Decision Tree

1) How decision tree works for a regression problem?

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Decision trees: Key terms

- **Root Node:** The top-most decision node in a decision tree.
- **Decision Node:** A tree node or parent node that splits into one ore more child nodes is called a decision node.
- **Leaf or Terminal Node:** Bottom nodes that (generally speaking) don't split any further.
- **Splitting:** Process of dividing a node into two or more child nodes.
- **Pruning:** The opposite process of splitting. Removing the child nodes of a decision node is called pruning.

Regression Trees:

The split in Regression trees is done by the loss in Mean Squared Error. Here $\hat{y}$ for a particular region is the average of all the $y_i s$ in that region!

The loss in regression trees is represented like this:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \bar{y}_{R_m})^2$$

For regression tree, the algorithm that be used is called CART. CART can handle both classification and regression tasks.

The splitting criteria for CART is MSE(mean squared error).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y_i} - Y_i)^2$$

Suppose we are doing a binary tree. the algorithm first will pick a value, and split the data into two subset. For each subset, it will calculate the $\hat{Y_i}$, and calculate the MSE for each set separately.

$$MSE(node) = \frac{1}{n_i}\sum_{data_i}(\hat{Y_i} - Y_i)^2$$

$$MSE(tree) = \frac{1}{n}\sum_{i=1}^{2}\sum_{data_i}(\hat{Y_i} - Y_i)^2$$

The tree chooses the value with smallest MSE value to split the tree. The $\hat{Y_i}$ for each subset is just the mean value with subset. I will skip this proof, if you have interest on it, you can open a question and require me to answer it.

> The most common method for constructing regression tree is CART (Classification and Regression Tree) methodology, which is also known as *recursive partitioning*.
>
> Take basic regression tree as example:
>
> The method starts by searching for every distinct values of all its predictors, and splitting the value of a predictor that minimizes the following statistic (other regression tree models have different optimization criteria):
>
> $$SSE = \sum_{i\in S_1}(y_i - \overline{y_1}) + \sum_{i\in S_2}(y_i - \overline{y_2})$$
>
> where $\overline{y_1}$ and $\overline{y_2}$ are the average values of the dependent variable in groups S_1 and S_2.
>
> For groups S_1 and S_2, the method will recursively split the predictor values within groups. In practice, the method stops when the sample size of the split group falls below certain threshold, e.g., 50.
>
> To prevent over-fitting, the constructed tree can be pruned by penalizing the SSE with tree size:
>
> $$SSE_{c_p} = SSE + c_p \times S_t$$

where S_t is the size of the tree (number of terminal nodes), and c_p is complexity parameter. Smaller c_p will lead to larger trees, and vice versa. Of course, this parameter can also be tuned by cross-validation.

Unlike linear regression models that calculate the coefficients of predictors, tree regression models calculate the relative importance of predictors. The relative importance of predictors can be computed by summing up the overall reduction of optimization criteria like SSE.

Decision Tree for Regression

When performing regression with a decision tree, we try to divide the given values of X into distinct and non-overlapping regions, e.g. for a set of possible values X1, X2,..., Xp; we will try to divide them into J distinct and non-overlapping regions R1, R2, . . . , RJ. For a given observation falling into the region Rj, the prediction is equal to the mean of the response(y) values for each training observations(x) in the region Rj. The regions R1,R2, . . . , RJ are selected in a way to reduce the following sum of squares of residuals :

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

Where, yrj (second term) is the mean of all the response variables in the region 'j'.

2) Why Recursive binary splitting is called Greedy Approach?

Algorithm is recursive in nature as the groups formed can be sub-divided using same strategy. Due to this procedure, this algorithm is also known as the greedy algorithm, as we have an excessive desire of lowering the cost. This makes the root node as best predictor/classifier.

Greedy Splitting Creating a binary decision tree is actually a process of dividing up the input space. A greedy approach is used to divide the space called recursive binary splitting.

This is a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function. The split with the best cost (lowest cost because we minimize cost) is selected.

All input variables and all possible split points are evaluated and chosen in a greedy manner (e.g. the very best split point is chosen each time).

For regression predictive modeling problems the cost function that is minimized to choose split points is the sum squared error across all training samples that fall within the rectangle:

sum(y – prediction)^2

Where y is the output for the training sample and prediction is the predicted output for the rectangle.

Recursive binary splitting(Greedy approach)

As mentioned above, we try to divide the X values into j regions, but it is very expensive in terms of computational time to try to fit every set of X values into j regions. Thus, decision tree opts for a top-down greedy approach in which nodes are divided into two regions based on the given condition, i.e. not every node will be split but the ones which satisfy the condition are split into two branches. It is called greedy because it does the best split at a given step at that point of time rather than looking for splitting a step for a better tree in upcoming steps. It decides a threshold value(say s) to divide the observations into different regions(j) such that the RSS for Xj>= s and Xj <s is minimum.

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

Here for the above equation, j and s are found such that this equation has the minimum value. The regions R1, R2 are selected based on that value of s and j such that the equation above has the minimum value. Similarly, more regions are split out of the regions created above based on some condition with the same logic. This continues until a stopping criterion (predefined) is achieved. Once all the regions are split, the prediction is made based on the mean of observations in that region.

The process mentioned above has a high chance of overfitting the training data as it will be very complex.

3) What do you understand by Greedy approach?

How do Decision Trees work?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on the type of target variables. Let us look at some algorithms used in Decision Trees:

ID3 → (extension of D3) C4.5 → (successor of ID3) CART → (Classification And Regression Tree) CHAID → (Chi-square automatic interaction detection Performs multi-level splits when computing classification trees) MARS → (multivariate adaptive regression splines)

The ID3 algorithm builds decision trees using a top-down greedy search approach through the space of possible branches with no backtracking. A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment.

Steps in ID3 algorithm:

It begins with the original set S as the root node. On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates Entropy(H) and Information gain(IG) of this attribute. It then selects the attribute which has the smallest Entropy or Largest Information gain. The set S is then split by the selected attribute to produce a subset of the data. The algorithm continues to recur on each subset, considering only attributes never selected before.

Greedy Splitting

Creating a binary decision tree is actually a process of dividing up the input space. A greedy approach is used to divide the space called recursive binary splitting.

This is a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function. The split with the best cost (lowest cost because we minimize cost) is selected.

All input variables and all possible split points are evaluated and chosen in a greedy manner (e.g. the very best split point is chosen each time).

For regression predictive modeling problems the cost function that is minimized to choose split points is the sum squared error across all training samples that fall within the rectangle:

sum(y – prediction)^2

Where y is the output for the training sample and prediction is the predicted output for the rectangle.

For classification the Gini index function is used which provides an indication of how “pure” the leaf nodes are (how mixed the training data assigned to each node is).

G = sum(pk * (1 – pk))

Where G is the Gini index over all classes, pk are the proportion of training instances with class k in the rectangle of interest. A node that has all classes of the same type (perfect class purity) will have G=0, where as a G that has a 50-50 split of classes for a binary classification problem (worst purity) will have a G=0.5.

For a binary classification problem, this can be re-written as:

G = 2 * p1 * p2 or G = 1 – (p1^2 + p2^2)

The Gini index calculation for each node is weighted by the total number of instances in the parent node. The Gini score for a chosen split point in a binary classification problem is therefore calculated as follows:

G = ((1 – (g1_1^2 + g1_2^2)) * (ng1/n)) + ((1 – (g2_1^2 + g2_2^2)) * (ng2/n))

Where G is the Gini index for the split point, g1_1 is the proportion of instances in group 1 for class 1, g1_2 for class 2, g2_1 for group 2 and class 1, g2_2 group 2 class 2, ng1 and ng2 are the total number of instances in group 1 and 2 and n are the total number of instances we are trying to group from the parent node.

Stopping Criterion

The recursive binary splitting procedure described above needs to know when to stop splitting as it works its way down the tree with the training data.

The most common stopping procedure is to use a minimum count on the number of training instances assigned to each leaf node. If the count is less than some minimum then the split is not accepted and the node is taken as a final leaf node.

The count of training members is tuned to the dataset, e.g. 5 or 10. It defines how specific to the training data the tree will be. Too specific (e.g. a count of 1) and the tree will overfit the training data and likely have poor performance on the test set.

Pruning The Tree

The stopping criterion is important as it strongly influences the performance of your tree. You can use pruning after learning your tree to further lift performance.

The complexity of a decision tree is defined as the number of splits in the tree. Simpler trees are preferred. They are easy to understand (you can print them out and show them to subject matter experts), and they are less likely to overfit your data.

The fastest and simplest pruning method is to work through each leaf node in the tree and evaluate the effect of removing it using a hold-out test set. Leaf nodes are removed only if it results in a drop in the overall cost function on the entire test set. You stop removing nodes when no further improvements can be made.

More sophisticated pruning methods can be used such as cost complexity pruning (also called weakest link pruning) where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree.

Greedy nature of Decision Trees

It should be clear that decision stumps are a fairly simple class of models that use only 1 feature at a time and hence, not very accurate for most tasks. A decision tree, on the other hand, considers a lot of features and allows a sequence of split based on these features. It is important to note that they are a very general class of models and can attain high accuracies but the task of finding the optimal decision tree is not feasible computationally and therefore, we use what is called a greedy approach to choose our decision tree.

In this approach, we find the decision stump with the best rule and when the data gets split into two datasets using this rule, we recursively apply the same technique to the two obtained smaller datasets. What is important to learn and remember here is that we would not get the optimal decision tree using this process but what we obtain is a good decision tree to use for our prediction task. Another thing to note is that we can actually split the data such that any object gets completely split by itself and we would obtain each observation in a leaf node with the correct prediction. This would give us a 100% accuracy on our training dataset but this is something that is called overfitting and we would always want to avoid doing this. This is because such a model is not a generalized one and it won't be able to predict well on new and unseen examples. For more information on overfitting, you can look at this blog post by William Koehrsen. Before we move on to implementing decision trees in python, let us look at some advantages and disadvantages in terms of only what we have learned so far.

Advantages

Easy to understand - Decision trees and the underlying principle that they work on are easy to interpret and understand as compared to other complex machine learning algorithms. Fast to learn - Decision trees are relatively quite fast to learn as you will see when you learn about other complex algorithms.

Disadvantages

Difficult to find an optimal set of rules - As we have already discussed, getting an optimal set of rules is hard and it is computationally inefficient to find such a set of rules and we have to use a greedy approach instead. Greedy splitting not accurate - This may require building very deep trees which might not be a good idea.

4) What is Pruning?

Pruning

The performance of a tree can be further increased by pruning. **It involves removing the branches that make use of features having low importance. This way, we reduce the complexity of tree,** and thus increasing its predictive power by reducing overfitting.

Pruning can start at either root or the leaves. The simplest method of pruning starts at leaves and removes each node with most popular class in that leaf, this change is kept if it doesn't deteriorate accuracy. Its also called reduced error pruning. More sophisticated pruning methods can be used such as cost complexity pruning where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree. This is also known as weakest link pruning.

How to avoid/counter Overfitting in Decision Trees?

The common problem with Decision trees, especially having a table full of columns, they fit a lot. Sometimes it looks like the tree memorized the training data set. If there is no limit set on a decision tree, it will give you 100% accuracy on the training data set because in the worse case it will end up making 1 leaf for each observation. Thus this affects the accuracy when predicting samples that are not part of the training set.

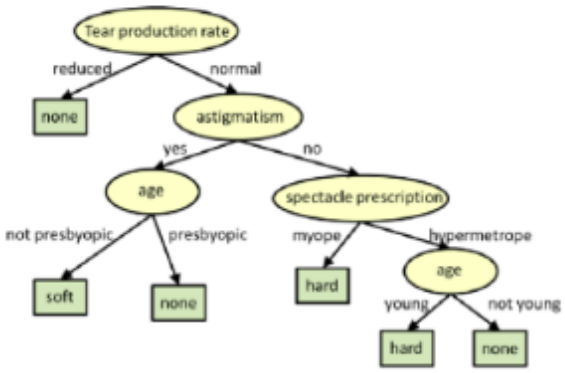
Here are two ways to remove overfitting:

1. Pruning Decision Trees.
2. Random Forest

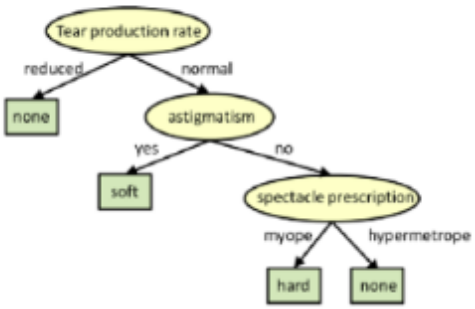
Pruning Decision Trees

The splitting process results in fully grown trees until the stopping criteria are reached. But, the fully grown tree is likely to overfit the data, leading to poor accuracy on unseen data.

In pruning, you trim off the branches of the tree, i.e., remove the decision nodes starting from the leaf node such that the overall accuracy is not disturbed. This is done by segregating the actual training set into two sets: training data set, D and validation data set, V. Prepare the decision tree using the segregated training data set, D. Then continue trimming the tree accordingly to optimize the accuracy of the validation data set, V.



Original Tree



Pruned Tree

Pruning

In the above diagram, the 'Age' attribute in the left-hand side of the tree has been pruned as it has more importance on the right-hand side of the tree, hence removing overfitting.

5) What's the difference between pre pruning and post pruning?

Tree Pruning ¶

Tree pruning is the method of trimming down a full tree (obtained through the above process) to reduce the complexity and variance in the data. Just as we regularised linear regression, we can also regularise the decision tree model by adding a new term.

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Where, T is the subtree which is a subset of the full tree T0 And α is the non-negative tuning parameter which penalises the MSE with an increase in tree length. By using cross-validation, such values of α and T are selected for which our model gives the lowest test error rate. This is how the decision tree regression model works. Let's now see the working algorithm of doing classification using a decision tree. Greedy Algorithm As per Hands-on machine learning book "greedy algorithm greedily searches for an optimum split at the top level, then repeats the process at each level. It does not check whether or not the split will lead to the lowest possible impurity several levels down. A greedy algorithm often produces a reasonably good solution, but it is not guaranteed to be the optimal solution."

Post-pruning

Post-pruning, also known as backward pruning, is the process where the decision tree is generated first and then the non-significant branches are removed. Cross-validation set of data is used to check the effect of pruning and tests whether expanding a node will make an improvement or not. If any improvement is there then we continue by expanding that node else if there is reduction in accuracy then the node not be expanded and should be converted in a leaf node.

Pre-pruning

Pre-pruning, also known as forward pruning, stops the non-significant branches from generating. It uses a condition to decide when should it terminate splitting of some of the branches prematurely as the tree is generated.

==========

Classification and regression trees (CART) CART is one of the most well-established machine learning techniques. In non-technical terms, CART algorithms works by repeatedly finding the best predictor variable to split the data into two subsets. The subsets partition the target outcome better than before the split. Pruning is a technique associated with classification and regression trees.

I am not going to go into details here about what is meant by the best predictor variable, or a better partition. Instead I am going to discuss two enhancements to that basic outline: pruning and early stopping. Sometimes these are referred to simplistically as post-pruning and pre-pruning. As the names suggest, pre-pruning or early stopping involves stopping the tree before it has completed classifying the training set and post-pruning refers to pruning the tree after it has finished. I prefer to differentiate these terms more clearly by using early-stopping and pruning.

Pruning or post-pruning As the name implies, pruning involves cutting back the tree. After a tree has been built (and in the absence of early stopping discussed below) it may be overfitted. The CART algorithm will repeatedly partition data into smaller and smaller subsets until those final subsets are homogeneous in terms of the outcome variable. In practice this often means that the final subsets (known as the leaves of the tree) each consist of only one or a few data points. The tree has learned the data exactly, but a new data point that differs very slightly might not be predicted well.

I will consider 3 pruning strategies,

Minimum error. The tree is pruned back to the point where the cross-validated error is a minimum. Cross-validation is the process of building a tree with most of the data and then using the remaining part of the data to test the accuracy of the decision tree. Smallest tree. The tree is pruned back slightly further than the minimum error. Technically the pruning creates a decision tree with cross-validation error within 1 standard error of the minimum error. The smaller tree is more intelligible at the cost of a small increase in error. None. Early stopping or pre-pruning An alternative method to prevent overfitting is to try and stop the tree-building process early, before it produces leaves with very small samples. This heuristic is known as early stopping but is also sometimes known as pre-pruning decision trees.

At each stage of splitting the tree, we check the cross-validation error. If the error does not decrease significantly enough then we stop. Early stopping may underfit by stopping too early. The current split may be of little benefit, but having made it, subsequent splits more significantly reduce the error.

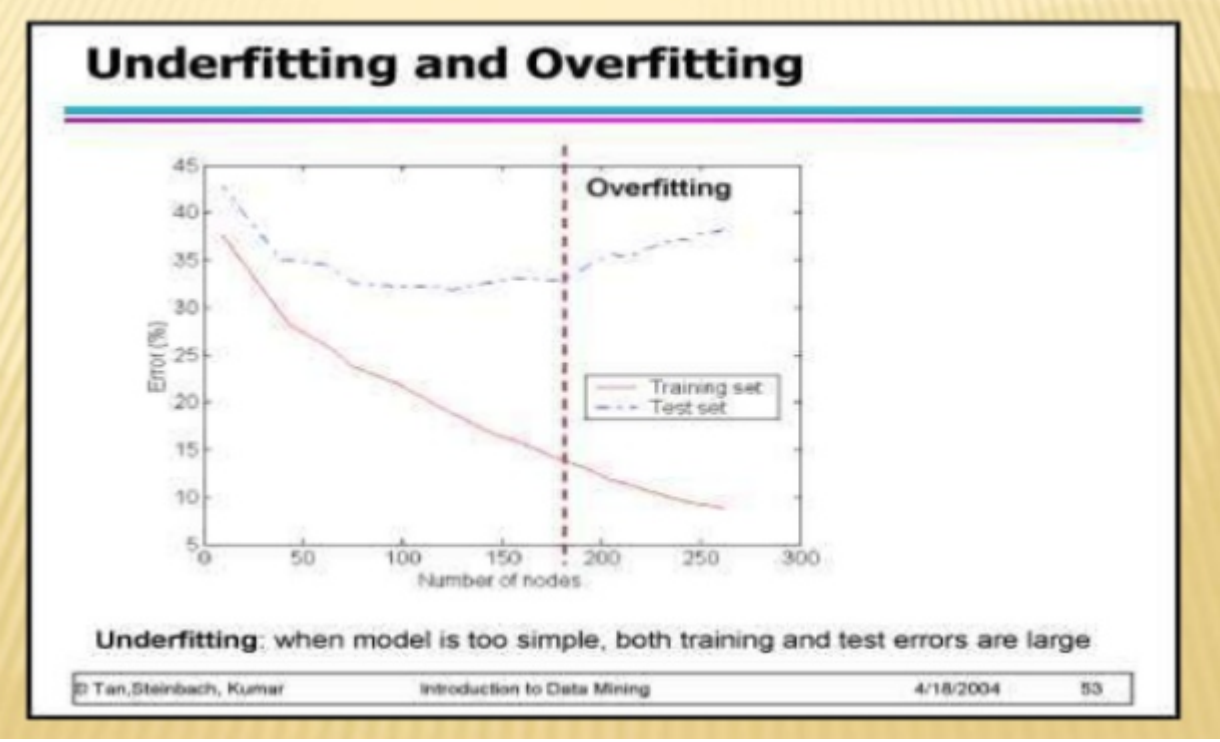
Early stopping and pruning can be used together, separately, or not at all. Post pruning decision trees is more mathematically rigorous, finding a tree at least as good as early stopping. Early stopping is a quick fix heuristic.

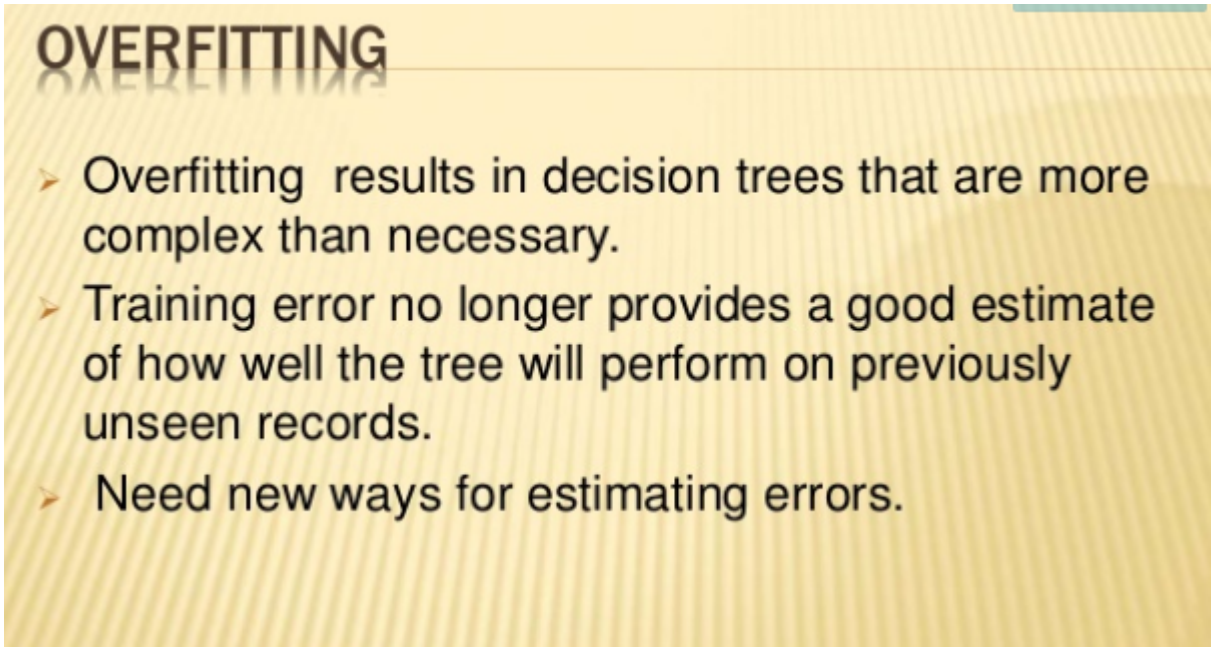
If used together with pruning, early stopping may save time. After all, why build a tree only to prune it back again?

========

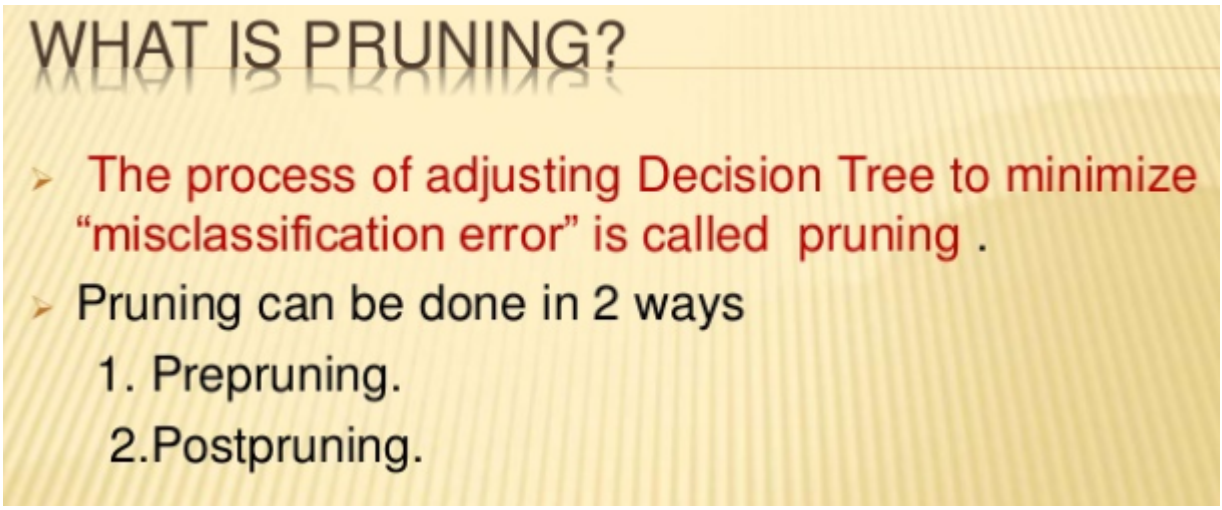
- Decision trees are made to classify the item set.
- While classifying we meet with 2 problems
  1. Underfitting .
  2. Overfitting .

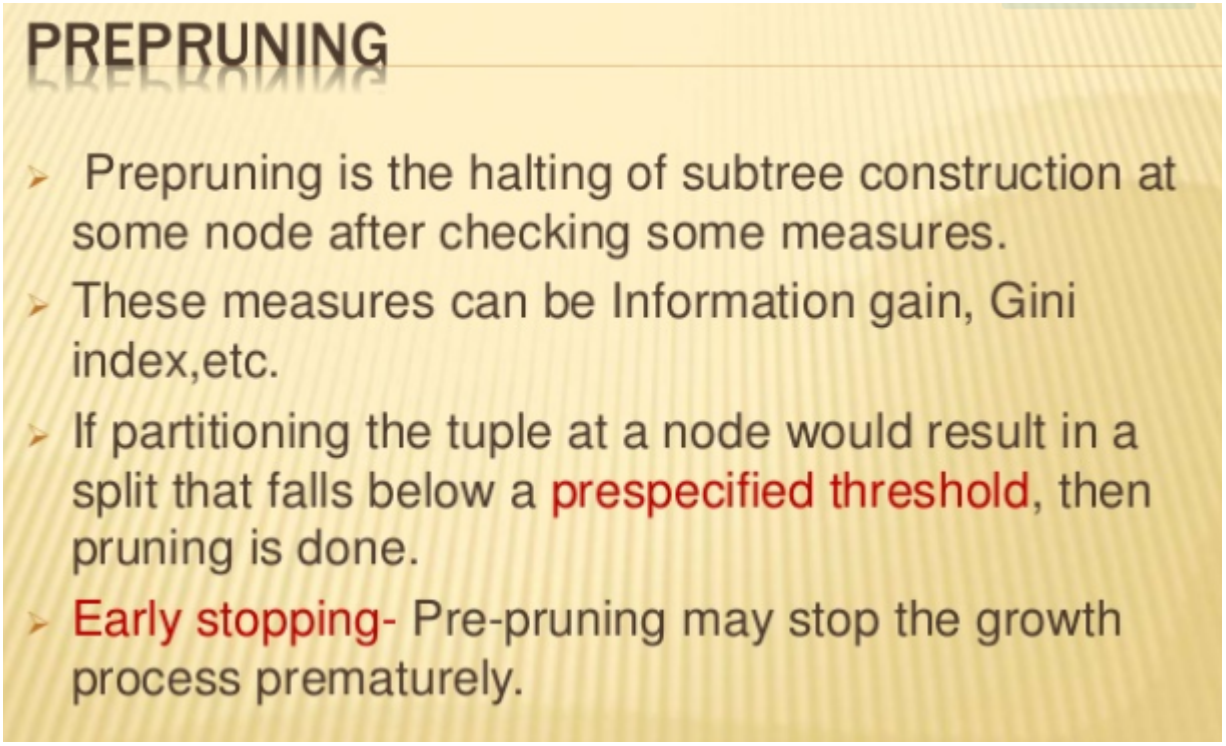
- Underfitting problem arises when both the "training errors and test errors are large"
- This happens when the developed model is made very simple.
- Overfitting problem arises when "training errors are small but test errors are large"

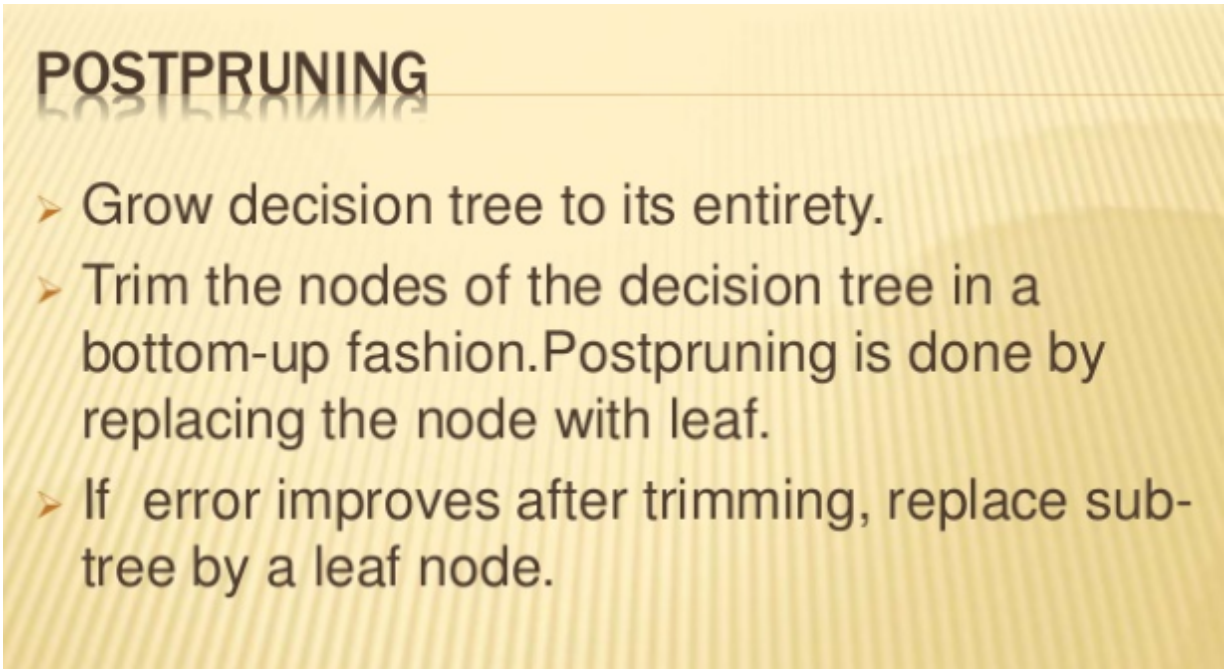


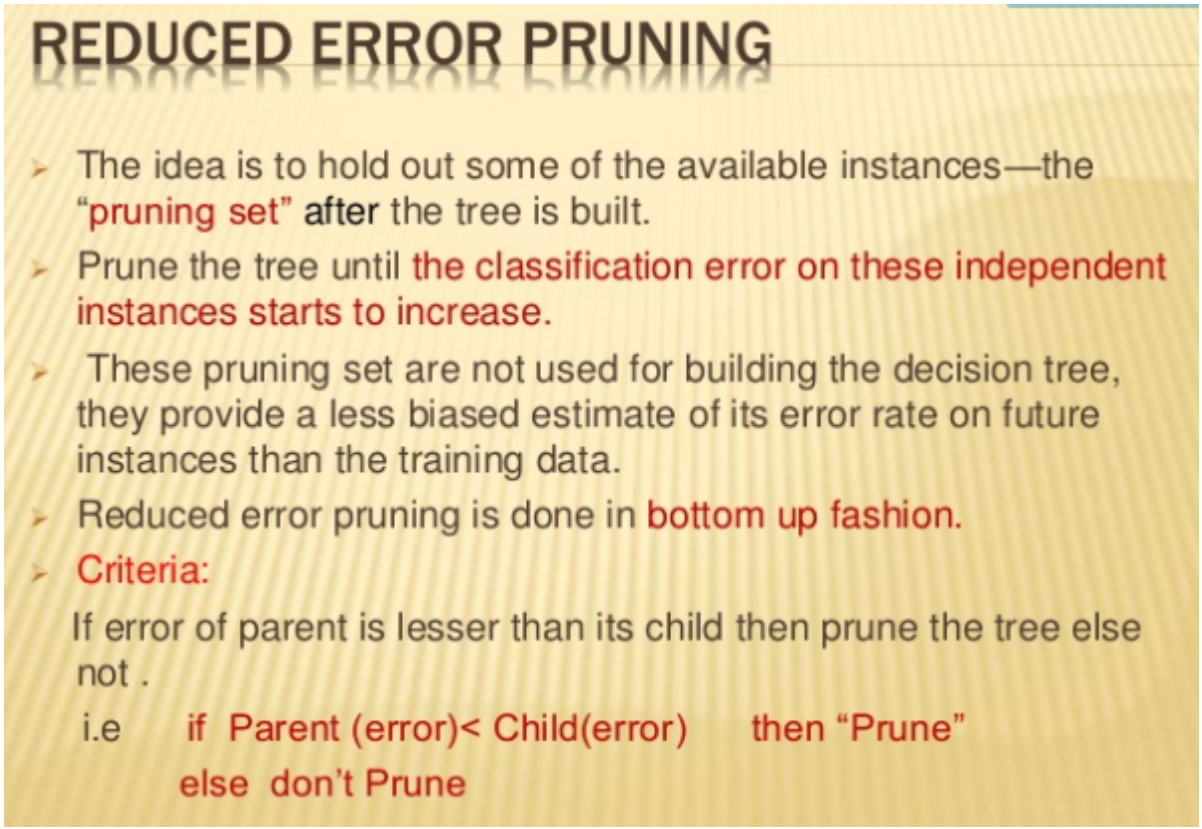

OVERFITTING
Overfitting results in decision trees that are more complex than necessary.
Training error no longer provides a good estimate of how well the tree will perform on previously unseen records.
Need new ways for estimating errors.


How to address overfitting ?
"Tree Pruning"


WHAT IS PRUNING?
The process of adjusting Decision Tree to minimize "misclassification error" is called pruning .
Pruning can be done in 2 ways
1. Prepruning.
2.Postpruning.


PREPRUNING
Prepruning is the halting of subtree construction at some node after checking some measures.
These measures can be Information gain, Gini index,etc.
If partitioning the tuple at a node would result in a split that falls below a prespecified threshold, then pruning is done.
Early stopping- Pre-pruning may stop the growth process prematurely.


POSTPRUNING
Grow decision tree to its entirety.
Trim the nodes of the decision tree in a bottom-up fashion.Postpruning is done by replacing the node with leaf.
If error improves after trimming, replace sub-tree by a leaf node.



========

Why is tree pruning useful in decision tree induction.

When decision trees are built, many of the branches may reflect noise or outliers in the training data.

Tree pruning methods address this problem of overfittingthe data.

Tree pruning attempts to identify and remove such branches, with the goal of improving classification accuracy on unseen data.

Decision trees can suffer from repetition and replication, making them overwhelming to interpret.

Repetition occurs when an attribute is repeatedly tested along a given branch of the tree

In replication, duplicate subtrees exist within the tree.

These situations can impede the accuracy and comprehensibility of a decision tree.

Pruned trees

• These tend to be smaller and less complex and, thus, easier to comprehend.

• They are usually faster and better at correctly classifying independent test data than unpruned trees.

• Pruned trees tend to be more compact than their unpruned counterparts

There are two common approaches to tree pruning:

1.prepruning : In the pre-pruning approach, a tree is “pruned” by halting its construction early (e.g. by deciding not to further split or partition the subset of training tuples at a given node).

When constructing a tree, measures such as statistical significance, information gain, Gini index, and so on can be used to assess the goodness of a split.

If partitioning the tuples at a node would result in a split that falls below a pre specified threshold, then further partitioning of the given subset is halted.

There are difficulties, however, in choosing an appropriate threshold.

High thresholds could result in oversimplified trees, whereas low thresholds could result in very little simplification.

2.post pruning.

The second and more common approach is post pruning, which removes subtrees from a “fully grown” tree.

A subtree at a given node is pruned by removing its branches and replacing it with a leaf.

The leaf is labeled with the most frequent class among the subtree being replaced.

The cost complexity pruning algorithm used in CART is an example of the post pruning approach.

The basic idea is that the simplest solution is preferred.

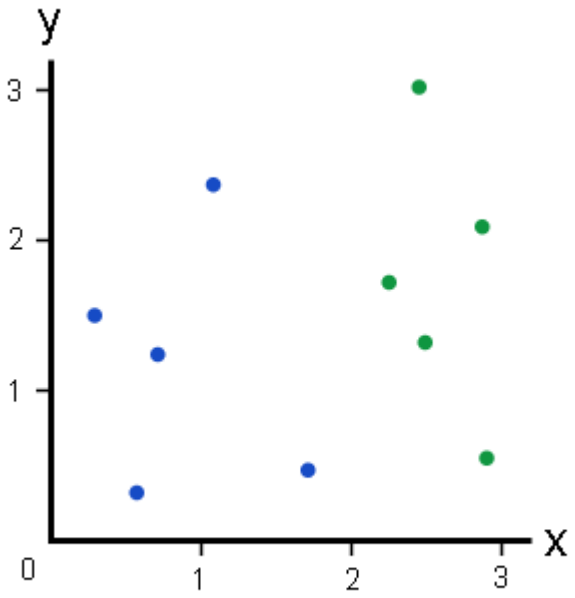
Unlike cost complexity, pruning does not require an independent set of tuples.

Post pruning leads to a more reliable tree.

6) What is Entropy? How is it calculated?

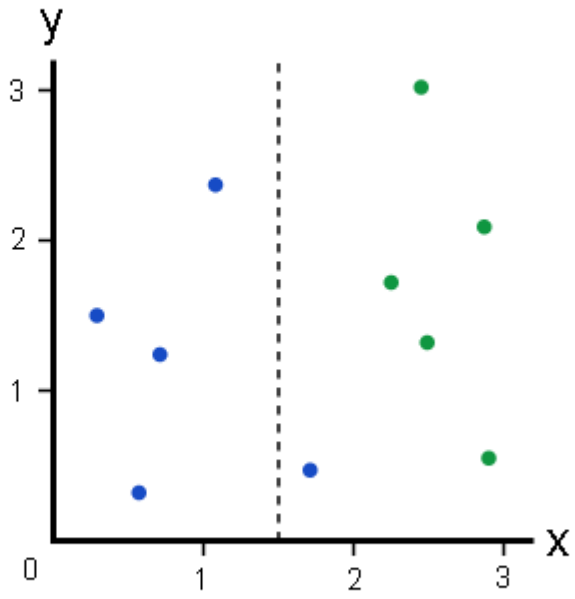
A Simple Explanation of Information Gain and Entropy

Information Gain, like Gini Impurity, is a metric used to train Decision Trees. Specifically, these metrics measure the quality of a split. For example, say we have the following data:



The Dataset

What if we made a split at $x = 1.5$?



An Imperfect Split

This imperfect split breaks our dataset into these branches:

- Left branch, with 4 blues. ● ● ● ●
- Right branch, with 1 blue and 5 greens. ● ● ● ● ● ●

It's clear this split isn't optimal, but how good is it? **How can we *quantify* the quality of a split?**

That's where Information Gain comes in.

Information Entropy

Before we get to Information Gain, we have to first talk about Information Entropy. In the context of training Decision Trees, Entropy can be roughly thought of as **how much variance the data has**. For example:

- A dataset of only blues ●●●● would have very **low** (in fact, zero) entropy.
- A dataset of mixed blues, greens, and reds ●●●●●● would have relatively **high** entropy.

Here's how we calculate Information Entropy for a dataset with C classes:

$$E = -\sum_{i}^{C} p_i \log_2 p_i$$

where p_i is the probability of randomly picking an element of class i (i.e. the proportion of the dataset made up of class i).

The easiest way to understand this is with an example. Consider a dataset with 1 blue, 2 greens, and 3 reds: ●●●●●●. Then

$$E = -(p_b \log_2 p_b + p_g \log_2 p_g + p_r \log_2 p_r)$$

We know $p_b = \frac{1}{6}$ because $\frac{1}{6}$ of the dataset is blue. Similarly, $p_g = \frac{2}{6}$ (greens) and $p_r = \frac{3}{6}$ (reds). Thus,

$$\begin{aligned} E &= -(\frac{1}{6}\log_2(\frac{1}{6}) + \frac{2}{6}\log_2(\frac{2}{6}) + \frac{3}{6}\log_2(\frac{3}{6})) \\ &= \boxed{1.46} \end{aligned}$$

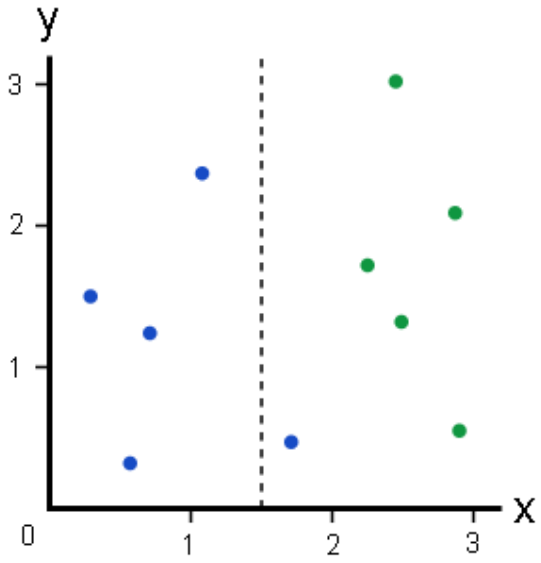
What about a dataset of all one color? Consider 3 blues as an example: ●●●. The entropy would be

$$E = -(1\log_2 1) = \boxed{0}$$

Information Gain

It's finally time to answer the question we posed earlier: **how can we quantify the quality of a split?**

Let's consider this split again:



An Imperfect Split

Before the split, we had 5 blues and 5 greens, so the entropy was

$$\begin{aligned} E_{before} &= -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) \\ &= \boxed{1} \end{aligned}$$

After the split, we have two branches.

Left Branch has 4 blues, so $E_{left} = \boxed{0}$ because it's a dataset of all one color.

Right Branch has 1 blue and 5 greens, so

$$\begin{aligned} E_{right} &= -(\frac{1}{6}\log_2(\frac{1}{6}) + \frac{5}{6}\log_2(\frac{5}{6})) \\ &= \boxed{0.65} \end{aligned}$$

Now that we have the entropies for both branches, we can determine the quality of the split by **weighting the entropy of each branch by how many elements it has**. Since Left Branch has 4 elements and Right Branch has 6, we weight them by 0.4 and 0.6, respectively:

$$\begin{aligned} E_{split} &= 0.4 * 0 + 0.6 * 0.65 \\ &= \boxed{0.39} \end{aligned}$$

We started with $E_{before} = 1$ entropy before the split and now are down to 0.39! **Information Gain = how much Entropy we removed**, so

$$\text{Gain} = 1 - 0.39 = \boxed{0.61}$$

This makes sense: **higher Information Gain = more Entropy removed**, which is what we want. In the perfect case, each branch would contain only one color after the split, which would be zero entropy!

Recap

Information Entropy can be thought of as how how unpredictable a dataset is.

- A set of only one class (say, blue ●●●) is extremely predictable: anything in it is blue. This would have **low** entropy.
- A set of many mixed classes ●●● is unpredictable: a given element could be any color! This would have **high** entropy.

The actual formula for calculating Information Entropy is:

$$E = -\sum_{i}^{C} p_i \log_2 p_i$$

Information Gain is calculated for a split by subtracting the weighted entropies of each branch from the original entropy. When training a Decision Tree using these metrics, the best split is chosen by maximizing Information Gain.

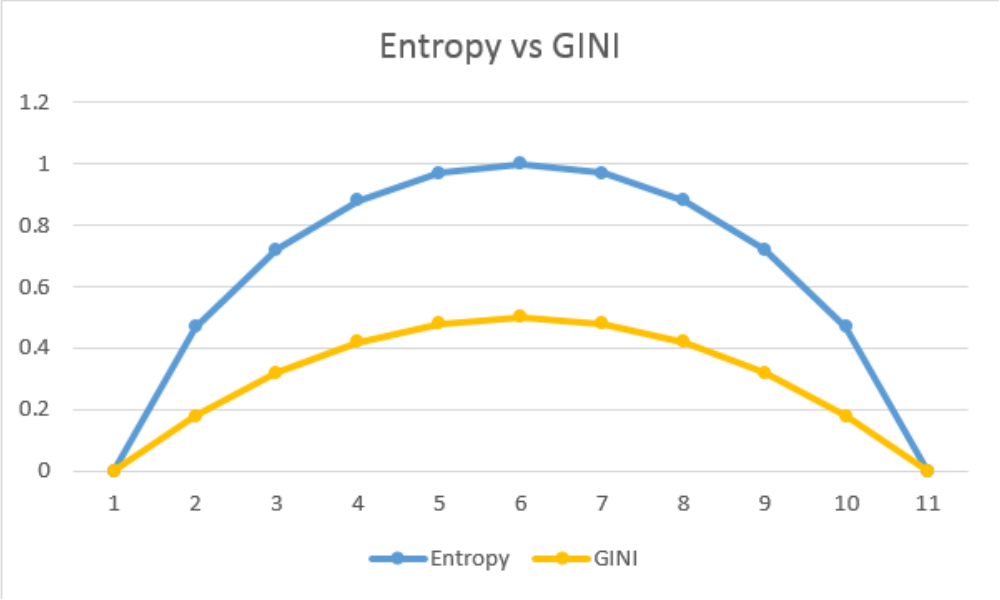
7) What is Gini Impurity?

Gini Impurity

Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

Gini impurity is **lower bounded by 0**, with 0 occurring if the data set contains only one class.



There are many algorithms there to build a decision tree. They are

1. **CART** (Classification and Regression Trees) — This makes use of Gini impurity as metric.
2. **ID3** (Iterative Dichotomiser 3) — This uses entropy and information gain as metric.

Advantages of CART

- Simple to understand, interpret, visualize.
- Decision trees *implicitly perform variable screening or feature selection.*
- Can *handle both numerical and categorical data.* Can also *handle multi-output problems.*
- Decision trees require relatively *little effort from users for data preparation.*
- *Nonlinear relationships between parameters do not affect tree performance.*

Disadvantages of CART

- Decision-tree learners *can create over-complex trees* that do not generalize the data well. This is called *overfitting*.
- Decision trees can be unstable because *small variations in the data might result in a completely different tree being generated.* This is called *variance*, which needs to be *lowered by methods like bagging and boosting*.
- Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
- Decision tree learners create *biased trees if some classes dominate*. It is therefore recommended to balance the data set prior to fitting with the decision tree.

8) What do you understand by Information Gain? How does it help in tree building?

What Is Information Gain?

Information Gain, or IG for short, measures the reduction in entropy or surprise by splitting a dataset according to a given value of a random variable.

A larger information gain suggests a lower entropy group or groups of samples, and hence less surprise.

You might recall that information quantifies how surprising an event is in bits. Lower probability events have more information, higher probability events have less information. Entropy quantifies how much information there is in a random variable, or more specifically its probability distribution. A skewed distribution has a low entropy, whereas a distribution where events have equal probability has a larger entropy.

In information theory, we like to describe the “surprise” of an event. Low probability events are more surprising therefore have a larger amount of information. Whereas probability distributions where the events are equally likely are more surprising and have larger entropy.

Skewed Probability Distribution (unsurprising): Low entropy. Balanced Probability Distribution (surprising): High entropy.

9) How does node selection take place while building a tree?

Type *Markdown* and LaTeX: α^2

10) What are different algorithms available for decision tree?

Different Algorithms for Decision Tree ID3 (Iterative Dichotomiser) : It is one of the algorithms used to construct decision tree for classification. It uses Information gain as the criteria for finding the root nodes and splitting them. It only accepts categorical attributes.

C4.5 : It is an extension of ID3 algorithm, and better than ID3 as it deals both continuous and discreet values.It is also used for classfication purposes.

Classfication and Regression Algorithm(CART) : It is the most popular algorithm used for constructing decison trees. It uses ginni impurity as the default calculation for selecting root nodes, however one can use "entropy" for criteria as well. This algorithm works on both regression as well as classfication problems. We will use this algorithm in our pyhton implementation. Entropy and Ginni impurity can be used reversibly. It doesn't affects the result much. Although, ginni is easier to compute than entropy, since entropy has a log term calculation. That's why CART algorithm uses ginni as the default algorithm.

If we plot ginni vs entropy graph, we can see there is not much difference between them:

Advantages of Decision Tree: It can be used for both Regression and Classification problems. Decision Trees are very easy to grasp as the rules of splitting is clearly mentioned. Complex decision tree models are very simple when visualized. It can be understood just by visualising. Scaling and normalization are not needed. Disadvantages of Decision Tree: A small change in data can cause instability in the model because of the greedy approach. Probability of overfitting is very high for Decision Trees. It takes more time to train a decision tree model than other classification algorithms.

11) What’s the main difference between Gini Impurity and Entropy on the basis of computation time?

Both Gini Impurity and Entropy are criteria to split a node in a decision tree. They are standard metrics to compute “impurity” or “information level”. They guide to split a node in the decision tree only based on the information that exists at that node. They are computed as below.

Gini=1−∑cj=1p2j Entropy=−∑cj=1pjlogpj These equations show that Gini Impurity and Entropy are computed in a similar fashion. However, Gini Impurity is calculated with less computation, i.e., no logarithmic function is involved. That is it!

The criteria to split nodes in a decision tree are different for supervised or unsupervised algorithms. In supervised algorithms, we use the Gini Impurity or Entropy to split nodes since data labels are known. In unsupervised algorithms, we use a clustering quality metric (an overall performance metric) to split nodes. We can compute clustering quality using different methods based on inter and intra-cluster distances.

Both Gini Impurity and Entropy are **criteria to split a node in a decision tree**. They are standard metrics to compute "impurity" or "information level". They guide to split a node in the decision tree only based on the information that exists at that node. They are computed as below.

- $Gini = 1 - \sum_{j=1}^{c} p_j^2$
- $Entropy = -\sum_{j=1}^{c} p_j \log p_j$

These equations show that Gini Impurity and Entropy are computed in a similar fashion. However, Gini Impurity is calculated with less computation, i.e., no logarithmic function is involved. That is it!

The criteria to split nodes in a decision tree are different for supervised or unsupervised algorithms. In supervised algorithms, we use the Gini Impurity or Entropy to split nodes since data labels are known. In unsupervised algorithms, we use a clustering quality metric (an overall performance metric) to split nodes. We can compute clustering quality using different methods based on inter and intra-cluster distances.

Gini impurity and Entropy both are used to find the equality or randomness of the dataset when it is split into the nodes. This mathematical approach is used in the Decision tree for the classification algorithm.

This is the formula for GINI impurity

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

where (pi) is the probability of class i in a node.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Gini:- It shows that the data point that we have randomly chosen for the splitting from the dataset how it is incorrectly labeled. It is used for **CART** (Classification and regression tree) and it gives more accurate value and less than the entropy index which is its best quality, the small values show **less impurity**.

Entropy:- It is not easy to understand, It is more computationally complex because of the log in the equation. It gives larger values that are not much use for the splitting criteria. Higher values give inappropriate result, and it is used for C4.5 decision tree.

Example :

We have a total of 4 balls of red and blue Now we will find the Gini index and entropy

Gini index for 4 red and 0 blue

Gini index = 1-((Probability of red)^2+(Probability of blue)^2)

Gini index = 1-((1)^2+(0)^2)

Gini index = 0

Gini index for 2 red and 2 blue

Gini index = 1-((Probability of red)^2+(Probability of blue)^2)

Gini index = 1-((0.5)^2+(0.5)^2)

Gini index = 0.5

Gini index for 3 red and 1 blue

Gini index = 1-((Probability of red)^2+(Probability of blue)^2)

Gini index = 1-((0.75)^2+(0.25)^2)

Gini index = 0.375

Hence we can see the probability of the data points with the Gini gain

Entropy index for 4 red and 0 blue

Entropy = [(Probabolity of red)*log2(Probability of red)] — [(Probabolity of blue)*log2(Probability of blue)]

Entropy = [(4/4)*log(4/4)] — [(0/4*log2(0/4)]

Entropy = 0

Entropy index for 2 red and 2 blue

Entropy = [(Probabolity of red)*log2(Probability of red)] — [(Probabolity of blue)*log2(Probability of blue)]

Entropy = [(2/4)*log(2/4)] — [(2/4*log2(2/4)]

Entropy = 1

Entropy index for 3 red and 1 blue

Entropy = [(Probabolity of red)*log2(Probability of red)] — [(Probabolity of blue)*log2(Probability of blue)]

Entropy = [(3/4)*log(3/4)] — [(1/4*log2(1/4)]

Entropy = 0.811

So there are difference between the Gini and entropy to find the best probability for the datapoint while splitting the data in decision tree.

12) What are the disadvantages and advantages of using a Decision Tree?

Type *Markdown* and LaTeX: α^2

13) How do you deploy model in Heroku?

Type *Markdown* and LaTeX: α^2

14) What challenges you faced while deploying the model?

Type *Markdown* and LaTeX: α^2

Cross Validation

1) What is Cross validation?

In general, we partition the dataset into training and test sets. Then, call the fit method on the training set to build the model and apply the model on the test set to estimate the target value and evaluate the model's performance. The reason why we divided the data into training and test sets was to use the test set to estimate how well the model trained on the training data and how well it would perform on the unseen data.

However, cross-validation is a method that goes beyond evaluating a single model using a single train and test split of the data. It is applied to more subsets created using the training dataset and each of which is used to train and evaluate a separate model. That is, we split our training dataset into k subsets and the ith model will be built on the union of all subsets except the ith. We then test for the performance of the model i on the ith part.

Cross validation in machine learning is a technique that provides an accurate measure of the performance of a machine learning model. This performance will be closer to what you can expect when the model is used on a future unseen dataset.

The application of the machine learning models is to learn from the existing data and use that knowledge to predict future unseen events. The cross validation in machine learning model needs to be thoroughly done to profitably trade in live trading

Here are my five reasons why you should use Cross-Validation:

1. Use All Your Data

When we have very little data, splitting it into training and test set might leave us with a very small test set. Say we have only 100 examples, if we do a simple 80–20 split, we'll get 20 examples in our test set. It is not enough. We can get almost any performance on this set only due to chance. The problem is even worse when we have a multi-class problem. If we have 10 classes and only 20 examples, It leaves us with only 2 examples for each class on average. Testing anything on only 2 examples can't lead to any real conclusion. If we use cross-validation in this case, we build K different models, so we are able to make predictions on all of our data. For each instance, we make a prediction by a model that didn't see this example, and so we are getting 100 examples in our test set. For the multi-class problem, we get 10 examples for each class on average, and it's much better than just 2. After we evaluated our learning algorithm (see #2 below) we are now can train our model on all our data because if our 5 models had similar performance using different train sets, we assume that by training it on all the data will get similar performance. By doing cross-validation, we're able to use all our 100 examples both for training and for testing while evaluating our learning algorithm on examples it has never seen before.

2. Get More Metrics

As mentioned in #1, when we create five different models using our learning algorithm and test it on five different test sets, we can be more confident in our algorithm performance. When we do a single evaluation on our test set, we get only one result. This result may be because of chance or a biased test set for some reason. By training five (or ten) different models we can understand better what's going on. Say we trained five models and we use accuracy as our measurement. We could end up in several different situations. The best scenario is that our accuracy is similar in all our folds, say 92.0, 91.5, 92.0, 92.5 and 91.8. This means that our algorithm (and our data) is consistent and we can be confident that by training it on all the data set and deploy it in production will lead to similar performance. However, we could end up in a slightly different scenario, say 92.0, 44.0, 91.5, 92.5 and 91.8. These results look very strange. It looks like one of our folds is from a different distribution, we have to go back and make sure that our data is what we think it is. The worst scenario we can end up in is when we have considerable variation in our results, say 80, 44, 99, 60 and 87. Here it looks like that our algorithm or our data (or both) is no consistent, it could be that our algorithm is unable to learn, or our data is very complicated. By using Cross-Validation, we are able to get more metrics and draw important conclusion both about our algorithm and our data.

3. Use Models Stacking

Sometimes we want to (or have to) build a pipeline of models to solve something. Think about Neural Networks for example. We can create many layers. Each layer may use previews layer output and learn a new representation of our data so eventually, it will be able to produce good predictions. We are able to train those different layers because we use the back-propagation algorithm. Each layer computes its error and passes it back to the previous layer. When we do something similar but not using Neural Networks, we can't train it in the same way because there's not always a clear "error" (or derivative) that we can pass back. For example, we may create a Random Forest Model that predicts something for us, and right after that, we want to do a Linear Regression that will rely on previous predictions and produce some real number. The critical part here is that our second model must learn on the predictions of our first model. The best solution here is to use two different datasets for each model. We train our Random Forest on dataset A. Then we use dataset B to make a prediction using it. Then we use the dataset B predictions to train our second model (the logistic regression) and finally, we use dataset C to evaluate our complete solution. We make predictions using the first model, pass them to our second model and then compare it to the ground truth. When we have limited data (as in most cases), we can't really do it. Also, we can't train both our models on the same dataset because then, our second model learns on predictions that our first model already seen. These will probably be over-fitted or at least have better results than on a different set. This means that our second algorithm is trained not on what it will be tested on. This may lead to different effects in our final evaluations that will be hard to understand. By using cross-validation, we can make predictions on our dataset in the same way as described before and so our second's models input will be real predictions on data that our first model never seen before.

4. Work with Dependent/Grouped Data

When we perform a random train-test split of our data, we assume that our examples are independent. That means that by knowing/seeing some instance will not help us understand other instances. However, that's not always the case. Consider a speech recognition system. Our data may include different speakers saying different words. Let's look at spoken digits

recognition. In this dataset, for example, there are 3 speakers and 1500 recordings (500 for each speaker). If we do a random split, our training and test set will share the same speaker saying the same words! This is, of course, will boost our algorithm performance but once tested on a new speaker, our results will be much worse. The proper way to do it is to split the speakers, i.e., use 2 speakers for training and use the third for testing. However, then we'll test our algorithm only on one speaker. It is not enough. We need to know how our algorithm performs on different speakers. We can use cross-validation on the speakers level. We will train 3 models, each time using one speaker for testing and two others for training. This way we'll be able to evaluate better our algorithm (as described above) and finally build our model on all speakers.

5. Parameters Fine-Tuning

This is one of the most common and obvious reasons to do cross validation. Most of the learning algorithms require some parameters tuning. It could be the number of trees in Gradient Boosting classifier, hidden layer size or activation functions in a Neural Network, type of kernel in an SVM and many more. We want to find the best parameters for our problem. We do it by trying different values and choosing the best ones. There are many methods to do this. It could be a manual search, a grid search or some more sophisticated optimization. However, in all those cases we can't do it on our training test and not on our test set of course. We have to use a third set, a validation set. By splitting our data into three sets instead of two, we'll tackle all the same issues we talked about before, especially if we don't have a lot of data. By doing cross-validation, we're able to do all those steps using a single set.

2) Why do we need to implement Cross validation?

So why is this better than our original method of a single train and test split?

Well, you may have noticed that by choosing a different length for the train and the test data split, the model performance will vary quite a bit, depending on the specific samples that happen to end up in the training set. Cross-validation gives more stable estimates of how the model is likely to perform on average, instead of relying completely on a single training set.

The most common type of cross-validation technique is the k-fold cross-validation. To do five-fold cross-validation, the training dataset is partitioned into five parts of equal or close to equal size. Each of these parts is called a "fold". The first model is trained using folds 1 through 4 and evaluated on fold 5. The second model is trained using folds 1, 2, 3, and 5 and evaluated on fold 4, and so on. When this process is done, we have five accuracy values, one for each model. It's typical to then compute the mean and standard deviation of all the accuracy scores and use it to compare how accurate we can expect the model to be on average.

Why Cross Validation in Machine Learning is Used?

Cross validation machine learning technique is very useful for evaluating the effectiveness of your model mainly when you need to mitigate over-fitting. However, it is also used in determining the hyper parameters of your model, in terms of finding that which parameters will give results in lowest test error. It is one of the most widely used and effective technique of machine learning model validation used by the machine learning engineers worldwide to create a fully functional AI model with best level of accuracy for flawless results.

Cogito is providing human-backed ML validation service to check the accuracy of models in unbiased manner at affordable pricing. It is specialized in validating the models developed for different fields like AI-enabled CCTV cameras to capture the people and other moving objects on the computer or authenticating the facial recognition annotations used into various fields to detect the human faces and authenticate the facial attributes in right manner.

Cross validation is necessary when dataset for training and testing is too small. To avoid overfitting problem, the dataset is usually divided into N random parts with equal volume. The method is then trained with N-1 parts and tested with the remaining part. The overall metric is calculated as average of the metrics on the N times training - test runs.

==========

Cross validation doesn't help you to avoid overfitting. Two important parameters you get from cross-validation are: accuracy and Kappa (a parameter of confusion matrix). Both helps you to get insight how your model could be to new dataset or, it provides how much robust your predictive model is. However, in my opinion, the confusion matrix slightly indicates the chances of overfitting. This is my intuition only!

To avoid over-fitting you have to follow the typical rules like standardization, PCA, feature engineering etc. Another important thing is hyperparameter optimization. It sometimes provides better cross validation accuracy, but somehow may over fit to new data. Trust me, if your model does have many hyperparameters (for example: Noisy Random Forest/Gradient Boosting Trees), chances that your cross validation will provide poor insights increase.

That being said, end of the day, everything depends on the amount and quality of data. This is something I always believe. Only by judging the performance on test dataset and comparing training and testing errors, you can be sure whether your model overfits or underfits. Sad to say, if your dataset is small, cross validation (both k-fold and repeated) can't help you much. But, from my experience, if your dataset is large enough and if the value of kappa is between 0.5–0.8 (highly arguable topic) , the model wouldn't overfit and will perform quite the same way it does during validation. Lastly, in my view, while cross-validating and deciding what to believe, prioritize this way:

Repeated K-fold CV > k-fold CV > LOOCV (Leave One Out CV)

3) What are different types of CV methods?

Variations on Cross-Validation There are a number of variations on the k-fold cross validation procedure.

Three commonly used variations are as follows:

Train/Test Split: Taken to one extreme, k may be set to 2 (not 1) such that a single train/test split is created to evaluate the model. LOOCV: Taken to another extreme, k may be set to the total number of observations in the dataset such that each observation is given a chance to be the held out of the dataset. This is called leave-one-out cross-validation, or LOOCV for short. Stratified: The splitting of data into folds may be governed by criteria such as ensuring that each fold has the same proportion of observations with a given categorical value, such as the class

outcome value. This is called stratified cross-validation. Repeated: This is where the k-fold cross-validation procedure is repeated n times, where importantly, the data sample is shuffled prior to each repetition, which results in a different split of the sample.

Cross validation is kind of model validation technique used machine learning. It is basically used the subset of the data-set and then assess the model predictions using the complementary subset of the data-set. K-fold cross-validation is one of the popular method used under this technique to evaluate the model on the subset that was not used for training the model.

K-fold Cross Validation

Under the K-fold cross validation the entire data is divided into k subsets and holdout method is repeated k times such that each time one of the k subsets is used as the test set/validation set and the other k-1 subsets are put together to form a training set.

This popular method because it is simple to understand all machine learning engineers because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

Process of K-fold Cross Validation

First Process: Shuffle the dataset randomly.

Second Process: Split the dataset into k groups.

For each unique group:

Step 1: Take the group as a hold out or test data set

Step 2: Take the remaining groups as a training data set

Step 3: Fit a model on the training set and evaluate it on the test set

Step 4: Retain the evaluation score and discard the model

Third Process: Summarize the skill of the model using the sample of model evaluation scores.

It is very much important that in the each observation in the data sample is assigned to an individual group and remains in that group for the duration of the procedure. That means each sample is provided the chance to used in the hold out set 1 times and used to train the model k-1 times. As, the results of a k-fold cross-validation run are often summarized with the mean of the model skill scores.

Leave One Out Cross Validation

Similarly, LOOCV (Leave One Out Cross Validation) is another cross validation method the validation process is performed by training on the whole data-set except only one data-point of the available data-set and then iterates for each data point. The benefit of using this method is that it leads to higher variation in the testing model as we are testing against one data point.

While the disadvantage of using this method is it takes a lot of execution time as it iterates over “the number of data points” times. This method is generally choose over the previous one because it does not suffer from the intensive computation, as number of possible combinations is equal to number of data points in original sample or n.

Validation In this method, we perform training on the 50% of the given data-set and rest 50% is used for the testing purpose. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e higher bias.

LOOCV (Leave One Out Cross Validation) In this method, we perform training on the whole data-set but leaves only one data-point of the available data-set and then iterates for each data-point. It has some advantages as well as disadvantages also. An advantage of using this method is that we make use of all data points and hence it is low bias. The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over 'the number of data points' times.

K-Fold Cross Validation In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

Note: It is always suggested that the value of k should be 10 as the lower value of k is takes towards validation and higher value of k leads to LOOCV method. Example The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here, we have total 25 instances. In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training([1-5] testing and [5-25] training) while in the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training([5-10] testing and [1-5 and 10-25] training), and so

==========

Validation

This process of deciding whether the numerical results quantifying hypothesized relationships between variables, are acceptable as descriptions of the data, is known as validation. Generally, an error estimation for the model is made after training, better known as evaluation of residuals. In this process, a numerical estimate of the difference in predicted and original responses is done, also called the training error. However, this only gives us an idea about how well our model does on data used to train it. Now its possible that the model is underfitting or overfitting the data. So, the problem with this evaluation technique is that it does not give an indication of how well the learner will generalize to an independent/ unseen data set. Getting this idea about our model is known as Cross Validation.

Holdout Method

Now a basic remedy for this involves removing a part of the training data and using it to get predictions from the model trained on rest of the data. The error estimation then tells how our model is doing on unseen data or the validation set. This is a simple kind of cross validation technique, also known as the holdout method. Although this method doesn't take any overhead to compute and is better than traditional validation, it still suffers from issues of high variance. This is because it is not certain which data points will end up in the validation set and the result might be entirely different for different sets.

K-Fold Cross Validation

As there is never enough data to train your model, removing a part of it for validation poses a problem of underfitting. By reducing the training data, we risk losing important patterns/ trends in data set, which in turn increases error induced by bias. So, what we require is a method that provides ample data for training the model and also leaves ample data for validation. K Fold cross validation does exactly that. In K Fold cross validation, the data is divided into k subsets. Now the holdout method is repeated k times, such that each time, one of the k subsets is used as the test set/ validation set and the other k-1 subsets are put together to form a training set. The error estimation is averaged over all k trials to get total effectiveness of our model. As can be seen, every data point gets to be in a validation set exactly once, and gets to be in a training set k-1 times. This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set. Interchanging the training and test sets also adds to the effectiveness of this method. As a general rule and empirical evidence, K = 5 or 10 is generally preferred, but nothing's fixed and it can take any value.

Stratified K-Fold Cross Validation

In some cases, there may be a large imbalance in the response variables. For example, in dataset concerning price of houses, there might be large number of houses having high price. Or in case of classification, there might be several times more negative samples than positive samples. For such problems, a slight variation in the K Fold cross validation technique is made, such that each fold contains approximately the same percentage of samples of each target class as the complete set, or in case of prediction problems, the mean response value is approximately equal in all the folds. This variation is also known as Stratified K Fold. Above explained validation techniques are also referred to as Non-exhaustive cross validation methods. These do not compute all ways of splitting the original sample, i.e. you just have to decide how many subsets need to be made. Also, these are approximations of method explained below, also called Exhaustive Methods, that computes all possible ways the data can be split into training and test sets.

Leave-P-Out Cross Validation

This approach leaves p data points out of training data, i.e. if there are n data points in the original sample then, n-p samples are used to train the model and p points are used as the validation set. This is repeated for all combinations in which original sample can be separated this way, and then the error is averaged for all trials, to give overall effectiveness. This method is exhaustive in the sense that it needs to train and validate the model for all possible combinations, and for moderately large p, it can become computationally infeasible. A particular case of this method is when p = 1. This is known as Leave one out cross validation. This method is generally preferred over the previous one because it does not suffer from the intensive computation, as number of possible combinations is equal to number of data points in original sample or n.

Cross Validation is a very useful technique for assessing the effectiveness of your model, particularly in cases where you need to mitigate overfitting. It is also of use in determining the hyper parameters of your model, in the sense that which parameters will result in lowest test error. This is all the basic you need to get started with cross validation. You can get started with all kinds of validation techniques using Scikit-Learn, that gets you up and running with just a few lines of code in python.

=======

Simple K-Folds — We split our data into K parts, let's use K=3 for a toy example. If we have 3000 instances in our dataset, We split it into three parts, part 1, part 2 and part 3. We then build three different models, each model is trained on two parts and tested on the third. Our first model is trained on part 1 and 2 and tested on part 3. Our second model is trained to on part 1 and part 3 and tested on part 2 and so on.

Leave One Out — This is the most extreme way to do cross-validation. For each instance in our dataset, we build a model using all other instances and then test it on the selected instance.

Stratified Cross Validation — When we split our data into folds, we want to make sure that each fold is a good representative of the whole data. The most basic example is that we want the same proportion of different classes in each fold. Most of the times it happens by just doing it randomly, but sometimes, in complex datasets, we have to enforce a correct distribution for each fold.

4) How bias and variance varies for each CV method?

Type *Markdown* and LaTeX: α^2

5) Is Train Test Split a kind of CV? True or False.

Type *Markdown* and LaTeX: α^2

6) How can we check over fitting using CV?

Cross-validation helps to tune hyperparameter. hence CV is a way to avoid overfit or underfit.

The idea behind cross-validation is the same as with a single holdout validation set, to estimate the model's predictive performance on unseen data. Cross-validation simply does this more robustly, by repeating the experiment multiple times, using all the different parts of the training set as validation sets. This gives a more accurate indication of how well the model generalizes to unseen data.

In other words, cross-validation does not prevent overfitting in itself, but it may help in identifying a case of overfitting.

Why breaking data in to another partition(validation data) solves the problem? when validation data is used to find optimal hyper parameter. The test data acts as unseen data points to the trained model. Now when we measure accuracy on test data we will ensure that it generalizes well and perform well on future unseen data points.

Type *Markdown* and LaTeX: α^2

In [ ]:

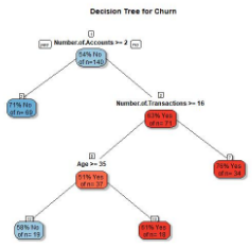
What is a Decision Tree? How does it work?

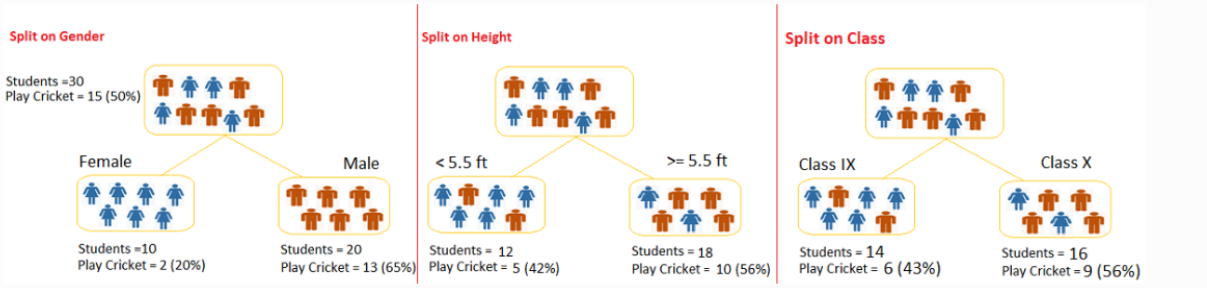
Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

Example:

Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class(IX/ X) and Height (5 to 6 ft). 15 out of these 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.

This is where decision tree helps, it will segregate the students based on all values of three variable and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other). In the snapshot below, you can see that variable Gender is able to identify best homogeneous sets compared to the other two variables.





As mentioned above, decision tree identifies the most significant variable and it's value that gives best homogeneous sets of population. Now the question which arises is, how does it identify the variable and the split? To do this, decision tree uses various algorithms, which we will shall discuss in the following section.

Types of Decision Trees

Types of decision tree is based on the type of target variable we have. It can be of two types:

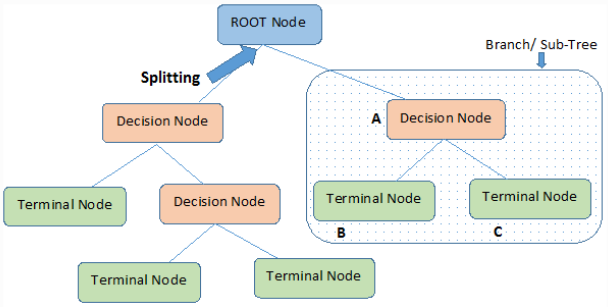
- **Categorical Variable Decision Tree:** Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example: In above scenario of student problem, where the target variable was "Student will play cricket or not" i.e. YES or NO.
- **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Example: Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that income of customer is a significant variable but insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product and various other variables. In this case, we are predicting values for continuous variable.

Important Terminology related to Decision Trees

Let's look at the basic terminology used with Decision trees:

- **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
- **Splitting:** It is a process of dividing a node into two or more sub-nodes.
- **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
- **Leaf/Terminal Node:** Nodes do not split is called Leaf or Terminal node.



Note:- A is parent node of B and C.

- **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
- **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
- **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

These are the terms commonly used for decision trees. As we know that every algorithm has advantages and disadvantages, below are the important factors which one should know.

Advantages

- **Easy to Understand**: Decision tree output is very easy to understand even for people from non-analytical background. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive and users can easily relate their hypothesis.
- **Useful in Data exploration:** Decision tree is one of the fastest way to identify most significant variables and relation between two or more variables. With the help of decision trees, we can create new variables / features that has better power to predict target variable. You can refer article (Trick to enhance power of regression model) for one such trick. It can also be used in data exploration stage. For example, we are working on a problem where we have information available in hundreds of variables, there decision tree will help to identify most significant variable.
- **Less data cleaning required:** It requires less data cleaning compared to some other modeling techniques. It is not influenced by outliers and missing values to a fair degree.
- **Data type is not a constraint:** It can handle both numerical and categorical variables.
- **Non Parametric Method:** Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

Disadvantages

- **Over fitting:** Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning (discussed in detailed below).
- **Not fit for continuous variables**: While working with continuous numerical variables, decision tree looses information when it categorizes variables in different categories.

Regression Trees vs Classification Trees

We all know that the terminal nodes (or leaves) lies at the bottom of the decision tree. This means that decision trees are typically drawn upside down such that leaves are the the bottom & roots are the tops (shown below).

Both the trees work almost similar to each other, let's look at the primary differences & similarity between classification and regression trees:

- Regression trees are used when dependent variable is continuous. Classification trees are used when dependent variable is categorical.
- In case of regression tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mean value.
- In case of classification tree, the value (class) obtained by terminal node in the training data is the mode of observations falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mode value.
- Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions. For the sake of simplicity, you can think of these regions as high dimensional boxes or boxes.
- Both the trees follow a top-down greedy approach known as recursive binary splitting. We call it as 'top-down' because it begins from the top of tree when all the observations are available in a single region and successively splits the predictor space into two new branches down the tree. It is known as 'greedy' because, the algorithm cares (looks for best variable available) about only the current split, and not about future splits which will lead to a better tree.
- This splitting process is continued until a user defined stopping criteria is reached. For example: we can tell the the algorithm to stop once the number of observations per node becomes less than 50.
- In both the cases, the splitting process results in fully grown trees until the stopping criteria is reached. But, the fully grown tree is likely to overfit data, leading to poor accuracy on unseen data. This bring 'pruning'. Pruning is one of the technique used tackle overfitting. We'll learn more about it in following section.

How does a tree decide where to split?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria is different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on type of target variables. Let's look at the four most commonly used algorithms in decision tree:

Gini Index

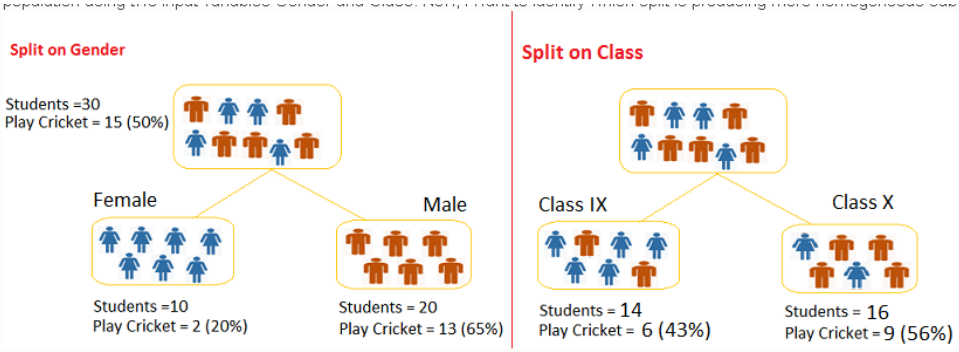
Gini index says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.

- It works with categorical target variable "Success" or "Failure".
- It performs only Binary splits
- Higher the value of Gini higher the homogeneity.
- CART (Classification and Regression Tree) uses Gini method to create binary splits.

Steps to Calculate Gini for a split

- Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure (p^2+q^2).
- Calculate Gini for split using weighted Gini score of each node of that split

Example: – Referring to example used above, where we want to segregate the students based on target variable (playing cricket or not). In the snapshot below, we split the population using two input variables Gender and Class. Now, I want to identify which split is producing more homogeneous sub-nodes using Gini index.



Split on Gender:

- Calculate, Gini for sub-node Female = (0.2)*(0.2)+(0.8)*(0.8)=0.68
- Gini for sub-node Male = (0.65)*(0.65)+(0.35)*(0.35)=0.55
- Calculate weighted Gini for Split Gender = (10/30)*0.68+(20/30)*0.55 = **0.59**

Similar for Split on Class:

- Gini for sub-node Class IX = (0.43)*(0.43)+(0.57)*(0.57)=0.51
- Gini for sub-node Class X = (0.56)*(0.56)+(0.44)*(0.44)=0.51
- Calculate weighted Gini for Split Class = (14/30)*0.51+(16/30)*0.51 = **0.51**

Above, you can see that Gini score for *Split on Gender* is higher than *Split on Class*, hence, the node split will take place on Gender.

Chi-Square

Chi-Square

It is an algorithm to find out the statistical significance between the differences between sub-nodes and parent node. We measure it by sum of squares of standardized differences between observed and expected frequencies of target variable.

- It works with categorical target variable "Success" or "Failure".
- It can perform two or more splits.
- Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.
- Chi-Square of each node is calculated using formula,
- Chi-square = ((Actual – Expected)^2 / Expected)^1/2
- It generates tree called CHAID (Chi-square Automatic Interaction Detector)

Steps to Calculate Chi-square for a split:

- Calculate Chi-square for individual node by calculating the deviation for Success and Failure both
- Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

Example: Let's work with above example that we have used to calculate Gini.
Split on Gender:

- First we are populating for node Female, Populate the actual value for "**Play Cricket**" and **"Not Play Cricket"**, here these are 2 and 8 respectively.
- Calculate expected value for "**Play Cricket**" and "**Not Play Cricket**", here it would be 5 for both because parent node has probability of 50% and we have applied same probability on Female count(10).
- Calculate deviations by using formula, Actual – Expected. It is for "**Play Cricket**" (2 – 5 = -3) and for "**Not play cricket**" (8 – 5 = 3).
- Calculate Chi-square of node for "**Play Cricket**" and "**Not Play Cricket**" using formula with formula, **= ((Actual – Expected)^2 / Expected)^1/2**. You can refer below table for calculation.
- Follow similar steps for calculating Chi-square value for Male node.
- Now add all Chi-square values to calculate Chi-square for split Gender.

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
Female	2	8	10	5	5	-3	3	1.34	1.34
Male	13	7	20	10	10	3	-3	0.95	0.95
							Total Chi-Square	**4.58**	

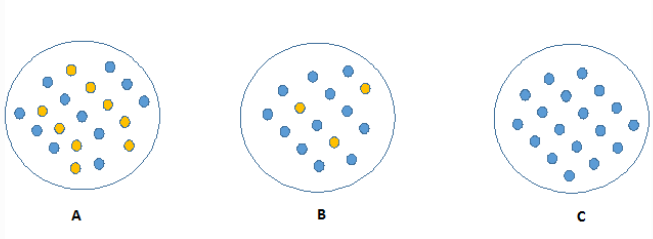
Split on Class:
Perform similar steps of calculation for split on Class and you will come up with below table.

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
IX	6	8	14	7	7	-1	1	0.38	0.38
X	9	7	16	8	8	1	-1	0.35	0.35
							Total Chi-Square	**1.46**	

Above, you can see that Chi-square also identify the Gender split is more significant compare to Class.

Information Gain:

Look at the image below and think which node can be described easily. I am sure, your answer is C because it requires less information as all values are similar. On the other hand, B requires more information to describe it and A requires the maximum information. In other words, we can say that C is a Pure node, B is less Impure and A is more impure.



Now, we can build a conclusion that less impure node requires less information to describe it. And, more impure node requires more information. Information theory is a measure to define this degree of disorganization in a system known as Entropy. If the sample is completely homogeneous, then the entropy is zero and if the sample is an equally divided (50% – 50%), it has entropy of one.

Entropy can be calculated using formula:- $Entropy = -p \log_2 p - q \log_2 q$

Here p and q is probability of success and failure respectively in that node. Entropy is also used with categorical target variable. It chooses the split which has lowest entropy compared to parent node and other splits. The lesser the entropy, the better it is.

Steps to calculate entropy for a split:

- Calculate entropy of parent node
- Calculate entropy of each individual node of split and calculate weighted average of all sub-nodes available in split.

Example: Let's use this method to identify best split for student example.

- Entropy for parent node = -(15/30) log2 (15/30) – (15/30) log2 (15/30) = **1**. Here 1 shows that it is a impure node.
- Entropy for Female node = -(2/10) log2 (2/10) – (8/10) log2 (8/10) = 0.72 and for male node, -(13/20) log2 (13/20) – (7/20) log2 (7/20) = **0.93**
- Entropy for split Gender = Weighted entropy of sub-nodes = (10/30)*0.72 + (20/30)*0.93 = **0.86**
- Entropy for Class IX node, -(6/14) log2 (6/14) – (8/14) log2 (8/14) = 0.99 and for Class X node, -(9/16) log2 (9/16) – (7/16) log2 (7/16) = 0.99.
- Entropy for split Class = (14/30)*0.99 + (16/30)*0.99 = **0.99**

Above, you can see that entropy for *Split on Gender* is the lowest among all, so the tree will split on *Gender*. We can derive information gain from entropy as **1- Entropy.**

Reduction in Variance

Till now, we have discussed the algorithms for categorical target variable. Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:

$$Variance = \frac{\Sigma(X - \overline{X})^2}{n}$$

Above X-bar is mean of the values, X is actual and n is number of values.

Steps to calculate Variance:

- Calculate variance for each node.
- Calculate variance for each split as weighted average of each node variance.

Example: Let's assign numerical value 1 for play cricket and 0 for not playing cricket. Now follow the steps to identify the right split:

- Variance for Root node, here mean value is (15*1 + 15*0)/30 = 0.5 and we have 15 one and 15 zero. Now variance would be ((1-0.5)^2+(1-0.5)^2+... .15 times+(0-0.5)^2+(0-0.5)^2+... 15 times) / 30, this can be written as (15*(1-0.5)^2+15*(0-0.5)^2) / 30 = **0.25**
- Mean of Female node = (2*1+8*0)/10=0.2 and Variance = (2*(1-0.2)^2+8*(0-0.2)^2) / 10 = 0.16
- Mean of Male Node = (13*1+7*0)/20=0.65 and Variance = (13*(1-0.65)^2+7*(0-0.65)^2) / 20 = 0.23
- Variance for Split Gender = Weighted Variance of Sub-nodes = (10/30)*0.16 + (20/30) *0.23 = **0.21**
- Mean of Class IX node = (6*1+8*0)/14=0.43 and Variance = (6*(1-0.43)^2+8*(0-0.43)^2) / 14= 0.24
- Mean of Class X node = (9*1+7*0)/16=0.56 and Variance = (9*(1-0.56)^2+7*(0-0.56)^2) / 16 = 0.25

- Variance for Split Gender = (14/30)*0.24 + (16/30) *0.25 = **0.25**

Above, you can see that Gender split has lower variance compare to parent node, so the split would take place on *Gender* variable.

Until here, we learnt about the basics of decision trees and the decision making process involved to choose the best splits in building a tree model. As I said, decision tree can be applied both on regression and classification problems. Let's understand these aspects in detail.

What are the key parameters of tree modeling and how can we avoid over-fitting in decision trees?

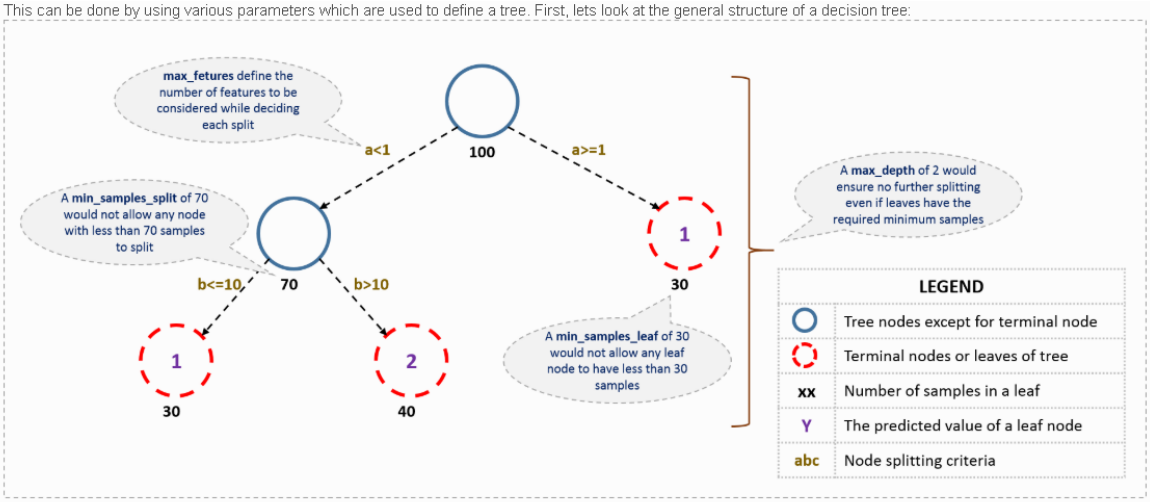
Overfitting is one of the key challenges faced while modeling decision trees. If there is no limit set of a decision tree, it will give you 100% accuracy on training set because in the worse case it will end up making 1 leaf for each observation. Thus, preventing overfitting is pivotal while modeling a decision tree and it can be done in 2 ways:

- Setting constraints on tree size
- Tree pruning

Lets discuss both of these briefly.

Setting Constraints on Tree Size

This can be done by using various parameters which are used to define a tree. First, lets look at the general structure of a decision tree:

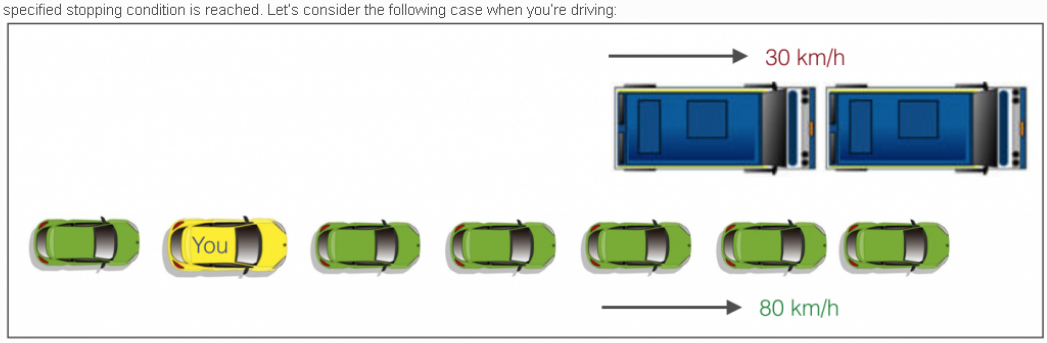


The parameters used for defining a tree are further explained below. The parameters described below are irrespective of tool. It is important to understand the role of parameters used in tree modeling. These parameters are available in R & Python.

- **Minimum samples for a node split**
  - Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
  - Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
  - Too high values can lead to under-fitting hence, it should be tuned using CV.
- **Minimum samples for a terminal node (leaf)**
  - Defines the minimum samples (or observations) required in a terminal node or leaf.
  - Used to control over-fitting similar to min_samples_split.
  - Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.
- **Maximum depth of tree (vertical depth)**
  - The maximum depth of a tree.
  - Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
  - Should be tuned using CV.
- **Maximum number of terminal nodes**
  - The maximum number of terminal nodes or leaves in a tree.
  - Can be defined in place of max_depth. Since binary trees are created, a depth of 'n' would produce a maximum of 2^n leaves.
- **Maximum features to consider for split**
  - The number of features to consider while searching for a best split. These will be randomly selected.
  - As a thumb-rule, square root of the total number of features works great but we should check upto 30-40% of the total number of features.
  - Higher values can lead to over-fitting but depends on case to case.

Tree Pruning

As discussed earlier, the technique of setting constraint is a greedy-approach. In other words, it will check for the best split instantaneously and move forward until one of the specified stopping condition is reached. Let's consider the following case when you're driving:



There are 2 lanes:

- A lane with cars moving at 80km/h
- A lane with trucks moving at 30km/h

At this instant, you are the yellow car and you have 2 choices:

- Take a left and overtake the other 2 cars quickly
- Keep moving in the present lane

Lets analyze these choice. In the former choice, you'll immediately overtake the car ahead and reach behind the truck and start moving at 30 km/h, looking for an opportunity to move back right. All cars originally behind you move ahead in the meanwhile. This would be the optimum choice if your objective is to maximize the distance covered in next say 10 seconds. In the later choice, you sale through at same speed, cross trucks and then overtake maybe depending on situation ahead. Greedy you!
This is exactly the difference between normal decision tree & pruning. A decision tree with constraints won't see the truck ahead and adopt a greedy approach by taking a left. On the other hand if we use pruning, we in effect look at a few steps ahead and make a choice.
So we know pruning is better. But how to implement it in decision tree? The idea is simple.

- We first make the decision tree to a large depth.
- Then we start at the bottom and start removing leaves which are giving us negative returns when compared from the top.
- Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is +10 and keep both leaves.

Are tree based models better than linear models?

"If I can use logistic regression for classification problems and linear regression for regression problems, why is there a need to use trees"? Many of us have this question. And, this is a valid one too.

Actually, you can use any algorithm. It is dependent on the type of problem you are solving. Let's look at some key factors which will help you to decide which algorithm to use:

- If the relationship between dependent & independent variable is well approximated by a linear model, linear regression will outperform tree based model.
- If there is a high non-linearity & complex relationship between dependent & independent variables, a tree model will outperform a classical regression method.
- If you need to build a model which is easy to explain to people, a decision tree model will always do better than a linear model. Decision tree models are even simpler to interpret than linear regression!

Based on an analyticsvidhya.com article

In [ ]: