LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

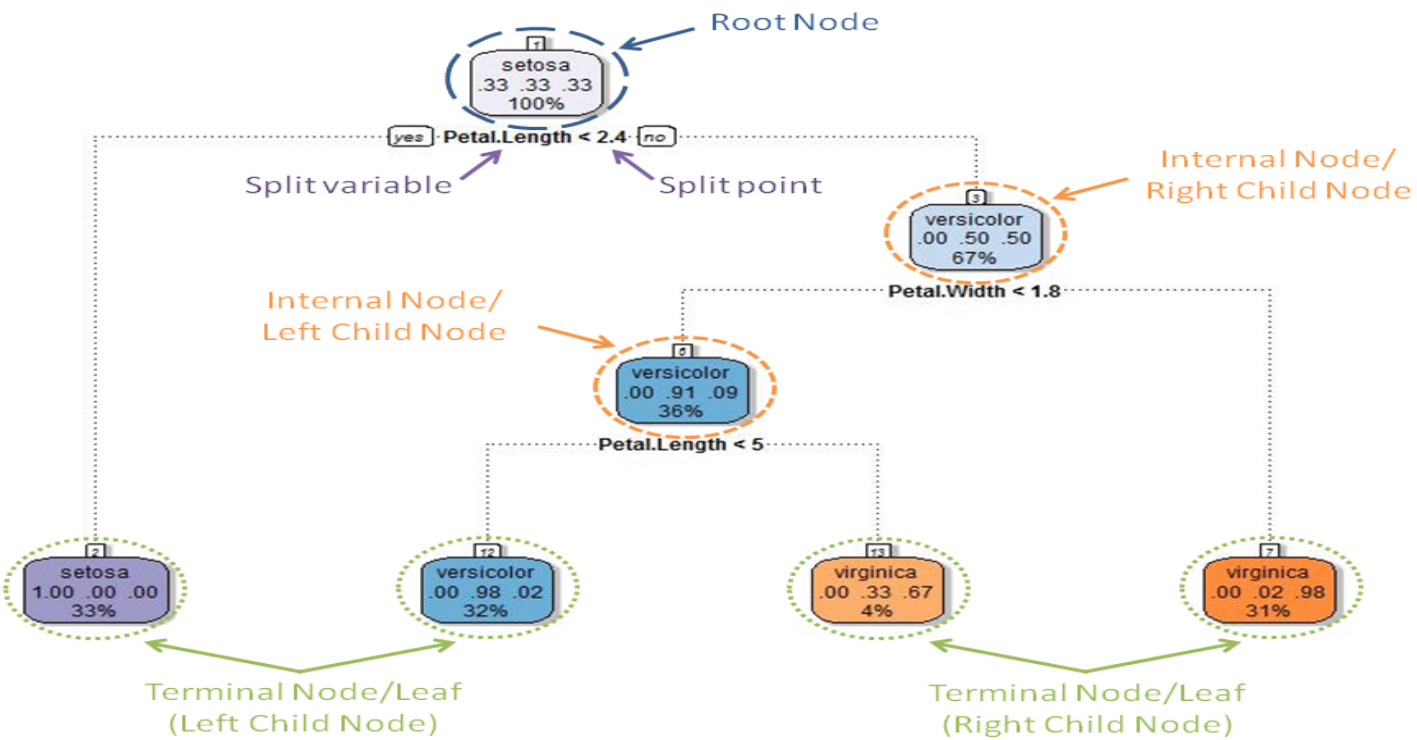
Introduction to Machine Learning

CART 1

Department of Statistics – LMU Munich

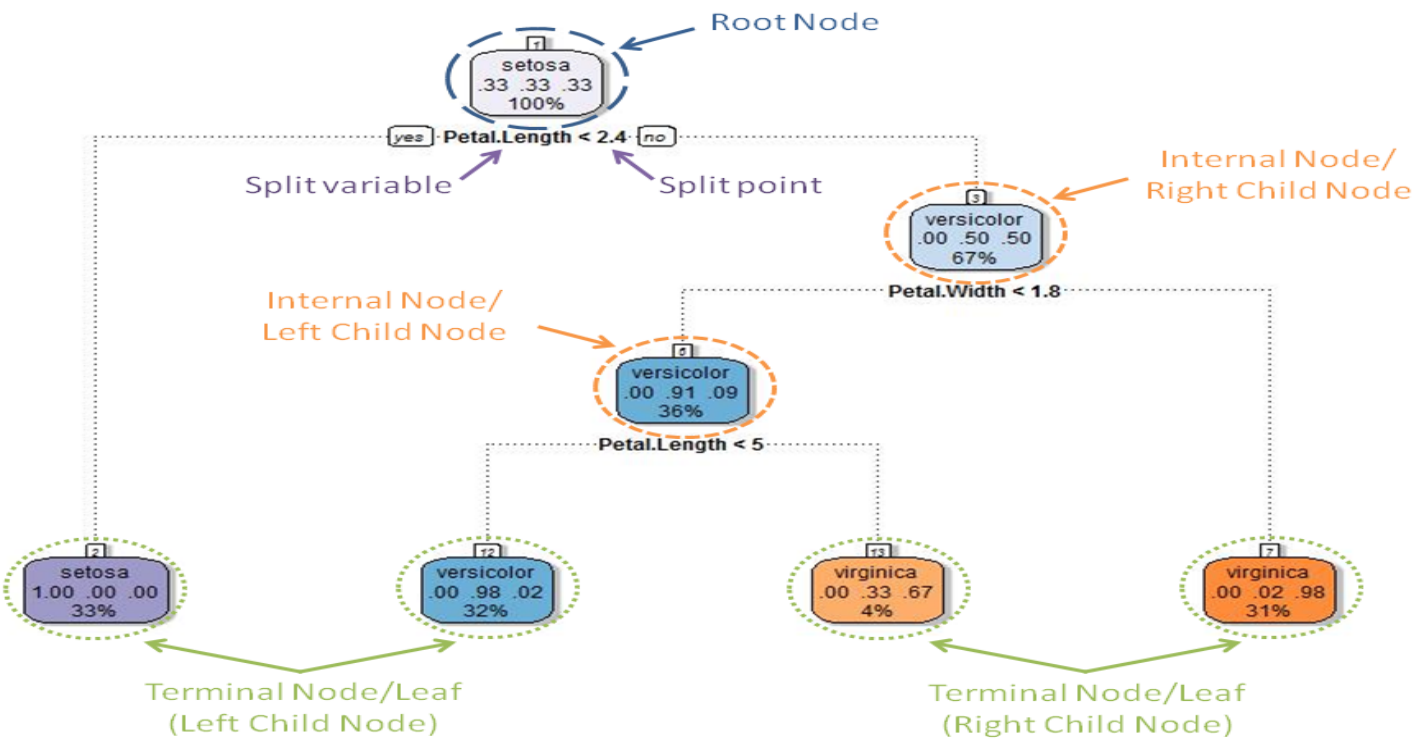
TREE MODEL AND PREDICTION

- Classification and Regression Trees, introduced by Breiman
- Binary splits are constructed top-down
- Only constant prediction in each leaf, either a numerical value, a class label or a probability vector



TREE MODEL AND PREDICTION

- During prediction, observations are passed down the tree, according to the splitting rules in each node
- An observation will end up in exactly one leaf node
- The constant label of that leaf node determines the prediction

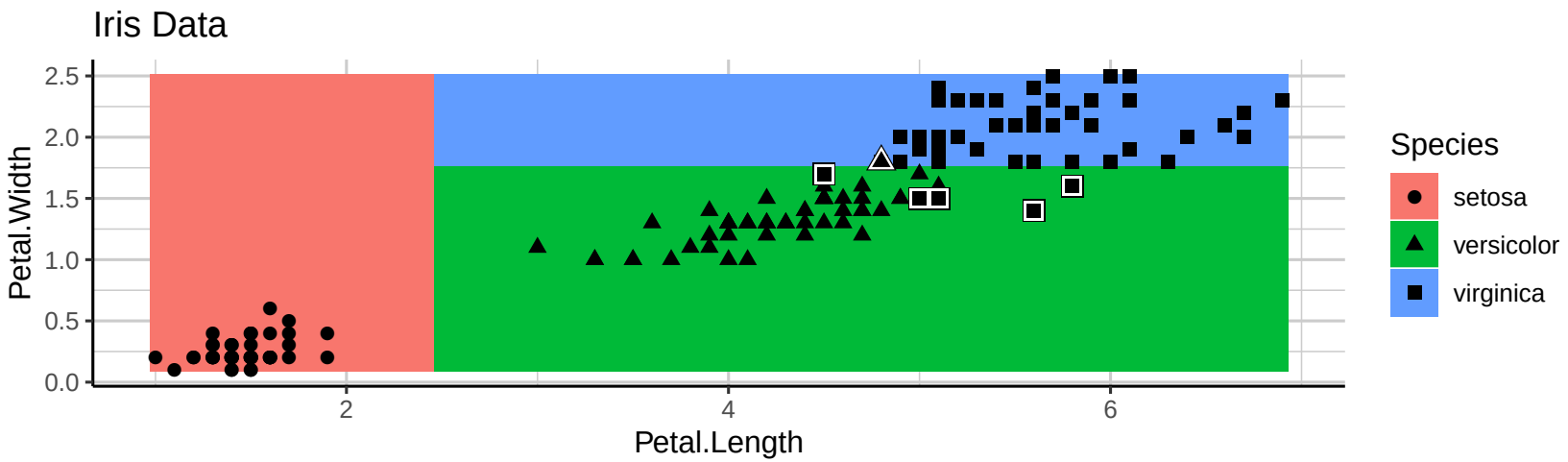


TRESS AS AN ADDITIVE MODEL

Each point in input space is assigned to one leaf node, and each leaf node has a set of input points leading to it, through axis-parallel splits. Hence, trees divide the feature space $\mathcal{X}$ into rectangles. We can write the tree as a additive model over the leaf-rectangles:

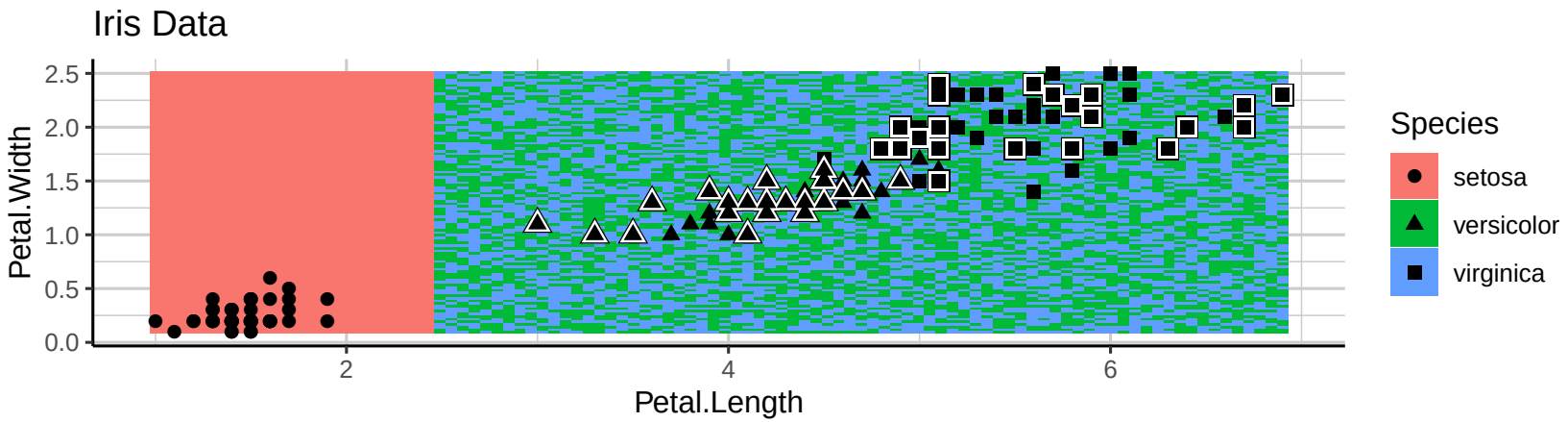
$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m \mathbb{I}(\mathbf{x} \in Q_m),$$

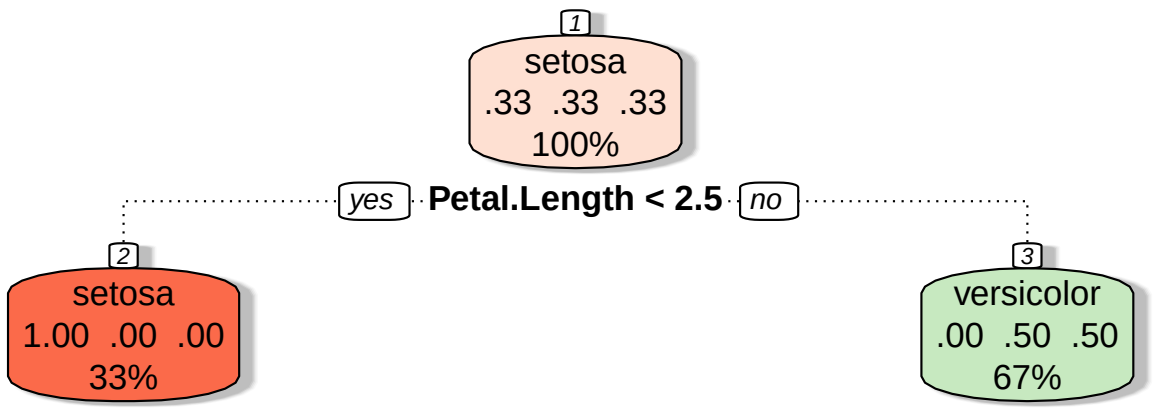
where M rectangles Q_m are used. c_m is a predicted numerical response, a class label or a class distribution.



TREE GROWING

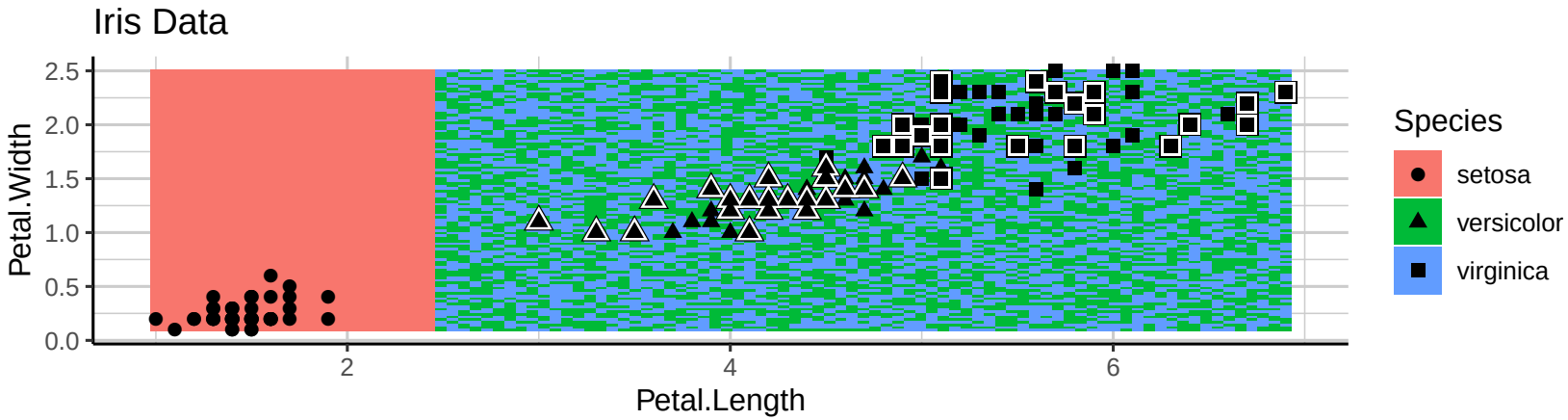
In the greedy top-down construction, features and split points are selected by exhaustive search. The training data is distributed to child nodes according to the splits - see below that nodes display the percentage of currently contained data.

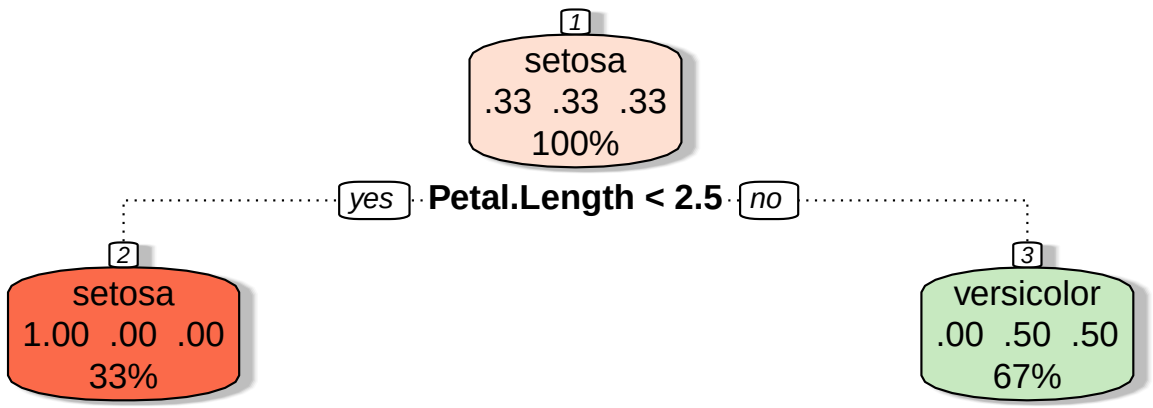




TREE GROWING

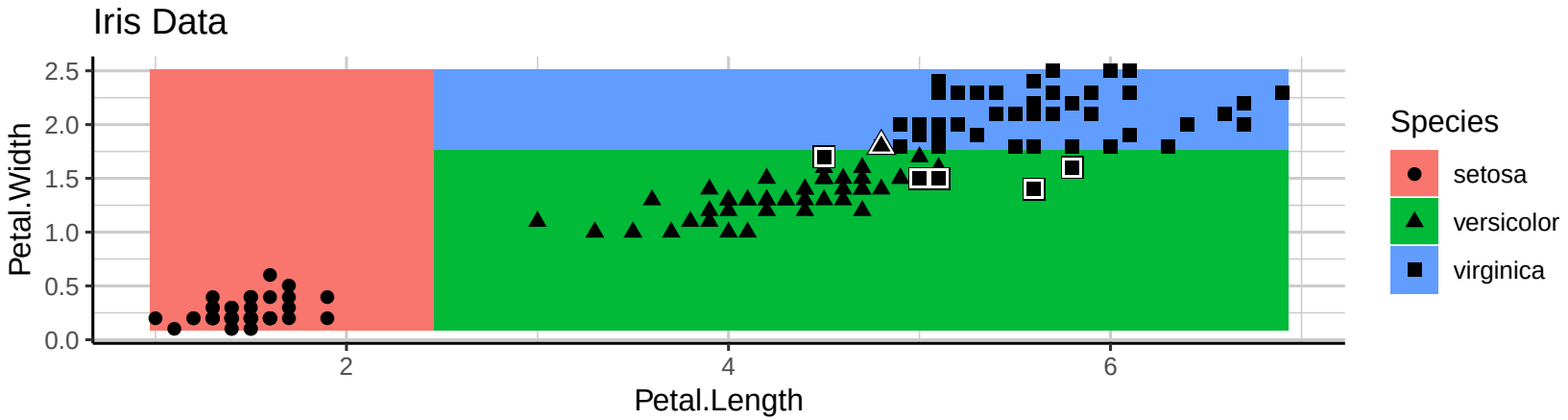
We start with an empty tree, a full root node of all data. We search for a feature and split-point, which makes the label distribution in the resulting left and right child (or sub-rectangles) most pure (defined later). See below that nodes display their current label distribution.

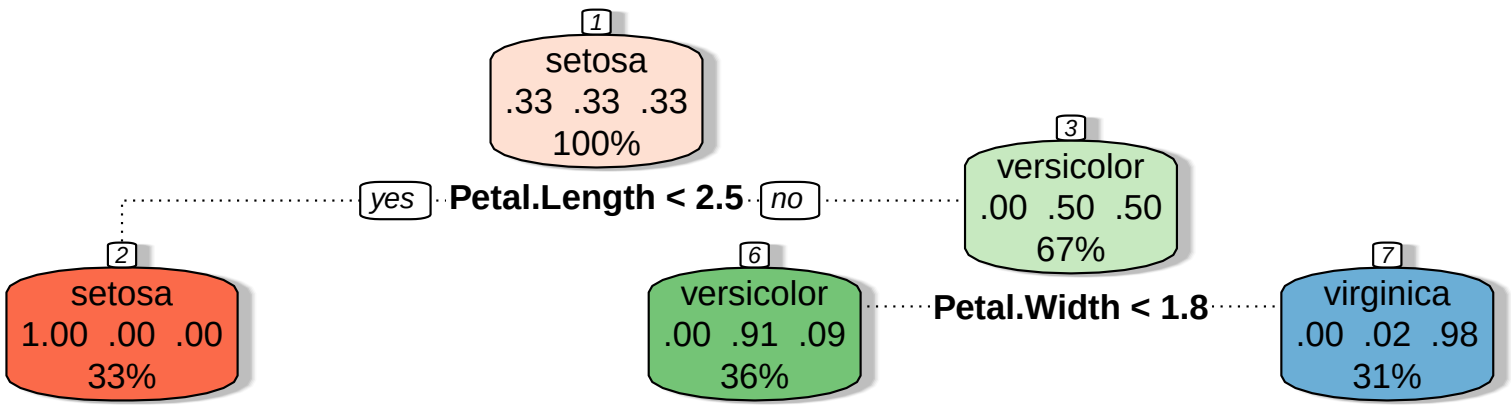




TREE GROWING

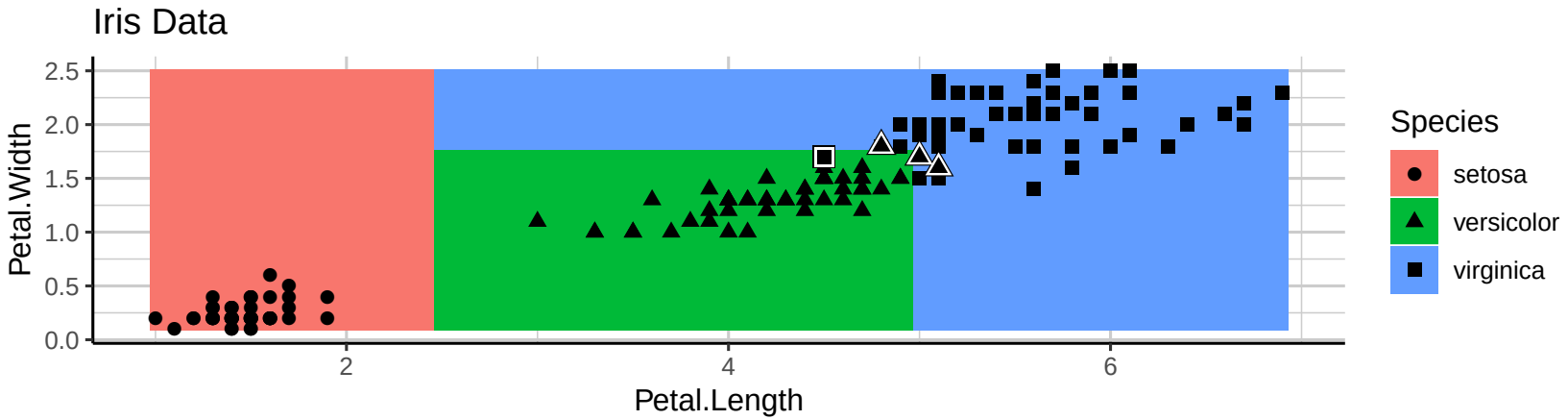
We proceed then recursively for each child node: Iterate over all features, and for each feature over all split points. Select the "best" split, divide training data from parent into left and right child.

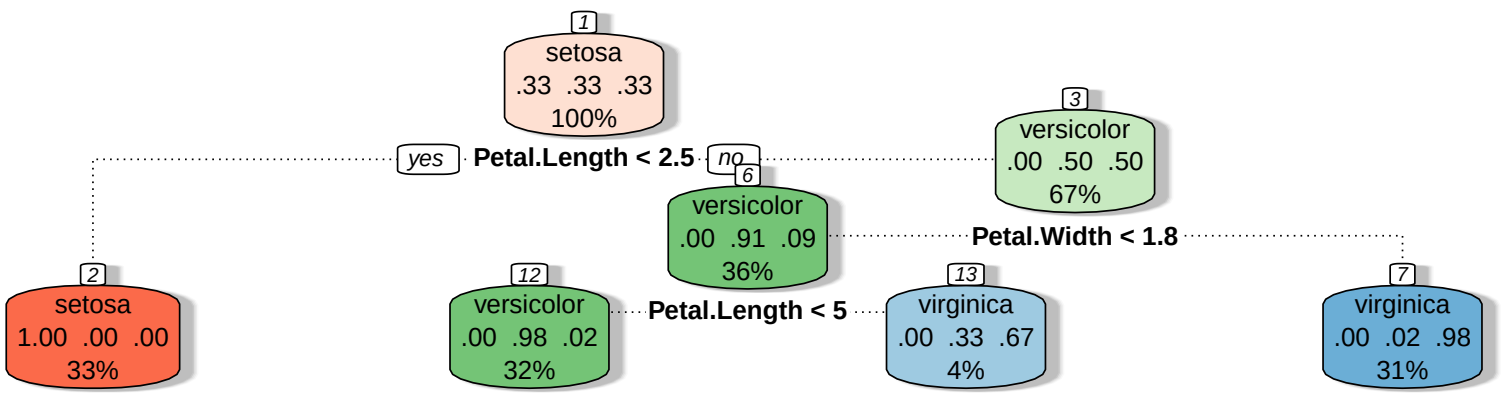




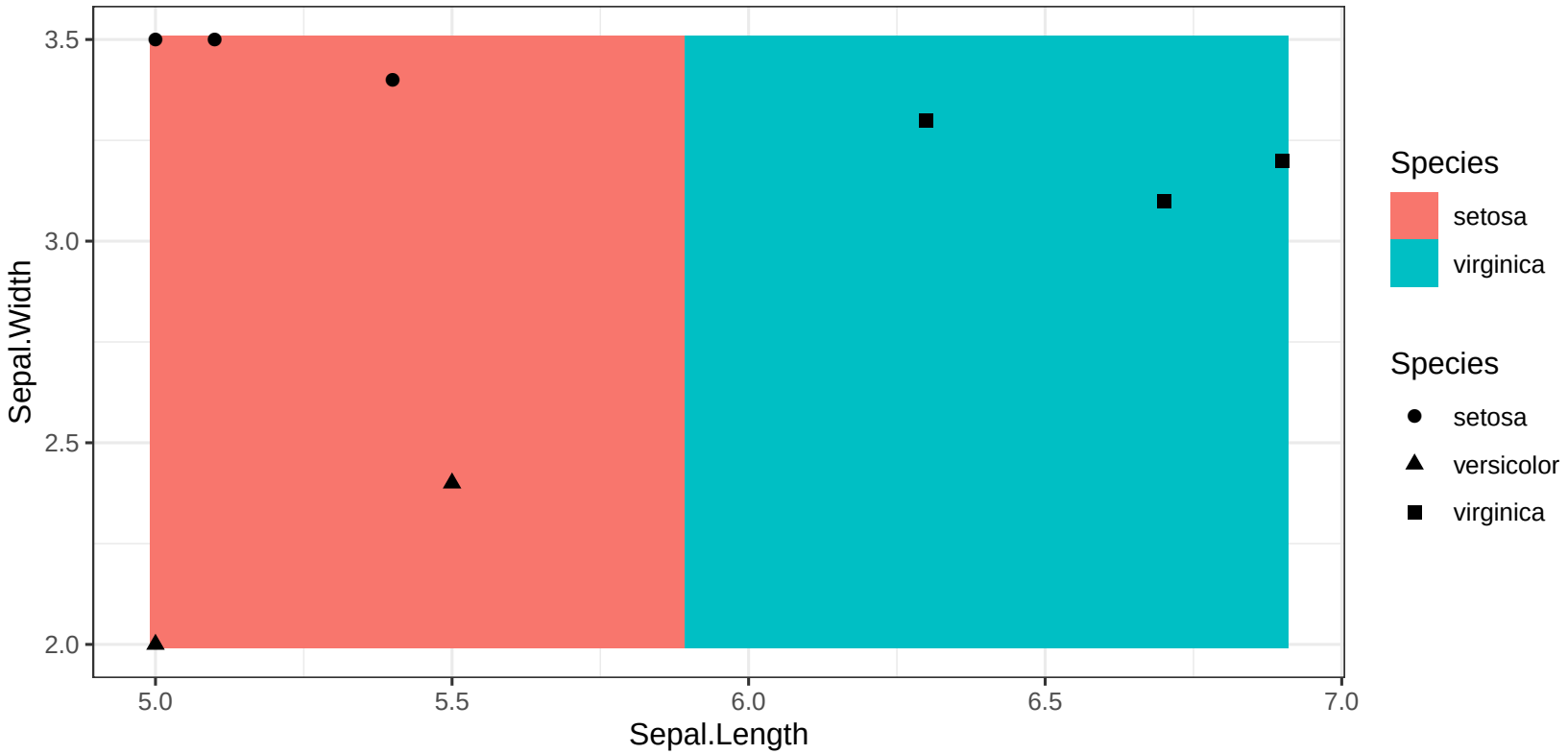
TREE GROWING

We proceed then recursively for each child node: Iterate over all features, and for each feature over all split points. Select the "best" split, divide training data from parent into left and right child.





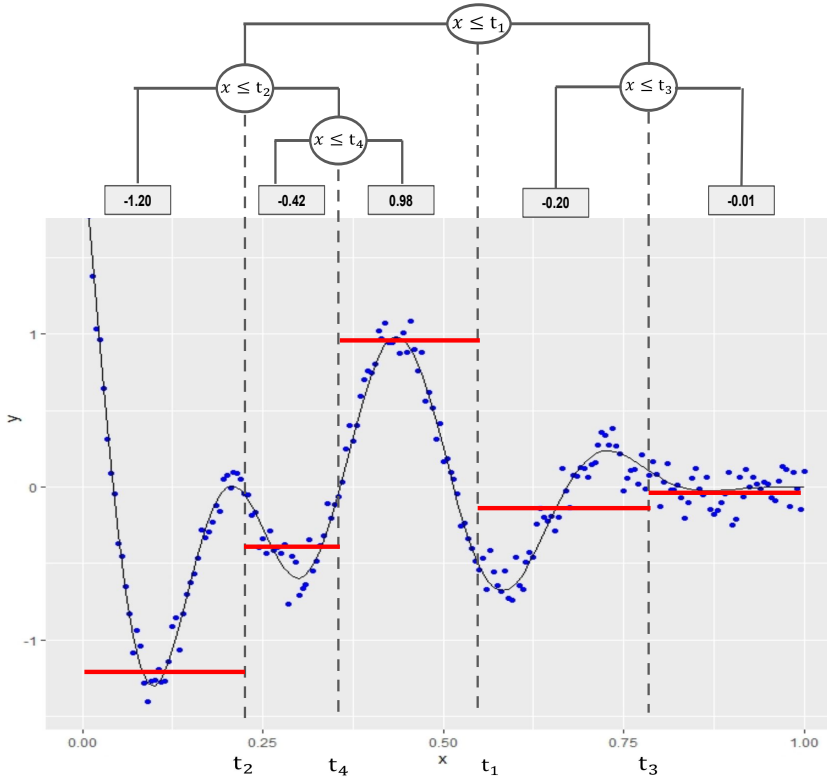
SPLIT PLACEMENT



Splits are usually placed "in the middle" between the observations they split, so the large margin to the next observations ensures better generalization.

REGRESSION EXAMPLE

x	y
0.078	-1.136
0.237	-0.136
0.327	-0.445
0.571	-0.660
0.673	0.012
0.806	0.046
0.942	-0.002



Data points (red) were generated from the underlying function (black):
$sin(4x - 4) * (2x - 2)^2 * sin(20x - 4)$

STOPPING CRITERIA

At some point we have to stop the recursive splitting of child nodes and produce a leaf. Multiple simple criteria for stopping can be defined:

- Minimal number of observations per node, for a split to be tried
- Minimal number of observations that must be contained in a leaf
- Minimal increase in label distribution purity that must be reached for the best split
- Maximum number of levels for the tree

CONSTANT PREDICTIONS IN LEAFS

After growing, for each leaf, we know which training observations are assigned to it. We produce a constant optimal rule here, for the complete associated rectangle Q_m

- Regression: We usually use the average value of all $y^{(i)}$ contained in the leaf. This is the optimal constant prediction under squared loss. We could also fit the median, which would be optimal under L1 loss.
- Classification: We fit the most common label of the data contained in that node. In order to estimate probabilities, we simply count class proportions for the training data contained in that leaf.

SPLITTING CRITERIA

Let $\mathcal{N} \subseteq \mathcal{D}$ be the data of a parent node with two child nodes $\mathcal{N}_1$ and $\mathcal{N}_2$, which are created by a split w.r.t. feature x_j at split point t:

$$\mathcal{N}_1 = \{(x, y) \in \mathcal{N} : x_j \leq t\} \text{ and } \mathcal{N}_2 = \{(x, y) \in \mathcal{N} : x_j > t\}.$$

In order to quantify the (negative) quality of the considered split we compute the empirical risks of both child nodes and sum it up

$$\mathcal{R}(j, t) = \mathcal{R}(\mathcal{N}_1) + \mathcal{R}(\mathcal{N}_2)$$

The risk $\mathcal{R}(N)$ for a node is simply the summed loss for the data contained in that node under a selected loss function L

$$\mathcal{R}(\mathcal{N}) = \sum_{(x,y) \in \mathcal{N}} L(y, c),$$

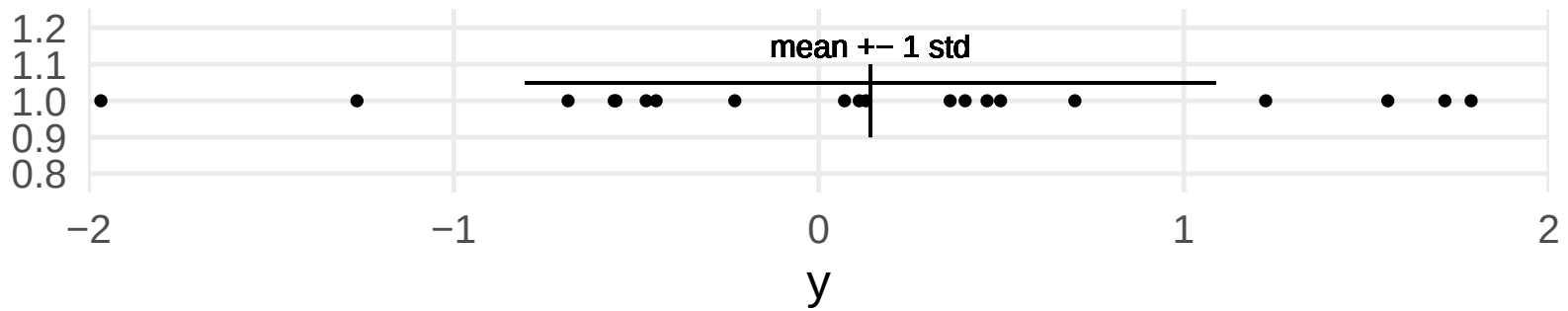
if we model the data in that node with an optimal constant $c = \arg\min_c \mathcal{R}(\mathcal{N})$. This is basically pretending that after split (j, t) our two child nodes become leafs.

SPLIT CRITERIA: REGRESSION

- For regression, we usually use L_2 loss / the SSE-criterion

$$\mathcal{R}(\mathcal{N}) = \sum_{(x,y) \in \mathcal{N}} (y - c)^2$$

- The best constant under L_2 is the mean $c = \bar{y}_{\mathcal{N}} = \frac{1}{|\mathcal{N}|} \sum_{(x,y) \in \mathcal{N}} y$
- Up to a constant, we just computed the variance of the label distribution in $\mathcal{N}$; we can also interpret this as a way of measuring the impurity of the distribution / fluctuation around the constant.
- We could have also used the L_1 loss and the median

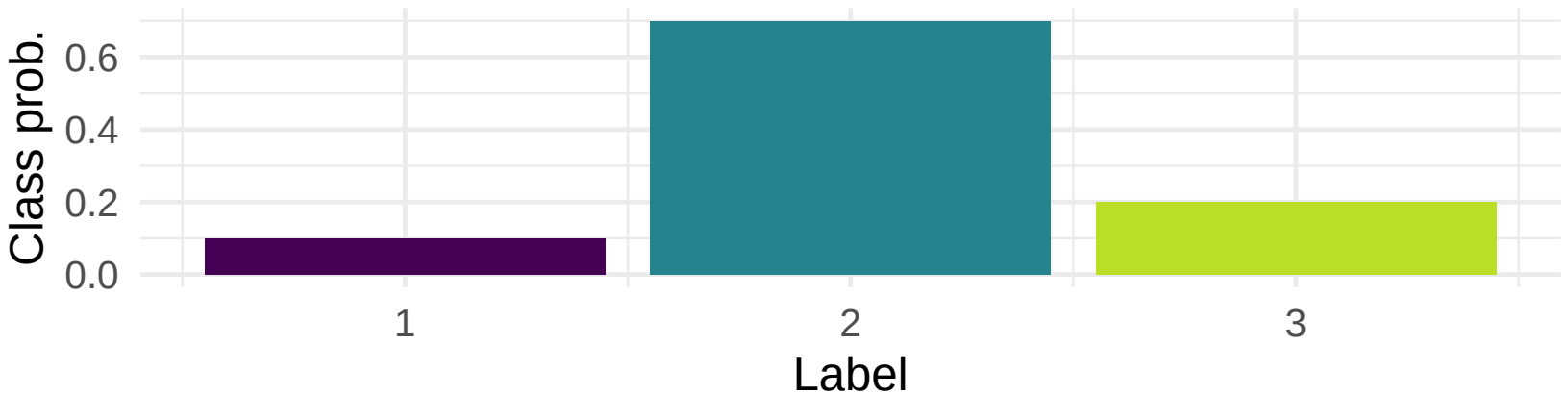


SPLITTING CRITERIA: CLASSIFICATION

- We normally use either Brier (so L_2 loss on probabilities) or the Bernoulli loss (from logistic regression) as loss function
- We usually model constant predictions in node $\mathcal{N}$ by simply calculating the class proportions

$$\pi_k^{(\mathcal{N})} = \frac{1}{|\mathcal{N}|} \sum_{(x,y) \in \mathcal{N}} [y = k]$$

This is the optimal constant under the 2 mentioned losses above



SPLITTING CRITERIA: COMMENTS

- Tree splitting is usually introduced under the concept of "impurity reduction", but our approach above is simpler and more in line with empirical risk minimization and our previous concepts
- Splitting on Brier score is normally called splitting on Gini impurity

$$I(\mathcal{N}) = \sum_{k \neq k'} \pi_k^{(\mathcal{N})} \hat{\pi}_{\mathcal{N}k'} = \sum_{k=1}^{g} \pi_k^{(\mathcal{N})} (1 - \pi_k^{(\mathcal{N})})$$

- Splitting on Bernoulli loss is normally called splitting on entropy impurity

$$I(\mathcal{N}) = - \sum_{k=1}^{g} \pi_k^{(\mathcal{N})} \log \pi_k^{(\mathcal{N})}$$

- The pairs Brier score / Gini and Bernoulli / entropy are equivalent (which is not hard to prove, but will not be done here)

SPLITTING WITH MISCLASSIFICATION LOSS

- Why don't we simply split according to the misclassification loss? We could use the majority class in each child as "best constant" and count how many errors we make? Aren't we often interested in minimizing this error, but have to approximate it? We do not have to compute derivatives when we optimize the tree!
- Actually, that is possible, but Brier score and Bernoulli loss are more sensitive to changes in the node probabilities, and therefore often preferred

SPLITTING WITH MISCLASSIFICATION LOSS

Example: two-class problem with 400 obs in each class and two possible splits:

Split 1:

	class 0	class 1
Left node	300	100
Right node	100	300

Split 2:

	class 0	class 1
Left node	400	200
Right node	0	200

- Both splits misclassify 200 observations
- Split 2 produces a pure node and is probably preferable.
- Brier and Bernoulli loss (slightly) prefer the 2nd split
- Calculation for Brier:
  $Split1 : 300(0-\frac{1}{4})^2+100(1-\frac{1}{4})^2+100(0-\frac{3}{4})^2+300(1-\frac{3}{4})^2 = 150$
  $Split2 : 400(0-\frac{1}{3})^2+200(1-\frac{1}{3})^2 = 133.3$