Decision Trees

Wiebke Köpp

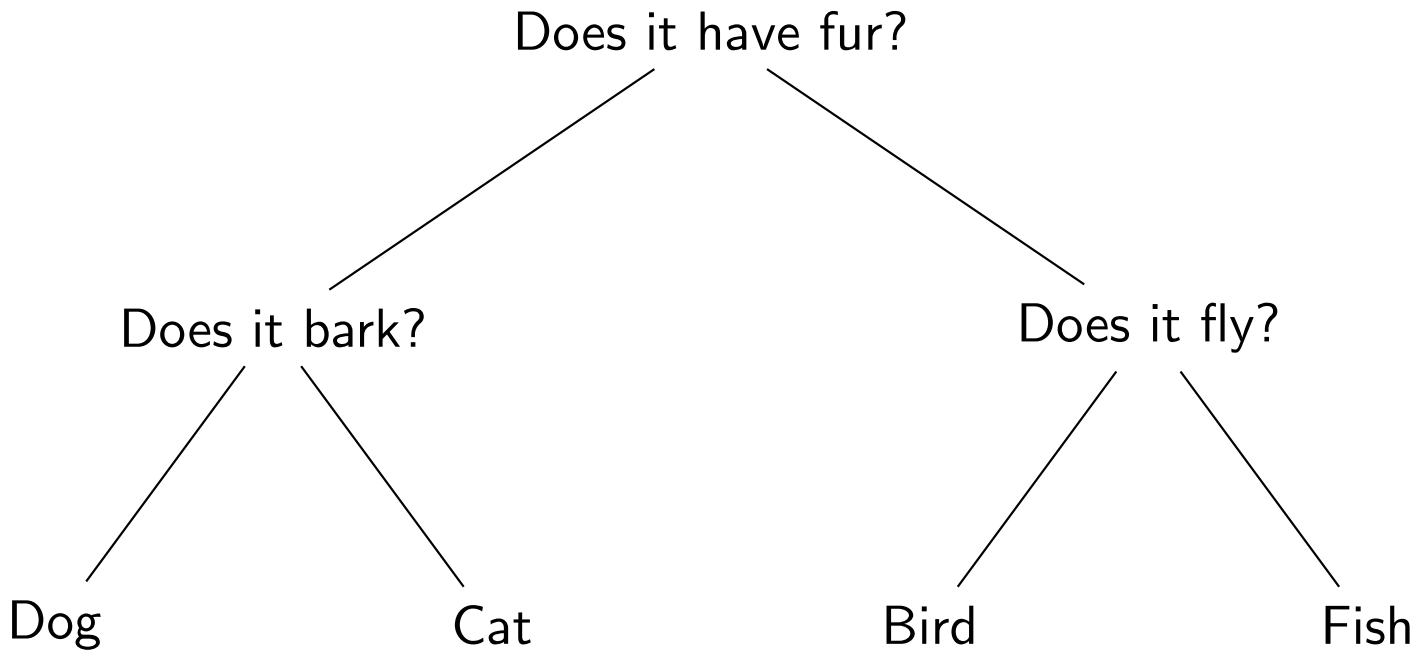
Technische Universität München

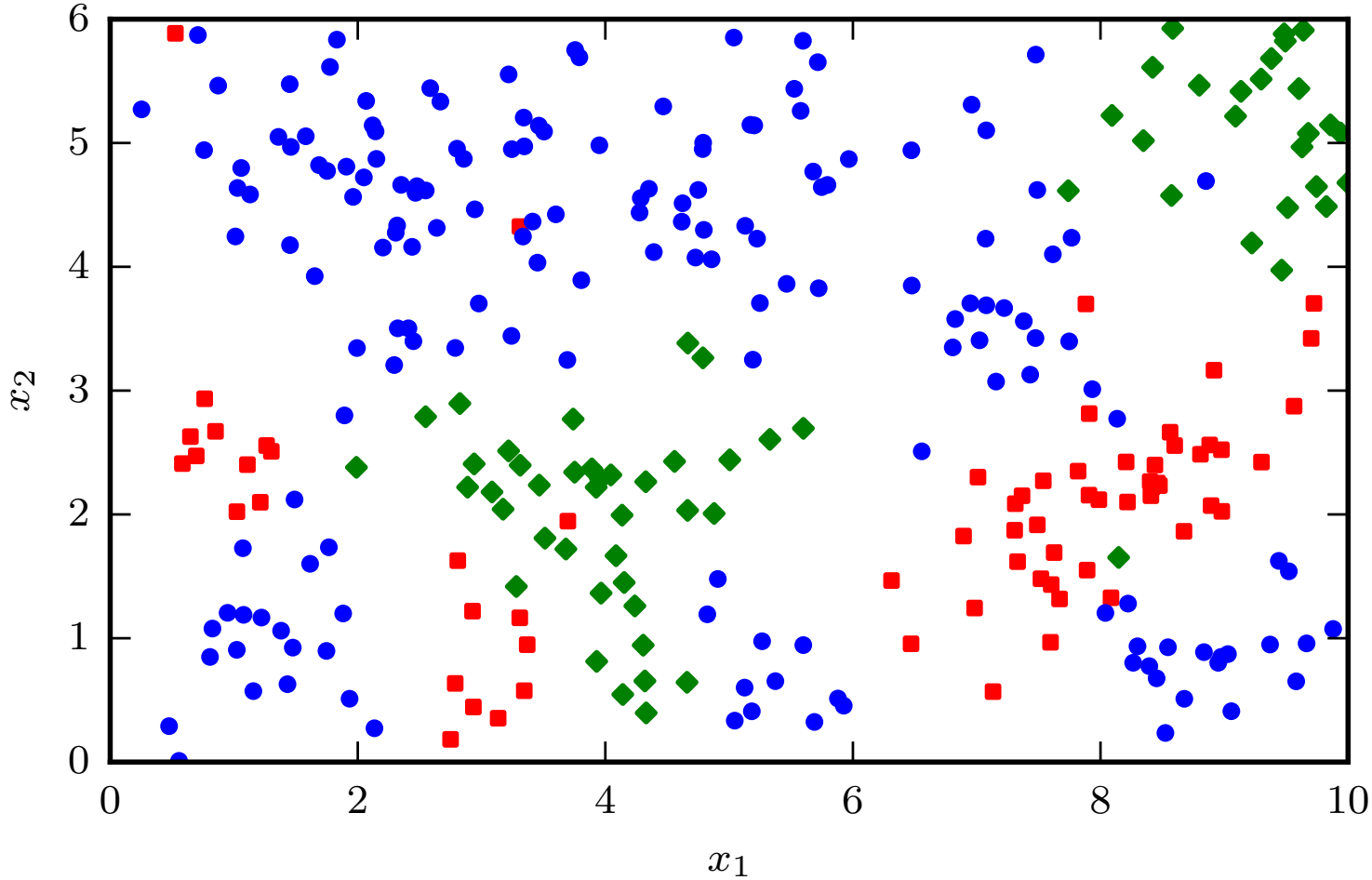
Reading Material:
"Machine Learning: A Probabilistic Perspective" by Murphy [ch. 16.2]

Further extra reading:
"Pattern Recognition and Machine Learning" by Bishop [ch. 14.4]

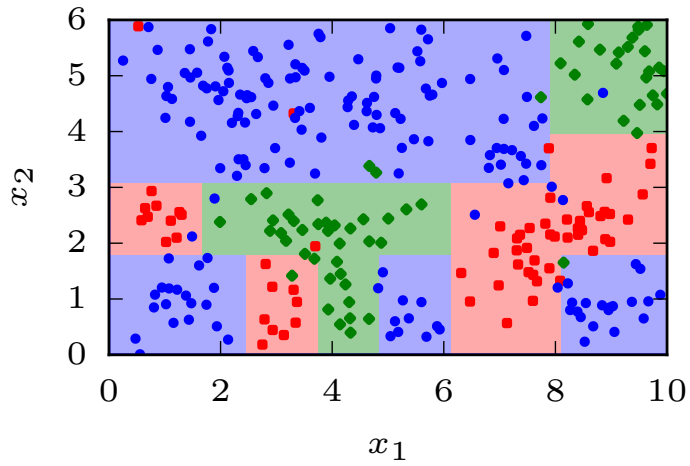
Note: These slides are adapted from slides originally by Daniala Korhammer. Additionally, some of them are inspired by *Understanding Random Forests* (Phd thesis by Gilles Louppes)

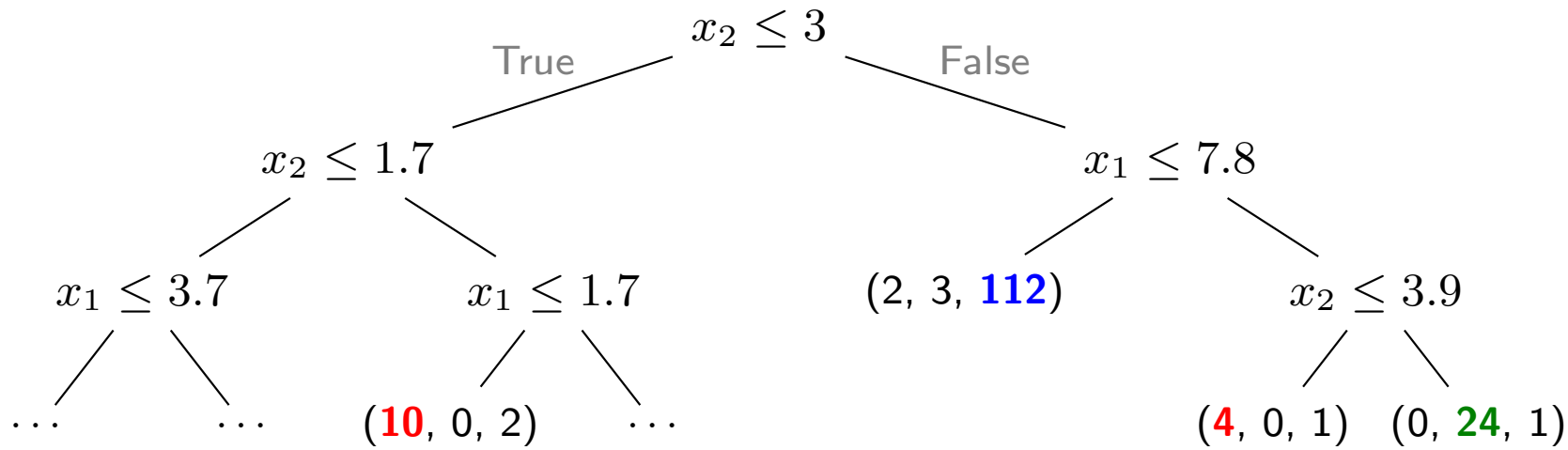
The 20-Questions Game





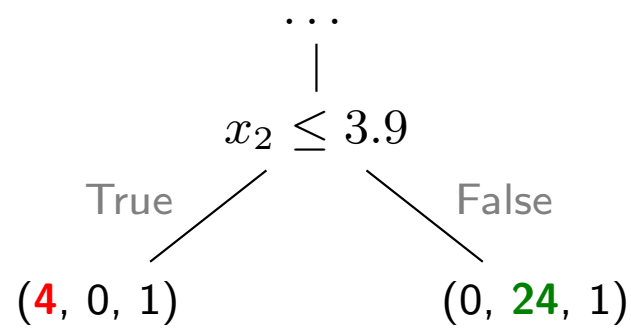
Example: data $\boldsymbol{X}$ with two *features* x_1 and x_2 and class labels $\boldsymbol{z}$
Goal: *classification* of unseen instances (*generalization*)





Distribution of classes in leaf: (red, green, blue)

Interpretation of a Decision Tree



- Node $\widehat{=}$ *feature* test $\rightarrow$ leads to decision boundaries.
- Branch $\widehat{=}$ different outcome of the preceding feature test.
- Leaf $\widehat{=}$ region in the input space and the distribution of *samples* in that region.

Decision trees partition the input space into cuboid regions.

To classify a new sample $\boldsymbol{x}$:

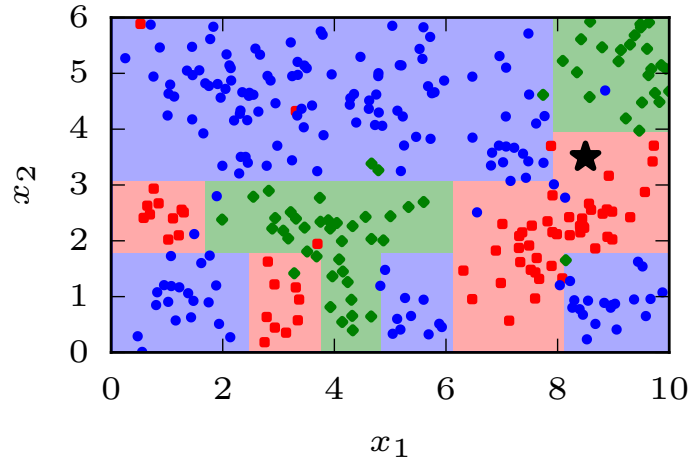
- Test the attributes of $\boldsymbol{x}$ to find the region $\mathcal{R}$ that contains it and get the class distribution $\boldsymbol{n}_{\mathcal{R}} = (n_{c_1,\mathcal{R}}, n_{c_2,\mathcal{R}}, \ldots, n_{c_k,\mathcal{R}})$ for $C = \{c_1, \ldots, c_k\}$.
- The probability that a data point $\boldsymbol{x} \in \mathcal{R}$ should be classified belonging to class c is then:

$$p(z = c \mid \mathcal{R}) = \frac{n_{c,\mathcal{R}}}{\sum_{c_i \in C} n_{c_i,\mathcal{R}}}$$

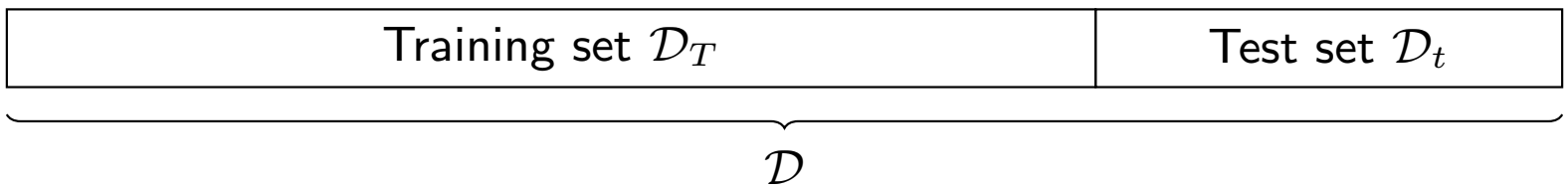
- A new unseen sample $\boldsymbol{x}$ is simply given the label which is most common in its corresponding region:

$$y = \arg\max_c \; p(z = c \mid \boldsymbol{x}) = \arg\max_c \; p(z = c \mid \mathcal{R}) = \arg\max_c \; n_{c,\mathcal{R}}$$

Classifcation of $\boldsymbol{x} = (8.5, 3.5)^T$



What does the perfect decision tree look like?
It performs well on new data. $\rightarrow$ *generalization*
How can we test this?

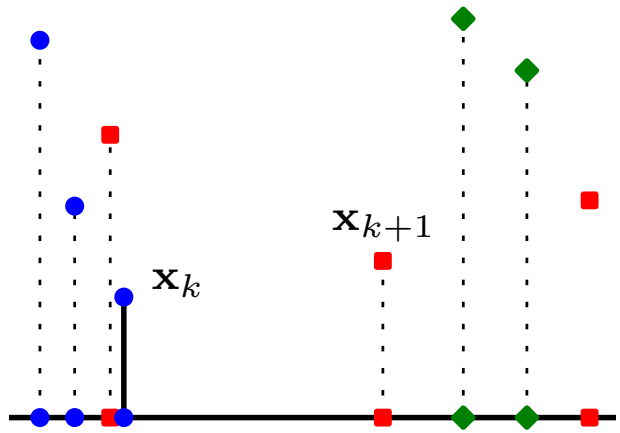


- Split training data $\mathcal{D}$ into a *training set* $\mathcal{D}_T = (\boldsymbol{X}_T, \boldsymbol{z}_T)$ and a *test set* $\mathcal{D}_t = (\boldsymbol{X}_t, \boldsymbol{z}_t)$,
- build tree from training set $\mathcal{D}_T$,
- predict *test set* labels $\boldsymbol{y}_t$ using the tree,
- and compare predictions $\boldsymbol{y}_t$ to true labels $\boldsymbol{z}_t$ for evaluation.

Idea: Build all possible trees and evaluate how they perform on new data.

All combinations of features and values can serve as tests in the tree:

feature	tests
x_1	$\leq$ 0.36457631
	$\leq$ 0.50120369
	$\leq$ 0.54139549
	$\leq \ldots$
x_2	$\leq$ 0.09652214
	$\leq$ 0.20923062
	$\leq \ldots$



In our simple example:
2 features $\times$ 300 unique values per feature
2 features $\times$ 299 possible thresholds per feature:
598 possible tests at the root node, slightly fewer at each descendant

Iterating over all possible trees is possible only for very small examples ($\rightarrow$ low number of possible tests) because the number of trees quickly explodes.

Finding the optimal tree is *NP-complete*.

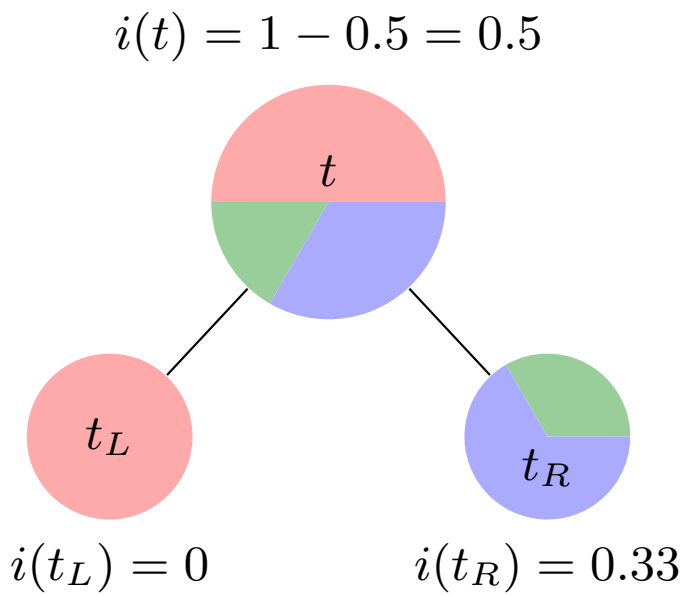
Instead: Grow the tree top-down and choose the best split node-by-node using a *greedy heuristic* on the *training data*.

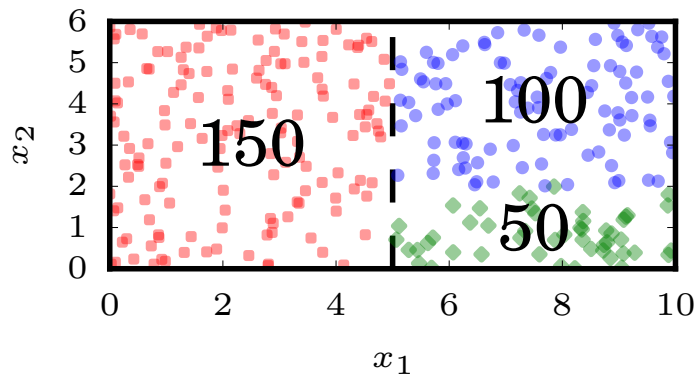
For example: Split the node when it improves the missclassification rate i_E at node t

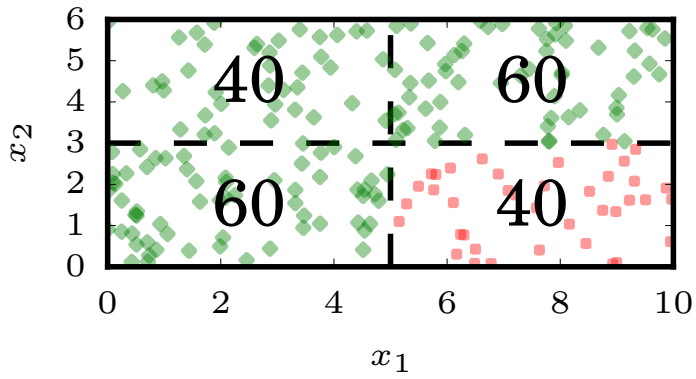
$$i_E(t) = 1 - \max_c \; p(z = c \mid t)$$

The improvement when performing a split s of t into t_R and t_L for $i(t) = i_E(t)$ is given by

$$\Delta i(s, t) = i(t) - p_L i(t_L) - p_R i(t_R)$$







no split:	$\frac{40}{200}$
$x_1 \leq 5$	$\frac{40}{200}$
$x_2 \leq 3$	$\frac{40}{200}$

Node is not split even though combining the two tests would result in perfect classification

Use a criterion $i(t)$ that measures how *pure* the class distribution at a node t is. It should be

- **maximum** if classes are equally distributed in the node
- **minimum**, usually 0, if the node is pure
- **symmetric**

Impurity measures

With $\pi_c = p(z = c \mid t)$:

Misclassification rate:

$$i_E(t) = 1 - \max_c \ \pi_c$$

Entropy:

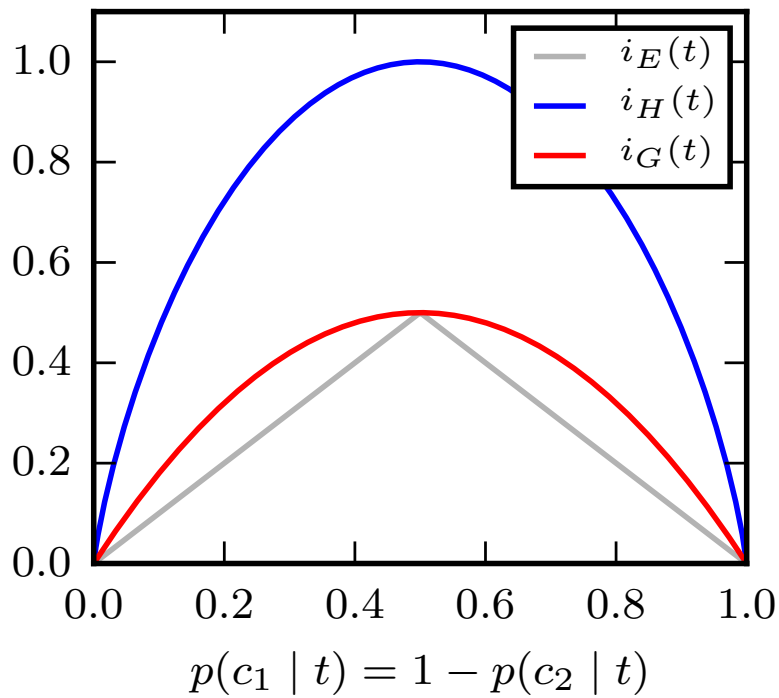
$$i_H(t) = -\sum_{c_i \in C} \pi_{c_i} \log \pi_{c_i}$$

(Note that $\lim_{x \to 0+} x \log x = 0$.)

Gini index:

$$i_G(t) = \sum_{c_i \in C} \pi_{c_i}(1 - \pi_{c_i})$$
$$= 1 - \sum_{c_i \in C} \pi_{c_i}^2$$

For $C = \{c_1, c_2\}$:



Detour: Information Theory

Information theory is about encoding and transmitting information

We would like to encode four messages:

- $m_1 =$ "You have a lecture." $p(m_1) = 0.01$ $\rightarrow$ Code 111
- $m_2 =$ "You have an exam." $p(m_2) = 0.02$ $\rightarrow$ Code 110
- $m_3 =$ "There is free beer." $p(m_3) = 0.30$ $\rightarrow$ Code 10
- $m_4 =$ "Nothing happening." $p(m_4) = 0.67$ $\rightarrow$ Code 0

The code above is called a *Huffman Code*.

On average:

$$0.01 \times 3\,\text{bits} + 0.02 \times 3\,\text{bits} + 0.3 \times 2\,\text{bits} + 0.67 \times 1\text{bit} = 1.36\,\text{bits}$$

Shannon Entropy

Shannon entropy gives a lower bound on the average number of bits needed to encode a set of messages.

It is defined over a discrete random variable X with possible values $\{x_1, \ldots, x_n\}$

$$H(X) = -\sum_{i}^{n} p(X = x_i) \log_2 p(X = x_i)$$

$H(X)$ measures the uncertainty about the outcome of X.

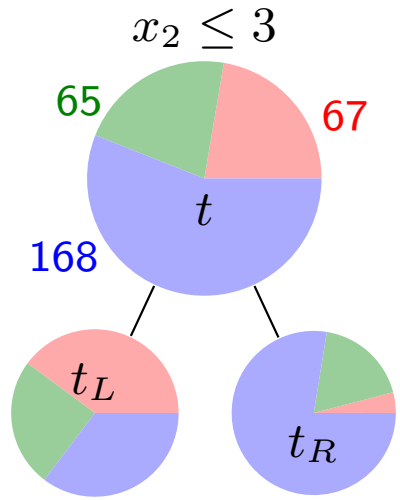
Building a decision tree

Compare all possible tests and choose the one where the improvement $\Delta i(s,t)$ for some splitting criterion $i(t)$ is largest

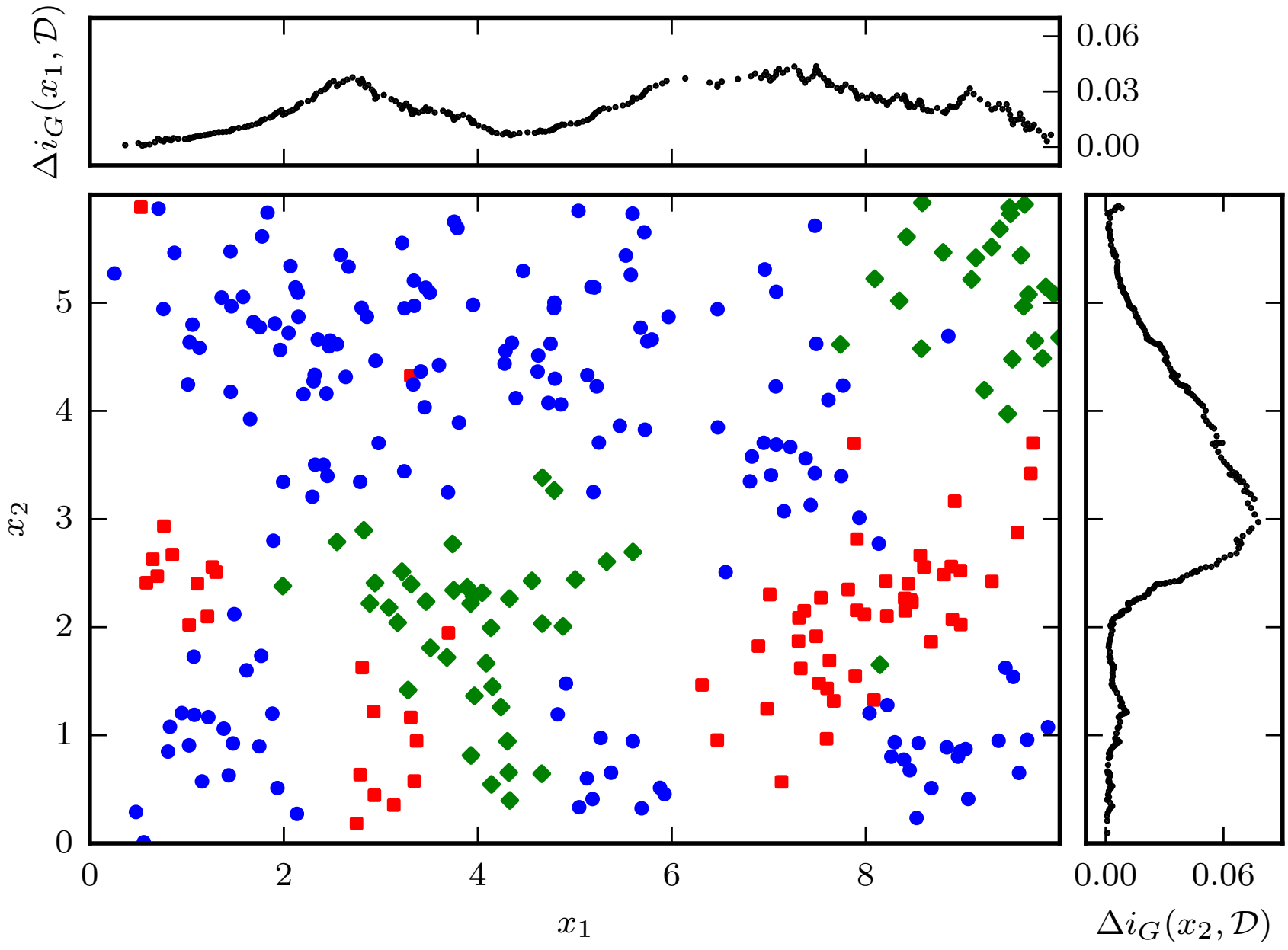
$$i_G(t) = 1 - \left(\frac{67}{300}\right)^2 - \left(\frac{65}{300}\right)^2 - \left(\frac{168}{300}\right)^2$$
$$\approx 0.5896$$

After testing $x_2 \leq 3$:

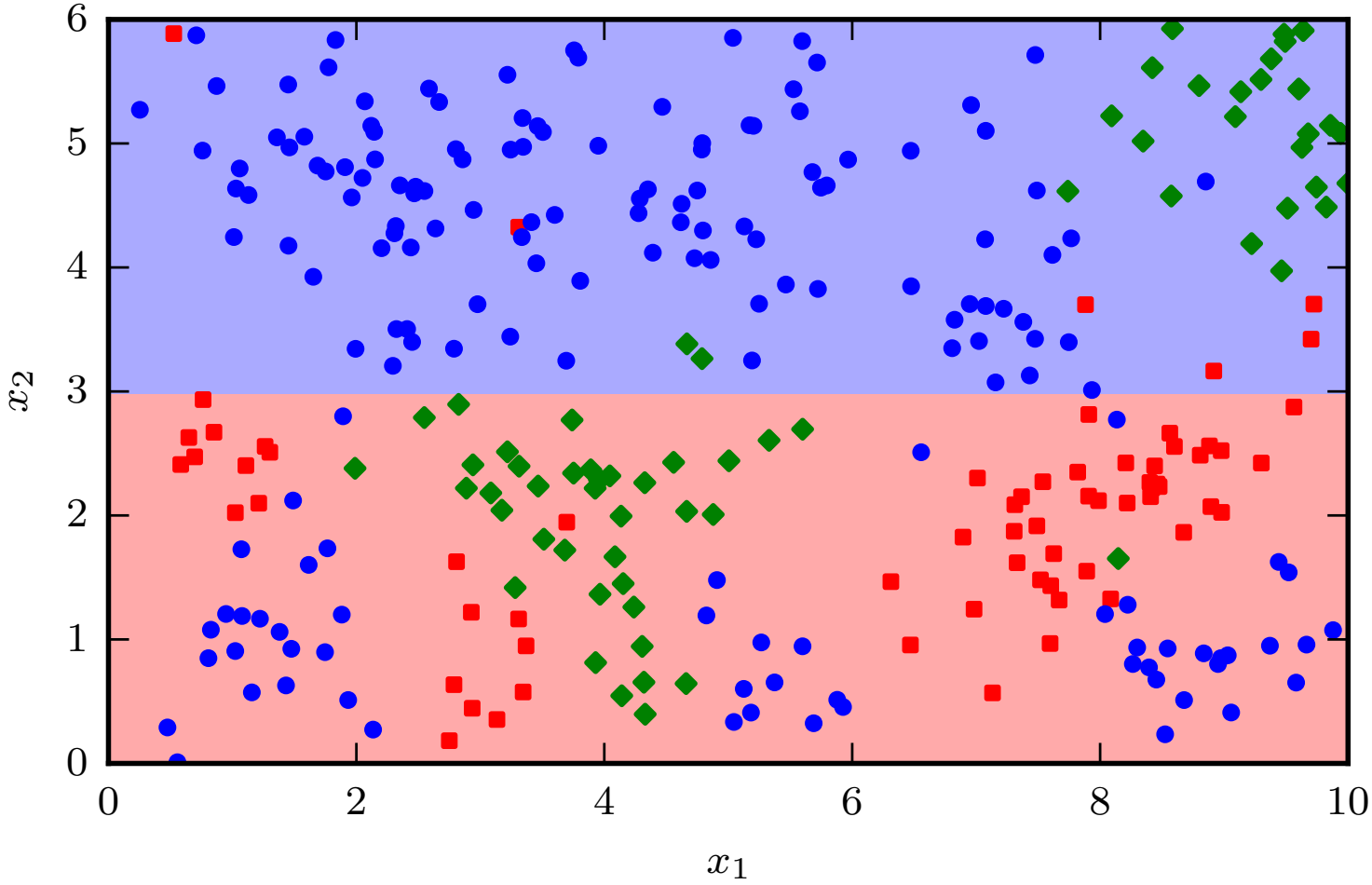
$i_G(t_L) \approx 0.6548$ and $i_G(t_R) \approx 0.3632$



$$\Rightarrow \Delta i_G(x_2 \leq 3, t) = i_G(t) - \frac{153}{300} i_G(t_L) - \frac{147}{300} i_G(t_R)$$
$$\approx 0.07768$$

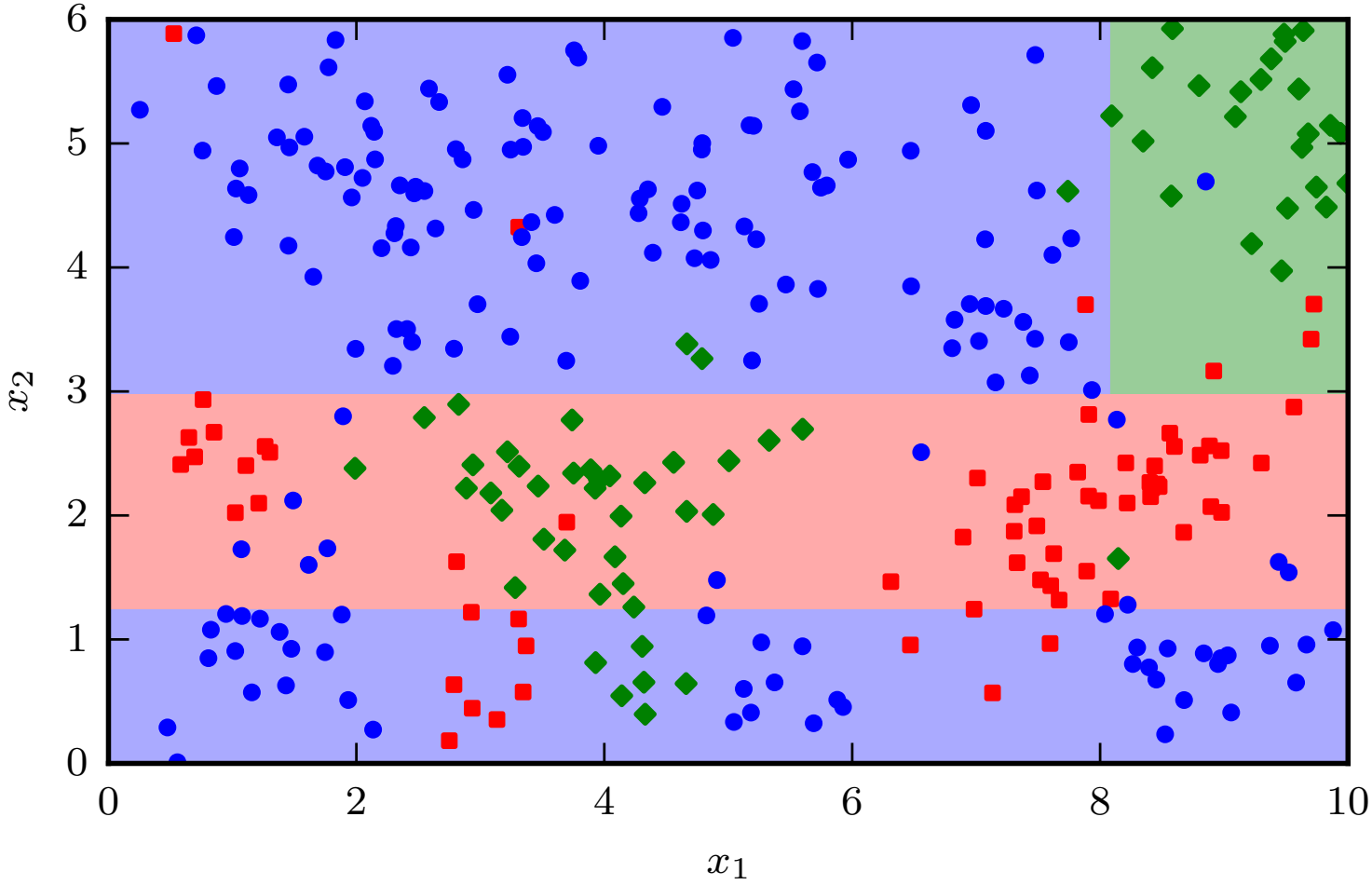

$\Delta i_G(x_1, \mathcal{D})$
0.06
0.03
0.00
x_2
0
1
2
3
4
5
0
2
4
6
8
x_1
0.00
0.06
$\Delta i_G(x_2, \mathcal{D})$

Decision boundaries at depth 1



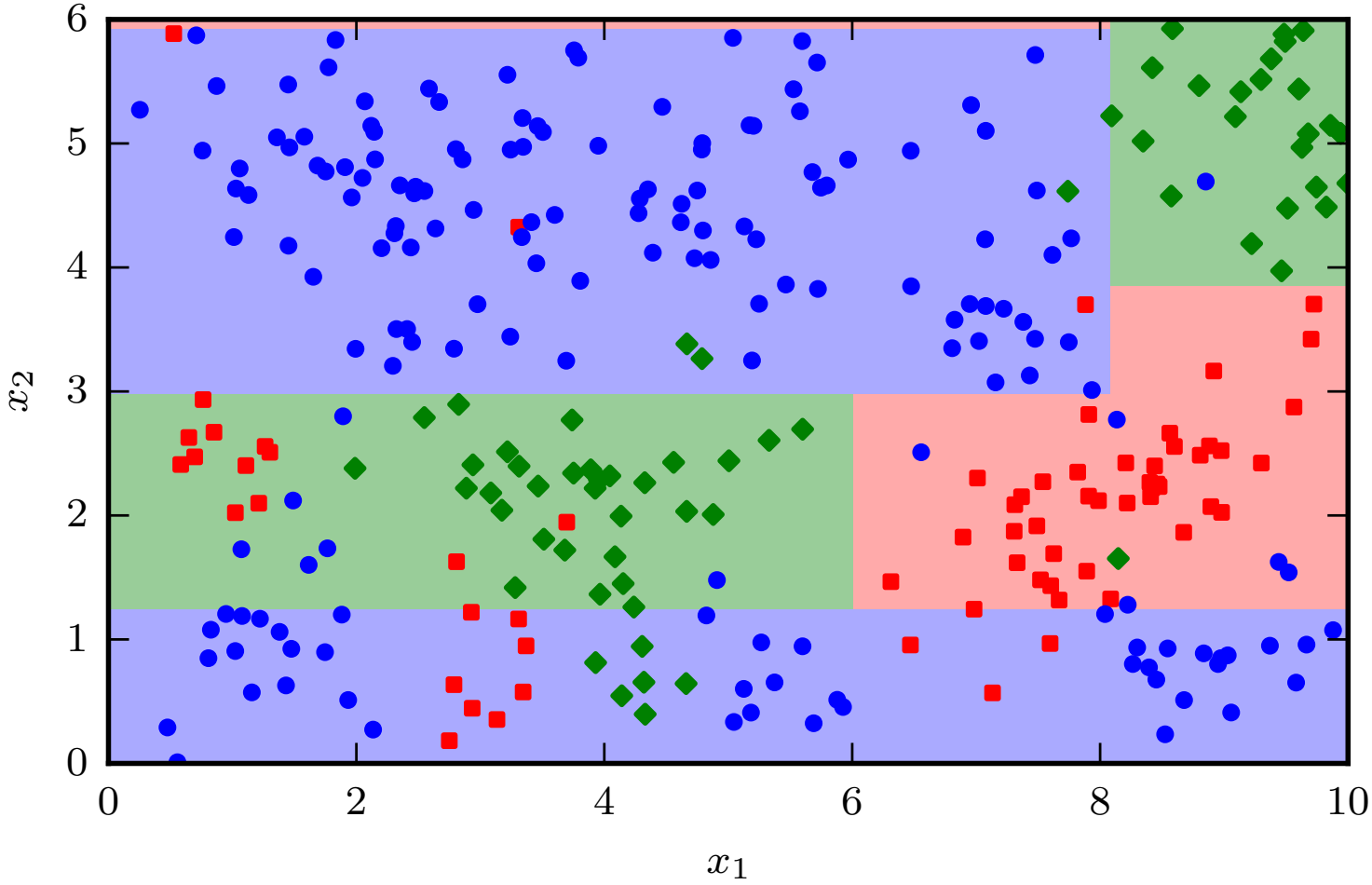
Accuracy on the whole data set: 58.3%

Decision boundaries at depth 2



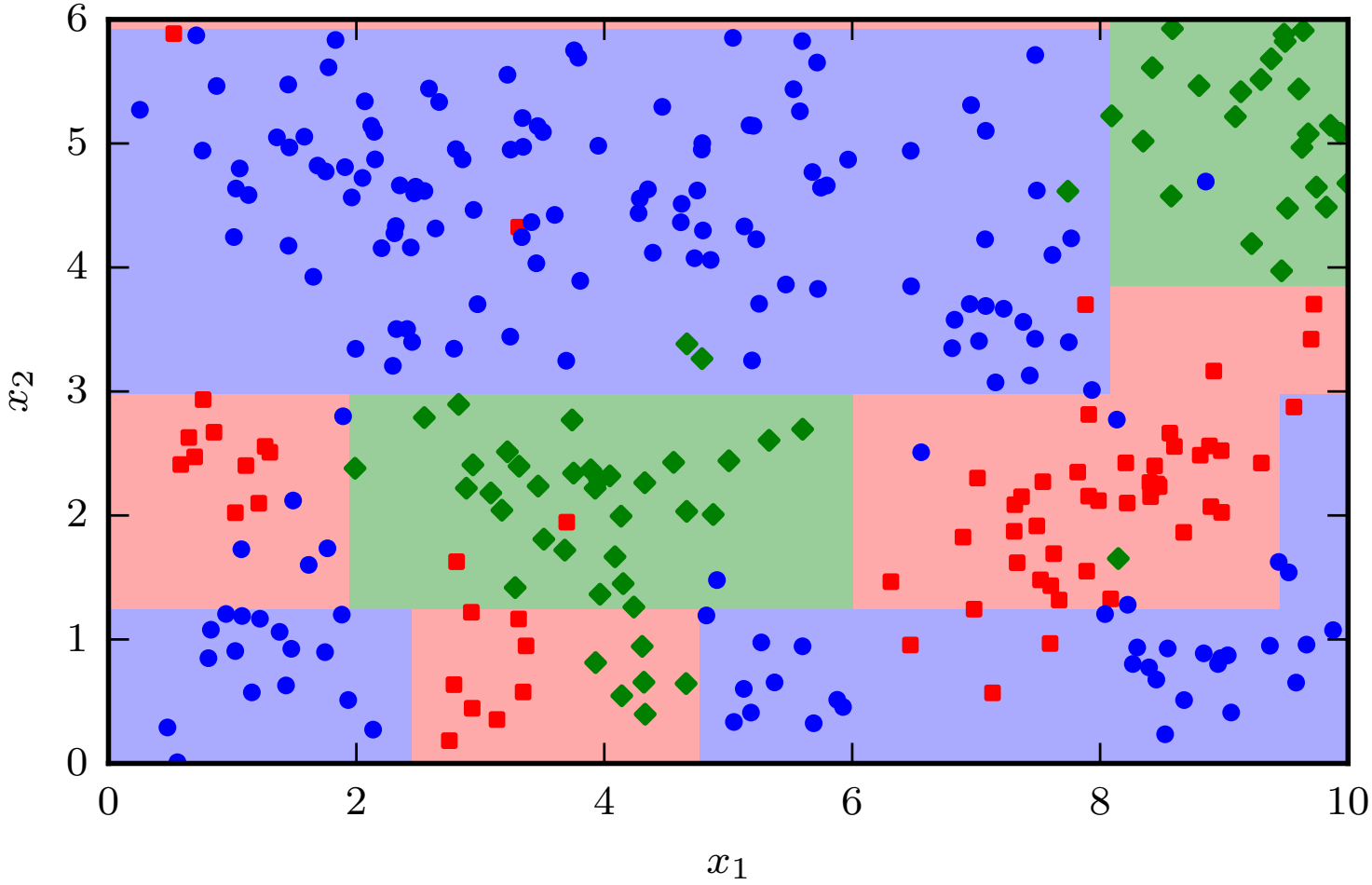
Accuracy on the whole data set: 77%

Decision boundaries at depth 3



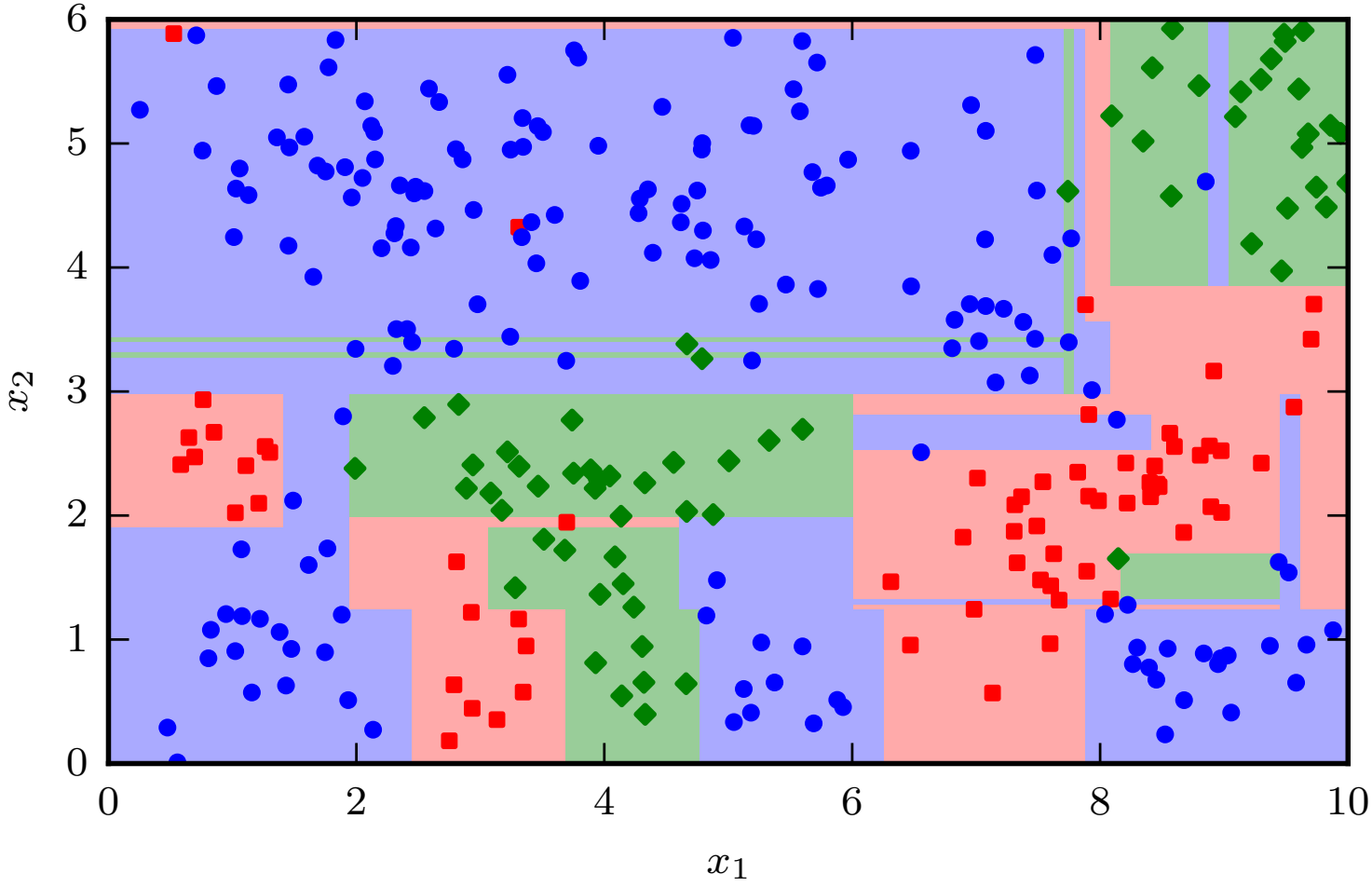
Accuracy on the whole data set: 84.3%

Decision boundaries at depth 4



Accuracy on the whole data set: 90.3%

Decision boundaries of a maximally pure tree



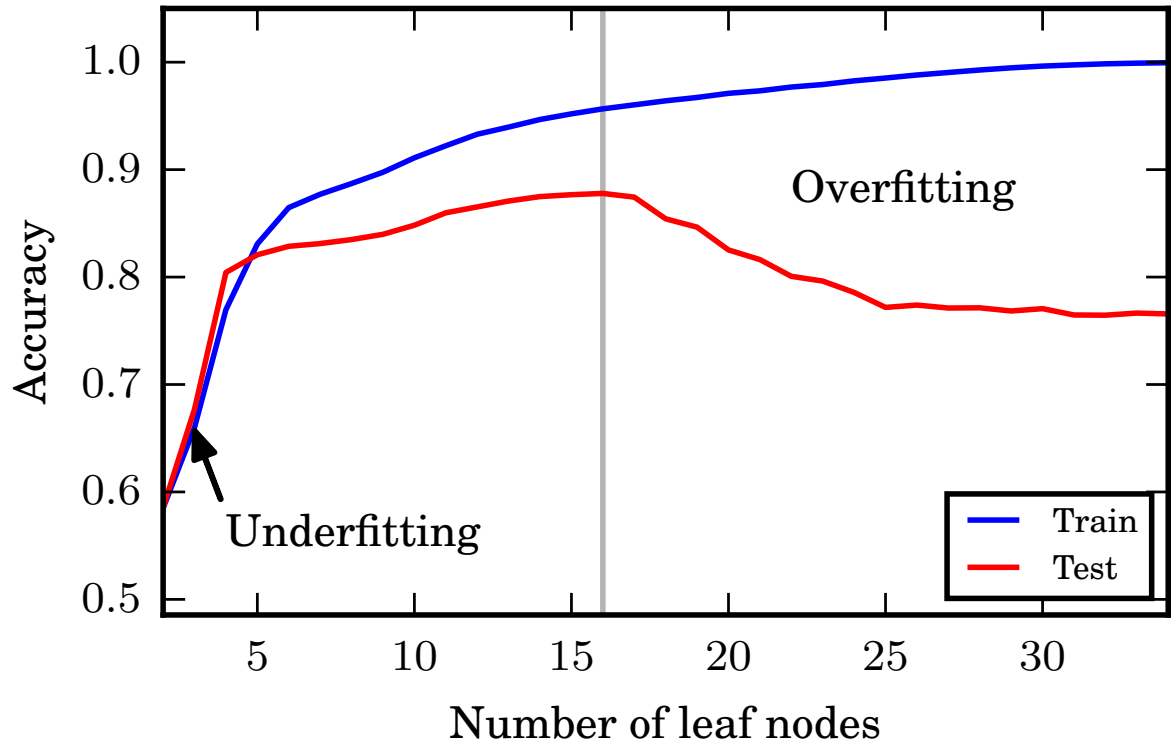
Accuracy on the whole data set: 100% $\rightarrow$ *generalisation*?

Overfitting

Overfitting typically occurs when we try to model the training data perfectly.

Overfitting means poor generalisation! How can we spot overfitting?

- low training error, possibly 0
- testing error is comparably high.



The training performance monotonically increases with every split.

The test performance tells us how well our model generalises, not the training performance! $\rightarrow$ validation set

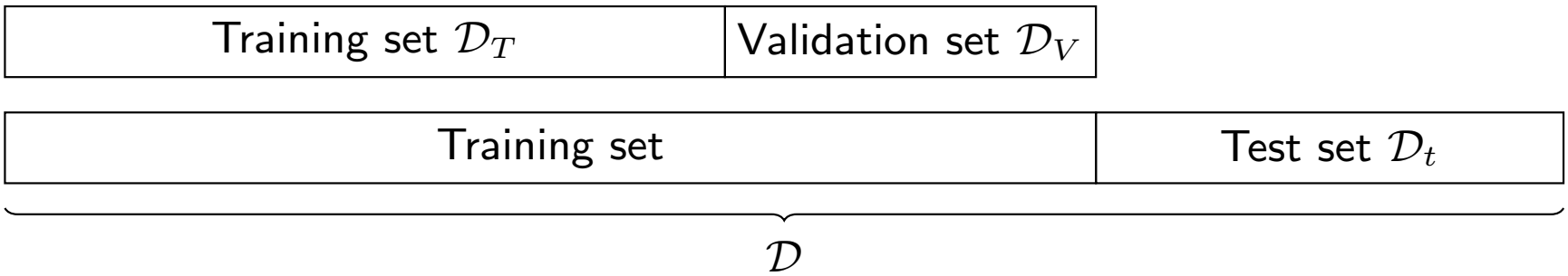
When to stop growing?

Possible stopping (or *pre-pruning*) criteria:

- distribution in branch is *pure*, i.e $i(t) = 0$
- maximum depth reached
- number of samples in each branch below certain threshold t_n
- benefit of splitting is below certain threshold $\Delta i(s, t) < t_\Delta$

Or we can grow a tree maximally and then (post-)prune it.

Creating the validation set



Be sure to have separate training, validation and test sets:

- training set $\mathcal{D}_T$ to build the tree,
- validation set $\mathcal{D}_V$ to prune the tree,
- test set $\mathcal{D}_t$ to evaluate the final model.

Splits of $(\frac{2}{3}, \frac{1}{3})$ are common.

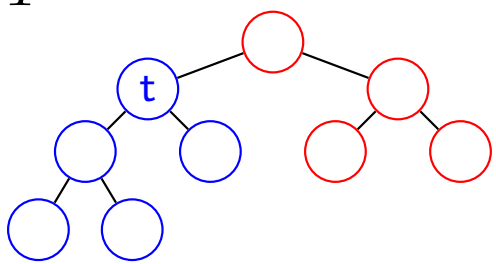
Test data for later evaluation should not be used for pruning or you will not get an honest estimate of the model's performance!

Reduced Error Pruning

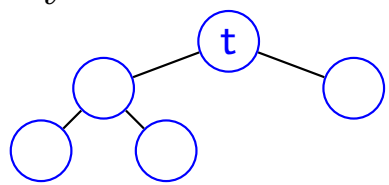
Let T be our decision tree and t one of its inner nodes.

Pruning T w.r.t. t means deleting all descendant nodes of t (but not t itself). We denote the pruned tree $T \backslash T_t$.

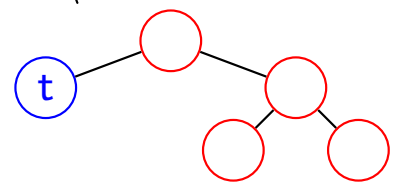
T



T_t



$T \backslash T_t$



- Use validation set to get an error estimate: $\mathrm{err}_{\mathcal{D}_V}(T)$.
- For each node t calculate $\mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$
- Prune tree at the node that yields the highest error reduction.
- Repeat until for all nodes t: $\mathrm{err}_{\mathcal{D}_V}(T) < \mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$.

After pruning you may use both training and validation data to update the labels at each leaf.

Random Forests

Train not only one decision tree but an army of them,
also called an *ensemble*.

To label a new data point, evaluate all trees and let them vote. Label the data point with the most "popular" class.

But. . . if we build all trees in the same way they will always vote in unison?

$\rightarrow$ We need to introduce variance between the trees!

Random Forests: Introducing Variance

Let N denote the number of samples in the training set $\mathcal{D}_T$ and D denote the number of features (attributes, dimensions).

- Instead of using the whole training set to build the tree, we build each tree T_i from a bootstrap sample:
  From $\mathcal{D}_T$ create $\mathcal{D}_{T_i}$ by *randomly* drawing N samples *with replacement*.

  By sampling with replacement we only use ≈ 63.2 % of $\mathcal{D}_T$ for each tree.
- Build decision trees T_i top-down with a greedy heuristic and a small variation:
  At each node, instead of picking the best of all possible tests, *randomly* select a subset of $d < D$ features and consider only tests on these features.
- There is no need for pruning.

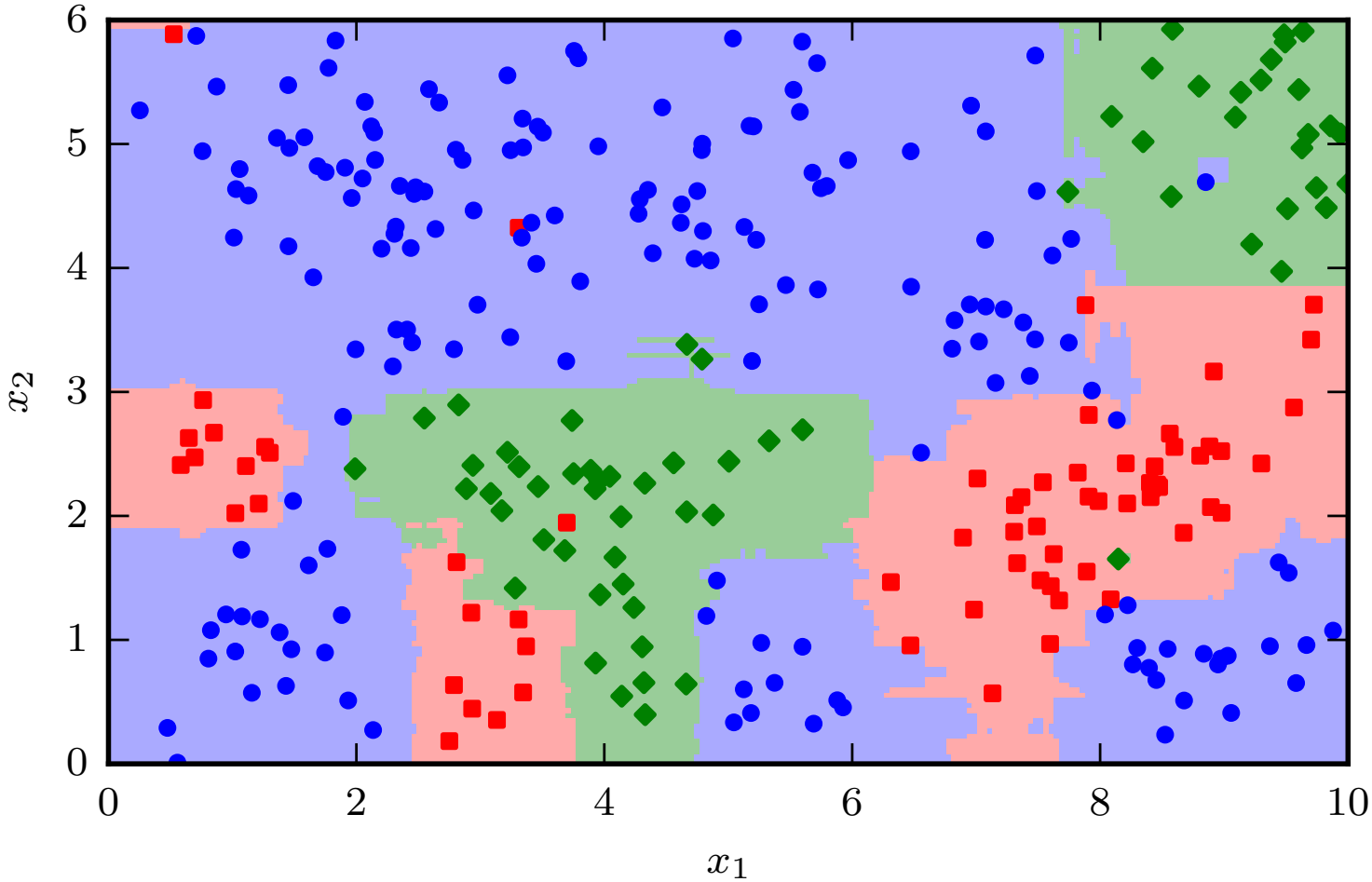
Random Forests: Setting the hyper parameter d

Choosing a small d creates variance between the trees, but if we choose d too small, we will build random trees with poor split choices and little individual predictive power.

Choosing d too large will create an army of very similar trees, such that there is hardly any advantage over a single tree.

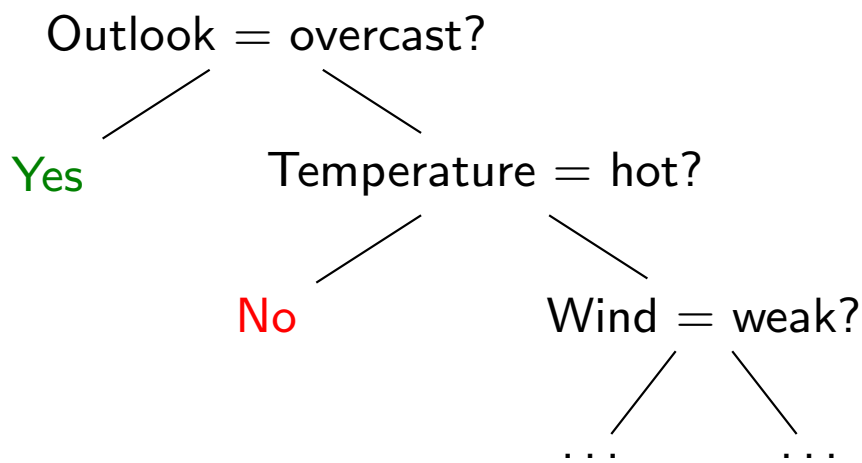
The answer is somewhere in between!

Decision surface for a random forest of 1000 trees



Decision Trees with Categorical Teatures

Day	Outlook	Temperature	Humidity	Wind	Play Tennis?
D1	sunny	hot	high	weak	No
D2	sunny	hot	high	strong	No
...					

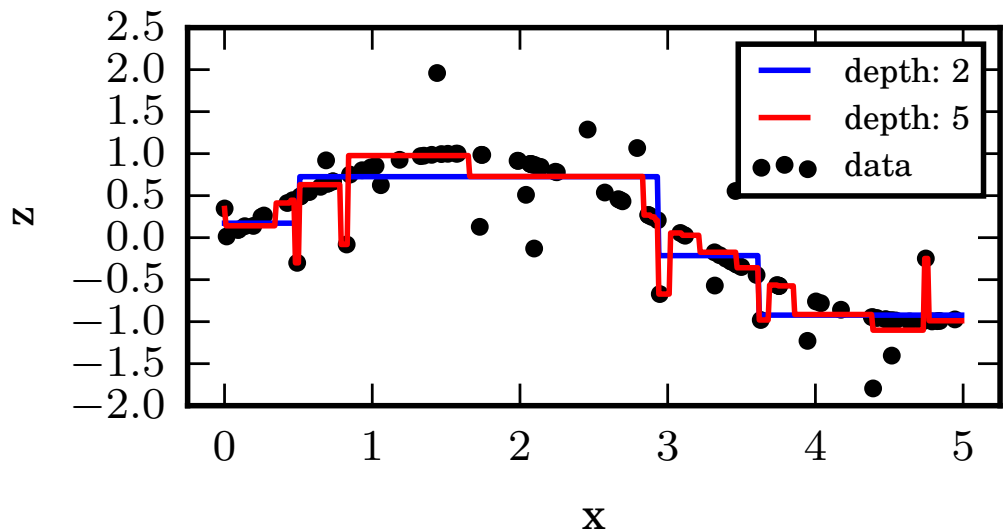


Different algorithm variants (ID3, C4.5, CART) handle these things differently.

Decision Trees for Regression

For regression (if z_i is a real value rather than a class):

- At the leaves compute the mean (instead of the mode) over the outputs.
- Use the mean-squared-error as splitting heuristic.



What we learned

- Interpretation and Building of Decision Trees
- Impurity functions / Splitting heuristics
- Training / Test / Validation Set
- Overfitting
- Random forests