Decision Trees: Introduction to Tree-Based Models

ISE – 364 / 464

DEPT. INDUSTRIAL & SYSTEMS ENGINEERING

GRIFFIN DEAN KENT

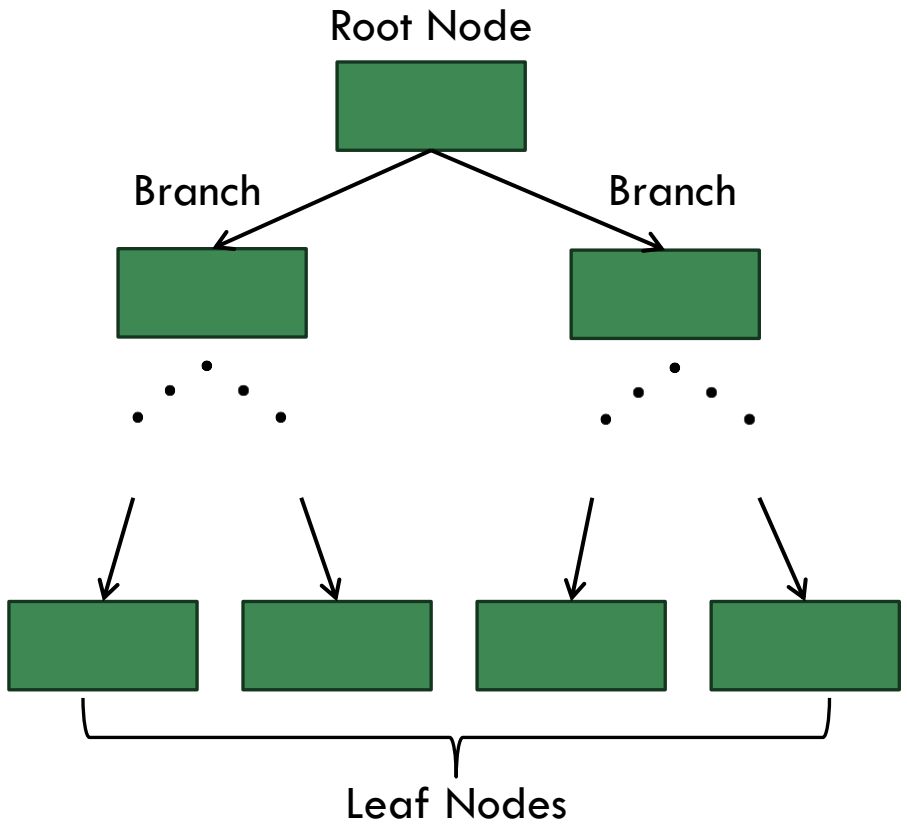
LEHIGH UNIVERSITY.

Overview of Decision Trees

Decision Tree

A decision tree is a discriminative supervised machine learning algorithm that can be used for both classification and regression problems. These models learn a series of "rules" (or decisions) which can be written in a tree-like structure (hence the name). Each decision in the tree can be represented as a node that can split into a variety of different branches depending on the outcome of the decision, and where each branch represents an outcome of that decision. In a decision tree, there are three main components:

- **The Root Node**: The starting point, representing the entire dataset.
- **The Branches**: The possible outcomes of decisions based on the features of the dataset.
- **The Leaf Nodes**: The final decision nodes of the tree which return a predicted class or a predicted numerical value.



Mathematical Notation for Decision Trees

Tree Notation

We denote a decision tree $\mathcal{T}$ as a set of $K+1 \in \mathbb{N}$ nodes $N \in \mathcal{T}$, which are either "**internal**" nodes (decision nodes) or "**leaf**" nodes (prediction nodes). We can write such a tree as

$$\mathcal{T} \coloneqq \{N_0, N_1, \dots, N_K\},$$

where N_0 is the "**root**" node. We can use the cardinality function to denote the total number of nodes in the tree as $|\mathcal{T}|$ (notice that in this general case, $|\mathcal{T}| = K+1$).

Internal Node: We define these nodes as

$$N_k \coloneqq \{j(k), \theta_k, N_{k,\text{left}}, N_{k,\text{right}}\},$$

where $j(k)$ represents the index of the feature that is used for splitting the node with the decision rule $x_{j(k)} \leq \theta_k$ to generate the two child nodes $N_{k,\text{left}}$ and $N_{k,\text{right}}$. Further, every node $N_k \in \mathcal{T}$ has associated with it some subset $S_{N_k} \subseteq \mathcal{D}$ (which we refer to as "region" S_k or "split" S_k) of the original dataset that satisfy the sequential decisions that lead to node N_k. As such, we can denote the new regions that are defined by the decision rule $x_{j(k)} \leq \theta_k$ as

$$S_{k,\text{left}} \coloneqq \{x \in S_k | x_{j(k)} \leq \theta_k\} \quad \text{and} \quad S_{k,\text{right}} \coloneqq \{x \in S_k | x_{j(k)} > \theta_k\}.$$

Lastly, notice that $S_0 = \mathcal{D}$.

Leaf Node: We define these nodes as

$$N_k \coloneqq \{c_k\},$$

where c_k is the predicted target class (for classification) or numeric value (for regression) and is computed as some form of aggregation of the target entries in the split S_k.

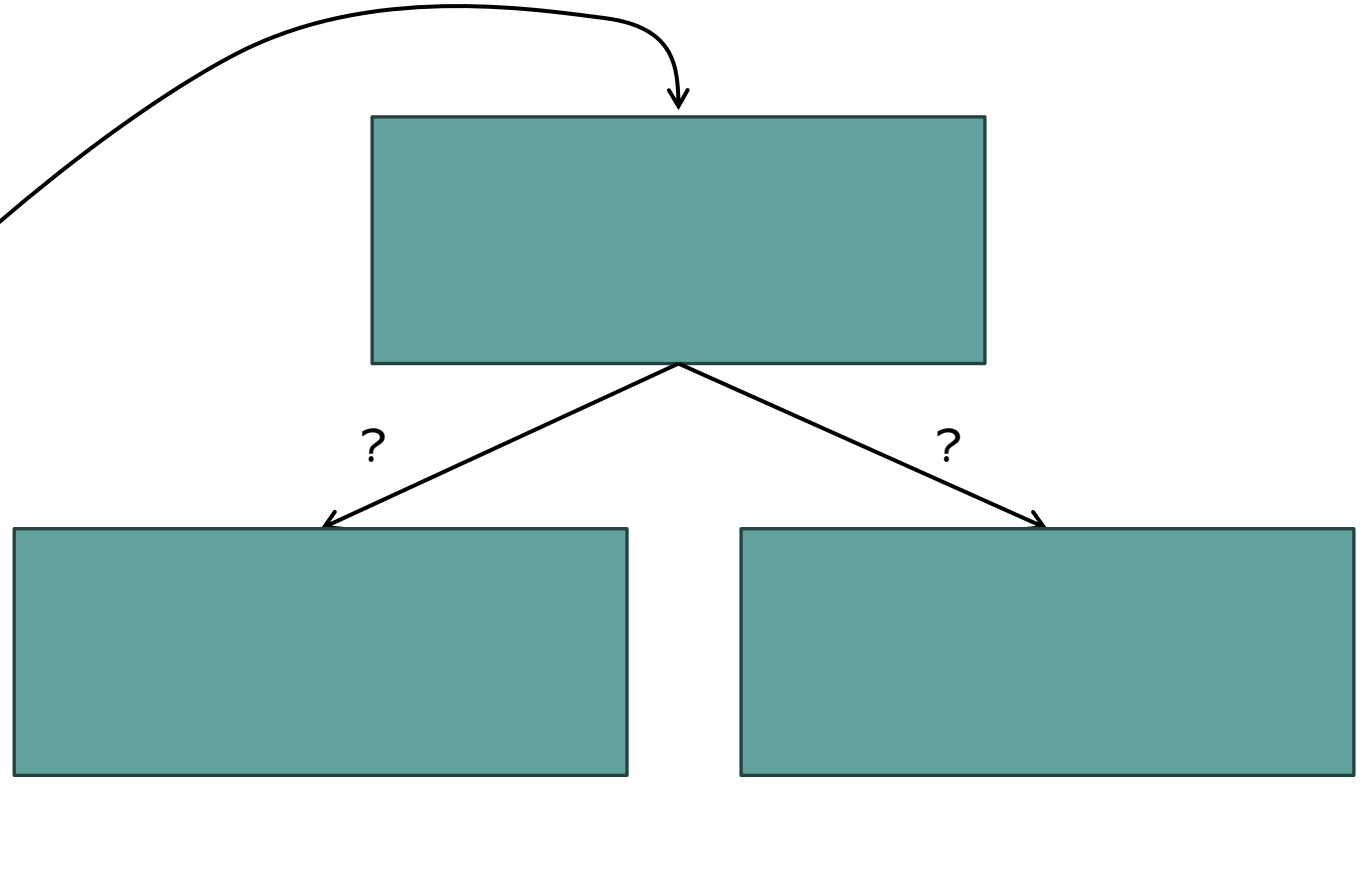
A Simple Classification Example

Original Dataset: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, where we only have 1 feature x_1 and a binary target $y \in \{0,1\}$. Specifically, the data are given as:

$$x_1 = \begin{bmatrix} 7 \\ 12 \\ 18 \\ 35 \\ 38 \\ 50 \end{bmatrix} \text{ and } y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

Thus, our dataset is given by the set of datapoints:

$$\mathcal{D} = \{(7,0), (12,0), (18,1), (35,1), (38,0), (50,1)\}.$$



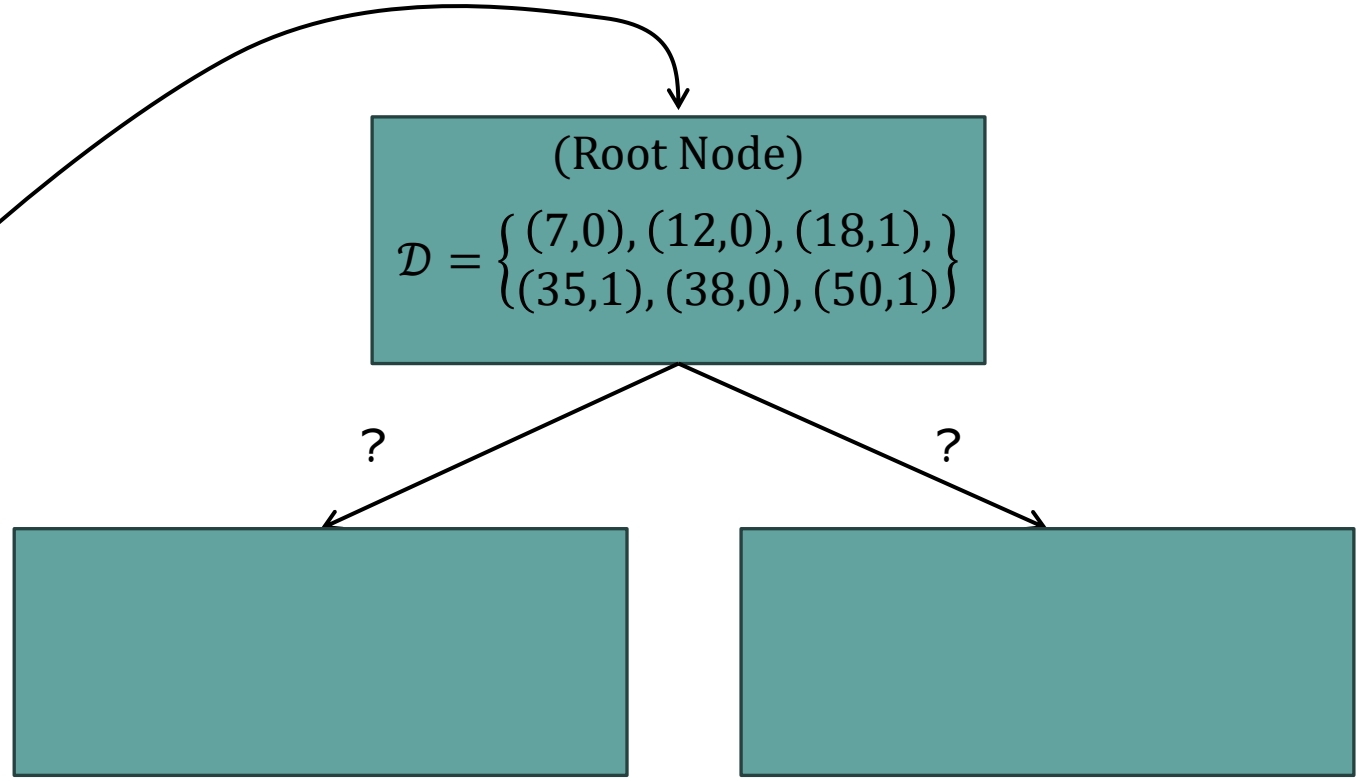
A Simple Classification Example

Original Dataset: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, where we only have 1 feature x_1 and a binary target $y \in \{0,1\}$. Specifically, the data are given as:

$$x_1 = \begin{bmatrix} 7 \\ 12 \\ 18 \\ 35 \\ 38 \\ 50 \end{bmatrix} \text{ and } y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

Thus, our dataset is given by the set of datapoints:

$$\mathcal{D} = \begin{Bmatrix} (7,0), (12,0), (18,1), \\ (35,1), (38,0), (50,1) \end{Bmatrix}.$$



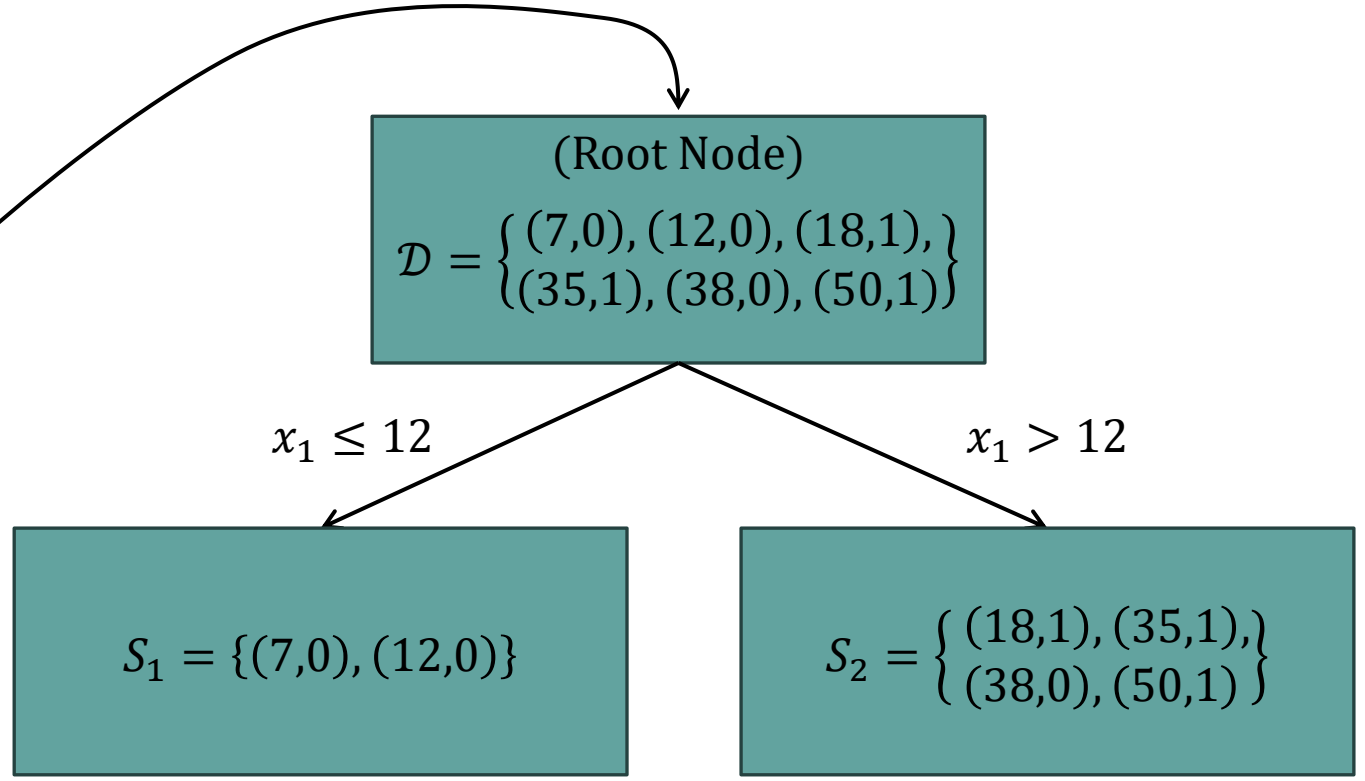
A Simple Classification Example

Original Dataset: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$, where we only have 1 feature x_1 and a binary target $y \in \{0,1\}$. Specifically, the data are given as:

$$x_1 = \begin{bmatrix} 7 \\ 12 \\ 18 \\ 35 \\ 38 \\ 50 \end{bmatrix} \text{ and } y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}.$$

Thus, our dataset is given by the set of datapoints:

$$\mathcal{D} = \begin{Bmatrix} (7,0), (12,0), (18,1), \\ (35,1), (38,0), (50,1) \end{Bmatrix}.$$



This split of the data **minimizes the impurity** of the split (i.e., it **maximizes the class homogeneity** of the split).

A Simple Classification Example

We can define this tree using our mathematical notation as:

$$\mathcal{T} = \{N_0, N_1, N_2\},$$

where

$$N_0 = \{1{,}12, N_1, N_2\},$$
$$N_1 = \{0\},$$
$$N_2 = \{1\},$$

and where

$$S_0 = \mathcal{D} = \begin{Bmatrix}(7{,}0), (12{,}0), (18{,}1),\\ (35{,}1), (38{,}0), (50{,}1)\end{Bmatrix},$$
$$S_1 = \{(7{,}0), (12{,}0)\},$$
$$S_2 = \begin{Bmatrix}(18{,}1), (35{,}1),\\ (38{,}0), (50{,}1)\end{Bmatrix}.$$

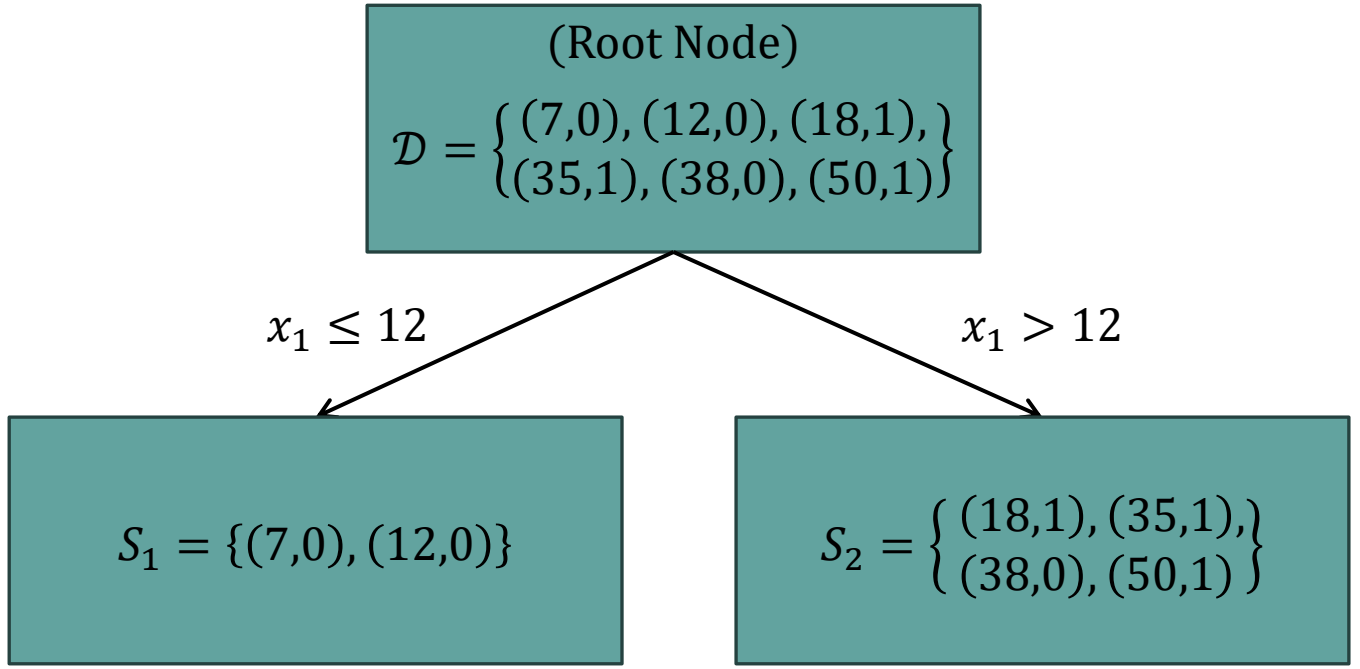
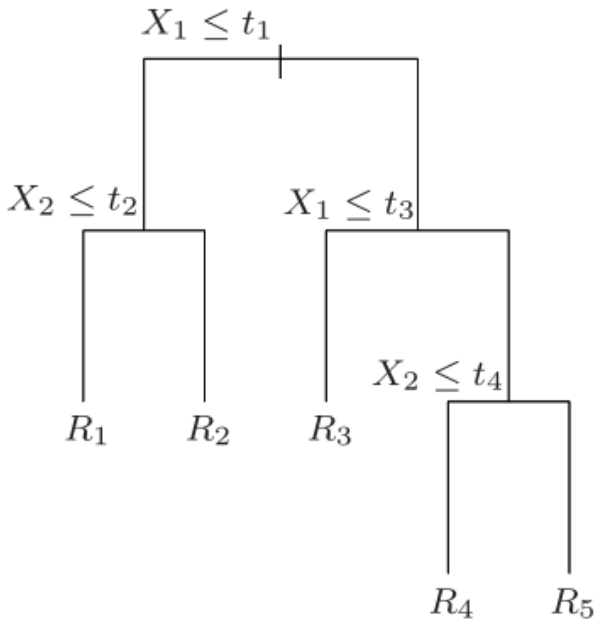
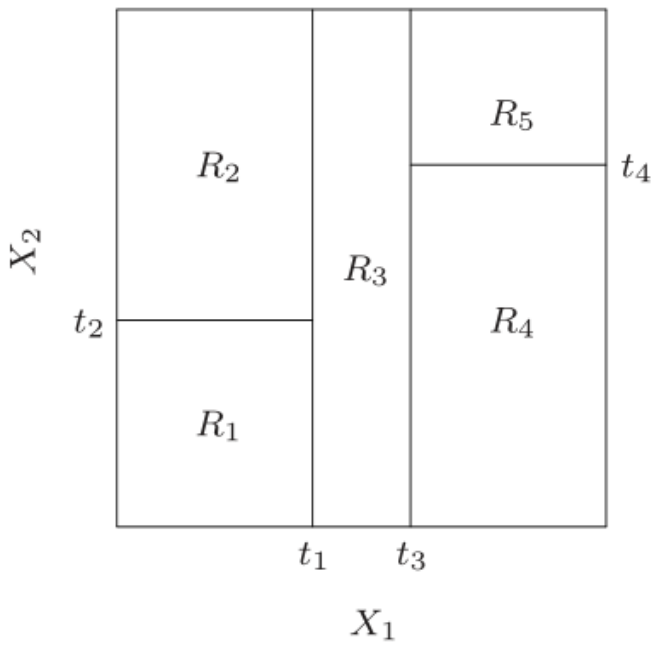
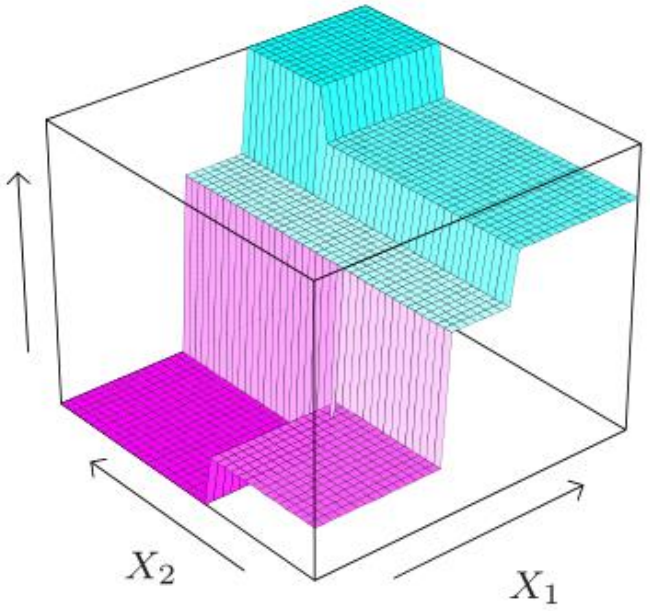


Illustration of the Decision Space

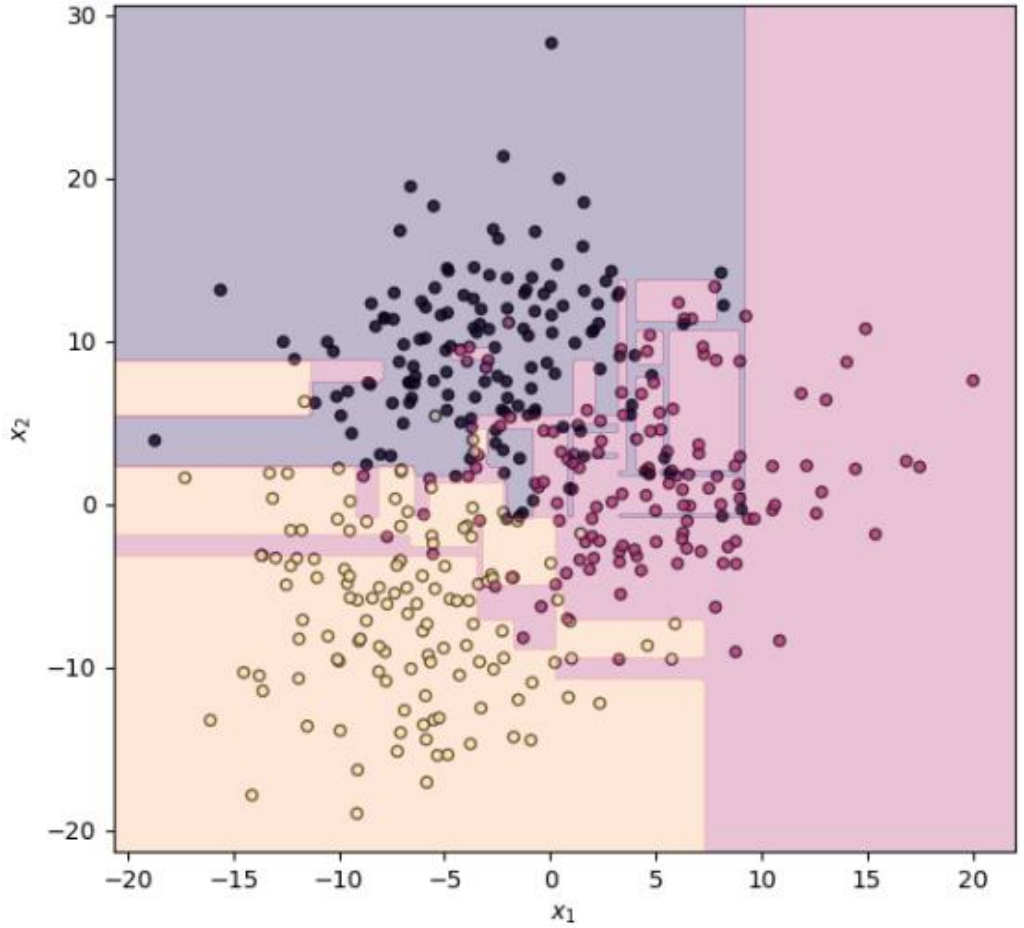






Reference: T. Hastie, R. Tibshirani, J. Friedman. "*The Elements of Statistical Learning*". Springer, 2009.

Illustration of the Decision Space



Classification Trees Vs Regression Trees

Decision trees are popular and easy to understand due to their interpretability and intuitive nature, and which can be simply summarized as **a series of rules** (or decisions) **that are used to come to a prediction**.

At a high level, if our goal is to predict a discrete target variable, then we will utilize a **classification tree**. Similarly, if our goal is to predict a continuous target variable, then we will utilize a **regression tree**.

- Although both types of decision trees are similar in their structure and algorithmic implementation, they both differ in the process of choosing which features in the dataset to split on as well as the evaluation metrics used.

The **primary goal** in a decision tree is to learn a series of decision rules that will ultimately enable predictive capabilities. The training process of choosing the decision rules typically amounts to **choosing to split the nodes** of the tree on features that will **maximize the homogeneity** (how similar the classes are in the resulting nodes) of the split in the target classes. Thus, we will want to **choose a splitting criteria** that will ideally separate all datapoints with a particular target class value into one branch and the datapoints with a different target class into a different branch.

- There are typically two evaluation metrics used to measure this homogeneity.

Measures of Impurity: Classification Tree Metrics

A Criteria to Measure the "Goodness" of a Node Split

As mentioned earlier, when dealing with decision tree predictive models, the primary challenge one faces is to **identify (or learn) the decisions** that will do the branching in the tree which will yield the best predictive power.

- The most intuitive way to talk about this is in terms of classification trees, but the same reasoning holds for regression trees.

Since decision trees are trying to learn a set of "decisions" to correctly separate a dataset into distinct target classes based solely on the features of the design matrix, one can evaluate how well a decision tree goes about doing this by measuring the mixture of target classes that are present in each of the leaf nodes.

- In a "perfect" decision tree (with 100% classification accuracy), each leaf node would be completely **homogeneous** (i.e., the datapoints in each leaf node are of a single target class, and there are no more than a single target class present in each leaf node).
- There are two main evaluation metrics for measuring the "level of homogeneity" (and correspondingly the "level of impurity") that a decision results in: **Entropy** / **Information Gain** and **Gini Impurity**.

Entropy: Measures of Impurity

Entropy

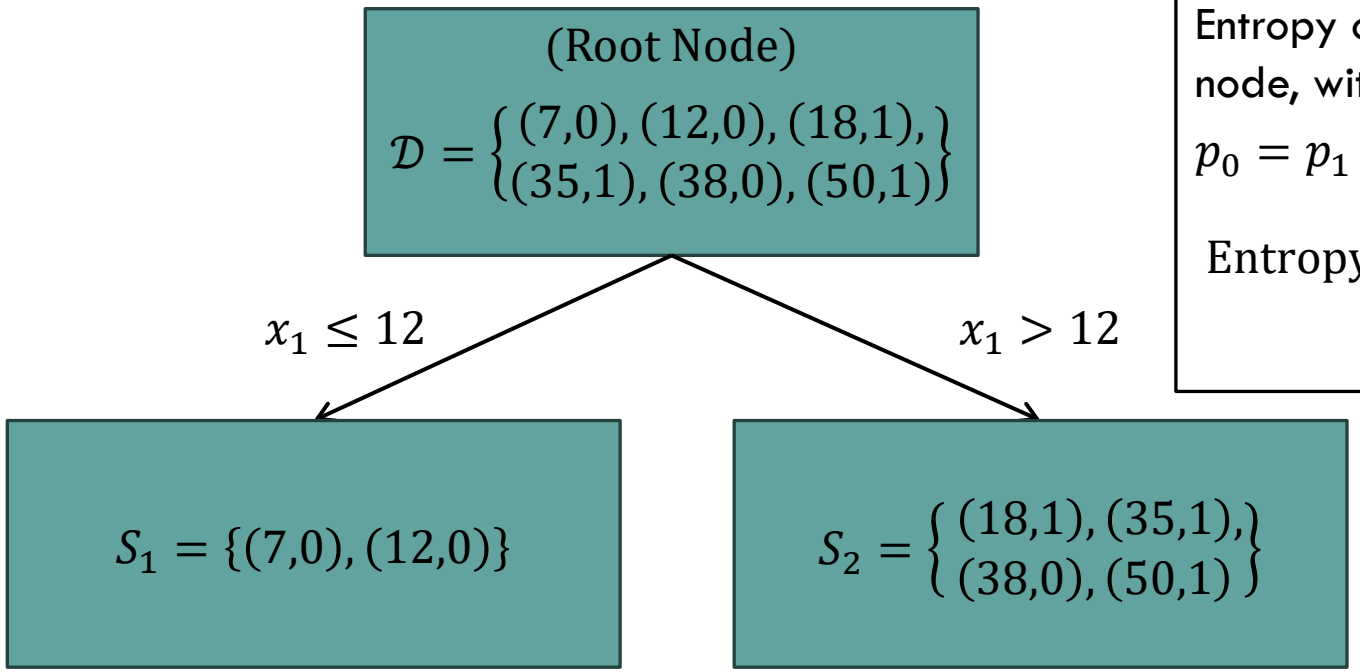
The entropy is a measure of the uncertainty or randomness in a dataset. It quantifies the impurity of a dataset (the level of disorder). If a node contains samples of only one class, then the entropy is 0, meaning the data is completely pure. Conversely, if a node has an entropy score of 1 (the largest value it can have), then the split of data generated by the node is completely impure and has the largest score. Given a dataset with K target classes, the entropy of a node N is defined as

$$\text{Entropy}(N) \coloneqq -\sum_{j=1}^{K} p_j \log_2 p_j \,,$$

Where p_j denotes the proportion of examples belonging to target class j.

Entropy is a metric that one would aim to **minimize** when training a decision tree.

Entropy Example



Entropy of original dataset $\mathcal{D}$. Total of 6 datapoints in this (root) node, with 2 target classes, and 3 examples of each class. Thus, $p_0 = p_1 = \frac{3}{6}$. The entropy for this node is:

$$\text{Entropy}(\mathcal{D}) = -\left(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}\right) = -\left(\frac{1}{2}(-1) + \frac{1}{2}(-1)\right) = 1.$$

Entropy of node N_1:

$$\text{Entropy}(N_1) = -\left(\frac{2}{2}\log_2\frac{2}{2} + \frac{0}{2}\log_2\frac{0}{2}\right) = 0.$$

Entropy of node N_2:

$$\text{Entropy}(N_2) = -\left(\frac{1}{4}\log_2\frac{1}{4} + \frac{3}{4}\log_2\frac{3}{4}\right) = \left(\frac{1}{4}(-2) + \frac{3}{4}(-0.415)\right) = 0.81125.$$

Information Gain: Measures of Impurity

Information Gain

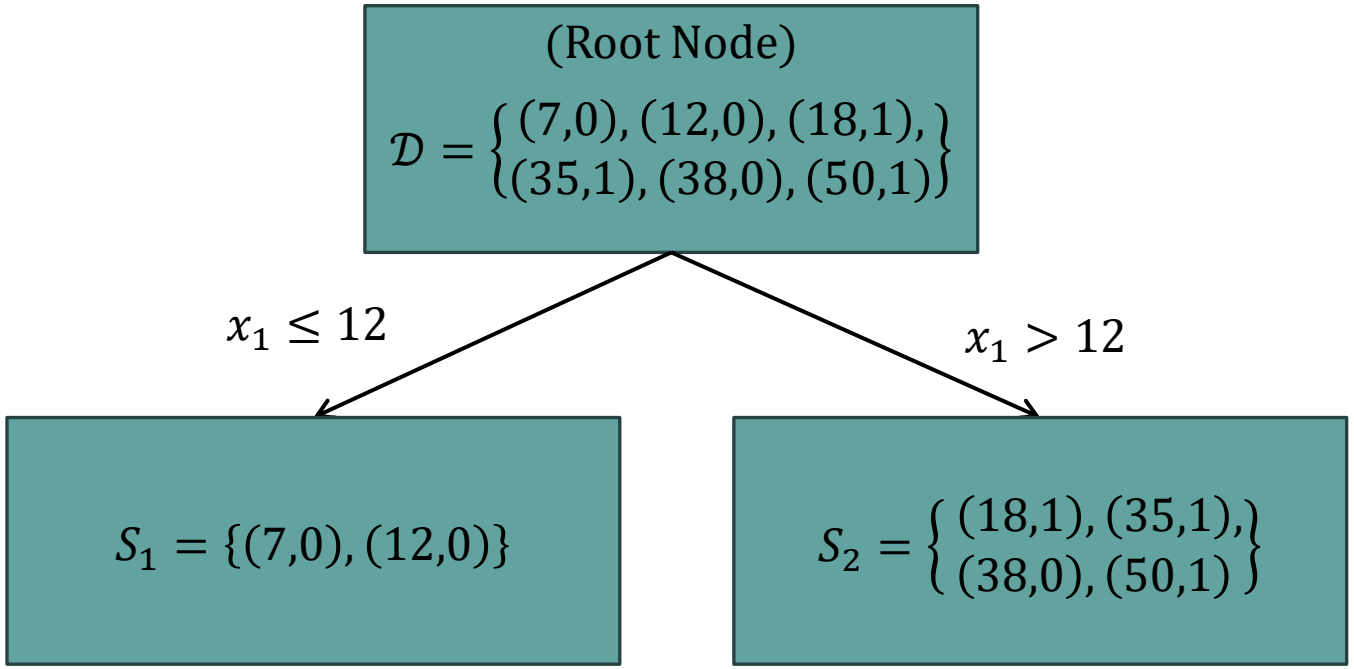
The information gain is defined as the reduction in entropy after a dataset has been split via a particular decision rule. This is another measure of how well a decision rule on a feature can split the dataset into different target classes. Given a dataset with K target classes, and letting $|\cdot|$ denote the cardinality of a set, the information gain of a node N is defined as

$$\text{Information Gain}(N, F; \theta) \coloneqq \text{Entropy}(N) - \sum_{\theta \in F} \frac{|N_\theta|}{|N|} \text{Entropy}(N_\theta),$$

Where F is the feature in the dataset that is being used to split the dataset, θ is the value that is used to split F into disjoint subsets, and N_θ is the subset of the datapoints defined by node N which satisfy the decision rule $\theta \in F$ (this would be the binary rule of $x_1 \leq 12$ in the examples we have seen thus far).

When training a decision tree, one would wish to **maximize** the information gain.

Information Gain Example



Recall from the entropy example that

$$\text{Entropy}(\mathcal{D}) = 1$$
$$\text{Entropy}(S_1) = 0$$
$$\text{Entropy}(S_2) = 0.81125$$

The information gain for this choice of decision rule is:

$$\text{Information Gain}(D, x_1; 12) := \text{Entropy}(\mathcal{D}) - \frac{|S_2|}{|\mathcal{D}|}\text{Entropy}(S_2) = 1 - \frac{4}{6}(0.81125) = 0.45917.$$

Gini Impurity: Measures of Impurity

Gini Impurity

The Gini impurity measures the probability that a randomly chosen datapoint within a subset will be incorrectly classified according to the proportions of the target classes present. Given a dataset with K target classes, the Gini impurity (also known as the **"Gini Index"**) of a node N is defined as

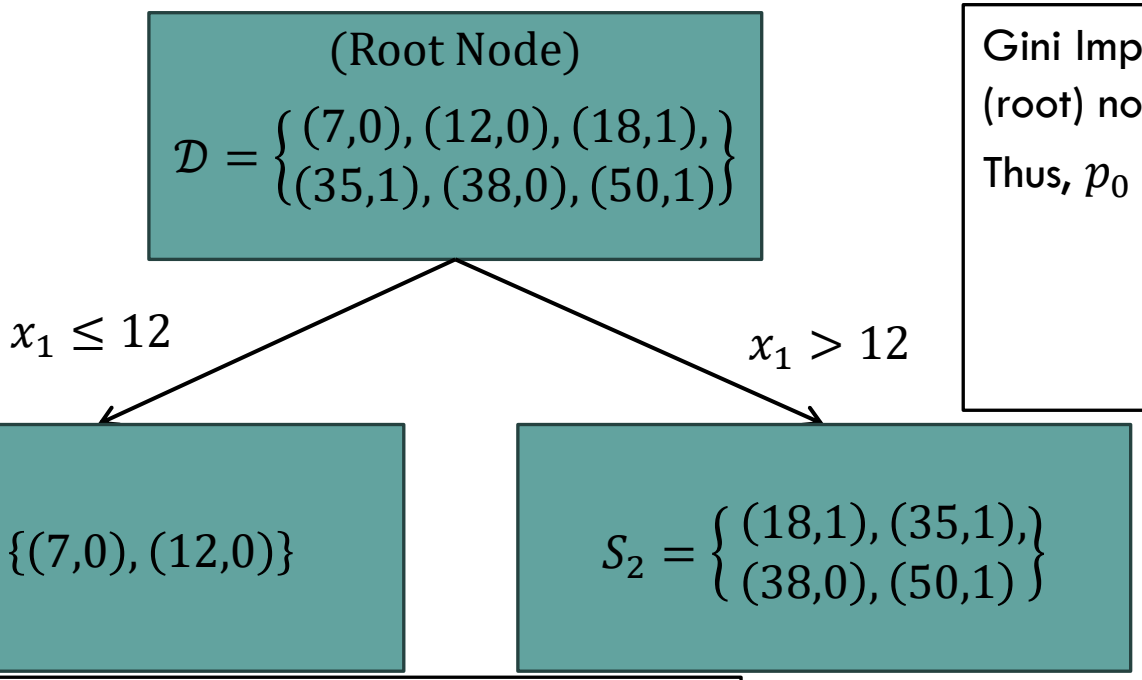
$$\text{Gini}(N) := 1 - \sum_{j=1}^{K} p_j^2 \,,$$

Where p_j denotes the proportion of examples belonging to target class j.

- It bears mentioning that Gini impurity is the most common measure of impurity that is used in practice and is the default metric in the Scikit-Learn implementations of decision trees, random forests, etc.

Gini impurity is a metric that one would aim to **minimize** when training a decision tree.

Gini Impurity Example



Gini Impurity of original dataset $\mathcal{D}$. Total of 6 datapoints in this (root) node, with 2 target classes, and 3 examples of each class. Thus, $p_0 = p_1 = \frac{3}{6}$. The Gini impurity for this node is:

$$\text{Gini}(\mathcal{D}) = 1 - \left(\left(\frac{3}{6}\right)^2 + \left(\frac{3}{6}\right)^2\right) = 0.5.$$

Gini Impurity of node N_1:

$$\text{Gini}(N_1) = 1 - \left(\left(\frac{2}{2}\right)^2 + \left(\frac{0}{2}\right)^2\right) = 0.$$

Gini Impurity of node N_2:

$$\text{Gini}(N_2) = 1 - \left(\left(\frac{1}{4}\right)^2 + \left(\frac{3}{4}\right)^2\right) = 0.375.$$

Performance Profiles of Different Impurity Metrics

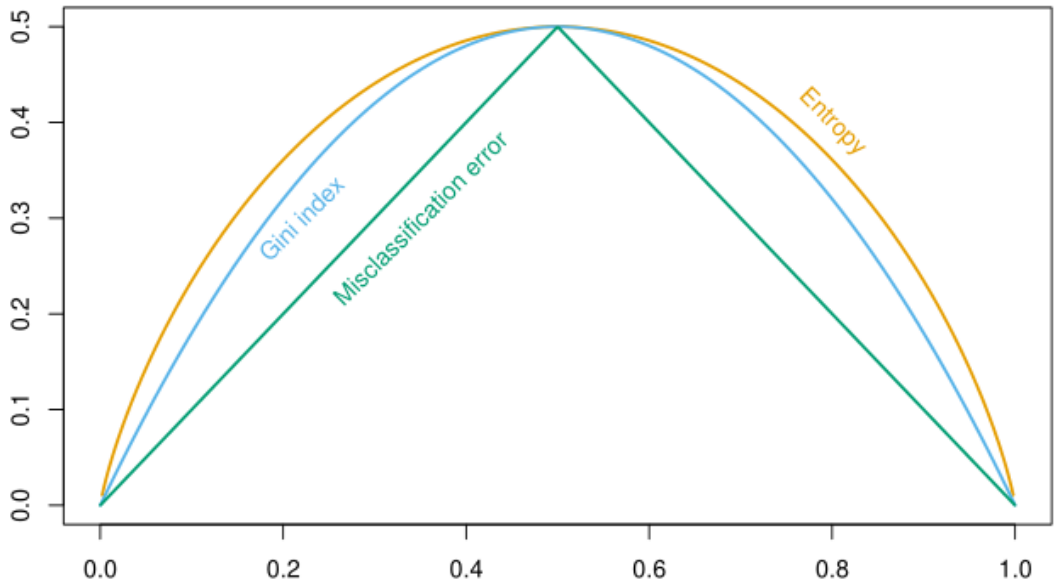
Although entropy, information gain, and the Gini impurity are both used to evaluate the splits in decision trees, they do not necessarily return the same tree structure when they are both used as decision criteria. However, any differences between the metrics are very minor; as such the most computationally efficient metric is typically used: **Gini Impurity**.

Entropy

Pros	Cons
More sensitive to class distribution; this could help classification on imbalanced datasets.	More computationally expensive since it requires the computation of logarithms

Gini Index

Pros	Cons
Faster to compute since it only requires sums and squares; typically yields compact and more efficient trees.	Could possibly create biased splits toward features with more unique values.



Proportion of Class 0 (for $y \in \{0,1\}$)

Regression Tree Metrics

Regression trees are the exact same in concept as classification trees, where the only difference is in the metrics that are used in evaluating the goodness of a decision split. Instead of the target variable y taking on a discrete class, we have that $y \in \mathbb{R}$.

Sum of Squared Errors

Given a **leaf node** N_k, for some $k \in \{0,1,2,\dots,K\}$, the sum of squared errors (SSE, otherwise known as the residual sum of squares (RSS)) of this leaf node is given by

$$SSE(N_k) = \sum_{i:\, x_i \in S_k} \left(y_i - \hat{y}_{S_k}\right)^2,$$

where $\hat{y}_{S_k} \in \mathbb{R}$ is the mean of the target values of the datapoints that fall into the region S_k, and the sum is over all the datapoints (x_i, y_i) that fall into the region S_k.

In greater detail, for an **internal node** N_k, since we are interested in minimizing the error of a decision rule $x_{j(k)} \leq \theta_k$ that results in two child nodes, we can define the SSE of this internal node as

$$SSE(N_k) = \sum_{i:\, x_i \in S_{k,\text{left}}} \left(y_i - \hat{y}_{S_{k,\text{left}}}\right)^2 + \sum_{i:\, x_i \in S_{k,\text{right}}} \left(y_i - \hat{y}_{S_{k,\text{right}}}\right)^2.$$

Total Loss of a Decision Tree

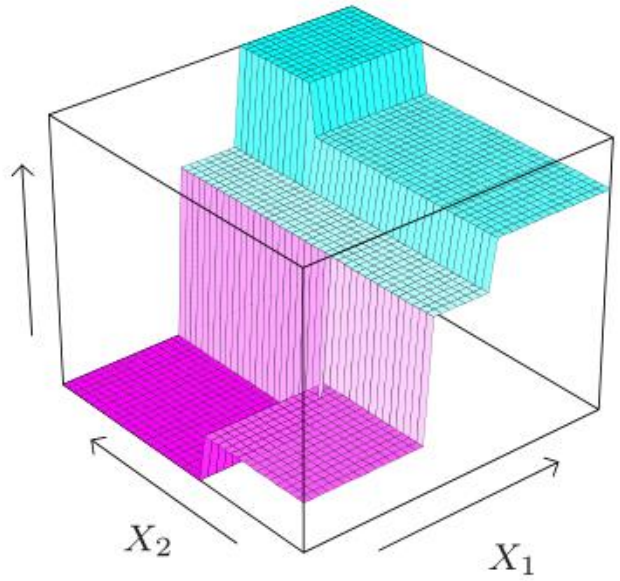
Total Loss (Decision Tree)

Given a dataset $\mathcal{D} = (X, y)$, where X is the design matrix and y is the target vector, and a decision tree $\mathcal{T} = \{N_0, N_1, \ldots, N_K\}$ with K nodes, the total loss of the decision tree model h_θ, parameterized by the decision rules θ, can be defined as

$$h_\theta(\mathcal{T}; X, y) \coloneqq \sum_{k=0}^{K} c_{N_k} \cdot \mathbb{I}[N_k \text{ is a leaf node}],$$

where c_{N_k} is the predicted class (or numeric value) returned by the leaf node N_k and $\mathbb{I}[\cdot]$ is the indicator function. Notice that c_{N_k} is formally defined as

$$c_{N_k} \coloneqq \begin{cases} \hat{y}_{S_k} = \dfrac{1}{|S_k|} \displaystyle\sum_{i:x_i \in S_k}^{|S_k|} y_i, & \text{for a regression tree,} \\ \hat{y}_{S_k} = \underset{i:x_i \in S_k}{\text{mode}}[y_i], & \text{for a classification tree.} \end{cases}$$



Notice that this loss function yields a non-smooth surface (recall the plot on the right)…

Decision Tree Hyperparameters

Although there are many different hyperparameters to choose before training a decision tree model (especially in SciKit-Learn's implementations), here are the some of the more important ones.

The Max-Depth

This hyperparameter defines the maximum depth of the tree that you are trying to train. Deeper trees can learn more complex patterns but also have a higher chance of overfitting to the dataset.

The Metric

This is the loss metric that you wish to minimize, which will yield slightly different tree structures (Gini Index, entropy, information gain, etc., for classification trees) (SSE or Absolute Error for regression trees).

The Min-Samples Per Split

This is the minimum number of samples that are required to be in a node's region before the training algorithm will generate another split from that node. Increasing this value will help prevent overfitting.

Min-Samples Per Leaf

This is the minimum number of samples required to be a leaf node. Again, increasing this value will help prevent overfitting.

Overview of Training a Decision Tree

- Now that we have discussed how we can measure how good a decision rule is for a particular node (via node impurity for classification trees or the SSR for regression trees), we now wish to **train** a decision tree to **minimize the overall impurity** (or residual errors) when given a set of hyperparameters.
- The algorithm that is used to train decision trees is "**greedy**" (it makes choices based on what the immediate "best" choice is for the current iterate instead of considering choices that may be slightly worse in the short-term, but which lead to better solutions overall).

Decision Tree Training Algorithm (Overview)

The training algorithm for a decision tree can be summarized in the following steps – for every node (starting with the root node):

- Check if a stopping criteria is satisfied: If so, do not branch (this node will be a leaf node).
- Determine the optimal split for that node $\mathcal{O}(mn)$ (This requires testing $m \cdot n$ (in the worst case, when each datapoint has a unique value) different decision rules to determine the split that yields the minimum loss; hence, you must iterate over all m datapoints for all n features).
- Create child nodes based on this optimal split.
- Apply this same process recursively for all nodes in the tree generated this way.

Recursive Node Splitting Algorithm

Algorithm Recursive Node Splitting

Input: The current node N which consists of some subset of datapoints $\mathcal{S} \subseteq \mathcal{D}$, some criteria L to measure the level of impurity of a split, and a set of stopping criteria $\mathcal{B}$.

Base Case. If node N meets a stopping criteria $b \in \mathcal{B}$, return N and do not proceed with splitting it further.

1) Feature Selection and Splitting.

1a) For every feature x_j, for all $j \in \{1, 2, ..., n\}$, compute the set of possible splits S_{x_j} corresponding to feature x_j.

1b) For every split $s \in S_{x_j}$, for all $j \in \{1, 2, ..., n\}$, partition N into two subsets $N_{left}(s) \subseteq N$ and $N_{right}(s) \subseteq N$ based on the splitting criteria.

1c) For every split $s \in S_{x_j}$, for all $j \in \{1, 2, ..., n\}$, compute the impurity of the split with the criteria L.

1d) Select the optimal split s^* that either maximizes the level of homogeneity (i.e., minimizes the level of impurity) (for classification trees) or minimizes some residual error such as MSE or SSE (for regression trees) of a split.

2) Create Child Nodes.

2a) Based on the optimal split s^* that was chosen, create two child nodes $N_{left}(s^*)$ and $N_{right}(s^*)$, both of which are determined by the criteria of s^*.

3) Recursion Step.

3a) Apply this Algorithm recursively to each child node $N_{left}(s^*)$ and $N_{right}(s^*)$.

End do

Training a Decision Tree

Algorithm Decision Tree

Input: The dataset $\mathcal{D}$, the root node N_0 (the root node will be $N_0 = \mathcal{D}$), and some set of stopping criteria $\mathcal{B}$.

1) Recursive Construction of the Tree. Call the Recursive Node Splitting Algorithm with the root node N_0 as input. This will build the entire tree $\mathcal{T}$ until a stopping criteria defined by $\mathcal{B}$ is reached.

2) Assign Class Labels to Leaf Nodes. For each leaf node N in the tree $\mathcal{T}$, assign to the node the class label that is the most common amongst the target classes in that node.

3) Return the decision tree $\mathcal{T}$.

Pruning: Preventing Overfitting

Decision Tree Pruning

Pruning a decision tree is exactly what is sounds like: it is a process by which a decision tree is altered by reducing the number of leaf nodes in the tree by getting rid of the decision rules that yielded those nodes. This has the effect of reducing the complexity of the tree and is an **essential step** in training a decision tree; if a decision tree has no stopping criteria other than to simply minimize the loss function, it will end up with a leaf node for every datapoint – clearly leading to an egregiously overfit model with no generalization capabilities to new data.

- **Pruning via Hyperparameters and Stopping Criteria:** This is the most straightforward way to prune a tree. This amounts to the designer specifying specific hyperparameter values (like the max-depth, min-samples-per-split, min-samples-per-leaf, etc.) which will prevent the tree from splitting past a certain point, ideally ensuring that there are enough datapoints in each leaf node.
- **Pruning via the Loss Function:** These methods incorporate an added term in the loss function itself that measures the number of nodes in the tree. This is very similar to regularization, in that more complex trees are weighted more in the loss function, which will help prevent the tree from being too complex. Perhaps the most common form of this is known as **Cost-Complexity Pruning**.

Why Binary Splitting?

We have only considered binary split decision rules for building trees at this point. However, a natural question may be:

"What about multi-way decision splits"?

The consensus on multi-way splitting is that, although it can sometimes be helpful in occasional cases, it is **not typically a good strategy**. The reason for this is because multi-way splitting generally fragments the dataset too quickly, which can lead to insufficient amounts of data at the lower levels.

- Further, since multi-way splits on a single variable can be achieved by implementing a series of binary splits, the latter strategy of simply using binary splits is preferred.

What About General Linear Combination Splits?

Most of the machine learning models we have seen up to this point have been based on hyperplanes in some form (either fitting a hyperplane to data, as in linear regression, or learning a hyperplane to separate target class as in logistic regression). Although we have been talking about decision trees in terms of decision rules of the form $x_{j(k)} \leq \theta_k$, a natural question would be

"What about general linear combination decision rules of the form $a^T x \leq \theta$"?

Although this can be done (and it can indeed improve predictive performance!), it comes at the expense of one of the most attractive benefits that decision trees offer: interpretability. One of the main reasons one would utilize a decision tree is for its intuitive nature and explainability. The moment one starts implementing linear combination rules, this explainability disappears.

- However, if one does want to incorporate these types of rules, one of the best ways is via Hierarchical Mixtures of Experts (HME) models (a type of ensemble model).

Other Closing Remarks

- Overall, decision trees are intuitive and simple models to understand and explain to large non-technical audiences (the tree structure itself, not the training process)

- Decision trees by themselves are most useful on relatively small datasets (small numbers of features) as their strength lies in being able to identify the most important rules that govern the structure of the prediction space.

- However, decision trees by themselves are not typically able to yield the best predictive power when compared to other machine learning algorithms (high bias). A method that aims to reduce this issue is known as **Boosting** (next lecture topic).

- Decision trees often struggle with overfitting (high variance). Often, small changes in the data can result in a very different tree structure, which makes interpretations somewhat precarious. A method that aims to reduce this issue is known as **Bagging** (next lecture topic).

References

- T. Hastie, R. Tibshirani, J. Friedman. "*The Elements of Statistical Learning*". Springer, 2009.