Decision Trees

Lecture series „Machine Learning“

Niels Landwehr

Research Group „Data Science“
Institute of Computer Science
University of Hildesheim

Agenda for Lecture

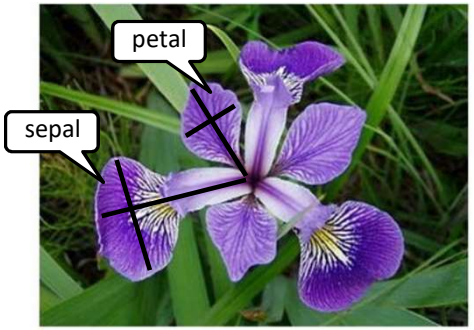
- Introduction to decision trees
- Learning decision trees

Agenda for Lecture

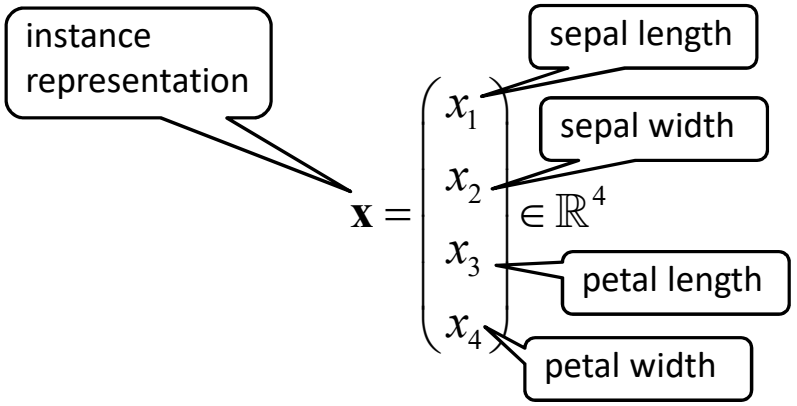
- Introduction to decision trees
- Learning decision trees

Recap: Iris Classification Data Set

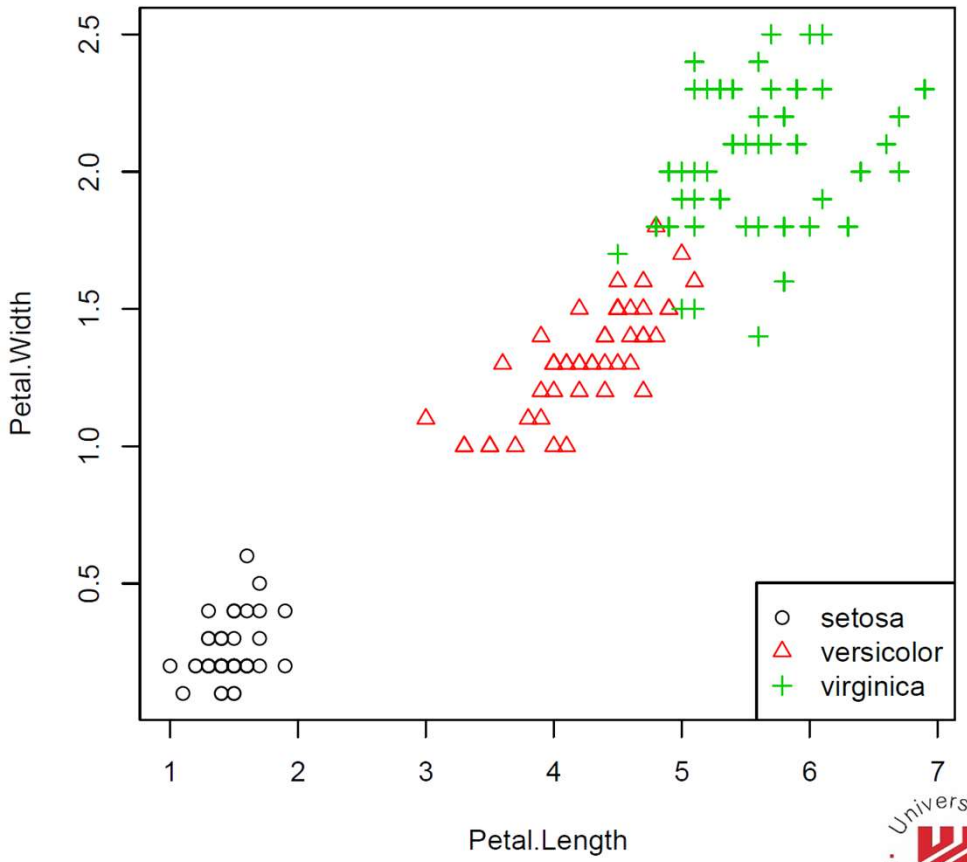
- Review: Iris classification data set



„Sepal“ and „petal“ refer to different parts of the Iris flower



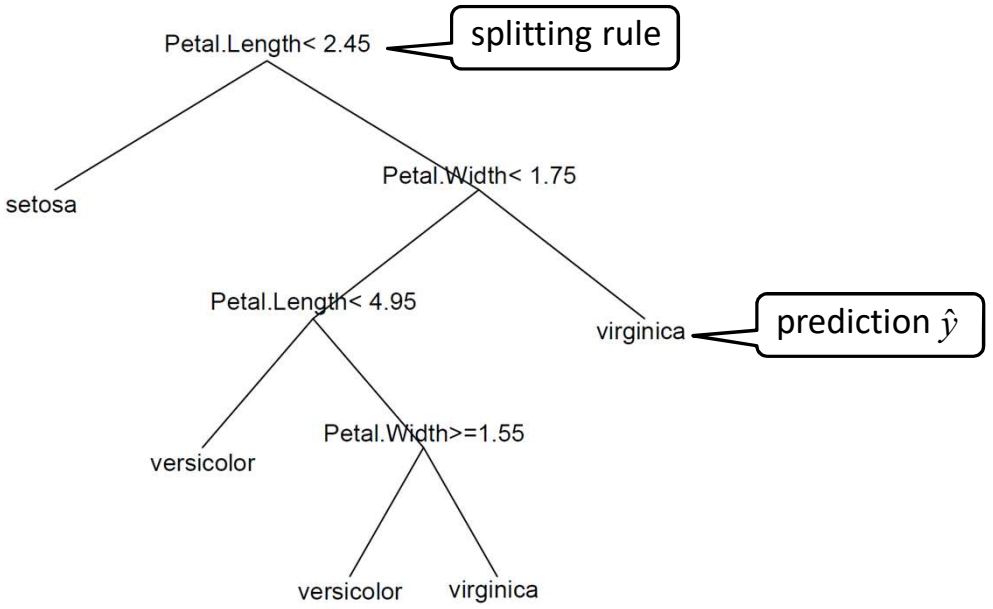
Scatter plot: petal width versus petal length
Classes color coded



Introduction: Decision Tree

- A **decision tree** is a tree-based model for classification that implements a function $f : \mathcal{X} \to \{1,...,T\}$, where T is the number of classes

- It consists of a tree that has
  - At each inner node: a so-called splitting rule that assigns an instance uniquely to one of the child nodes of the current node
  - At each leaf: a prediction, that is, a class label $\hat{y} \in \{1,...,T\}$



Example decision tree for the Iris data set, using the features petal length and petal width

Introduction: Decision Tree

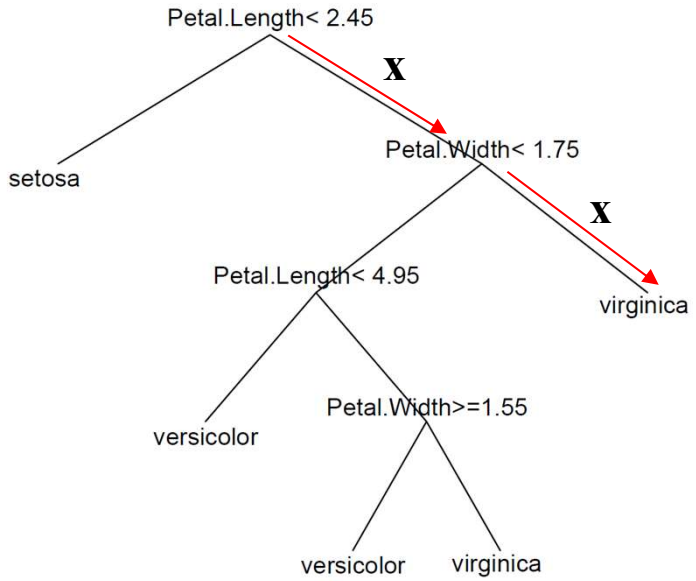
- The decision tree predicts a class for an instance $\mathbf{x} \in \mathcal{X}$ as follows:
  - Starting at the root node, at each interior node
    - evaluate the splitting rule for $\mathbf{x}$
    - branch to the child node picked by the splitting rule (default: left=„True“, right=„False“)
  - Once a leaf node is reached
    - return the class $\hat{y} \in \{1,...,T\}$ stored at the leaf as the prediction $f(\mathbf{x})$

Example:

$$\mathbf{x} = \begin{pmatrix} 8 \\ 2 \\ 6 \\ 1.8 \end{pmatrix} \in \mathbb{R}^4$$

petal length

petal width



$$f(\mathbf{x}) = \text{"virginica"}$$

Adapted from lecture „Decision Trees“ of ISMLL group, Prof. Schmidt-Thieme

Decision Tree as a Set of Rules

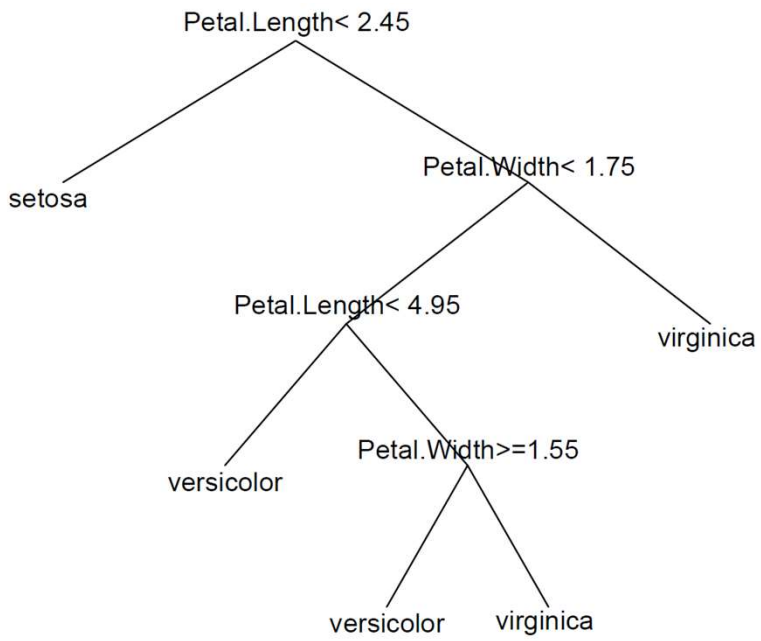
- Each path in a decision tree from the root node to a leaf can be formulated as a decision rule that predicts the class based on a conjunction of splitting rules:

$$\text{if } condition_1(\mathbf{x}) \wedge condition_2(\mathbf{x}) \wedge \ldots \wedge condition_k(\mathbf{x})$$
$$\text{then predict the class stored in the leaf node}$$

- A decision tree is equivalent to a set of such rules
- For any input $\mathbf{x}$, exactly one rule applies (that is, the conjunction of conditions is fulfilled)

Example: Decision Tree as a Set of Rules

- Example: rule set from decision tree



set of rules:

Petal.Length $< 2.45 \rightarrow$ class=setosa

Petal.Length ≥ 2.45 and Petal.Width < 1.75 and Petal.Length $< 4.95 \rightarrow$ class=versicolor

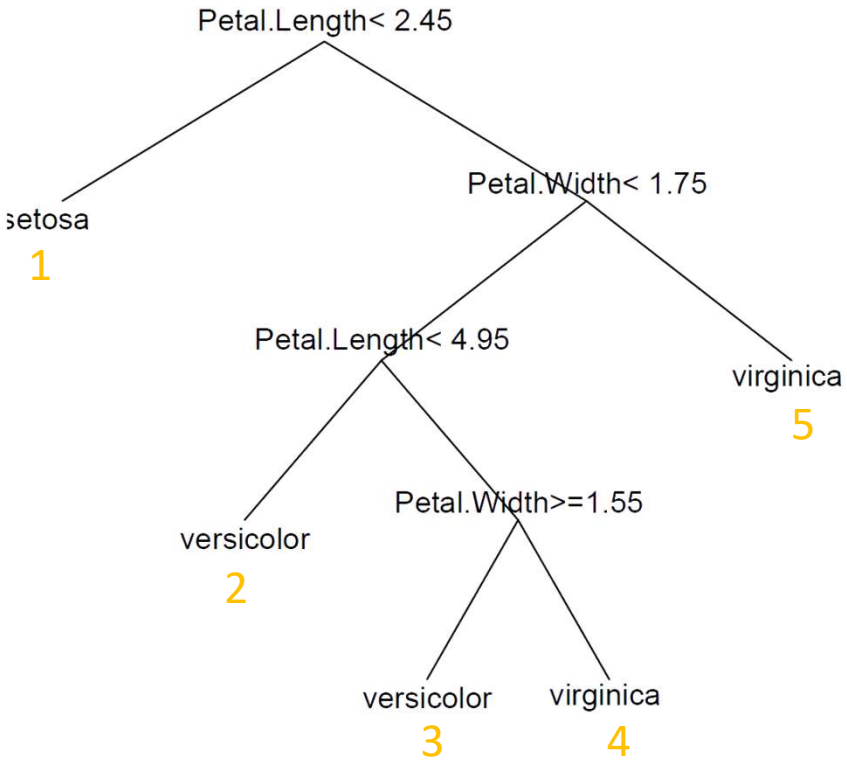
Petal.Length ≥ 2.45 and Petal.Width < 1.75 and Petal.Length ≥ 4.95 and Petal.Width ≥ 1.55 $\rightarrow$ class=versicolor

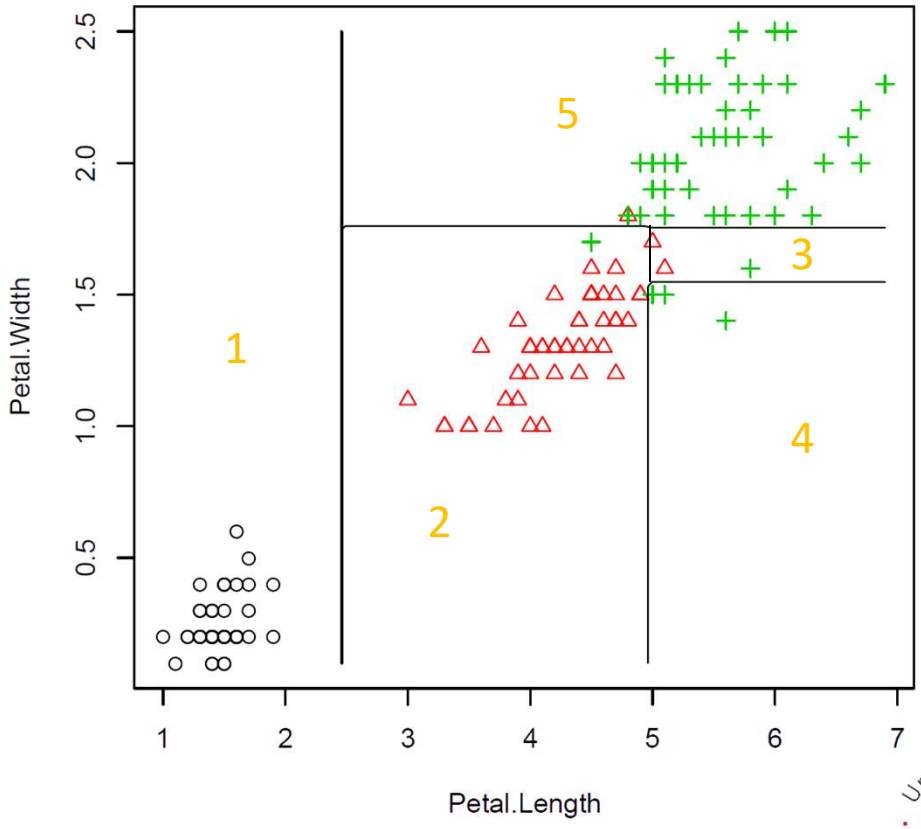
Petal.Length ≥ 2.45 and Petal.Width < 1.75 and Petal.Length ≥ 4.95 and Petal.Width < 1.55 $\rightarrow$ class=virginica

Petal.Length ≥ 2.45 and Petal.Width $\geq 1.75 \rightarrow$ class=virginica

Decision Boundaries are Rectangular

- Decision boundaries from splitting rules comparing the value of a feature to a constant are rectangular
- Leafs partition the space $\mathcal{X}$ into rectangular regions





Regression and Probability Trees

- It is straightforward to adapt decision trees to predicting continuous targets (regression problems) and to estimating class probabilities for classification

- A **regression tree** is a tree-based model for regression that implements a function $f : \mathcal{X} \to \mathbb{R}$

- It consists of a tree that has
  - At each inner node: a so-called splitting rule that assigns an instance uniquely to one of the child nodes of the current node
  - At each leaf: a prediction, that is, a target value $\hat{y} \in \mathbb{R}$

Regression and Probability Trees

- It is straightforward to adapt decision trees to predicting continuous targets (regression problems) and to estimating class probabilities for classification

- A **probability tree** is a probabilistic tree-based model for classification that implements a function $f : \mathcal{X} \to \mathbb{R}^T$. That is, it returns a vector of probabilities for the T classes

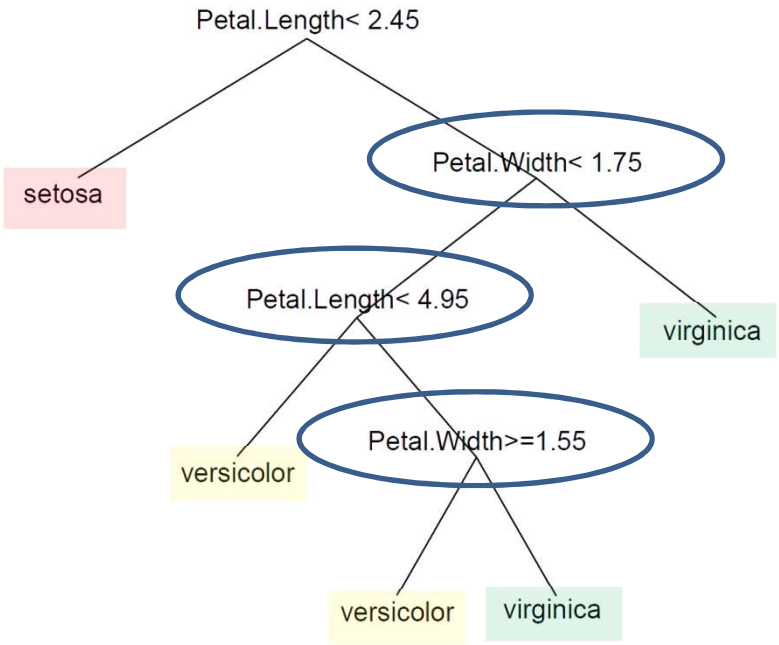
- It consists of a tree that has
  - At each inner node: a splitting rule that assigns an instance uniquely to one of the child nodes of the current node
  - At each leaf: a vector $\mathbf{p} \in \mathbb{R}^T$ of class probabilities

- The only difference between decision trees, regression trees, and probability trees is the predictions (or distributions) stored at the leaves, the tree structure is of the same form

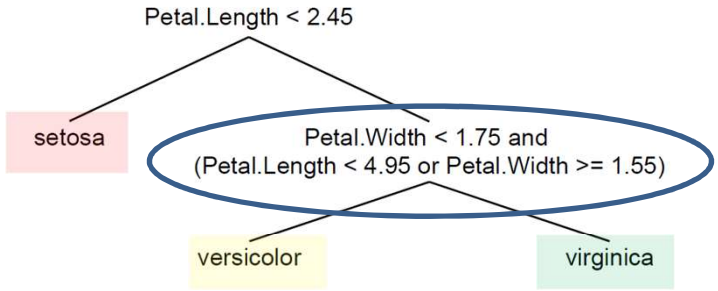
Simple Splits in Decision Trees

- Different splitting rules can be allowed at interior nodes of decision trees (or regression or probability trees)
- Generally, splits should be relatively simple, and more complex structures are captured by chaining several simple decisions in a tree structure:

Simple splits + tree structure (preferred)



More complicated split, less structure



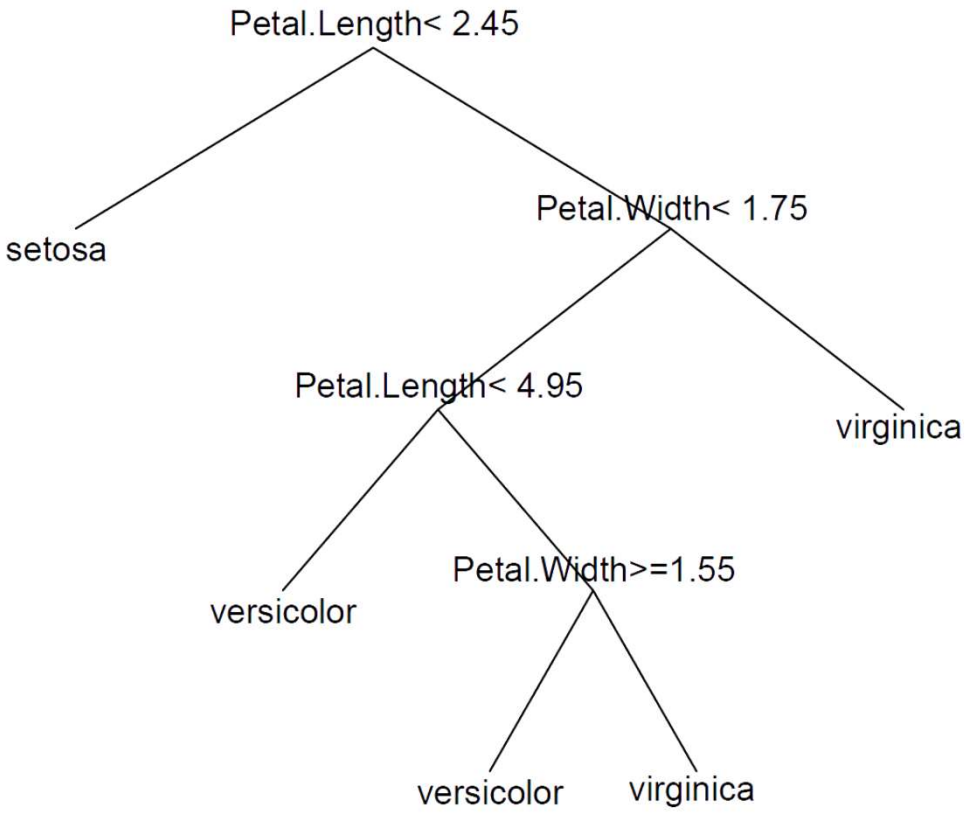
Types of Splits in Decision Trees

- Splits used in a decision tree are usually limited to a few simple types
- Usually assume that an instance $\mathbf{x} \in \mathcal{X}$ is represented as a vector of features, and most often a split only refers to a single variable (called „**univariate split**“)
- If the split results in two branches, it is called a **binary split**, otherwise an **n-ary split**

- Binary features: $x_i \in \{0,1\}$
  - only possible split is a binary split: left branch if $x_i = 1$, right branch if $x_i = 0$
- Nominal features $x_i \in \{c_1,...,c_K\}$
  - can use a so-called complete split: K outgoing branches, one per value c_k
  - can also use a binary split: left branch if $x_i \in \mathcal{C} \subset \{c_1,...,c_K\}$, right branch otherwise
  - or have *L* branches with subsets $\mathcal{C}_1,...,\mathcal{C}_L \subset \{c_1,...,c_K\}$, $\mathcal{C}_i \cap C_j = \varnothing$ and $\mathcal{C}_1 \cup ... \cup \mathcal{C}_L = \{c_1,...,c_K\}$
- Continuous features $x_i \in \mathbb{R}$
  - most often, a binary split of the form: left branch if $x_i < C$, right branch otherwise

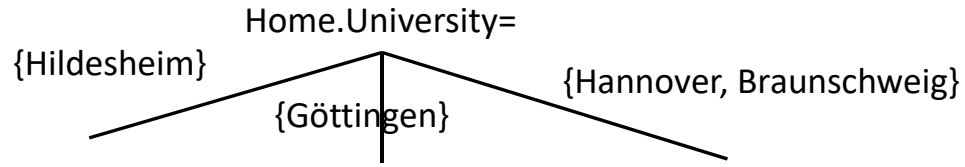
Example: Binary Splits

- Example: continuous attributes, binary splits

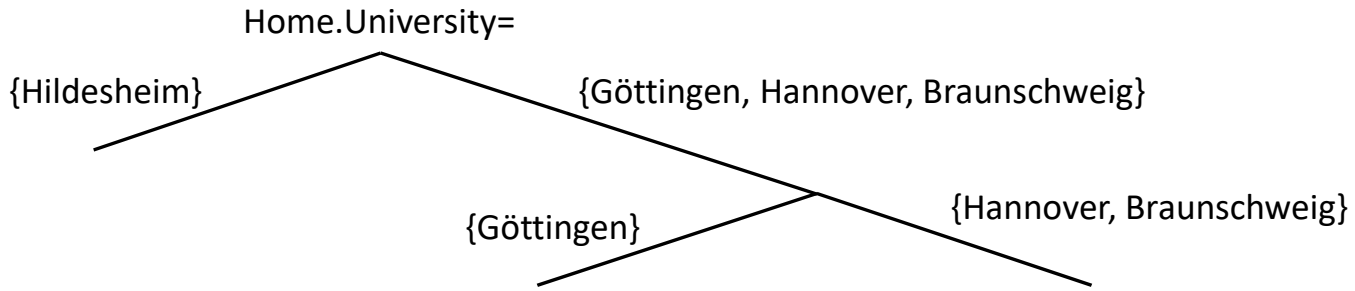


Example: Nominal Feature

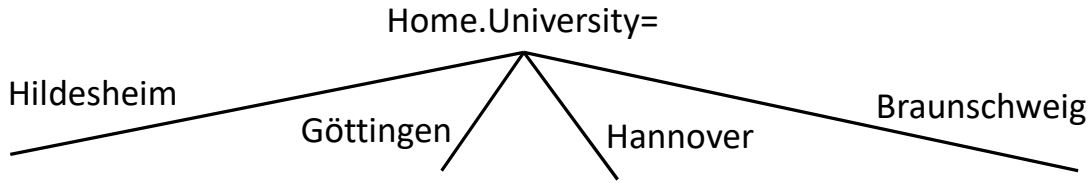
- Example: nominal feature, splitting on subsets, n-ary split



- Any n-ary split can be represented as a tree of binary splits, in this case e.g.



- A complete split on the feature would be



Types of Splits in Decision Trees

- Can also have „multivariate splits“ involving several variables
  - conjunctions or disjunctions of univariate splits (but maybe better use tree structure...)
  - more general splitting rules involving several variables, e.g. $\frac{x_i}{x_j} < 1$
- A tree is called
  - univariate
  - binary
  - n-ary
  - with complete splits

  if all splits in the tree have the corresponding property

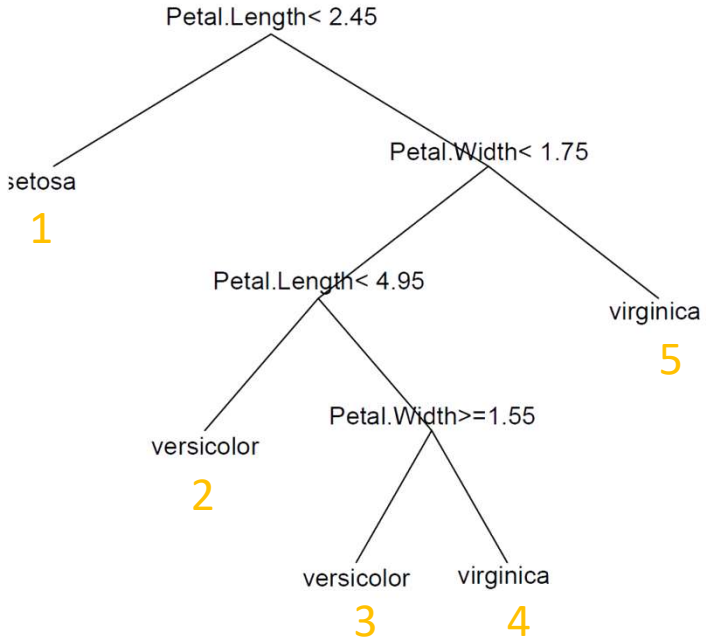
Agenda for Lecture

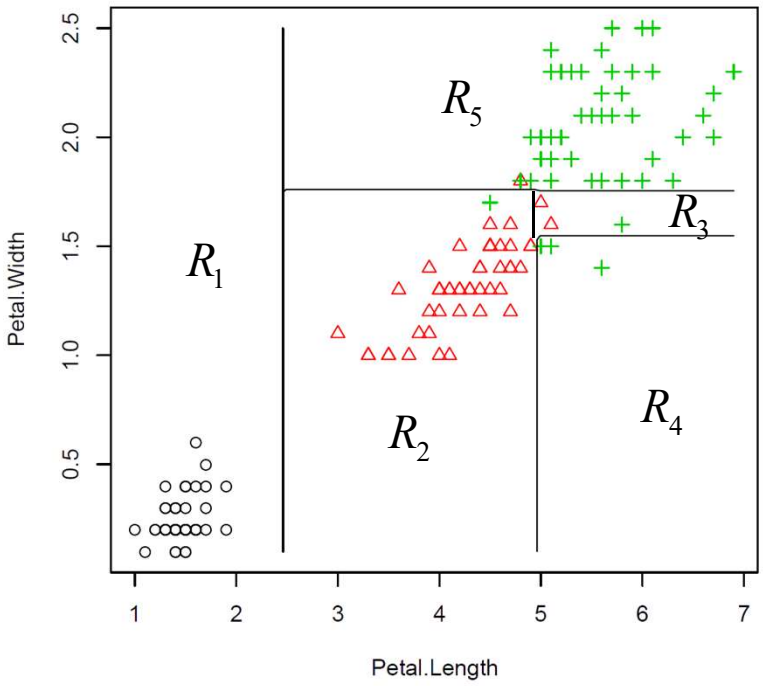
- Introduction to decision trees
- Learning decision trees

Class Predictions at Leaves For Given Tree Structure

- A decision tree (and equally a regression tree or probability tree) imposes a partition on the instance space $\mathcal{X}$ into cells $R_1,....,R_K$
- Each cell represents one leaf in the decision tree
- For continuous attributes and splits of the form $x_i < C$, this partition takes the form of rectangular cells:

Example: *K=5*





Class Predictions at Leaves For Given Tree Structure

- Assume that the tree structure is already given
- Thereby, the partitioning of the instance space $\mathcal{X}$ into cells $R_1,....,R_K$ is given

- To complete the decision tree model $f : \mathcal{X} \to \{1,...,T\}$, we have to decide which label $\hat{y}_k \in \{1,...,T\}$ to predict for each of the cells R_k
- Assume a training data set $\mathcal{D} = \{(\mathbf{x}_1, y_1),...,(\mathbf{x}_N, y_N)\}$ with $\mathbf{x}_n \in \mathcal{X}$ and $y_n \in \{1,...,T\}$
- Let's start from the criterion that we want to minimize the classification error

$$\min \sum_{n=1}^{N} I(y_n \neq f(\mathbf{x}_n))$$

- This error can be decomposed over the cells:

$$\sum_{n=1}^{N} I(y_n \neq f(\mathbf{x}_n)) = \sum_{k=1}^{K} \sum_{n=1}^{N} I(\mathbf{x}_n \in R_k) I(y_n \neq f(\mathbf{x}_n))$$

$$= \sum_{k=1}^{K} \sum_{n \in N_k} I(y_n \neq f(\mathbf{x}_n))$$

where $N_k \subseteq \{1,...,N\}$ is the set of indices of all instances in cell R_k

Class Predictions at Leaves For Given Tree Structure

- Because of the decomposition of the error, the criterion

$$\min \sum_{n=1}^{N} I(y_n \neq f(\mathbf{x}_n))$$

is satisfied when setting the prediction $\hat{y}_k$ for each cell R_k to the majority class of the training instances in that cell:

$$\hat{y}_k = \arg\max_y \sum_{n \in N_k} I(y_n = y)$$

Probability Estimates at Leaves For Given Tree Structure

- How do we estimate probability vectors $\mathbf{p} \in \mathbb{R}^T$ in a probability tree, given a fixed tree structure and thereby fixed partition of the instance space $\mathcal{X}$?

- For a probability tree, a natural objective is to maximize the likelihood:

$$\max \prod_{n=1}^{N} p(y = y_n \mid \mathbf{x}_n, f)$$

  where $p(y = y_n \mid \mathbf{x}_n, f)$ is given by the y_n - th entry in the probability vector $\mathbf{p} \in \mathbb{R}^T$ associated with the leaf the instance $\mathbf{x}_n$ is sorted into

- Taking the logarithm of the likelihood leads to the equivalent criterion

$$\max \sum_{n=1}^{N} \log p(y = y_n \mid \mathbf{x}_n, f)$$

Probability Estimates at Leaves For Given Tree Structure

- Again, this can be decomposed over the cells in the partition given by the tree:

$$\sum_{n=1}^{N} \log p(y = y_n \mid \mathbf{x}_n, f) = \sum_{k=1}^{K} \sum_{n \in N_k} \log p(y = y_n \mid \mathbf{x}_n, f)$$

$$= \sum_{k=1}^{K} \sum_{n \in N_k} \log p(y = y_n \mid \mathbf{x}_n \in R_k)$$

where $p(y = y_n \mid \mathbf{x}_n \in R_k)$ is given by the y_n-th element of the probability vector $\mathbf{p}_k \in \mathbb{R}^T$ associated with the leaf that represents cell R_k

- This likelihood is maximized by setting for each $k \in \{1,...,K\}$ the probability of each label to the relative frequency of that label in the training data in that cell:

$$p(y \mid \mathbf{x}_n \in R_k) = \frac{1}{|N_k|} \sum_{n \in N_k} I(y_n = y)$$

Continuous Target Estimates at Leaves For Given Tree Structure

- How do we estimate the continuous targets for the leaves in a regression tree, given a fixed tree structure and thereby fixed partition of the instance space $\mathcal{X}$?

- For a regression tree, a natural objective is to minimize the squared errors:

$$\min \sum_{n=1}^{N} (y_n - f(\mathbf{x}_n))^2$$

- As for the other cases, this can be decomposed into the cells by

$$\sum_{n=1}^{N} (y_n - f(\mathbf{x}_n))^2 = \sum_{k=1}^{K} \sum_{n \in N_k} (\hat{y}_k - y_n)^2$$

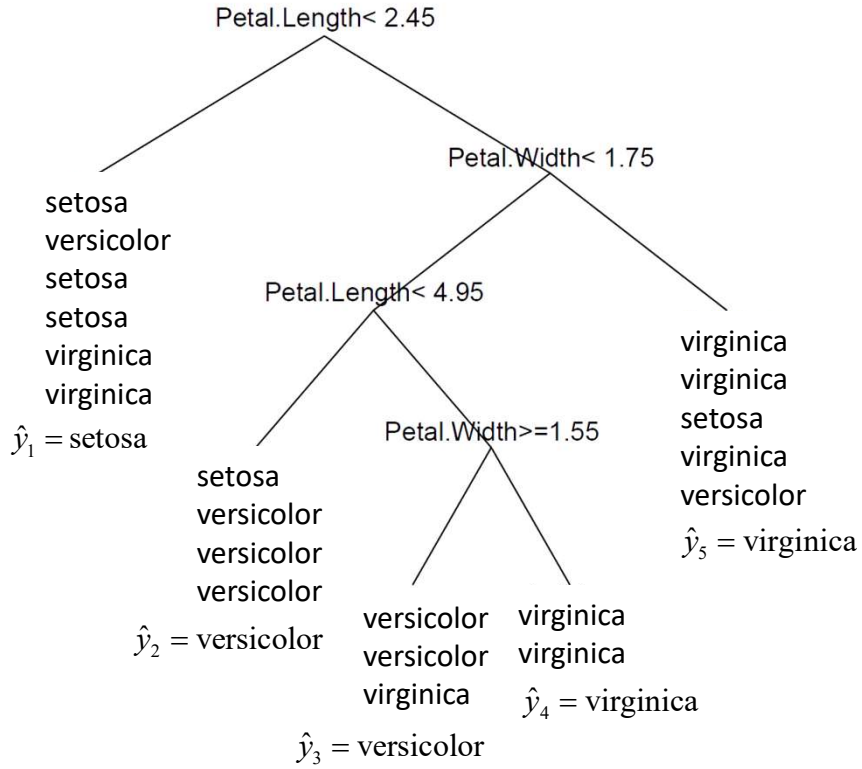
  where we use $\hat{y}_k$ to denote the continuous prediction stored at the leaf representing cell R_k

- This squared error can be minimized by setting the prediction to the average of the labels in the cell R_k:

$$\hat{y}_k = \frac{1}{|N_k|} \sum_{n \in N_k} y_n$$

Example: Predictions at Leaves for Classification

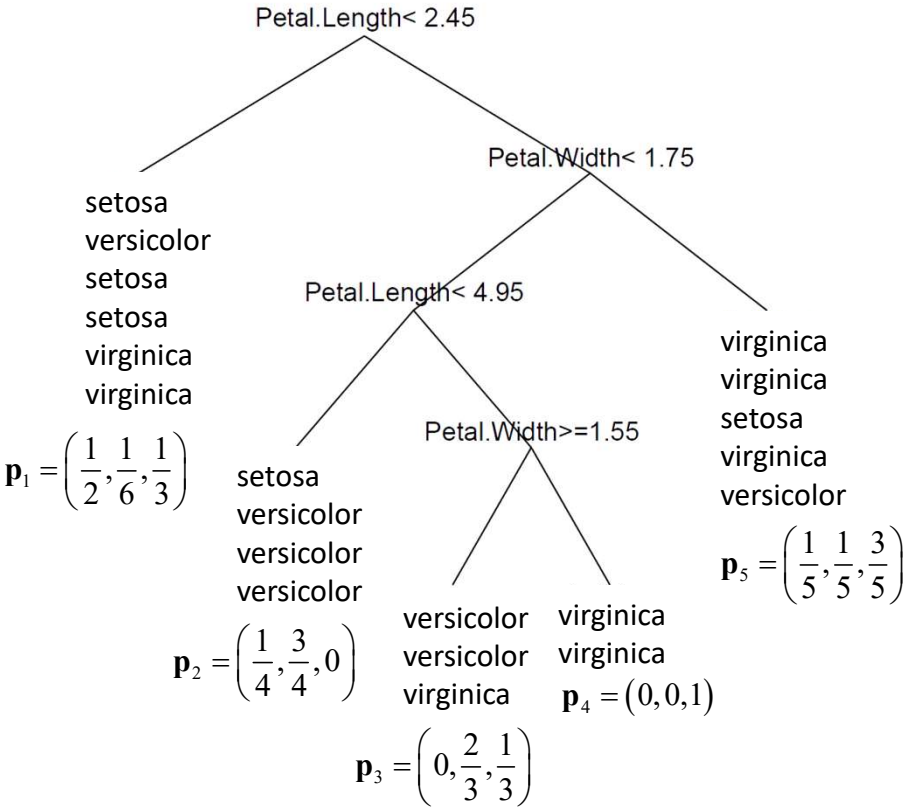
- Assume that the following decision tree structure is fixed, and there is a training data set of *N=20* instances. At each leaf we show the labels of the training instances sorted into this leaf



- Then the predictions $\hat{y}_k$ at the leaves are the majority labels as shown

Example: Predictions at Leaves for Probability Estimation

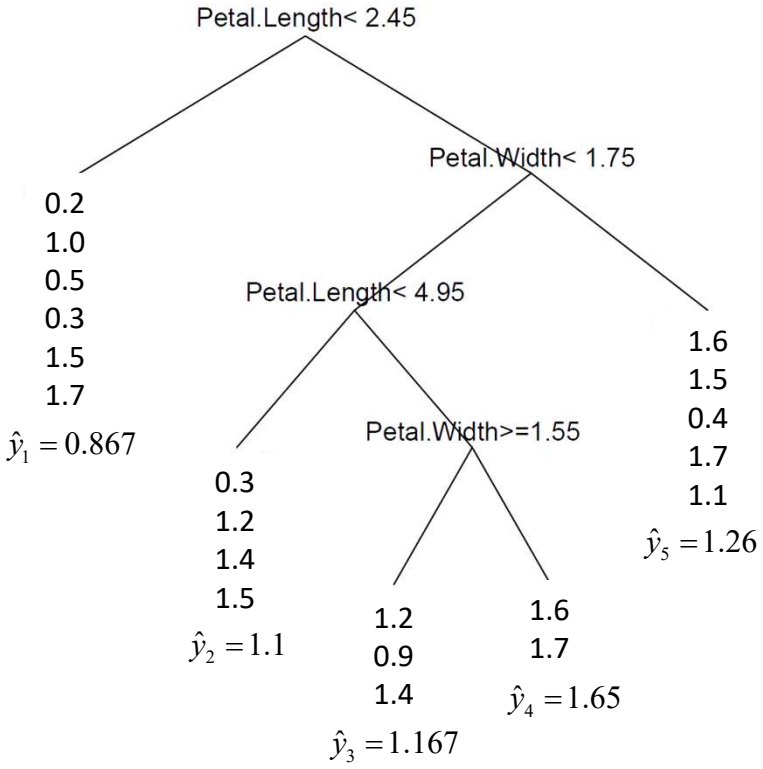
- Assume that the following probability tree structure is fixed, and there is a training data set of *N=20* instances. At each leaf we show the labels of the training instances sorted into this leaf



- Then the probability estimates $\mathbf{p}_k$ are relative class frequencies as shown

Example: Predictions at Leaves for Probability Estimation

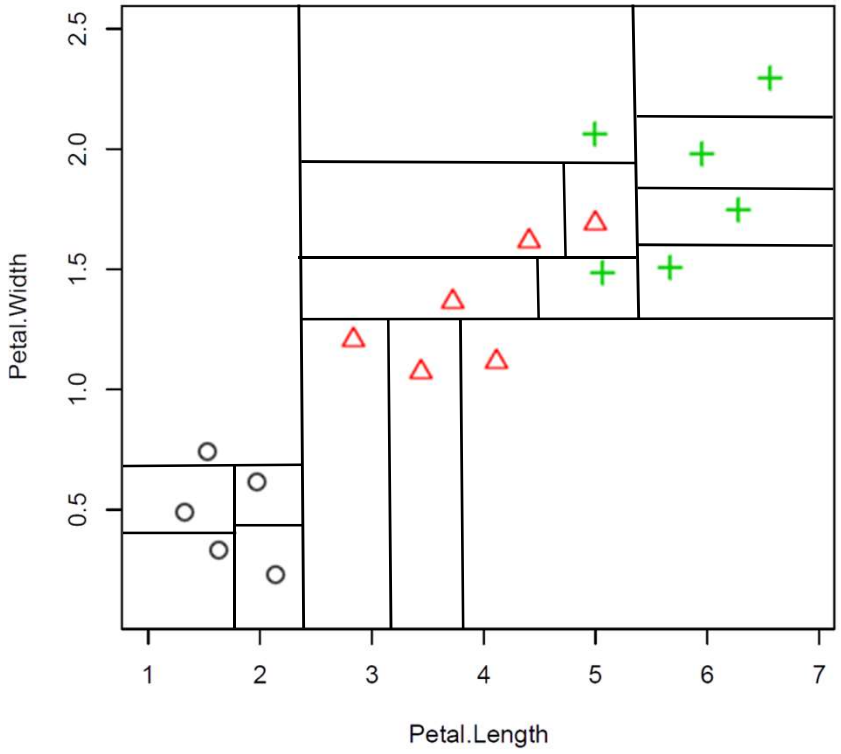
- Assume that the following regression tree structure is fixed, and there is a training data set of *N=20* instances. At each leaf we show the targets of the training instances sorted into this leaf



- Then the regression predictions $\hat{y}_k$ are the average targets at the leaf as shown

Overfitting for Decision Trees

- How can we learn the structure of a decision tree from data?
- Observation: decision trees can easily overfit any data set. For any training data set $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, there is a decision tree in which each $\mathbf{x}_i$ lies in a different cell (assuming no duplicate instances). Such a tree has error zero but likely overfits



Regularization for Decision Trees

- Decision tree learning therefore needs to be regularized
- Roughly speaking, we prefer **small trees** that still have **high accuracy** on the training data
- There is, of course, a trade-off between these two properties

- The preference for small trees can be formalized in different ways:
  - Limit the minimum number of instances that fall into a cell: avoid overly complex partitioning of input space
  - Limit the overall number of cells in the partitions (the overall number of leaves in the tree)
  - Limit the maximum depth of the tree

- We could treat these parameters as hyperparameters that restrict the model space and then search for the most accurate tree in the restricted space
- Unfortunately, there is no algorithm for carrying out this search efficiently, and an exhaustive search through all trees is computationally prohibitive

Learning the Tree Structure

- Because a global search through all tree structures is infeasible, decision tree learning algorithms are based on a greedy search through the space of trees:
  - Start with a tree consisting only of a single node, which will become the root node of the final tree. In this tree, all instances fall into this single node
  - Build the tree recursively:
    - split the current node by a splitting rule that sorts the instances at the node into two or several child nodes
    - recursively call the tree building algorithm on the new child nodes

- The selection of the split is greedy: try to find a locally optimal (or at least locally good) split
- A good split is generally a split that makes the subsets of examples at the child node more „pure“, that is, more homogeneous in terms of their labels
- If the labels are reasonably homogeneous, we can stop the splitting process and assign a prediction to the resulting leaf that agrees with most examples at that leaf
- This reduces the tree size

Decision Tree Learning: Pseudocode

- Recursive algorithm for learning a decision tree from training data
- Assuming binary splits and a limit on the minimum number of instances at which a node will still be split

Algorithm learn-decision-tree

Input : training instances $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_L, y_L)\}$, set of features $\mathcal{V}$,
minimum number of examples $N_{\min}$ for splitting a node

Output : learned decision tree represented by root node

1. node := createNode($\mathcal{D}$)
2. $\mathcal{Y}_{node} = \{y \mid (x, y) \in \mathcal{D}\}$
2. if $|\mathcal{D}| < N_{min}$ or $|\mathcal{Y}_{node}| = 1$:
3. node.prediction := $\arg\max_y \sum_{(x_n, y_n) \in \mathcal{D}} I(y_n = y)$
4. return node
5. node.splittingRule := bestSplit($\mathcal{D}, \mathcal{V}$)
6. $\mathcal{D}_{left}, \mathcal{D}_{right}$:= applySplit(node.splittingRule, $\mathcal{D}$)
7. node.leftChild := learn-decision-tree($\mathcal{D}_{left}, \mathcal{V}, N_{min}$)
8. node.rightChild := learn-decision-tree($\mathcal{D}_{right}, \mathcal{V}, N_{min}$)
9. return node

This node becomes the root node (of final decision tree or, in recursive call, a subtree)

Node becomes a leaf if all examples are of one class or not enough instances left to split

Evaluates a set of possible splits at the node and chooses the locally best one (see below)

Which Splits to Consider

- We still have to search for a locally good split
  - Which splits should we consider?
  - Which criterion should we use for selecting a split (see below)

- **Splits to consider: continuous features**
  - For a continuous feature $x_i \in \mathbb{R}$, consider in principle all splits of the form $x_i < C$ for $C \in \mathbb{R}$
  - However, not necessary to try out multiple splitting constants $C \in \mathbb{R}$ that would all lead to the same assigment of training instances to the child nodes
  - Let $v_1^{(i)},...,v_{N_i}^{(i)}$, with $N_i \leq N$, denote the distinct values of the feature x_i in the training data on which to split, sorted in ascending order ($v_j^{(i)} < v_{j+1}^{(i)}$)
  - Only consider the $N_i \leq N$ splitting constants $C_n = \frac{v_n^{(i)} + v_{n+1}^{(i)}}{2}$
- Example:
  - Values of feature $x_i \in \mathbb{R}$ are $(15,5,10,5,10,15,15)$, $N = 7$
  - $v_1^{(i)},...,v_{N_i}^{(i)} = (5,10,15)$
  - Try splits $x_i < 7.5$ and $x_i < 12.5$

Selecting Splits

- We still have to search for a locally good split
  - Which splits should we consider?
  - Which criterion should we use for selecting a split (see below)

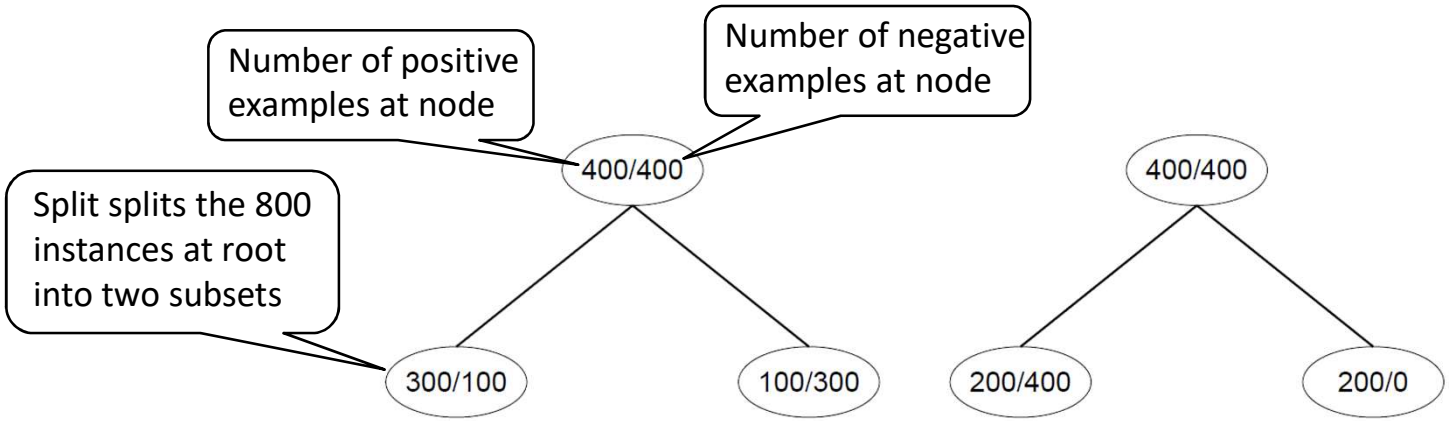
- **Splits to consider: nominal features (assuming binary splits)**
  - For a nominal feature $x_i \in \{c_1,...,c_K\}$ consider all subsets $\mathcal{C} \subset \{c_1,...,c_K\}$
  - For each subset, consider the splitting rule: left branch if $x_i \in \mathcal{C}$, right branch otherwise
  - There are $2^{K-1}-1$ possible splits of this kind: 2^K subsets, but the splits for $\mathcal{C}$ and $\{c_1,...,c_K\} \setminus \mathcal{C}$ are the same, and we do not allow $\mathcal{C}=\varnothing$

- Example:
  - Let $x_i \in \{\text{Hildesheim, Göttingen, Hannover}\}$
  - Splits are

$$\{\text{Hannover}\} \text{ versus } \{\text{Hildesheim, Göttingen}\}$$
$$\{\text{Hildesheim}\} \text{ versus } \{\text{Göttingen, Hannover}\}$$
$$\{\text{Göttingen}\} \text{ versus } \{\text{Hildesheim, Hannover}\}$$

Accuracy as Splitting Criterion?

- **Which criterion should we use for selecting splits?**
- Attempt 1: we could use the „natural“ criterium of classification error
- Example: consider the following two splits for a binary classification problem:



Accuracy if we do not split the node (treat the root node as leaf): 0.5
Accuracy if we perform the first split and treat the child nodes as leaves: 0.75
Accuracy if we perform the second split and treat the child nodes as leaves: 0.75

But the second split is better: creates a pure node that does not need to be split further

Entropy

- While the goal of tree learning is to obtain high training accuracy, we also aim for small trees, that is, we prefer splits that will require fewer splits in the subtrees
- Therefore, we use splitting criteria that aim to create „pure“ subsets, that is, subsets that are homogeneous in terms of their labels

- One such splitting criterion is based on the **entropy** of the resulting subsets
- Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_L, y_L)\}$ denote a set of examples with $y_l \in \{1, ..., T\}$, and let

$$p_t = \frac{\sum_{l=1}^{L} I(y_l = t)}{L} \qquad \text{t} \in \{1, ..., T\}$$

  denote the relative frequencies of classes in the set of examples
- The **Shannon entropy** of the set $\mathcal{D}$ is

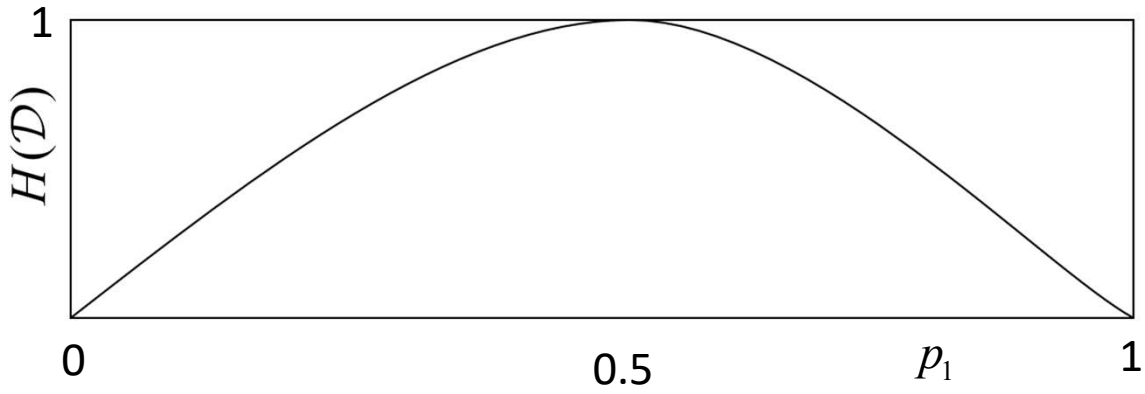
$$H(\mathcal{D}) = -\sum_{t=1}^{T} p_t \log_2 p_t$$

  where $\log_2$ denotes the base-two logarithm

Entropy

- The Shannon entropy is a measure of the „pureness“ of the labels in the set $\mathcal{D}$
- The entropy is maximal if $p_1 = p_2 = ... = p_T$, and minimal if all examples are of the same class

Visualization for *T=2*:



Information Gain

- Now assume a split that splits the set $\mathcal{D}$ at the current node into the two subsets $\mathcal{D}_1$ and $\mathcal{D}_2$
- The **information gain** of the split is the reduction in entropy obtained after splitting the set $\mathcal{D}$:

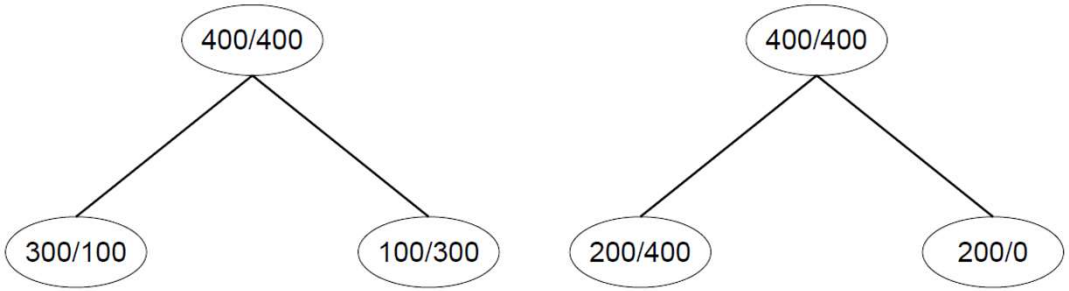
$$IG(\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2) = H(\mathcal{D}) - \sum_{r=1}^{2} \frac{|\mathcal{D}_r|}{|\mathcal{D}|} H(\mathcal{D}_r)$$

- The higher the information gain, the better the split
- For example, if the split results in two pure nodes, the information gain is equal to $H(\mathcal{D})$

- In decision tree learning algorithm, we can evaluate all possible splits according to information gain and choose the split that has highest information gain

Example: Information Gain

- In the example of two possible splits given above, the information of the second split is indeed higher than for the first split:



Information Gain:

$$H(\mathcal{D}) = -0.5 \cdot \log_2 0.5 - 0.5 \cdot \log_2 0.5 = 1$$
$$H(\mathcal{D}_1) = -0.75 \cdot \log_2 0.75 - 0.25 \cdot \log_2 0.25 \approx 0.811$$
$$H(\mathcal{D}_2) = -0.25 \cdot \log_2 0.25 - 0.75 \cdot \log_2 0.75 \approx 0.811$$
$$IG(\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2) = H(\mathcal{D}) - \sum_{r=1}^{2} \frac{|\mathcal{D}_r|}{|\mathcal{D}|} H(\mathcal{D}_r)$$
$$\approx 1 - 0.5 \cdot 0.811 - 0.5 \cdot 0.811 \approx 0.189$$

Information Gain:

$$H(\mathcal{D}) = -0.5 \cdot \log_2 0.5 - 0.5 \cdot \log_2 0.5 = 1$$
$$H(\mathcal{D}_1) \approx -0.67 \cdot \log_2 0.67 - 0.33 \cdot \log_2 0.33 \approx 0.915$$
$$H(\mathcal{D}_2) = 0 \cdot \log_2 0 - 1 \cdot \log_2 1 = 0$$
$$IG(\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2) = H(\mathcal{D}) - \sum_{r=1}^{2} \frac{|\mathcal{D}_r|}{|\mathcal{D}|} H(\mathcal{D}_r)$$
$$\approx 1 - 0.75 \cdot 0.915 - 0.25 \cdot 0 \approx 0.314$$

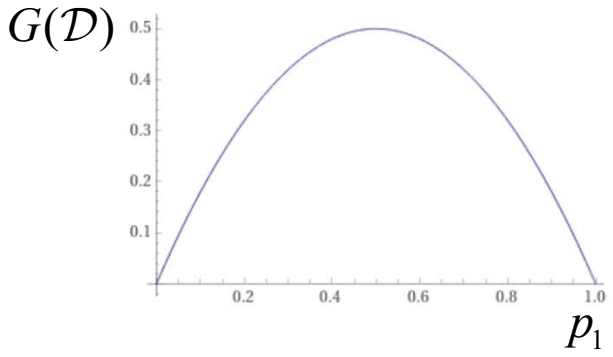
Gini-Impurity

- An alternative to using entropy and information gain in order to score possible splits in decision tree learning is the **Gini index**
- In the Gini index, we replace Shannon entropy with Gini-impurity:

$$G(\mathcal{D}) = 1 - \sum_{t=1}^{T} p_t^2$$

- Similar as for entropy, Gini impurity is highest if the label distribution in the sample is uniform and lowest if all labels are of the same class

Visualization: *T=2*

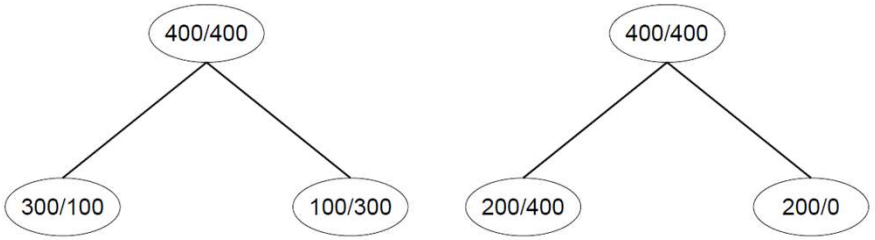


Gini-Index

- The Gini-index measures the reduction in Gini-impurity for a split:

$$GI(\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2) = G(\mathcal{D}) - \sum_{r=1}^{2} \frac{|\mathcal{D}_r|}{|\mathcal{D}|} G(\mathcal{D}_r)$$

- Example:



Gini-index:

$$G(\mathcal{D}) = 1 - 0.5^2 - 0.5^2 = 0.5$$
$$G(\mathcal{D}_1) = 1 - 0.75^2 - 0.25^2 = 0.375$$
$$G(\mathcal{D}_2) = 1 - 0.25^2 - 0.75^2 = 0.375$$
$$GI(\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2) = G(\mathcal{D}) - \sum_{r=1}^{2} \frac{|\mathcal{D}_r|}{|\mathcal{D}|} G(\mathcal{D}_r)$$
$$= 0.5 - 0.5 \cdot 0.375 - 0.5 \cdot 0.375 = 0.125$$

Gini-index:

$$G(\mathcal{D}) = 1 - 0.5^2 - 0.5^2 = 0.5$$
$$G(\mathcal{D}_1) \approx 1 - 0.67^2 - 0.33^2 \approx 0.442$$
$$G(\mathcal{D}_2) = 1 - 1^2 - 0^2 = 0$$
$$GI(\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2) = G(\mathcal{D}) - \sum_{r=1}^{2} \frac{|\mathcal{D}_r|}{|\mathcal{D}|} G(\mathcal{D}_r)$$
$$\approx 0.5 - 0.75 \cdot 0.442 - 0.25 \cdot 0 \approx 0.1685$$

Popular Decision Tree Configurations

- Aspects of decision tree learning algorithm such as the types of splits considered, the criterion for selecting splits, and the stopping criterion can in principle be arbitrarily combined
- There are certain popular, widely used decision tree configurations:

name	ChAID	CART	ID3	C4.5
author	Kass 1980	Breiman et al. 1984	Quinlan 1986	Quinlan 1993
selection measure	χ^2	Gini index, twoing index	information gain	information gain ratio
splits	all	binary nominal, binary quantitative, binary bivariate quantitative	complete	complete, binary nominal, binary quantitative
stopping criterion	χ^2 independence test	minimum number of cases/node	χ^2 independence test	lower bound on selection measure
pruning technique	none	error complexity pruning	pessimistic error pruning	pessimistic error pruning, error based pruning

- Many of these involve a „pruning“ step: after learning, reduce the size of the decision tree

Decision Trees: Advantages and Disadvantages

- The decision tree algorithms discussed in this lecture are relatively basic, and usually do not quite achieve state-of-the-art predictive performance

- Their simplicity also has some advantages, though:
  - Learning decision trees is relatively fast
  - Prediction in decision trees is extremely fast, even if there are many features
  - The resulting model is quite interpretable: by looking at the tree, we easily understand how a certain prediction was obtained

- Moreover, decision trees are often used as building blocks in ensemble models such as random forest (see lecture Machine Learning 2), and these are widely used and often achieve state-of-the-art performance

Summary Decision Trees

- Decision trees are tree-structured prediction models that
  - contain splitting rules at inner nodes, to sort instances into one of the subtrees
  - a prediction at the terminal nodes (leaves)

- Decision trees are learned from data using a greedy recursive algorithm
  - starting at the root node, pick a splitting rule that divides the instances at the node into subsets that are more homogeneous in terms of their labels
  - recursively call the algorithm on the nodes resulting from the split
  - goal is to obtain a small tree with reasonably high training set accuracy

- Popular criteria for selecting splitting rules are information gain or Gini index

Further Reading

- Further reading: [Hastie et al., 2005, chapter 9.2+6+7], [Murphy, 2012, chapter 16.1{2], [James et al., 2013, chapter 8.1+3]

- Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The Elements of Statistical Learning: Data Mining, Inference and Prediction, volume 27. Springer, 2005.
- Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An Introduction to Statistical Learning. Springer, 2013.
- Kevin P. Murphy. Machine Learning: A Probabilistic Perspective. The MIT Press, 2012.