**The Complete Guide to Decision Trees**

Everything you need to know about a top algorithm in Machine Learning

[Diego Lopez Yse](https://lopezyse.medium.com/?source=post_page-----28a4e3c7be14--------------------------------)

[Apr 17, 2019·17 min read](https://towardsdatascience.com/the-complete-guide-to-decision-trees-28a4e3c7be14?source=post_page-----28a4e3c7be14--------------------------------)

In the beginning, learning Machine Learning (ML) can be intimidating. Terms like “Gradient Descent”, “Latent Dirichlet Allocation” or “Convolutional Layer” can scare lots of people. But there are friendly ways of getting into the discipline, and I think starting with Decision Trees is a wise decision.

Decision Trees (DTs) are probably one of the most useful supervised learning algorithms out there. As opposed to unsupervised learning (where there is no output variable to guide the learning process and data is explored by algorithms to find patterns), in **supervised learning** your existing data is already labelled and you know which behaviour you want to predict in the new data you obtain. This is the type of algorithms that autonomous cars use to recognize pedestrians and objects, or organizations exploit to estimate customers lifetime value and their churn rates.

In a way, supervised learning is like learning with a teacher, and then apply that knowledge to new data.

DTs are ML algorithms that progressively divide data sets into smaller data groups based on a descriptive feature, until they reach sets that are small enough to be described by some **label**. They require that you have data that is labelled (tagged with one or more labels, like the plant name in pictures of plants), so they try to label new data based on that knowledge.

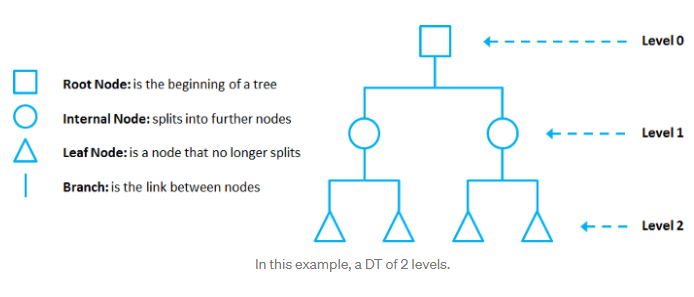
DTs algorithms are perfect to solve **classification** (where machines sort data into classes, like whether an email is spam or not) and **regression** (where machines predict values, like a property price) problems. Regression Trees are used when the dependent variable is continuous or quantitative (e.g. if we want to estimate the probability that a customer will default on a loan), and Classification Trees are used when the dependent variable is categorical or qualitative (e.g. if we want to estimate the blood type of a person).

The importance of DTs relies on the fact that they have lots of applications in the real world. Being one of the mostly used algorithms in ML, they are applied to different functionalities in several industries:

* DTs are being used in the healthcare industry to improve the screening of positive cases in the early detection of **cognitive impairment**, and also to identify the main[risk factors of developing some type of dementia](https://www.news-medical.net/news/20190117/Scientists-design-two-AI-algorithms-to-improve-early-detection-of-cognitive-impairment.aspx) in the future.
* Sophia, [the robot that was made a citizen of Saudi Arabia](https://www.theverge.com/2017/11/10/16617092/sophia-the-robot-citizen-ai-hanson-robotics-ben-goertzel), uses DTs algorithms to chat with humans. In fact, **chatbots** that use these algorithms are already bringing benefits in industries like [health insurance](https://yourstory.com/2018/11/age-alexa-siri-chatbots-next-health-assistants/) by gathering data from customers through the application of innovative surveys and friendly chats. Google recently acquired Onward, a company that uses DTs to develop [chatbots that are exceptionally functional in delivering world-class customer care](https://martechseries.com/predictive-ai/ai-platforms-machine-learning/google-purchases-customer-service-automation-firm-onward/), and [Amazon is investing in the same direction](https://venturebeat.com/2019/01/10/why-25-of-companies-are-copying-amazons-customer-support-model-using-bots-and-ai-vb-live/) to guide customers quickly to a path of resolution.
* It is possible to [predict the most likely causes of forest disturbances](https://www.europeanscientist.com/en/agriculture/new-analysis-reveals-causes-of-global-forest-loss/), like wildfire, logging of tree plantations, large or small scale agriculture, and urbanization by training DTs to recognize different causes of **forest loss** from satellite imagery. DTs and satellite imagery are also used in **agriculture** to [classify different crop types](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4567051/) and identify their phenological stages.
* DTs are great tools to [perform sentiment analysis of texts](https://www.infoq.com/articles/sentiment-analysis-whats-with-the-tone), and identify the emotions behind them. **Sentiment analysis** is a powerful technique that can help organizations to learn about customers choices and their decision drivers.
* In **environmental sciences**, DTs can help to determine the best strategy for [dealing with invasive species](https://phys.org/news/2018-05-ai-canada-invasive-species.html), ranging from eradication to containment, and mitigation of spread.
* DTs are also used to improve **financial fraud detection**. The MIT showed that it could significantly improve the performance of alternative ML models by using DTs that were trained with several sources of raw data to [find patterns of transactions and credit cards that match cases of fraud](http://news.mit.edu/2018/machine-learning-financial-credit-card-fraud-0920).
* DTs are extremely popular for a variety of reasons, being their interpretability probably their most important advantage. They can be trained very fast and are easy to understand, which opens their possibilities to frontiers far beyond scientific walls. Nowadays, DTs are very popular in business environments and their usage is also expanding to civil areas, where some applications are raising big concerns.
* The firm Sesame Credit (a company affiliated with Alibaba) uses [DTs and other algorithms to engine a system of social evaluation](https://bluenotes.anz.com/posts/2016/11/china-credit-and-seeing-unforseen-risk), taking into consideration various factors such as the punctuality with which bills are paid and other online activities. [The benefits of a good “Sesame score” in China range from a higher visibility on dating sites to skipping the waiting line if you need to see a doctor](https://aleteia.org/2018/06/28/the-chinese-social-credit-system-the-real-big-brother-of-the-future/). Actually, after the [chinese government announced it will apply its so-called social credit system to flights and trains](https://www.reuters.com/article/us-china-credit/china-to-bar-people-with-bad-social-credit-from-planes-trains-idUSKCN1GS10S) and stop people who have committed misdeeds from taking such transport for up to a year, there is a concern that the system will end up creating a massive “ML-backed Big Brother”.

# ****The Basics****

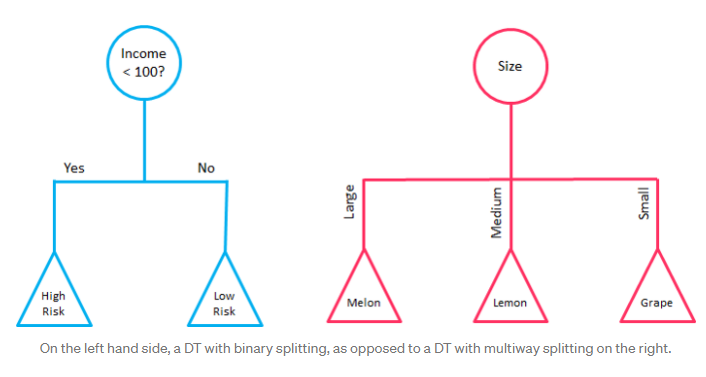
* In the movie Bandersnatch (a stand-alone Black Mirror episode from Netflix), the viewer can interactively choose different narrative paths and reach different story lines and endings. There is a complex set of decisions hidden behind the movie storytelling that lets the audience move in a kind of Choose Your Own Adventure mode, for which Netflix had to work out a way of loading multiple versions of each scene while presenting it in a simple way. In practice, what Netflix producers did was to segment the movie and set different branch points for the viewer to move through, and come up with different results. In other words, this is just like building a DT.
* DTs are composed of nodes, branches and leafs. Each **node** represents an attribute (or feature), each **branch** represents a rule (or decision), and each **leaf** represents an outcome. The **depth** of a Tree is defined by the number of levels, not including the root node.



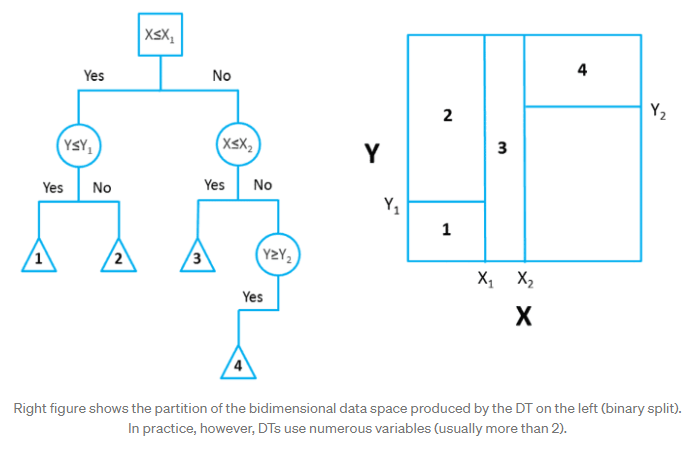
DTs apply a top-down approach to data, so that given a data set, they try to group and label observations that are similar between them, and look for the best rules that split the observations that are dissimilar between them until they reach certain degree of similarity.

*They use a layered splitting process, where at each layer they try to split the data into two or more groups, so that data that fall into the same group are most similar to each other (****homogeneity****), and groups are as different as possible from each other (****heterogeneity****).*

The splitting can be **binary** (which splits each node into *at most* two sub-groups, and tries to find the optimal partitioning), or **multiway**(which splits each node into multiple sub-groups, using as many partitions as existing distinct values). In practice, it is usual to see DTs with binary splits, but it’s important to know that multiway splitting has some advantages. Multiway splits exhaust all information in a nominal attribute, which means that an attribute rarely appears more than once in any path from the root to the leaf, which make DTs easier to comprehend. In fact, it could happen that the best way to split data might be to find a set of intervals for a given feature, and then split that data up into several groups based on those intervals.



In bidimensional terms (using only 2 variables), DTs partition the data universe into a set of rectangles, and fit a model in each one of those rectangles. They are simple yet powerful, and a great tool for data scientists.



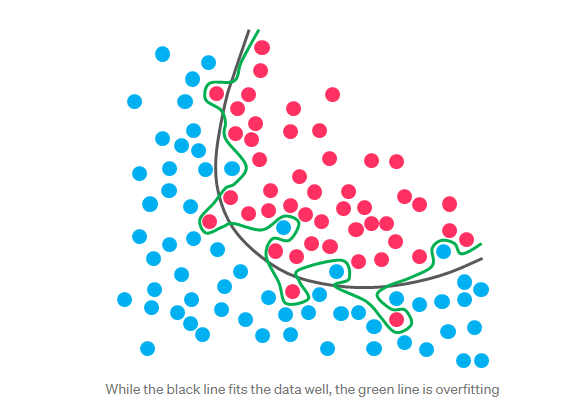
Each node in the DT acts as a test case for some condition, and each branch descending from that node corresponds to one of the possible answers to that test case.

# ****Prune that Tree****

As the number of splits in DTs increase, their complexity rises. In general, simpler DTs are preferred over super complex ones, since they are easier to understand and they are less likely to fall into overfitting.

**Overfitting** refers to a model that learns the **training data** (the data it uses to learn) so well that it has problems to generalize to new (unseen) data.

In other words, the model learns the detail and noise (irrelevant information or randomness in a dataset) in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model.



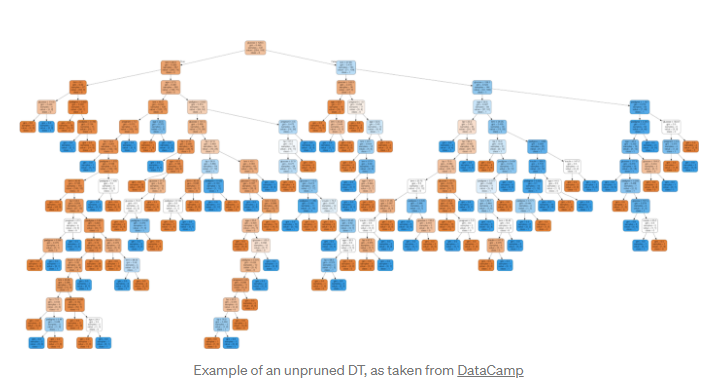
Under this condition, your model works perfectly well with the data you provide upfront, but when you expose that same model to new data, it breaks down. It’s unable to repeat its highly detailed performance.

So, how do you avoid overfitting in DTs? You need to exclude branches that fit data too specifically. You want a DT that can generalize and work well on new data, even though this may imply losing precision on the training data. It’s always better to avoid a DT model that learns and repeats specific details like a parrot, and try to develop one that has the power and flexibility to have a decent performance on new data you provide to it.

***Pruning****is a technique used to deal with overfitting, that reduces the size of DTs by removing sections of the Tree that provide little predictive or classification power.*

The goal of this procedure is to reduce complexity and gain better accuracy by reducing the effects of overfitting and removing sections of the DT that may be based on noisy or erroneous data. There are two different strategies to perform pruning on DTs:

* **Pre-prune:**When you stop growing DT branches when information becomes unreliable.
* **Post-prune:**When you take a fully grown DT and then remove leaf nodes only if it results in a better model performance. This way, you stop removing nodes when no further improvements can be made.



In summary, a big DT that correctly classifies or predicts every example of the training data might not be as good as a smaller one that does not fit all the training data perfectly.

# ****Main DTs algorithms****

Now you may ask yourself: how do DTs know which features to select and how to split the data? To understand that, we need to get into some details.

*All DTs perform basically the same task: they examine all the attributes of the dataset to find the ones that give the best possible result by splitting the data into subgroups. They perform this task recursively by splitting subgroups into smaller and smaller units until the Tree is finished (stopped by certain criteria).*

This decision of making splits heavily affects the Tree’s accuracy and performance, and for that decision, DTs can use **different algorithms** that differ in the possible structure of the Tree (e.g. the number of splits per node), the criteria on how to perform the splits, and when to stop splitting.

So, how can we define which attributes to split, when and how to split them? To answer this question, we must review the main DTs algorithms:

## CHAID

The Chi-squared Automatic Interaction Detection (CHAID) is one of the oldest DT algorithms methods that produces **multiway** DTs (splits can have more than two branches) suitable for classification and regression tasks. When building **Classification Trees**(where the dependent variable is categorical in nature), CHAID relies on the Chi-square independence tests to determine the best split at each step. **Chi-square tests** check if there is a relationship between two variables, and are applied at each stage of the DT to ensure that each branch is significantly associated with a statistically significant predictor of the response variable.

In other words, it chooses the independent variable that has the strongest interaction with the dependent variable.

Additionally, categories of each predictor are merged if they are not significantly different between each other, with respect to the dependent variable. In the case of **Regression Trees** (where the dependent variable is continuous), CHAID relies on **F-tests** (instead of Chi-square tests) to calculate the difference between two population means. If the F-test is significant, a new partition (child node) is created (which means that the partition is statistically different from the parent node). On the other hand, if the result of the F-test between target means is not significant, the categories are merged into a single node.

CHAID does not replace missing values and handles them as a single class which may merge with another class if appropriate. It also produces DTs that tend to be wider rather than deeper (multiway characteristic), which may be unrealistically short and hard to relate to real business conditions. Additionally, it has no pruning function.

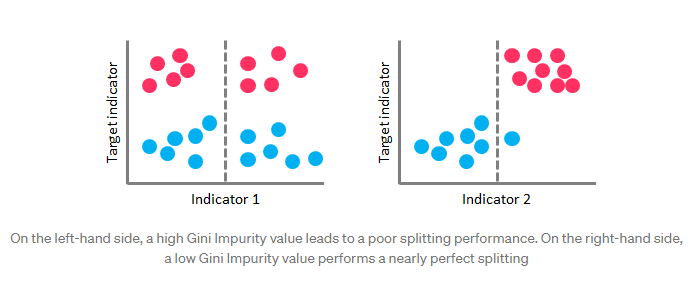
Although not the most powerful (in terms of detecting the smallest possible differences) or fastest DT algorithm out there, CHAID is easy to manage, flexible and can be very useful.

You can find an implementation of CHAID with R in this [link](https://www.r-bloggers.com/chaid-and-r-when-you-need-explanation-may-15-2018/)

## CART

CART is a DT algorithm that produces **binary** Classification or Regression Trees, depending on whether the dependent (or target) variable is categorical or numeric, respectively. It handles data in its raw form (no preprocessing needed), and can use the same variables more than once in different parts of the same DT, which may uncover complex interdependencies between sets of variables.

In the case of **Classification Trees**, CART algorithm uses a metric called Gini Impurityto create decision points for classification tasks. **Gini Impurity** gives an idea of how fine a split is (a measure of a node’s “purity”), by how mixed the classes are in the two groups created by the split. When all observations belong to the same label, there’s a perfect classification and a Gini Impurity value of 0 (minimum value). On the other hand, when all observations are equally distributed among different labels, we face the worst case split result and a Gini Impurity value of 0.5 (maximum value).



In the case of **Regression Trees**, CART algorithm looks for splits that minimize the Least Square Deviation (LSD), choosing the partitions that minimize the result over all possible options. The **LSD**(sometimes referred as “variance reduction”) metric minimizes the sum of the squared distances (or deviations) between the observed values and the predicted values. The difference between the predicted and observed values is called “residual”, which means that LSD chooses the parameter estimates so that the sum of the squared residuals is minimized.

*LSD is well suited for metric data and has the ability to correctly capture more information about the quality of the split than other algorithms.*

The idea behind CART algorithm is to produce a sequence of DTs, each of which is a candidate to be the “optimal Tree”. This optimal Tree is identified by evaluating the performance of every Tree through **testing** (using new data, which the DT has never seen before) or performing **cross-validation** (dividing the dataset into “k” number of folds, and perform testings on each fold).

CART**doesn’t use an internal performance measure**for Tree selection. Instead, DTs performances are always measured through testing or via cross-validation, and the Tree selection proceeds only after this evaluation has been done.

## ID3

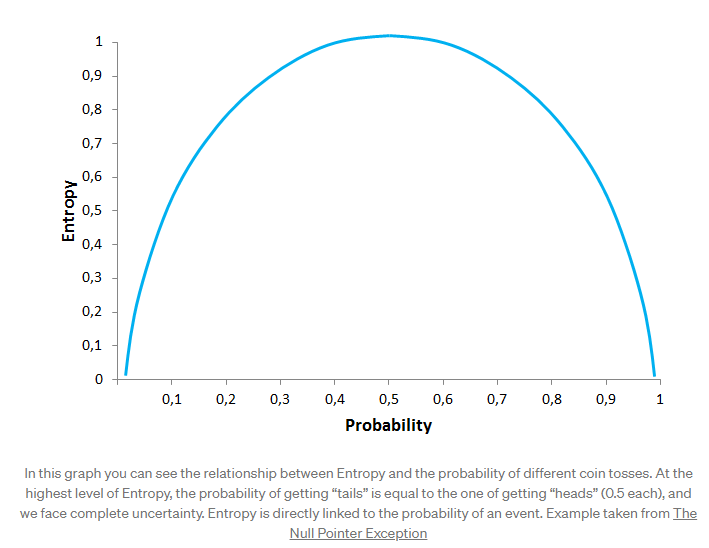
The Iterative Dichotomiser 3 (ID3) is a DT algorithm that is mainly used to produce Classification Trees. Since it hasn’t proved to be so effective building Regression Trees in its raw data, ID3 is mostly used for classification tasks (although some techniques such as building numerical intervals can improve its performance on Regression Trees).

ID3 splits data attributes (dichotomizes) to find the most dominant features, performing this process iteratively to select the DT nodes in a top-down approach.

For the splitting process, ID3 uses the Information Gain metric to select the most useful attributes for classification. **Information Gain**is a concept extracted from Information Theory, that refers to the decrease in the level of randomness in a set of data: basically it measures how much “information” a feature gives us about a class. ID3 will always try to maximize this metric, which means that the attribute with the highest Information Gain will split first.

Information Gain is directly linked to the concept of **Entropy**, which is the measure of the amount of uncertainty or randomness in the data. Entropy values range from 0 (when all members belong to the same class or the sample is completely homogeneous) to 1 (when there is perfect randomness or unpredictability, or the sample is equally divided).

You can think it this way: if you want to make an unbiased coin toss, there is complete randomness or an Entropy value of 1 (“heads” and “tails” are equally like, with a probability of 0.5 each). On the other hand, if you make a coin toss, with for example a coin that has “tails” on both sides, randomness is removed from the event and the Entropy value is 0 (probability of getting “tails” will jump to 1, and probability of “heads” will drop to 0).



This is important because **Information Gain is the decrease in Entropy**, and the attribute that yields the largest Information Gain is chosen for the DT node.

But ID3 has some disadvantages: it can’t handle numeric attributes nor missing values, which can represent serious limitations.

## C4.5

C4.5 is the successor of ID3 and represents an improvement in several aspects. C4.5 can handle both continuous and categorical data, making it suitable to generate Regression and Classification Trees. Additionally, it can deal with missing values by ignoring instances that include non-existing data.

Unlike ID3 (which uses Information Gain as splitting criteria), C4.5 uses Gain Ratio for its splitting process. **Gain Ratio** is a modification of the Information Gain concept that reduces the bias on DTs with huge amount of branches, by taking into account the number and size of the branches when choosing an attribute. Since Information Gain shows an unfair favoritism towards attributes with many outcomes, Gain Ratio corrects this trend by considering the intrinsic information of each split (it basically “normalizes” the Information Gain by using a split information value). This way, **the attribute with the maximum Gain Ratio is selected as the splitting attribute**.

Additionally, C4.5 includes a technique called windowing, which was originally developed to overcome the memory limitations of earlier computers. **Windowing**means that the algorithm randomly selects a subset of the training data (called a “window”) and builds a DT from that selection. This DT is then used to classify the remaining training data, and if it performs a correct classification, the DT is finished. Otherwise, all the misclassified data points are added to the windows, and the cycle repeats until every instance in the training set is correctly classified by the current DT. This technique generally results in DTs that are more accurate than those produced by the standard process due to the use of randomization, since it captures all the “rare” instances together with sufficient “ordinary” cases.

Another capability of C4.5 is that it can **prune** DTs.

C4.5’s pruning method is based on estimating the error rate of every internal node, and replacing it with a leaf node if the estimated error of the leaf is lower. In simple terms, if the algorithm estimates that the DT will be more accurate if the “children” of a node are deleted and that node is made a leaf node, then C4.5 will delete those children.

The latest version of this algorithm is called C5.0, which was released under proprietary license and presents some improvements over C4.5 like:

* **Improved speed:**C5.0 is significantly faster than C4.5 (by several orders of magnitude).
* **Memory usage:** C5.0 is more memory efficient than C4.5.
* **Variable misclassification costs:** in C4.5 all errors are treated as equal, but in practical applications some classification errors are more serious than others. C5.0 allows to define separate cost for each predicted/actual class pair.
* **Smaller decision trees:** C5.0 gets similar results to C4.5 with considerably smaller DTs.
* **Additional data types:** C5.0 can work with dates, times, and allows values to be noted as “not applicable”.
* **Winnowing:** C5.0 can automatically winnow the attributes before a classifier is constructed, discarding those that may be unhelpful or seem irrelevant.

You can find a comparison between C4.5 and C5.0 [here](https://rulequest.com/see5-comparison.html)

# ****The dark side of the Tree****

Surely DTs have lots of advantages. Because of their simplicity and the fact that they are easy to understand and implement, they are widely used for different solutions in a large number of industries. But you also need to be aware of its disadvantages.

DTs tend to **overfit** on their training data, making them perform badly if data previously shown to them doesn’t match to what they are shown later.

They also suffer from **high variance**, which means that a small change in the data can result in a very different set of splits, making interpretation somewhat complex. They suffer from an inherent instability, since due to their hierarchical nature, the effect of an error in the top splits propagate down to all of the splits below.

In Classification Trees, the consequences of **misclassifying observations are more serious in some classes** than others. For example, it is probably worse to predict that a person will not have a heart attack when he/she actually will, than vice versa. This problem is mitigated in algorithms like C5.0, but remains as a serious issue in others.

DTs can also create **biased** Trees if some classes dominate over others. This is a problem in unbalanced datasets (where different classes in the dataset have different number of observations), in which case it is recommended to balance de dataset prior to building the DT.

In the case of Regression Trees, DTs can only predict within the range of values they created based on the data they saw before, which means that they have**boundaries on the values they can produce**.

At each level, DTs look for the best possible split so that they optimize the corresponding splitting criteria.

*But DTs splitting algorithms can’t see far beyond the current level in which they are operating (they are****“greedy”****), which means that they look for a locally optimal and not a globally optimal at each step.*

DTs algorithms grow Trees one node at a time according to some splitting criteria and don’t implement any backtracking technique.

# ****The power of the crowd****

But here are the good news: there are different strategies to overcome these drawbacks. **Ensemble methods** combine several DTs to improve the performance of single DTs, and are a great resource to get over the problems already described. The idea is to train multiple models using the same learning algorithm to achieve superior results.

Probably the 2 most common techniques to perform ensemble DTs are Bagging and Boosting.

**Bagging** (or Bootstrap Aggregation) is used when the goal is to reduce the variance of a DT. **Variance**relates to the fact that DTs can be quite unstable because small variations in the data might result in a completely different Tree being generated. So, the idea of Bagging is to solve this issue by creating **in parallel** random subsets of data (from the training data), where any observation has the **same probability** to appear in a new subset data. Next, each collection of subset data is used to train DTs, resulting in an ensemble of different DTs. Finally, an average of all predictions of those different DTs is used, which produces a more robust performance than single DTs. **Random Forest** is an extension over Bagging, which takes one extra step: in addition to taking the random subset of data, it also takes a random selection of features rather than using all features to grow DTs.

**Boosting** is another technique that creates a collection of predictors to reduce the variance of a DT, but with a different approach. It uses a **sequential method**where it fits consecutive DTS, and at every step, tries to reduce the errors from the prior Tree. With Boosting techniques, each classifier is trained on data, taking into account the previous classifier success. After each training step, the weights are redistributed based on the previous performance. This way, **misclassified data increases its weights**to emphasize the most difficult cases, so that subsequent DTs will focus on them during their training stage and improve their accuracy. Unlike Bagging, in Boosting the observations are weighted and therefore some of them will take part in the new subsets of data more often. As a result of this process, the combination of the whole sets improves the performance of DTs.

Within Boosting, there are several alternatives to determine the weights to use in the training and classification steps (e.g. Gradient Boost, XGBoost, AdaBoost, and others).

You can find a description of similarities and differences between both techniques [here](https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/)