Machine Learning (Semester 1 2024)

Tree-Based Classifiers

Nina Hernitschek
Centro de Astronomía CITEVA
Universidad de Antofagasta

July 9, 2024

Motivation

So far, we have seen a general overview about **machine learning**, along with how **Support Vector Machines** work.

Today we will learn about **Tree-Based Classifiers**, another important type of classifier.

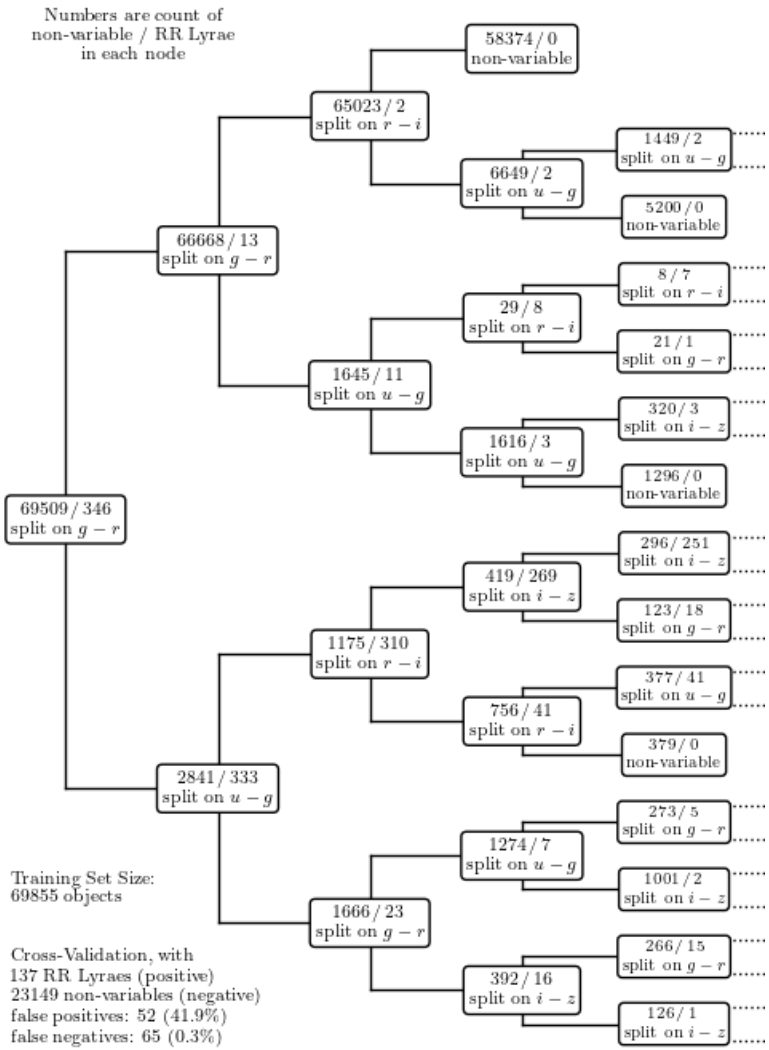
Motivation

A decision tree is a hierarchical application of decision boundaries:

- the top node contains the entire data set
- define some criteria to split the sample into 2 groups (not necessarily equal)
- splitting repeats, recursively, until a predefined stopping criteria is reached

Motivation

example:



The terminal nodes (leaf nodes) record the fraction of points that have one classification or the other in the training set.

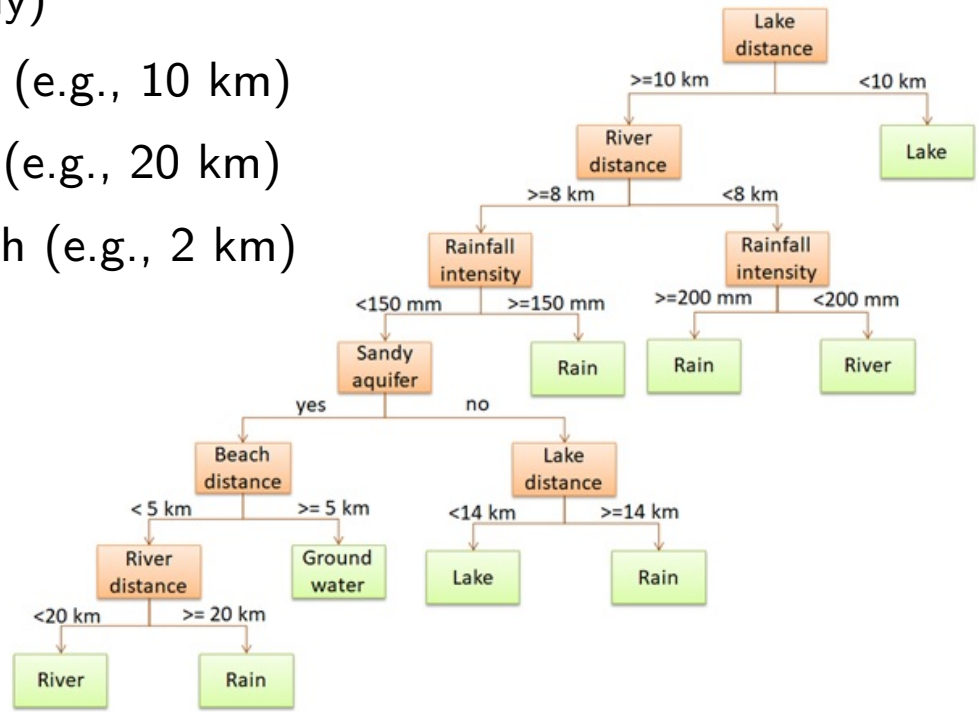
The fraction of points from the training set classified as one belonging to one class or the other (in the leaf node) defines the class associated with that leaf node.

The binary splitting makes this extremely efficient. The trick is to ask the right questions.

Decision Trees

Let's consider a simple **decision tree** created to predict the most suitable water resource for a location. The features include:

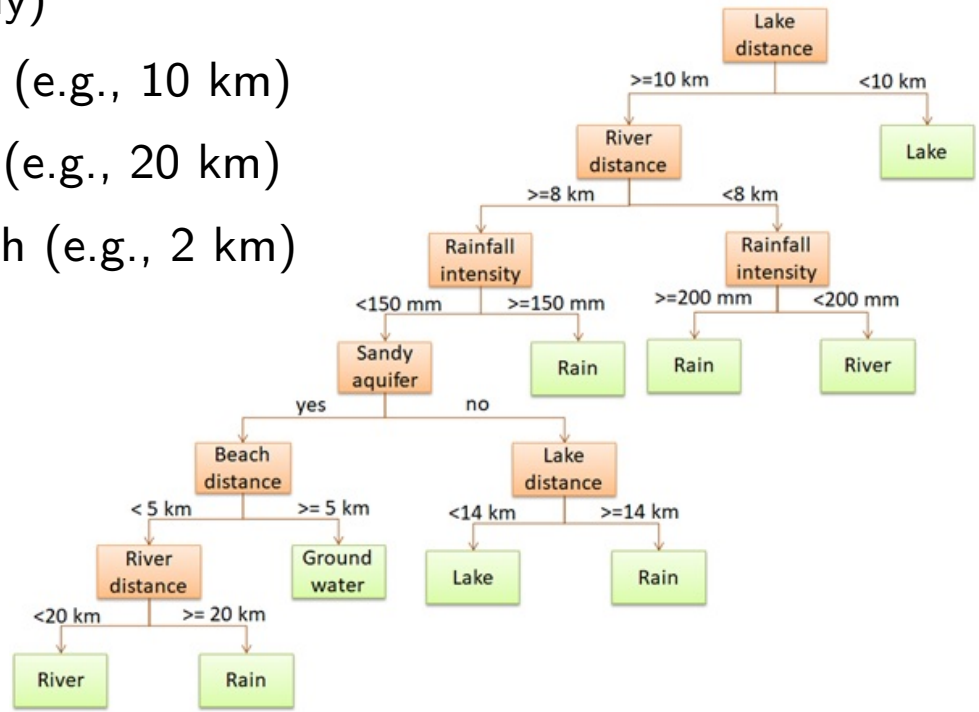
- Rainfall (e.g., 120 mm/month)
- Aquifer type (e.g., sandy)
- Distance from the river (e.g., 10 km)
- Distance from the lake (e.g., 20 km)
- Distance from the beach (e.g., 2 km)



Decision Trees

Let's consider a simple **decision tree** created to predict the most suitable water resource for a location. The features include:

- Rainfall (e.g., 120 mm/month)
- Aquifer type (e.g., sandy)
- Distance from the river (e.g., 10 km)
- Distance from the lake (e.g., 20 km)
- Distance from the beach (e.g., 2 km)



This decision tree could be constructed manually, or by a tree-based supervised learning algorithm.

Tree-Based Models

Tree-based models are supervised learning algorithms that use one (or multiple) decision tree(s) to represent how different input variables can be used to predict a target value.

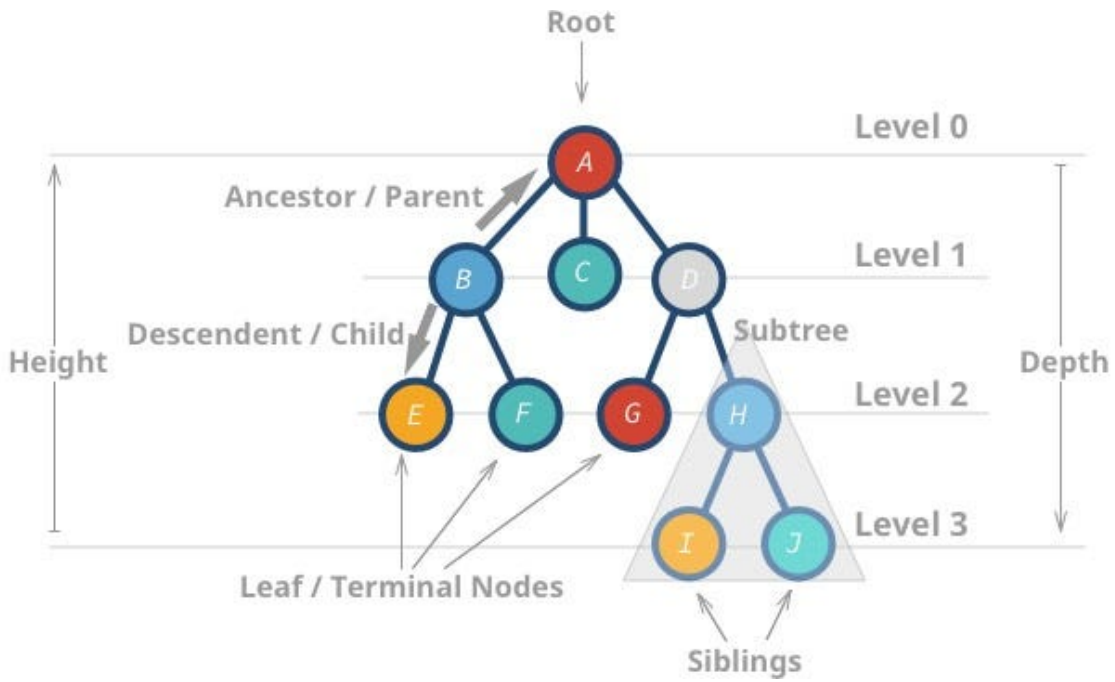
Machine learning uses tree-based models for **both classification and regression problems**, such as the type of an astronomical object or its temperature.

Why are Tree-Based Models Important?

Tree-based models are a **popular approach** in machine learning because of a number of benefits:

- Decision trees are easy to understand and interpret, and outcomes can be easily explained.
- The way they are constructed naturally gives a **feature ranking**, i.e. how important which features are.
- They accommodate both **categorical and numerical data** and can be used for both **classification and regression** problems. Computationally, they perform well even for large data sets and require less data preparation than other techniques.

Terminology



Root Node: It represents the entire training set.

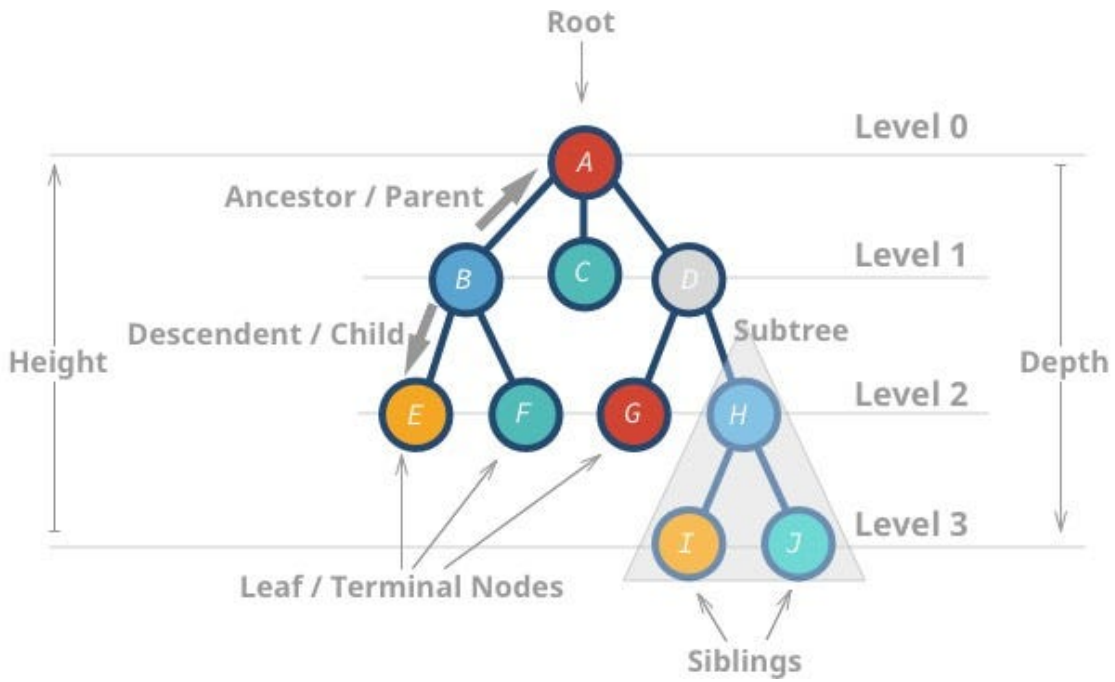
Splitting: It is a process of dividing a node into two or more sub-nodes.

Decision Node: When a sub-node splits into further sub-nodes, then it is called decision node.

Leaf/ Terminal Node: Nodes do not split are called Leaf or Terminal node.

Branch / Sub-Tree: A sub section of entire tree is called branch or sub-tree.

Terminology



Parent and Child Node: A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

Depth of a tree: The length of the longest path from a root to a leaf.

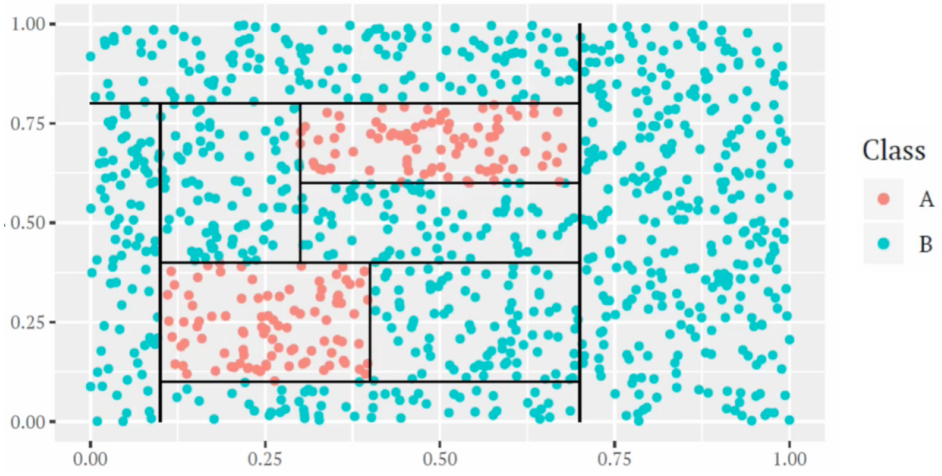
Size of a tree: The number of terminal nodes in the tree.

Learning in Tree-Based Models

During **learning**, decision trees are constructed by partitioning the feature space into a number of smaller (non-overlapping) regions with similar response values using a set of splitting rules:

The input variables are repeatedly segmented into subsets to build the decision tree, and each branch is tested for prediction accuracy and evaluated for efficiency and effectiveness.
Generating a successful decision tree results in the most important variables at the top of the tree hierarchy.

For **example**, the following figures show a tree-based classification model built on two predictors.



Learning in Tree-Based Models

We've seen:

Decision tree identifies the most significant variable and it's value that gives best homogeneous sets.

How does it identify the variables and the split points?

Learning in Tree-Based Models

We've seen:

Decision tree identifies the most significant variable and it's value that gives best homogeneous sets.

How does it identify the variables and the split points?

To do this, decision tree uses various algorithms, which we will discuss in the following.

The CART Algorithm

There are many methodologies for constructing tree-based models, but the most well-known and classic is the **classification and regression tree (CART) algorithm**.

A classification and regression tree partitions the training data into homogeneous subgroups (i.e., groups with similar response values) and then fits a simple constant in each subgroup.
Mathematically, a classification and regression tree ends in a leaf node which corresponds to some hyper-rectangle in feature (predictor) space, R_j, $j = 1, ..., J$, i.e. the feature space defined by $X_1, X_2, ..., X_p$ is divided into J distinct and non-overlapping regions.

The CART Algorithm

There are many methodologies for constructing tree-based models, but the most well-known and classic is the **classification and regression tree (CART) algorithm**.

A classification and regression tree partitions the training data into homogeneous subgroups (i.e., groups with similar response values) and then fits a simple constant in each subgroup.
Mathematically, a classification and regression tree ends in a leaf node which corresponds to some hyper-rectangle in feature (predictor) space, R_j, $j = 1, ..., J$, i.e. the feature space defined by $X_1, X_2, ..., X_p$ is divided into J distinct and non-overlapping regions.

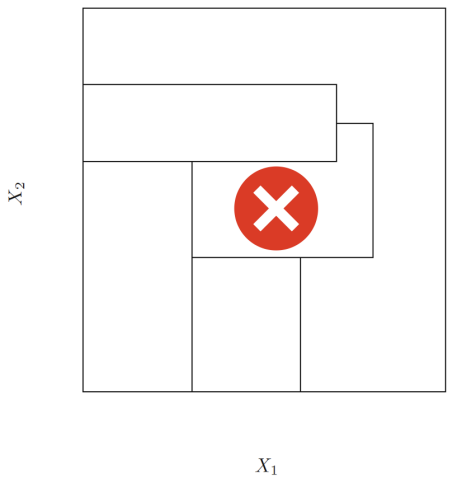
As a **regression tree**, it assigns a value to each R_j, that is the model predicts the output based on the average response values for all observations that fall in that subgroup.

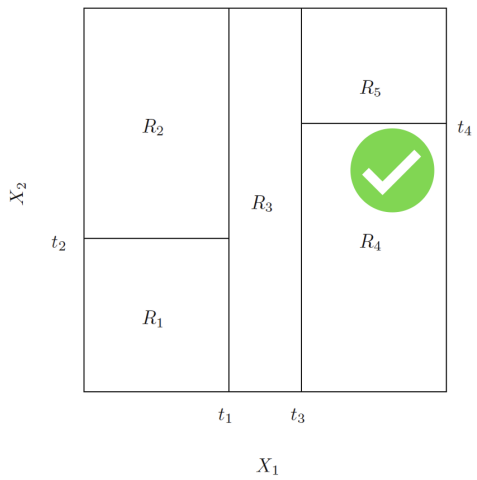
As a **classification tree**, it assigns a class label to each R_j, that is the model predicts the output based on the class that has majority representation or provides predicted probabilities using the proportion of each class within the subgroup.

The CART Algorithm

In theory, the regions R_j could have any shape. But we choose hyper (high-dimensional)-rectangles (boxes) for simplicity.

The left panel is an example of an invalid partitioning, and the right is an example of a valid one.





Performance Metric

For **regression trees**, the goal is to find R_j, $j = 1, ..., J$ minimizing the RSS

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

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth rectangle.

For **classification trees**, we may adopt classification error rate $E = 1 - \max_k(\hat{p}_{jk})$ as performance criteria, where $\hat{p}_{jk}$ represents the proportion of training observations in the jth region that are from the kth class. However, it turns out that the classification error rate is not sufficiently sensitive for tree-growing, especially when the data is noisy.

In practice, two other metrics are preferable:

Gini index: $G = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$

Entropy: $D = -\sum_{k=1}^{K} \hat{p}_{jk} \log \hat{p}_{jk}$

Note that both functions will take on a value near zero if the $\hat{p}_{jk}$ are all near zero or near one.

Greedy fitting algorithm

It is computationally infeasible to consider every possible partition of the feature space into J boxes. Think about how many ways one could divide this 2-D space into 5 boxes. Not to mention high-dimensional feature space. Therefore, in practice, we adopt the so-called **Greedy (top-down) fitting algorithm**:

1. Initialize hyper-rectangle $R = \mathbb{R}^d$
2. Find the best (based on some objective function) split on variable x_i at locations s, gives $R_1(i, s) = \{x \in R : x_i < s\}$ and $R_2(i, s) = \{x \in R : x_i \geq s\}$
3. Repeat step 2 twice, once with $R = R_1(is, s)$ and once with $R = R_2(i, s)$.

There will be some stopping rule, for example, requiring a minimum number of training observations in each hyper-rectangle (box).

Note that such a greedy fitting algorithm does not guarantee an optimal tree since it is constructed stepwisely.

Pros and Cons of CART

The **advantages** of CART are:

- Simple and easy to interpret
- Akin to common decision-making processes
- Good visualisations
- Easy to handle qualitative predictors (avoid dummy variables)

Pros and Cons of CART

The **advantages** of CART are:

- Simple and easy to interpret
- Akin to common decision-making processes
- Good visualisations
- Easy to handle qualitative predictors (avoid dummy variables)

The **disadvantages** of CART are:

- Trees tend to have high variance, small changes in the sample lead to dramatic cascading changes in the fit
- Poor prediction accuracy

Regression Trees vs. Classification Trees

differences between Regression and Classification Trees:

The dependent variable is continuous.

The value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, its prediction is made with mean value.

The dependent variable is categorical.

The value (class) obtained by terminal node in the training data is the most common of observations falling in that region. Thus, if an unseen data observation falls in that region, its prediction is made with the most common value.

Regression Trees vs. Classification Trees

similarities between Regression and Classification Trees:

- Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions. For the sake of simplicity, you can think of these regions as high dimensional boxes.
- Both the trees follow a top-down greedy approach known as recursive binary splitting. We call it as *top-down* because it begins from the top of tree when all the observations are available in a single region and successively splits the predictor space into two new branches down the tree. It is called *greedy* because the algorithm focuses on only the current split, looking for the best available variable, and not considering future splits that could result in a better tree.

Split Criteria

We are still left with the question: How does a tree based algorithms decide where to split?

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

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. Decision tree splits the nodes on all available variables and then select the split which results in most homogeneous sub-nodes.

In the following, we will see commonly used algorithms for split criteria in decision trees.

Gini Impurity

Gini impurity is a measure of the lack of homogeneity in a dataset. It specifically calculates the probability of misclassifying an instance chosen uniformly at random. The splitting process involves evaluating potential splits based on Gini impurity for each feature. The split that minimizes the weighted sum of impurities in the resulting subsets is selected, aiming to create nodes with predominantly homogeneous class distributions.

If we have C total classes and $p(i)$ is the probability of picking a datapoint with class i, then the **Gini impurity** is calculated as

$$G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$$

Gini Impurity

Gini impurity is a measure of the lack of homogeneity in a dataset. It specifically calculates the probability of misclassifying an instance chosen uniformly at random. The splitting process involves evaluating potential splits based on Gini impurity for each feature. The split that minimizes the weighted sum of impurities in the resulting subsets is selected, aiming to create nodes with predominantly homogeneous class distributions.

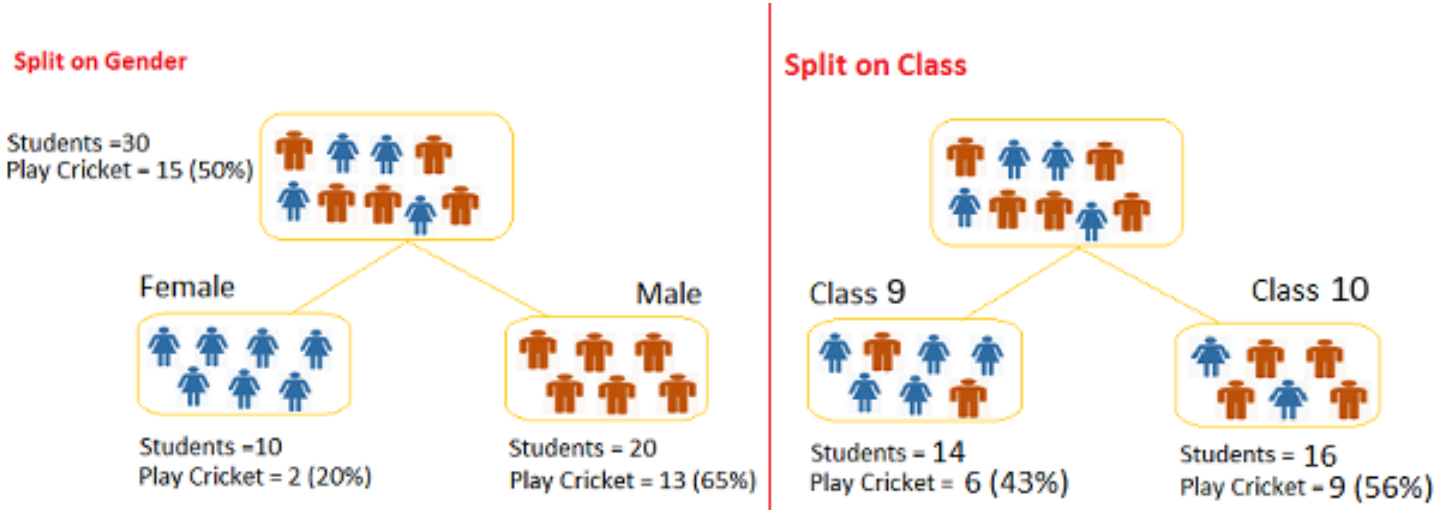
If we have C total classes and $p(i)$ is the probability of picking a datapoint with class i, then the **Gini impurity** is calculated as

$$G = \sum_{i=1}^{C} p(i) \times (1 - p(i))$$

The **weighted Gini impurity** is the Gini impurity multiplied by the number of samples in that node divided by the total number of samples in the parent node.
CART uses the Gini method to create binary splits.

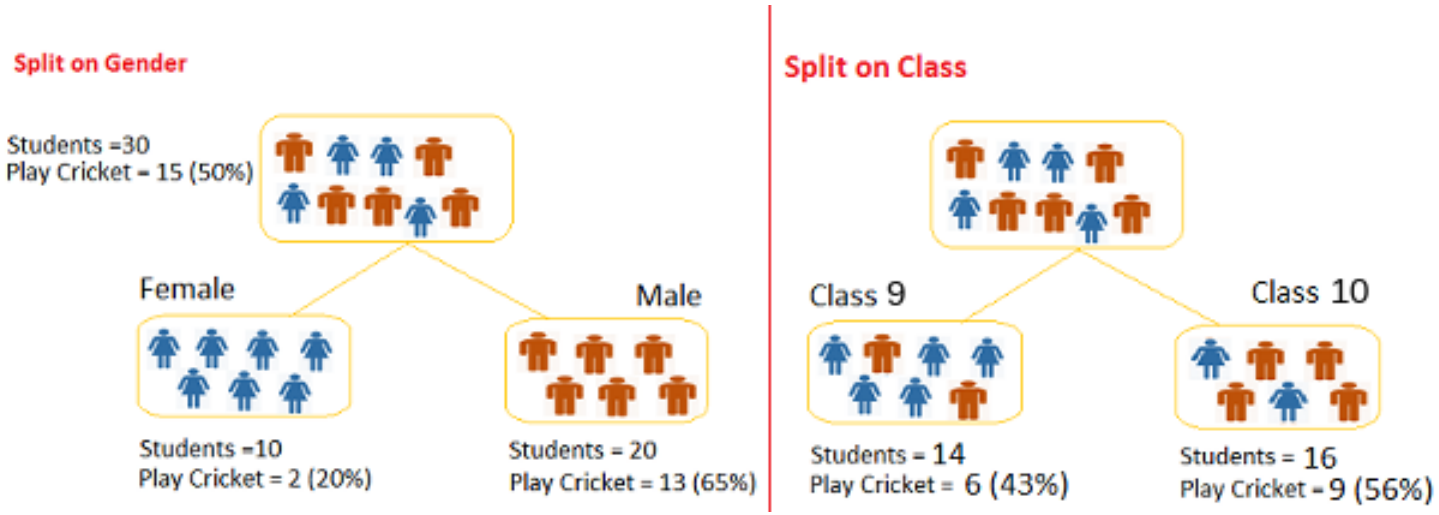
Gini Impurity

Example: We want to split students based on target variable (playing cricket or not).
We split the population using two input variables Gender and Class:



Gini Impurity

Example: We want to split students based on target variable (playing cricket or not).
We split the population using two input variables Gender and Class:

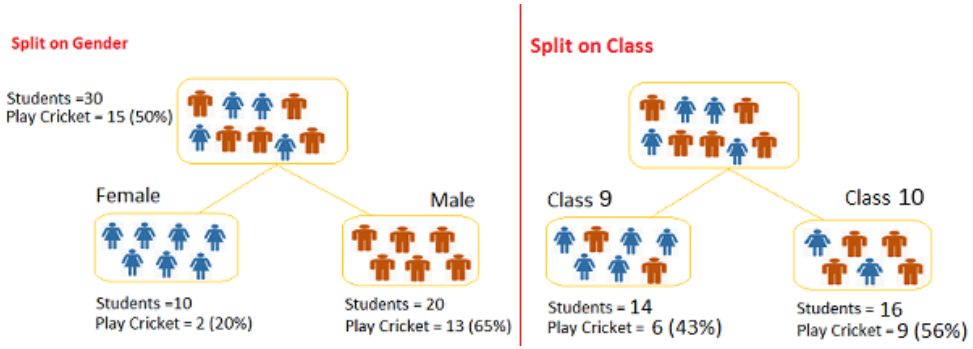


Now, we want to identify which split is producing more homogeneous sub-nodes using Gini.

Steps to calculate Gini impurity for a split:

- Calculate Gini impurity for sub-nodes.
- Calculate Gini impurity for split using weighted Gini score.

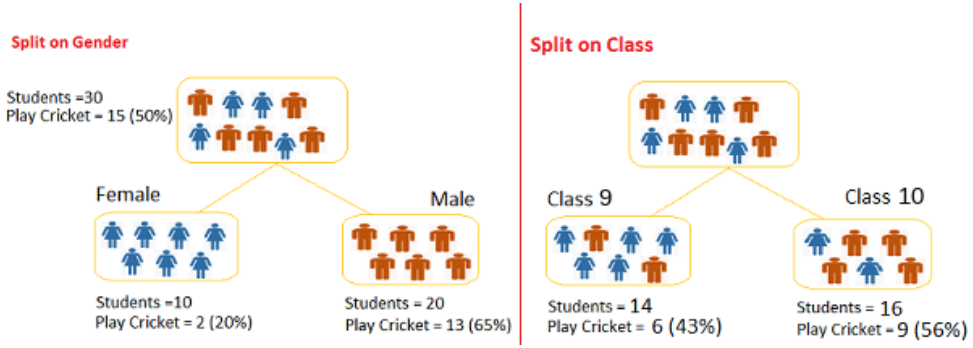
Gini Impurity



Split on Gender:
Gini for sub-node Female $= 0.2 \times 0.8 + 0.8 \times 0.2 = 0.32$
Gini for sub-node Male $= 0.65 \times 0.35 + 0.35 \times 0.65 = 0.45$
weighted Gini for Split Gender $= 10/30 \times 0.32 + 20/30 \times 0.45 = 0.41$

Split on Class:
Gini for sub-node Class 9 $= 0.43 \times 0.57 + 0.57 \times 0.43 = 0.49$
Gini for sub-node Class 10 $= 0.56 \times 0.44 + 0.44 \times 0.56 = 0.51$
weighted Gini for Split Class $= 14/30 \times 0.49 + 16/30 \times 0.51 = 0.50$

Gini Impurity



Split on Gender:
Gini for sub-node Female $= 0.2 \times 0.8 + 0.8 \times 0.2 = 0.32$
Gini for sub-node Male $= 0.65 \times 0.35 + 0.35 \times 0.65 = 0.45$
weighted Gini for Split Gender $= 10/30 \times 0.32 + 20/30 \times 0.45 = 0.41$

Split on Class:
Gini for sub-node Class 9 $= 0.43 \times 0.57 + 0.57 \times 0.43 = 0.49$
Gini for sub-node Class 10 $= 0.56 \times 0.44 + 0.44 \times 0.56 = 0.51$
weighted Gini for Split Class $= 14/30 \times 0.49 + 16/30 \times 0.51 = 0.50$

$\Rightarrow$ The Gini impurity for Split on Gender is lower than Split on Class, hence, the (fist) node split will take place on Gender.

Chi-Square (χ^2)

Chi-Square is an algorithm to find out the statistical significance between the differences between sub-nodes and parent node. We measure it by sum of squares of standardized differences between observed and expected frequencies of target variable.
The higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node. Chi-Square of each node is calculated using the equation:

$\chi^2 = ((\text{Actual - Expected})^2 \ / \ \text{Expected})^2/2$

Chi-Square is used for tree-based classifiers called CHAID (Chi-square Automatic Interaction Detector)

Steps to calculate Chi-square for a split:

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

Chi-Square (χ^2)

Example: We again use the students vs. cricket example that we have used to calculate Gini, but not calculate Chi-Square.

Split on Gender:

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

Split on Class:

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

Chi-Square (χ^2)

Example: We again use the students vs. cricket example that we have used to calculate Gini, but not calculate Chi-Square.

Split on Gender:

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

Split on Class:

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

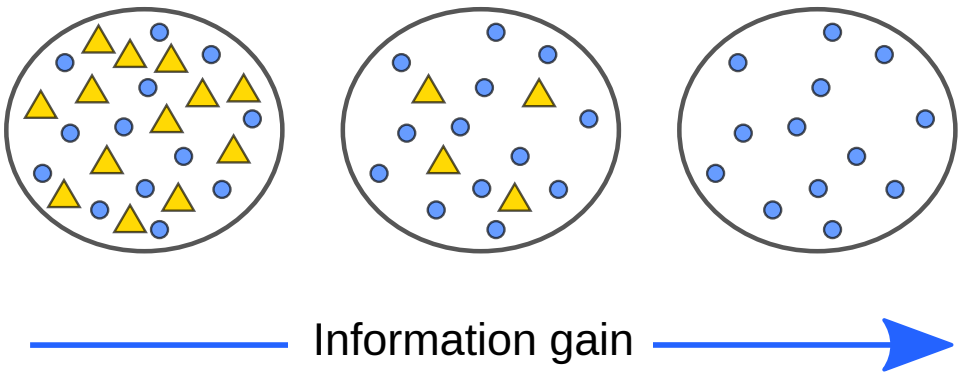
$\Rightarrow$ Chi-square also identifies the Gender split as more significant than the split on Class.

Information Gain

Information gain is a concept derived from entropy, measuring the reduction in uncertainty about the outcome variable achieved by splitting a dataset based on a particular feature. In tree-based algorithms, the splitting process involves selecting the feature and split point that maximize information gain.

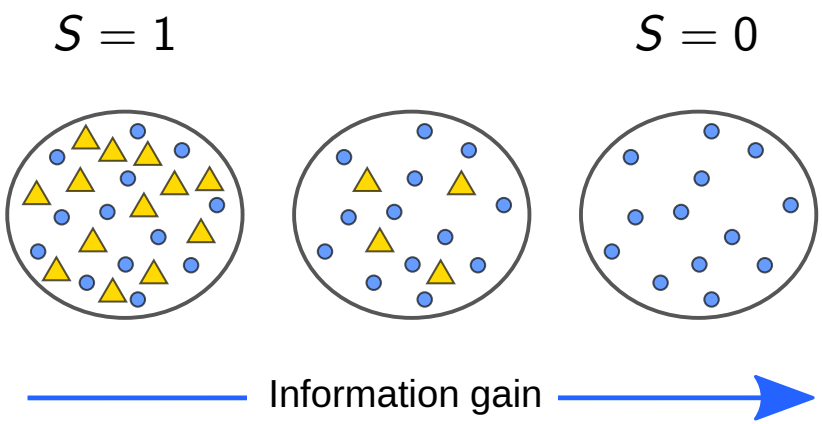
The goal is to iteratively choose splits that collectively lead to a tree structure capable of making accurate predictions on unseen data.

Entropy is a measure of information uncertainty in a dataset. In the context of decision trees, it quantifies the impurity or disorder within a node. Information gain is 1-Entropy.



Information Gain

Entropy S can be calculated using formula: $-S = -p \log_2 p - q \log_2 q$
If the sample is completely homogeneous, then the entropy is zero and if the sample is an equally divided, it has entropy of one.



Here p and q is probability of success and failure respectively in that node.

Entropy is also used with categorical target variable. It chooses the split which has lowest entropy compared to parent node and other splits. The lesser the entropy, the better it is.

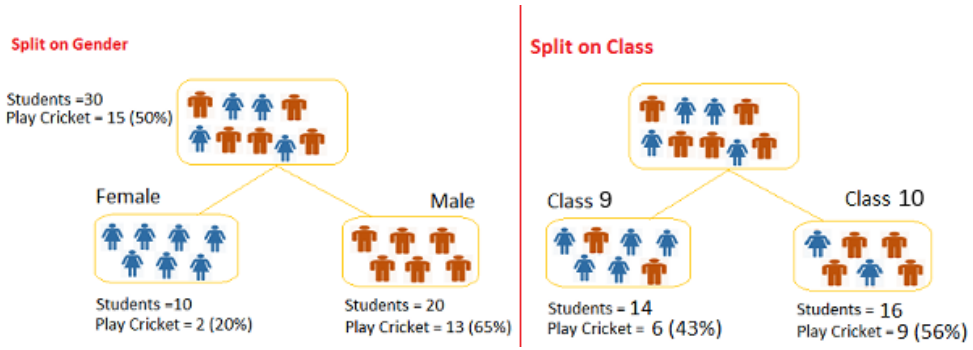
Information Gain

Steps to calculate Entropy for a split:

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

Information Gain

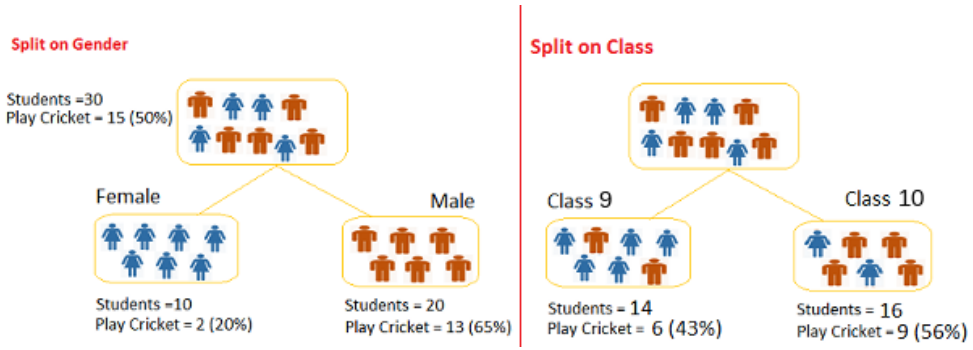
Example: We again use the students vs. cricket example.



Entropy parent node: $-(15/30)\log_2(15/30) - (15/30)\log_2(15/30) = 1.$
Entropy Female node: $-(2/10)\log_2(2/10) - (8/10)\log_2(8/10) = 0.72$
Entropy Male node: $-(13/20)\log_2(13/20) - (7/20)\log_2(7/20) = 0.93$
Entropy split Gender = Weighted sub-node entropy:
$(10/30) \times 0.72 + (20/30) \times 0.93 = 0.86$
Entropy Class 9 node: $-(6/14)\log_2(6/14) - (8/14)\log_2(8/14) = 0.99$
Entropy Class 10 node: $-(9/16)\log_2(9/16) - (7/16)\log_2(7/16) = 0.99.$
Entropy split Class: $(14/30) \times 0.99 + (16/30) \times 0.99 = 0.99$

Information Gain

Example: We again use the students vs. cricket example.



Entropy parent node: $-(15/30)\log_2(15/30) - (15/30)\log_2(15/30) = 1.$
Entropy Female node: $-(2/10)\log_2(2/10) - (8/10)\log_2(8/10) = 0.72$
Entropy Male node: $-(13/20)\log_2(13/20) - (7/20)\log_2(7/20) = 0.93$
Entropy split Gender = Weighted sub-node entropy:
$(10/30) \times 0.72 + (20/30) \times 0.93 = 0.86$
Entropy Class 9 node: $-(6/14)\log_2(6/14) - (8/14)\log_2(8/14) = 0.99$
Entropy Class 10 node: $-(9/16)\log_2(9/16) - (7/16)\log_2(7/16) = 0.99.$
Entropy split Class: $(14/30) \times 0.99 + (16/30) \times 0.99 = 0.99$

$\Rightarrow$entropy for split on Gender is the lowest among all, so the tree will (first) split on Gender

Reduction in Variance

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

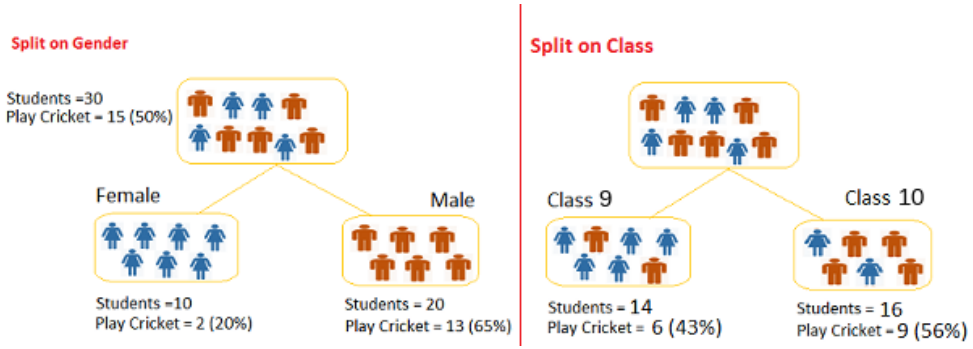
$$V = \frac{\sum(X - \bar{X})^2}{n}$$

$\bar{X}$ is mean of the values, X is actual and n is number of values.
Steps to calculate Variance for a split:

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

Reduction in Variance

Example: We assign 1 for play cricket and 0 for not playing cricket.



Mean for Root node: $(15 \times 1 + 15 \times 0)/30 = 0.5$
Variance for Root node: $(15 \times (1 - 0.5)^2 + 15 \times (0 - 0.5)^2)/30 = 0.25$
Mean of Female node: $(2 \times 1 + 8 \times 0)/10 = 0.2$
Variance of Female node: $(2 \times (1 - 0.2)^2 + 8 \times (0 - 0.2)^2)/10 = 0.16$
Mean of Male Node: $(13 \times 1 + 7 \times 0)/20 = 0.65$
Variance of Male Node: $(13 \times (1 - 0.65)^2 + 7 \times (0 - 0.65)^2)/20 = 0.23$
Variance for Split Gender: $(10/30) \times 0.16 + (20/30) \times 0.23 = 0.21$
Mean of Class 9 node: $(6 \times 1 + 8 \times 0)/14 = 0.43$
Variance of Class 9 node: $(6 \times (1 - 0.43)^2 + 8 \times (0 - 0.43)^2)/14 = 0.24$
Mean of Class 10 node: $(9 \times 1 + 7 \times 0)/16 = 0.56$
Variance of Class 10 node: $(9 \times (1 - 0.56)^2 + 7 \times (0 - 0.56)^2)/16 = 0.25$
Variance for Split Class: $(14/30) \times 0.24 + (16/30) \times 0.25 = 0.25$

$\Rightarrow$ Gender split has lower variance, so the split would take place on Gender

Avoiding Overfitting

Overfitting is one of the key challenges faced while using tree based algorithms. If there is no limit set of a decision tree, it will give you 100% accuracy on training set because in the worse case it will end up making 1 leaf for each observation.

Thus, preventing overfitting is essential while modeling a decision tree. This can be done in 2 ways:

- Setting constraints on tree size
- Tree pruning

In the following we will discuss both of these approaches.

Setting Constraints on Tree-based Algorithms

This can be done by placing constraints on various parameters defining a decision tree. First, lets look at the general structure of a decision tree:

In the following, typical constraints are given. It is important to understand the role of parameters used in tree modeling. They are available in typical machine-learning libraries such as scikit-learn.

Minimum samples for a node split
Defines the minimum number of samples which are required in a node to be considered for splitting. This controls overfitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree. Too high values can lead to underfitting. Thus, it should be tuned using CV.

Minimum samples for a terminal node (leaf)
Defines the minimum samples required in a terminal node or leaf. Used to control overfitting. Generally, lower values should be chosen for imbalanced class problems, as the regions in which the minority class will be in majority will be very small.

Setting Constraints on Tree-based Algorithms

Depth of tree (vertical depth)
The maximum depth of a tree. Used to control overfitting as higher depth will allow model to learn relations very specific to a particular sample. Should be tuned using CV.

Maximum number of terminal nodes
The maximum number of terminal nodes or leaves in a tree. Can be defined in place of the maximum depth. Since binary trees are created, a depth of n would produce a maximum of 2^n leaves.

Features to consider for split
The number of features to consider while searching for a best split. These will be randomly selected. As a rule-of-thumb, square root of the total number of features works great but we should check up to 30-40% of the total number of features. Higher values can lead to overfitting.

Pruning in Tree-based Algorithms

Decision tree pruning is a technique used to prevent decision trees from overfitting the training data.

Pruning aims to **simplify the decision tree** by removing parts of it that do not provide significant predictive power, thus improving its ability to generalize to new data. This results in more fast, more accurate and more effective predictions.

In simpler terms, the aim of Decision Tree Pruning is to construct an algorithm that will **perform worse on training data** but will **generalize better on test data**.

Pruning in Tree-based Algorithms

Decision tree pruning is a technique used to prevent decision trees from overfitting the training data.

Pruning aims to **simplify the decision tree** by removing parts of it that do not provide significant predictive power, thus improving its ability to generalize to new data. This results in more fast, more accurate and more effective predictions.

In simpler terms, the aim of Decision Tree Pruning is to construct an algorithm that will **perform worse on training data** but will **generalize better on test data**.

There are two main types of decision tree pruning: **Pre-Pruning** and **Post-Pruning**.

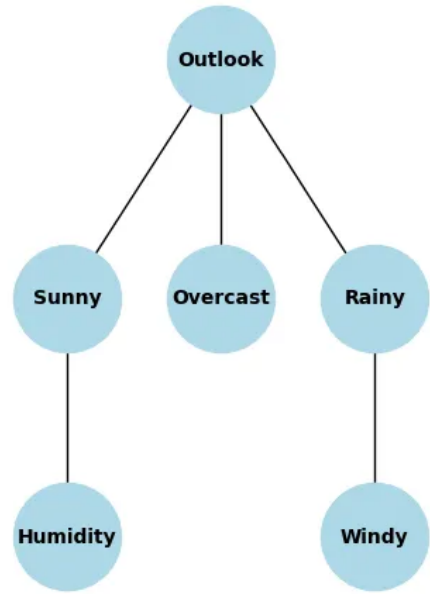
Pruning in Tree-based Algorithms

Pre-Pruning (Early Stopping)

Stopping the growth of the decision tree before it gets too complex is called pre-pruning.

Some common pre-pruning techniques include:

- Maximum Depth
- Minimum Samples per Leaf
- Minimum Samples per Split
- Maximum Features



stop after second level

By pruning early, we get a simpler tree that is less likely to overfit the training facts.

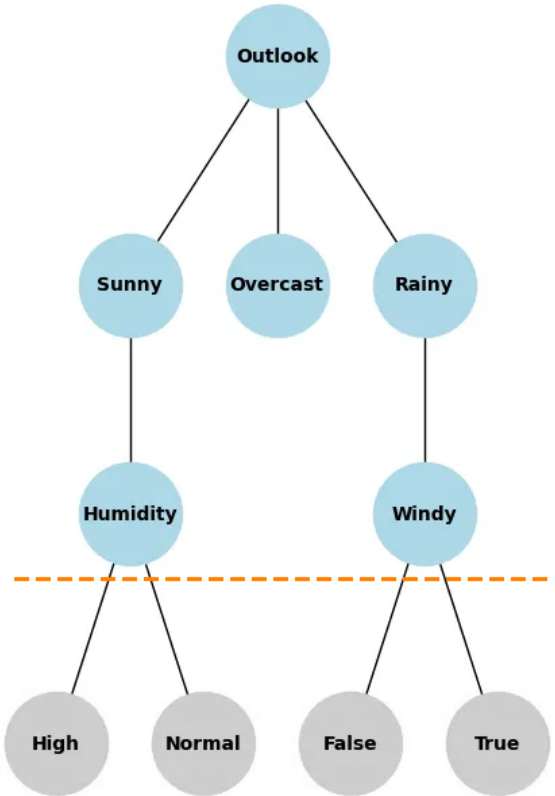
Pruning in Tree-based Algorithms

Post-Pruning (Reducing Nodes)

After the tree is fully grown, post-pruning involves removing branches or nodes to improve the model's ability to generalize.

Some common post-pruning techniques include:

- Cost-Complexity Pruning (CCP): This method assigns a price to each subtree primarily based on its accuracy and complexity, then selects the subtree with the lowest fee.
- Reduced Error Pruning: Removes branches that do not significantly affect the overall accuracy.
- Minimum Impurity Decrease: Prunes nodes if the decrease in impurity (Gini impurity or entropy) is beneath a certain threshold.
- Minimum Leaf Size: Removes leaf nodes with fewer samples than a specified threshold.



Pruning in Tree-based Algorithms

One of the simplest pruning algorithms is **Reduced Error Pruning**.

Starting at the leaves, each node is replaced with its most popular class. If the prediction accuracy is not affected then the change is kept. While somewhat naive, reduced error pruning has the advantage of simplicity and speed.

Pruning in Tree-based Algorithms

Weakest Link Pruning (also known as Cost complexity pruning) is a more advanced algorithm. It generates a series of trees $T_0 \dots T_m$ where T_0 is the initial tree and T_m is the root alone. At step i, the tree is created by removing a subtree from tree $i-1$ and replacing it with a leaf node with value chosen as in the tree building algorithm. The subtree that is removed is chosen as follows:
For doing so, Weakest link pruning introduces a nonnegative tuning parameter α, for regression trees minimizing

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

where $|T|$ is the number of terminal nodes (size) of the tree T. For classification trees, replace RSS with objective function, e.g. Gini index.

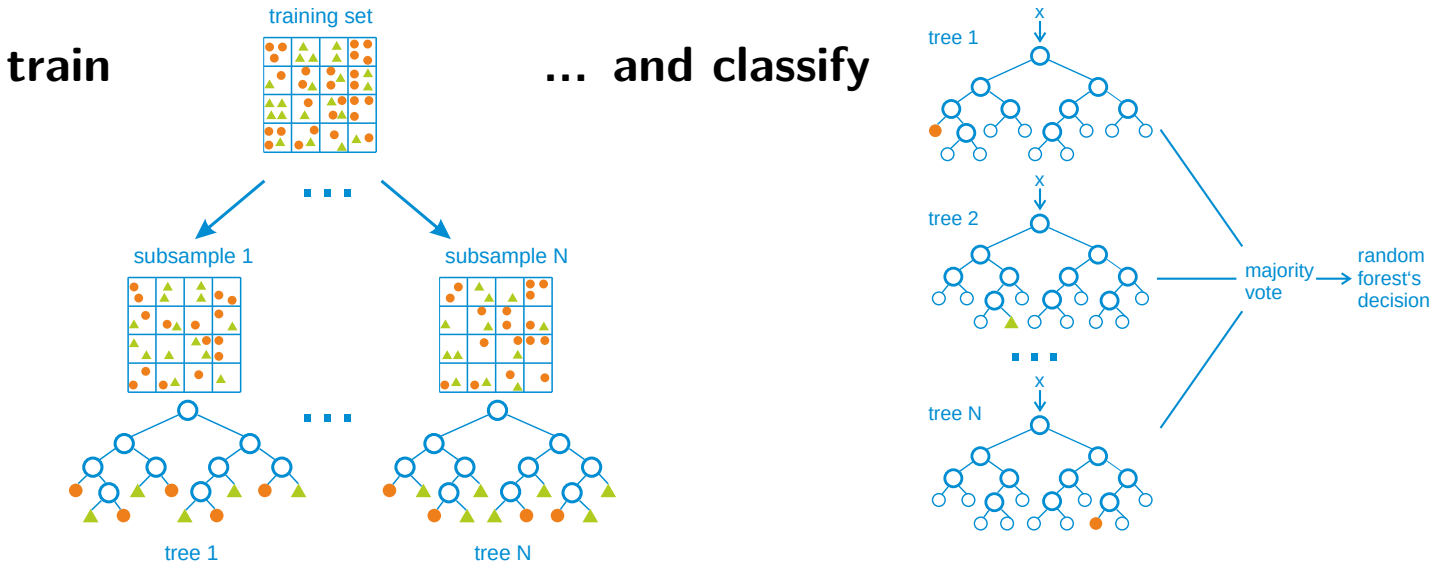
Minimization is achieved by attempting to remove terminal nodes from the base of the tree upward. Similar to Lasso regression, α is a penalty on the size (complexity) of the tree. When $\alpha = 0$, we will keep the whole tree and when $\alpha = \infty$, we will prune everything back to null tree. We select α using cross-validation.

Ensemble Methods

Decision trees have one problem: Their results often don't generalize to new datasets.

To overcome the problem of overfitting, an **ensemble** of decision trees can be created. The trees are built independently from bootstrap samples (with replacement).

Performance increases as more trees are added, to a limit. Final predictions are made either by voting or averaging predictions across all trees. A feature importance ranking can also be obtained.



Ensemble Methods

We will now take a look at different ensemble methods.

Bagging

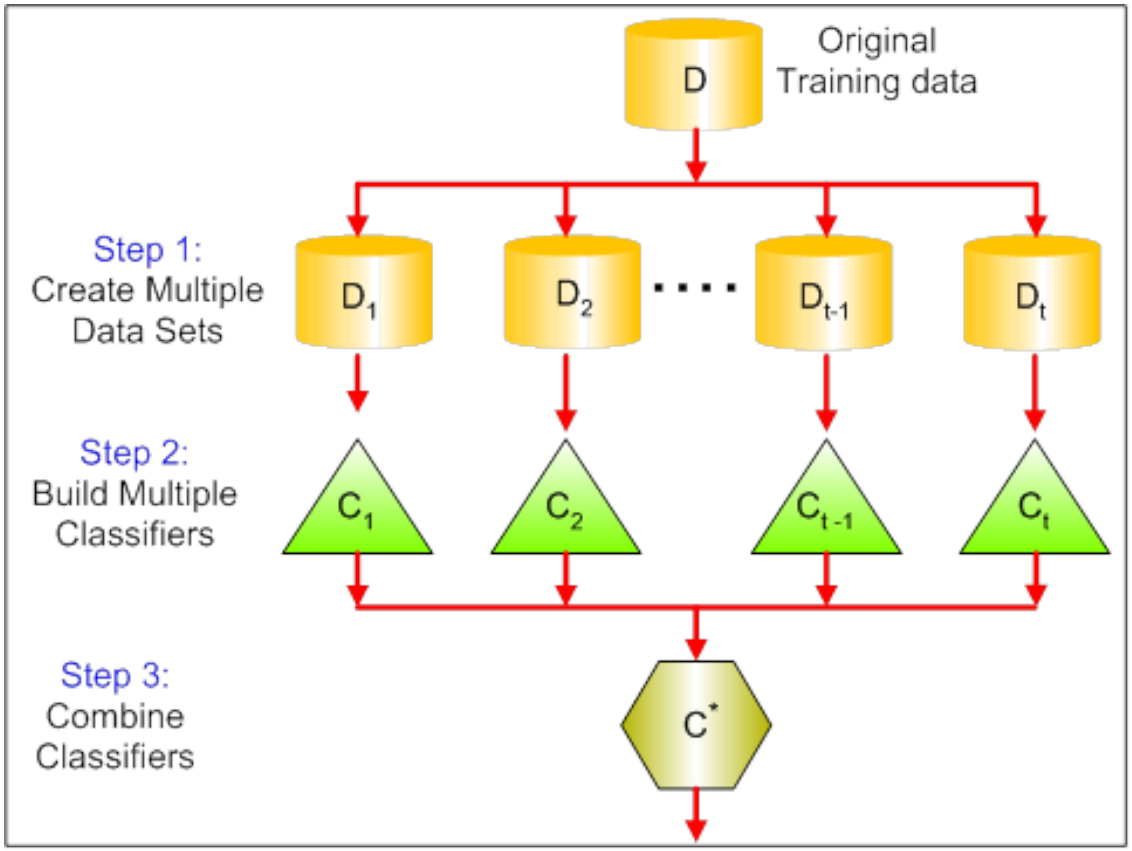
We have seen CART has attractive features, but the biggest problem is high variance. **Bootstrap aggregating (bagging)** is an ensemble method designed to improve the stability and accuracy of regression and classification algorithms by model averaging. Although bagging helps to reduce variance and avoid overfitting, model interpretability will be sacrificed.

A **bootstrap sample** is a random sample of the data taken with replacement, which means samples in the original data set may appear more than once in the bootstrap sample. Note that when samping from the training data $(y_1, \mathbf{x}_1), ..., (y_n, \mathbf{x}_n)$ in paris, relationship between y and $\mathbf{x}$ is preserved.

A bootstrap sample of size m is $(y_i^*, \mathbf{x}_i^*)_{i=1}^m$, were each $(y_i^*, \mathbf{x}_i^*)$ is a uniformly draw from the training data $(y_1, \mathbf{x}_1), ..., (y_n, \mathbf{x}_n)$.

Bagging

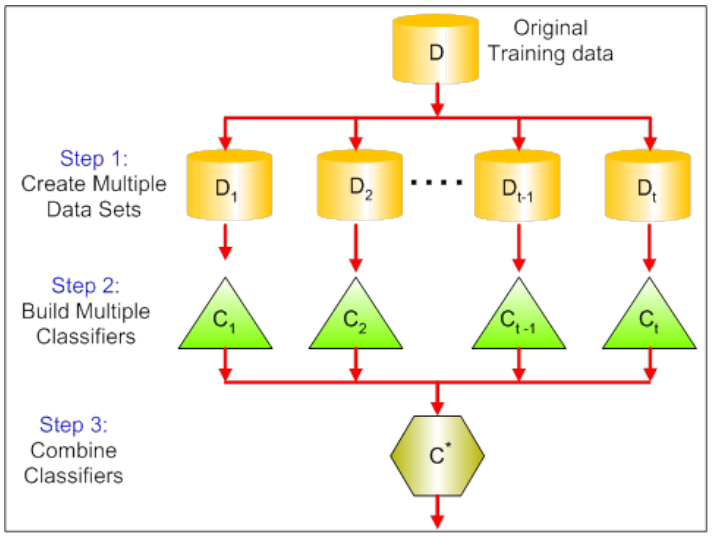
Bagging attempts at reducing the variance of our predictions by combining the result of multiple classifiers modeled on different sub-samples of the same data set.



Bagging

The steps followed in bagging are:

1. Create Multiple DataSets:
Sampling is done with replacement on the original data and new datasets are formed.
The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model.
Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting.



2. Build Multiple Classifiers:
Classifiers are built on each data set. Generally the same classifier is modeled on each data set and predictions are made.

3. Combine Classifiers:
The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand.

Bagging

Note that, here the number of models built is not a hyper-parameters. Higher number of models are always better or may give similar performance than lower numbers. Theoretical analysis demonstrates that the variance of the combined predictions is reduced to $1/n$ (where n is the number of classifiers) of the original variance, under certain assumptions.

There are various implementations of bagging models. Random forest is one of them.

Bagging applied to CART

For $b = 1, ..., B$, draw n bootstrap samples $(y_i^*, \mathbf{x}_i^*)_{i=1}^n$ and each time fit a tree and get $\hat{f}^{*b}(\mathbf{x})$, the prediction at a point $\mathbf{x}$.

For **regression trees**, average all the predictions to obtain:

$$\hat{f}^{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(\mathrm{x})$$

For **classification trees**, we choose the class label for which the most bootstrapped trees "voted", or construct a probabilistic prediction for jth class by directly averaging tree output probabilities:

$$\hat{f}_j^{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_j^{*b}(\mathrm{x})$$

Bagging applied to CART

For $b = 1, ..., B$, draw n bootstrap samples $(y_i^*, \mathbf{x}_i^*)_{i=1}^n$ and each time fit a tree and get $\hat{f}^{*b}(\mathbf{x})$, the prediction at a point $\mathbf{x}$.

For **regression trees**, average all the predictions to obtain:

$$\hat{f}^{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(\mathrm{x})$$

For **classification trees**, we choose the class label for which the most bootstrapped trees "voted", or construct a probabilistic prediction for jth class by directly averaging tree output probabilities:

$$\hat{f}_j^{\text{bag}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_j^{*b}(\mathrm{x})$$

A nice property of bagging for CART is that it automatically provides out-of-sample evaluation. On average, 36.8% of original data will not be in the bootstrap sample, so can be used as a test set. This is approximately equal to a 3-fold cross validation.

Bagging

The **advantages** of bagging are:

If you have a lot of data, bagging is a good choice because the empirical distribution will be close to the true population distribution

Bagging is not restricted to trees, it can be used with any type of method.

It is most useful when the variance of a predictor should be reduced. Under the (inaccurate) independence assumption, the variance will be reduced by a factor of $\sim 1/B$ for B bootstrap resamples (Central limit theorem).

Out-of-sample evaluation without using cross-validation, since any given observation will not be used in around 36.8% of the models.

The **disadvantages** of bagging are:

We lose interpretability as the final estimate is not a tree.

Computational cost is multiplied by a factor of at least B.

Bagging trees are correlated, the more correlated random variables are, the less the variance reduction of their average.

Random Forest

Bagging improved on a single CART model by reducing the variance through resampling. But, in fact, the bagged models are still quite correlated. The more correlated random variables are, the less the variance reduction of their average. Can we make them more independent in order to produce a better variance reduction?

Random Forest achieves this by not only resampling observations, but by restricting the model to random subspaces, $\chi^* \subset \chi$.

Random Forest

The Random Forest algorithm can be implemented in the following:

1. Take a bootstrap resample $(y_i^*, \mathbf{x}_i^*)_{i=1}^n$ of the training data as in bagging.

2. Building a tree, each time a split in a tree is considered, random select m predictors out of the full set of p predictors as split candidates, and find the optimal split within those m predictors. (Typically choose $m \approx \sqrt{p}$.) The best split on these m is used to split the node. The value of m is held constant while we grow the forest. Each tree is grown to the largest extent possible and there is no pruning.

3. Repeat 1) and 2), average the prediction of all trees.

Random Forest

The **advantages** of random forests are:

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Works well with high-dimensional data
- Tuning is rarely needed (easy to get good quality forecast)

The **disadvantages** of random forests are:

- Often quite suboptimal for regression.
- Same extrapolation issue as trees.
- Harder to implement and memory-hungry model.

Boosting

The term **Boosting** refers to a family of algorithms which converts weak machine-learning algorithms to stronger ones.
Boosting is similar to bagging in the sense that we combine results from multiple classifiers, but it is fundamentally different in approach:

1. Set $\hat{f}(\mathbf{x}) = 0$ and $\epsilon_i = y_i$ for $i = 1, ..., n$.
2. For $b = 1, ..., B$, iterate:
   i) Fit a model (e.g.: tree) $\hat{f}^b(\mathbf{x})$ to the response $\epsilon_1, ..., \epsilon_n$.
   ii) Update the predictive model to $\hat{f}^b(\mathbf{x}) \leftarrow \hat{f}^b(\mathbf{x}) + \lambda \hat{f}^b(\mathbf{x})$
   iii) update the residuals $\epsilon_i \leftarrow \epsilon_i - \lambda \hat{f}^b(\mathbf{x}) \; \forall i$
3. Output as final boosted model $\hat{f}(\mathbf{x}) = \sum_{b=1}^{B} \lambda \hat{f}^b(\mathbf{x})$.

It combines the outputs from weak learner and creates a strong learner which eventually improves the prediction power of the model. Boosting pays higher focus on examples which are mis-classified or have higher errors by preceding weak rules.

Boosting

properties of Boosting:

Intuitively, Boosting is a slow learning approach: Aim to learn usually simple model (low variance, high bias) in each round. Then bring bias down by repeating over many rounds.

Boosting is sequential learning: Each round of boosting aims to correct the error of the previous rounds.

Boosting has a generic methodology: Any model can be boosted.

Boosting is an incredibly powerful technique and regularly is winning machine learning competitions.

However, **tuning** is needed (not trivial) and some difficulties in interpretation and extrapolation.

Boosting

Tuning parameters for boosting

There are three main hyper-parameters that need to be tuned for building boosting tree model.

1. The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.

2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.

3. The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree consists of a single split and results in an additive model. More generally d is the interaction depth and controls the interaction order of the boosted model since d splits can involve at most d variables.

Summary

Tree-based algorithms are important for supervised machine-learning problems.

We discussed about tree-based algorithms from scratch. Additionally, we learned the importance of decision trees and how that simplistic concept is being used in ensemble method algorithms that can overcome the problem of overfitting.

Ensemble learning is the process of using multiple models, trained over the same data, averaging the results of each model ultimately finding a more powerful prediction/classification result. The result is a more robust classifier.

Summary

Are tree based algorithms better than linear models? Which algorithm to use?

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

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

Summary

Next time, in our final lecture, we will learn about **Unsupervised Learning**.