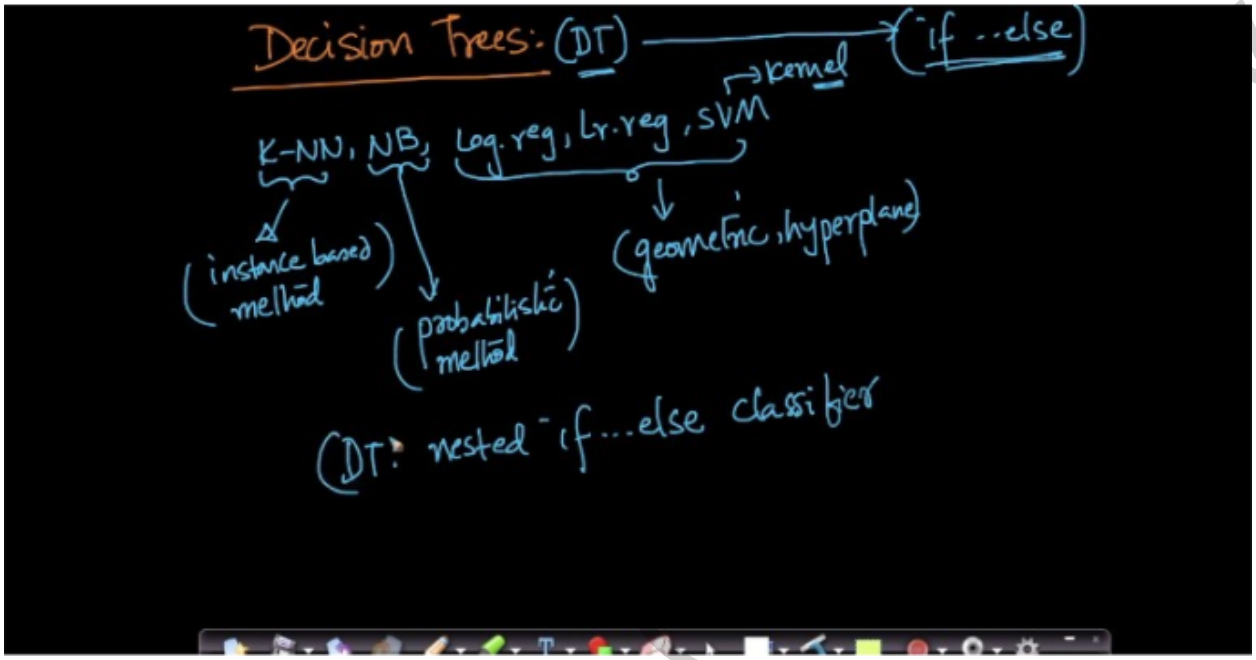
In this chapter we will look at an algorithm called Decision Trees

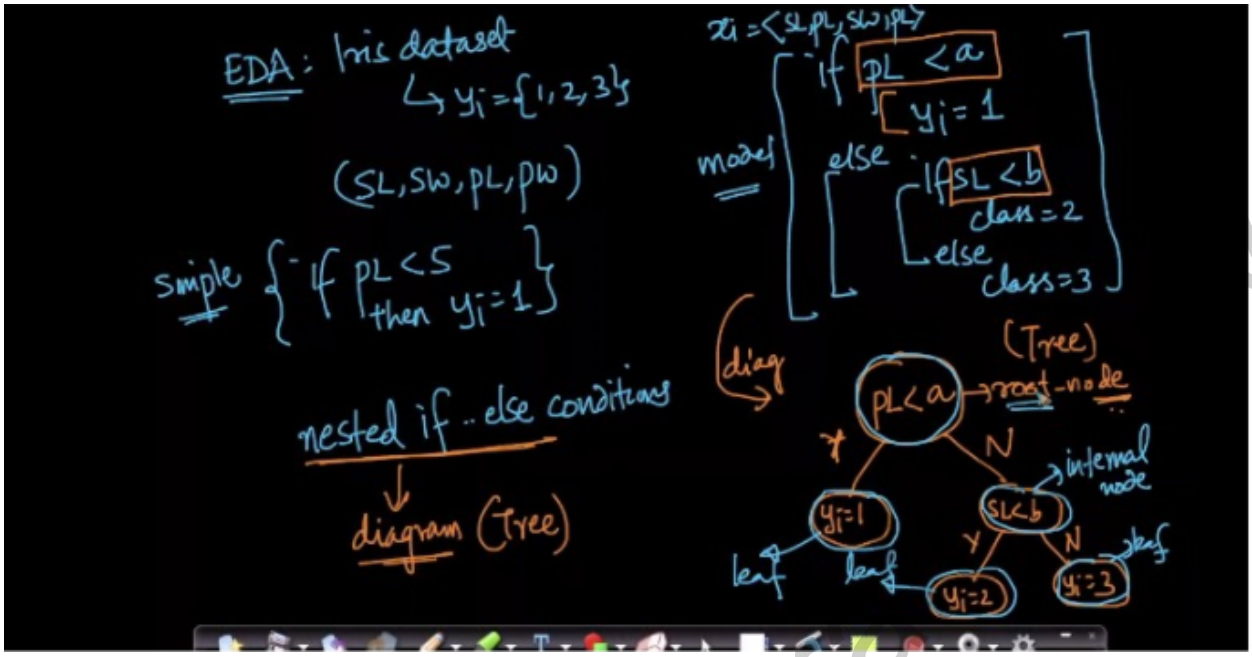
37.1 Geometric Intuition of decision tree: Axis parallel hyperplanes



Timestamp 2:19

Till now we have seen many different types of algorithms based on different ideas. Like KNN is an instance based algorithm, it is mostly heuristics. Naive Bayes is a probabilistic method. Algorithms like logistic regression, linear regression, SVM are mostly geometric in nature.

In this chapter we will look at another type of algorithm called Decision Trees which behaves like a nested if else classifier.



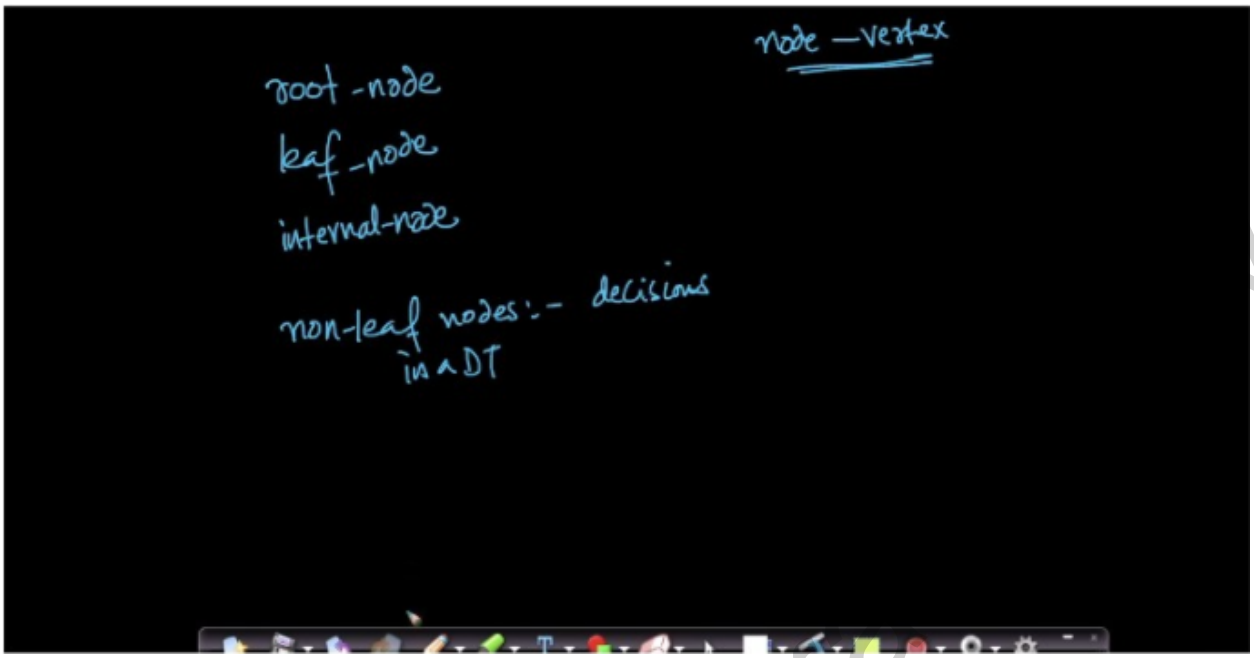
Timestamp 8:02

Let's take a look at the iris dataset that we saw during EDA. In the iris dataset we have y_i ϵ { 1, 2, 3 }. It contains 4 features: Sepal Length(SL), Sepal Width(SW), Petal Length(PL) and Petal Width(PW).
Now for this dataset we can create a very simple rule, say if the PL is less than 5 then y_i = 1.
We can write this in the if else format as

if PL < 5:
    then y_i = 1

This type of rule is called a if else rule.
Based on our dataset we can create a nested rules of if else as shown in the image above. This nested if else rules can also be effectively shown using a data structure called trees.

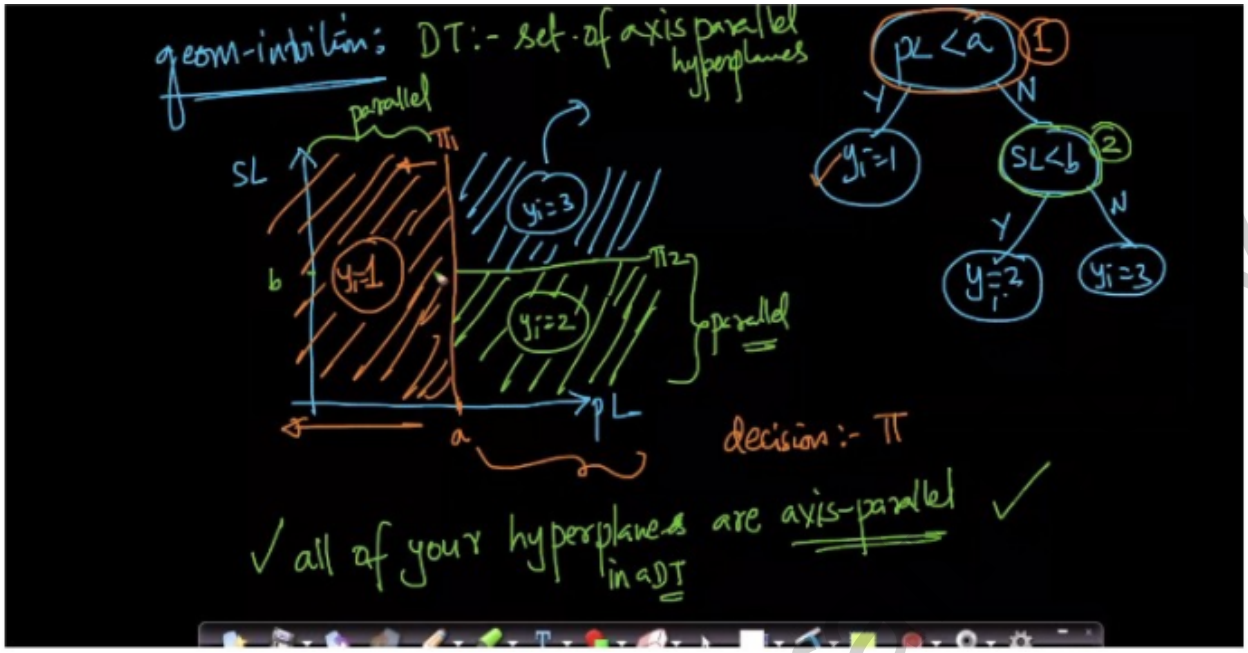
We can see in the image above how we have converted our nested if else rules into a tree. At each node of the tree we are essentially checking a condition similar to the if checks and based on the result of this checking we are deciding which path to take.



Timestamp 8:40

Let's briefly discuss trees.
A node is a structure which may contain a value or condition. Each node in a tree has zero or more child nodes. Nodes are often called vertices.
A root node is the first node in a tree.
Leaf nodes are the last nodes of a tree.
Any node which is not a leaf node is called an internal node.

In Decision Trees all the decisions are taken in non-leaf nodes.
Decision Trees are highly interpretable because we can read the conditions in simple plain english.



Timestamp 15:03

Let's now look at the geometric interpretation of Decision Trees.
Consider the same Iris Dataset that we saw earlier but with only two features, say Petal Length(PL) and Sepal Length(SL).Now we can make a decision tree using these features as shown in the image above.

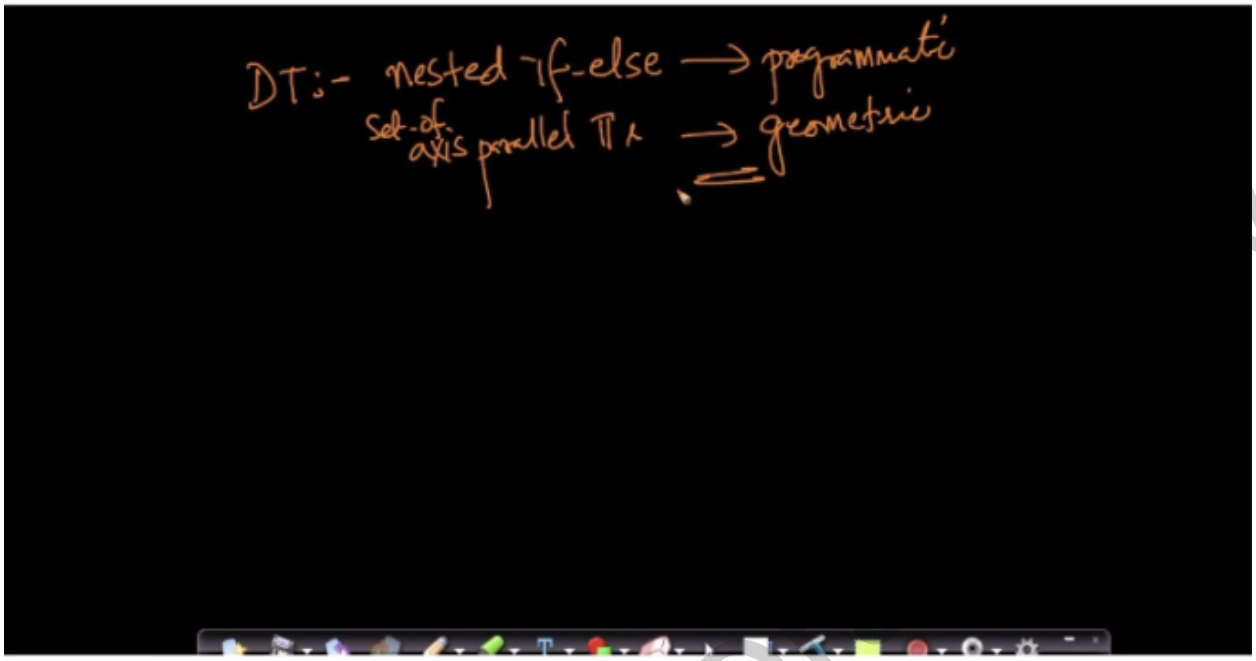
Let's draw these features i.e, PL and SL on x - axis and y-axis respectively.

Now let's look at the root node of the decision tree where it says that if PL < *a* (some threshold) then the class label y_i = 1, otherwise we need to make another decision.

We can draw a line on PL at *a* and say that if the value of PL is less than *a* then the class label is 1.
Now say for a data point it's PL value was greater than a, then we will move to the second condition marked 2 in decision tree, it says that if SL < *b* (some threshold) then the class label y_i = 2, otherwise y_i = 3.

For this, first we will go right of the threshold i.e. PL = a, and in this region we will draw a line on SL at *b* and say that if the value of SL is less than *b* then the class label is 2 , otherwise the class label is 3. Note that this line didn't pass through the PL = a, because we had a value of PL > *a.*

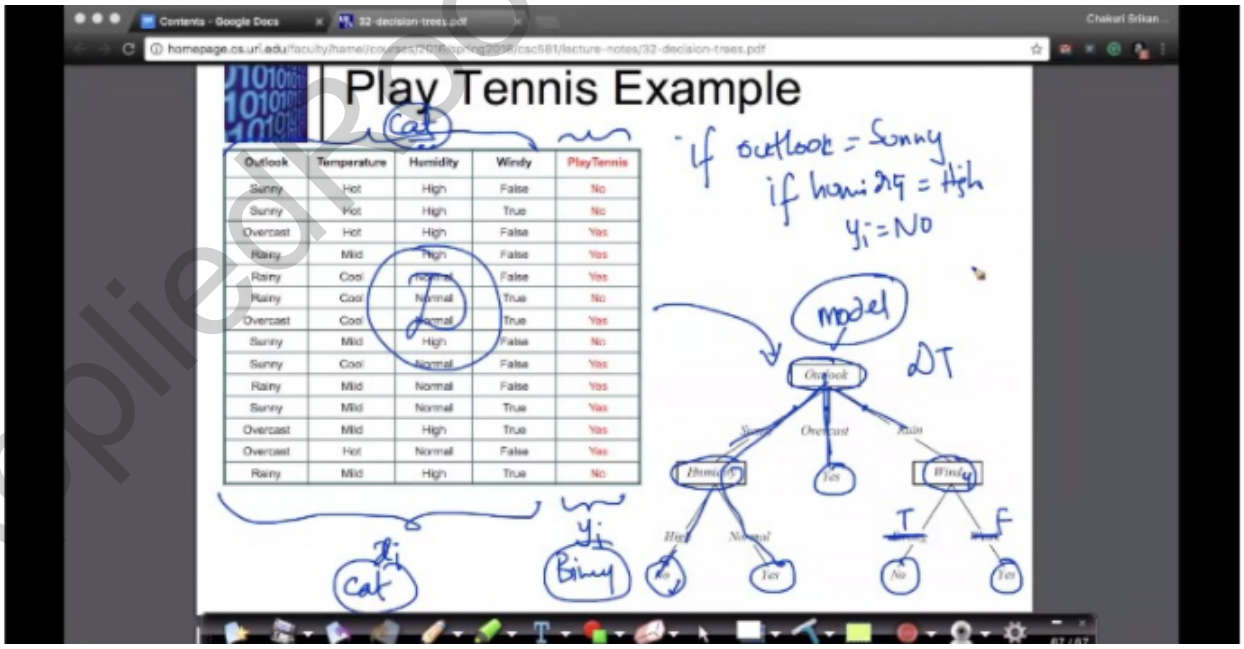
In this case we have drawn lines, but in higher dimensions we would have drawn hyperplanes. Also notice that essentially we are dividing up our space into various regions. Each region indicates different classes. These regions can be hyper cubes or hyper cuboids. All the hyperplanes that we are drawing are essentially parallel to the axes. So it can be said that geometrically Decision Trees are a set of axes parallel hyperplanes.



Timestamp 16:34

So to conclude we can say that programmatically we can think of Decision trees as nested if-else classifiers and geometrically Decision trees are a set of axis parallel hyperplanes.

37.2 Sample Decision tree

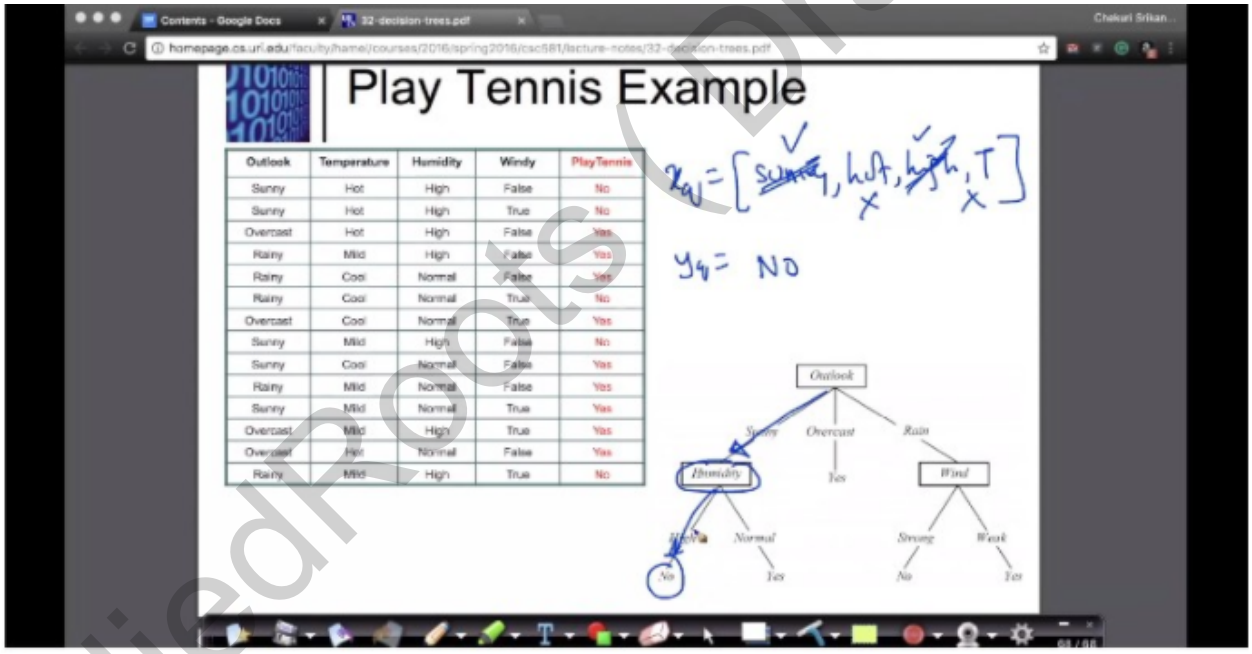


Timestamp 5:30

Here we have taken a small dataset and built a decision tree using that. The dataset can be accessed from here => https://homepage.cs.uri.edu/faculty/hamel/courses/2014/spring2014/csc581/lecture-notes/31-decision-trees.pdf

It's a binary classification problem with target variable play, $y_i \epsilon$ {Yes(Y), No(N)}. Our dataset contains 4 features => Outlook, Temperature, Humidity, Wind. All these features are categorical features.
Outlook can have 3 categories: sunny, overcast, rain. Temperature can have 3 categories: hot, mild and cool. Humidity can have 2 categories: high, normal. Wind can have 2 categories: strong, weak.

Now the decision tree built can be seen on the image above. At the root node we have the feature outlook, since it contains three features it has three paths one each for sunny, overcast and rain. Let's see the leftmost path (a path is essentially a set of connected nodes), it says that if the outlook is sunny and then if the humidity is high we cannot play tennis.

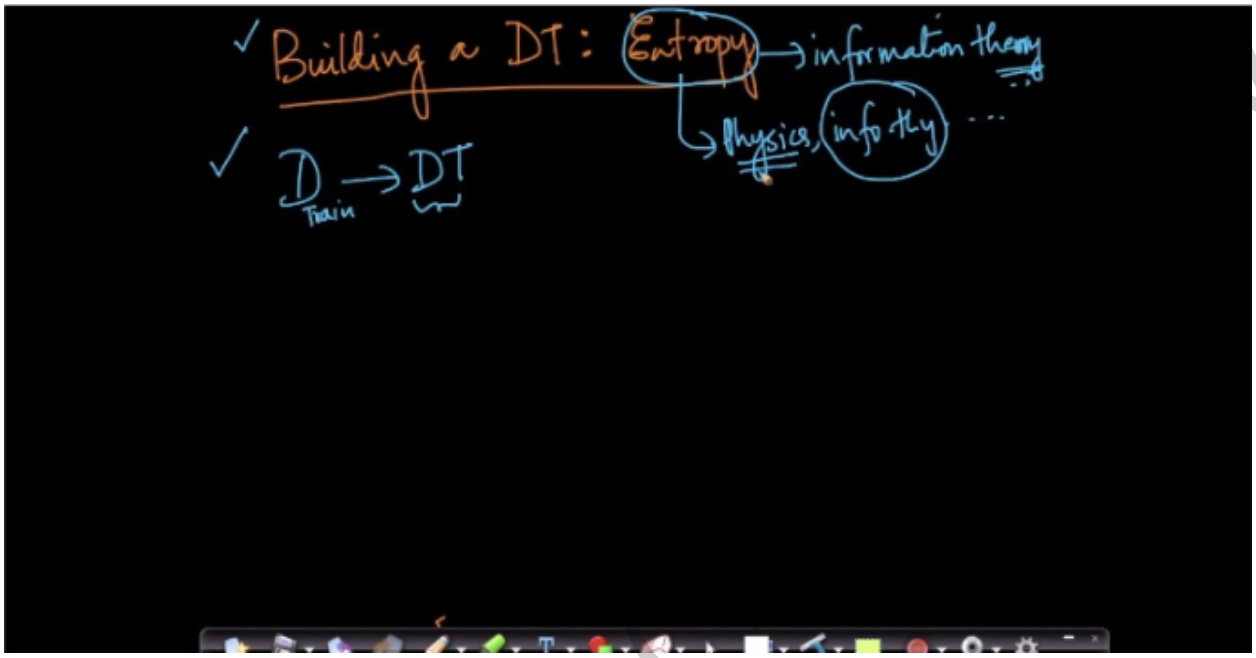


Timestamp 6:32

Now during test time suppose we get a query point x_q = [sunny, hot, high, strong]. We will take this query point and pass it over the decision tree. At the root node we have the feature outlook, our data is sunny so we take the left path, then the feature is humidity, our data is high so we take the left path, reaching a leaf node. The value of the leaf node is No, so we cannot play tennis.

So now we know how to predict a data point if we know the decision tree. The challenge is to build a decision tree.

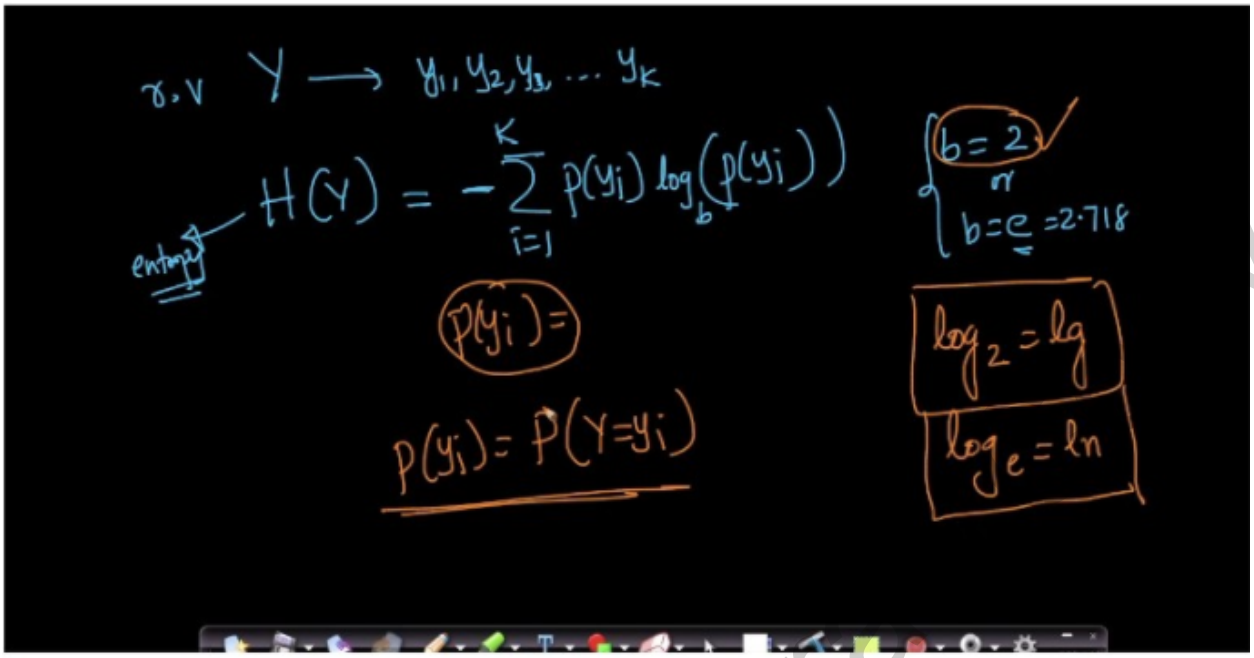
37.2 Building a decision Tree:Entropy



Timestamp 1:17

Here we will learn how to build a decision tree. Before doing that we need to learn about a few concepts.

Entropy is used in various disciplines of science, like physics, information theory, electronics,etc. We will focus more on the information theory's interpretation of entropy.



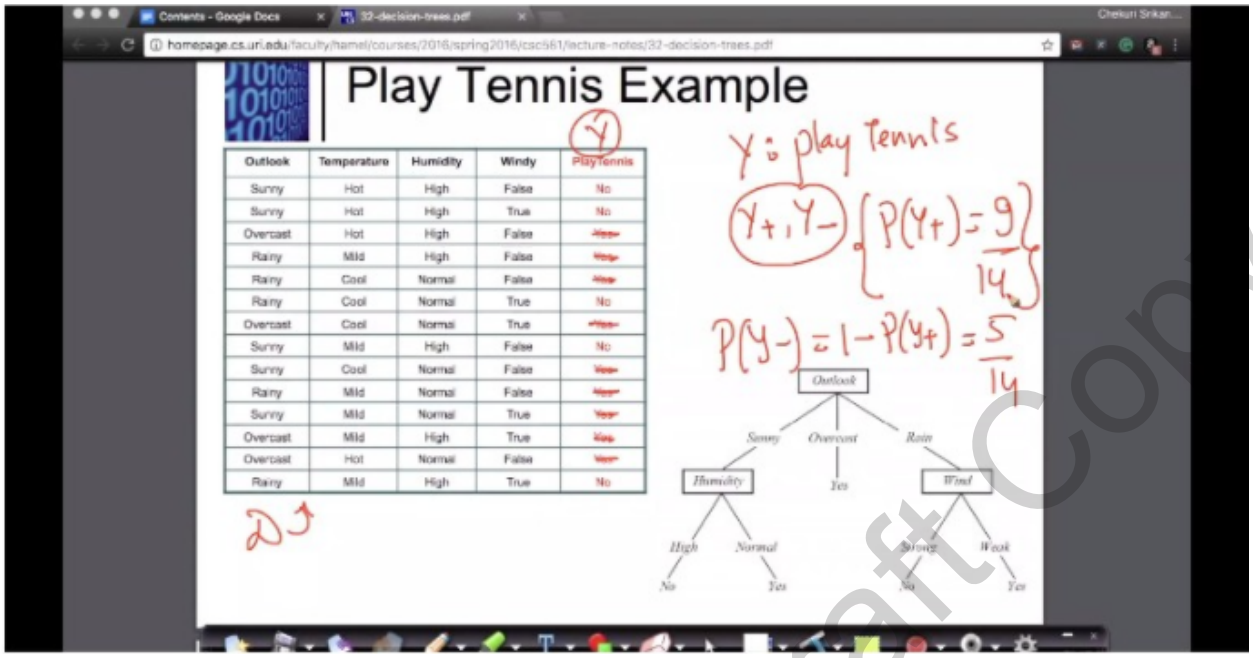
Timestamp 3:29

Let Y be a random variable which can take k values => { y_1, y_2, y_3, ... y_k}. Entropy which is usually represented by H for this random variable will be,

$$H(Y) = -\sum_{i=1}^{k} P(y_i) \times log_b(P(y_i))$$

Here, b is the base which we usually take as 2 or e(euler no) = 2.718. Typically we use base 2. Base 2 logarithm is typically written as lg , whereas for base e we write it as ln.

$P(y_i)$ represents the probability of the event where the random variable y_i occurs.

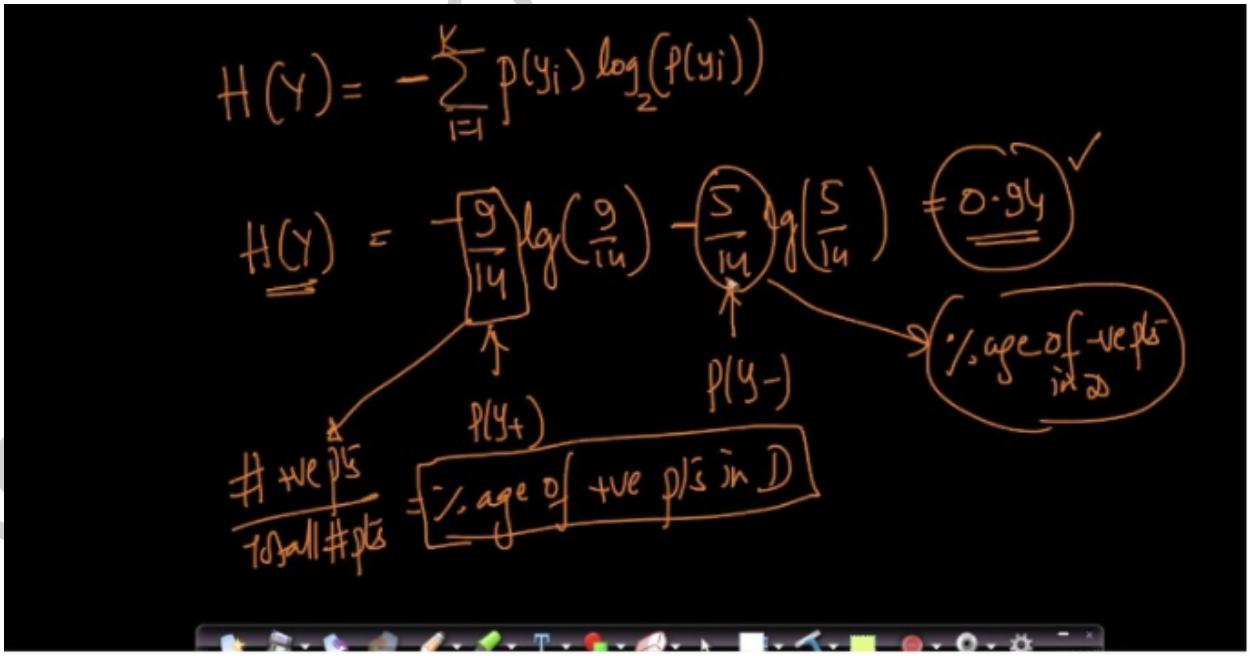


Timestamp 4:34

In our tennis dataset the target variable => Play Tennis(Y) can take two values i.e. {Yes, No}. In this dataset we have a total of 14 rows out of which 9 are Yes and 5 are No.

So, P(Y_+) = 9/14 [probability of yes]

and, P(Y_-) = 1 - P(Y_+) = 5/14 [probability of no]



Timestamp 6:27

So, the entropy for the random variable(Y) :

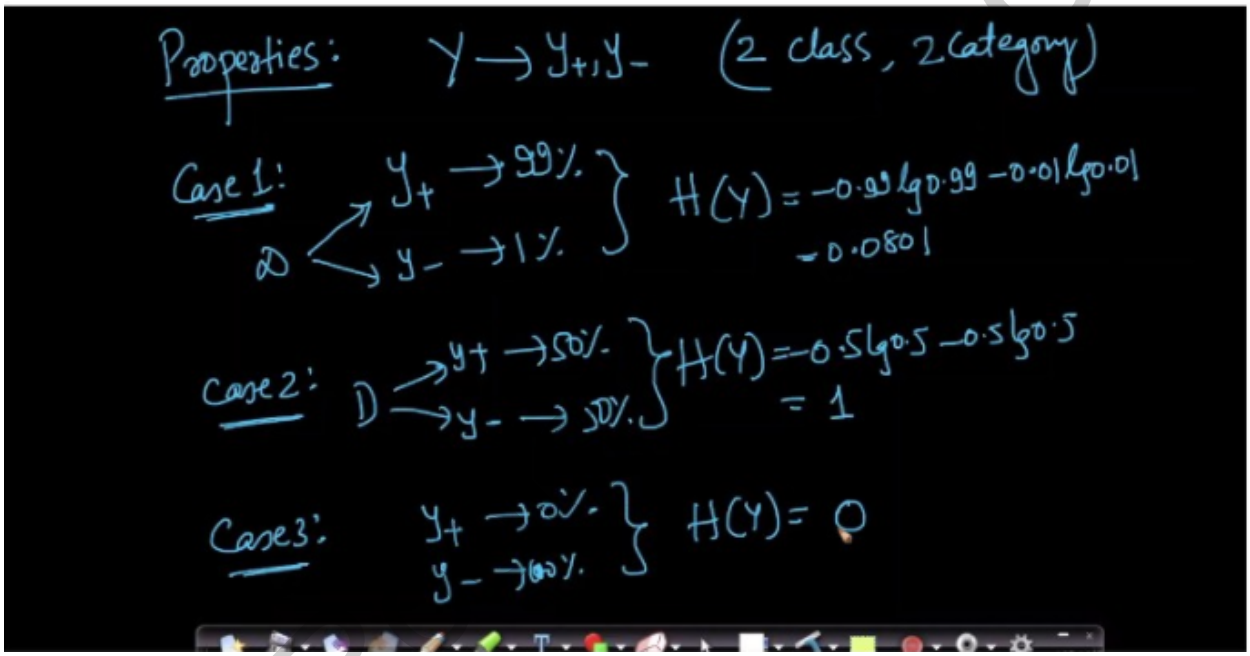
$H(Y) = -\sum_{i=1}^{2} P(y_i) \times log_2(P(y_i))$

= -9/14 * lg(9/14) - 5/14 * lg(5/14)

= 0.94

9/14 represents the ratio of positive points in our dataset($P(Y_+)$).

5/14 represents the ratio of negative points in our dataset($P(Y_-)$).



Timestamp 8:32

Now we will look at some of the important properties of entropy, we will see it in the 2 class case say Y → (Y_+, Y_-), however these will hold true in multiclass setting as well:

Let's see how the value of entropy changes with changes in the distribution of data.
case 1: Say in our dataset D , Y_+→ 99% and Y_-→ 1%. In this case entropy

H(Y) = -0.99 * lg0.99 - 0.01*lg0.01 = 0.0801

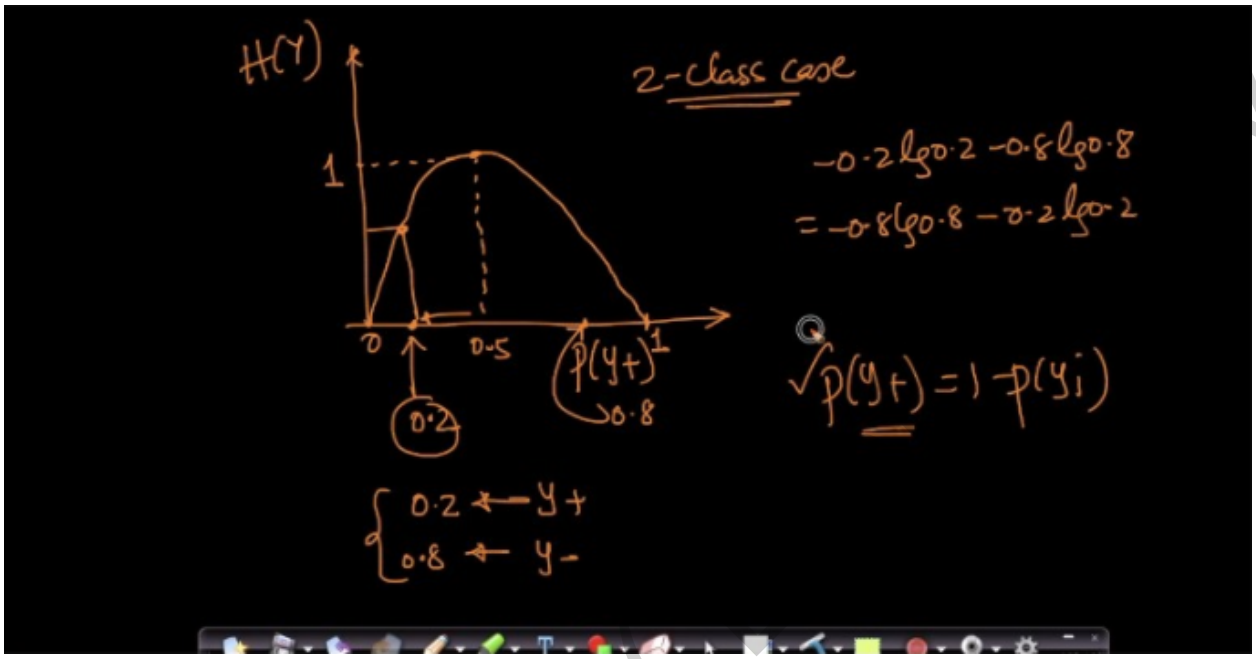
case 2: Say in our dataset D , Y_+→ 50% and Y_-→ 50%. In this case entropy

H(Y) = -0.5 * lg0.5 - 0.5*lg0.5 = 1

case 3: Say in our dataset D , Y_+→ 0% and Y_-→ 100%. In this case entropy

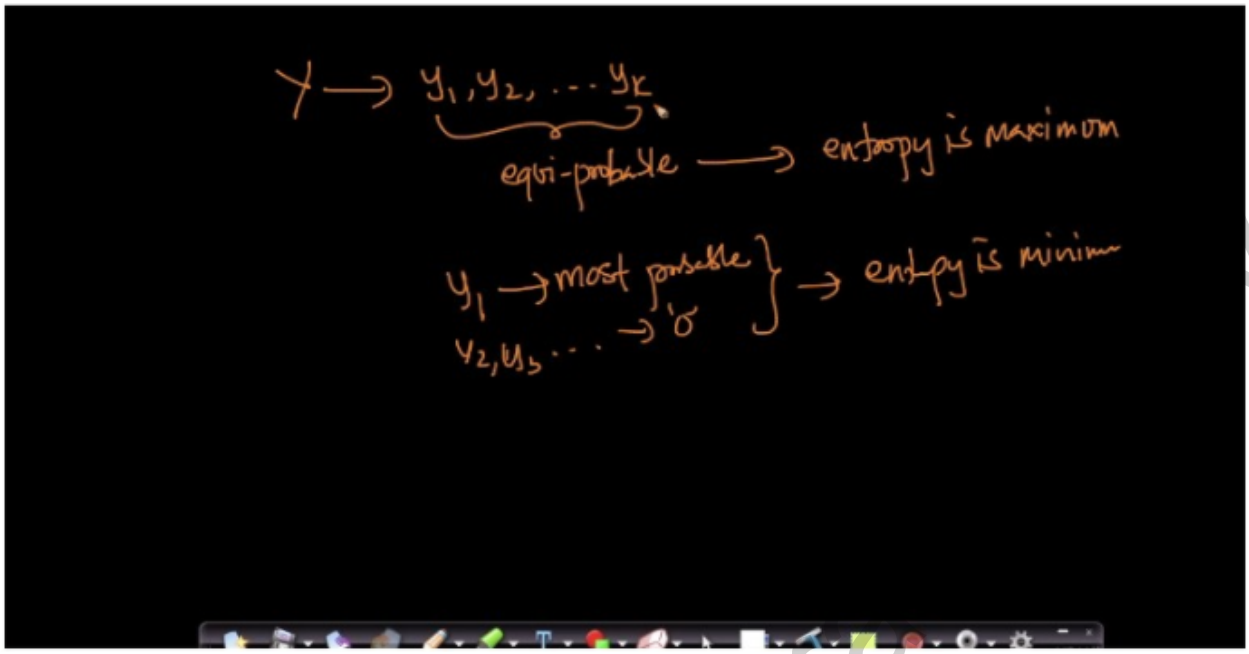
H(Y) = 0

We can see that the entropy value is maximum when the dataset is equally distributed. If one class fully dominates the other class like in case 3 the entropy is 0.



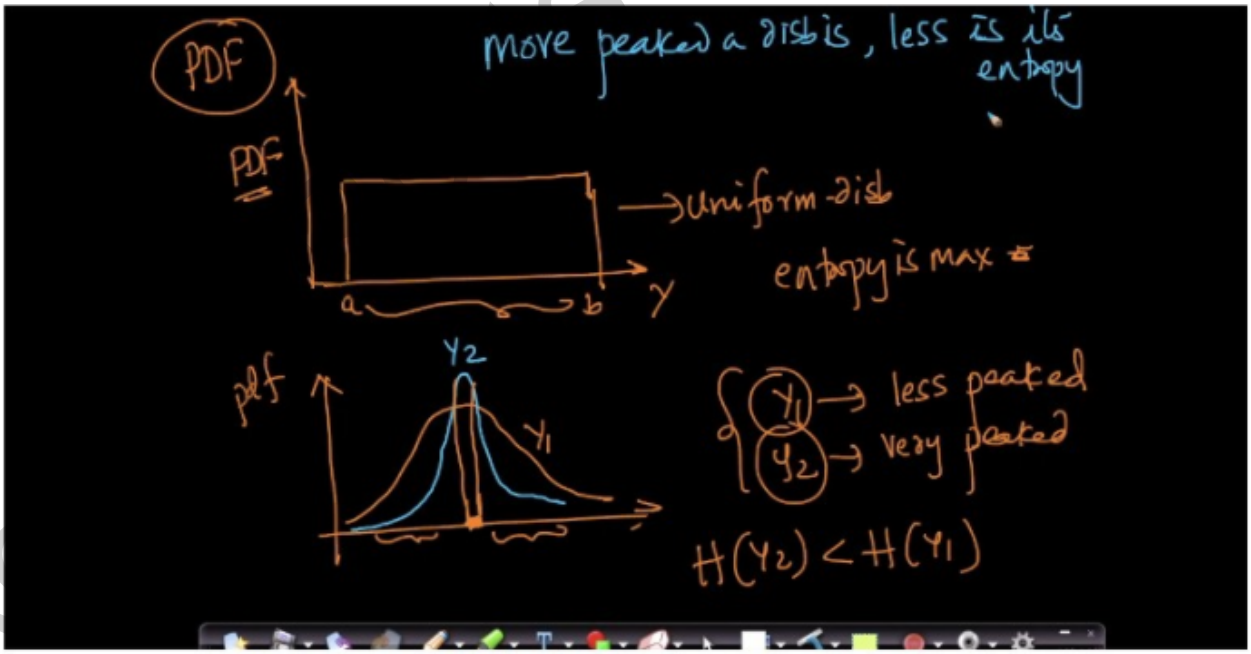
Timestamp 11:41

We can draw a plot for the probability and entropy as shown in the image above. We can see that the entropy is maximum at probability 0.5. If one class has probability 0.5, the other class will also have a probability of 0.5, since P(Y_+) = 1 - P(Y_-). Also note that the graph is symmetric.



Timestamp 12:55

Similarly in a multi class setting where a random variable Y can take multiple values say $\{ y_1, y_2, y_3, \ldots y_k \}$, the entropy will be maximum when all the values are equi-probable. Also the entropy will be minimum when one class heavily dominates all others.



Timestamp 17:40

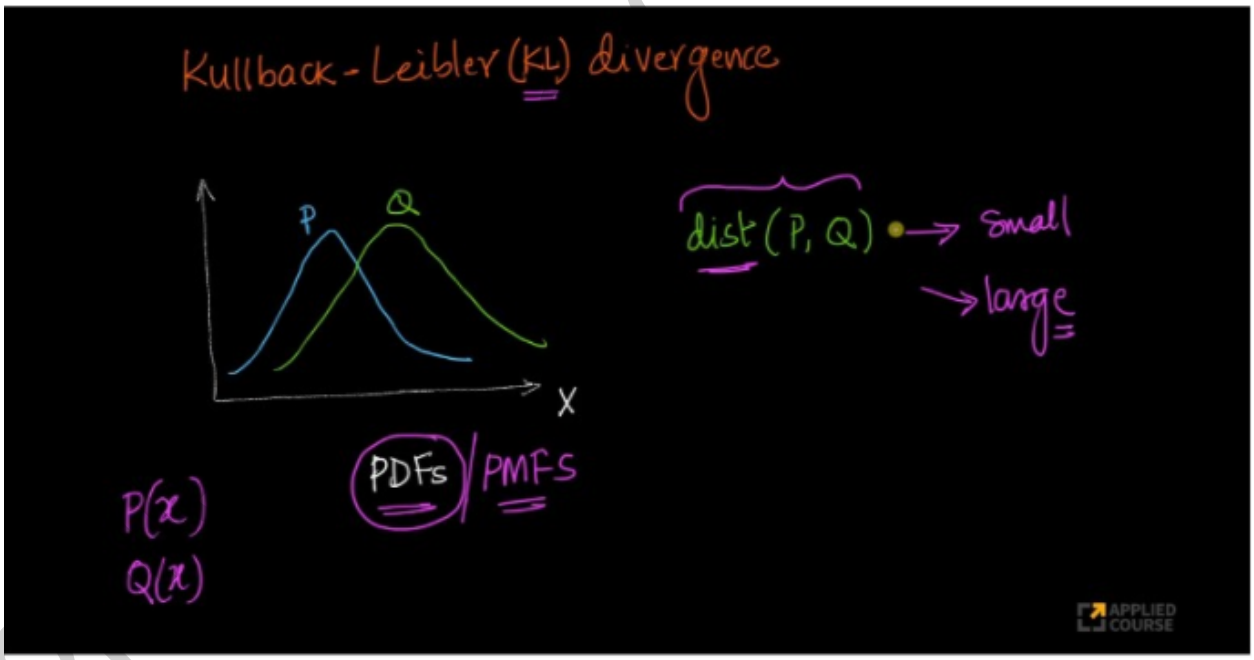
Now let's look at entropy from a probability density point of view.

Say we have a random variable Y which follows a uniform distribution, as shown in image above. Now we know that in uniform distribution all the values are equally likely. That means here we have maximum entropy.

Now let's see another case. Consider two random variables Y_1 and Y_2 both of them following a distribution as shown in the image above. We can see that Y_2 is much more peaked than Y_1, this means that Y_1 is much closer to a uniform distribution than Y_2. This means Y_1 will have a larger entropy than Y_2.

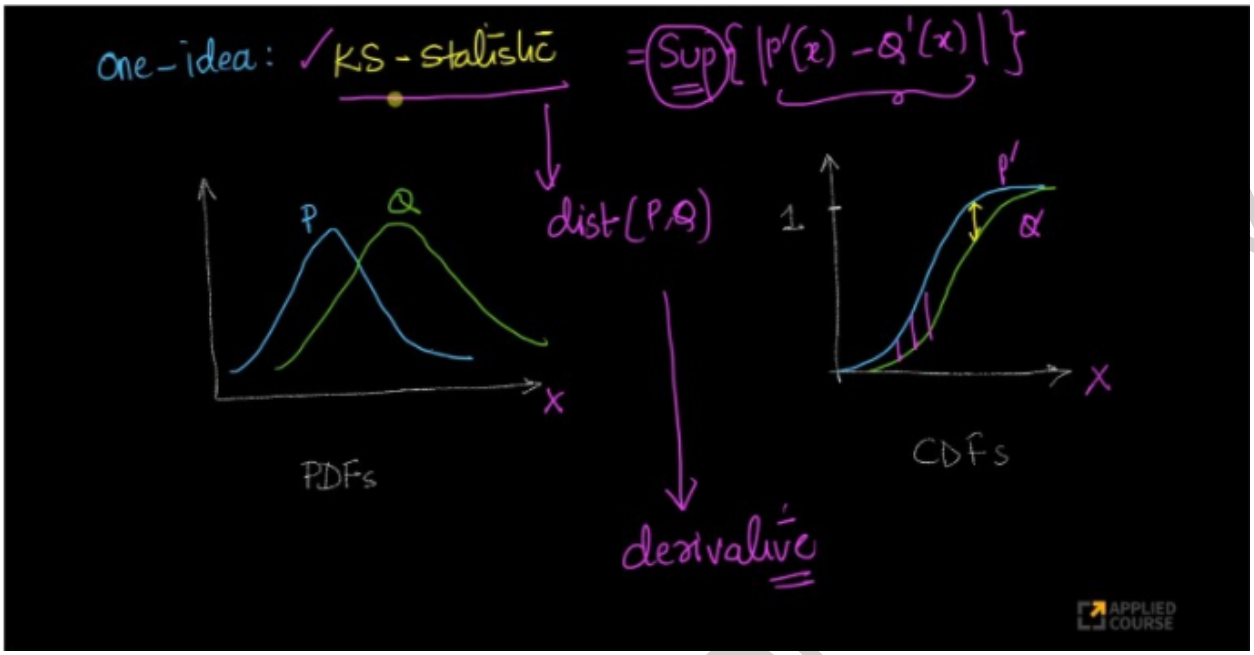
Entropy plays a major role in building Decision Trees, which we will see when we build a decision tree from scratch.

37.4 KL Divergence



Timestamp 2:07

Consider two random variables P and Q. We can see their pdf's in the above image. Now, what we are interested in is finding how similar these distributions are, i.e. we want to find dist(P, Q). If this quantity is small then it means that they are very similar and if this quantity is large then it means that they are very different.



Timestamp 6:11

In the statistics chapter we learnt something known as KS statistic which can be used to find out the difference between two pdf's.

Let's see how we can use KS statistic to determine the distance between two pdfs.

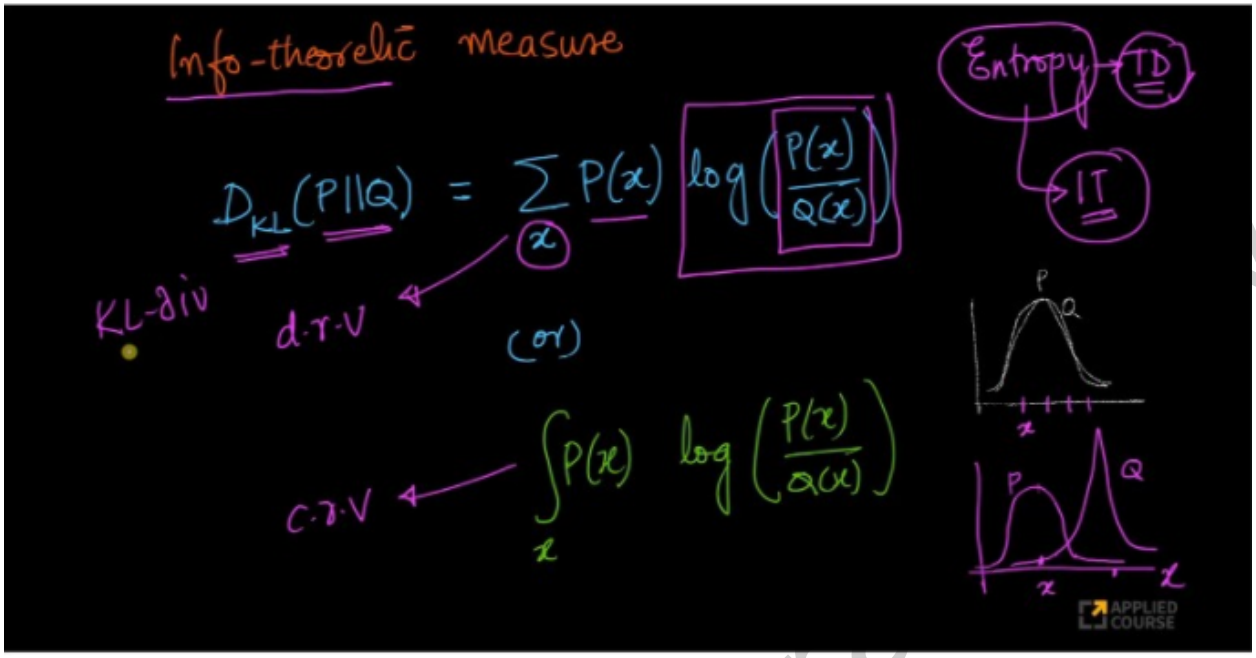
First we draw the cdf's of the random variables P & Q. Let's call it $P^{|}$and $Q^{|}$ as shown in the image above.

After this we compute the quantity => sup {|$P^{|}$(x) - $Q^{|}$(x) |}.

This quantity basically takes the absolute differences between the values of $P^{|}$(x) and $Q^{|}$(x), and then select the maximum among them.

It is very effective in calculating the similarity between two pdfs, but there is one issue with this, it is not differentiable.

In many of the ML algorithms when we work with optimization problems we want differentiable functions.



Timestamp 10:00

This brings us to KL divergence. It has its origins in Information - Theory.

KL divergence between two random variables P and Q can be defined as,

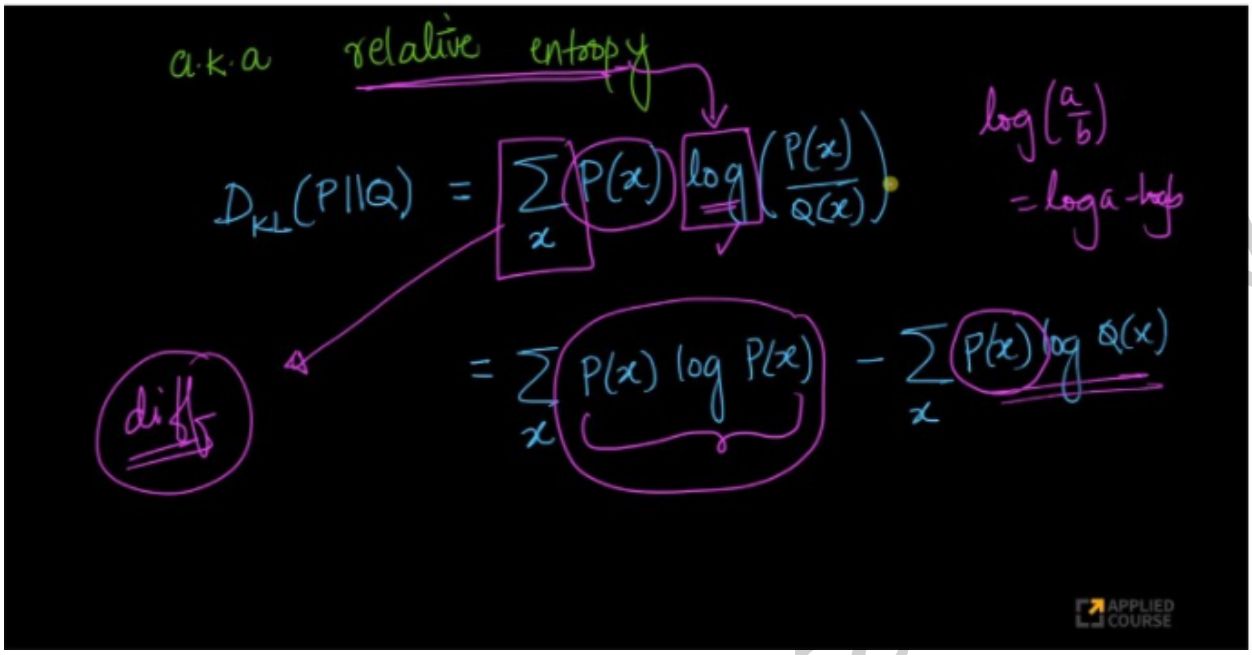
D_{KL}(P || Q) = $\sum_x$P(x) log(P(x)/Q(x)) [for discrete case]

or

$\int_x$P(x) log(P(x)/Q(x)) [for continuous case]

If PDF's of two random variables are very similar, like as shown in the image above, then the quantity P(x)/Q(x) ≈1. If this becomes close to 1, log becomes close to 0. So, the term D_{KL} (P || Q) becomes close to 0, showing that these two distributions are similar.
Similarly we can verify it for the case when the distributions are dissimilar.

So, KL divergence has higher value when the difference between two random variables is large and the value is small if the difference is small.

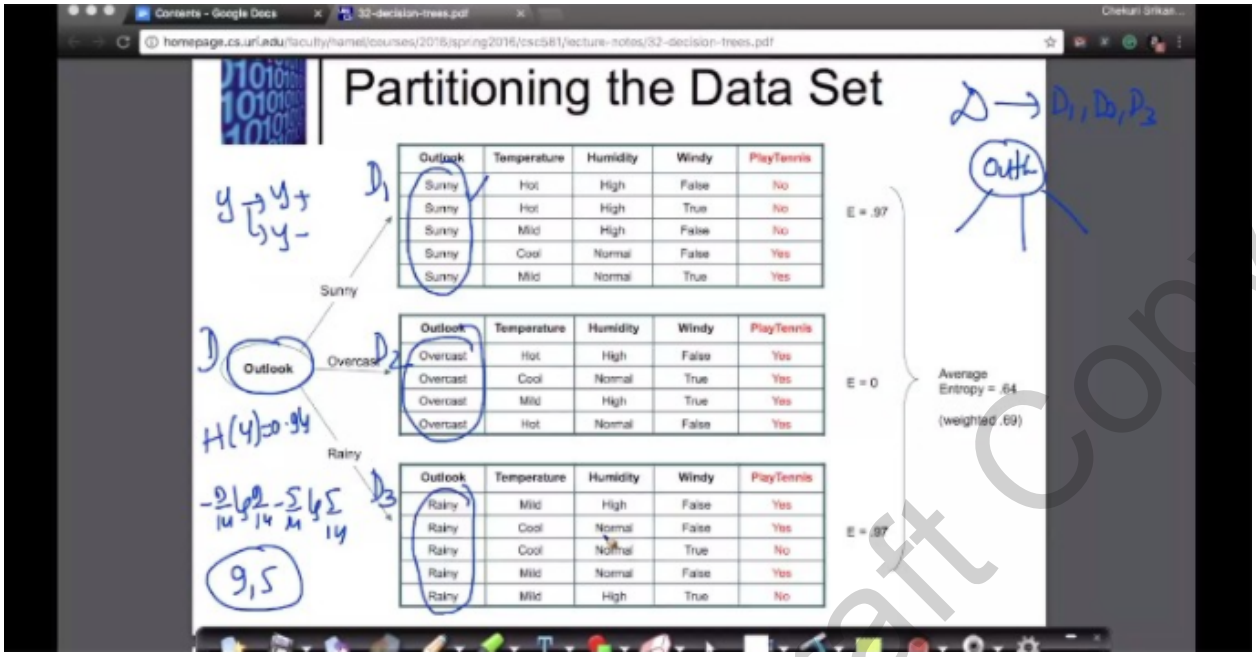


Timestamp 12:24

One can ask a question why there is a log in the KL divergence. The reason is that KL divergence is derived from entropy and entropy contains a log. In fact KL divergence is also known as relative entropy.

The most important factor of KL divergence is that it is differentiable unlike KS statistic.

37.5 Building a decision Tree:Information Gain



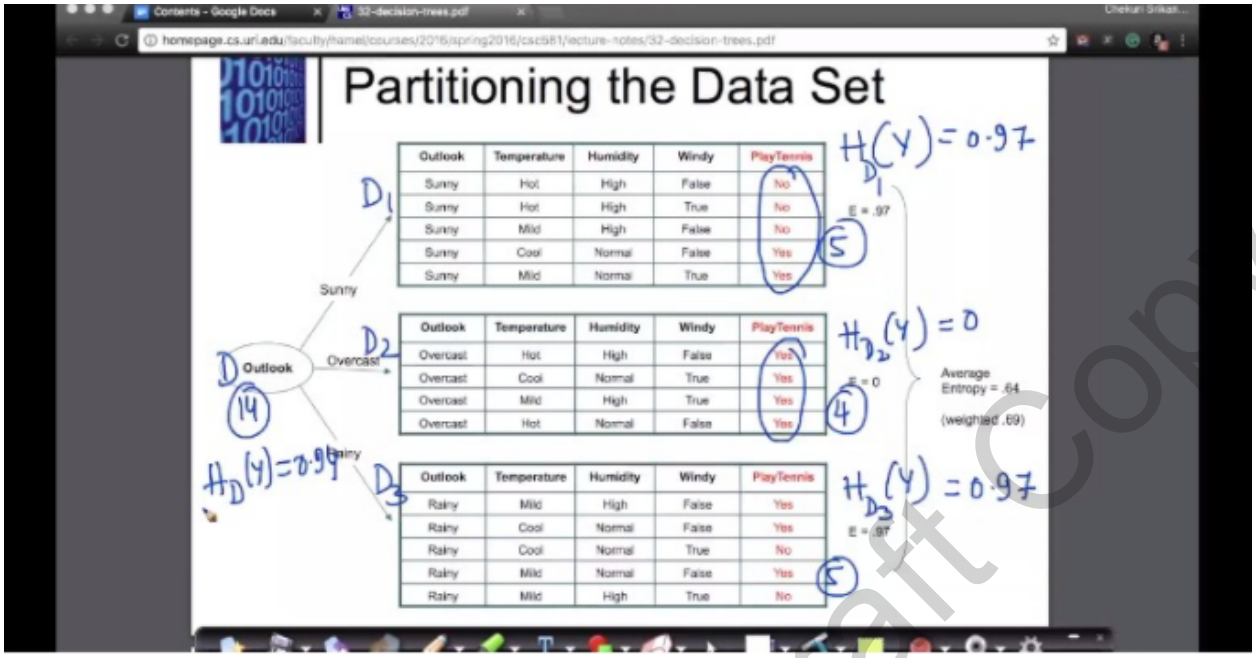
Timestamp 2:33

Let's again take the tennis dataset that we saw earlier.

Here we want to partition the dataset or split our dataset D.
Considering the feature outlook, we saw that it can have three values i.e. sunny, overcast and rainy. We will split the dataset D based on these categories. As you can see from the above image we have three sets, the first set D_1 has outlook = sunny, second set D_2has outlook = overcast and third set D_3has an outlook = rainy.

Now let's calculate the entropy value at the outlook node based on the target variable Y. We essentially want to calculate the entropy based on the target.

We had earlier calculated the entropy at outlook node H_D(Y) = 0.94



Timestamp 3:44

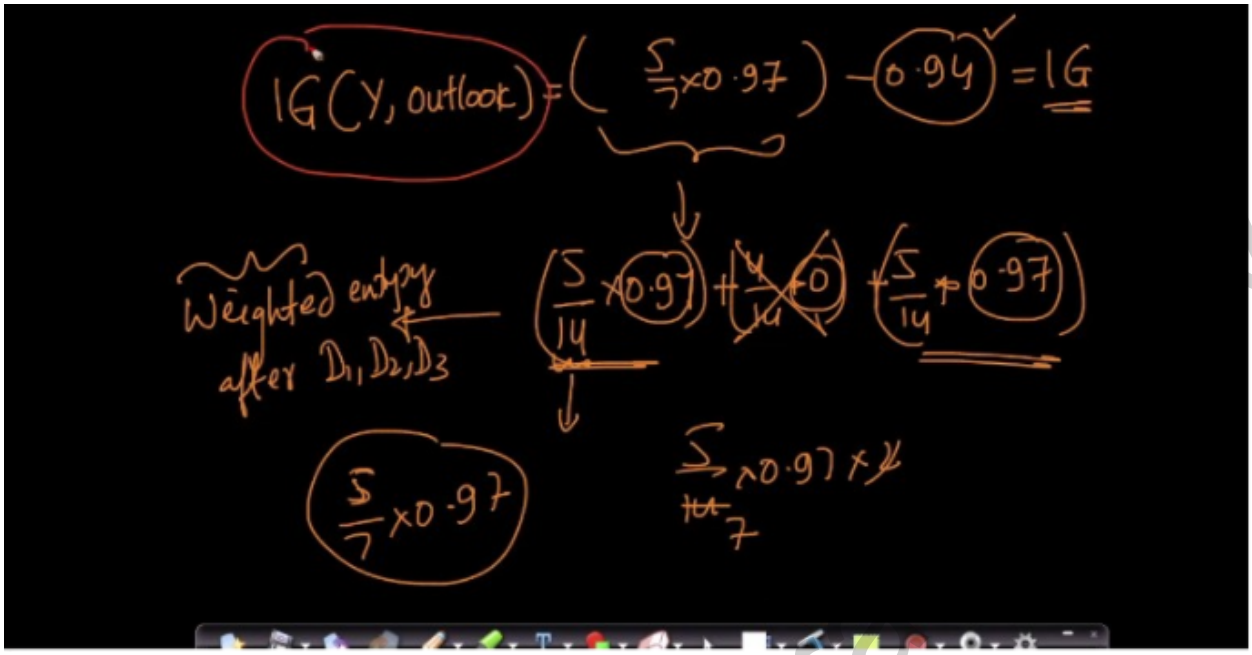
Now, we will calculate the entropy for each of the splitted datasets.

H_{D1}(Y) = 0.97

H_{D2}(Y) = 0

H_{D3}(Y) = 0.97

In total D has 14 data points. After splitting D_1 has 5 data points, D_2has 4 data points, D_3has 5 data points.

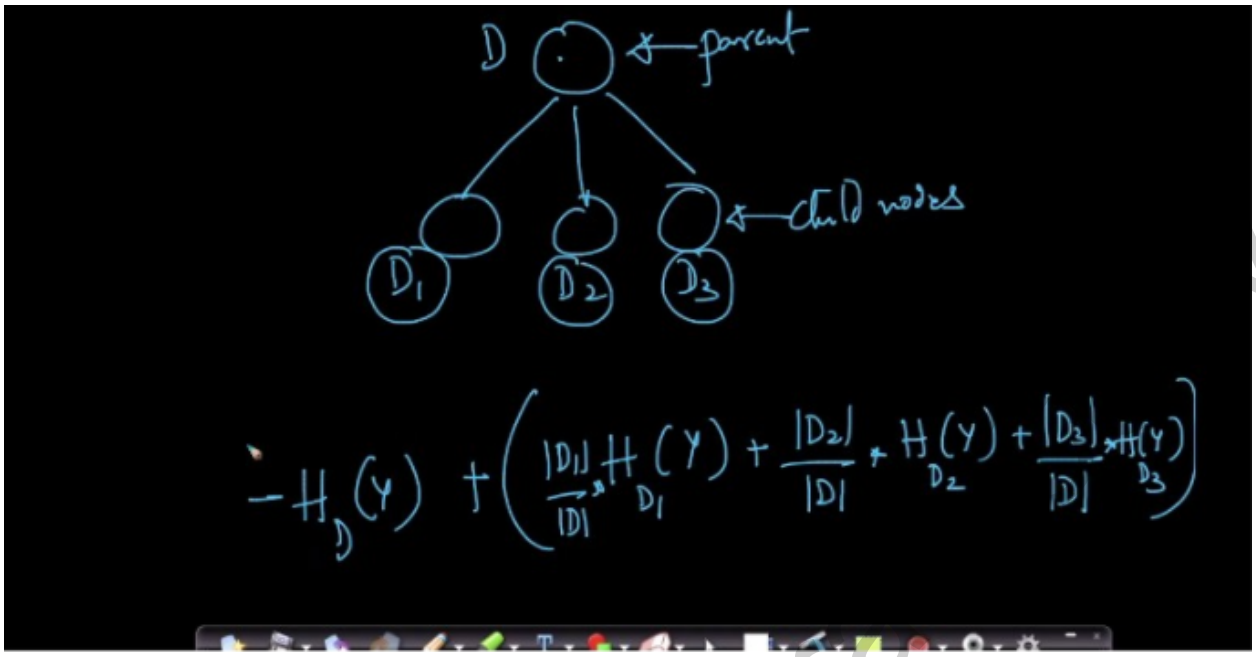


Timestamp 7:09

Note: There is a small correction in the formula from the one that we have on the slide. Corrections are reflected in the notes.

Now, Information Gain after splitting w.r.t outlook for target Y is,
IG(Y, outlook) = 0.94 - (5/14*0.97 + 4/14*0 + 5/14*0.97)

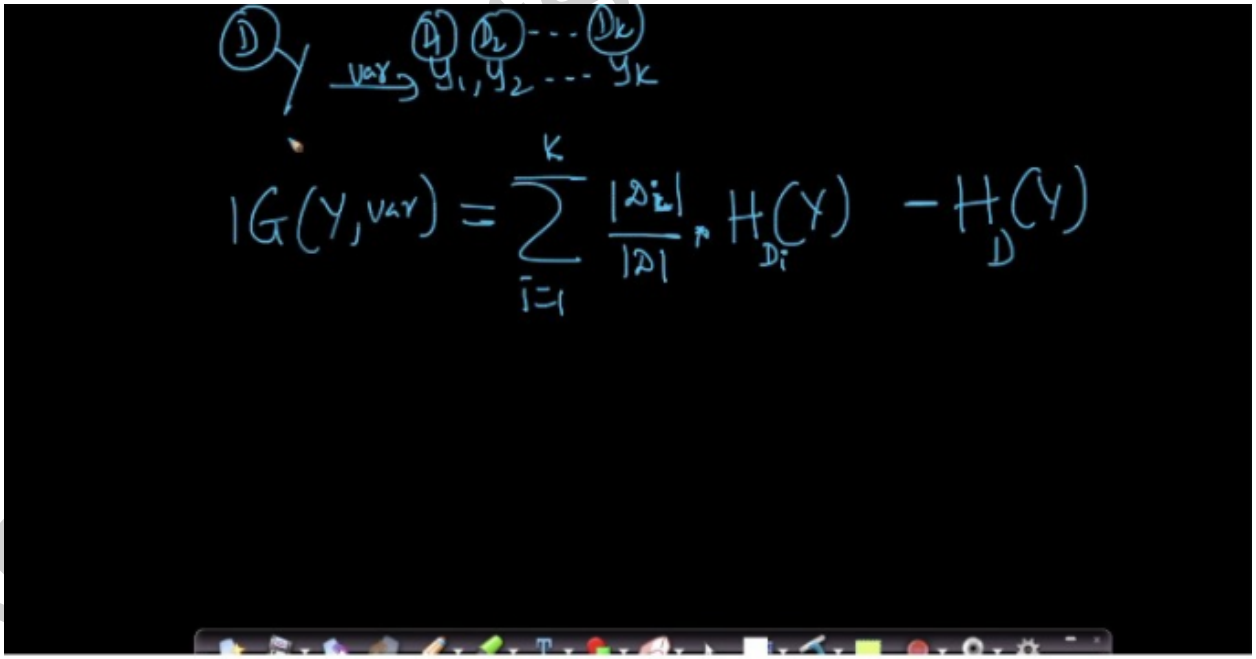
(5/14*0.97 + 4/14*0 + 5/14*0.97) => this is the weighted average of the entropy by the no of data points each set has.



Timestamp 8:25

So, for our dataset D the information gain after splitting the data into D_1, D_2, D_3 can be written as,

IG = H_D(Y) - (|D_1|/|D| * H_{D1}(Y) + |D_2|/|D| * H_{D2}(Y) + |D_3|/|D| * H_{D3}(Y))



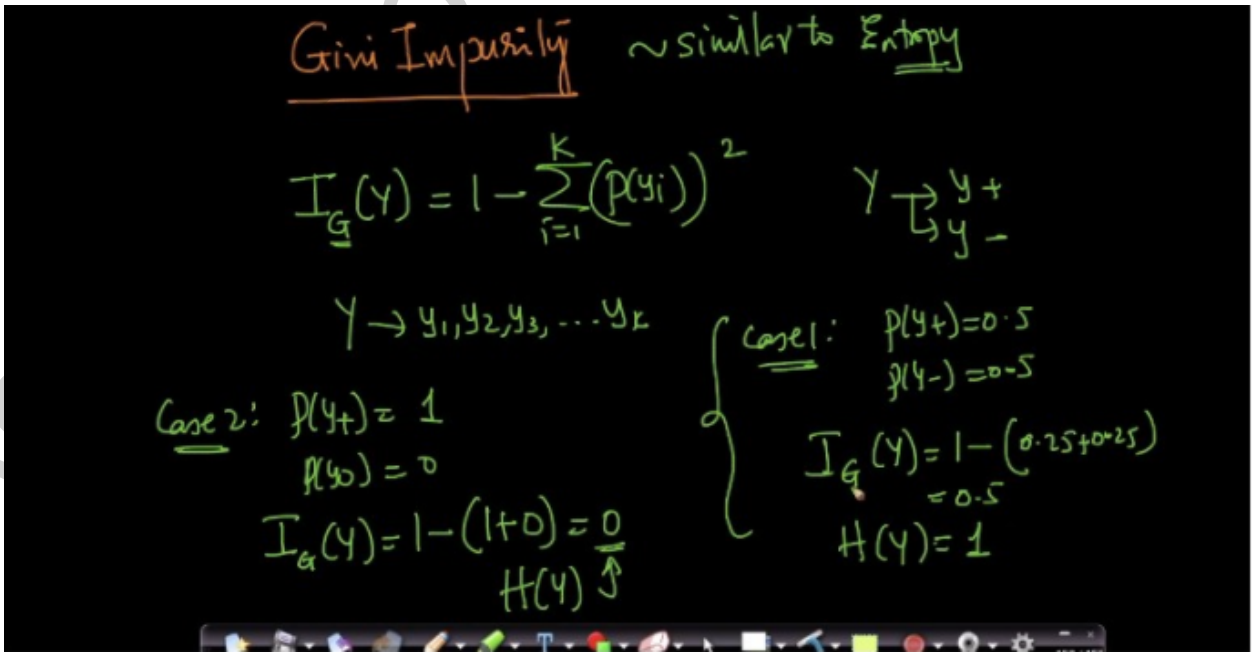
Timestamp 9:20

So, in general if we have dataset D which is broken into k subsets say D_1, D_2, … , D_k using some variable var.
Then, Information Gain w.r.t Y_i's of these dataset can be written as,

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

Information_Gain = [Entropy(parent)] – [Weighted Average Entropy of child nodes]

37.6 Building a decision Tree: Gini Impurity



Timestamp 2:37

Here we will talk about Gini Impurity which is similar to entropy but has certain advantages over it. It is represented as I_G(Y) (information gain is represented as IG(Y) both are not the same).

Gini Impurity for a random variable Y, which is split into k sets say y, y_2, … , y_k is

I_G(Y) = 1 - $\sum_{i=1}^{k} ((P(Y_i))^2$

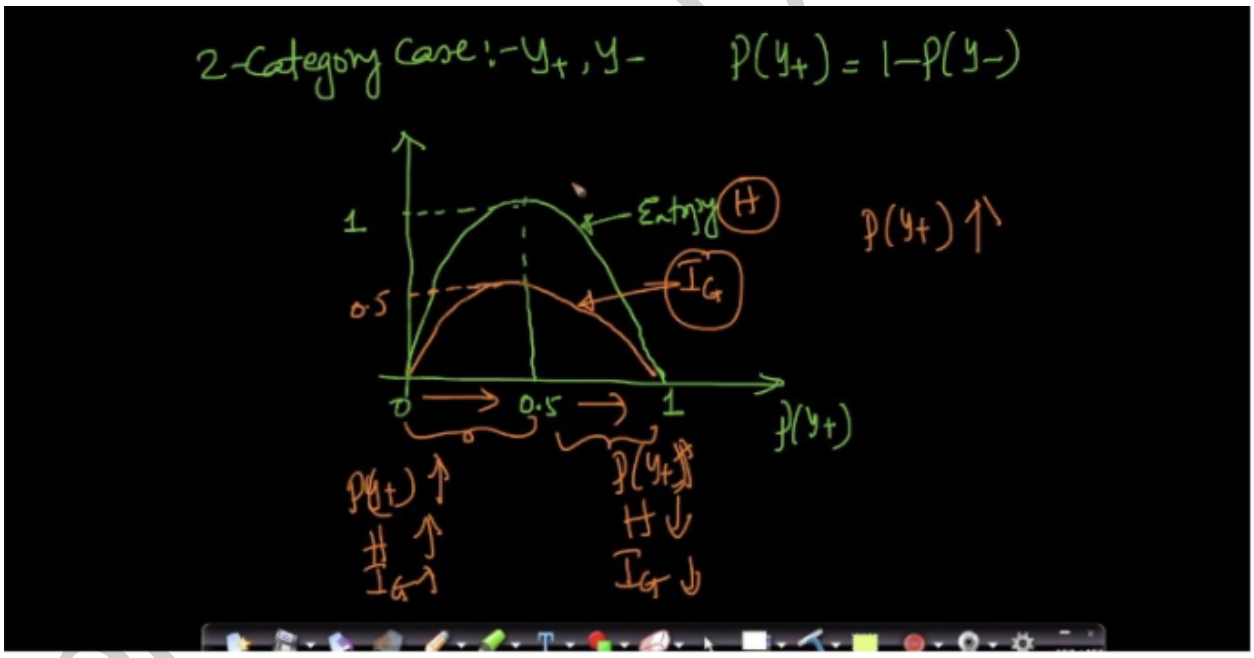
Let's compare it with entropy. Say we have a random variable Y with two classes Y_+, Y_-.

Case 1: P(Y_+) = 0.5 , P(Y_-) = 0.5

then I_G(Y) = 1 - (0.25 + 0.25) = 0.5, H(Y) = 1

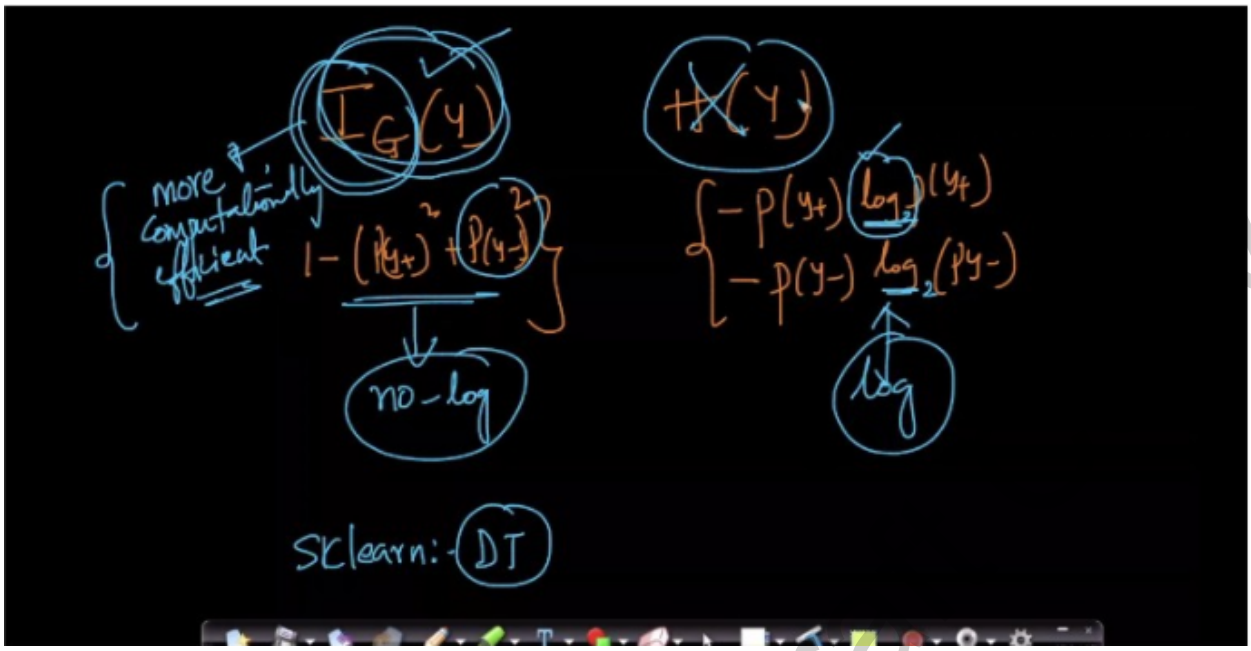
Case 1: P(Y_+) = 1 , P(Y_-) = 0

then I_G(Y) = 1 - (1 + 0) = 0, H(Y) = 0



Timestamp 4:28

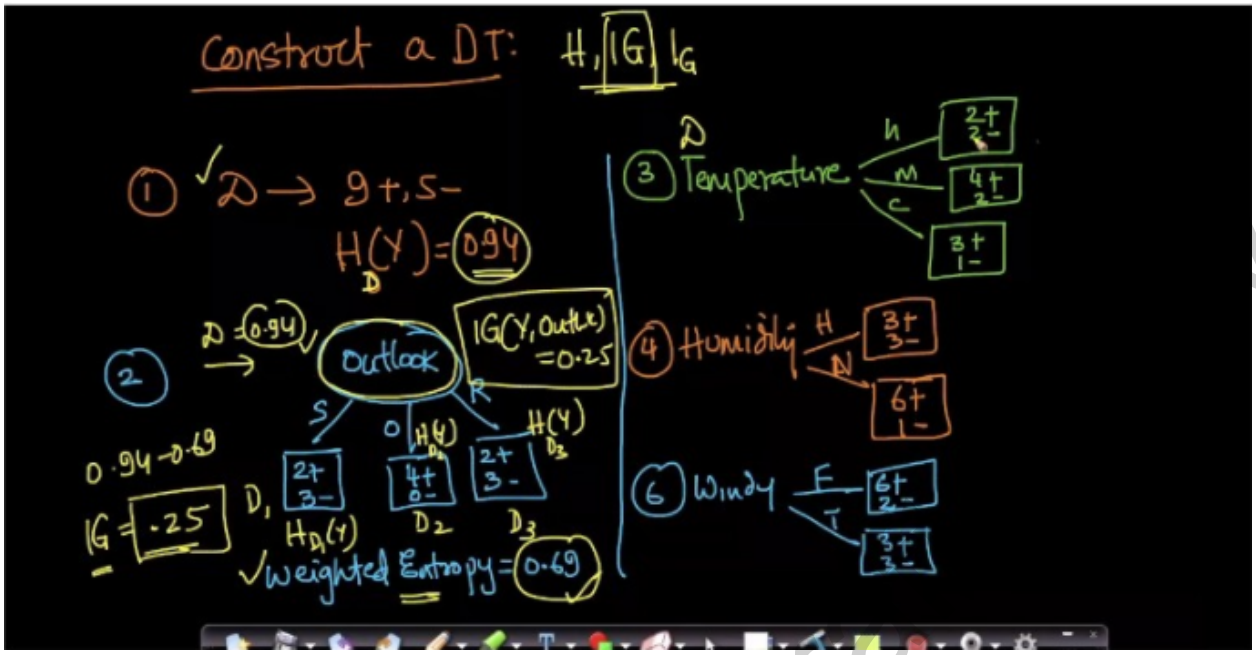
In the above image we have plotted gini impurity and entropy for the two class case. We can see that the maximum value gini impurity can take is 0.5 whereas the maximum value that entropy can take is 1. Gini Impurity seems like a scaled down version of entropy. Apart from this both are quite similar.



Timestamp 6:45

So, now the question is why did we study gini impurity? There is one fundamental advantage of gini impurity over entropy. As you can see from the above image we have written formulas for both these quantities. Entropy contains a log term which is computationally expensive whereas in gini impurity we just need to do simple arithmetic operations.
This is the reason why most people use gini impurity instead of entropy while calculating information gain. Also in sklearn implementation of decision trees gini impurity is used as default.

37.7 Building a decision Tree: Constructing a DT

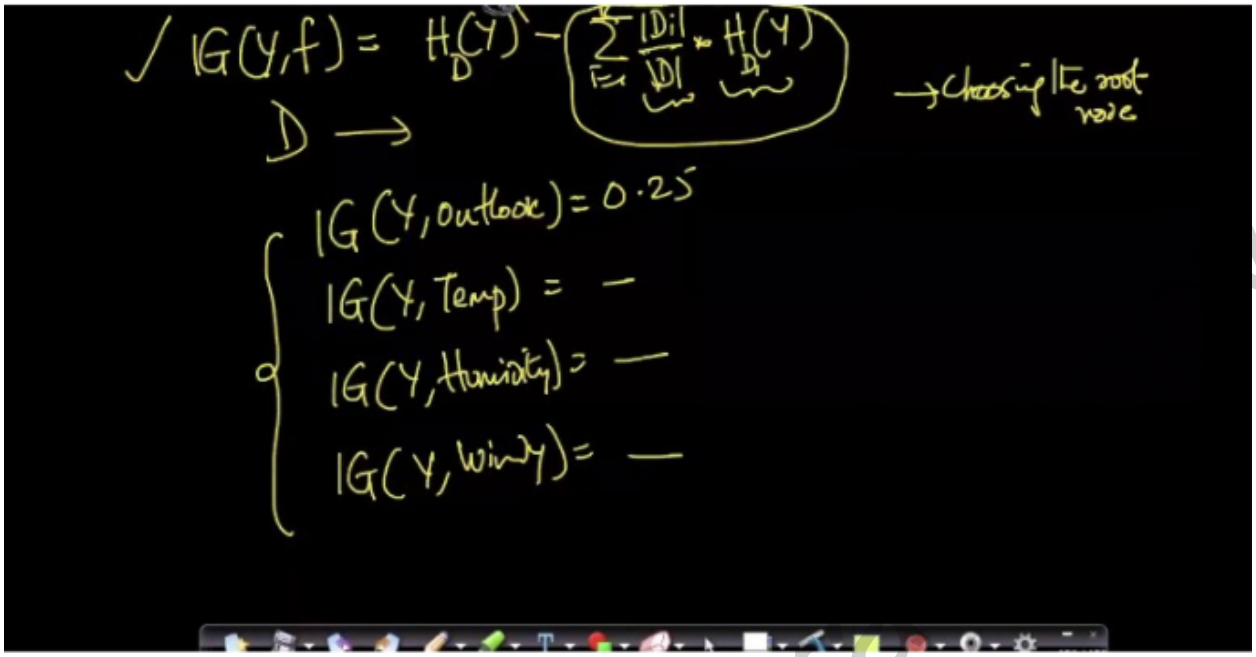


Timestamp 2:50

Till now we have learnt about entropy, information gain, and gini impurity. Let's now put them to use and build a decision tree.
We build decision trees using the top down approach i.e we start with the dataset as a whole and then break them into chunks.

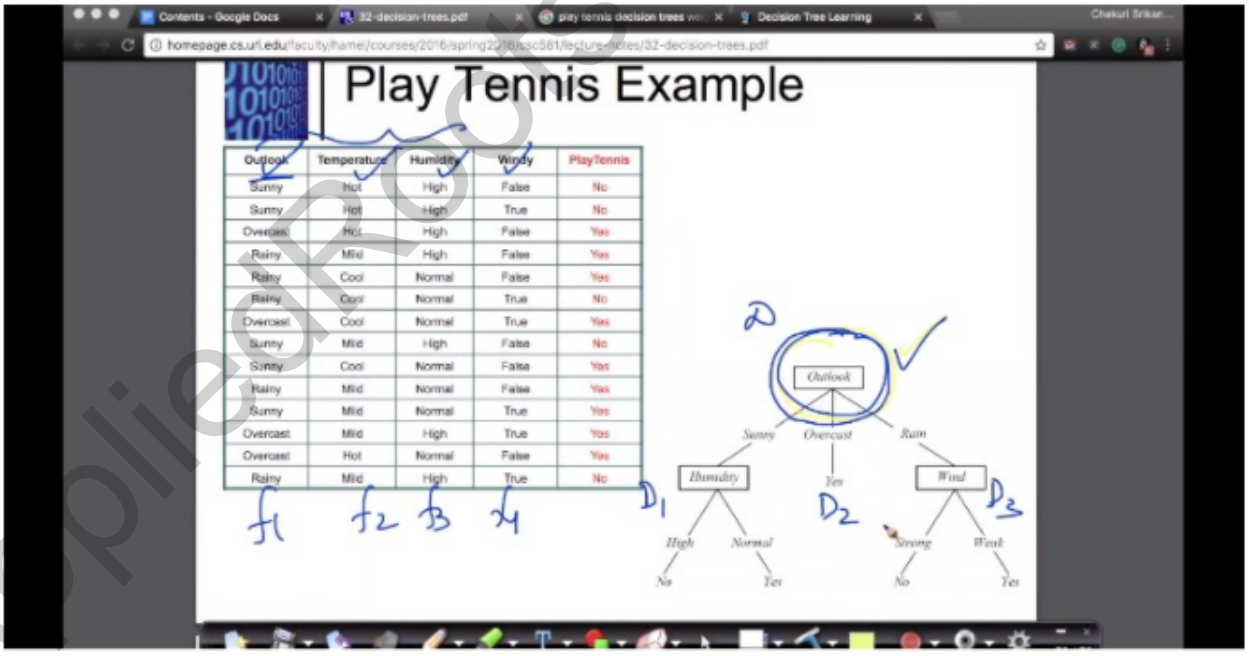
Here we will use the tennis dataset D which has 9 positive and 5 negative data points. So the entropy of whole dataset before splitting is H_D(Y) = 0.94.

Now for splitting we take all the features one by one and calculate the Information Gain.
Say at first we split our dataset D using outlook and calculate the weighted entropy for this splitted dataset which is 0.69 as shown in the image above. So the information gain is 0.94 - 0.69 = 0.25
So, IG(Y, outlook) = 0.25



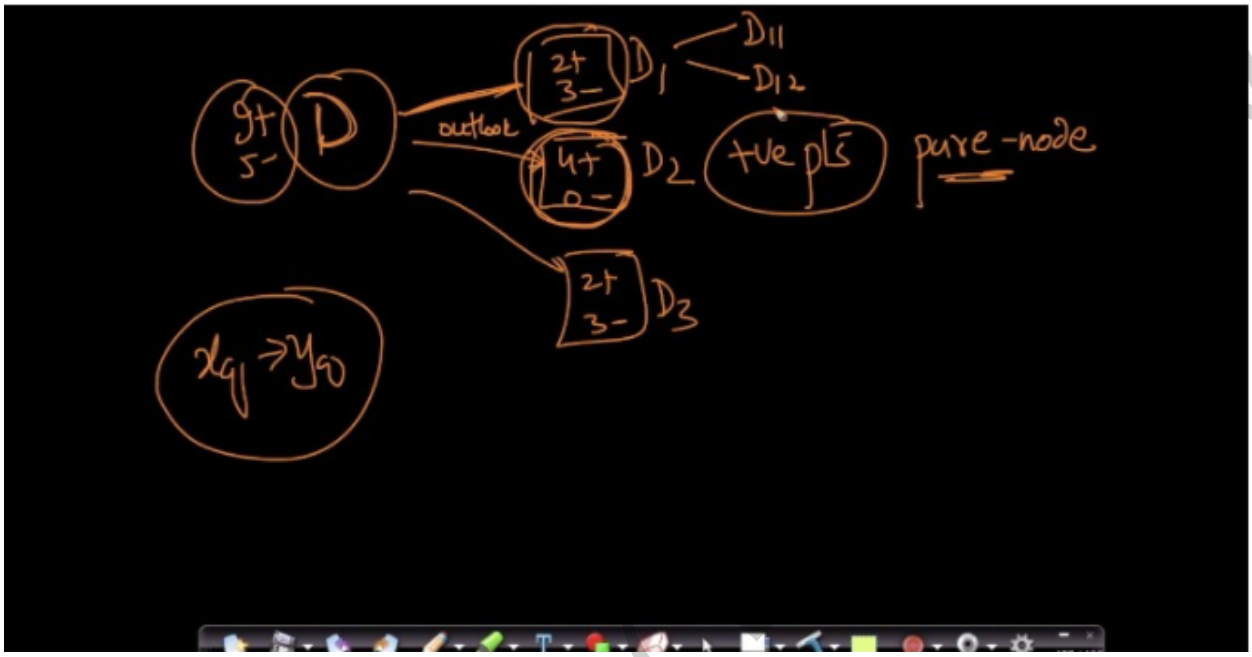
Timestamp 6:05

Right now we are finding a split for the root node. So,what we essentially do is to calculate the information gain for all the features and then select that feature which has the maximum Information Gain.



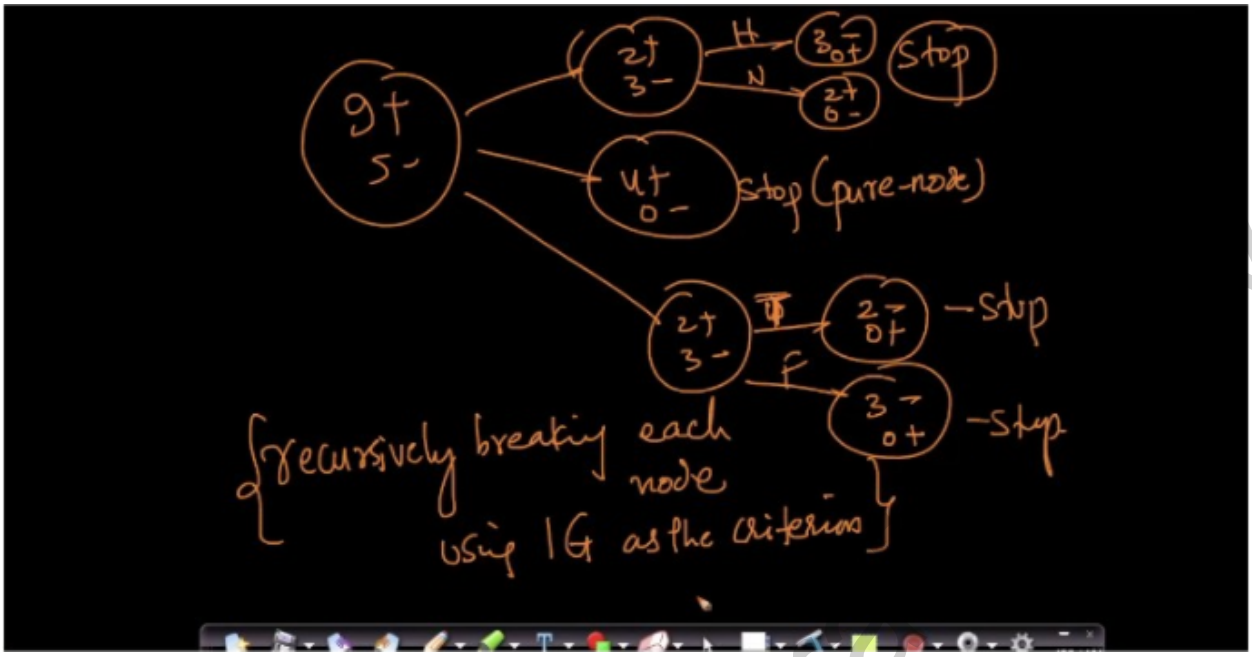
Timestamp 7:10

After calculating the information gain for all the features w.r.t D we find that outlook has the maximum information gain. So it becomes the root of the tree as shown in the image above. After splitting the dataset D with outlook it results in three different datasets say D_1, D_2 and D_3.



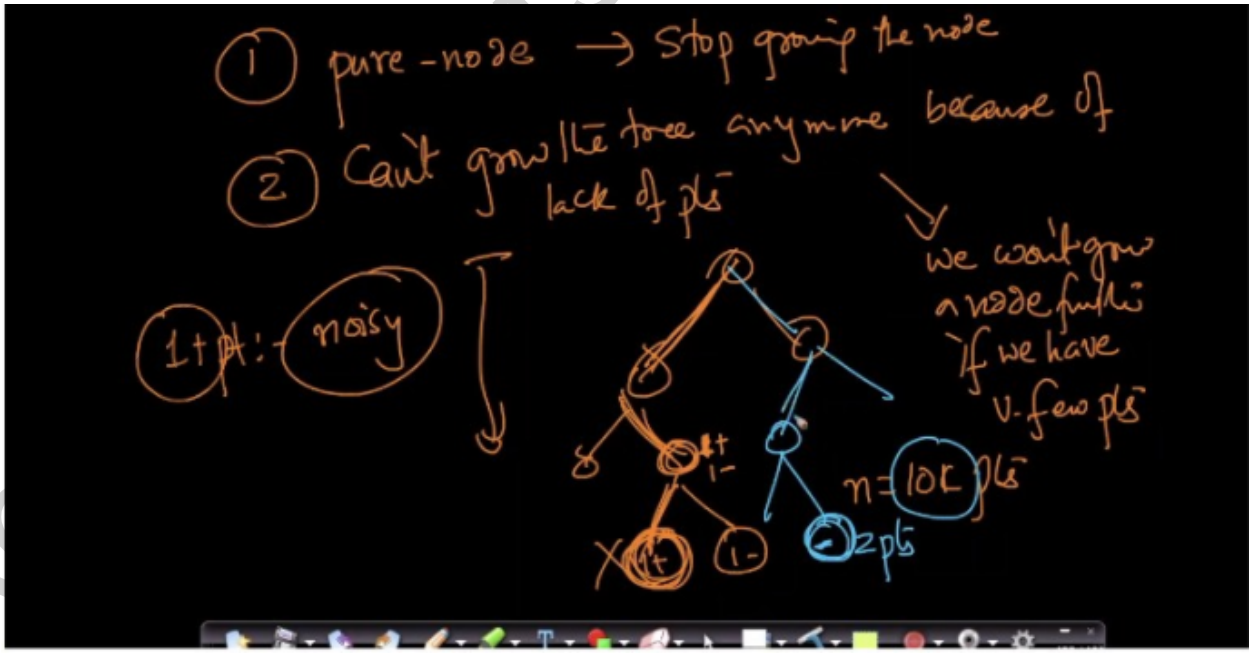
Timestamp 9:28

D_1 contains 2 positive points and 3 negative points, D_2contains 4 positive points and 0 negative points and D_3 contains 2 positive points and 3 negative points. Now we can see that D_2 contains only positive points. Such a node is called a pure node and we don't split pure nodes.



Timestamp 15:02

Again we can break D_1 using humidity, this results in two pure nodes. Similarly D_3 can be splitted using wind that will result into two pure nodes.
So, essentially we are recursively breaking each node into child nodes using Information Gain as the criterion.

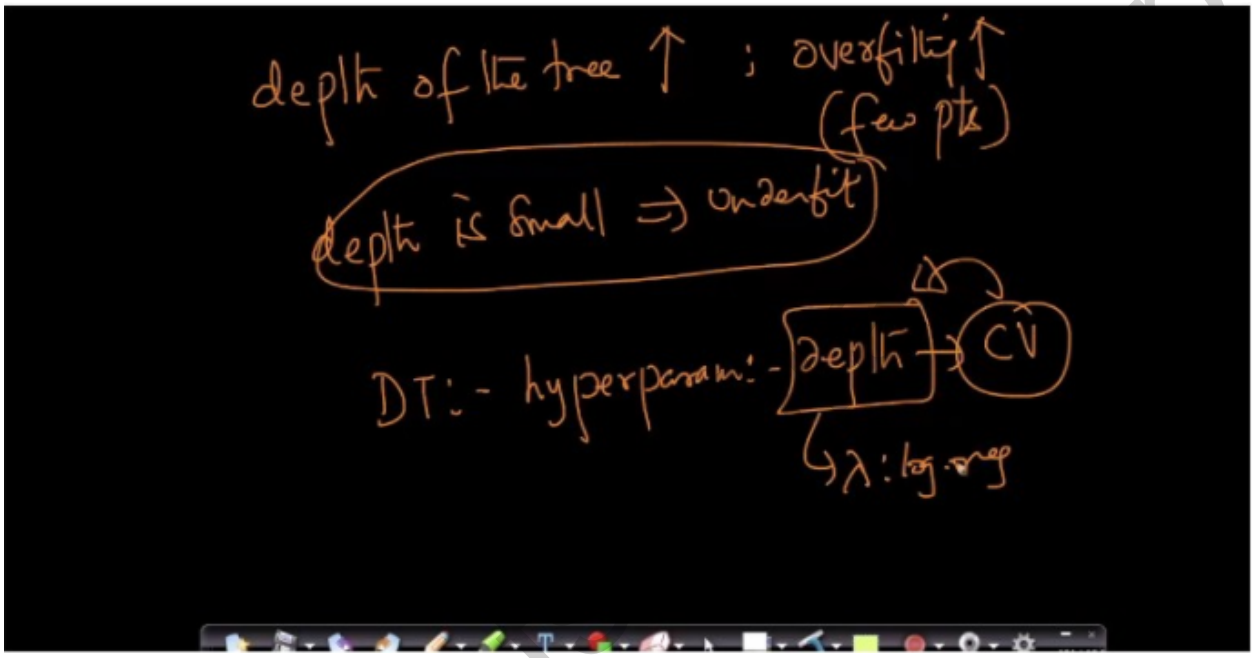


Timestamp 18:12

Now when should we stop further splitting a node:

1. Pure node - we stop growing our tree when we have a pure node because there is no point in splitting our node after it.

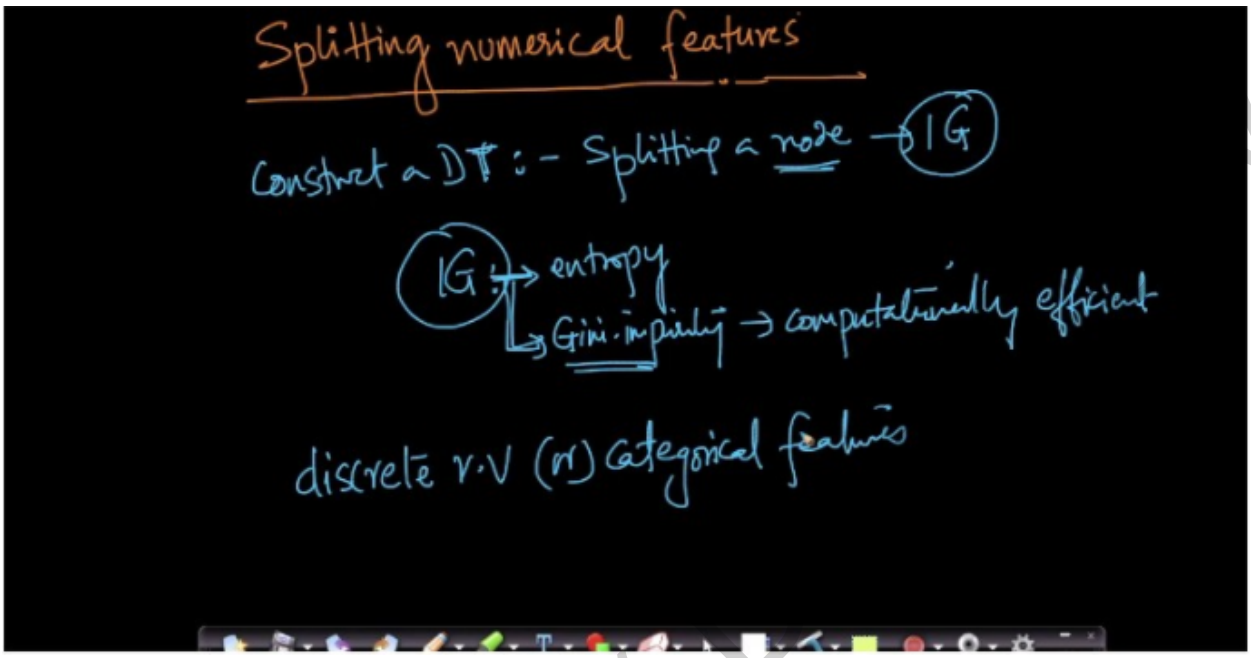
2. We also stop splitting if there are very few points in a node. Say we have a dataset of 10k points and in one of the nodes we have only 2 points - 1 positive and the other negative. Now if we split this node then we are giving way to importance to a single data point. This data point can be an outlier. So in a way we will overfit.



Timestamp 20:25

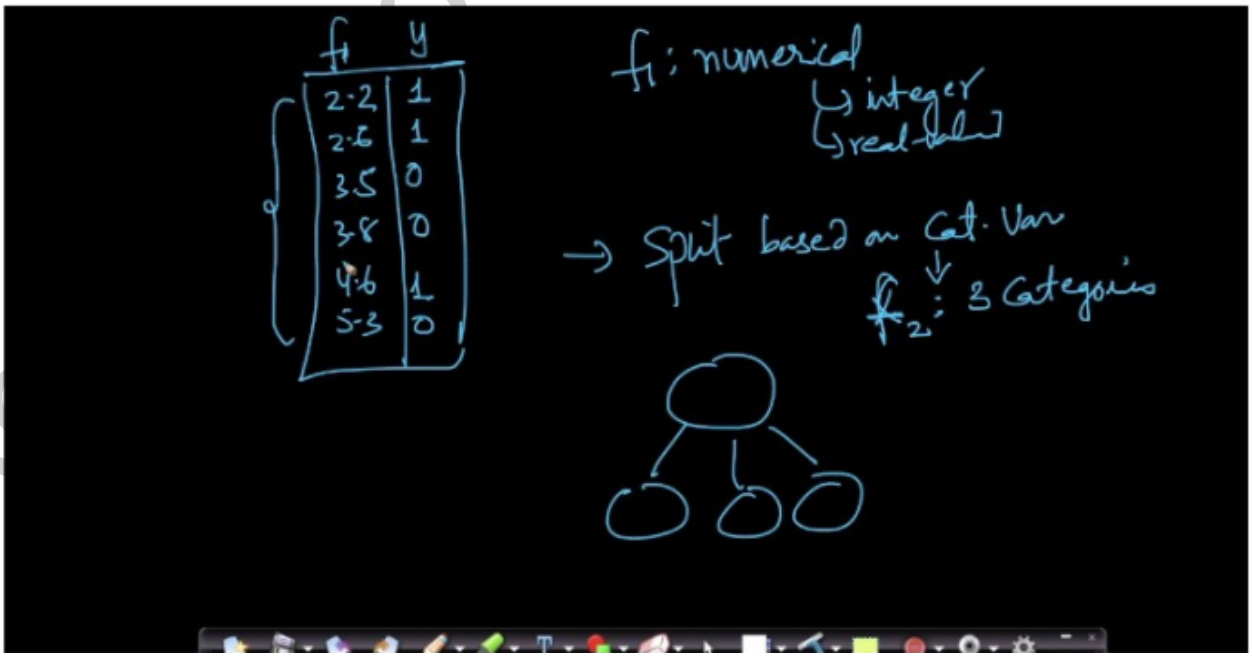
3. We also stop growing our tree beyond a certain depth, because as the depth increases there are fewer and fewer data points remaining. If we fit to these points then we are overfitting.
So, in general if the depth is large we are overfitting and if the depth is small we are underfitting. depth is a hyperparameter, which we find using standard practices like cross-validation.

37.8 Building a decision Tree: Splitting numerical features



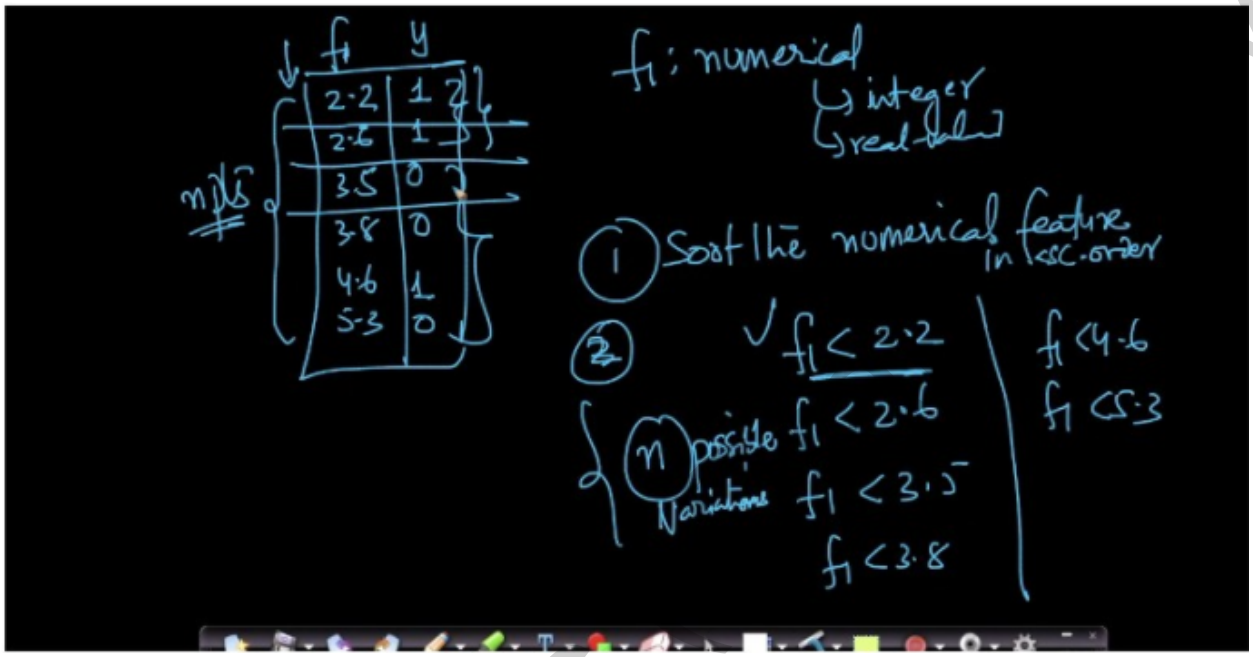
Timestamp 1:15

In the last lecture we learnt how to build a decision tree. The main operation for building a decision tree was splitting a node and calculating the information gain. All the examples we have seen till now were based on categorical variables. Now how do we handle numerical features?



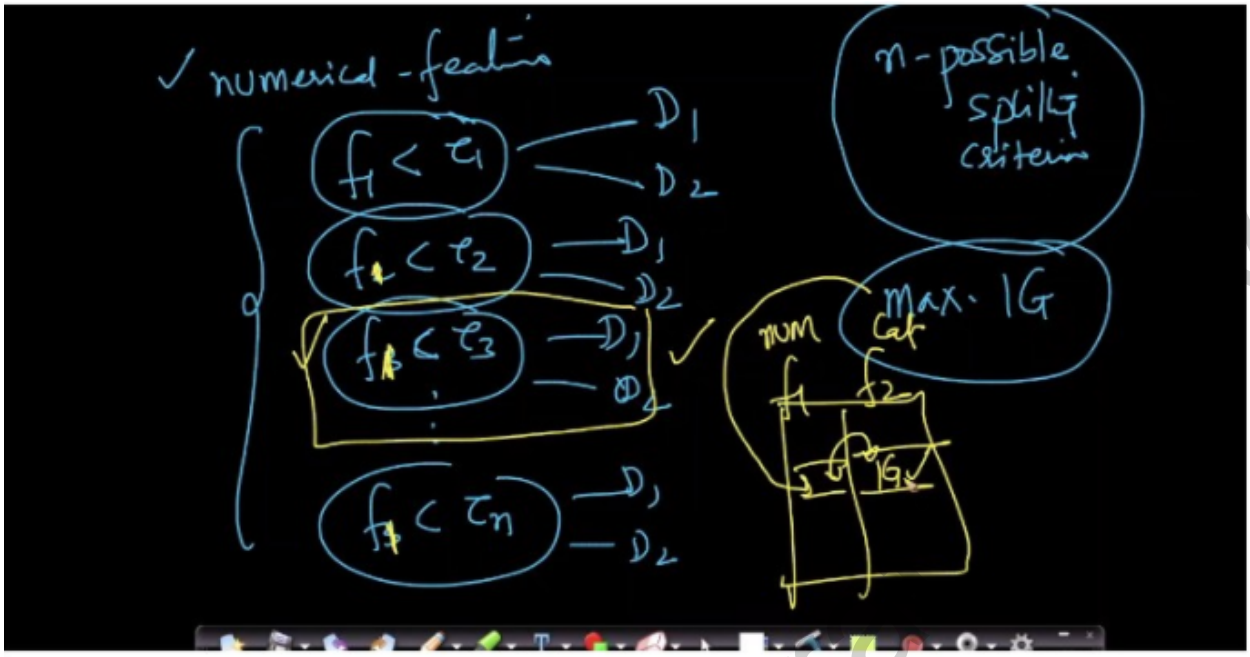
Timestamp 2:36

Numerical features are continuous in nature and it cannot be treated in the same way as categorical features. Say a categorical feature had 3 categories then we split it up into three 3 child nodes, but a numerical feature can have countless unique values.



Timestamp 4:54

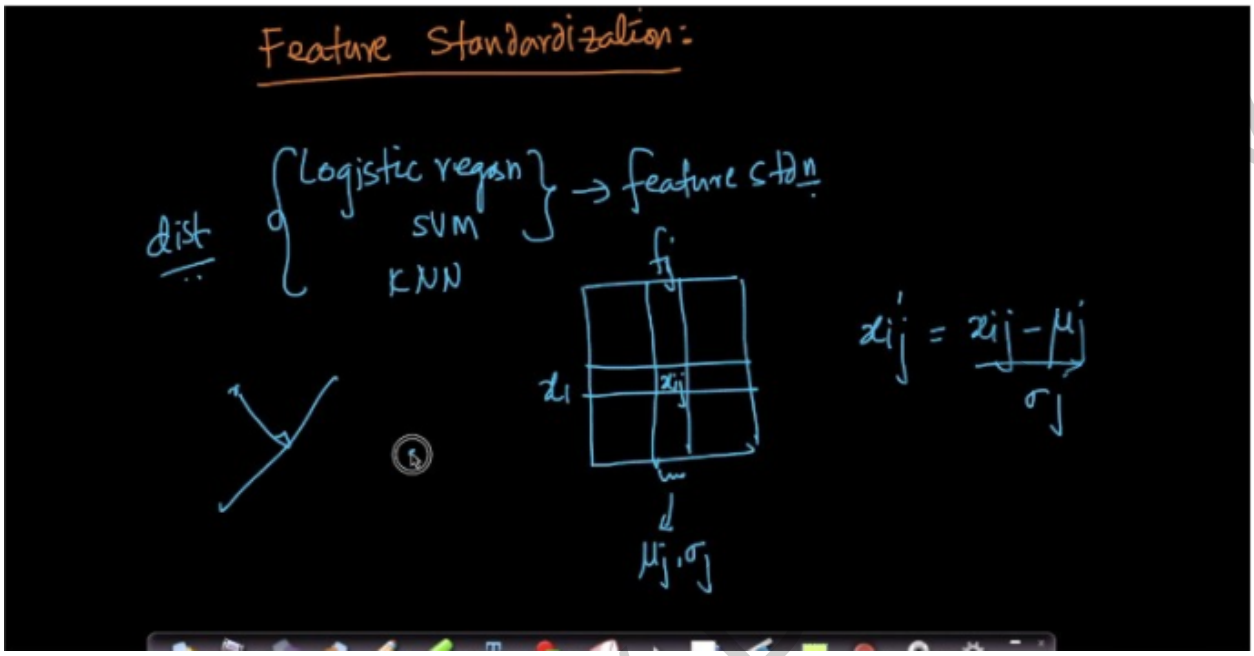
Consider that f_1 is a numerical feature with values as shown in image above. Now what we can do is sort the values into ascending order. After sorting we can pick a value say 2.6 from this column and make a rule such that all rows having values less than this threshold will go to the left branch and otherwise right branch. If there are n rows in the dataset then there are at most n different possibilities for this threshold.



Timestamp 7:14

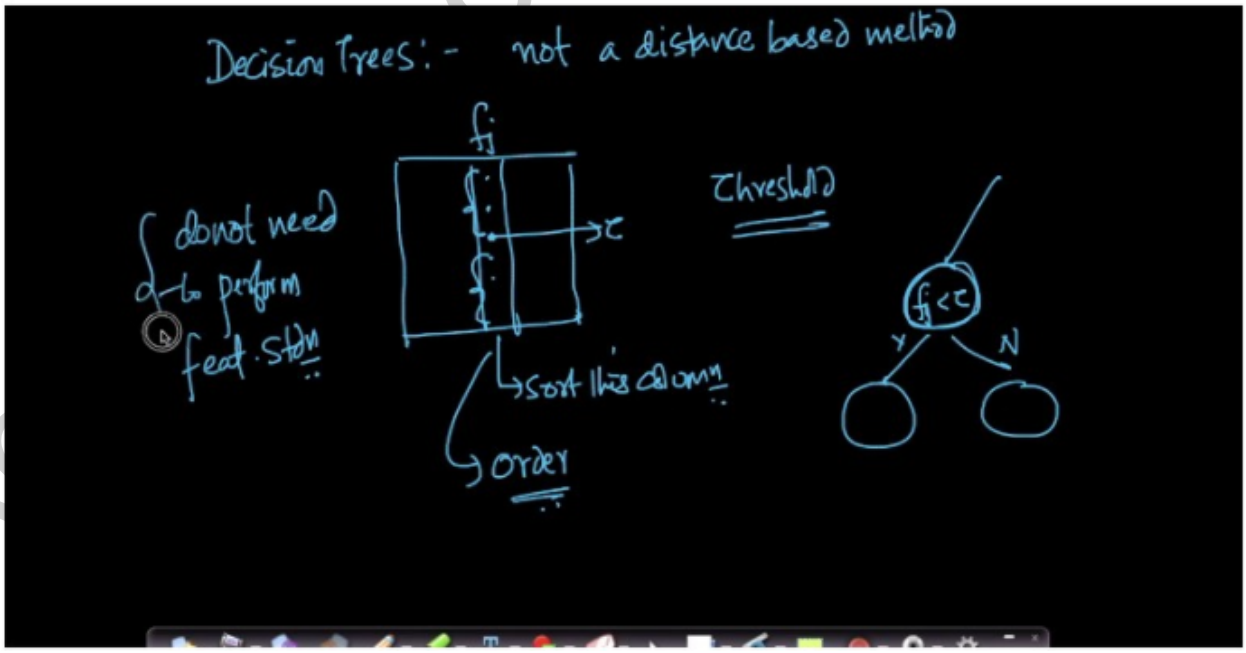
So essentially we need to check every value in our numerical feature. We take each value from the numerical feature column and use this as threshold and calculate the information gain. Among all the splits we select the one that caused the maximum information gain.
This is extremely time consuming. Also we need to compare the information gain with other features information gain.

37.9 Feature standardization



Timestamp 1:15

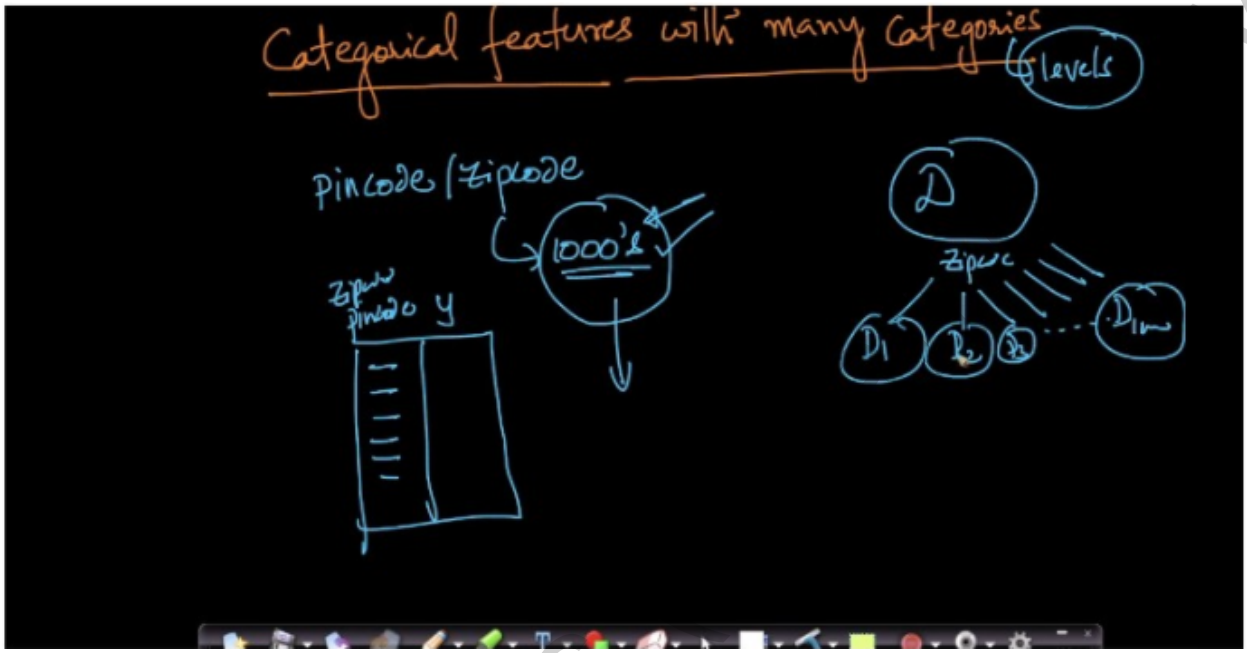
Algorithms like logistic regression, SVM, KNN are all distance based algorithms, so feature standardization was very important for them to work properly. Now decision trees is not a distance based method. So feature standardization is not important.



Timestamp 4:01

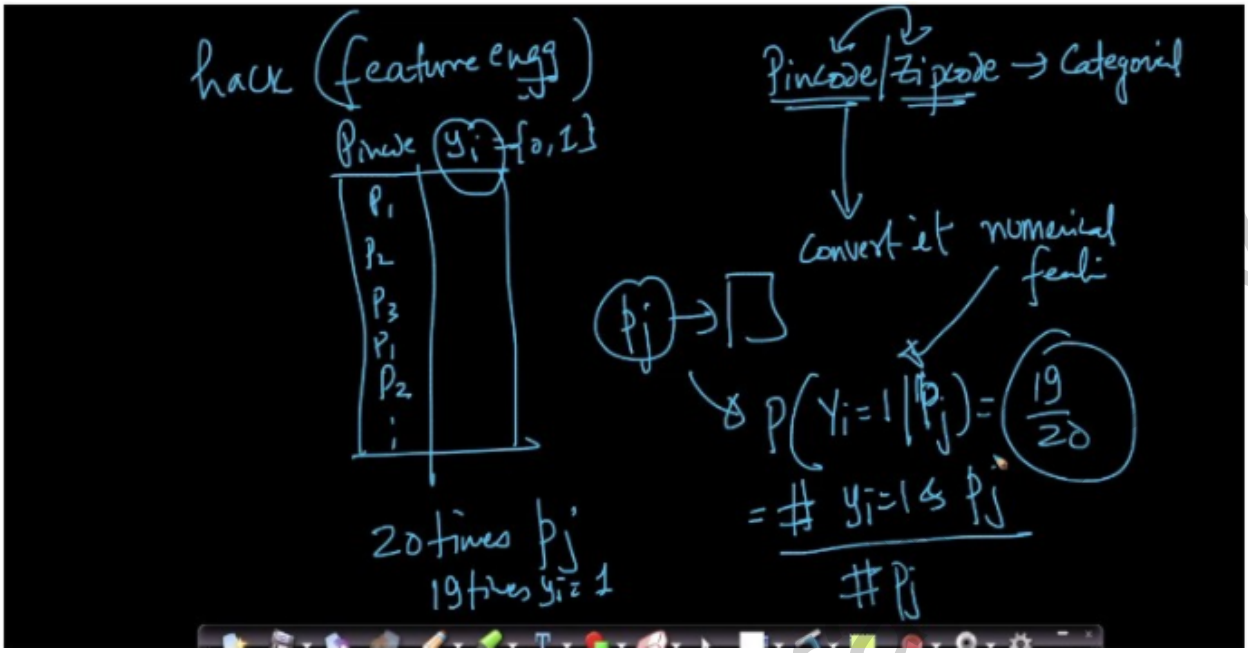
When we dealt with numerical features we sorted the column and chose a threshold value. So only ordering of the data mattered.

37.10 Building a decision Tree:Categorical features with many possible values



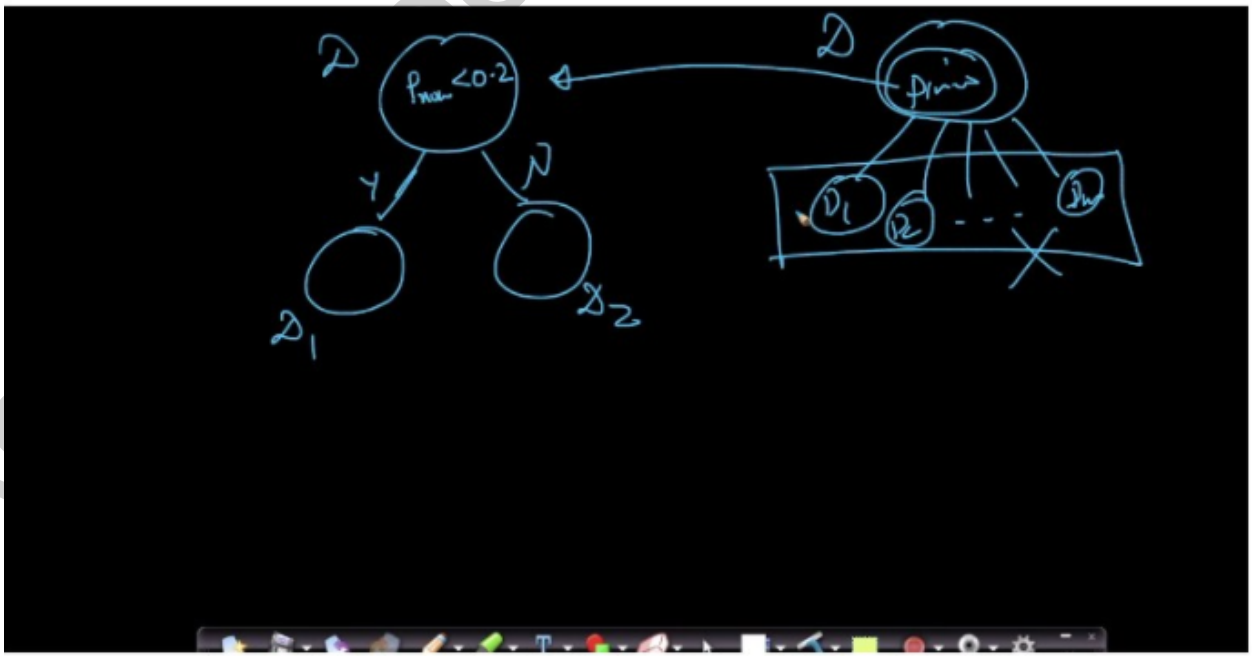
Timestamp 2:13

Suppose in our dataset we have a feature like PINCODE/ZIPCODE. Now it can contain 1000s of different values. It cannot be treated like numerical features because we cannot really compare two PINCODEs. If we treat it like a categorical feature then we need to split it into various categories, the categories can now run into thousands. Each split may not contain many data points. This can be problematic.



Timestamp 4:20

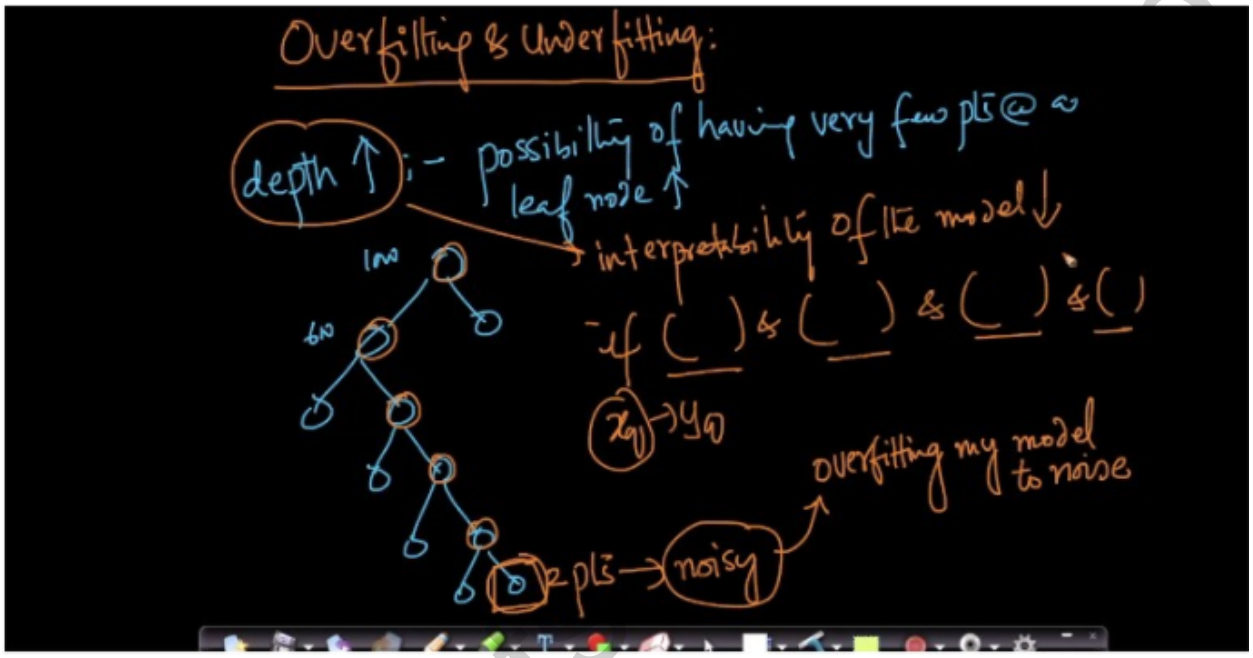
In order to deal with this problem we use a feature engineering hack that works quite well in practice.
Essentially we convert our categorical data into numerical data. Consider the column PINCODE as shown in the image above. Say, a row has PINCODE = P_j and class label y_i = 1, then we need to calculate the probability P(y_i = 1 | P_j), then we can replace the PINCODE P_j with this probability value.



Timestamp 6:22

After converting the categorical data to a numerical feature we can use this column to split our features as we learnt earlier. This considerably reduces the number of child nodes we have.

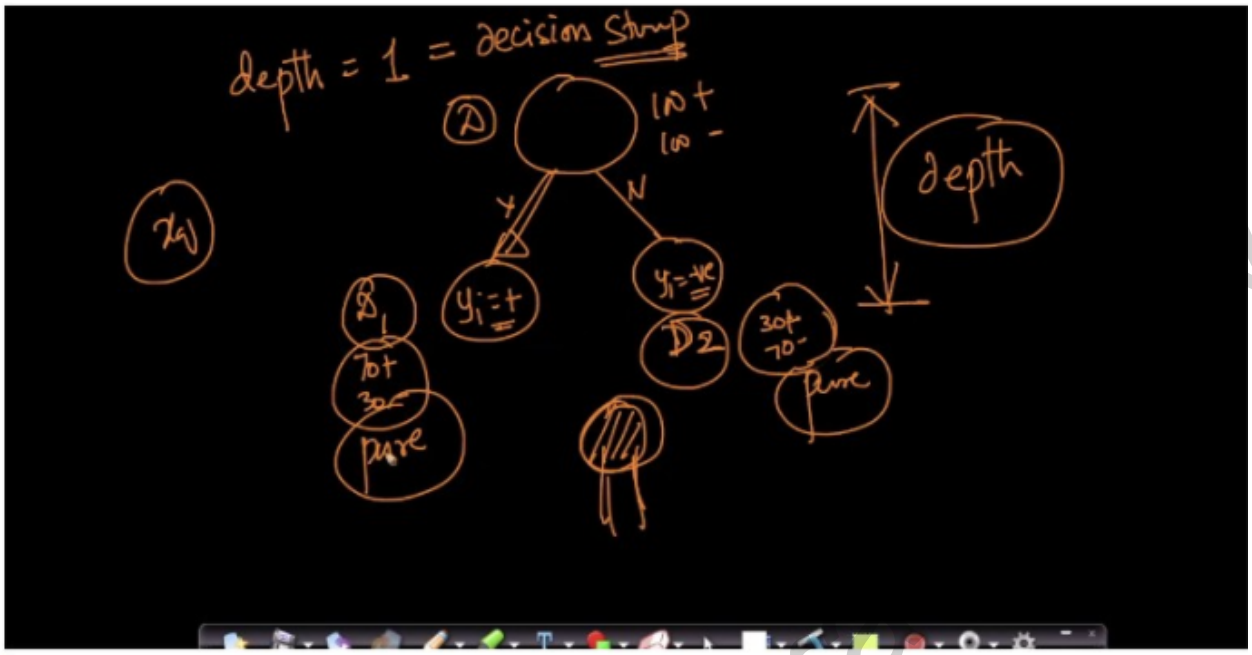
37.11 Overfitting and Underfitting



Timestamp 2:08

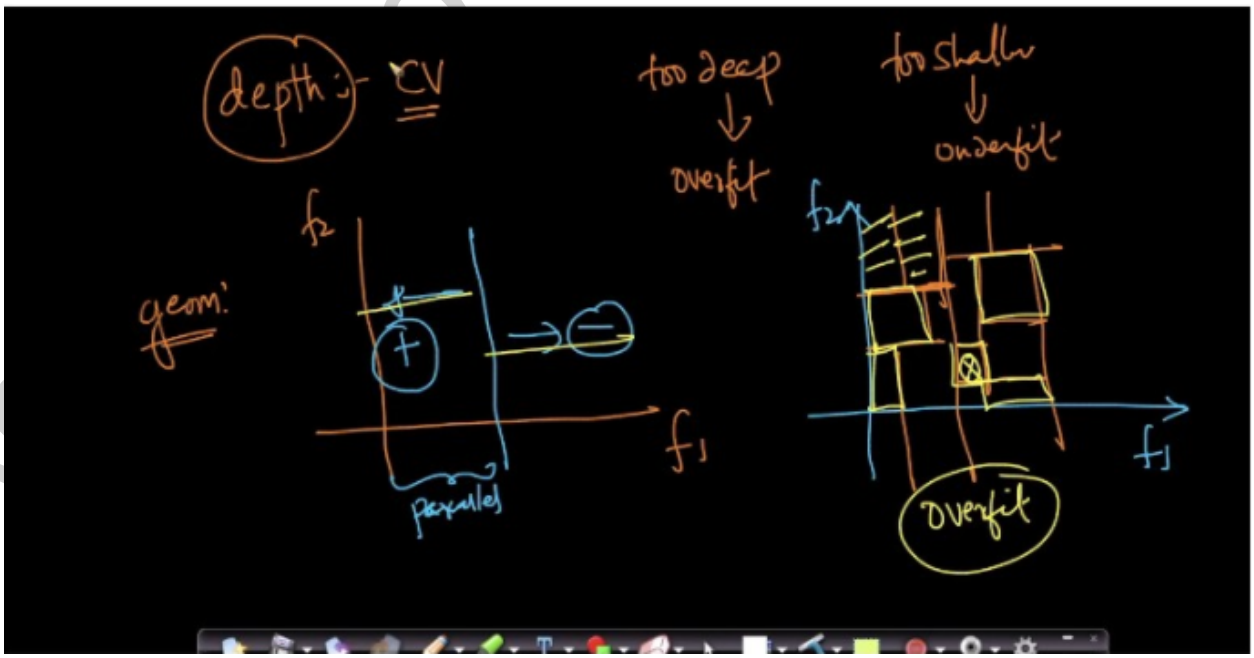
Now let's talk about the cases where our model may underfit or overfit.

Suppose we have a tree with large depth then it may happen that the leaf nodes will contain very few points. These points can be noisy points. So, in a way we are overfitting our model. Also the interpretability of the model goes down as the depth of the tree increases.



Timestamp 4:27

A tree can also have very small depth. Consider the tree given in the image above it's height is 1. A decision tree with height = 1 is called Decision Stump. Say we have a dataset D with 200 points with 100 positive and 100 negative. Now we make a split into two nodes, one containing 70 positive and 30 negative and the other containing 30 positive and 70 negative. Now both of them are not pure nodes. To make a prediction we take a majority vote.
So, in a way we are underfitting because the tree contains very few decision nodes and also there are no pure nodes.



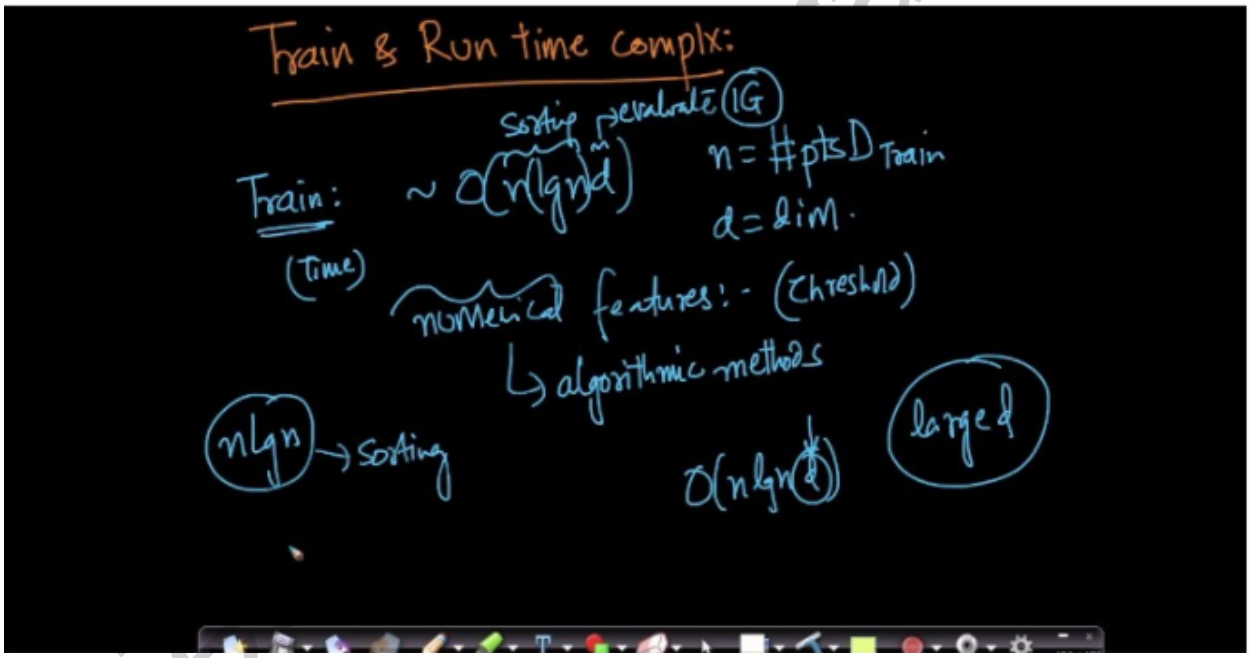
Timestamp 7:10

So essentially if the depth is too high we are overfitting and if the depth is low we are underfitting.

Now let's try to see what it means to have a large or small depth geometrically.
A tree with small depth essentially means that there are very few decision nodes. As we know that decision nodes are all axes parallel hyperplanes. If these hyperplanes are very few in number then there is not much splitting of the space of solutions as shown in image above. Essentially causing underfitting.

If the tree is of large depth then there are many hyperplanes in the solution space. This results in very small sized hyper cuboids. Essentially causing overfitting.

37.12 Train and Run time complexity

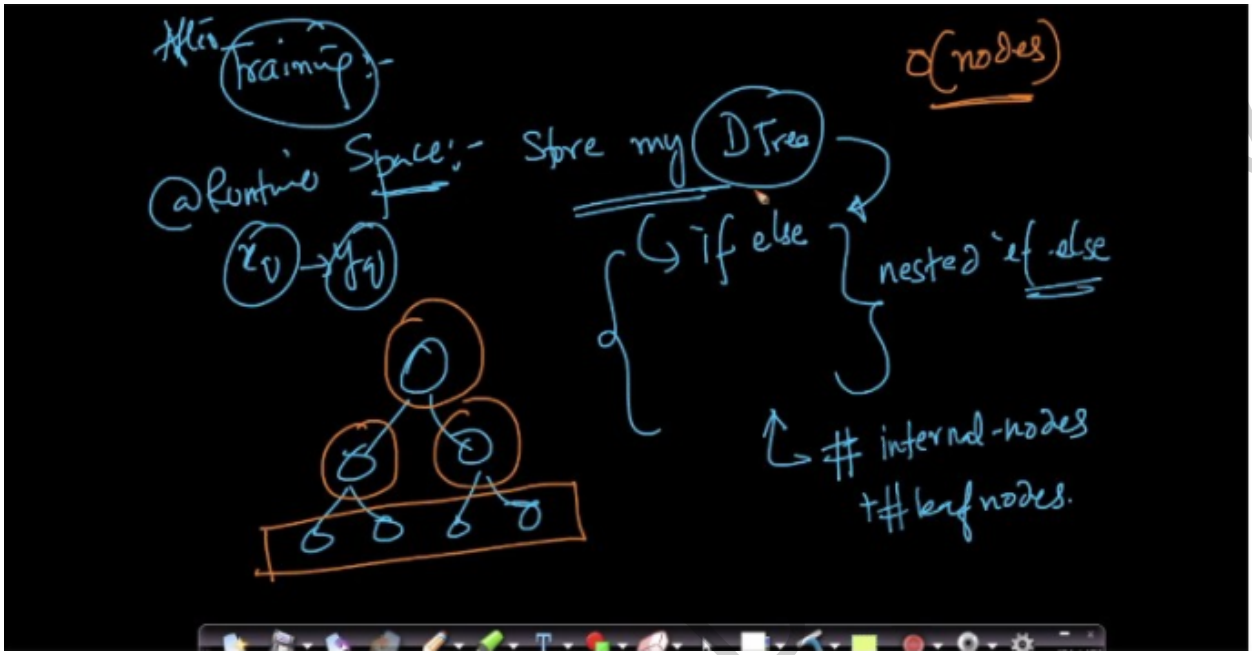


Timestamp 1:40

Train time complexity for Decision Trees is O(nlgn*d),
where n = #data points in train, d = #dimension.

The nlgn part is there because of the sorting we need for features to select a threshold. d represents the number of dimensions i.e. no of features in the dataset. We learnt earlier that numerical features were time consuming but there are certain hacks to get around this problem.

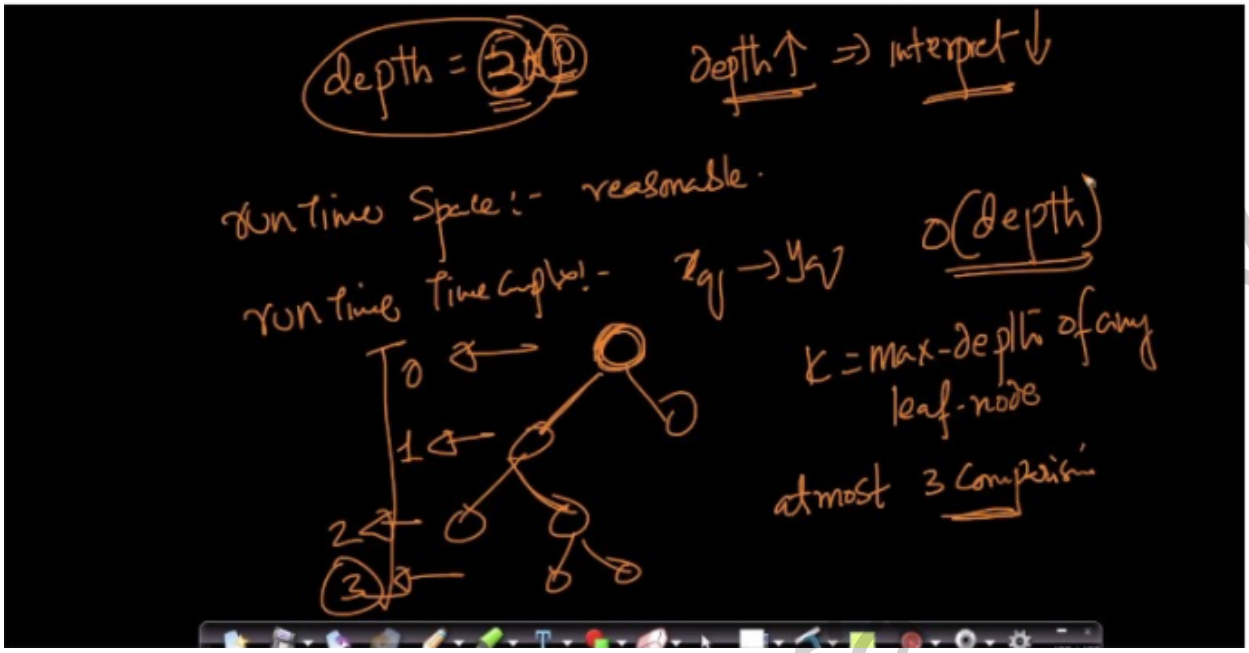
Decision trees are not very suitable for dataset having high dimension because of the d term present in the complexity.



Timestamp 3:20

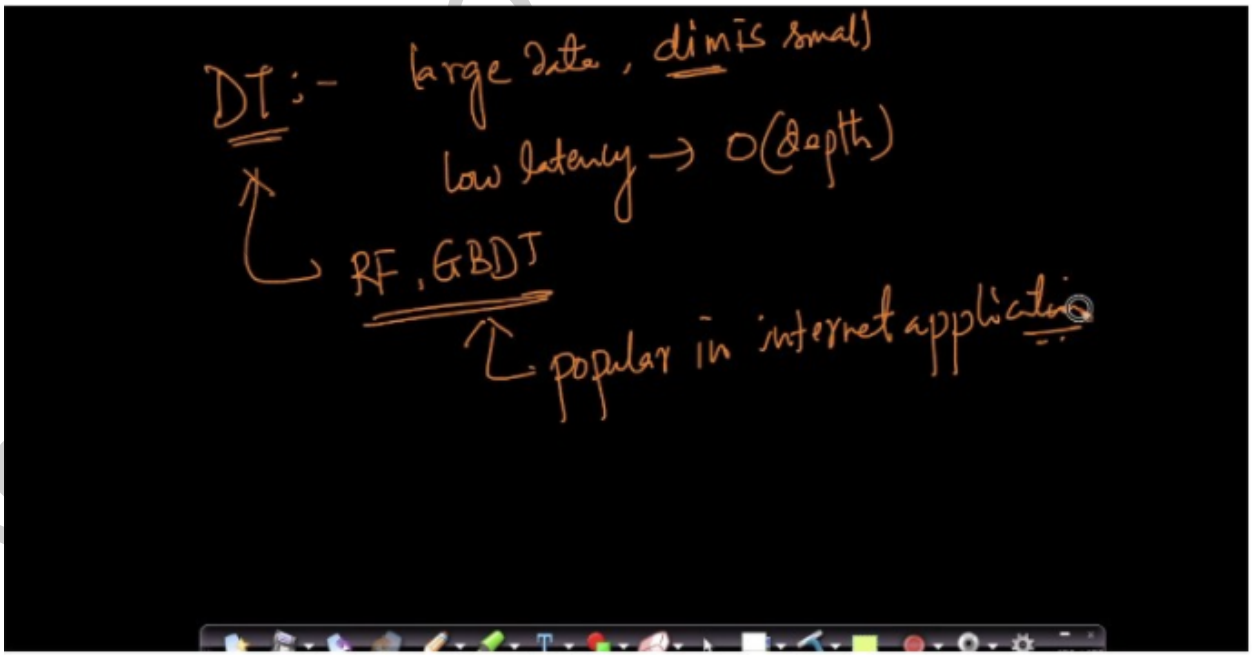
Run time space complexity is O(nodes), where nodes = #nodes

During run time we need the tree in memory to predict values. Now storing a decision tree isn't very space intensive. As we only need to store all the nodes i.e. internal nodes and leaf nodes. More often people convert decision trees into nested if else conditions and store them in memory.



Timestamp 5:25
Runtime time complexity is O(depth) , where depth = depth of the tree.

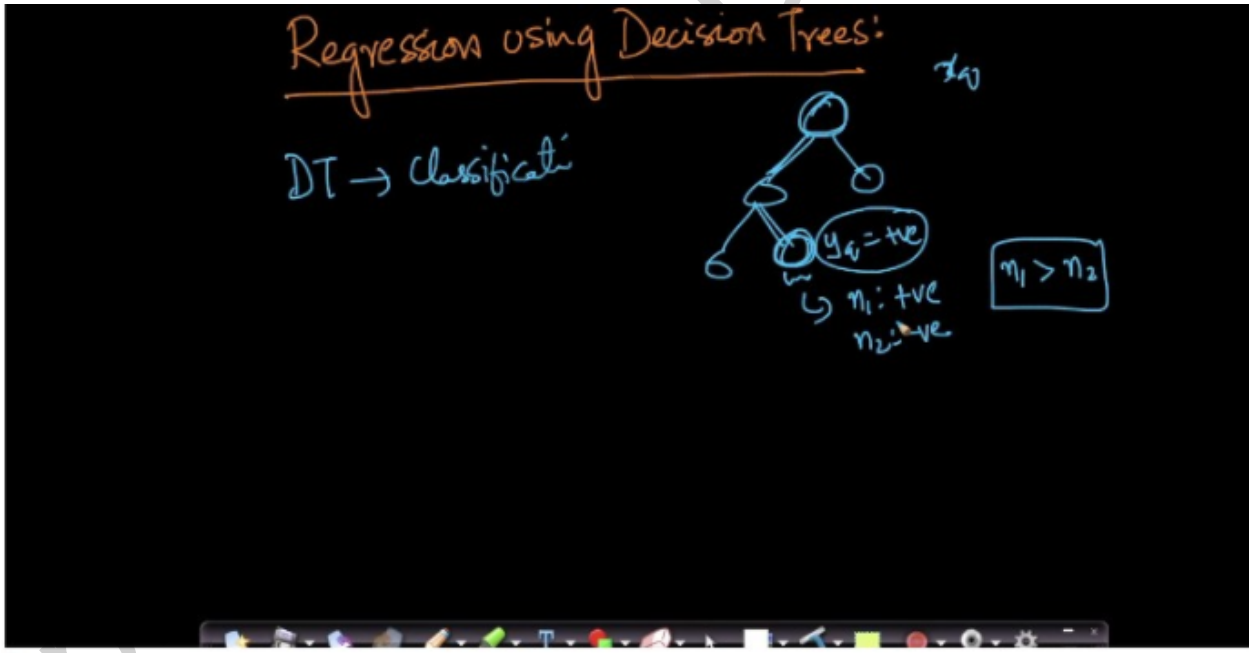
During runtime we just need to pass our query point x_q through the decision tree to get a output y_q i.e. $x_q \rightarrow y_q$. So for doing that in the worst case we need to traverse the depth of the tree.
Depth of the tree is defined as the longest path we can take from node to leaf node. Like in the image above the depth of the tree is 3. Depth of the root node is 0.



Timestamp 6:30

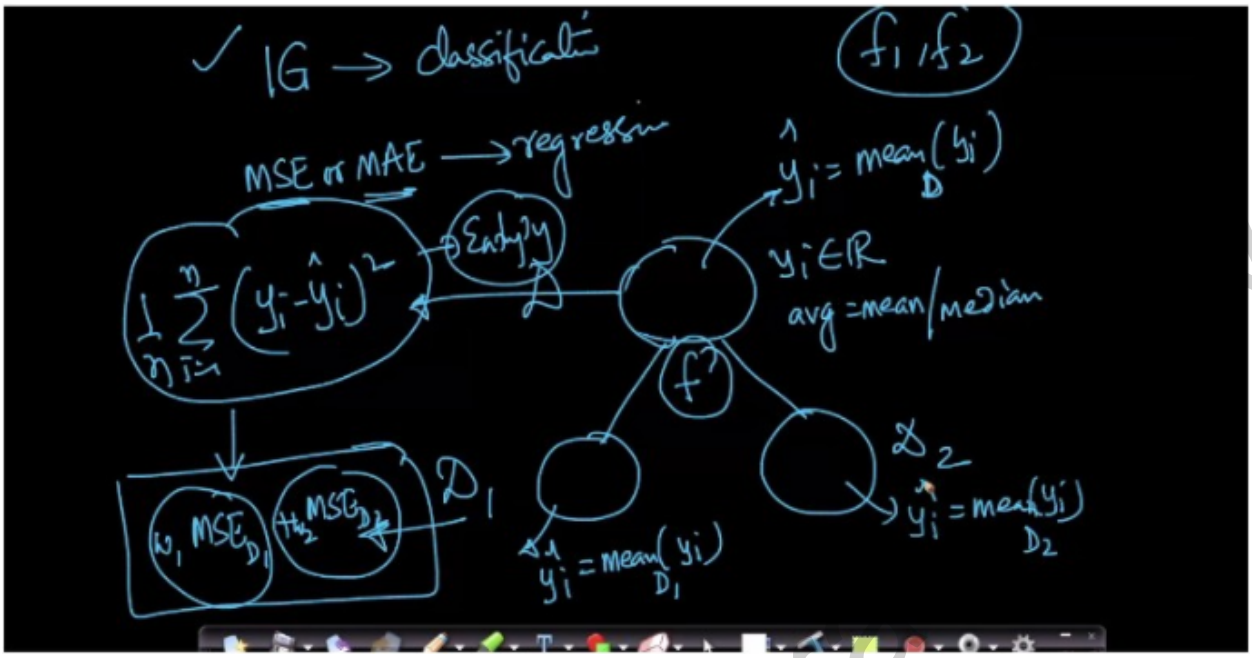
So, decision trees are extremely useful when we have large amounts of data, dimensionality is small.etc. Also it is extremely useful in cases where we have a requirement of low latency. As the run time complexity was only O(depth).
Later we will see variations of decision trees like Random Forests, Gradient Boosting Decision Trees.etc. These algorithms are very popular and are used in many internet companies.

37.13 Regression using Decision Trees



Timestamp 0:40

Till now we have seen decision trees for classification problems. We built trees using information gain as guidance. For prediction we traversed the tree and after reaching the leaf nodes we did a majority vote in the leaf node for our prediction.



Timestamp 3:37

Decision trees can also be used for regression problems. Infact, they work very well for regression problems.In classification problems we used information gain as criteria, in regression problems we use mean squared error(MSE) or median absolute error(MAE) as our criteria.

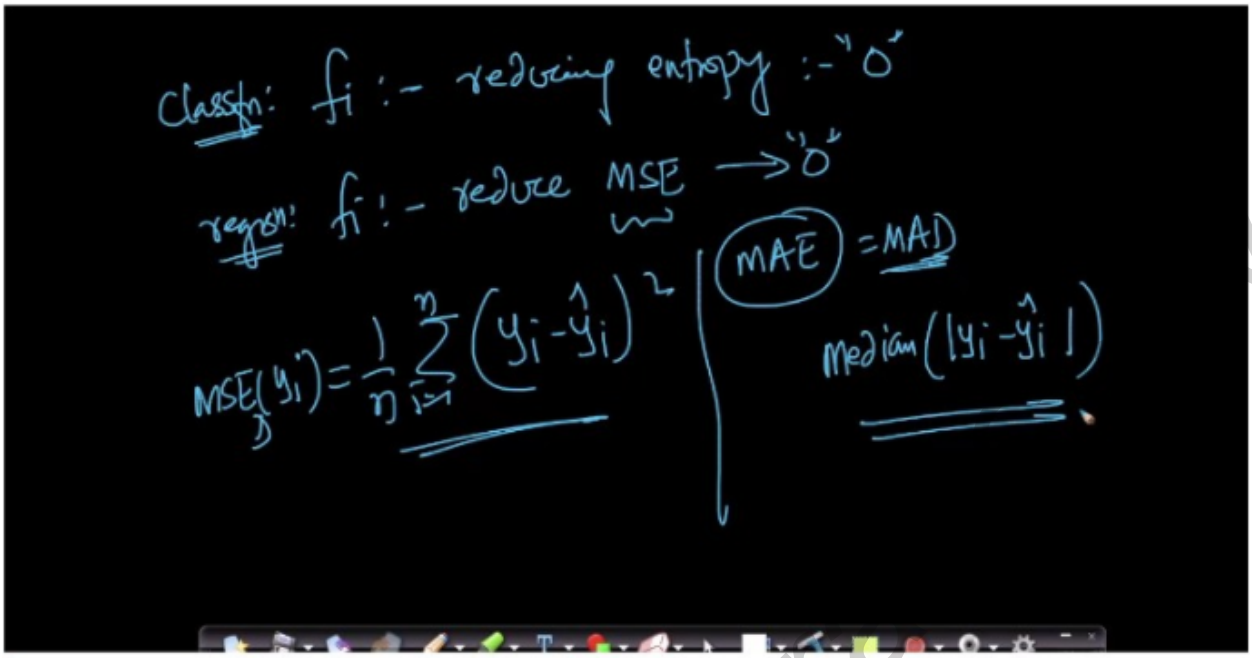
Suppose we have a dataset D with features f_1 and f_2, with target variable y ϵ R, now the mse of the dataset D can be calculated as,

MSE_D = 1/n* $\sum_{i=1}^{n} (y_i - \hat{y}_D)^2$,

here $\hat{y}_D$ = mean(y_i)

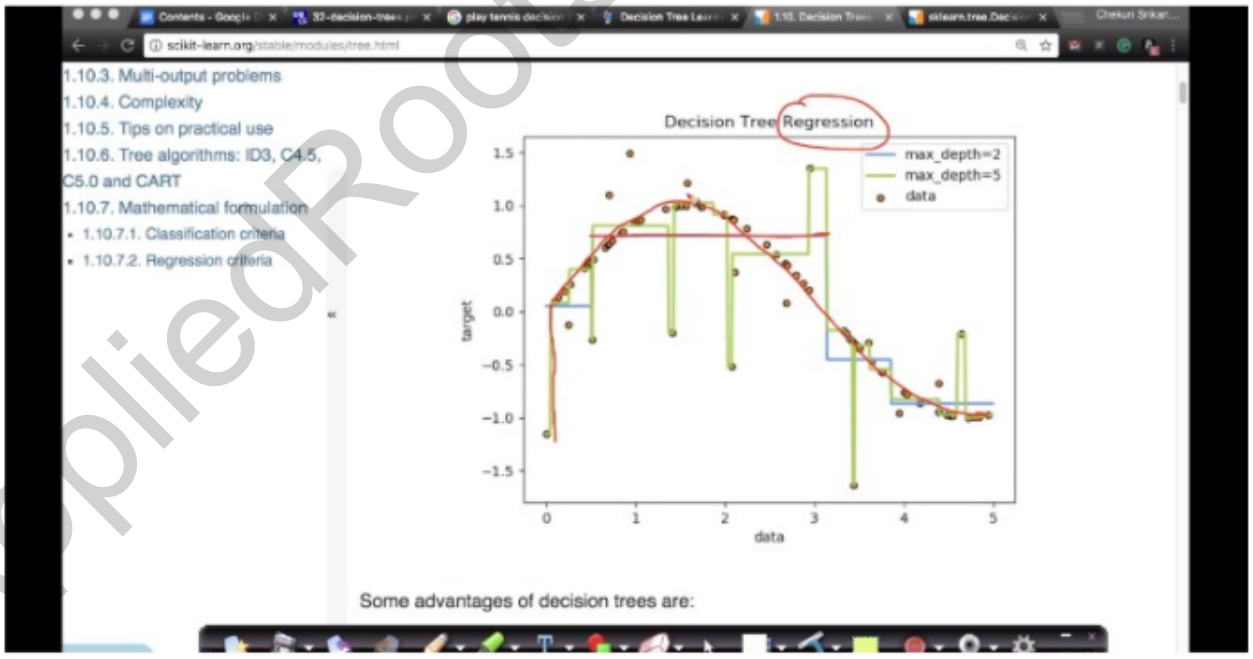
Now say we split our dataset into D_1 and D_2, so we will calculate the mse for both these datasets and take the weighted sum of them, just like we did in classification and then take the difference between the parent mse and child nodes weighted mse.

We do it for all the features and whichever feature reduces our mse most we select that feature just like we did with information gain.



Timestamp 5:32

So, like in classification we select features that reduce our entropy the most. Similarly in regression we select the feature that reduces MSE the most, both of them have a lowest value of 0.
We can also use MAE instead of mean, MAE is also robust to outliers.

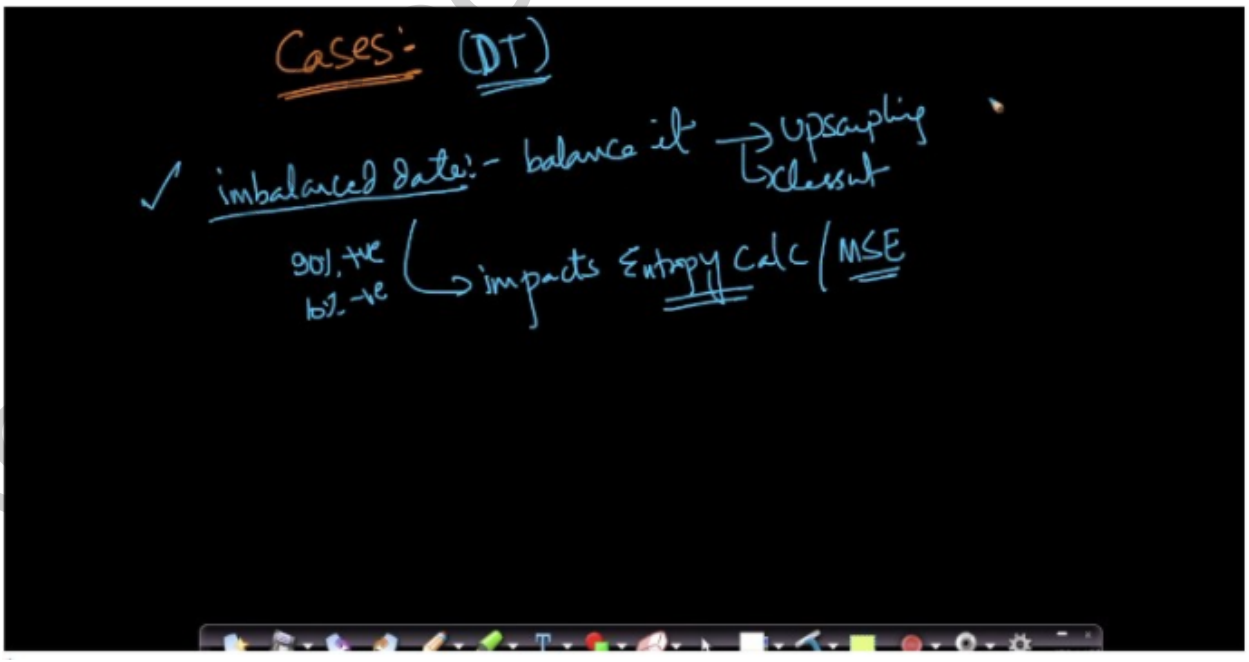


Timestamp 8:39

In the above diagram we have tried to show how a decision tree regressor behaves.

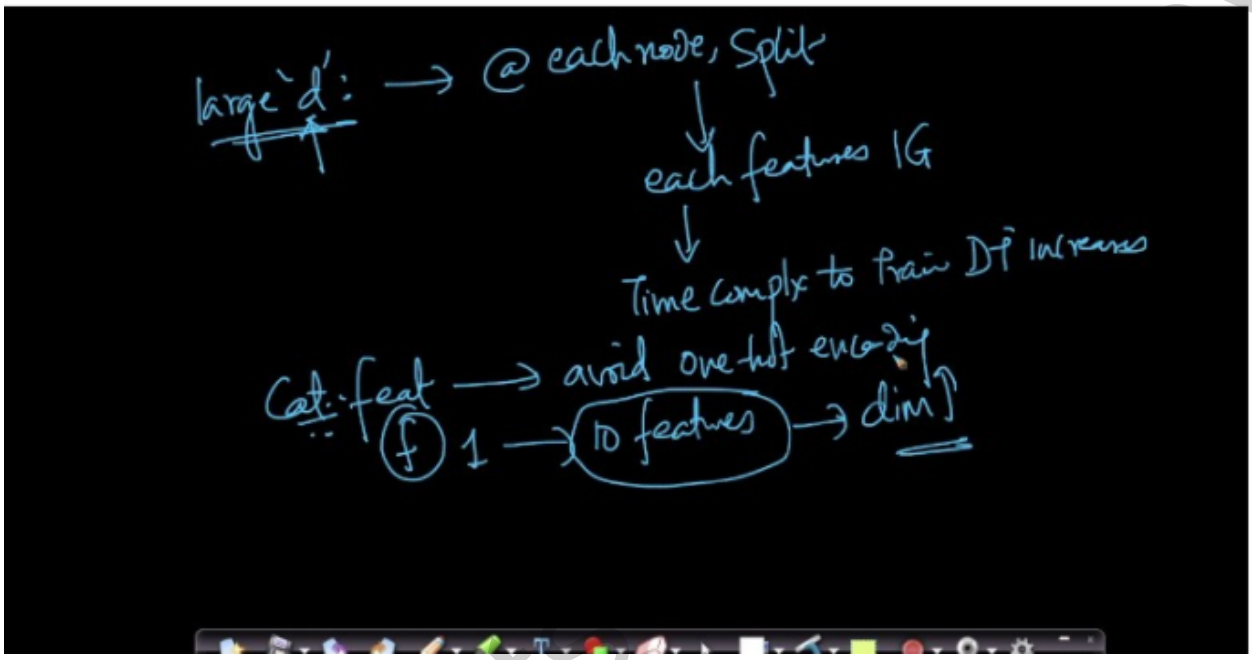
The red line with dotted points is our true function that we want to predict. Now we have fitted a decision tree regressor with max depth = 2 that can be seen in the blue line. We can see that it is underfitting.
Again we have fitted a decision tree regressor with max depth = 5 that can be seen with the green line. We can see that it is fitting even to the outliers in the data. It is overfitting.
The depth parameter behaves in a similar way in both regression and classification. Also if we notice we can see that all the lines are axis parallel just like we saw in classification.

37.13 Cases

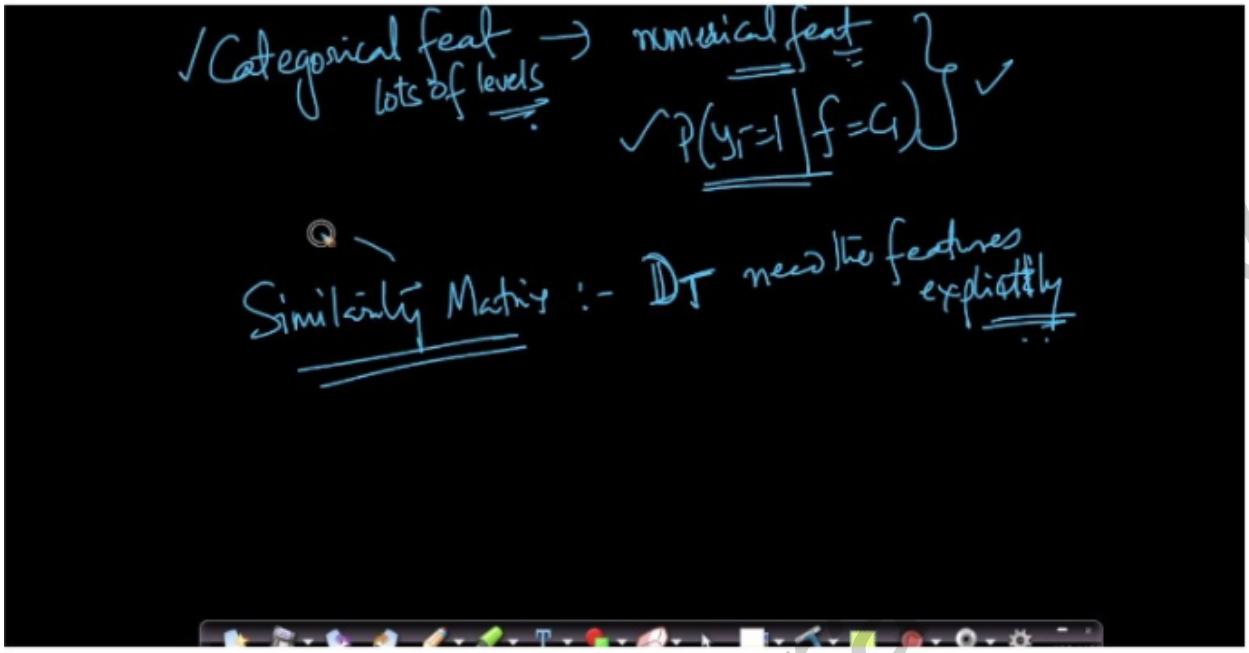


Timestamp 0:59

Let's look at various cases for our decision trees.
1. Imbalanced data: Decision trees are severely impacted by imbalanced data. It gets impacted because the entropy calculation or mse calculation is impacted by an imbalance dataset . We need to prevent it either by oversampling or undersampling.



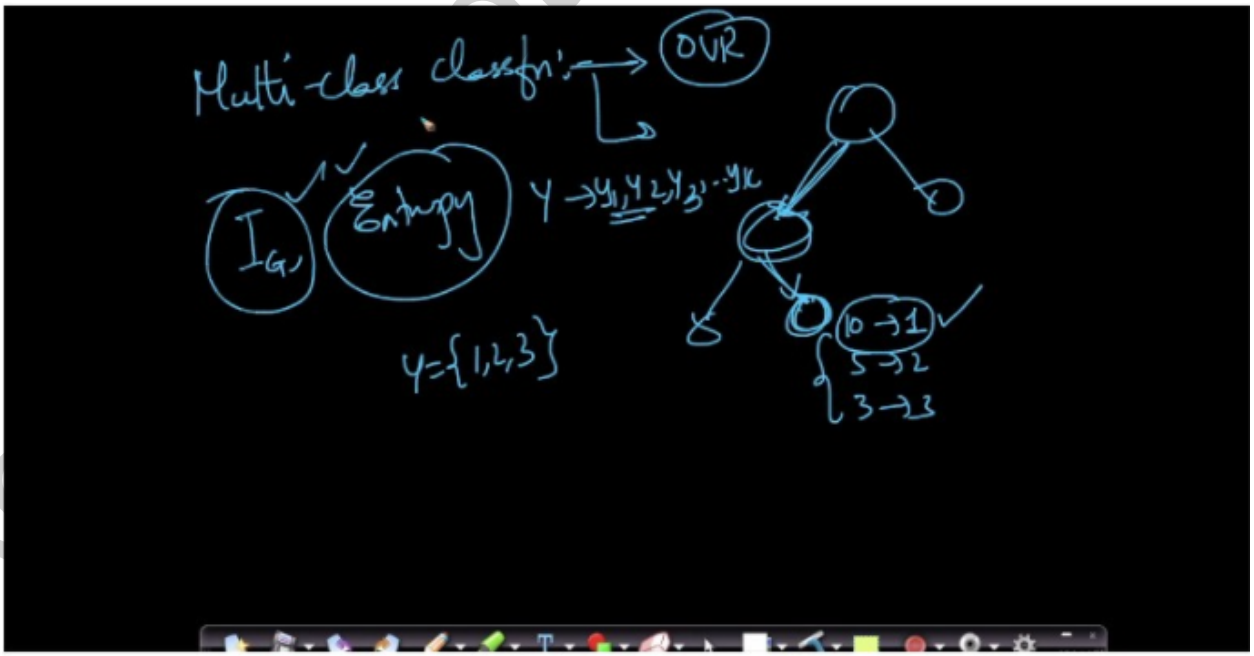
Timestamp 2:22
2. Large 'd': If the number of dimensions increases then during each split we need to consider a large number of features and calculate their information gain/ mse. This can significantly increase the training time.
In decision trees we typically avoid doing One Hot Encoding(OHE) of categorical features if there are large no of categories, because it significantly increases the number of features for consideration leading to larger training time.



Timestamp 3:50

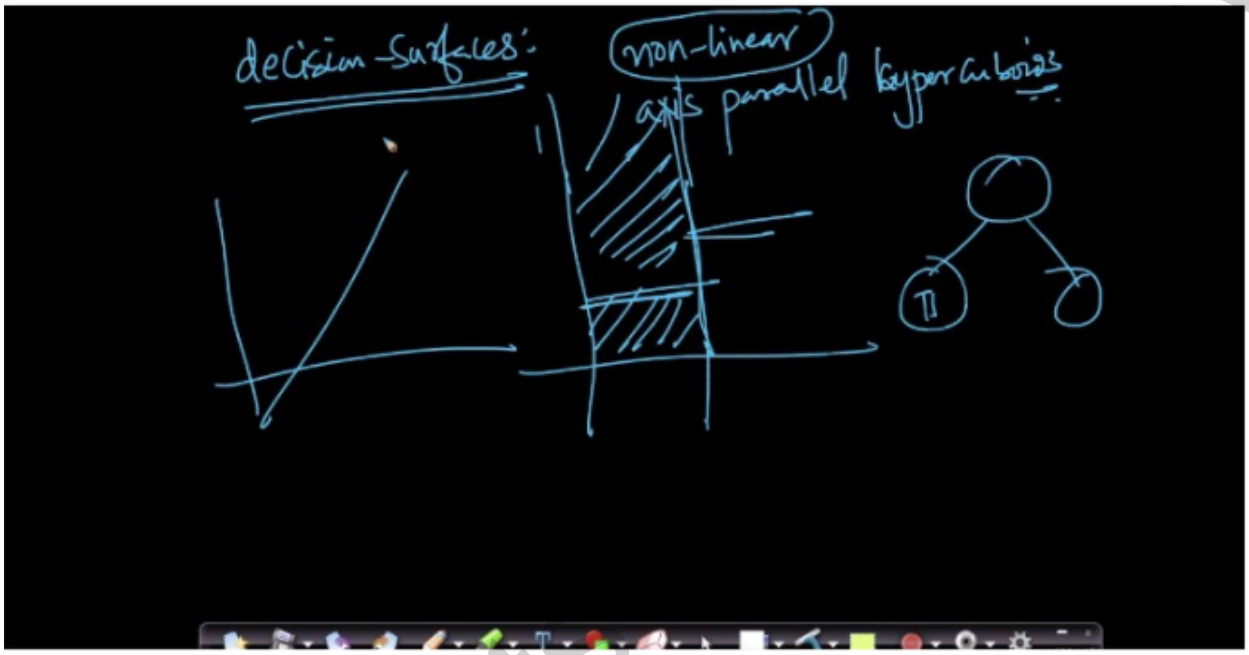
For categorical features having a large number of categories we typically convert them into numerical features as we have seen earlier.

3. Similarity Matrix: Decision trees cannot work with similarity matrices because they need features explicitly to calculate entropy/mse.

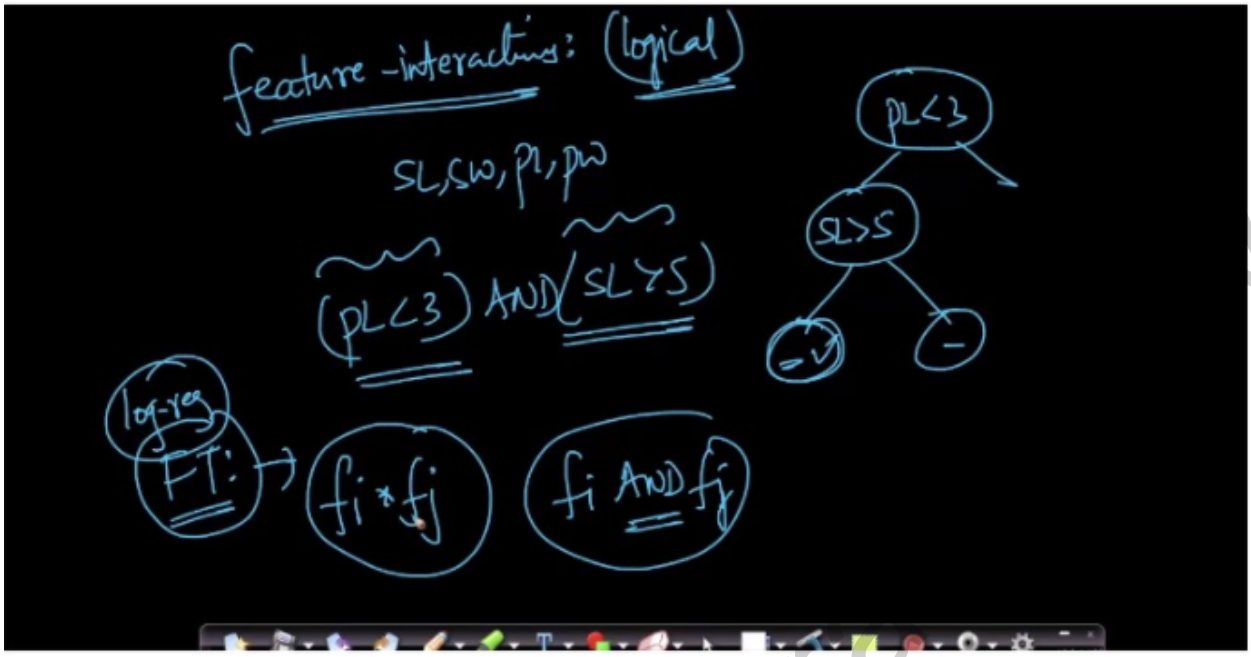


Timestamp 5:13

4. Multiclass Classification: Decision trees can naturally handle multiclass classification, we don't need one vs rest like techniques. Our criteria i.e. mse/entropy is applicable to random variables having more than two possible values. So they can be used effectively for multi class classification problems.
Now for the query point we can simply do a majority vote out of all possible classes as we did for binary cases.

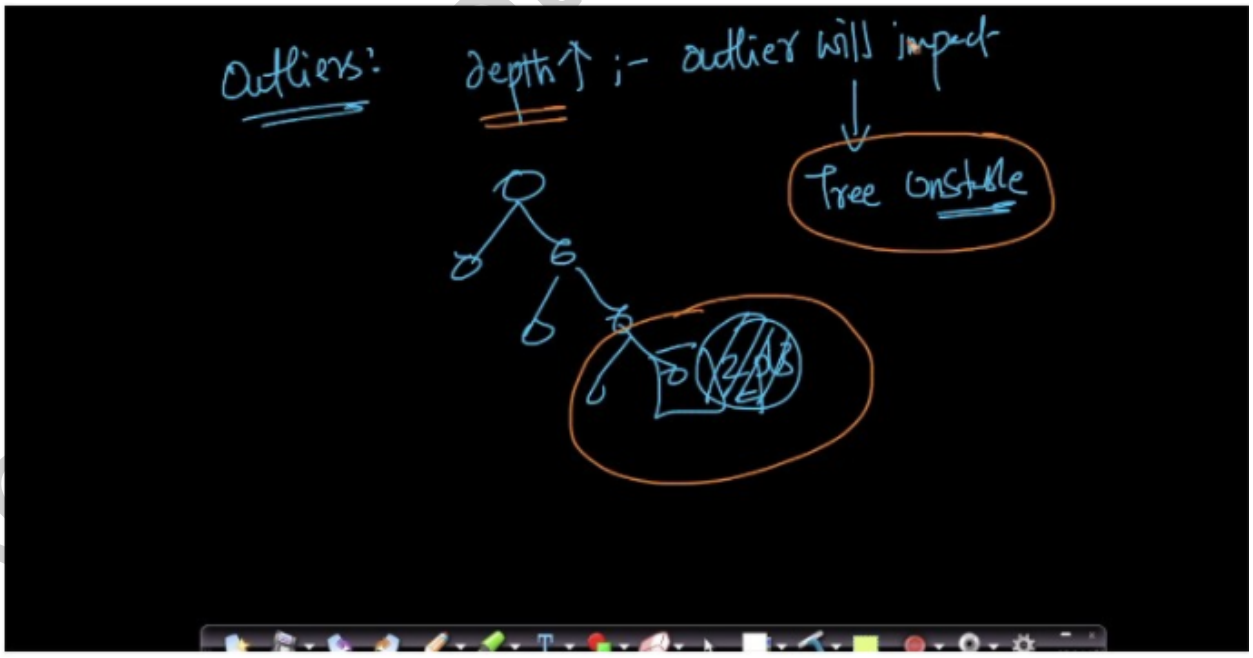


Timestamp 6:15
5. Decision Surfaces: Decision trees produce non-linear decision surfaces. As we have seen earlier, it produces axis parallel hyperplanes essentially breaking up the whole space into hypercubes and hyper cuboids.



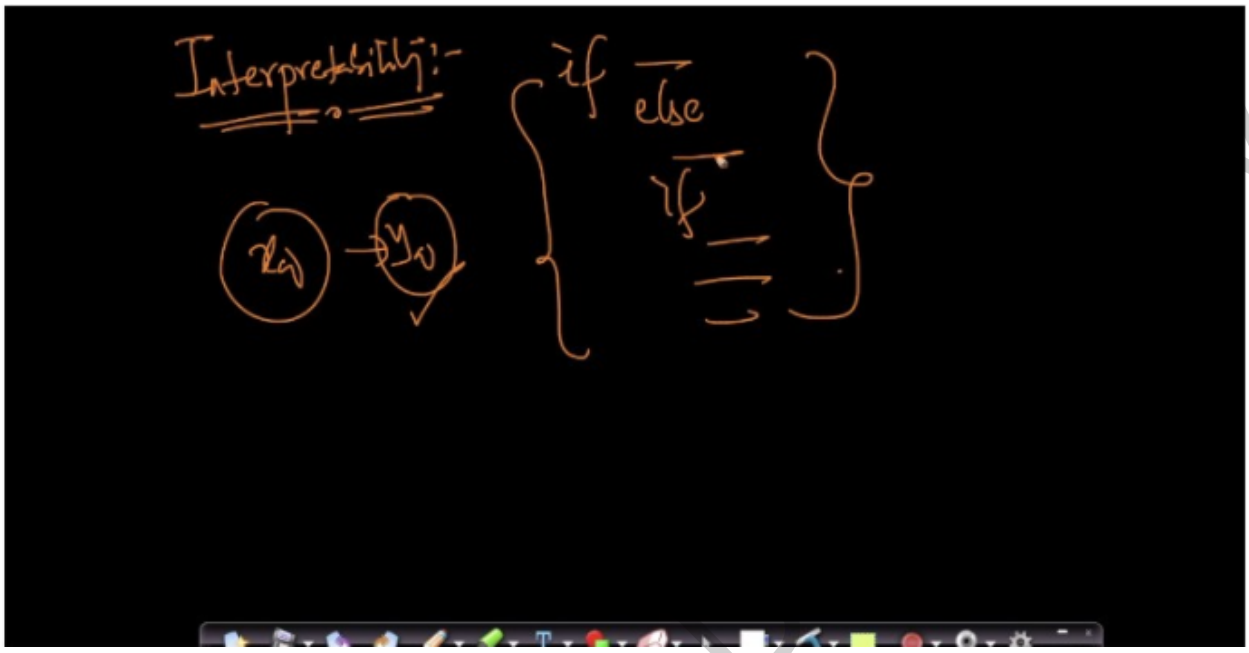
Timestamp 8:20

6. Feature Interactions: Decision trees have an inbuilt feature interaction mechanism. Say we have a tree as shown in the image above. Now, to reach the leftmost leaf we need to have a query point which has PL < 3 and SL > 5. Now these two features PL and SL together made an interaction for predicting a query. This was not possible in techniques like logistic regression where we needed to manually transform features.



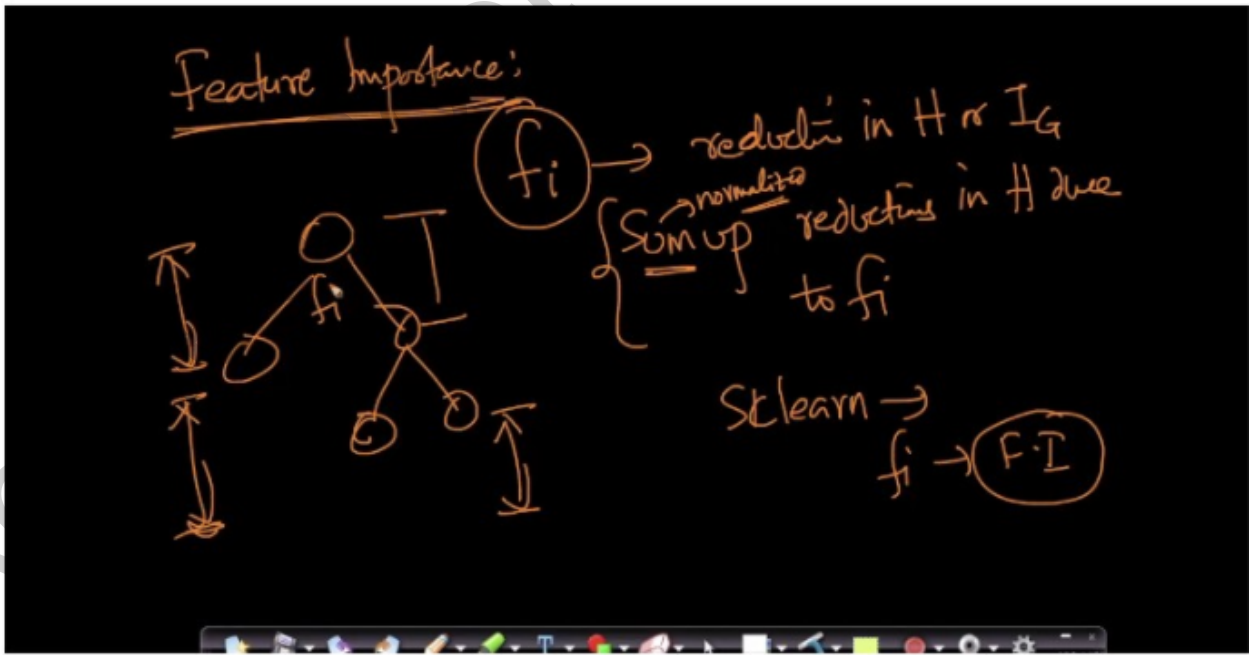
Timestamp 9:25

7. Outliers: Outliers can impact Decision trees especially if we have a tree of large depth. Trees of large depth can fit to outliers making it unstable.



Timestamp 10:11

8. Interpretability: Decision trees are highly interpretable if the depth is reasonable. Everything in a decision tree can be written in the form of nested if else statements. If the depth of the tree increases then interpretability reduces.



Timestamp 11:50

9. Feature Importance: Feature importance can be easily calculated in decision trees. The most important features will be the ones which cause the largest decline of entropy/gini impurity. Libraries like sklearn use this similar technique to calculate feature importance. It maintains a dictionary for each feature and calculates the total reduction caused by a feature in the tree.

37.14 Code Samples

We can see the scikit learn's implementation of Decision Tree here =>

https://scikit-learn.org/stable/modules/tree.html