upGrad

Table of Contents

DECISION TREES 2

Introduction and Motivation for Decision Trees 2

Structure of Decision Trees 3

Splitting a Decision Tree 5
Splitting on Numerical Predictors 6
Splitting with Multiple Predictors 7

Best Split – Classification 9
Gini Index 9
Entropy and Information Gain 11

Best Split – Regression 13
Variance as the Measure of Impurity 13

Training of Decision Trees 15

Making Predictions with Trees 16
Predictions in Classification Trees 17
Predictions in Regression Trees 19

Overfitting in Decision Trees 20
Stopping Criteria or Pre-Pruning 20
Post-Pruning 22

Feature Importance 25
Interpreting Feature Importance 25
Limitations of Feature Importance 26

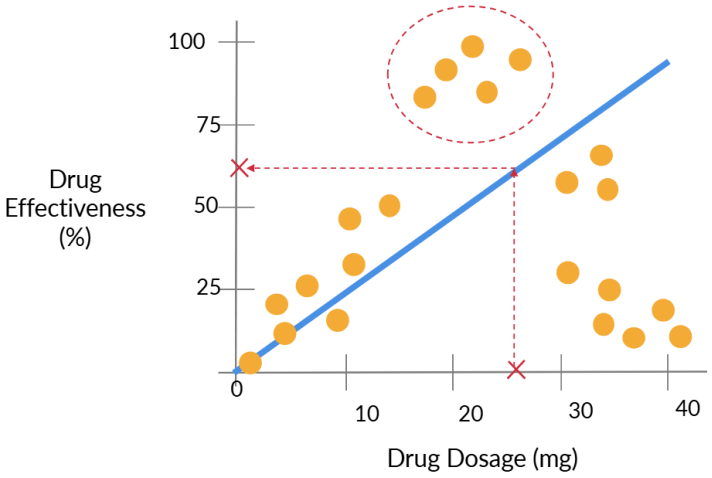
Strengths and Weaknesses of Decision Trees 26

Decision Trees

Introduction and Motivation for Decision Trees

When we studied linear regression, we assumed that the relationship between predictors and the target could be described by a single smooth function. This worked well for problems where the underlying patterns were simple and straight.

However, in many real-world situations, data does not behave in such a cooperative way. Patterns may bend, twist, or change direction entirely depending on the values of different features.



For example, the effect of a drug dosage on patient recovery may increase at first, plateau after a certain level, and then decline if the dosage becomes too high. Such patterns are hard to capture with a single linear equation.

Decision trees provide an alternative approach. Instead of forcing one global equation to fit all data points, they break the prediction problem into a series of smaller, simpler decisions. This process resembles how humans make choices in daily life.

For example, to decide whether to carry an umbrella, we might first check if it is cloudy, then check if the forecast predicts rain, and finally decide based on these checks. Each check reduces uncertainty and narrows the decision.

In a decision tree, these questions appear as split rules in the data. They mimic this reasoning with a sequence of if–else rules. Each rule splits the data into more homogeneous groups, and the process continues until a final decision is reached. The main goal is to maximise the discrimination between the subgroups after every split.

Because of this structure, decision trees can handle complex and irregular relationships that linear models struggle with. They are also interpretable, since the sequence of decisions can be followed step by step. This combination of flexibility and clarity makes decision trees a powerful tool for both classification and regression problems.

Structure of Decision Trees

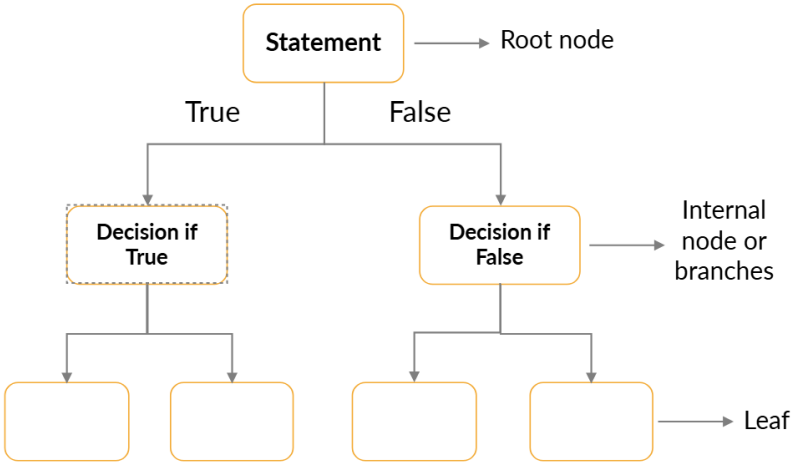
More formally, decision trees are a non-parametric supervised learning method used for classification and regression tasks. Non-parametric means that decision trees do not use parameters or weights to form a general equation of the model, instead they use a rule-based approach to categorise or predict data based on homogeneity.

A decision tree is organised as a hierarchy of nodes connected by branches. The origin of the tree is called the **root node**. This is the topmost node and represents the full dataset before any decisions are made.

From there, the data is split into subgroups using conditions on the features. Each split forms new branches and creates **child nodes**. This recursive process continues until the groups become simple enough to make a prediction.

Child nodes of the tree contain the decision rules, such as "*Age > 50*" or "*Blood Pressure ≤ 120*." Each branch corresponds to the outcome of one of these rules, typically a yes–no or true–false check.

The bottom nodes or the ends of a *branch* of a tree are called **leaf nodes**. These are the endpoints where predictions are produced. In a classification tree, each leaf assigns a class label, while in a regression tree it provides a numerical value.



The tree structure is flexible and adapts itself to the data. A small tree with just a few splits can capture broad patterns, while a deeper tree can describe finer and more detailed distinctions. Due to this flexibility, a trade-off is trees can easily learn too much from the training data and become highly tuned for minute patterns in the training set rather than meaningful trends, leading to poor generalisation on new data.

For example, if a specific data point in the training set has a true label of "*Yes*" when *age* < *60.5* and "*No*" when *age* > *60.5*, the tree may continue splitting during training and include this precise condition as a rule. However, this rule might only be true for that single data point within the training set, rather than representing a generalisable pattern.

Types of decision trees

Decision trees can be used for classification as well as regression tasks:

- **Classification trees**: used when the target is categorical. Leaves of the tree assign class labels such as "*diseased*" or "*healthy*."
- **Regression trees**: used when the target is numerical. Leaves predict values such as "*price of a house*" or "*number of years until an event*."

The basic structure of the tree remains similar in both cases. The difference arises in how splits and classification/predictions happen.

Splitting a Decision Tree

At the start of training, a decision tree has the entire dataset sitting at the root. This is the first step, without any splits – the same label or average would be given to every data point.

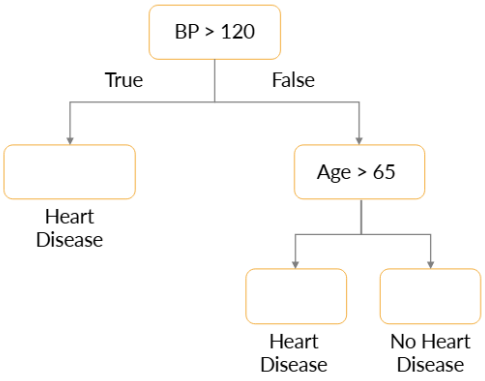
To improve on this, the tree begins by doing a check or test on a feature. For instance, we might check whether blood pressure is greater than 120. This condition divides the data into two groups. Because there is at least some discrimination in the two resulting groups, often these are more consistent internally than the dataset as a whole.

The usefulness of a decision rule lies in how well it separates the outcomes. Some rules lead to groups that are very different from each other, with each group showing a clear trend. Others hardly change the situation, leaving the data still mixed and difficult to predict from. We want a split that provides the strongest separation.

Once the first split is made, we move deeper and split the subgroups again using different rules.

At each node, the tree asks: what is the best feature and the best condition to divide this part of the data? By repeating this process, the tree grows deeper, and its decisions become more refined, until finally the groups at the leaves are simple enough that predictions can be made directly.

Age	BP	Heart Disease
70	130	Yes
67	115	Yes
57	124	Yes
65	120	No
75	120	Yes
56	130	Yes
60	140	Yes



For example, the first split of BP > 120 sends all true data points to the left node, which leads to all the data points belonging to the same class, "*Heart Disease.*" This is a good split.

On the right node, we still have some mixed labels, so we split again, based on Age, which leads to its children nodes finally being pure – the left one having diseased points and the right one having not diseased points.

Splitting on Categorical Features

When the feature is categorical, the splitting process is straightforward. Each split tests whether a data point belongs to a subset of categories.

In case of binary features, we only have two values for the feature – yes and no. For example, sex will have two values: male or female.

In case of categorical predictor with multiple levels, we can split the feature into level *k* vs the set of all other levels.

For example, consider a feature such as “weather” with values “sunny”, “rainy”, and “cloudy.” One possible split is to separate “sunny” from the other two; another is to separate “rainy” from the rest.

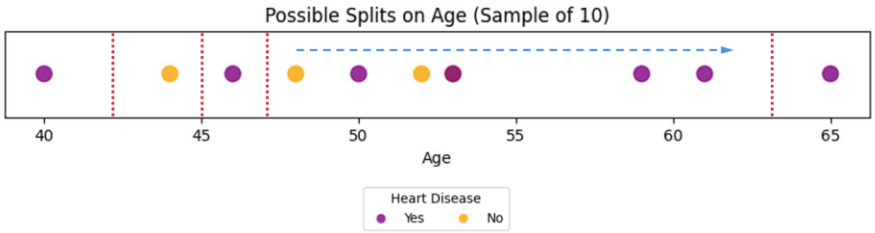
At different nodes, the categories can be regrouped differently, allowing the tree to adjust the splits as needed.

Each such split creates two groups of data. If one of these groups turns out to contain mostly “Yes” outcomes while the other contains mostly “No,” then the split has helped the tree make better predictions.

Splitting on Numerical Predictors

Numerical features are continuous and more spread out. With numerical features, the tree must decide on a threshold. To do this, we first arrange the data points in order and then considers all possible splits in them. Each split occurs at midpoint of two data point. Then the next split occurs at the midpoint of the next set of points.

For example, if we are splitting on the feature “age,” the algorithm might test conditions like “Age > 45” or “Age > 53.” Each threshold divides the data into two groups, one with smaller values and one with larger values.



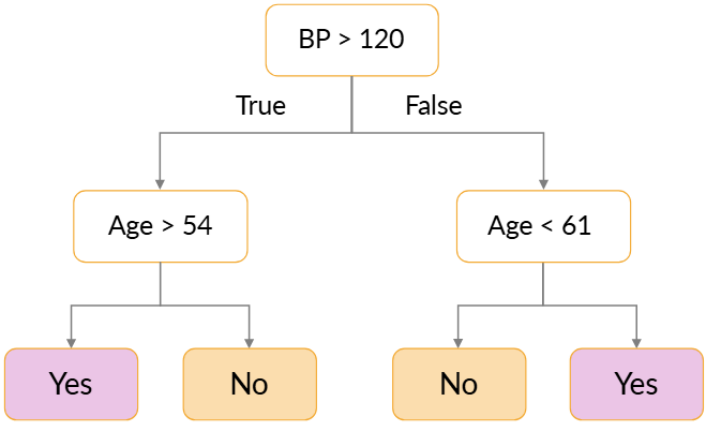
At every possible threshold, the tree checks how the outcomes are distributed in the two resulting groups. The goal is to find the threshold that best separates them. Once chosen, this threshold becomes the decision rule at that node, and the process continues within each subgroup.

Splitting with Multiple Predictors

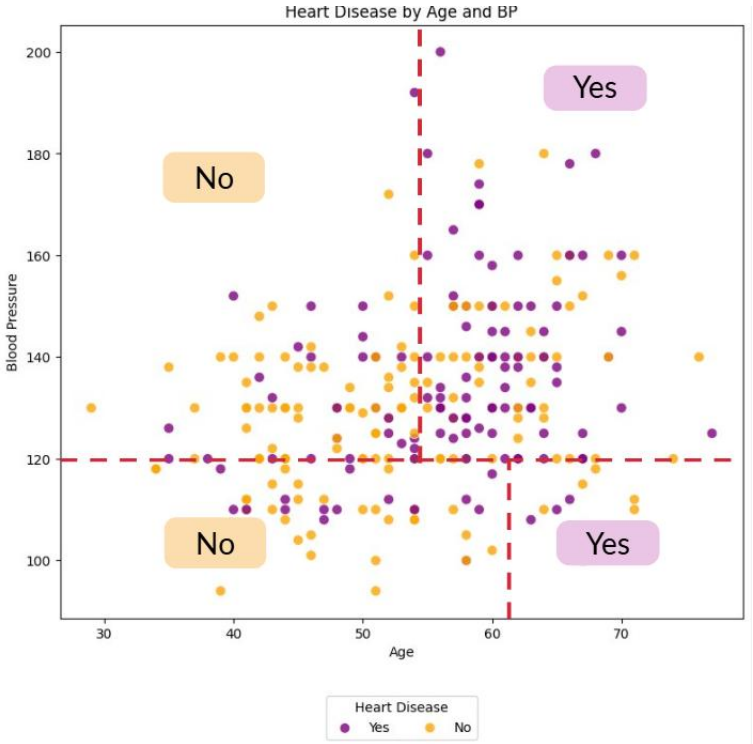
Datasets often contains several features, both categorical and numerical. At each step, the tree does not restrict itself to one feature; instead, it considers all of them. For every feature, it evaluates all possible splits, whether they are category-based or threshold-based, and compares the results. The split that gives the clearest division of outcomes is chosen as the decision rule for that node.

As a result, the first split might be on blood pressure, the second split on age, and further splits on other variables. This flexibility allows the tree to adapt its structure to the data and capture interactions between different predictors.

For example, consider this tree with multiple splits.



Each node splits the data into true and false groups. If we take combine these true and false regions for the whole tree, we get a decision boundary.



At this point, we have seen how decision trees can form splits on categorical and numerical features, and how the process extends when multiple predictors are available. The mechanics are the same whether we are building a classification tree or a regression tree. We divide the data into groups using simple questions about the features.

But we already know, not all splits are equally useful. Some questions may lead to groups that are much clearer in their outcomes, while others may leave the groups just as mixed as before. If we think back to the root node, the challenge is not only to split the data, but to do so in a way that genuinely improves our ability to predict.

This brings us to an important question: **how do we decide which split is the best?** For classification problems, the answer lies in how well the split separates different classes. For regression problems, the answer lies in how much the split reduces variation in numerical values. In both cases, we need a measure to judge the quality of a split and to grow the tree in the right direction.

Best Split – Classification

When we split the data at a node, the goal is not just to divide it, but to make the resulting groups more predictable. Imagine we are trying to predict whether a patient has heart disease. If after a split both groups still contain a 50–50 mix of "Yes" and "No," then the split has not improved our situation much. On the other hand, if one group contains mostly "Yes" and the other mostly "No," then the split is highly informative.

We need a way to measure the "purity" of a group of data points. A group is said to be pure if almost all the points in it belong to the same class. If every point belongs to the same class, the group is completely pure. Conversely, if the classes are evenly mixed, the group is very impure.

The decision tree algorithm evaluates every possible split using such a measure of purity and chooses the split that leads to the greatest improvement. In practice, the two most common measures are the Gini index and entropy. Both capture the same intuition: that the fewer mixed labels there are in a group, the better the split.

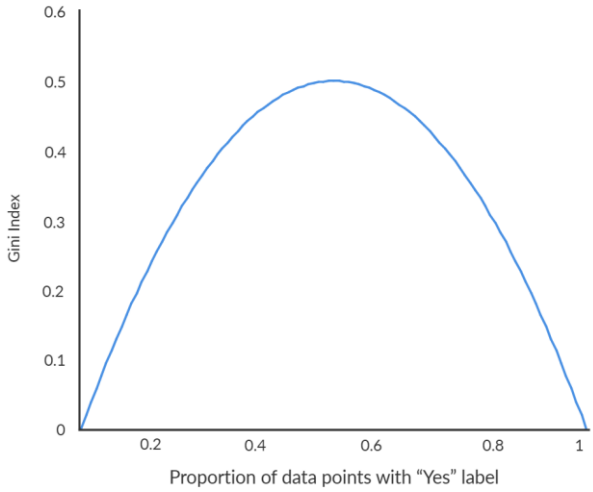
Gini Index

The Gini index measures the probability of misclassifying a randomly chosen point from the group. For a group with two classes, "Yes" and "No," if the proportion of "Yes" points is p, then the proportion of "No" points is $1 - p$. The Gini index of this group is

$$\text{Gini} = 1 - p^2 - (1 - p)^2$$

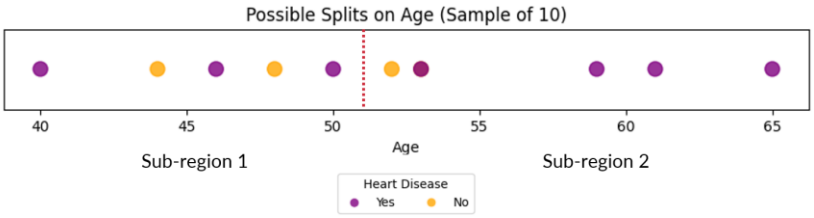
If p is close to 0 or 1, meaning that almost all points belong to the same class, the Gini index is near 0, showing high purity. If p is close to 0.5, the Gini index is higher, showing that the group is more mixed. A Gini index of 0.5 means equal proportions of both classes in the subnode.

The maximum value of Gini may vary based on the number of classes present in the data. For 2 classes, it is 0.5. For a large number of classes, it reaches 1.



Before splitting, we measure the Gini index for the parent node. When evaluating a split, the Gini index is computed for each subgroup and then averaged according to the number of points in each – a weighted average.

Let's try calculating this for the data below.



$$\text{Gini (Parent)} = 1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2 = 0.42$$

$$\text{Gini (Sub} - \text{region 1)} = 1 - \left(\frac{3}{5}\right)^2 - \left(\frac{2}{5}\right)^2 = 0.48$$

$$\text{Gini (Sub} - \text{region 2)} = 1 - \left(\frac{4}{5}\right)^2 - \left(\frac{1}{5}\right)^2 = 0.32$$

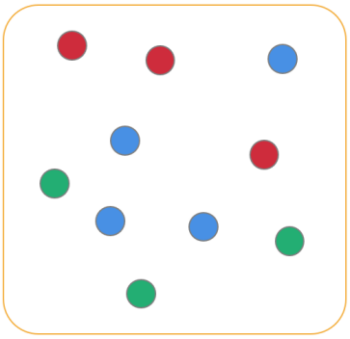
$$\text{Gini (Split)} = \frac{5 \times 0.48 + 5 \times 0.32}{10} = 0.4$$

This weighted average of the children nodes is compared to the parent node's Gini. The split that produces the lowest overall Gini index is chosen, since it leads to the purest outcome. The split above shows a slightly better purity as compared to the parent.

For C classes in general, we denote the Gini index as:

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

where C is the number of classes and P_i is probability of the i^{th} class in the node



The Gini index of this sample of 10 points with three classes will be:

$$\text{Gini} = 1 - \left(\frac{3}{10}\right)^2 - \left(\frac{4}{10}\right)^2 - \left(\frac{3}{10}\right)^2 = 0.66$$

Entropy and Information Gain

Another widely used measure of purity is entropy, borrowed from information theory. Entropy measures the amount of uncertainty in a group of data points. If all points belong to the same class, there is no uncertainty, and the entropy is zero. If the points are evenly divided between the classes, the uncertainty is at its maximum.

For a group with two classes, "Yes" and "No," and proportions p and $1 - p$ respectively, the entropy is defined as:

$$\text{Entropy} = -p \times \log_2 p - (1 - p) \times \log_2 (1 - p)$$

When p is close to 0 or 1, entropy approaches 0, indicating high purity. When $p = 0.5$, entropy reaches its maximum value of 1, showing maximum uncertainty.

Entropy is zero if all points are of the same class (perfect purity). It is maximum when the classes are evenly mixed, since that is when the outcome is most uncertain. A higher entropy signifies more disorder or information in the sample.

For samples with more than two classes, the formula generalises to:

$$\mathrm{E} = -\sum_{i=1}^{C} P_i \times \log_2(P_i)$$

where C is the number of classes and P_i is the proportion of the i^{th} class in the node.

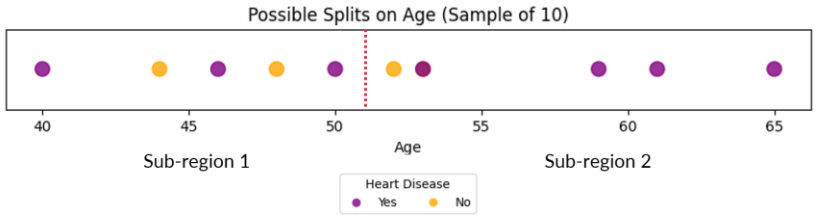
To judge a split, we compare the entropy before and after the split. Before splitting, we calculate the entropy of the parent node. After a split, the entropy is computed for each subgroup, and a weighted average is taken, just as with the Gini index.

The difference between the parent's entropy and this weighted average is called the **information gain**:

$$\text{Information Gain} = \mathrm{E}_{\text{parent}} - \mathrm{E}_{\text{children}}$$

The split that gives the highest information gain is chosen, since it reduces uncertainty the most.

For example, consider a dataset of 10 points with 7 "Yes" and 3 "No":



$$\mathrm{E}_{\text{parent}} = -\left(\left(\frac{3}{10}\right) \times \log_2\left(\frac{3}{10}\right) + \left(\frac{7}{10}\right) \times \log_2\left(\frac{7}{10}\right)\right) = 0.88$$

The split divides the dataset into two subgroups of size 5 each. In one subgroup there are 2 "Yes" and 3 "No," and in the other 5 "Yes" and 0 "No." The entropies are:

$$E_{\text{region 1}} = -\left(\left(\frac{2}{5}\right) \times \log_2\left(\frac{2}{5}\right) + \left(\frac{3}{5}\right) \times \log_2\left(\frac{3}{5}\right)\right) = 0.97$$

$$E_{\text{region 2}} = -\left(\left(\frac{1}{5}\right) \times \log_2\left(\frac{1}{5}\right) + \left(\frac{4}{5}\right) \times \log_2\left(\frac{4}{5}\right)\right) = 0.72$$

The information gain is therefore:

$$\text{Gain} = 0.88 - \frac{5 \times 0.97 + 5 \times 0.72}{10} = 0.035$$

This positive reduction shows that the split has improved the purity of the groups. Among all possible splits, the one with the maximum information gain is chosen.

Best Split – Regression

In regression trees, the target variable is numerical rather than categorical. The aim of a split is therefore not to separate different classes, but to create subgroups in which the target values are as consistent as possible. If the values within a subgroup vary widely, the predictions made from that subgroup will be unreliable. If they are tightly clustered around a central value, the predictions will be much more accurate.

A natural way to measure this consistency is by looking at the variance of the target values. A node with high variance is impure, while a node with low variance is purer. The best split is the one that reduces the overall variance the most.

Variance as the Measure of Impurity

Suppose a node contains target values $y_1, y_2, \ldots, y_n$. The variance of these values is

$$Var = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

where $\bar{y}$ is the mean of the values.

If all values are the same, the variance is zero, showing complete purity. If the values are spread out, the variance is larger, showing impurity.

When a split is considered, we calculate the variance for each subgroup and then take the weighted average according to the number of points in each.

The improvement from the split is the reduction in variance:

$$Variance\ Reduction = Var_{\text{parent}} - Var_{\text{children}}$$

where

$$Var_{\text{children}} = \frac{n_1}{n} Var_1 + \frac{n_2}{n} Var_2$$

with n_1 and n_2 being the number of points in the two subgroups. The split that gives the largest variance reduction is chosen.

For example, we have the target "years to heart risk" for 10 patients, with values:

$$\{11.5, 8, 8.5, 10, 10, 12, 3, 9, 7, 6\}$$

The mean of all values is 8.9, and the variance of the parent node is:

$$Var_{\text{parent}} = \frac{1}{10} \sum_{i=1}^{10} (y_i - 8.9)^2 \approx 4.0$$

Now consider a split based on age that divides the data into two groups:

- Region 1: {11.5, 8, 8.5, 10, 10, 12}
  - mean = 10.0
  - variance ≈ 2.1

- Region 2: {3, 9, 7, 6}
  - mean = 6.25
  - variance ≈ 0.9

The weighted variance after the split is:

$$Var_{\text{children}} = \frac{6}{10} \times 2.1 + \frac{4}{10} \times 0.9 = 1.62$$

Thus, the variance reduction is:

$$\text{Variance Reduction} = 4.0 - 1.62 = 2.38$$

Since the variance is significantly reduced, this split is considered effective.

Training of Decision Trees

For training a decision tree, we have a few different algorithms. A common one is **Classification and Regression Trees (CART)** algorithm, which builds decision trees through a process of recursive partitioning.

The algorithm repeatedly divides the dataset into smaller subsets, with each division chosen to maximise the improvement in prediction quality.

The process begins with the entire dataset at the root. At this point, the data is usually mixed, and predictions would be crude. The algorithm then considers all possible splits – across different features and for all data points in a single feature – and searches for the best possible split.

For each feature, all candidate thresholds (in the case of numerical variables) or category groupings (in the case of categorical variables) are evaluated. Each candidate split is scored using an impurity measure: the Gini index or entropy for classification, and variance reduction for regression.

The split that produces the largest improvement in purity is selected, and the data is divided accordingly. This produces two child nodes. The same procedure is then applied recursively within each child node, gradually growing the tree deeper.

This search is **greedy**: at each step, the algorithm chooses the best split available at that moment, without considering how the decision might affect later splits. While this does not guarantee the globally best tree, it is efficient and works well in practice.

The process continues until a stopping condition is met.

We can keep splitting until each leaf contains only a single data point. However, this would overfit the training data and lead to poor generalisation. Instead, common stopping rules are applied, such as:

- All points in a node belong to the same class or have zero variance
- The maximum allowed depth of the tree has been reached
- A node contains fewer than a minimum number of samples required for splitting
- The reduction in impurity from a potential split is smaller than a given threshold

These rules prevent the tree from growing unnecessarily deep and help balance the trade-off between model accuracy and simplicity.

The outcome of this training process is a partition of the feature space into disjoint regions; each associated with a prediction. Within each region, new data points are treated as similar and are assigned the same outcome.

Making Predictions with Trees

Once a tree has been trained, making predictions for new data points is simple. The point simply travels down the tree according to the rules at each node until it reaches a leaf. The outcome associated with that leaf becomes the prediction. Here, the way outcomes are assigned differs between classification and regression trees.

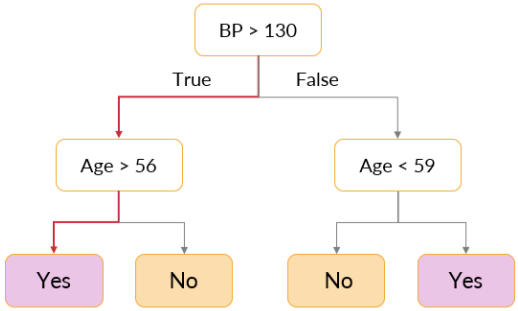
Whether in classification or regression, predictions can also be seen as defining **regions of the feature space**. Every path down the tree corresponds to a sequence of conditions on the features, and each leaf represents a region where the prediction is constant.

For classification, the region corresponds to a class label or probability distribution; for regression, it corresponds to a central numerical value.

This stepwise nature makes decision trees powerful and easy to interpret. Due to this, trees may sometimes produce sharp decision boundaries and also easily overfit if grown too deeply.

Predictions in Classification Trees

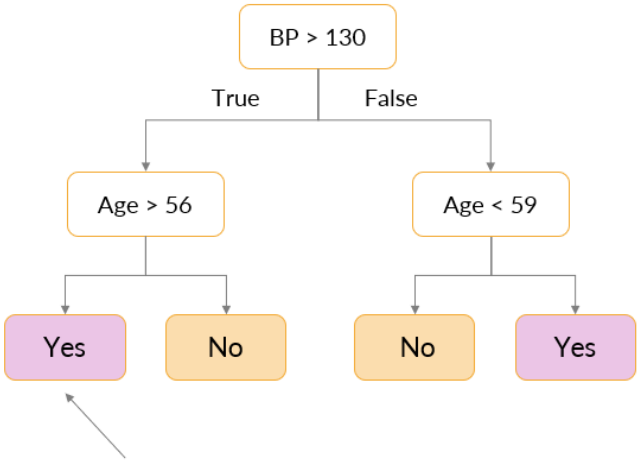
At a classification leaf, the algorithm assigns a class label based on the distribution of training examples that ended up in that leaf.



A 58-year-old person with a BP of 135 is predicted to have risk of heart disease

The simplest rule is **majority voting**: whichever class occurs most frequently in the leaf becomes the predicted label for all new points that land there.

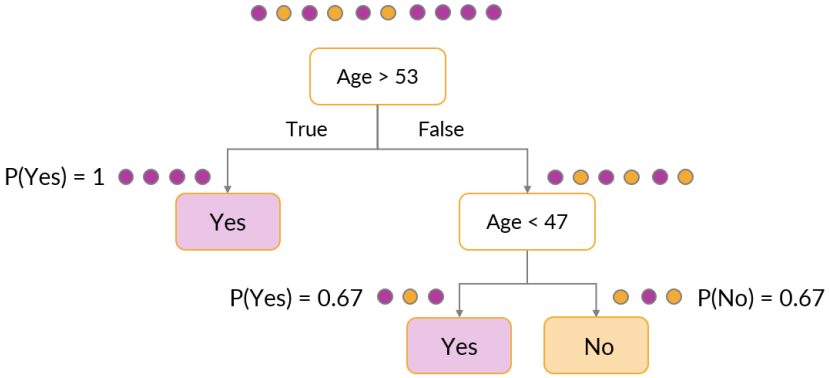
For example, if a leaf contains 12 training points of which 9 are "Yes" and 3 are "No," the predicted class for the leaf is "Yes."



All the data points in this leaf node need not be of the same class. The **majority** is "Yes", so the prediction for any new data point that falls here is "Yes".

A more refined approach is to use **class probabilities**. In the same example, we can assign probabilities $\mathrm{P(Yes)} = 7/10 = 0.7$ and $\mathrm{P(No)} = 3/10 = 0.3$.

The following decision tree shows the splits with the number of samples in each subnode.



In the above tree, the probability of class “Yes” is 0.67 in the left leaf node and 0.33 in the right one. We can assign “Yes” to the left leaf and “No” to the right one.

Predictions are then made by applying a **threshold** to these probabilities. A new data point in this leaf is then assigned these probability estimates as the output and labelled to the majority class. It provides richer information, as probabilities reflect uncertainty better than hard labels.

In binary classification, the default threshold is 0.5: if the probability of “Yes” data points in the node exceeds 0.5, we predict “Yes”; otherwise, we predict “No.” This is equivalent to majority voting.

For multiple classes, we label the leaf node to the class with the highest probability.

However, the threshold can also be adjusted to match the needs of the application. If missing a positive case is more costly than creating a false alarm, the threshold can be lowered.

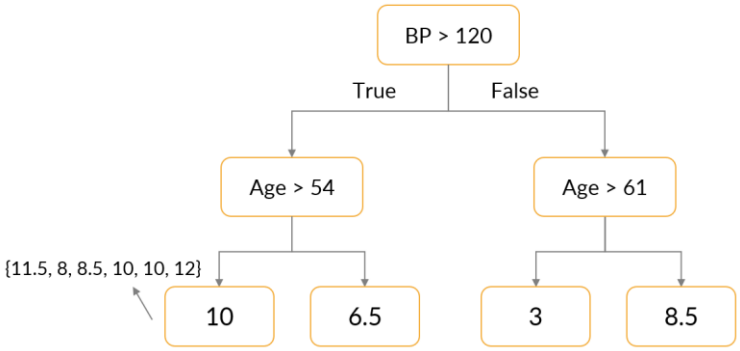
We can keep splitting until we reach a node where the probability of any class in the node exceeds the threshold. Having a large sample of data points to begin with is also beneficial.

In practice, many tree implementations output both the majority class and the probability distribution across classes.

Predictions in Regression Trees

In regression, the target is numerical, so each leaf must provide a single value as the prediction for any new data point that reaches it. The natural way to summarise the outcomes in a leaf is by using a **central value** that represents the group.

The most common choice is the **mean** of the target values in the leaf. Mean is chosen as it minimises the sum of squared errors within the group. Since variance reduction is the criterion used to grow the tree, predicting with the mean is consistent with the training objective.



A 58-year-old person with a BP of 135 is predicted to have heart risk in 10 years.

In this example, we choose the mean of the sample for the leftmost leaf, which comes out to be 10. This value minimises the squared distance between the prediction and the actual outcomes.

In some cases, however, the mean may not be the best representative. If the target values are skewed or contain outliers, the **median** may provide a better summary. The median minimises the sum of absolute errors, making it more robust to extreme values.

For example, in a leaf with values {5, 6, 7, 100}, the mean is 29.5, which is misleading; the median, 6.5, is a much more reasonable prediction.

So, while regression trees typically default to the mean, other measures of central tendency can be used depending on the nature of the data and the chosen loss function.

Overfitting in Decision Trees

An advantage of decision trees is their flexibility. They can capture complex patterns in the training set by repeatedly splitting. However, this same flexibility makes them prone to **overfitting**.

A tree can keep splitting until every training point is perfectly classified or predicted. In classification, this means some leaves may contain only a single data point. In regression, it means each leaf may simply output the exact value of the one point it contains. However, these single data points might be just quirks of the training set, instead of generalisable trends.

This tendency is particularly strong in decision trees because the **splitting process is greedy**. Even a small reduction in impurity may be enough to justify another split, and these small gains accumulate until the tree is very deep.

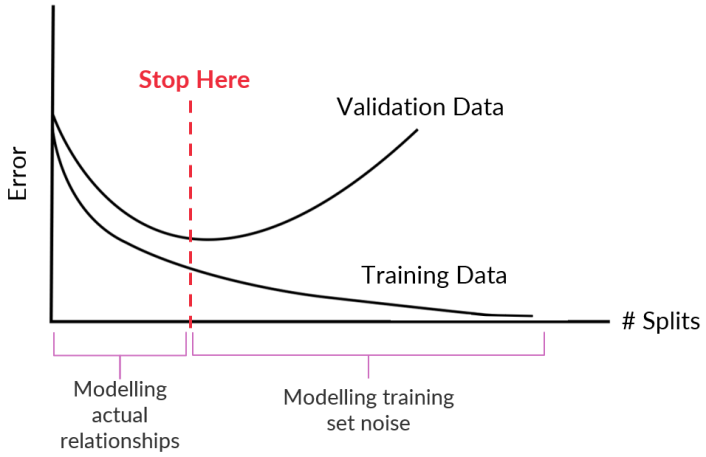
Another factor is the **piecewise-constant nature of predictions**. Since each leaf gives the same prediction to all points in its region, the only way for the tree to adapt to fine differences is by creating more and more splits.

This makes overfitting almost inevitable unless the growth of the tree is controlled. For this reason, pruning strategies are essential to keep trees interpretable, efficient, and generalisable.

Stopping Criteria or Pre-Pruning

To stop overfitting, we need mechanisms to stop the growth earlier, before overfitting sets in. This strategy is called **pre-pruning** or **early stopping** or **truncation**. Instead of allowing the tree to grow to its full size, we impose rules that limit its depth or the number of points in its leaves.

Pre-pruning is guided by the principle of **diminishing returns**. Each new split makes the nodes slightly purer, but the improvement eventually becomes marginal while the risk of overfitting increases. By stopping growth at the right time, we balance complexity with generalisation.



Common Pre-Pruning Rules

- **Maximum depth** restricts how many levels the tree can grow. A shallow tree may underfit, but a very deep tree almost always overfits.
- **Minimum samples for a split** require that a node must contain at least a certain number of data points before it can be split. This prevents the tree from creating tiny branches for a handful of cases.
- **Minimum samples per leaf** ensures that each leaf has a reasonable number of data points. This makes the final classifications for the leaves stable.
- **Minimum impurity decrease** specifies that a split is only made if it reduces impurity by more than a given threshold. This prevents the algorithm from chasing tiny improvements.

Each of these rules acts as a safeguard against the tendency of trees to keep splitting until noise is captured.

Another way to stop tree growth in classification problems is using the class probability threshold. If one class already dominates strongly enough, for example, 90% samples in a leaf, further splitting is unlikely to provide meaningful benefit. In such cases, the node is declared a leaf, and all new data points that fall there are assigned to the dominant class.

The effect of pre-pruning is to produce smaller, more compact trees. Such trees are easier to interpret and faster to compute with, and less likely to overfit.

However, aggressive pre-pruning can also cause underfitting. If growth is stopped too early, the tree may miss important structure in the data. The strength of pre-pruning rules is often tuned using validation data or cross-validation, to find the balance between bias and variance.

Post-Pruning

While pre-pruning limits the growth of a tree in advance, **post-pruning** takes the opposite approach. The tree is first grown to be as large as possible, often until every leaf is pure, and then simplified afterwards. The goal is to strike the right balance between accuracy and complexity, similar to how we used regularisation in linear and logistic regression.

In decision trees, pruning plays the same role by penalising and cutting unnecessary branches. A simpler model may fit the training data less perfectly but will usually generalise better to unseen data.

The Cost-Complexity Measure

The formal way to balance accuracy with simplicity in trees is through the **cost-complexity function**:

$$R_\alpha(T) = R(T) + \alpha \times |T|$$

where,

- $R(T)$ is the error of the tree on training data
- $|T|$ is the number of leaves
- $\alpha \geq 0$ is the complexity penalty

When $\alpha = 0$, the best tree is always the largest one, since complexity is not penalised. As α increases, simpler trees become preferable, even if their training error is higher.

Here, α acts exactly like a regularisation hyperparameter. A small value allows a more complex model, while a large value limits the complexity of the tree. It is represented as ***ccp_alpha*** in libraries like *sk-learn*, to be used with cross-validation or tuning.

In regularisation for regression, we use L1 and L2 methods to discourage large coefficients and retain feature importance for preventing overfitting.

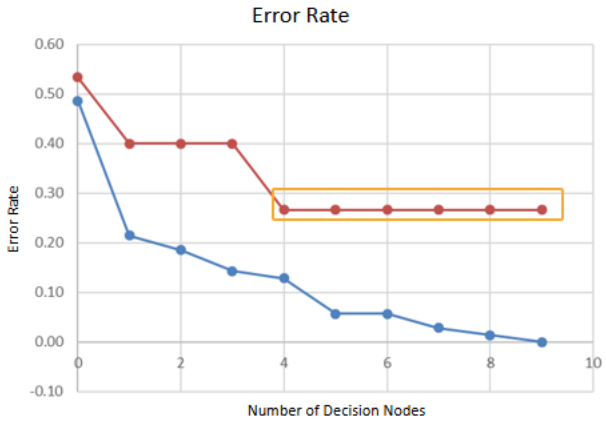
For decision trees, the situation is similar. The error term ensures that the model fits the training data. The complexity penalty $\alpha \times |T|$ discourages unnecessary leaves.

Cost-Complexity Pruning (CCP)

Cost-complexity pruning is performed in three stages:

1. Grow the full tree until every node is pure or contains too few samples to split
2. Generate a pruning sequence to iteratively remove branches that cause the smallest increase in the cost function. This creates a sequence of trees, from the largest down to the smallest
3. Select the optimal tree with cross-validation to evaluate the sequence and choose the tree that minimises error on unseen data.

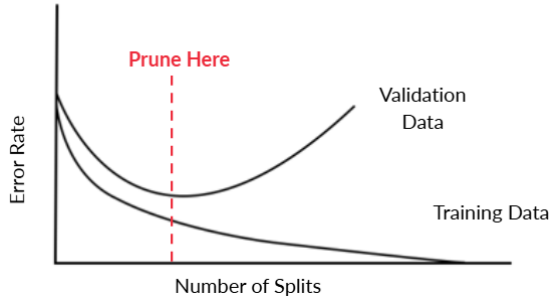
A helpful way to understand this process is to plot **error against tree complexity**. On the x-axis we put tree complexity, measured by the number of leaves; on the y-axis we put the error rate, usually computed on a validation set.



The blue curve shows training error, which always decreases as the tree grows larger. A very deep tree can reach zero training error, but this often means overfitting.

The red curve shows validation error, which typically decreases at first, reaches a minimum, and then rises again as the tree overfits.

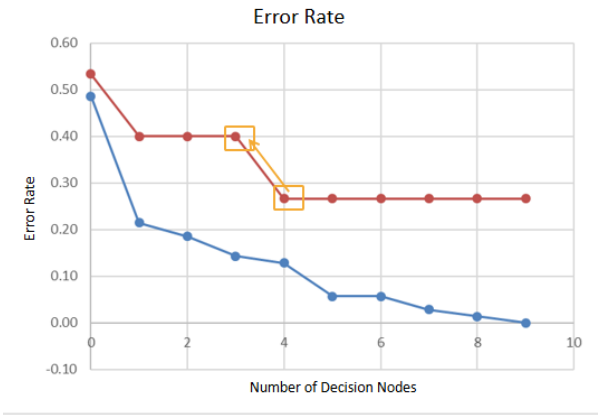
Recall the test-train error plot as complexity increases:



The tree corresponding to the lowest point on the validation error curve is called the **Minimum Error Tree**. If more than one tree achieves the same minimum error, we choose the smallest among them. This ensures interpretability without sacrificing accuracy.

However, in practice, we may not always want the absolute minimum. A tree that is only slightly larger than necessary may not generalise well, especially if the validation curve is flat near the bottom.

To handle this, we use the one-standard-error rule. This rule selects the **Best Pruned Tree**, defined as the smallest tree whose validation error is within one standard error of the minimum.



This graph is the tree-based equivalent of the bias–variance trade-off. It visually shows how backtracking from the full tree helps recover the balance between underfitting and overfitting.

Feature Importance

Decision trees not only make predictions but also provide insights into which features are most influential in the model.

Each time a feature is used to split the data, it reduces impurity in the resulting groups. The larger the reduction, the more useful that split is. To compute feature importance, we track the total impurity reduction brought by each feature across all the splits in the tree. The more a feature contributes to making the groups purer, the higher its importance score.

For a feature X_j, its importance is calculated as:

$$\text{Importance}(X_j) = \sum_{\text{splits on } X_j} \Delta \text{ Impurity}$$

where Δ Impurity is the decrease in Gini index, entropy, or variance produced by that split.

After summing these values over the whole tree, the importances are normalised so that they add up to 1 across all features.

Interpreting Feature Importance

A feature with high importance is one that consistently improves predictions by producing useful splits. Such features are the main drivers of the tree's decisions.

Features with low importance, on the other hand, either provide little discriminatory power or are overshadowed by stronger predictors.

For example, in a tree predicting heart disease, "blood pressure" and "cholesterol" may account for most of the impurity reduction, while "age" contributes less. In that case, blood pressure and cholesterol would have higher importance scores.

Feature importance is influenced by pruning. When a tree is pruned, weaker branches are cut off, and features that contributed less may disappear from the model altogether. Strong features, whose splits provide meaningful reductions in impurity, remain important.

This links to regularisation: pruning simplifies the tree and at the same time sharpens the picture of which features are actually informative.

Limitations of Feature Importance

- **Bias toward features with many levels:** A categorical variable with many categories has more opportunities to create splits and may appear artificially more important
- **Instability:** Decision trees are sensitive to small changes in data, so feature importance scores may vary from one dataset sample to another
- **Relative measure:** Importances are relative to the set of features in the model. If a strong predictor is removed, the importance of others may rise artificially

Strengths and Weaknesses of Decision Trees

Decision trees are among the most popular machine learning models because they combine flexibility with interpretability. But like any method, they have both advantages and disadvantages.

Strengths:

- Easy to understand and interpret; the rules can be followed step by step
- Naturally handle both numerical and categorical features
- No need for feature scaling or standardisation
- Capture non-linear relationships and interactions between features
- Robust to outliers, since splits depend only on orderings and groupings
- Can be visualised, making them useful for communication and explanation

Weaknesses:

- Highly prone to overfitting if not pruned
- Small changes in the data can lead to a very different tree (instability)
- Predictions are piecewise constant, which may be too crude for certain tasks
- Biased feature importance, especially toward variables with many categories
- Often less accurate than more advanced methods when used alone

Despite their limitations, decision trees play a central role in modern machine learning because they serve as the building blocks for powerful ensemble methods. By combining many trees, ensemble techniques like **random forests** and **boosting** overcome instability and overfitting, while achieving excellent predictive performance.

Thus, even though single decision trees are rarely the final choice in practice, understanding them is essential.