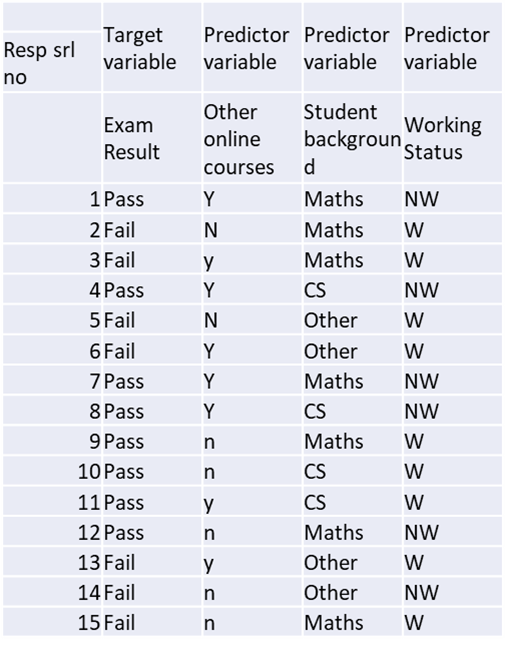
**Information Gain, Entropy and Gini Index**

We have data for 15 data points of student data on pass or fail an online ML exam. To understand the basic process we begin with a dataset which comprises a target variable that is binary (Pass/Fail) and various binary or categorical predictor variables such as:

* whether enrolled in other online courses
* whether student is from a maths, computer science or other background
* Whether working or not working

The dataset is given below:



Toy Dataset for Online ML Exam( source: author)

Notice that only one variable, ‘Student Background’ has more than 2 levels or categories — Maths, CS, Others. It is one for the advantages of Decision Trees compared to other classification models such as Logistic Regression or SVM, that we do not need to carry out one hot encoding to make these into dummy variables.

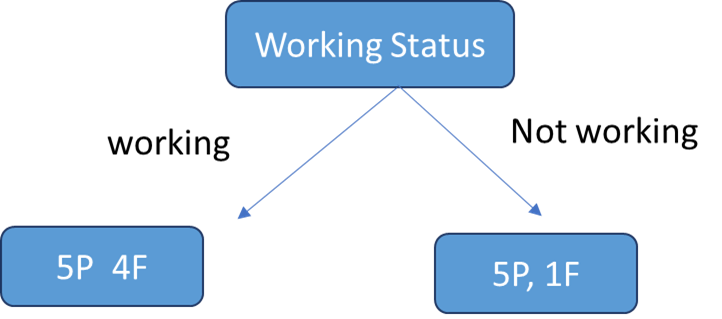
Let us first look at the flow of how a decision tree works and then we will dive into the complexities of how the decisions are actually made…

**Flow of a Decision Tree**

A decision tree begins with the target variable. This is usually called the parent node. The Decision Tree then makes a sequence of splits based in hierarchical order of impact on this target variable. From the analysis perspective the first node is the **root node**, which is the first variable that splits the target variable.

To identify the root node we would evaluate the impact of all the variables that we have currently on the target variable to identify the variable that splits the exam Pass/Fail classes into the most homogenous groups. Our candidates for splitting this are: Background, Working Status and Other Online Courses.

What do we hope to achieve with this split? Suppose we begin with Working Status as the root node. This splits into 2 sub nodes, one each for Working and Not working. Thus the Pass/Fail status is updated in each sub node respectively.



So, this is the basic flow of the Decision Tree. As long as there is a mixture of Pass and Fail in a sub node, there is scope to split further to try and get it to be only one category. This is termed the purity of the node. For example, Not Working has 5 Pass and 1 Fail, hence it is purer than the Working node which has 5P and 4F. A leaf node would be one which contains either Pass or Fail class only.

A node which is impure can be branched further for improving purity. However, most of the time we do not necessarily go down to the point where each leaf is ‘pure’. It is also important to understand that each node is standalone and hence the attribute that best splits the ‘Working’ node may not be the one that best splits the ‘Not Working’ node.

Now, let us move on to learning the core part of Decision Trees - key questions:

**How does the Tree decide which variable to branch out at each level?**

**Greedy top down approach**

Decision trees follow a top-down, greedy approach that is known as recursive binary splitting. The recursive binary splitting approach is top-down because it begins at the top of the tree and then it successively splits the predictor space. At each split the predictor space gets divided into 2 and is shown via two new branches pointing downwards. The algorithm is termed is greedy because at each step of the process, the best split is made for that step. It does not project forwards and try and pick a split that might be more optimal for the overall tree.

The algorithm therefore evaluates all variables on some statistical criteria and then chooses the variable that performs best on the criteria.

**Variable selection criterion**

Here is where the true complexity and sophistication of decision lies. Variables are selected on a complex statistical criterion which is applied at each decision node. Now, variable selection criterion in Decision Trees can be done via two approaches:

1. Entropy and Information Gain

2. Gini Index

Both criteria are broadly similar and seek to determine which variable would split the data to lead to the underlying child nodes being most homogenous or pure. Both are used in different Decision Tree algorithms. To add to the confusion, it is not clear which one is the preferred approach. So, one has to have an understanding of both.

Let us begin with Entropy and Information Gain criterion

**What is Entropy?**

Entropy is a term that comes from physics and means a measure of disorder. More specifically, we can define it as:

***Entropy****is a scientific concept as well as a measurable physical property that is most commonly associated with a state of disorder, randomness, or uncertainty. The term and the concept are used in diverse fields, from*[*classical thermodynamics*](https://en.wikipedia.org/wiki/Classical_thermodynamics)*, where it was first recognized, to the microscopic description of nature in*[*statistical physics*](https://en.wikipedia.org/wiki/Statistical_physics)*, and to the principles of*[*information theory*](https://en.wikipedia.org/wiki/Information_theory)*.*

<https://en.wikipedia.org/wiki/Entropy>

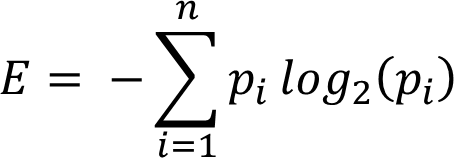
*In*[*information theory*](https://en.wikipedia.org/wiki/Information_theory)*, the****entropy****of a*[*random variable*](https://en.wikipedia.org/wiki/Random_variable)*is the average level of “information”, “surprise”, or “uncertainty” inherent to the variable’s possible outcomes.*

<https://en.wikipedia.org/wiki/Entropy_(information_theory)>

In the context of Decision Trees, entropy is a measure of disorder or impurity in a node. Thus, a node with more variable composition, such as 2Pass and 2 Fail would be considered to have higher Entropy than a node which has only pass or only fail. The maximum level of entropy or disorder is given by 1 and minimum entropy is given by a value 0.

Leaf nodes which have all instances belonging to 1 class would have an entropy of 0. Whereas, the entropy for a node where the classes are divided equally would be 1.

Entropy is measured by the formula:



Where the pi is the probability of randomly selecting an example in class i. Let us understand this a bit better in the context of our example. So, the initial entropy of at the parent node is given by the probability of getting a pass vs fail. In our dataset, the target variable has 9 passes and 6 fails. Hence the probabilities for the entropy formula are:

https://miro.medium.com/v2/resize:fit:875/1*gv29Li0d2BYJbT_J5ZiIXw.png

https://miro.medium.com/v2/resize:fit:875/1*---tWAO2Az30FZv7eUXGKg.png

Now essentially what a Decision Tree does to determine the root node is to calculate the entropy for each variable and its potential splits. For this we have to calculate a potential split from each variable, calculate the average entropy across both or all the nodes and then the change in entropy vis a vis the parent node. This change in entropy is termed Information Gain and represents how much information a feature provides for the target variable.

https://miro.medium.com/v2/resize:fit:875/1*MI9tnNAWiDT6_Q8DqTNZLg.png

Entropy\_parent is the entropy of the parent node and Entropy\_children represents the average entropy of the child nodes that follow this variable. In the current case since we have 3 variables for which this calculation must be done from the perspective of the split.

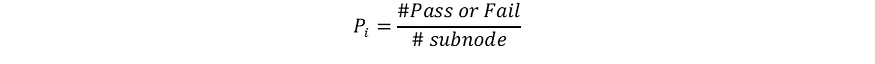
1. Work Status

2. Online Course Status

3. Student Background

To calculate entropy, first let us put our formulas for Entropy and Information Gain in terms of variables in our dataset:

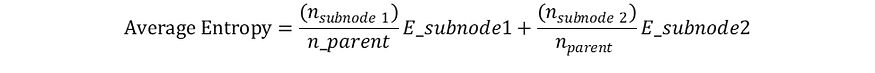
1. Probability of pass and fail at each node, i.e, the Pi:



2. Entropy:

https://miro.medium.com/v2/resize:fit:875/1*VvvrtxxWmgboe7aQ6wB5sw.png

3. Average Entropy at child nodes:



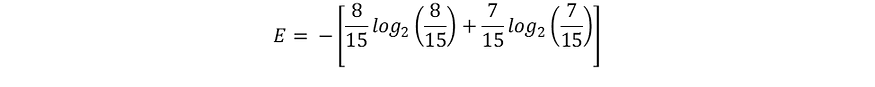
Note, average Entropy is the weighted average of all the sub nodes that a parent node splits into. Thus, in our example this would be 2 sub nodes for Working Status and 3 sub nodes for Student Background.

4. Information Gain:

https://miro.medium.com/v2/resize:fit:875/1*c5mJOERS526cQMAv2sY27A.png

**Parent Node Calculations**

First we will calculate parent node entropy using the formula above. Use any log2 calculator online to calculate the log values. In our case they work out to:



https://miro.medium.com/v2/resize:fit:875/1*OMYxawIWo9xeSJMD7kDkRA.png

(Mathematical note: log to base 2 of anything less than 1 is a negative number, hence we multiply by a minus sign to get a positive number)

So far, this is just the entropy of the parent node. Now we have to decide which attribute or variable to use to split this to get the root node.

**Calculating the Root Node**

For this we have to calculate a potential split from each variable, calculate the average entropy across both the nodes and then the change in entropy via a vis the parent node.

Let us begin with Work Status variable and calculate the entropy of the split.

https://miro.medium.com/v2/resize:fit:875/1*aNLlObfsb38iuSFdsy-mVA.png

https://miro.medium.com/v2/resize:fit:875/1*W-k9Ny9V0YH9f5AHCZoxMA.png

We then calculate the average entropy for the Working status split as a weighted average with weights of the share of observations from the total number of observations that fall in each sub node.

https://miro.medium.com/v2/resize:fit:875/1*o4hTr6H3p70zxrPEOjxC5g.png

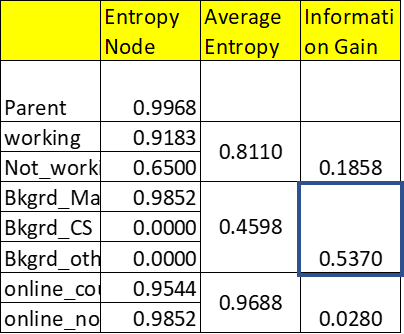
Information Gain = Entropy\_Parent — Entropy\_child =

0.9968–0.8110 = .1858

(Calculations are shown in the spreadsheet below)

In a similar fashion we can evaluate the entropy and information gain for Student Background and Online Courses variables. The results are provided in the table below:

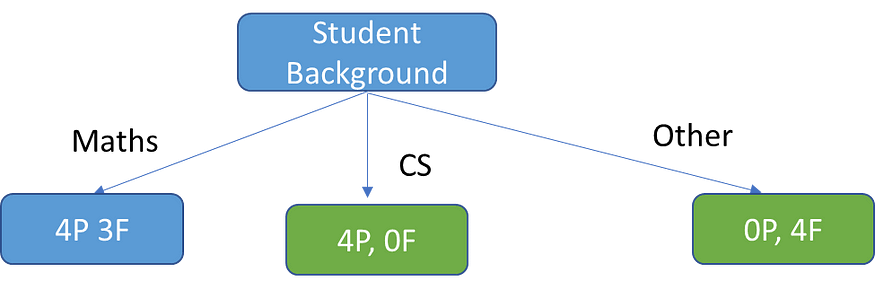
We will drop this variable for the time being and move on to evaluate the other variables. The spreadsheet below shows the Entropy calculations for all the variables:



Root Node Entropy Calculations (source: author)

To find the root node attribute we look at the Information gain from Student Background vis a vis initial parent entropy. This shows the maximum reduction of .5370. Hence, this is the attribute that would be selected as the root node. The other’s variables — ‘Working Status’ and ‘Online Courses’ show a much lower decrease in entropy vis a vis the parental node.

So, on the basis of the above calculations, we have determined what the root node would be. The tree would now look as follows:



Root Node of Decision Tree

Student Background splits the target variable into 3 groups. Everyone from CS background clearly passes and hence this is a terminal or leaf node. Everyone from Other backgrounds fails and this is also a terminal node. Maths background is split into 3 Pass and 4 Fail and hence it is impure and there is some scope for further splitting to attain greater purity.

Now to split the Maths background sub node, we need to calculate Entropy and Information Gain for the remaining variables, i.e., Working Status and Online Courses. We would then select the variable that shows the highest Information Gain.

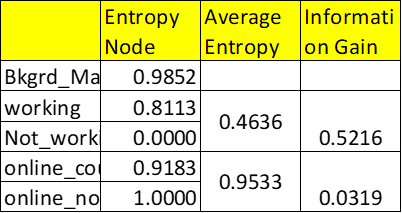
The Entropy and Information Gain Calculations for the Maths Background node can be seen in the table below. Notice, we now have the Maths Background as the node that is being split, hence average Entropy for the splits is calculated using it as a base.

Note: putting in log(0) throws an error. However mathematically we can use the limit. Normally we just don’t include the pj=0 case in the calculation. However, I have included just to show the complete calculation.

*By convention****0 log 0 = 0****, since y log y → 0 as y → 0.*

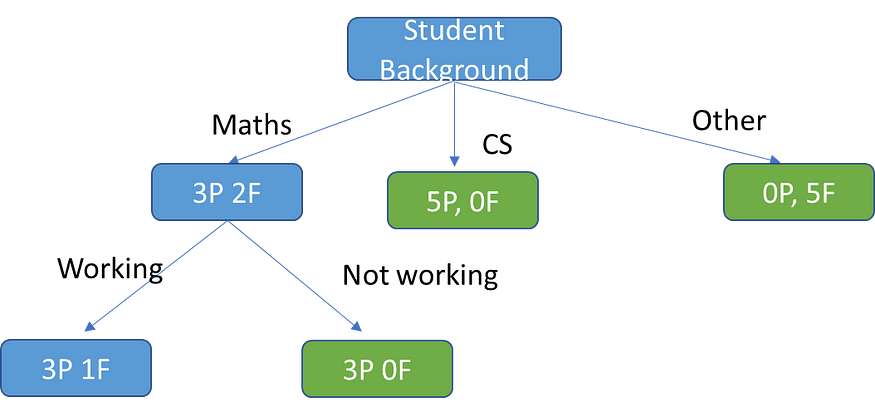
<https://www.icg.isy.liu.se/courses/infotheory/lect1.pdf>

The Entropy for each potential split is:



Splitting the Maths Subnode (Image source: author)

As we can see Information Gain is higher for the Working Status variable. Hence this is the variable used to continue branching.



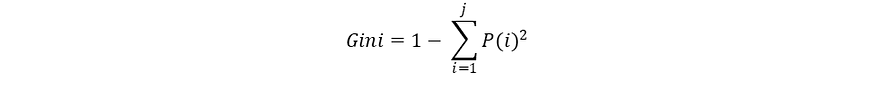
Maths Node Branching

We now see that the Maths node has split into 1 terminal node on the right and one node which is still impure. Notice, now almost all our nodes are terminal nodes. There is only one node which is not terminal. We can try splitting it further using Other Online Courses. Anyway, you get the picture. In any case most Decision Trees do not necessarily split to the point where every node is a terminal node. Most algorithms have built in stops which we will discuss a little further down. Further, if the Decision Tree continues to split we have another problem which is that of overfitting. Again we shall discuss that below after we have briefly reviewed an alternative approach to developing a Decision Tree using the Gini Index.

**Gini Index**

The other way of splitting a decision tree is via the Gini Index. The Entropy and Information Gain method focuses on purity and impurity in a node. The Gini Index or Impurity measures the probability for a random instance being misclassified when chosen randomly. The lower the Gini Index, the better the lower the likelihood of misclassification.

The formula for Gini Index



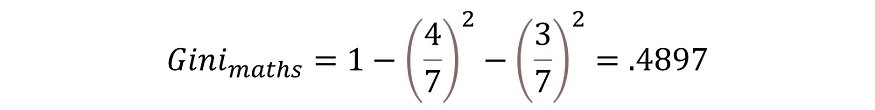
Where j represents the no. of classes in the target variable — Pass and Fail in our example

P(i) represents the ratio of Pass/Total no. of observations in node.

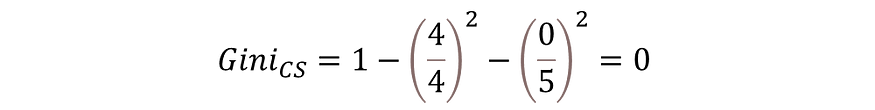
So, Let’s take an example from the decision tree above. Let’s begin with the root node and calculate the Gini Index for each of the splits. The Gini Index has a minimum (highest level of purity) of 0. It has a maximum value of .5. If Gini Index is .5, it indicates a random assignment of classes.

Now let us calculate the Gini index for the root node for Student Background attribute. In this case we have 3 nodes. Gini formula requires us to calculate the Gini Index for each sub node. Then do a weighted average to calculate the overall Gini Index for the node.

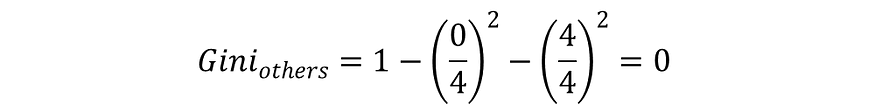
Maths sub node: 4Pass, 3Fail



CS sub node: 4Pass, 0 Fail



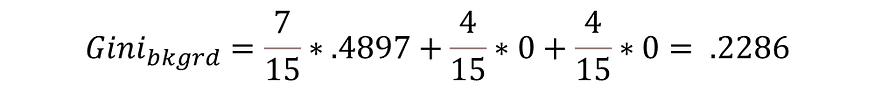
Others sub node: 0Pass, 4 Fail



Note: There are some miscalculations in journal. See comments in the journal if required.

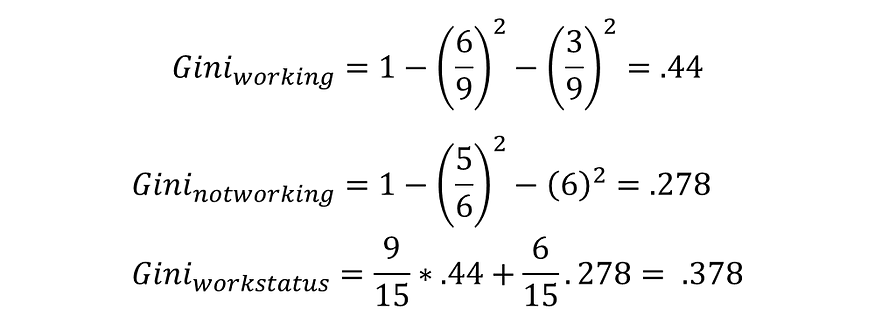
As we can see the probability for misclassification in CS node is zero, since everyone passes. Similarly no scope for misclassification on Others node as everyone fails. Only the maths node has possibility of misclassification, and this is quite high, given that the maximum Gini Index is .5.

The overall Gini Index for this split is calculated similarly to the entropy as weighted average of the distribution across the 3 nodes.

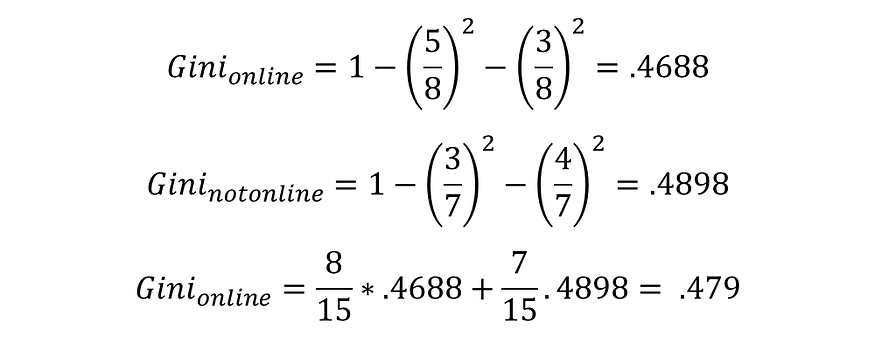


Similarly, we can also compute the Gini Index for Working Status and Online Courses. These are given below:

Working/Not working



Online Courses



The Gini Index is lowest for the Student Background variable. Hence, similar to the Entropy and Information Gain criteria, we pick this variable for the root node. In a similar fashion we would again proceed to move down the tree, carrying out splits where node purity is less.

**Gini Index vs Information Gain**

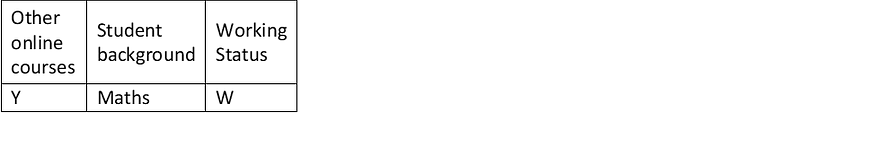
Depending on which impurity measurement is used, tree classification results can vary. This can make small (or sometimes large) impact on your model. There seems to be no one preferred approach by different Decision Tree algorithms. For example, CART uses Gini; ID3 and C4.5 use Entropy.

The Gini index has a maximum impurity is 0.5 and maximum purity is 0, whereas Entropy has a maximum impurity of 1 and maximum purity is 0.

**How does a prediction get made in Decision Trees**

Now that we have understood, hopefully in detail, how Decision Trees carry out splitting and variable selection, we can move on to how they do prediction. Actually, once a tree is trained and tested, prediction is easy. The tree basically provides a flow chart based on various predictor variables. Suppose we have a new instance entering the flow along with its values of different predictor variables. Unlike training and test data , it will not have the class for the target attribute. We are trying to predict this class by moving down the tree, testing its values of different predictor variables at different branches. Ultimately, the new instance will move into a leaf node and will be classified according to the class prevailing in the leaf node.

Suppose it looks like the below configuration:



Based on our tree, we would first check the Math branch, then Working Yes branch. That as we have seen is a leaf node and the new observation would be classified on the basis of the majority vote in this node, i.e., since it is Pass, this new observation would also be predicted to be Pass.

In practice, when the algorithm is evaluating a new example and reaches a leaf node, the prediction is based on the modal value of categories in the leaf node. As seen in the above case, the Working node is not fully pure. However, we go with the prediction of the modal value, which is Pass. In general, most leaf nodes are not pure and, hence for categorical prediction, we use the modal value for prediction. If it is a numerical prediction (regression tree), we predict the mean value of the target values at each leaf node.

**Overfitting and Decision Trees**

Overfitting can be a big challenge with Decision Trees. Even in our toy example, we can see the algorithm continues to split till it reaches a leaf node. Often the leaf node may just have one or two instances. This will clearly lead to a complex tree structure which may not generalize well to a test scenario. This is because each leaf will represent a very specific set of attribute combinations that are seen in the training data, and the tree will not be able to classify attribute combinations not seen in the training data. There are several ways we can prevent the decision tree from becoming too unwieldy: 3 broad approaches to avoiding overfitting are distinguished:

1. Pre pruning or Early stopping: Preventing the tree from growing too big or deep
2. Post Pruning: Allowing a tree to grow to its full depth and then getting rid of various branches based on various criteria

3. Ensembling or using averages of multiple models such as Random Forest

We will only briefly overview at pre and post pruning techniques here. Ensemble techniques such as Random Forest require more explanation and hence will be tackled in a separate article.

**Pruning**

Pre pruning

The pre-pruning technique refers to the early stopping of the growth of the decision tree. The pre-pruning technique involves tuning the hyperparameters of the decision tree model prior to the training pipeline.

The best way is to use the sklearn implementation of the GridSearchCV technique to find the best set of hyperparameters for a Decision Tree model. There are many ways to tackle this problem through hyperparameter tuning.

We can set the maximum depth of our decision tree using the ***max\_depth*** parameter. The more the value of ***max\_depth***, the more complex your tree will be. The training error will off-course decrease if we increase the ***max\_depth*** value but when our test data comes into the picture, we will get a very bad accuracy. Hence you need a value that will not overfit as well as underfit our data and for this, you can use GridSearchCV.

Another way is to set the minimum number of samples for each spilt. It is denoted by ***min\_samples\_split***. Here we specify the minimum number of samples required to do a spilt. For example, we can use a minimum of 10 samples to reach a decision. That means if a node has less than 10 samples then using this parameter, we can stop the further splitting of this node and make it a leaf node.

There are more hyperparameters such as:

* ***min\_samples\_leaf*** – represents the minimum number of samples required to be in the leaf node. The more you increase the number, the more is the possibility of overfitting.
* ***max\_features*** – it helps us decide what number of features to consider when looking for the best split.

A challenge with the early stopping approach is that it faces a ‘horizon’ problem, where an early stopping may prevent some more fruitful splits down the line.

Post Pruning

In this technique, we allow the tree to grow to its maximum depth. Then we remove parts of the tree to prevent overfitting. We effectively consider subtrees of the full tree which are evaluated on a criteria and then removed. Hence we are effectively going ‘up’ the tree and converting leaves to nodes and subtrees. The criteria whether a particular consolidation goes through or not is usually MSE for regression trees and classification error for classification trees.

A challenge with post pruning is that a decision tree can grow very deep and large and hence evaluating every branch can be computationally expensive. An important post pruning technique is **Cost complexity pruning** (ccp) which provides a more efficient solution in this regard. CCP is a complex and advanced technique which is parametrized by the parameter α in Scikit Learn DecisionTreesclassifier module.

So, how does CCP work and what does it do?

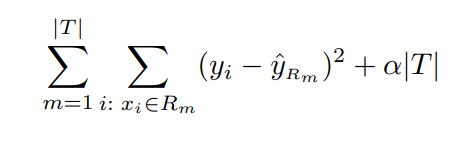
The basic problem that CCP addresses is: How to determine the best way to prune a tree? Intuitively, we would select a sub tree to prune such that its removal leads to a lower test error rate. This can be done using cross validation or, if we have sufficient sample, the validation set approach. However, given the number of sub trees in a fully grown tree for even a small sample, this is likely to be a very computationally and time intensive process.

*Cost complexity pruning — also known as weakest link pruning — gives us a way to do just this. Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α.…….*

*The tuning parameter α controls a trade-off between the subtree’s complexity and its fit to the training data. When α = 0, then the subtree T will simply equal T0, because then (8.4) just measures the training error. However, as α increases, there is a price to pay for having a tree with many terminal nodes, and so the quantity (8.4) will tend to be minimized for a smaller subtree*

An Introduction to Statistical Learning, P 333

Essentially the parameter α is very similar to the penalty term in Lasso regression. The basic equation for CCP is given below



CCP equation, Hastie, p332 (image source: James [et al](https://www.statlearning.com/))

This is one complex equation. But let’s try and understand a bit further. Some definitions:

Rm: Rm is the rectangle (i.e. the subset of predictor space) corresponding to the mth terminal node

yˆRm is the predicted response associated with mth terminal leaf

(yi -yˆRm) — MSE for the sub tree referenced by terminal node m ( we are using regression tree approach for this equation. For simplicity I am following the equation and approach in[James et al(2014)](https://www.statlearning.com/).

|T|: is the no. of terminal nodes in the tree T

Now let’s see what the equation is doing. We are essentially minimizing cost or loss given by (yi -yˆRm) across all terminal nodes. Now α is a term that multiplies the total number terminal nodes in the tree. If α=0, then we are minimizing the training loss. The tree will be the same as the original tree. However, with α>0, we add a penalty term which increases with the number of terminal nodes |T|. This means the overall cost gets minimized for a smaller subtree.

*Minimal cost complexity pruning recursively finds the node with the “weakest link”. The weakest link is characterized by an effective alpha, where the nodes with the smallest effective alpha are pruned first.*

<https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html>

What does this mean? It means that the algorithm is hunting out nodes where the training loss is already high and hence can only be minimized with a small α. On the other hand, nodes where the training loss is smaller, can accommodate a larger penalty term as part of minimization.

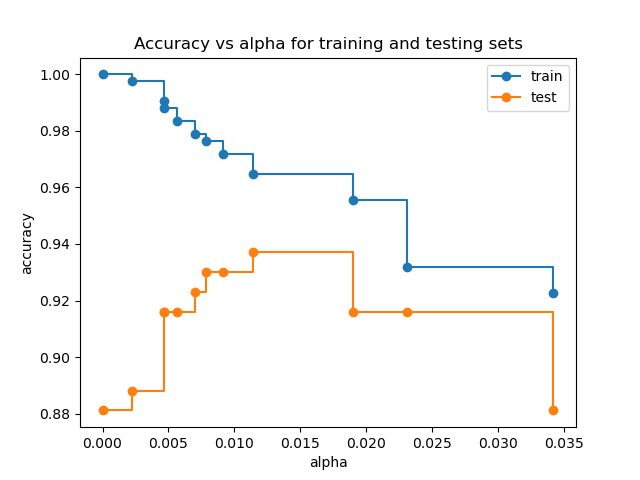
*To get an idea of what values of ccp\_alpha could be appropriate, scikit-learn provides*[***DecisionTreeClassifier.cost\_complexity\_pruning\_path***](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.cost_complexity_pruning_path)*that returns the effective alphas and the corresponding total leaf impurities at each step of the pruning process.*

As α increases, more of the tree is pruned. We then have a tradeoff between bias and variance. With the ccp\_alpha

Effectively we increase the bias of the model, i.e. , we simplify it. However, on the con side this means we have to tolerate increasing levels of impurity in the terminal nodes. We see that as α increases both no. of nodes and tree depth reduces.

**How to determine optimal α**

Plotting ccp\_alpha vs train and test accuracy we see that when α =0 and keeping the other default parameters of [**DecisionTreeClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier), the tree overfits, leading to a 100% training accuracy and 88% testing accuracy. As alpha increases, more of the tree is pruned, thus creating a decision tree that generalizes better. at some point, however, further increases in α actually lead to a decrease in test accuracy as the model becomes too simplified. In this example, setting ccp\_alpha=0.015 maximizes the testing accuracy. (refer to <https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html> for details).



**Advantages and Disadvantages of Trees Decision trees**

1. Trees give a visual schema of the relationship of variables used for classification and hence are more explainable. The hierarchy of the tree provides insight into variable importance.

2. At times they can actually mirror decision making processes.

3. White box model which is explainable and we can track back to each result of the model. This is in contrast to black box models such as neural networks.

4. In general there is less need to prepare and clean data such as normalization and one hot encoding of categorical variables and missing values.

Note the Sklearn implementation currently does not support categorical variables, so we do need to create dummy variables. Similarly it does not support missing values. But both can be handled in theory.

5. Model can be validated statistically

**Disadvantages**

1. Prone to overfitting and hence lower predictive accuracy

2. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem for example can be mitigated by using decision trees within an ensemble

3. Can be non-robust, i.e., a small change in the data can cause a large change in the final estimated tree

4. Predictions are approximate, based on relevant terminal nodes. Hence it it may not be the best method to extrapolate the results of the model to unseen cases.

5. Decision tree learners create biased trees if some classes dominate. It is required to balance the dataset prior to fitting with the decision tree.

Refer to below, if want to see original journal.

## [https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c#:~:text=The%20Gini%20Index%20or%20Impurity,lower%20the%20likelihood%20of%20misclassification.&text=P(i)%20represents%20the%20ratio,of%20observations%20in%20node.](https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c%23:~:text=The%20Gini%20Index%20or%20Impurity,lower%20the%20likelihood%20of%20misclassification.&text=P(i)%20represents%20the%20ratio,of%20observations%20in%20node.)