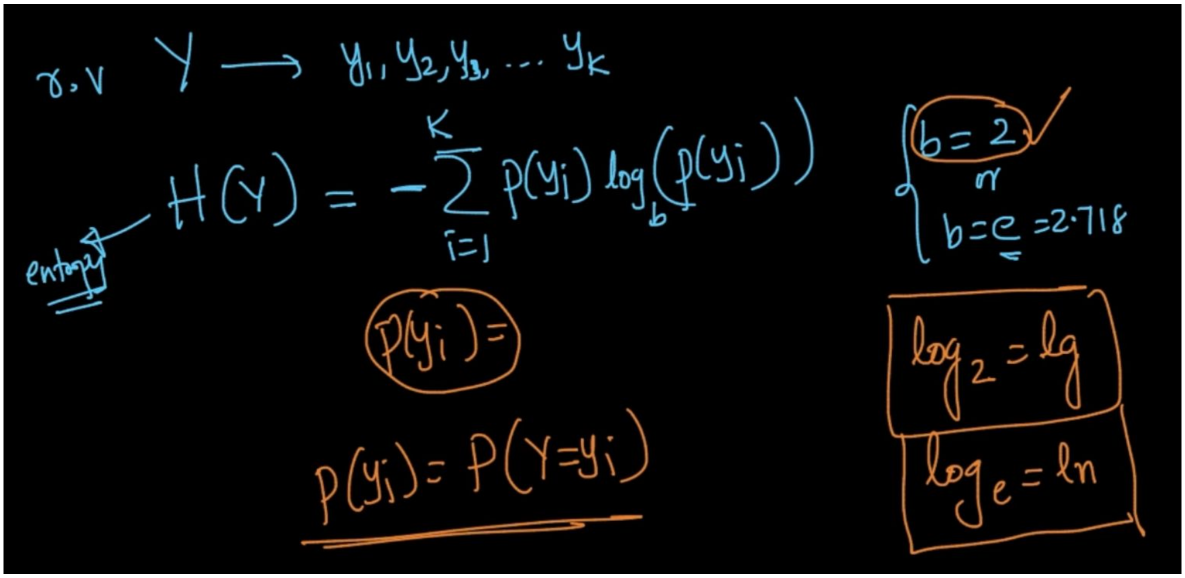
It’s a dataset given with categorical features and y is binary. We make our model(Decision Tree) using dataset. It works like a nested if-else as seen above.




How does it work with a new query $x_q$ = [ sunny, hot, high , True] then it'll work through the Decision Tree and follow the path. As it's visible we don't even need two features i.e hot and True

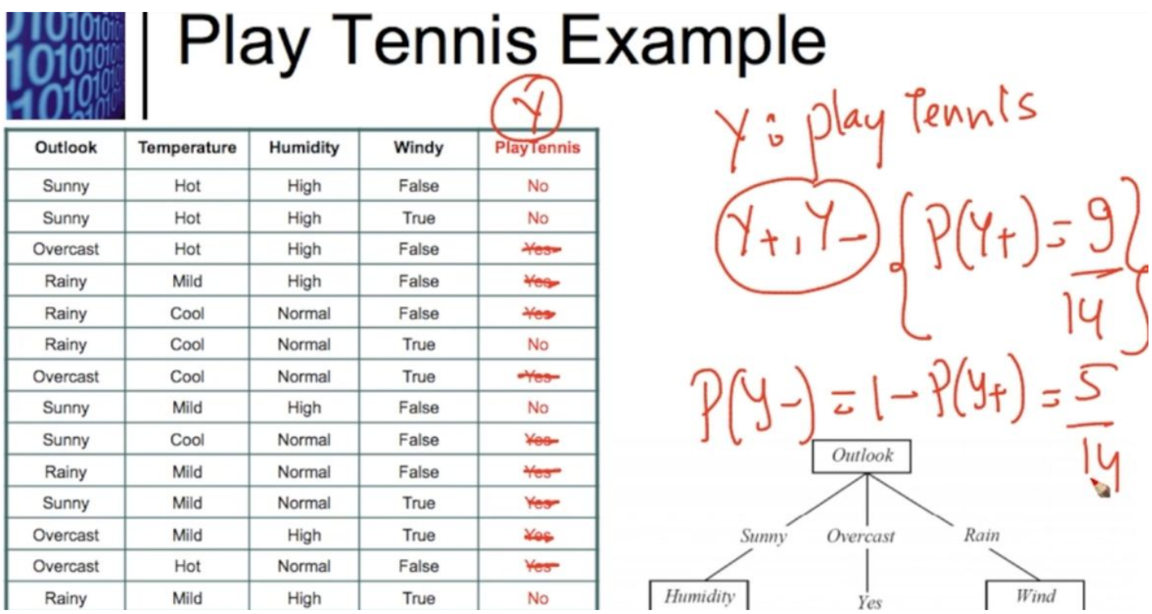
In Decision Tree the main part is to how to construct Decision Tree from Dataset after that
it's very easy to do $x_q$ => $y_q$

## BUILDING A DECISION TREE: ENTROPY



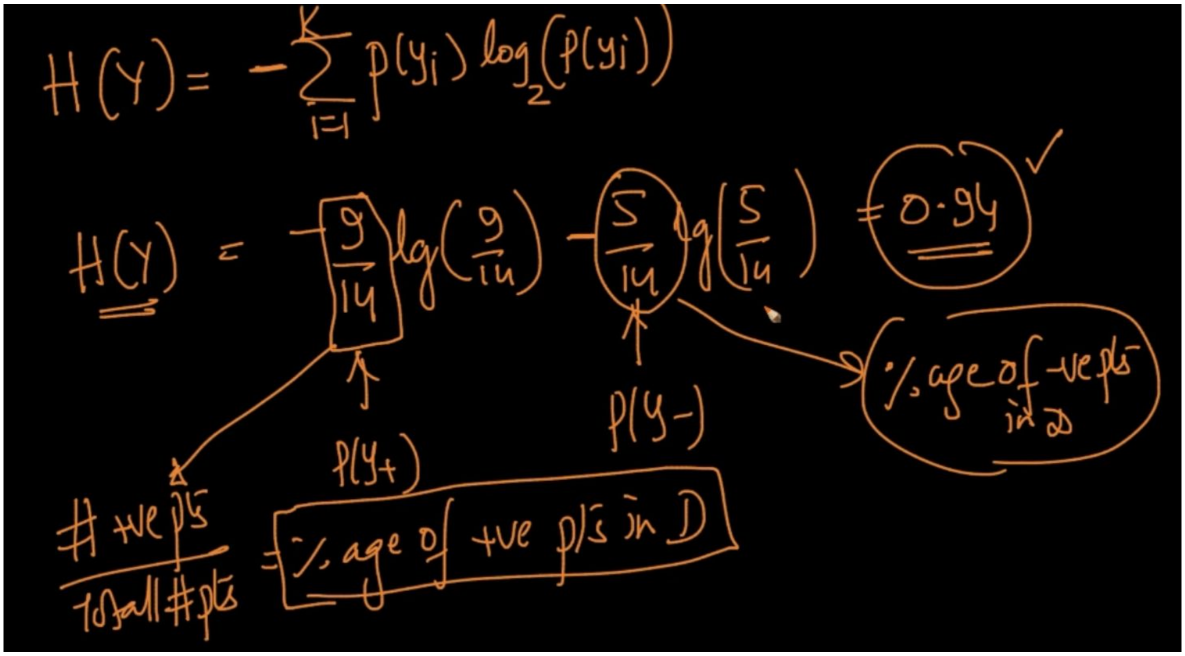

We are given a random variable Y -> $y_1, y_2 .....y_k$ i.e it can have k-values

Entropy : H(Y) = - $\sum_{i=1}^{k}$ P($y_i$) $log_b(P(y_i))$ where b = 2 or e typically. What is P($y_i$)? It's P(Y = $y_i$)

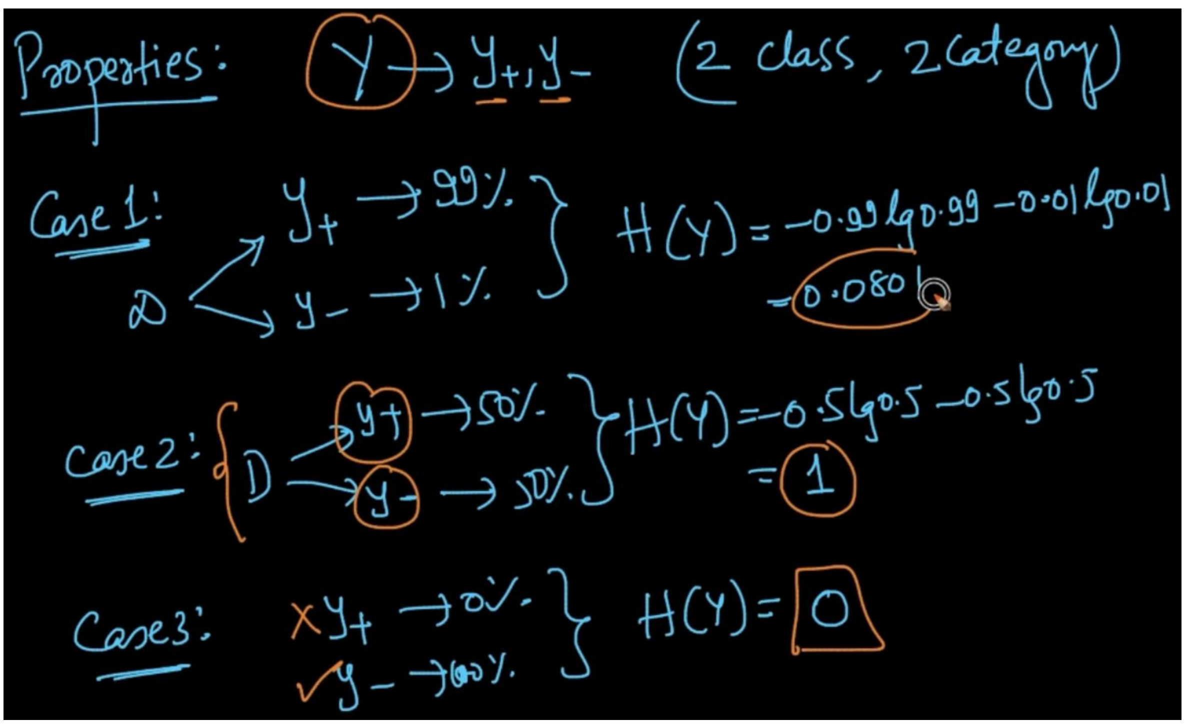



| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Sunny | Hot | High | False | No |
| Sunny | Hot | High | True | No |
| Overcast | Hot | High | False | ~~Yes~~ |
| Rainy | Mild | High | False | ~~Yes~~ |
| Rainy | Cool | Normal | False | ~~Yes~~ |
| Rainy | Cool | Normal | True | No |
| Overcast | Cool | Normal | True | ~~Yes~~ |
| Sunny | Mild | High | False | No |
| Sunny | Cool | Normal | False | ~~Yes~~ |
| Rainy | Mild | Normal | False | ~~Yes~~ |
| Sunny | Mild | Normal | True | ~~Yes~~ |
| Overcast | Mild | High | True | ~~Yes~~ |
| Overcast | Hot | Normal | False | ~~Yes~~ |
| Rainy | Mild | High | True | No |

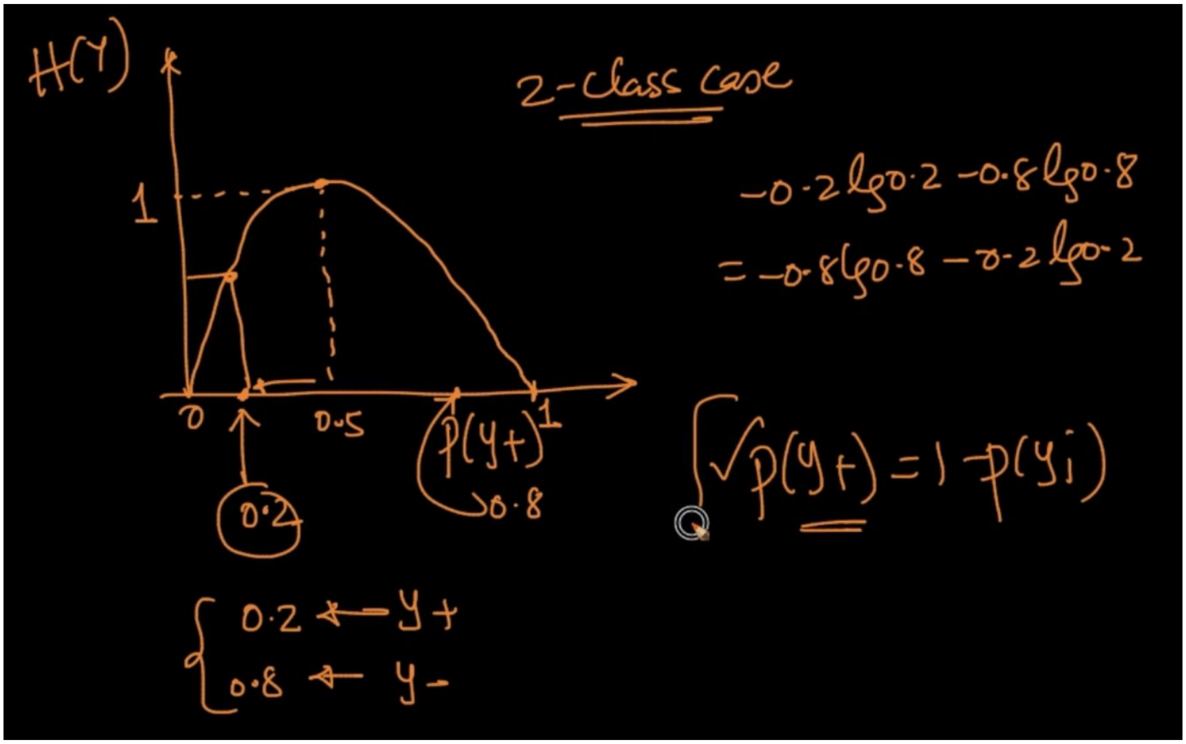
There are only 2 classes . So we need only P( $Y_+$ ) and P( $Y_-$ ) which is calculated



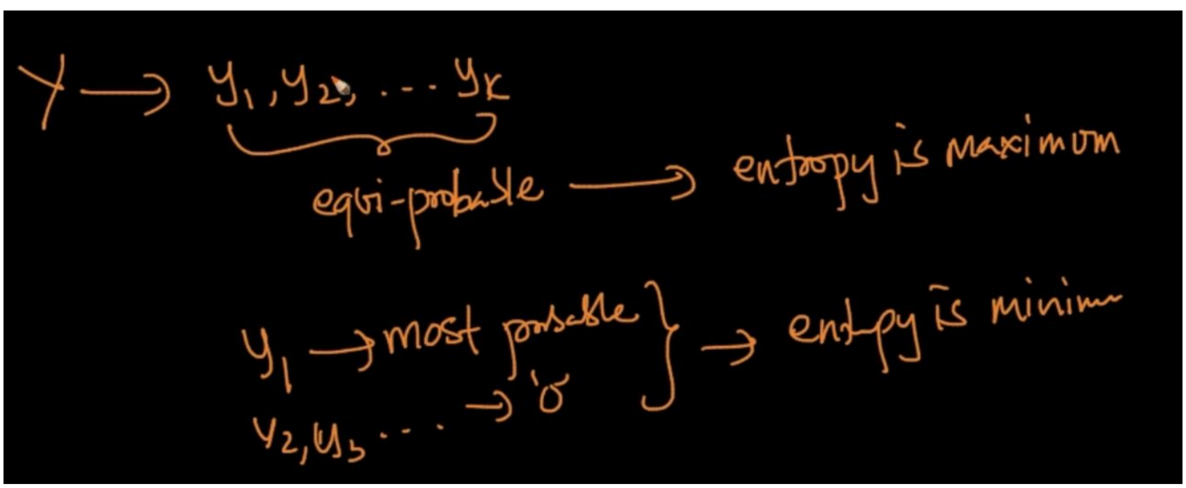

The Entropy of the above dataset is calculated like this



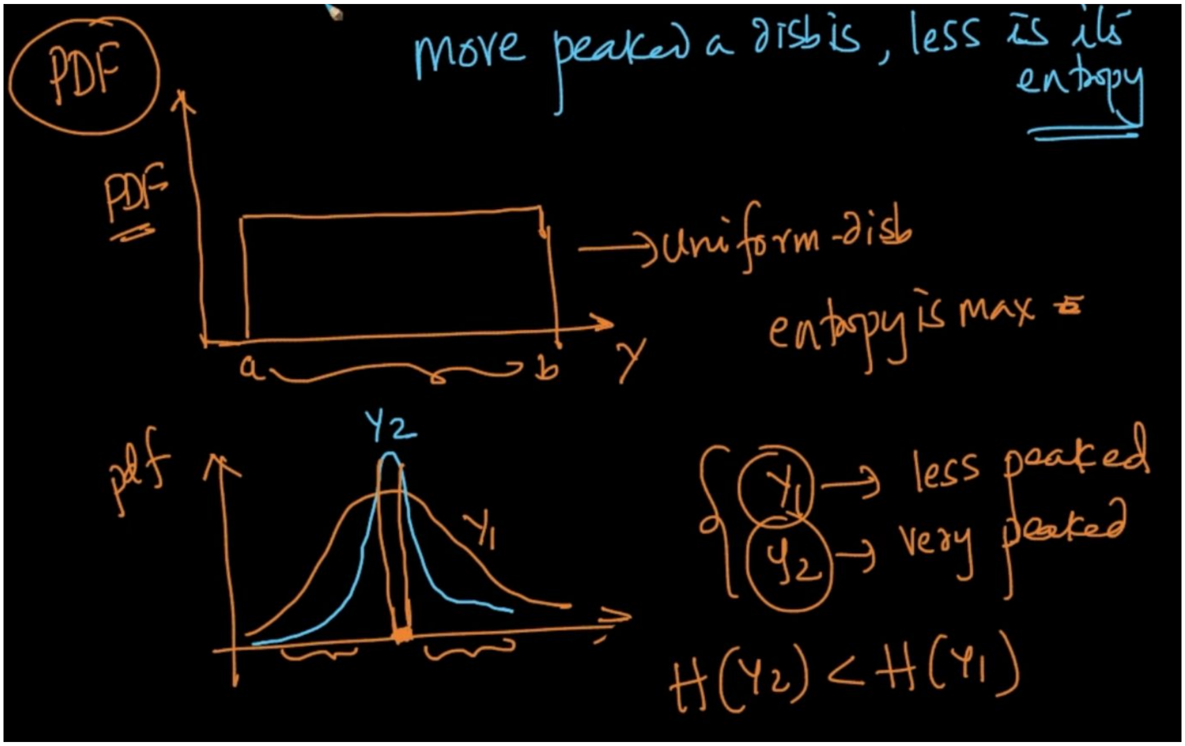

As we can see if both the classes are equiprobable(case 2) it's entropy is max. When 1 class is fully dominating the other class( Case 3) it's 0. In case 1 H(Y) decreased




P( $Y_+$ ) = 1 - P( $Y_-$ ). When we plot the graph of P( $Y_+$ ) v/s H(Y). When P( $Y_+$ ) = 0.5 or equiprobable H(Y) = 1 (It's maximum for a 2 class problem) . When P( $Y_+$ ) = 0.2 ;
P( $Y_-$ ) = 0.8 or we can say one class is dominating ,Entropy decreases




When a random variable Y is given and all it's class labels are equiprobable then entropy is maximum. When $y_1$ -> most probable and all the other y's -> 0 Entropy is minimum

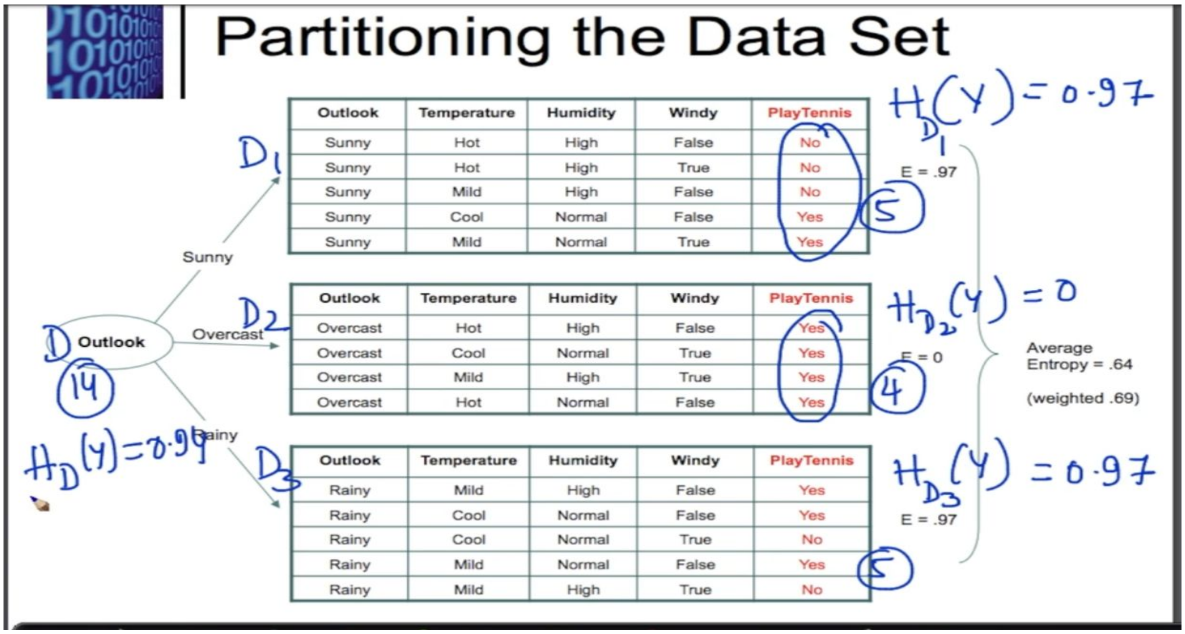



In the first figure all the values are uniformly distributed so entropy is maximum

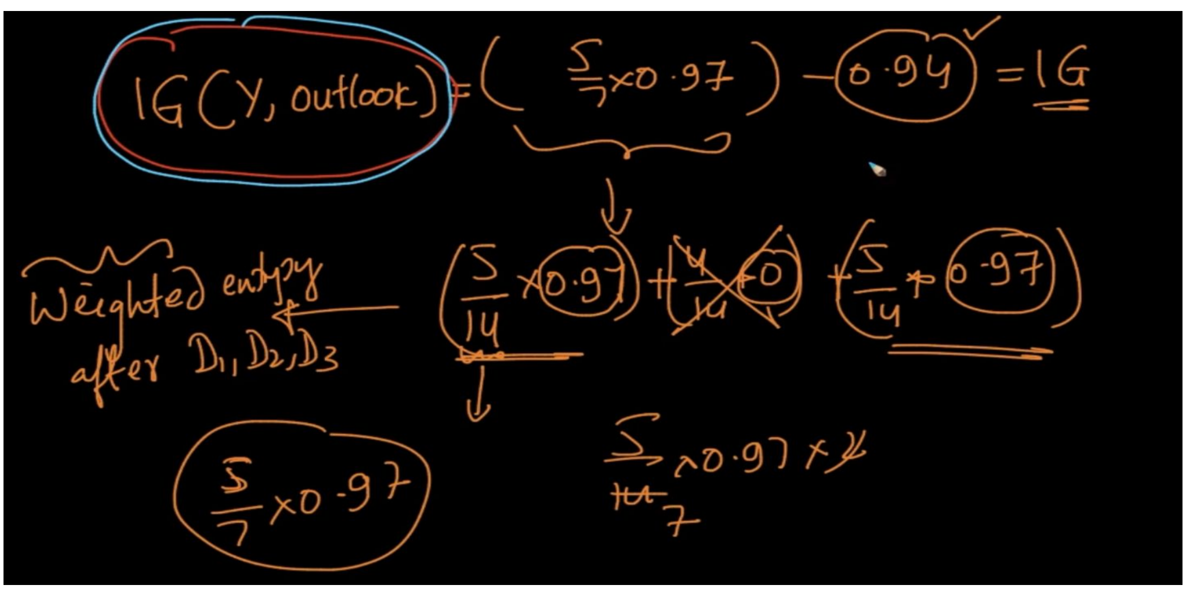
In the second one, $y_2$ is very peaked i.e it can only take small set of values and Probability of taking other values is less whereas $y_1$ is less peaked meaning it can take more values which means $y_1$ tending towards equiprobability. So, we can clearly say H($y_2$) < H($y_1$)

More peaked a distribution is less is it's entropy

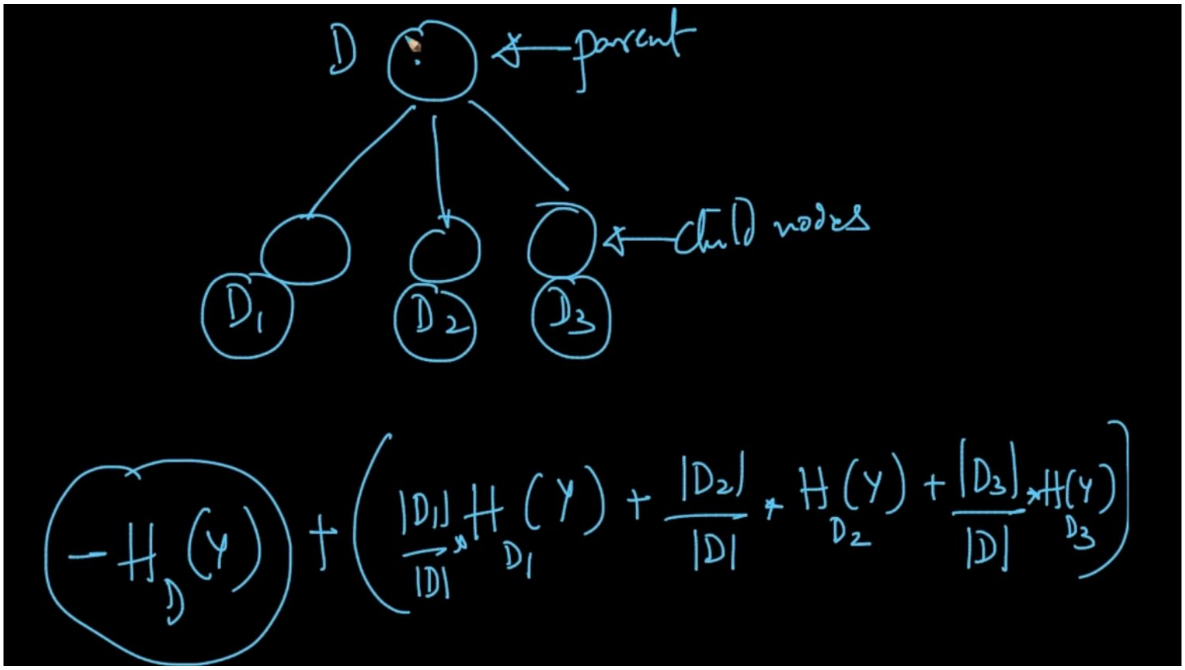
## BUILDING A DECISION TREE : INFORMATION GAIN



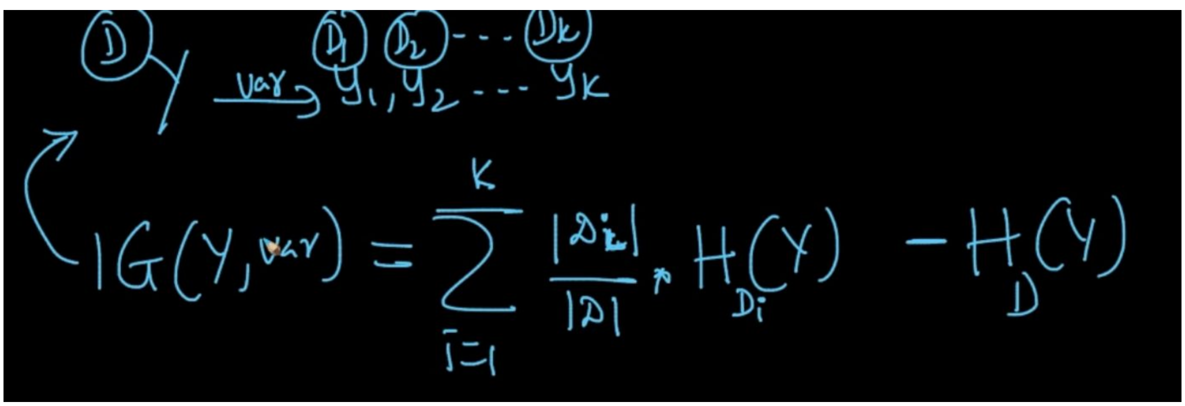

We divide the Dataset based on Root Node ''Outlook' which is a categorical feature having 3 discrete values = [Sunny, Overcast, Rainy]. Now we calculate the Entropies of the divided datasets.



Information Gain by ‘Outlook’ :  IG(Y, Outlook) is calculated as above . First we summate all the Entropies for the divided daatset and Entropy for the dataset is used for IG

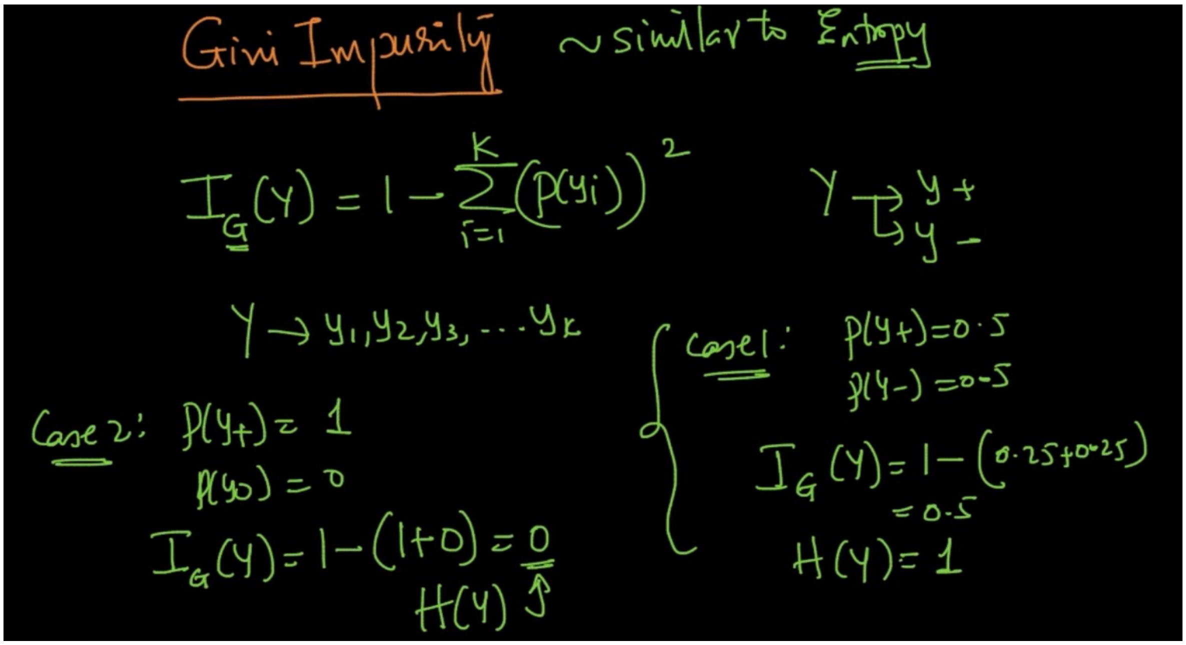



This is how Information Gain is calculated where $D_1, D_2, D_3$ are no. of points in the divided datasets respectively and D is total no. of features/rows

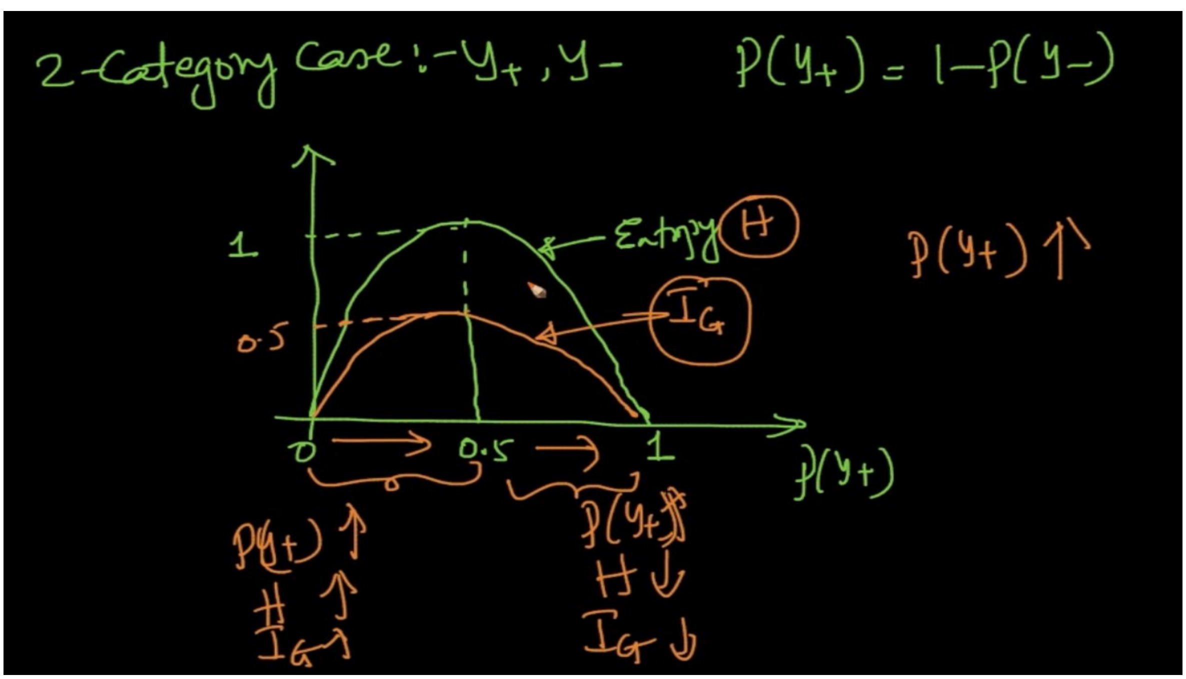



IG(Y, var) = $\sum_{i=1}^{k} \frac{|D_i|}{|D|}$ * $H_{D_i}(Y)$ - $H_D$ (Y)

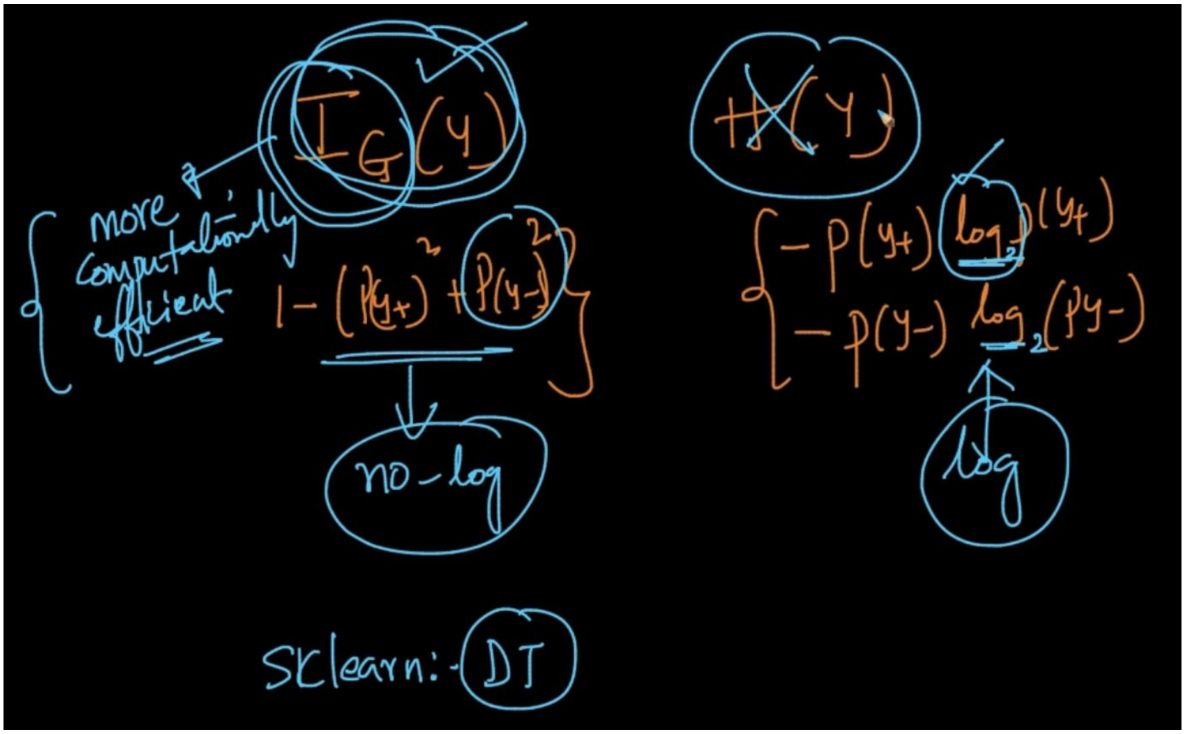
## GINI IMPURITY




$I_G(Y) = 1 - \sum_{i=1}^{k} (P(y_i))^2$. We've discussed different cases there



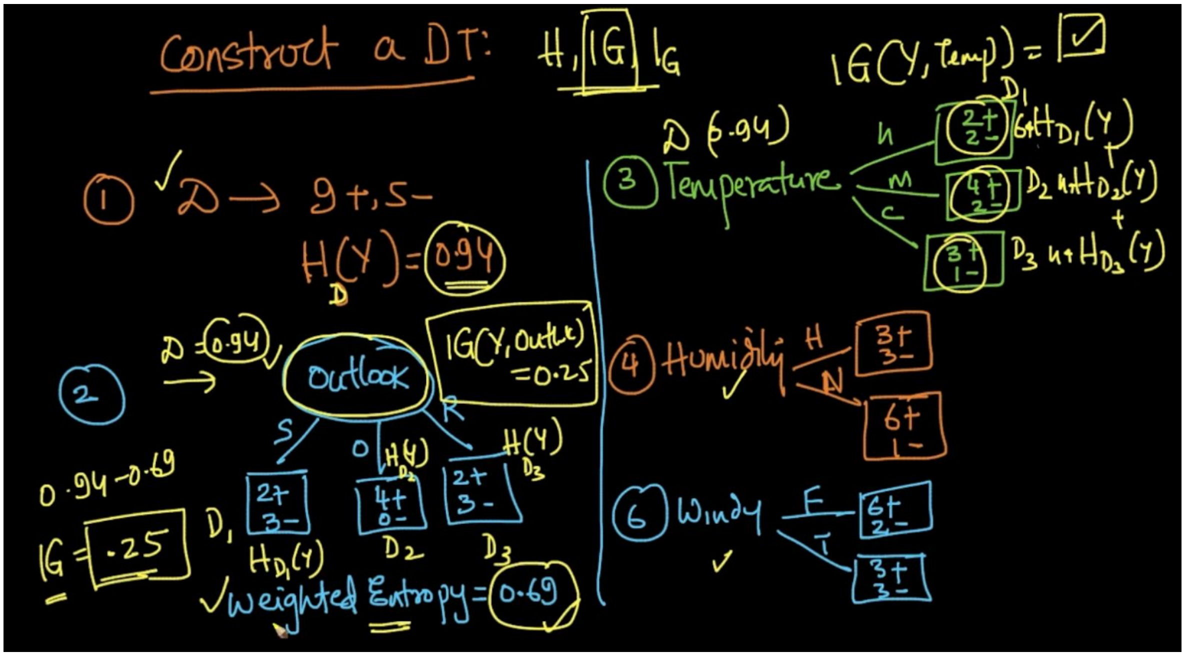
As seen, Entropy and Gini Impurity are very similar except the max value of $I_G = 0.5$



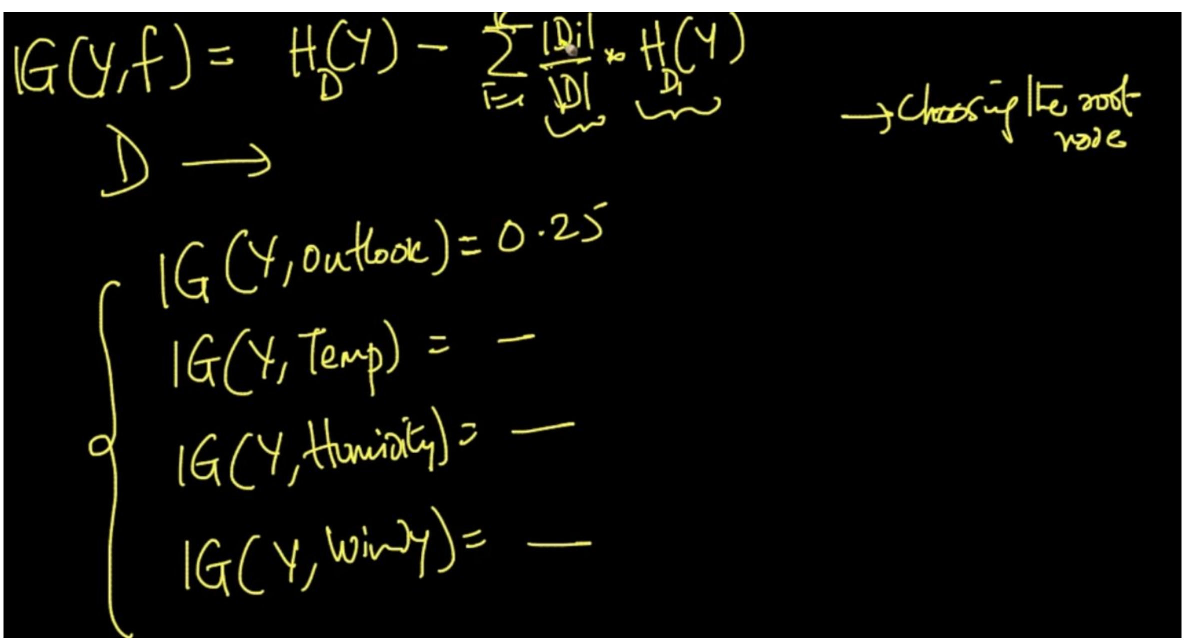

So if they are very similar then why do people prefer Gini's impurity ?

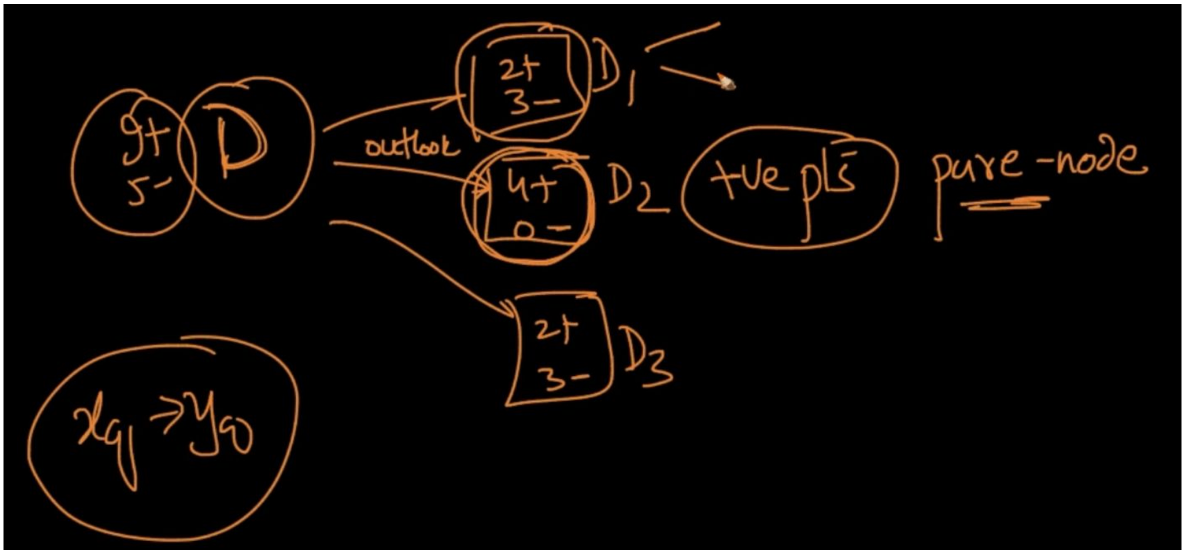
It's because as seen in the formula we take out squares in $I_G$ but in H(Y) we use log and computing squares is computationally more efficient than computing logs. That's why Gini is preferred

## CONSTRUCT A DECISION TREE: BUILDING A DECISION TREE

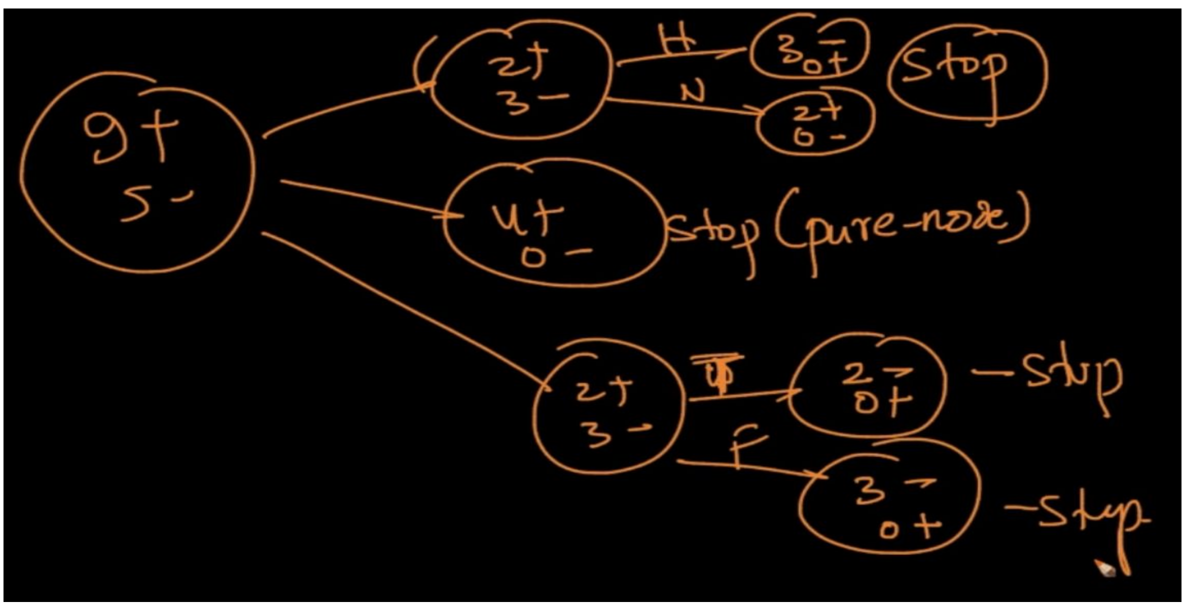



We calculate the Entropy of all the dataset. After that we have to decide the feature for Root Nodes. So for choosing that we choose one , we decide the feature use that as a Root Node and partition the data based on that and calculate the Entropies and their combined Weighted Entropies using the individual Entropies. We do it for all and calculate IG (Y, Feature). It is shown below



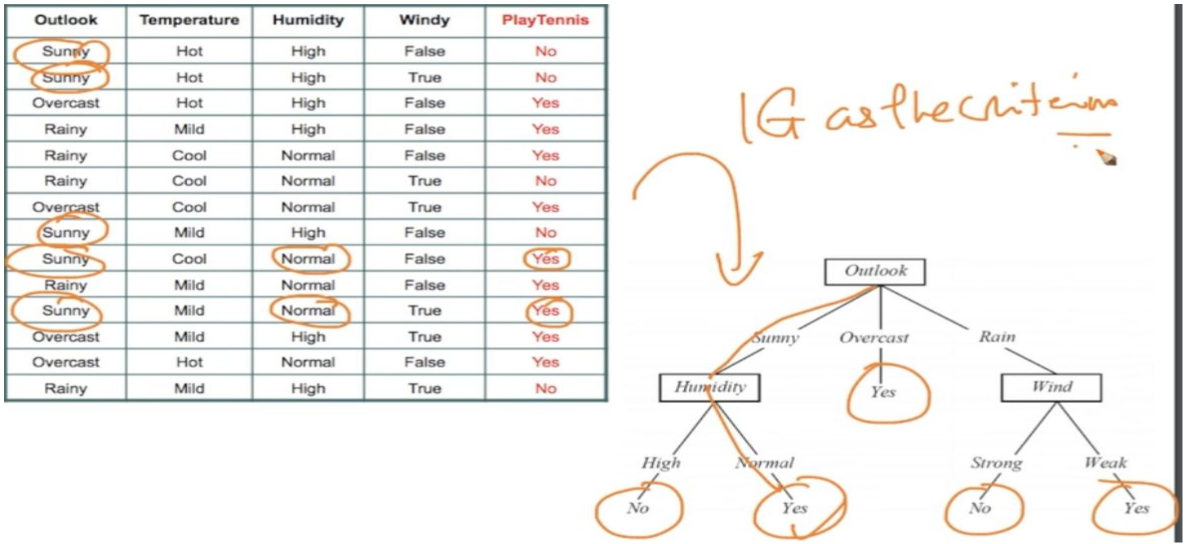



We decide Feature "Outlook" as it has the maximum IG. In the divided Dataset, $D_2$ is a pure node since it has only +ve points and for the other Datasets we partition further by using the remaining features and calculating their Information Gain and selecting the Maximum Information Gain




$D_1$ has 2+ and 3- and if we break it up by 'Humidity' we can see the dataset is further divided and we get pure nodes. Same procedure for $D_3$. We divide it on the basis of 'Windy'. So our DT is completed using IG as criteria

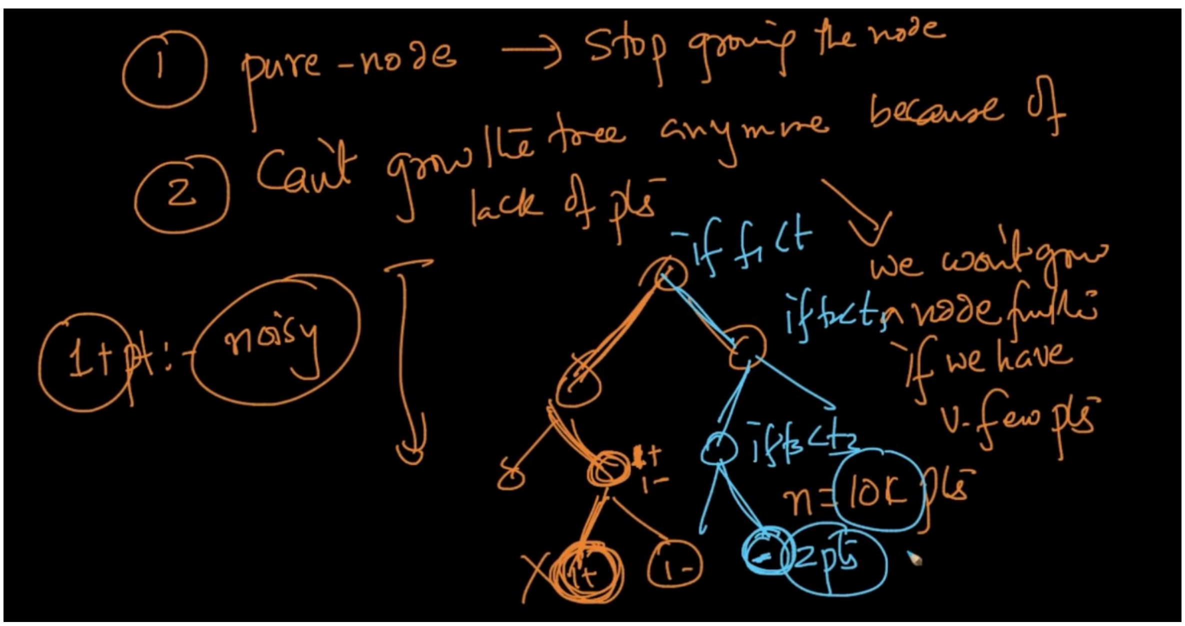
**Why is a Pure Node good ?**



| Outlook | Temperature | Humidity | Windy | PlayTennis |
|---|---|---|---|---|
| Sunny | Hot | High | False | No |
| Sunny | Hot | High | True | No |
| Overcast | Hot | High | False | Yes |
| Rainy | Mild | High | False | Yes |
| Rainy | Cool | Normal | False | Yes |
| Rainy | Cool | Normal | True | No |
| Overcast | Cool | Normal | True | Yes |
| Sunny | Mild | High | False | No |
| Sunny | Cool | Normal | False | Yes |
| Rainy | Mild | Normal | False | Yes |
| Sunny | Mild | Normal | True | Yes |
| Overcast | Mild | High | True | Yes |
| Overcast | Hot | Normal | False | Yes |
| Rainy | Mild | High | True | No |

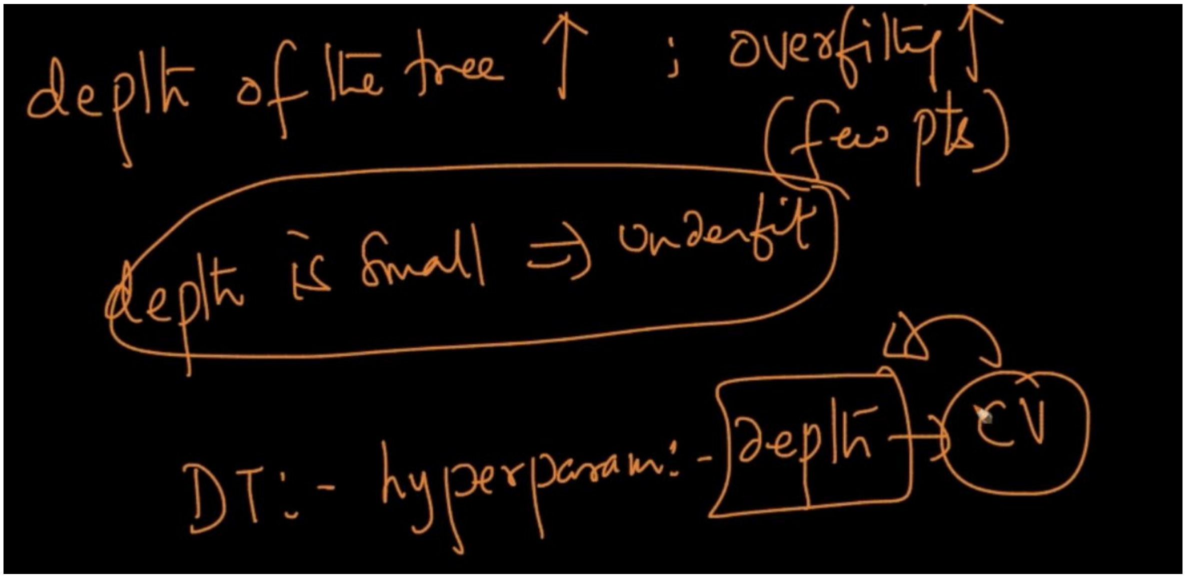


As we can see, with the completed DT . Suppose in a new query $x_q$ if 'Outlook' = Sunny And 'Humidity' = Normal then 'Yes'



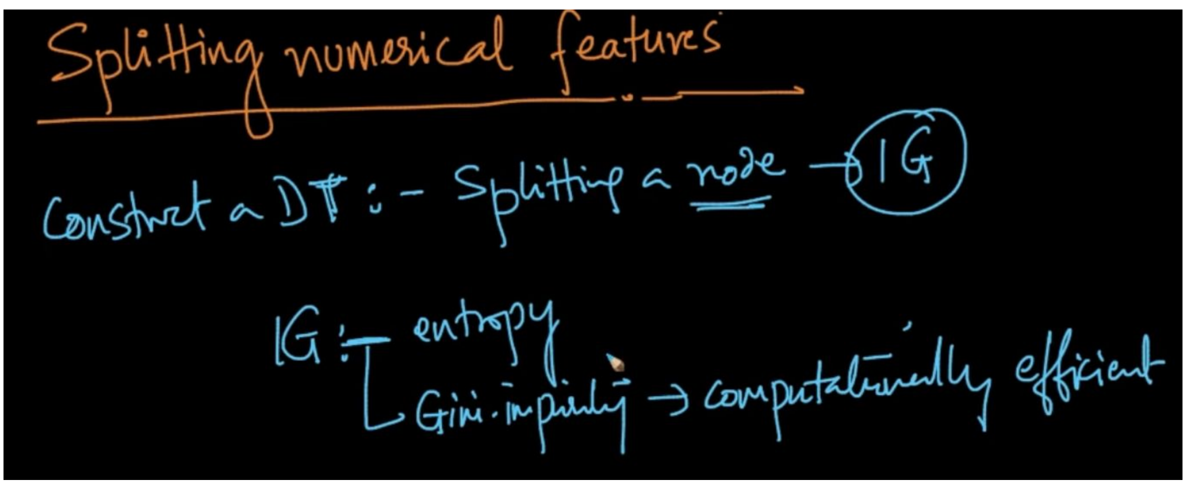

For the 2nd point if we have lack of points we stop growing a tree bceause suppose we've just 2 points : 1+ and 1- we can divide it but it's pointless if 'n' = 10k because we shouldn't make tree like this because it's literally overfitting for a single point

If the depth of the tree is also high then it's also overfitting. So choose the right depth

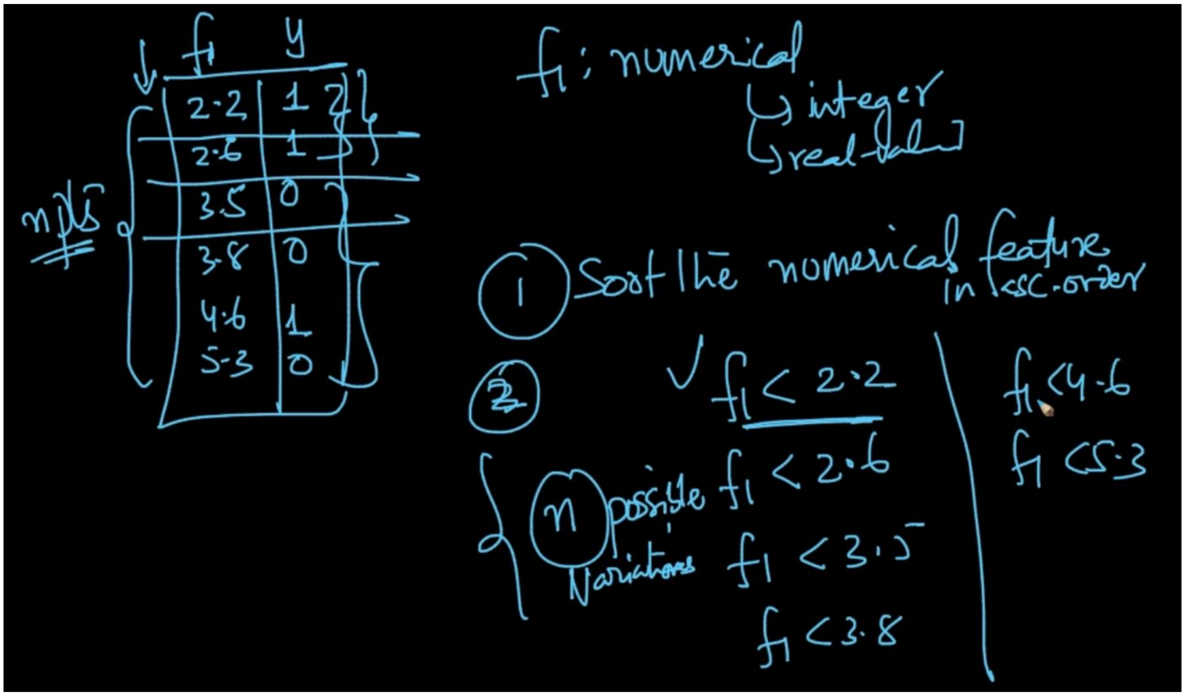



So our depth of the Tree is a hyper-parameter and we can know the right depth by Cross Validation

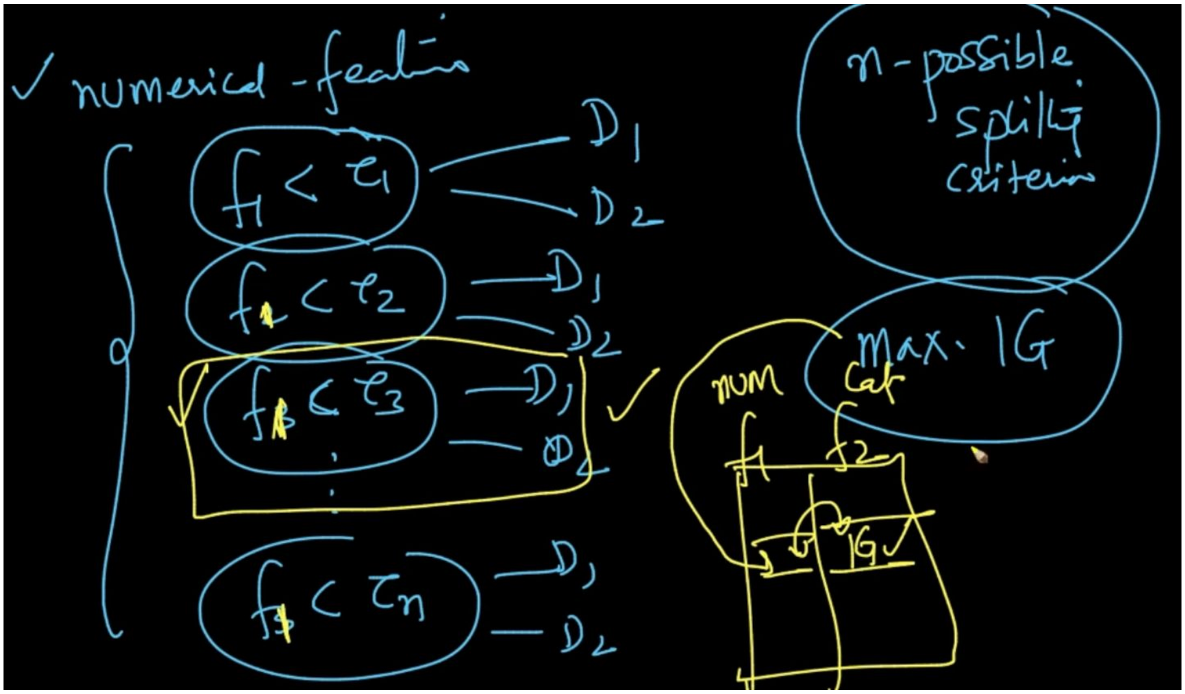
**BUILDING A DECISION TREE : SPLITTING NUMERICAL FEATURES**




The most important step in constructing a DT is Splitting a node and we did that by using IG - We used Entropy for it but we can use Gini Impurity as well since it’s computationally efficient. But we’ve a problem here since we only dealt with Categorical Features and Discrete Random Variables and constructing a DT is easy because we splitted with the categories but what if we’d Numerical data

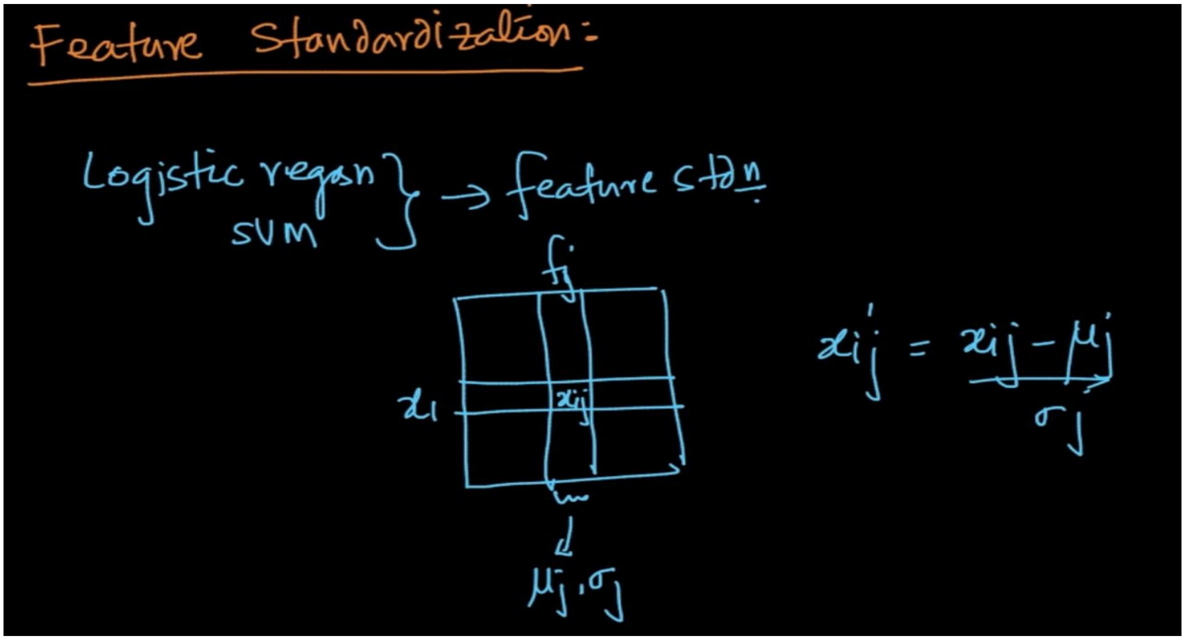



If we've numerical features like above then first we sort in ascending order and create Datasets using conditions like above. For Ex: If $f_1$ < 3.5 then we've 2 pts below it and 4 points above it. We can have a partitioned dataset . We can do it for every value. If we've 'n' points in a dataset then we've 'n' possibilities

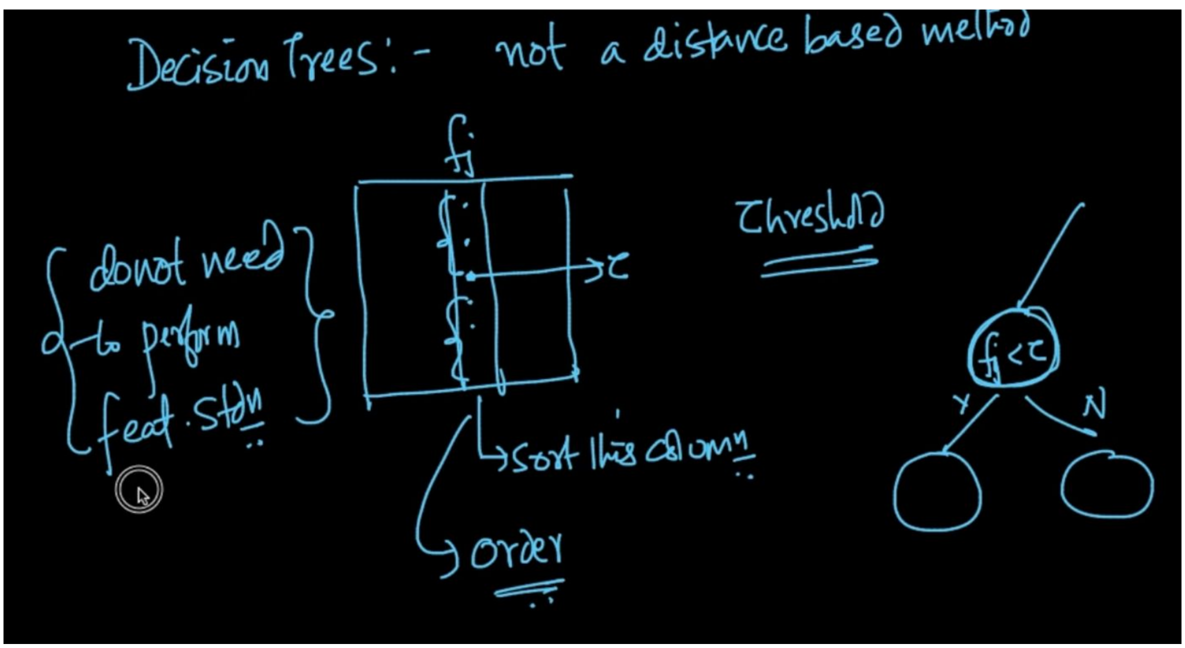



Sp for n points we can create partitioned datasets by applying Thresholds and we select the one which has maximum Information Gain. If we've 2 features $f_1$ and $f_2$ we compare the IG and select feature but since there are 'n' possibilities it can get time consuming

## FEATURE STANDARIZATION

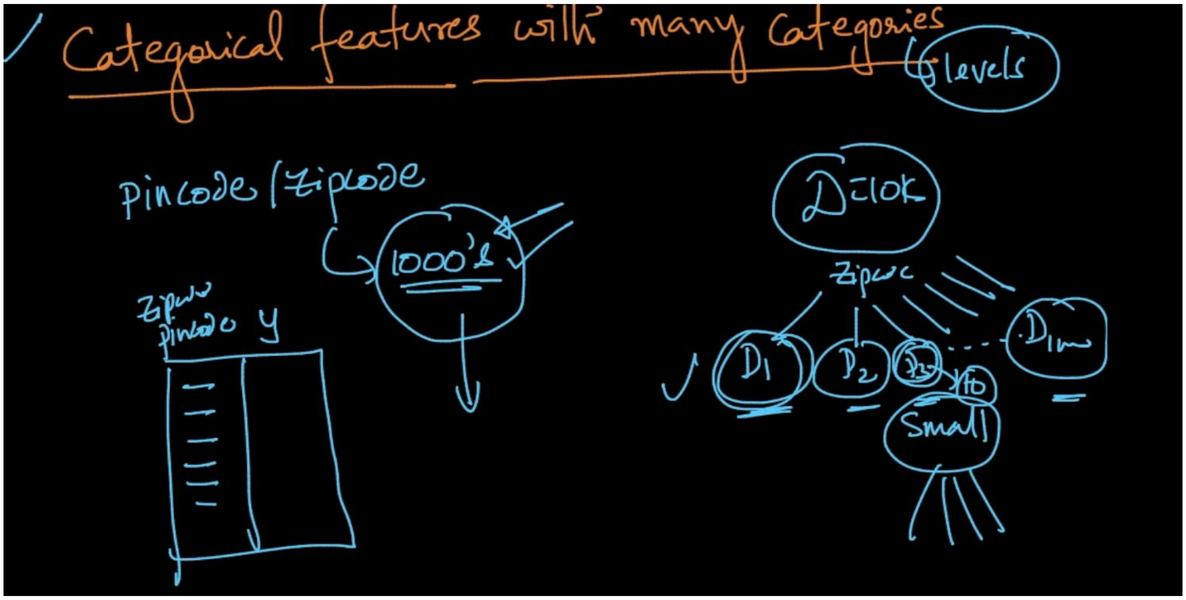



In the Logistic Regression, SVM we did feature Standardization like above

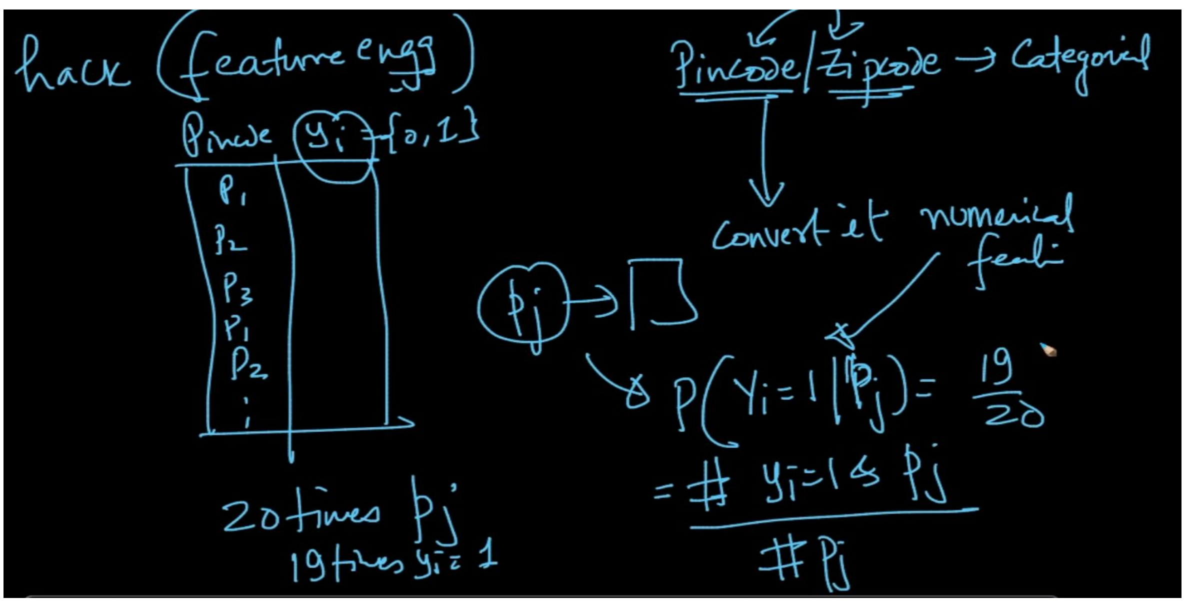



In Decision Trees we don't need to do Feature Standardization since it's not a distance based method like other ML algorithms we only care about whether something is greater than something or not. This is one of the great aspect of DT

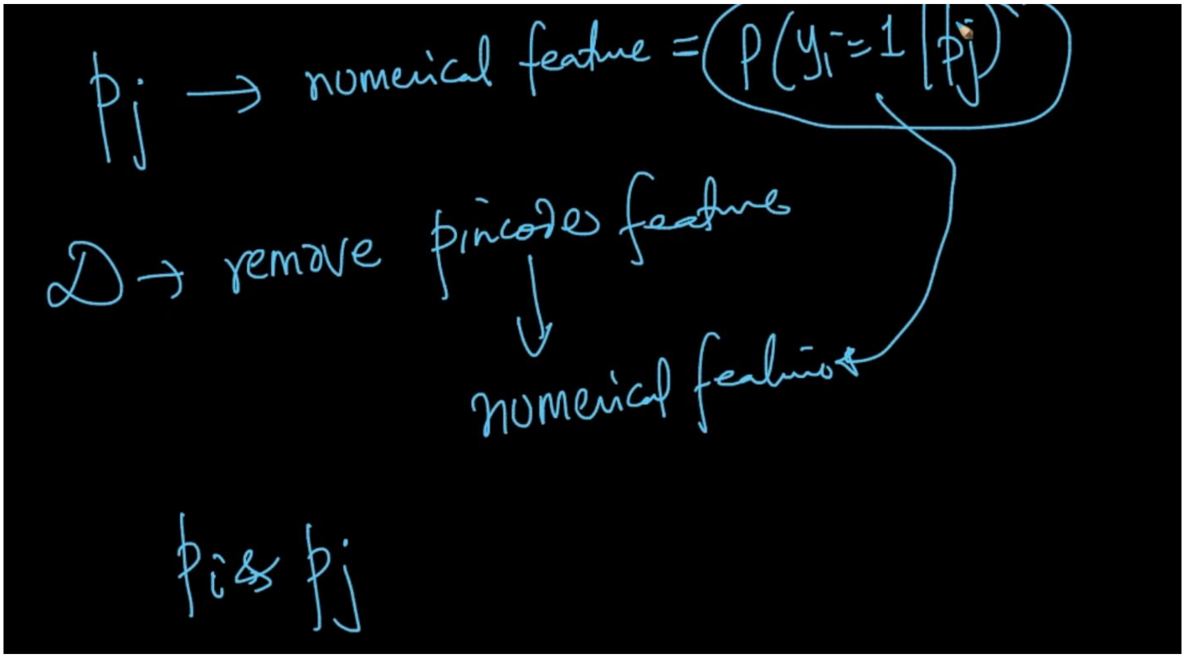
## CATEGORICAL FEATURES WITH MANY CATEGORIES



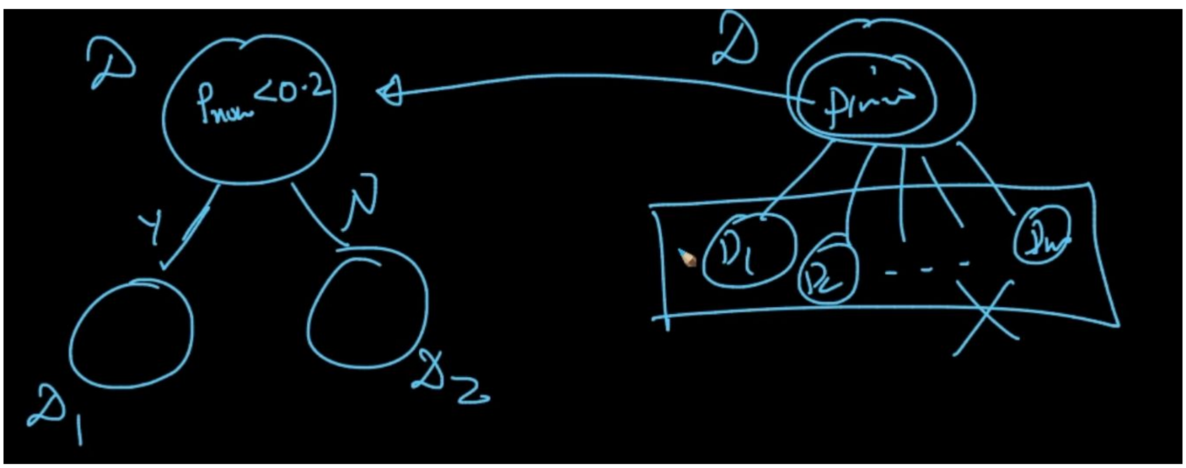

A Pincode is a Categorical feature since it can't be compared  and it can be in 1000's in number so we will create 1000's of partitioned datasets which can be trivial




We can convert our Categorical Feature to a Numerical Feature . We convert it like above where P($y_i$ $= 1 |$ $P_j$) i.e Probability of $y_i$ = 1 given Pincode $P_j$ = No of points where $y_i$ = 1 and $P_j$ / No. of points where Pincode is $P_j$

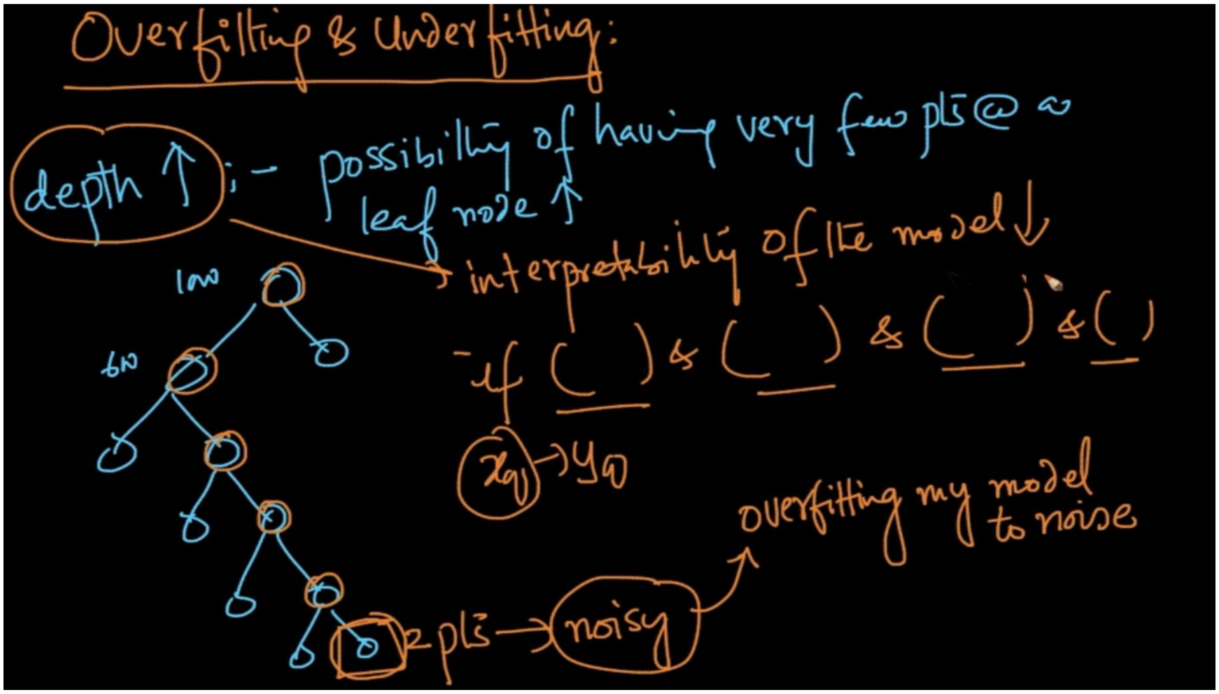



So we are removing PinCode feature by converting it into Numerical Feature.

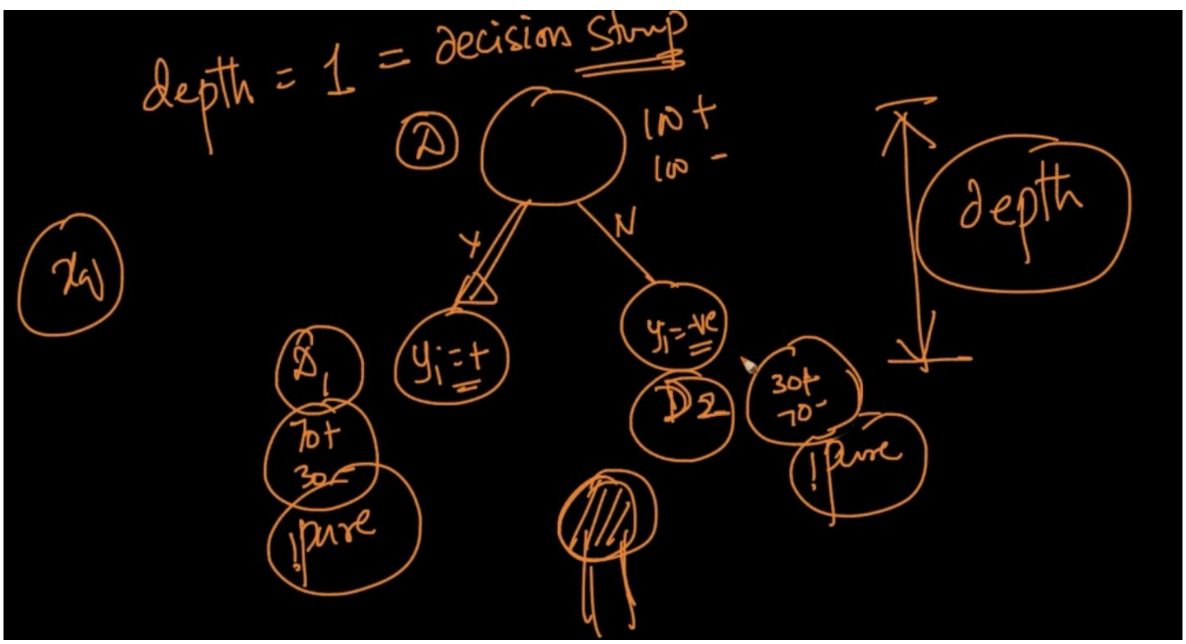



After converting The pincodes to Probability like above $P_{numerical}$ < 0.2 we create the ideal partitions and getting rid of the problem of Data Sparsity

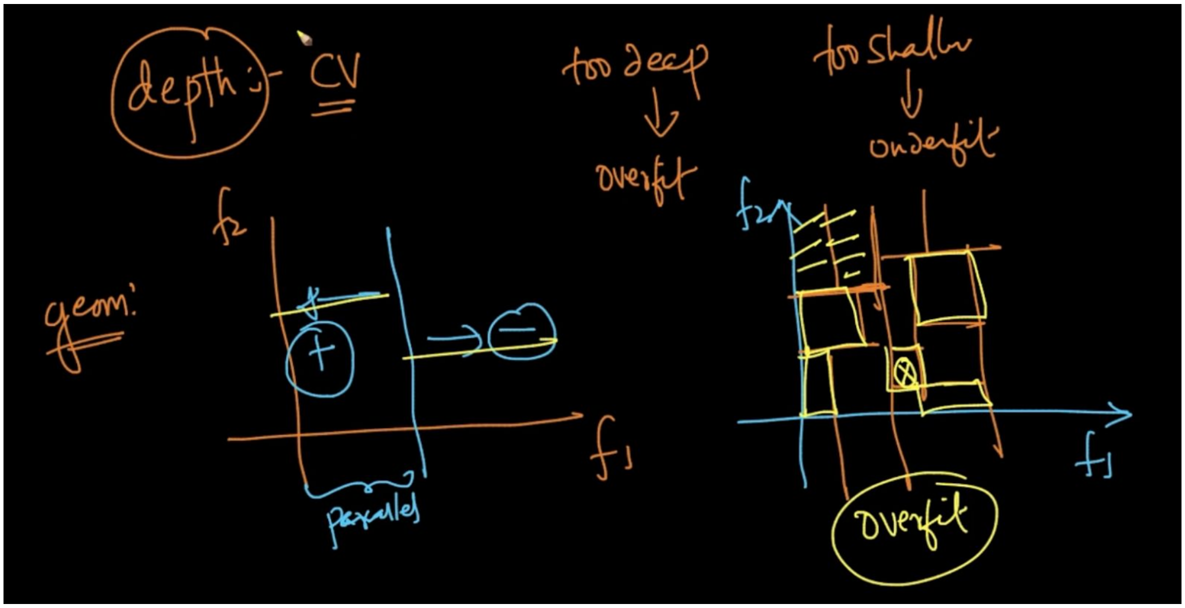
## OVERFITTING AND UNDERFITTING




If the depth of the tree is very high then the possibility of few points at leaf node ↑ which means we might overfit our model and interpretability of model ↓ since many if conditions.



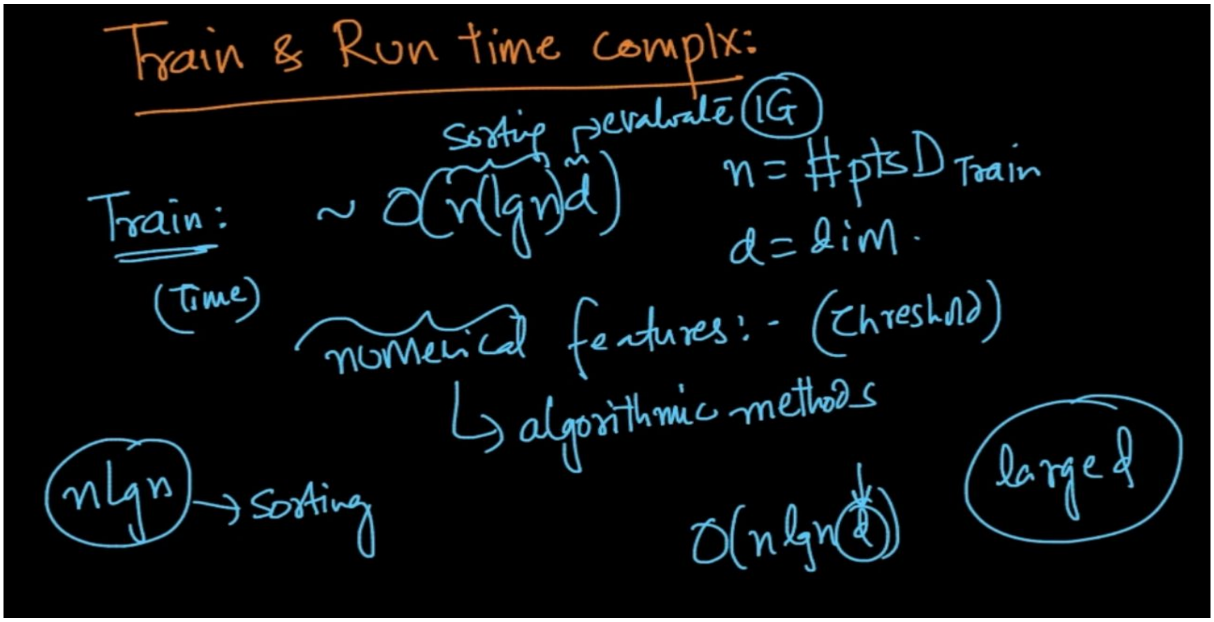
In a low depth tree suppose if we've 70+ve , 30 -ve points in $D_1$ and 70 -ve , 30 +ve points in $D_2$ we declare $D_1$ as +ve and $D_2$ as -ve although both are impure nodes



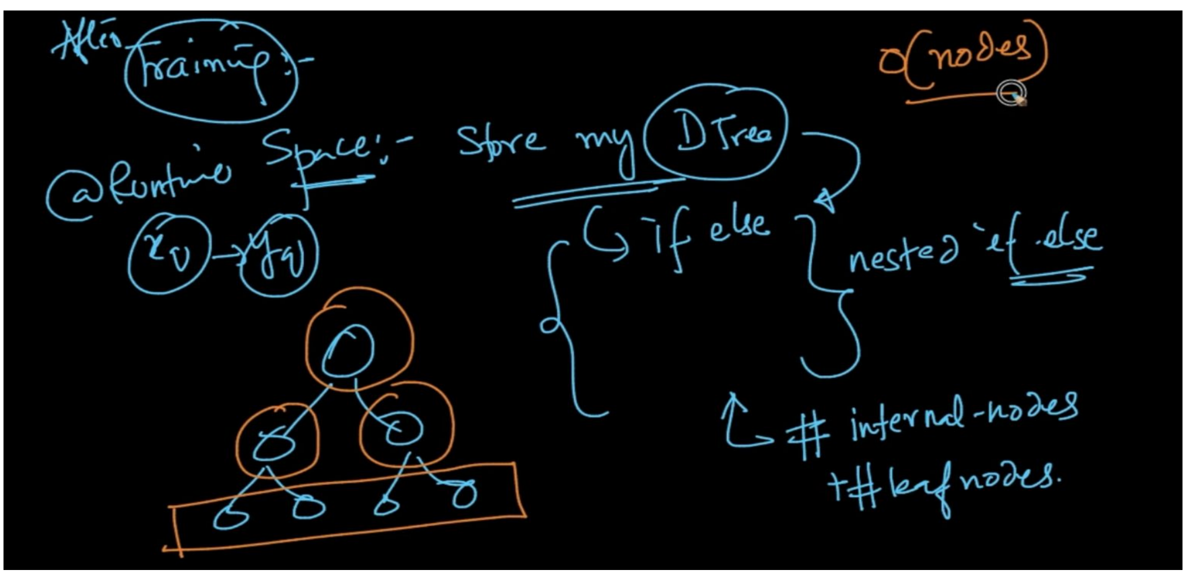

In the left figure , it is underfitting since only one axis parallel is dividing the dataset and in right there are many axis parallels so there can be an area where there's only 1 point and we might predict a query's class label using 1 point which is overfitting .

So in order to decide the right depth we do Cross Validation

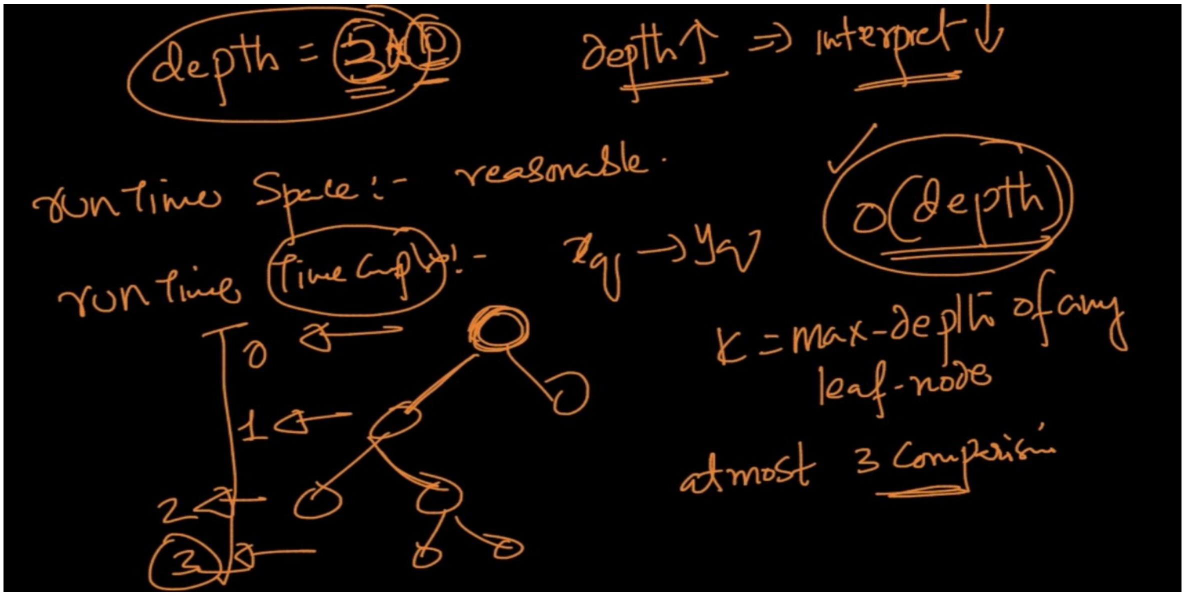
## TRAIN AND RUN TIME COMPLEXITY




Train : O(n * (log n) * d) where n = No. of points in Training Dataset and d = dimensions We've come up with algorithmic methods for Numerical Features. So , (n * log(n)) should be for sorting since sorting has Time Complexity : O (n * log(n)) and d is a multiplier since at every stage we've to compare the features so if our 'd' is very large then Decision Tree isn't a great option



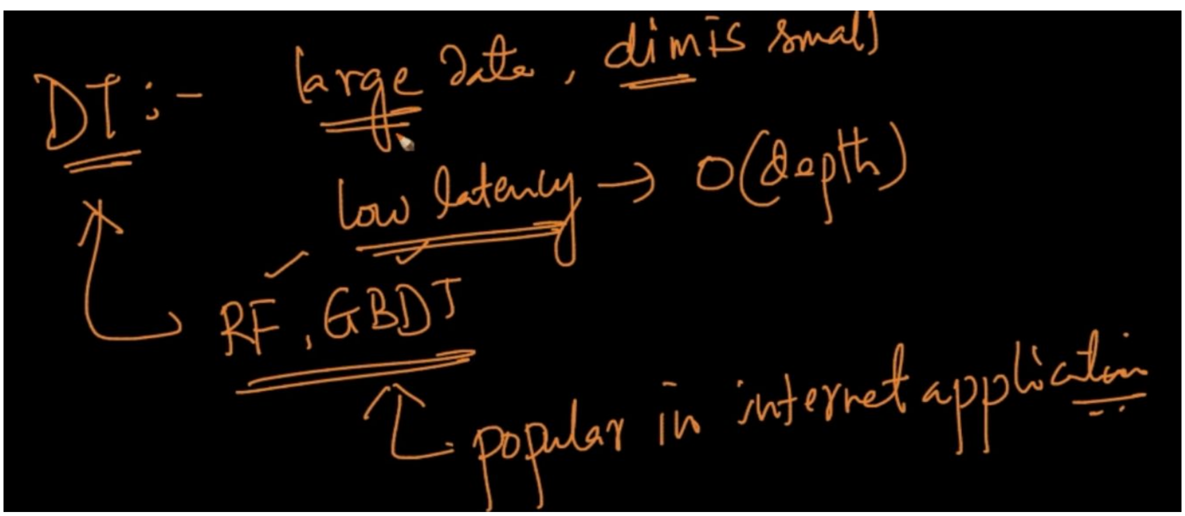
After Training : The Run Time Space Complexity :- Just to store a Decision Tree and it's not hard since a **Decision Tree** is basically a ***nested if*** - else and ***storing*** it is ***No. of Internal Nodes + No. of leaf Nodes*** which takes ***very less*** space = ***O(nodes)*** which is not too hard if you've ***reasonable depth*** of Decision Tree




Our decision Tree has only depth = 5 or 10 because as depth ↑ = inpretability ↓

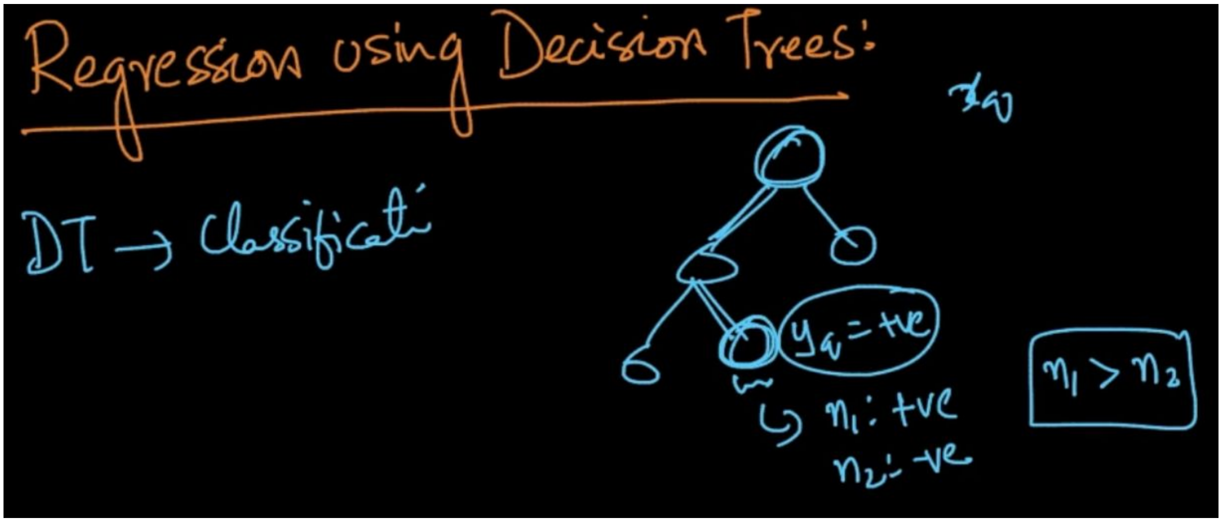
Runtime Time Complexity : It is just ***'k'*** where k = maximum depth of any leaf node

Ex:- In the above Tree 'k' = 3 because it has depth = 3 . So it's ***O(depth)***

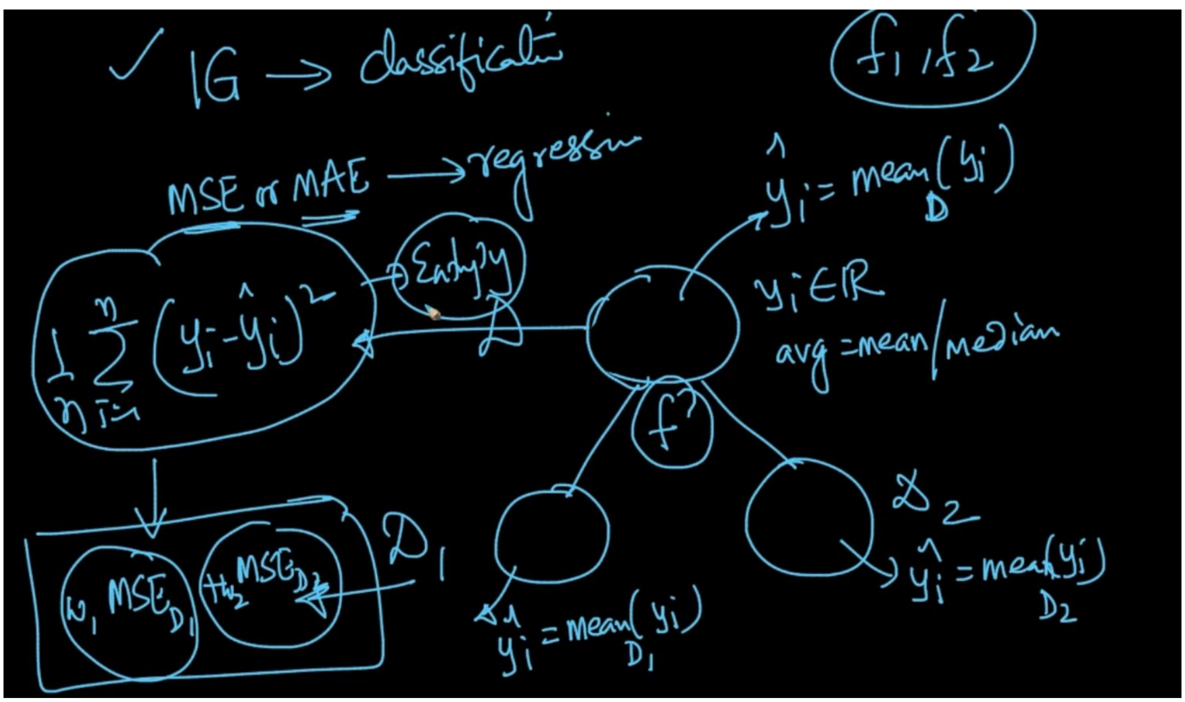



Decision Tree is good when we've the above features. Random Forest(RF), Gradient Boosting Decision Tree (GDBT) are different Decision Trees which are very popular

# REGRESSION USING DECISION TREES

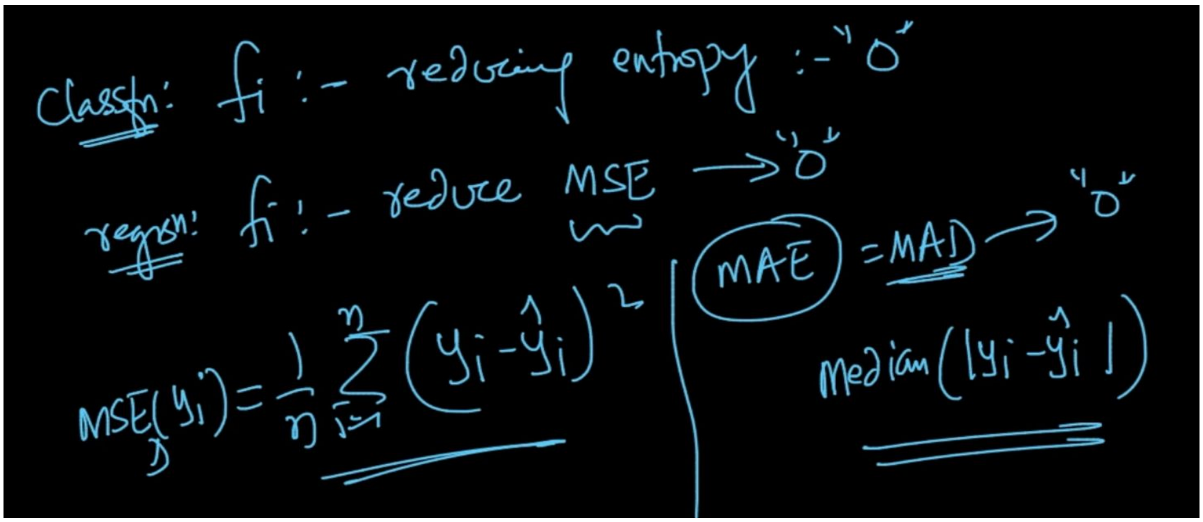



In Classification , in the leaf node $n_1$ : +ve and $n_2 :-ve$ and $n_1 > n_2$ so if a new query $x_q$ arrives at that leaf node it's $y_q$ = +ve since $n_1 > n_2$

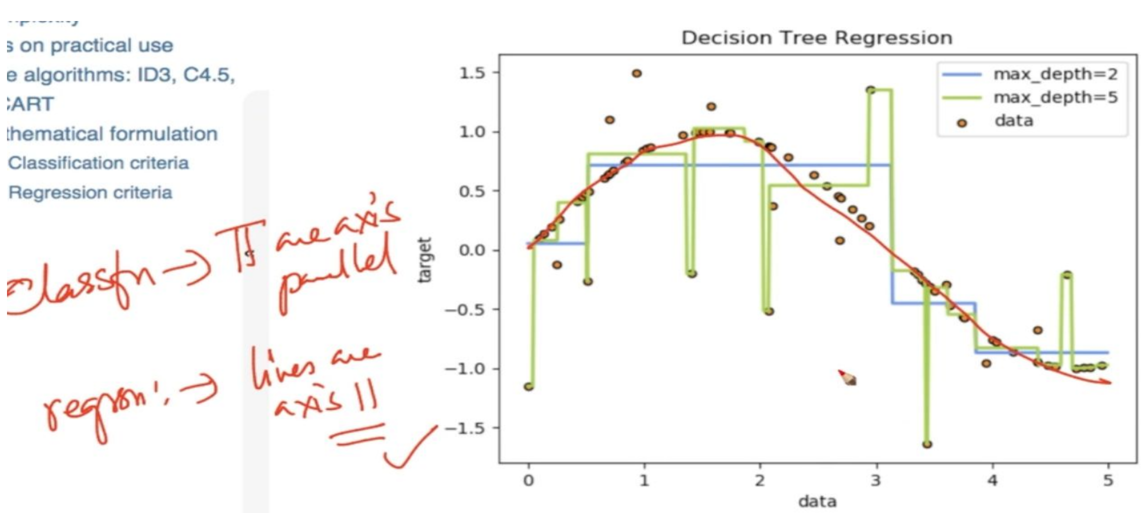



In Classification we used IG for feature selection . In Regression, we use Mean Squared Error or Mean Absolute Error . For the root node we take $\hat{y}_i$ = $mean_D\,(y_i)$ i.e avg/median Then we divide the dataset with some feature and for $D_1$ $\hat{y}_i$ = $mean_{D_1}(y_i)$ for all 'y' in $D_1$

Same for $D_2$. How do we decide which feature to use for breaking ? Just like IG we use Mean Squared Error (MSE) : $\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$ which is like Entropy and then we calculate $w_1MSE_{D_1}$ + $w_2MSE_{D_2}$. Suppose we've features $f_1, f_2$ we split by both and whichever reduces $w_1MSE_{D_1}$ + $w_2MSE_{D_2}$ we select that feature



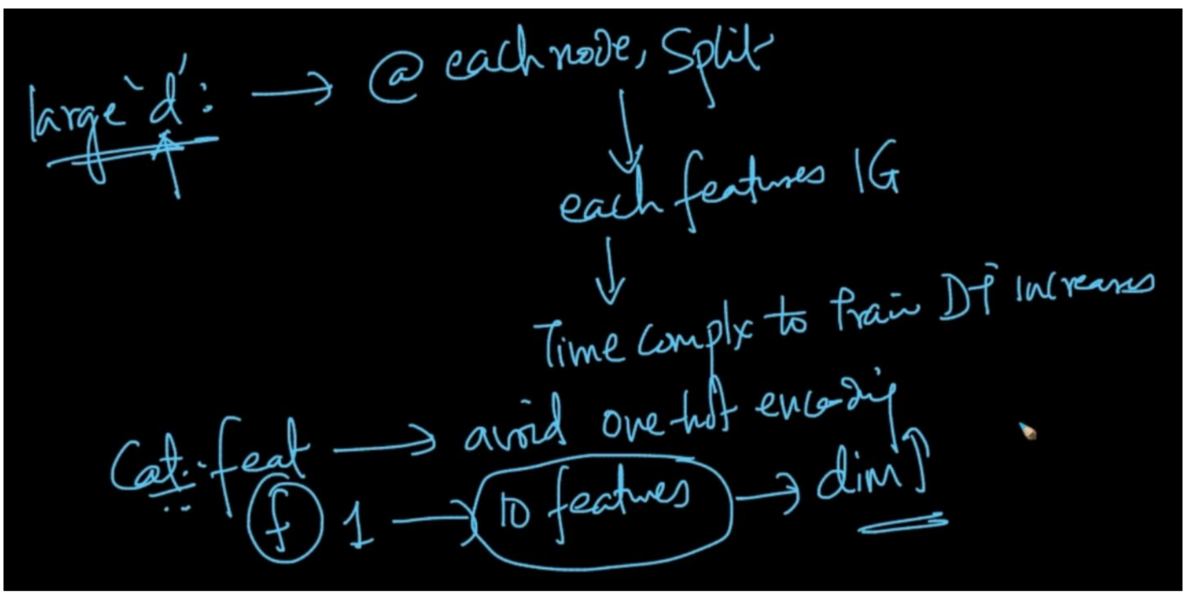

In classification we wanted to reduce Entropy :- "0" and in Regression we wanted to reduce MSE/MAE - 0




The red smooth curve is the ideal function but in Decision Trees when depth = 2 we get blue which is underfitting and depth = 5 (green) which is overfitting. In Classification and Regression with Decision Trees lines are axis-parallel

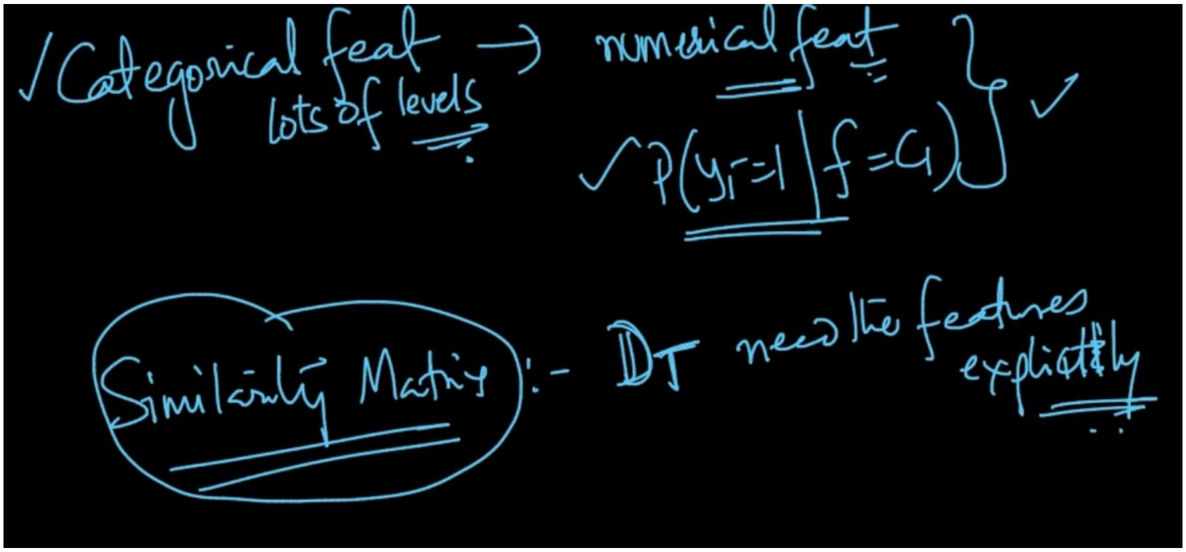
**CASES**


If we’ve Imbalanced data we need to balance it because it impacts Entropy Calculation



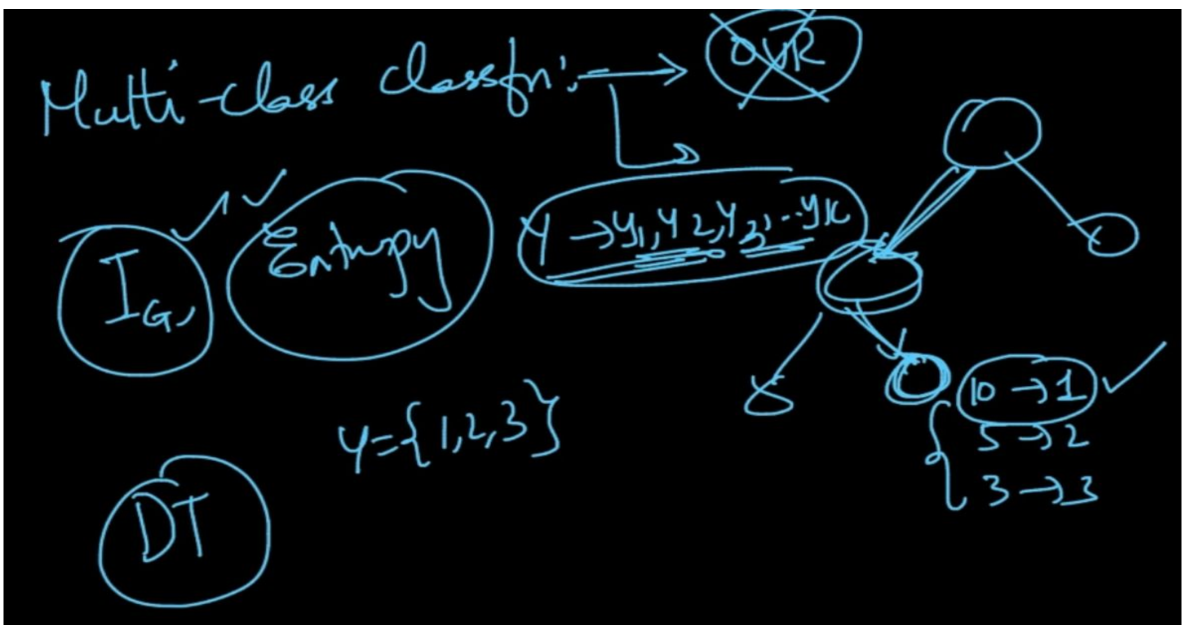

If we’ve large ‘dimensions’ we need to split for every feature for IG and if ‘d’ is large then Time complexity increases dramatically

If we’ve Categorical Features we need to avoid one-hot encoding because suppose we’ve a categorical feature f with 10 distinct values with one-hot encoding it’ll be converted to 10 features



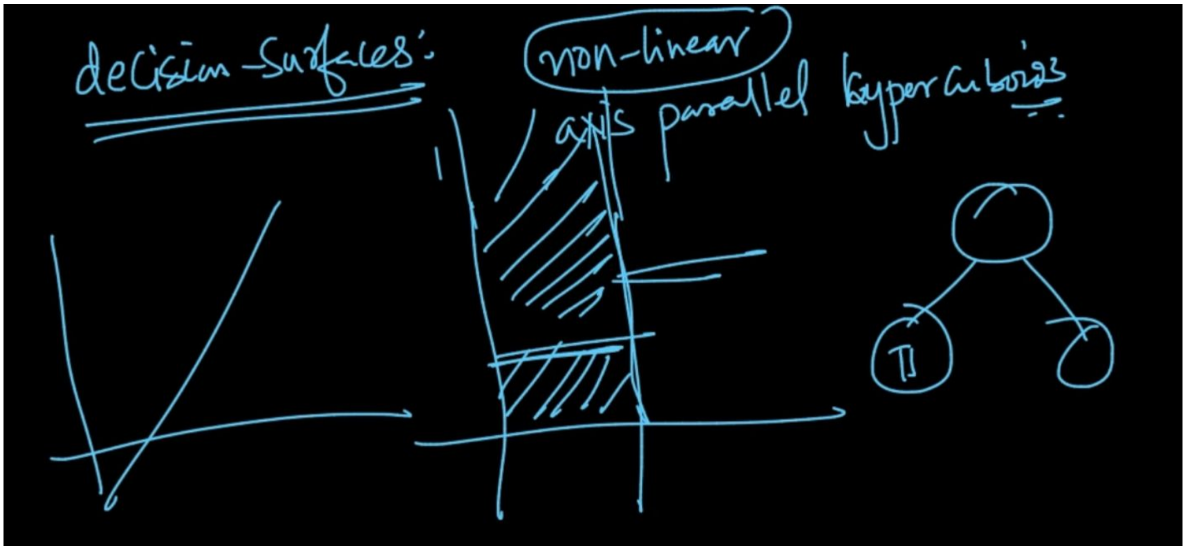

So if we've Categorical features with lot of categories we convert them into a numerical feature P( y = 1 | f = $C_1$ ) where C is category

We can't use Similarity Matrix since DT needs the features explicitly for Entropy calculation

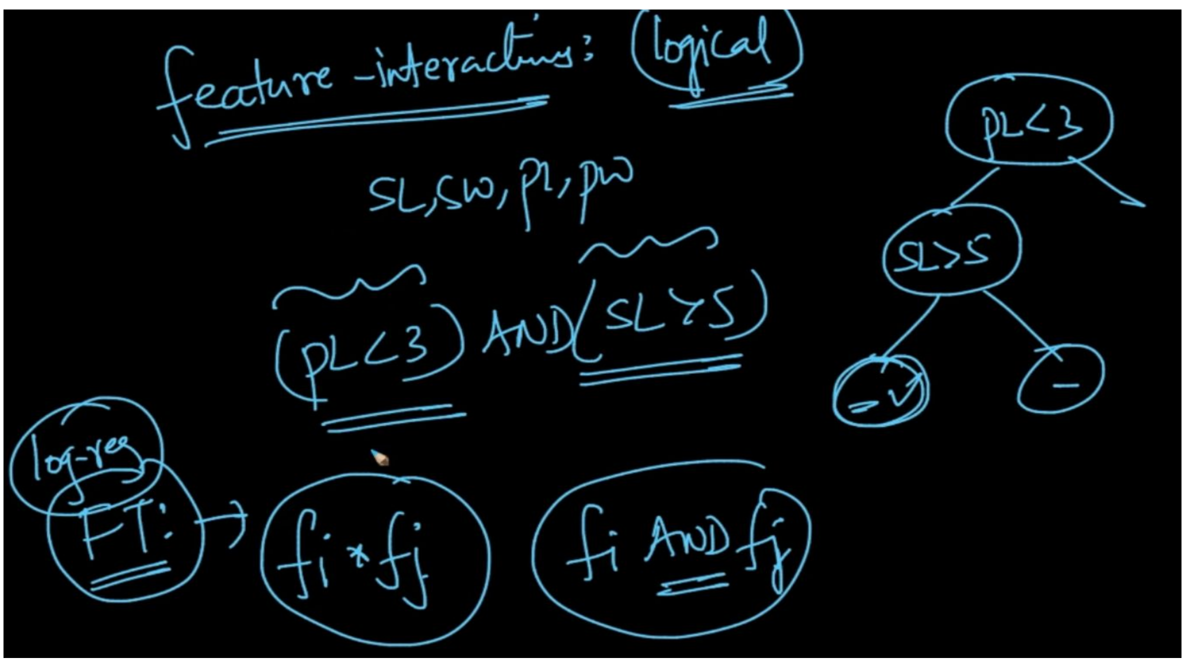



For multiclass classification we don't need to do One v/s Rest because suppose y = {1,2,3}

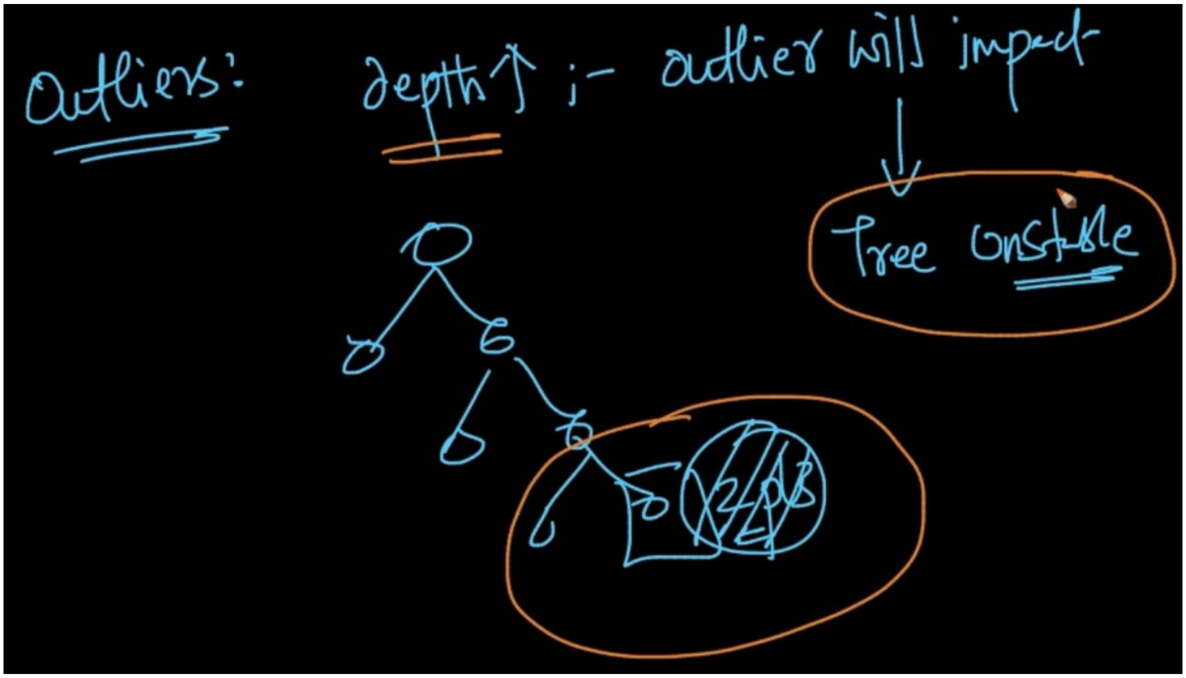
So if a query arrives at a leaf where 1 -> 10 pts, 2 -> 5 pts and 3 -> 3 pts it's y = 1 since it has majority



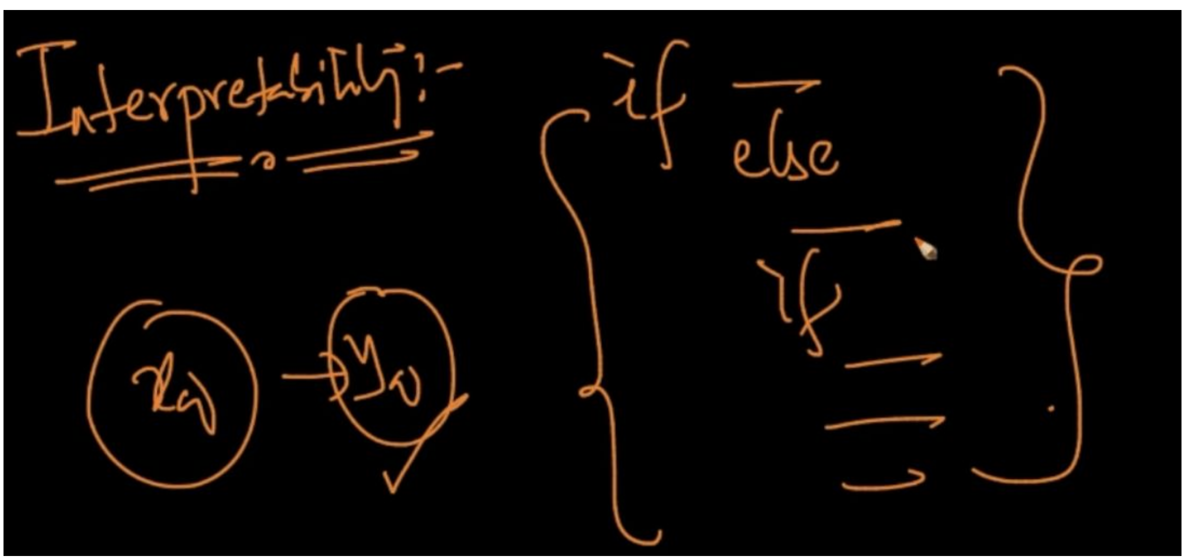

Decision Surfaces : They are axis parallel hypercuboids. Here there;s no just 1 hyperplane separating but multiple hyperplanes where each decision surface corresponds to Decision nodes



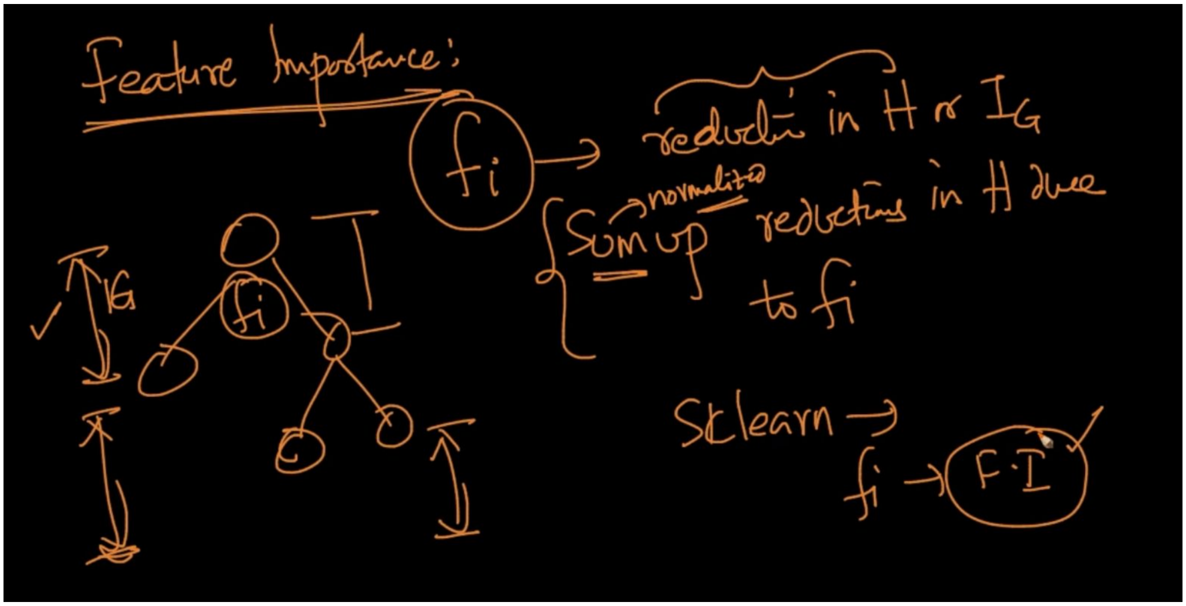

Feature Interactions : If a new query arrives at leaf nodes with (PL < 3 ) AND (SL > 5)
Then features PL, SL are interacting with each other




If the depth of the tree increases then Outliers affect Decision Trees




Decision Trees are super Interpretable since it's just Nested if-else which are very easy to understand if the depth is not too large




Feature Importance : A feature $f_i$ tells us about reduction in IG or H . We sum up reductions in H due to $f_i$. SO whichever features gives the maximum reduction is an important feature