Machine Learning

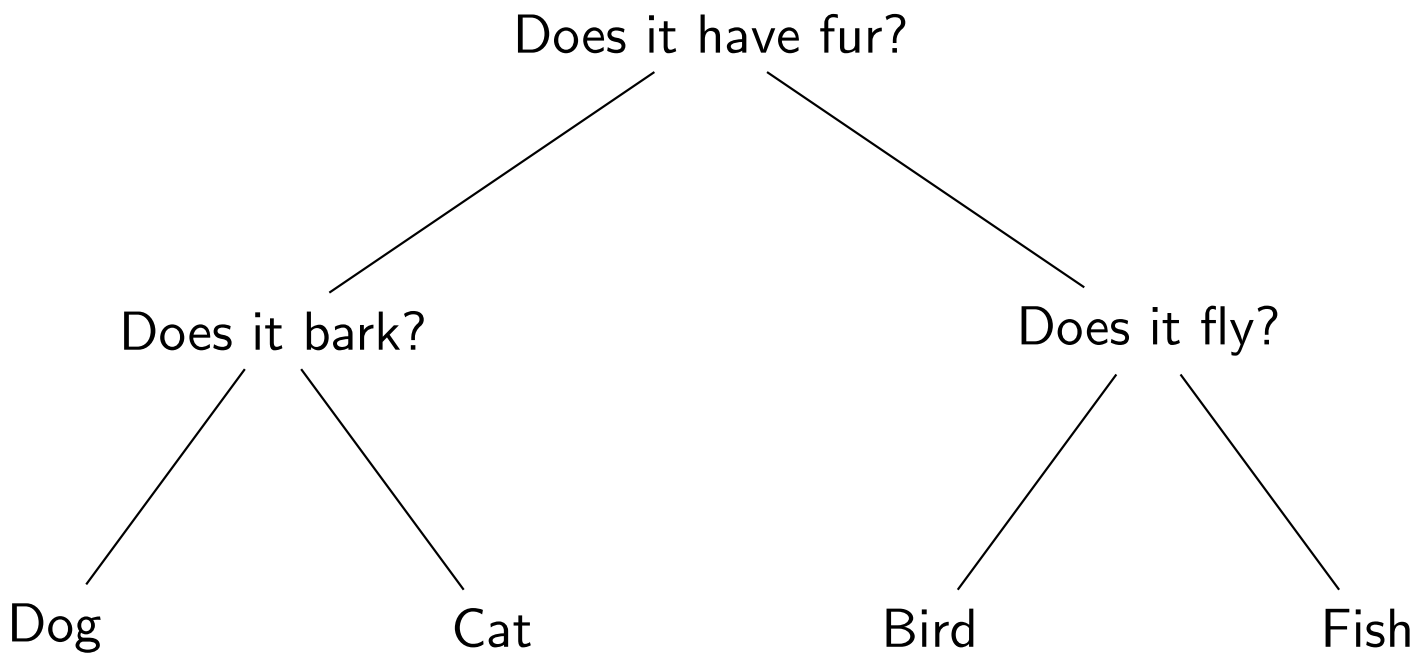
Lecture 2: Decision Trees

Prof. Dr. Stephan Günnemann

Data Analytics and Machine Learning
Technical University of Munich

Winter term 2023/2024

The 20-Questions Game



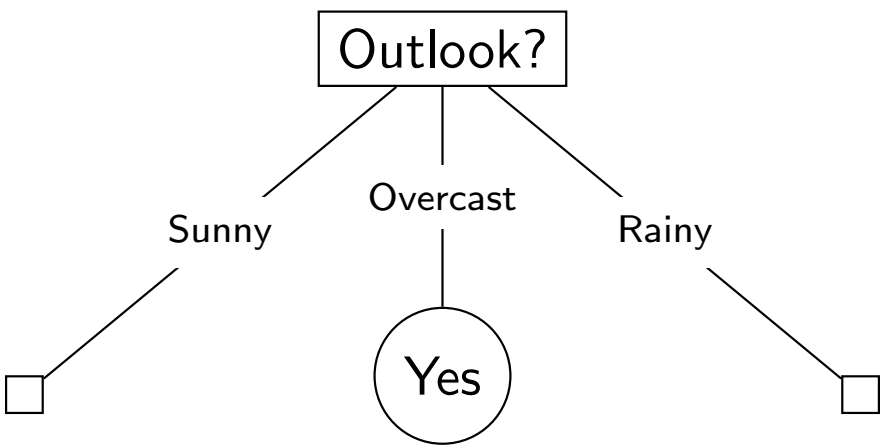
Tennis dataset

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

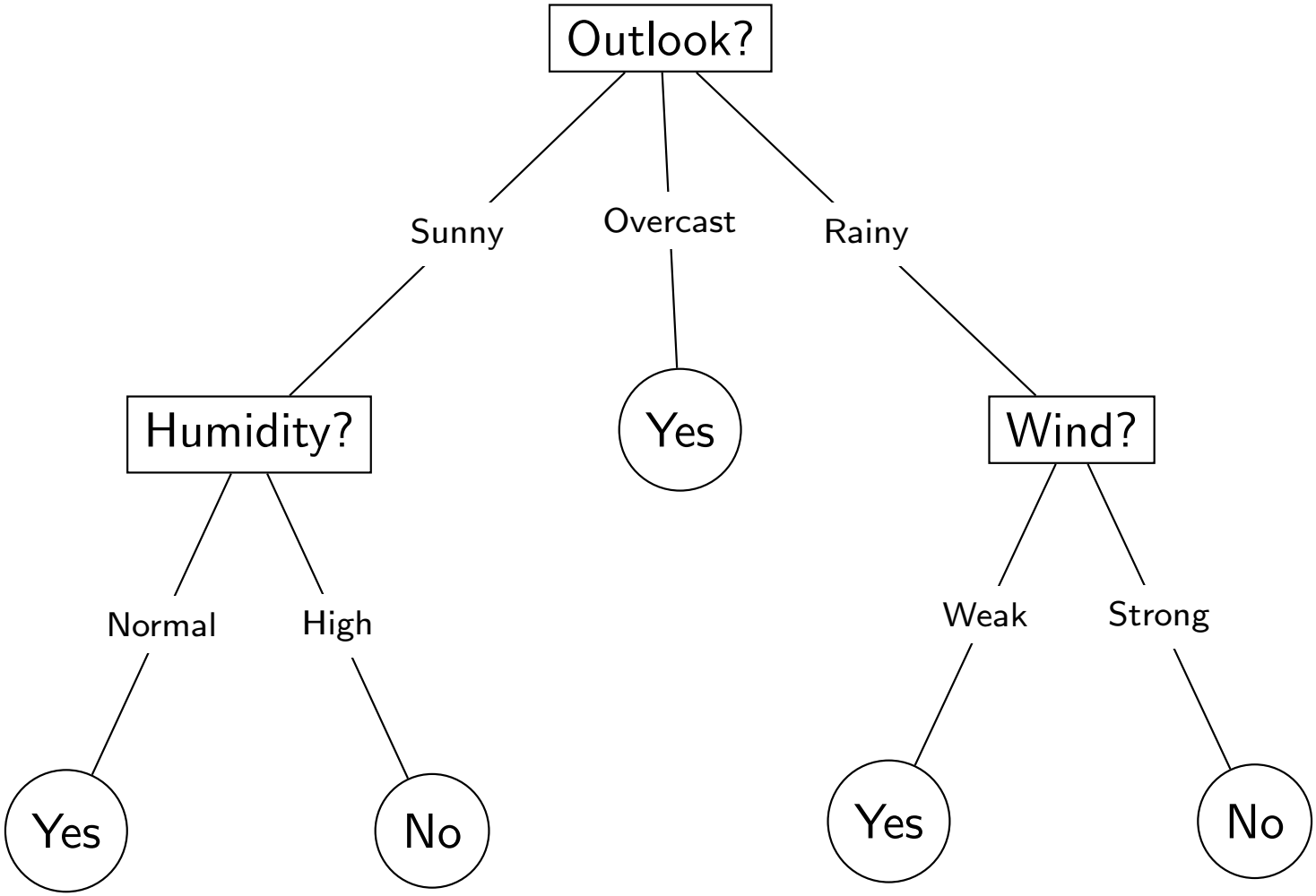
Goal: classification of unseen instances

Tennis dataset: decision tree

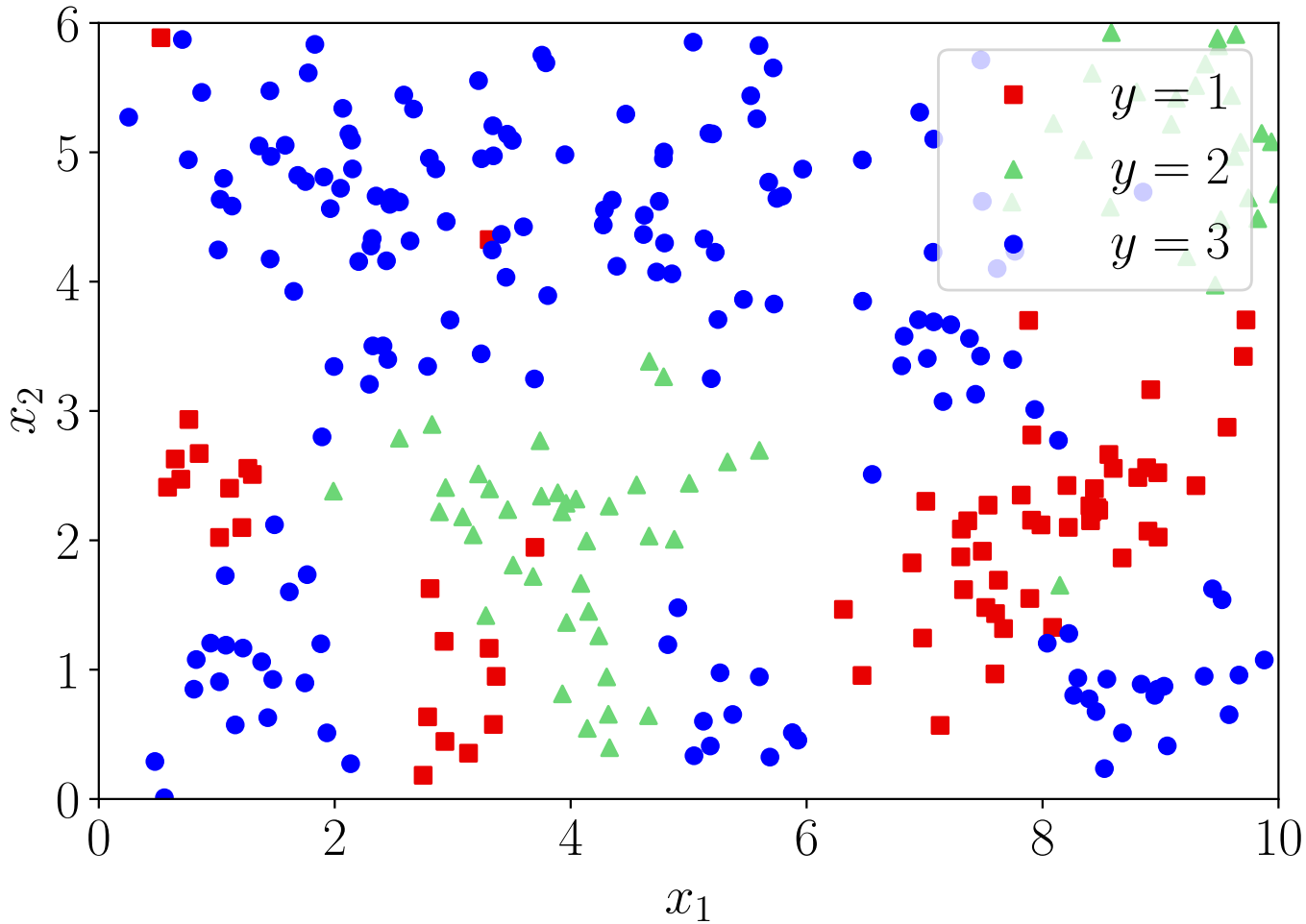
Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No



Tennis dataset: final decision tree

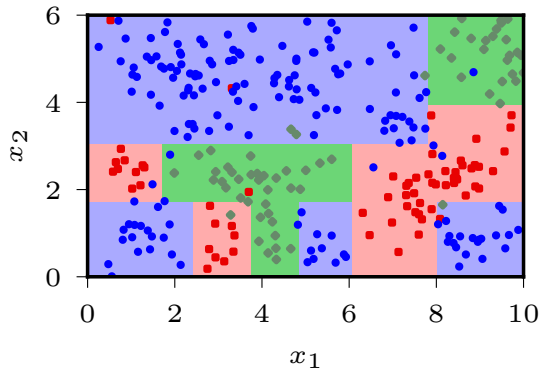


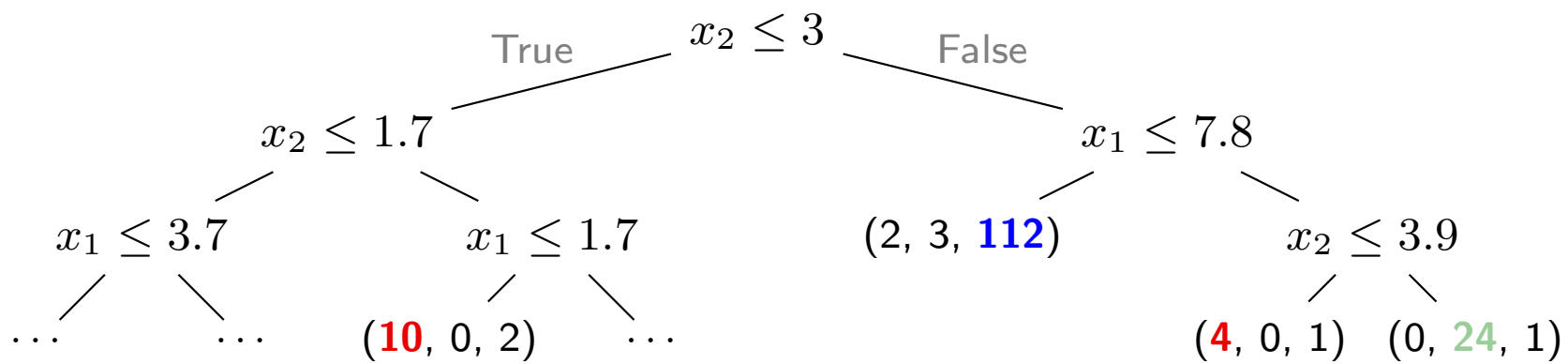
Numerical features



Example: data $\boldsymbol{X}$ with two *features* x_1 and x_2 and class labels y

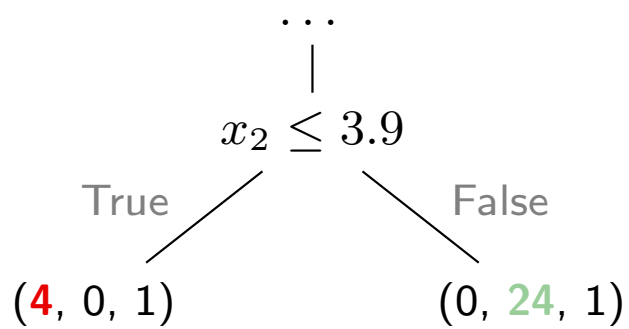
Binary split





Simplest decision: binary split on a single feature, $x_i \leq a$
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.

Inference on decision trees

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 $C = \{c_1, \ldots, c_k\}$.

[1] Majority label, similar to kNN

Inference on decision trees

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:

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

[1] Majority label, similar to kNN

Inference on decision trees

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:

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

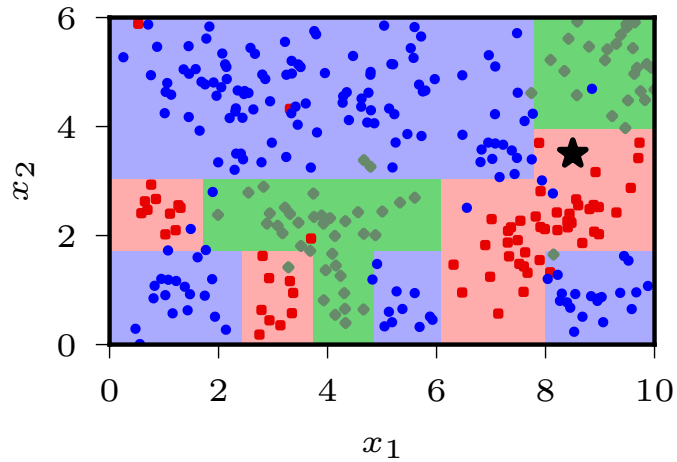
- A new unseen sample $\boldsymbol{x}$ is simply given the label which is most common[1] in its corresponding region:

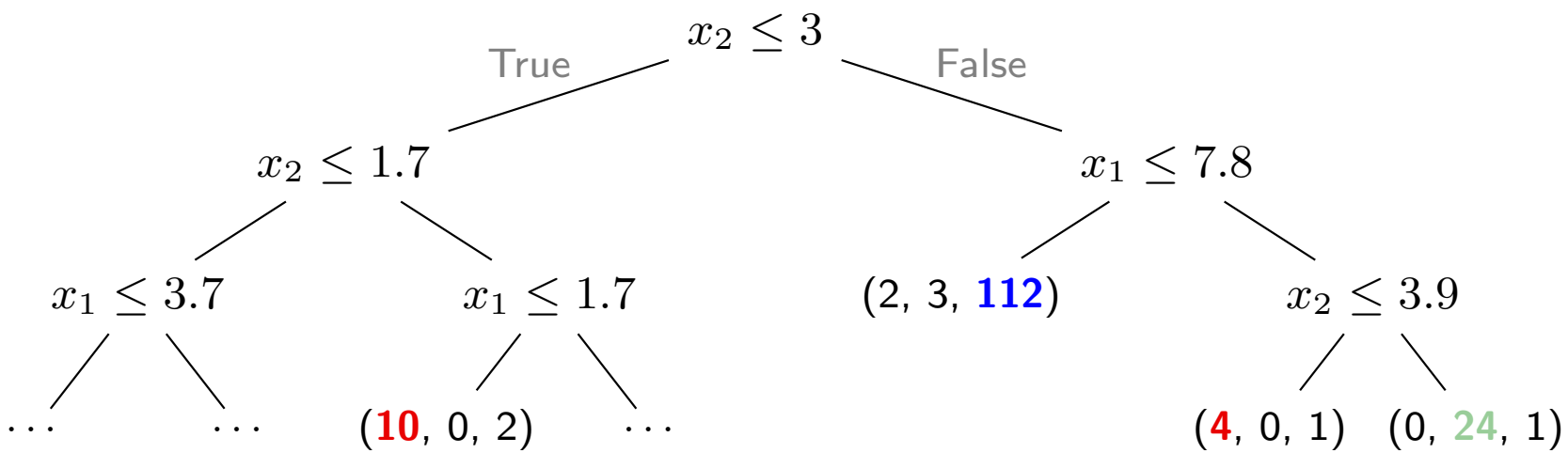
$$\hat{y} = \arg\max_c \; p(y = c \mid \boldsymbol{x}) = \arg\max_c \; p(y = c \mid \mathcal{R}) = \arg\max_c \; n_{c,\mathcal{R}}$$

[1] Majority label, similar to kNN

Example prediction

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

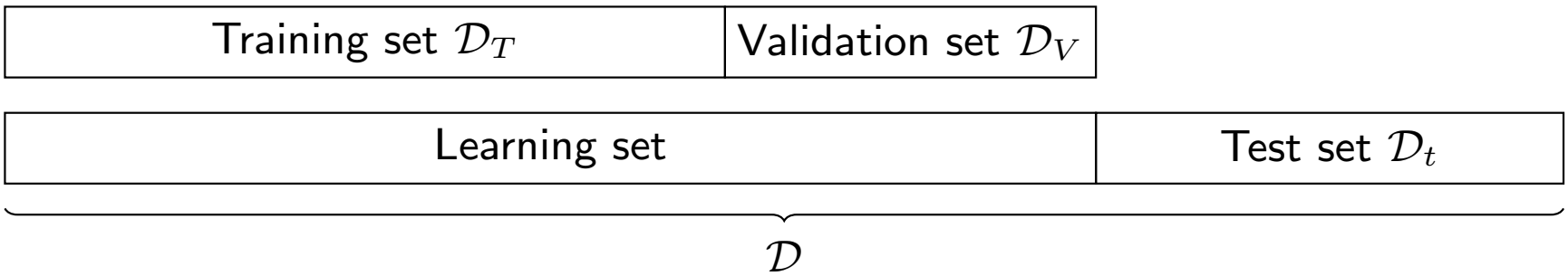




Optimal decision tree

Generalization: Find a DT that performs well on new (unseen) data.

Again, split the dataset:



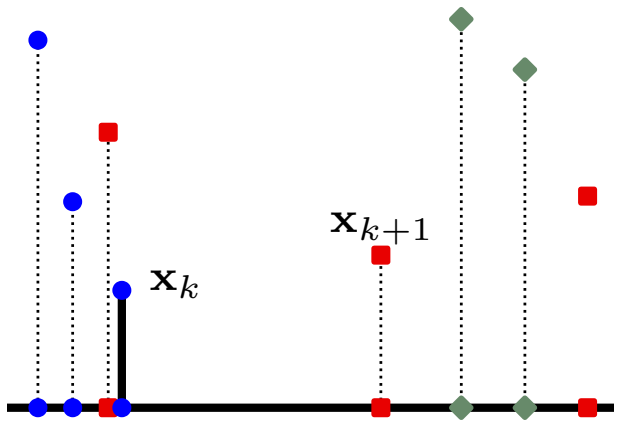
- build tree from training set $\mathcal{D}_T$,
- predict *validation set* labels $\hat{y}_i$ using the tree,
- evaluate by comparing predictions $\hat{y}_i$ to true labels y_i.
- pick the tree that performs the best on the validation set
- report final performance on the test set

Naive idea

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

Building the optimal decision tree is intractable

Iterating over all possible trees is possible only for very small examples because the number of trees quickly explodes.

Finding the optimal tree is *NP-complete*[2].

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

[2] Optimal in the sense of minimizing the expected number of tests required to classify an unknown sample. Even the problem of identifying the root node in an optimal strategy is NP-hard. And several other aspects of optimal tree construction are known to be intractable.

Example heuristic: misclassification rate

Split the node if it improves the misclassification rate (error) i_E at node t

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

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

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

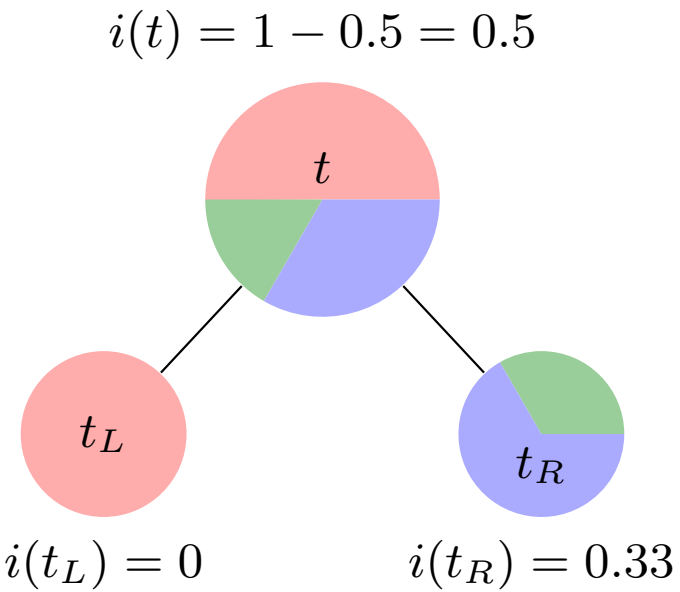
Example heuristic: misclassification rate

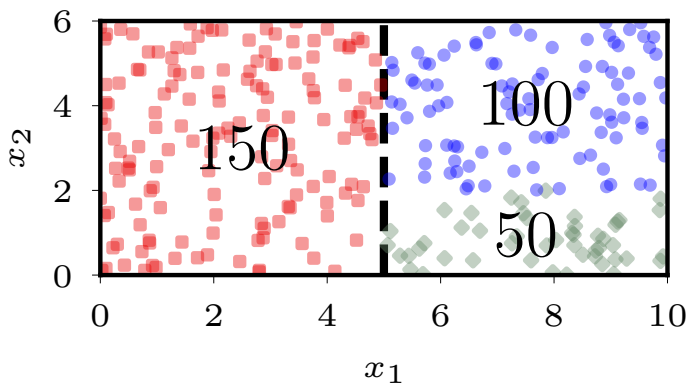
Split the node if it improves the misclassification rate (error) i_E at node t

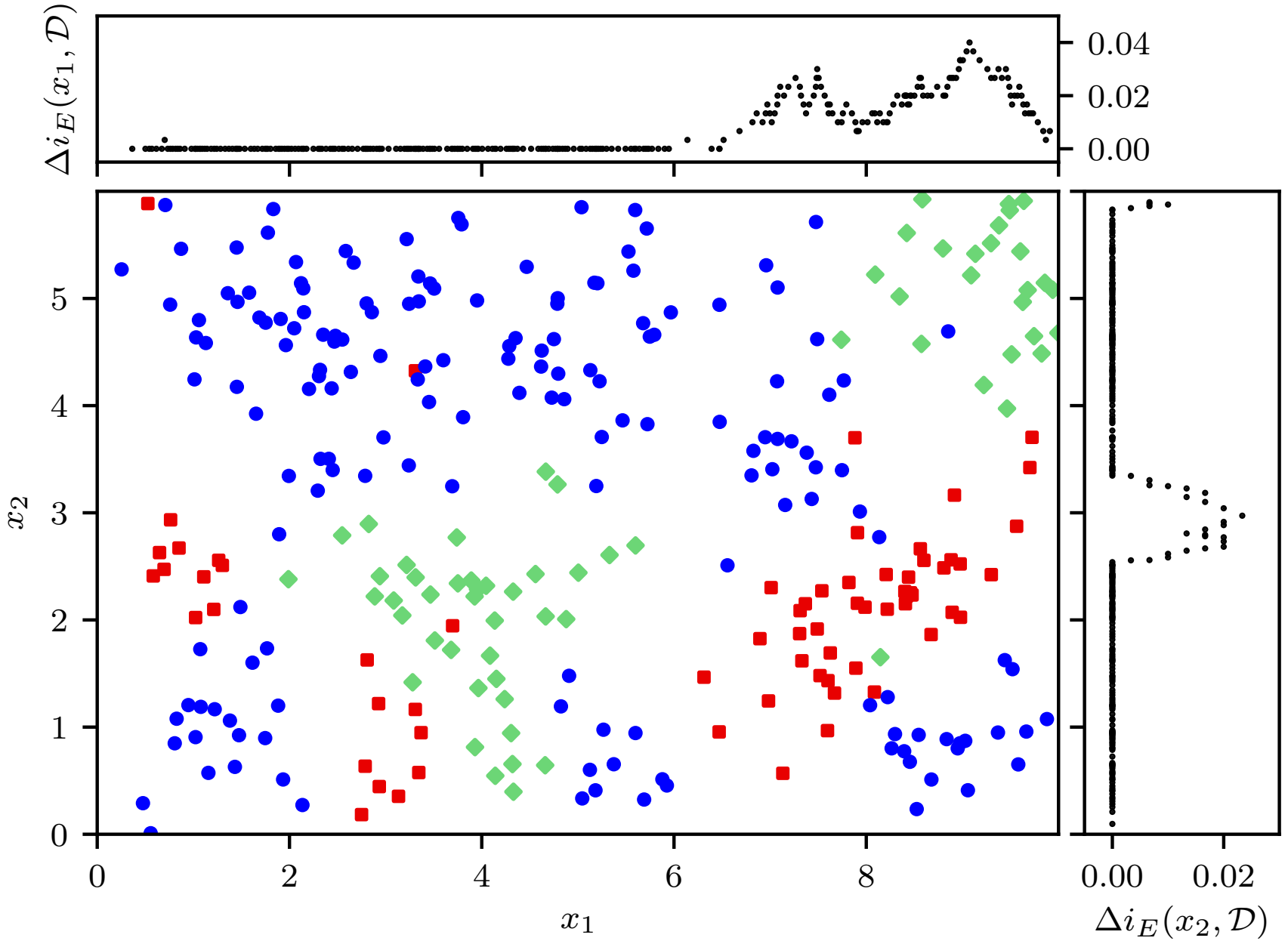
$$i_E(t) = 1 - \max_c \; p(y = c \mid t)$$

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

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

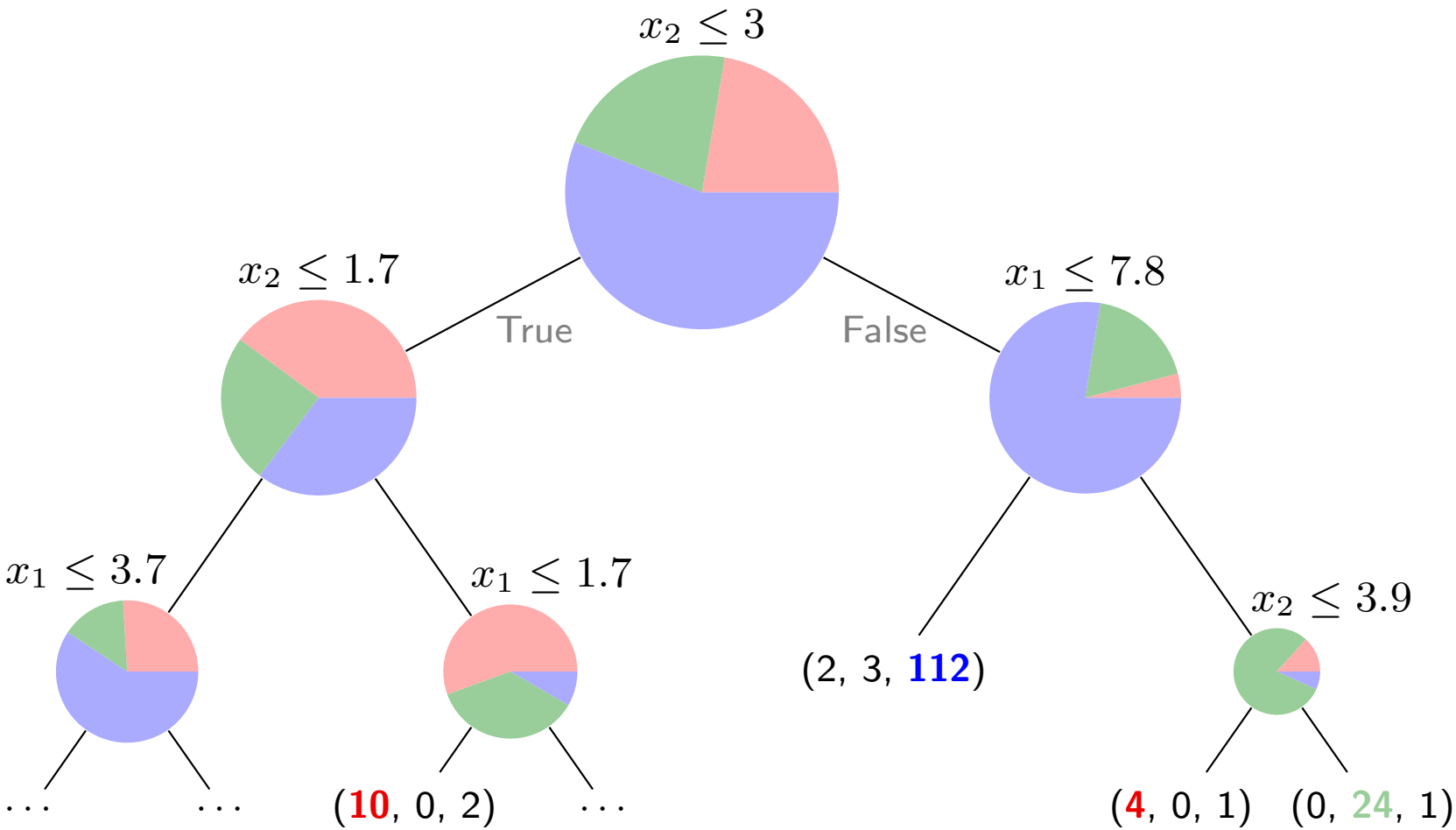





$\Delta i_E(x_1, \mathcal{D})$
0.04
0.02
0.00
x_2
0
1
2
3
4
5
0
2
4
6
8
x_1
0.00
0.02
$\Delta i_E(x_2, \mathcal{D})$

By repeatedly applying the heuristic

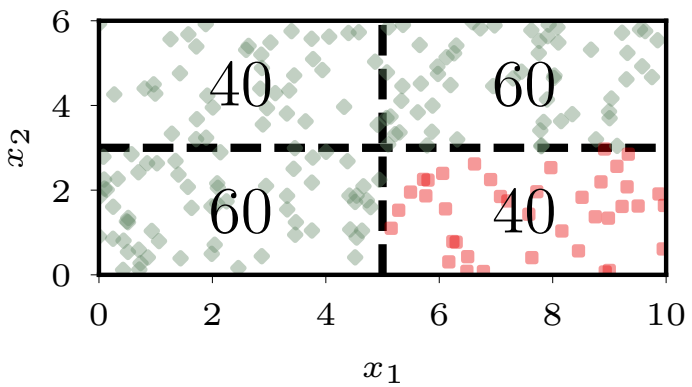
The distribution of labels becomes progressively more pure[3]...



[3] meaning we mostly have instances of the same class

Problems with misclassification rate

Problem 1: No split performed even though combining the two tests would result in perfect classification



no split: $i_E(t) = \frac{40}{200}$

$x_1 \leq 5$: $p_L \cdot i_E(t_L) + p_R \cdot i_E(t_R) = \frac{40}{200}$

$x_2 \leq 3$: $p_L \cdot i_E(t_L) + p_R \cdot i_E(t_R) = \frac{40}{200}$

Problem 2: No sensitivity to changes in class probability

Before split: $(400, 400)$
Split a: $\{(100, 300)\ ,\ (300, 100)\} \rightarrow i_E(t, a) = 0.25$
Split b: $\{(200, 400)\ ,\ (200, 0)\} \rightarrow i_E(t, b) = 0.25$

What is a suitable criterion

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 $\pi_c = p(y = c \mid t)$:

- Misclassification rate:

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

- Entropy:

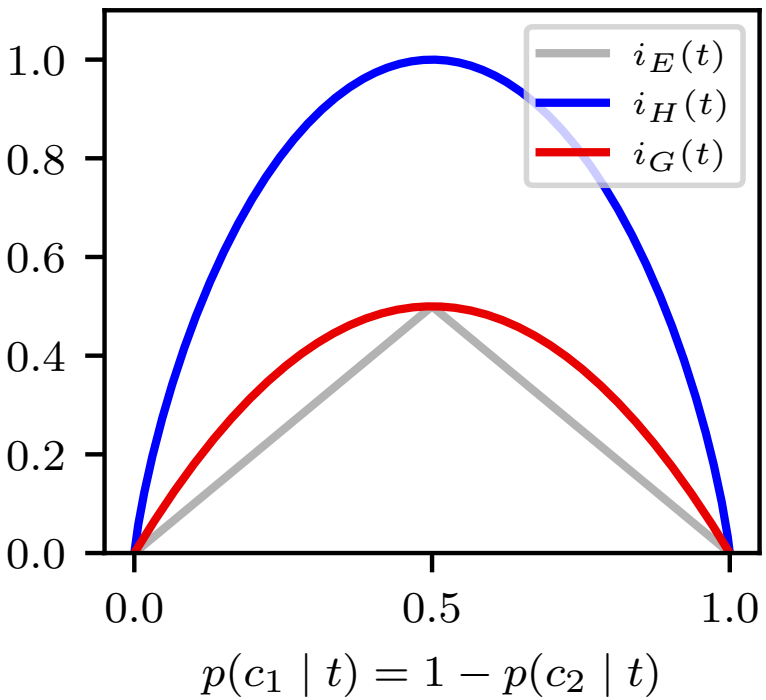
$$i_H(t) = -\sum_{c_i \in C} \pi_{c_i} \log_2 \pi_{c_i}$$

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

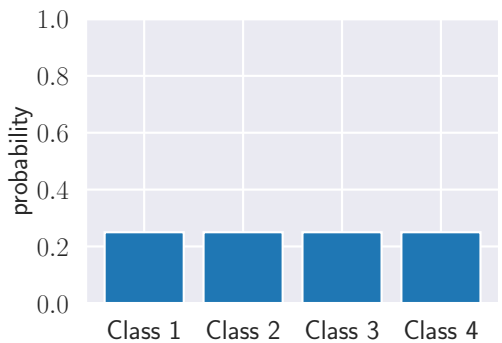


Shannon Entropy

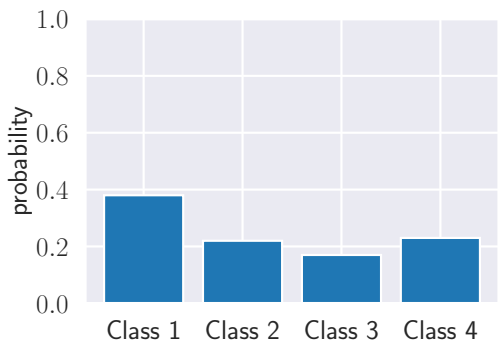
Expected number of bits needed to encode a randomly drawn value from a distribution (under most efficient code)

For a discrete random variable X with possible values $\{x_1, \ldots, x_n\}$

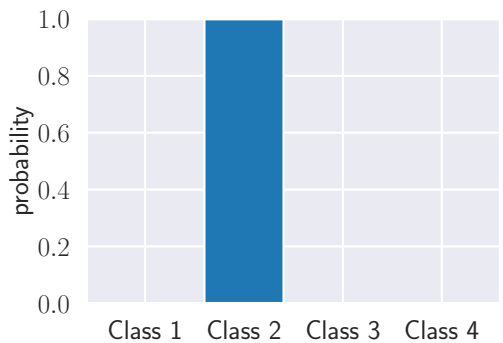
$$\mathbb{H}(X) = -\sum_{i}^{n} p(X = x_i) \log_2 p(X = x_i)$$



$\mathbb{H} = 2.0$



$\mathbb{H} = 1.9333$



$\mathbb{H} = 0.0$

Higher entropy → flatter histogram → sampled values less predictable
Lower entropy → peakier histogram → sampled values more predictable

Detour: Information Theory

Information theory is about encoding and transmitting information

We would like to encode four messages:

- $m_1 =$ "There is free beer."
- $m_2 =$ "You have an exam."
- $m_3 =$ "You have a lecture."
- $m_4 =$ "Nothing happening."

Detour: Information Theory

Information theory is about encoding and transmitting information

We would like to encode four messages:

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

Detour: Information Theory

Information theory is about encoding and transmitting information

We would like to encode four messages:

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

The code above is called a *Huffman Code*.

On average:

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

Gini Index

Measures how often a randomly chosen instance would be misclassified if it was randomly classified according to the class distribution

$$i_G(t) = \sum_{c_i \in C} \underbrace{\pi_{c_i}}_{\substack{\text{probability of} \\ \text{picking element}}} \cdot \underbrace{(1 - \pi_{c_i})}_{\substack{\text{probability is} \\ \text{misclassified}}}$$

Entropy vs Gini Index:

- It only matters in 2% of the cases which one you use.[4]
- Gini Index small advantage: no need to compute $\log$ which can be a bit faster

[4] See Raileanu LE, Stoffel K. Theoretical comparison between the gini index and information gain criteria.

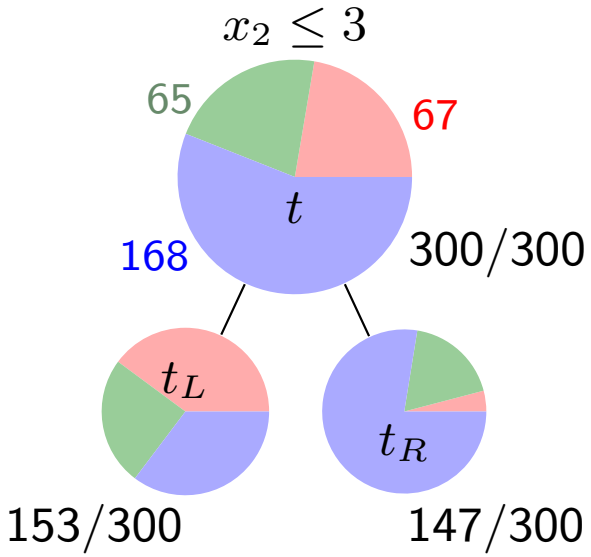
Building a decision tree

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

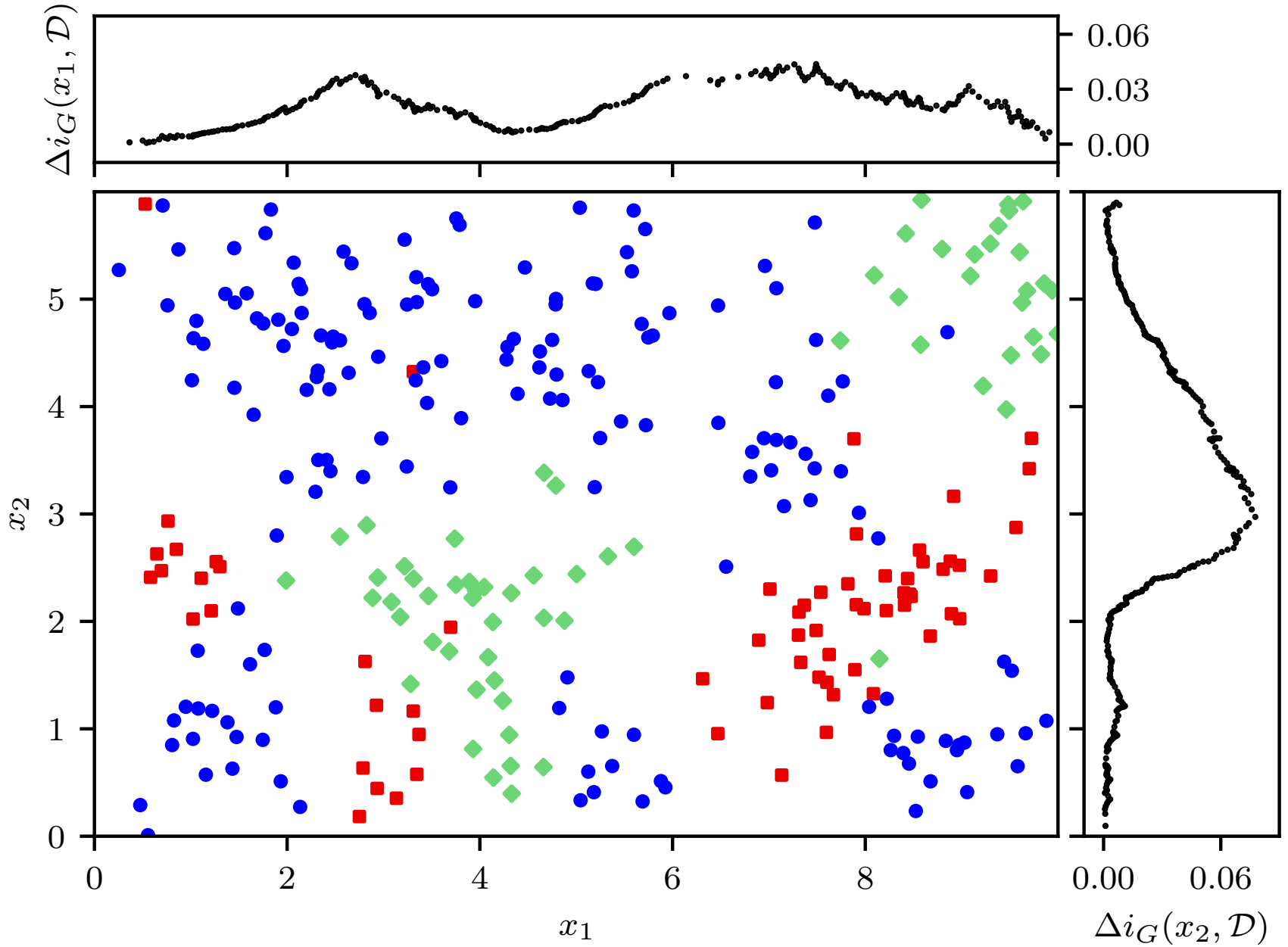
$$i_G(t) = 1 - \left(\frac{67}{300}\right)^2 - \left(\frac{65}{300}\right)^2 - \left(\frac{168}{300}\right)^2$$
$$\approx 0.5896$$

After testing $x_2 \leq 3$:

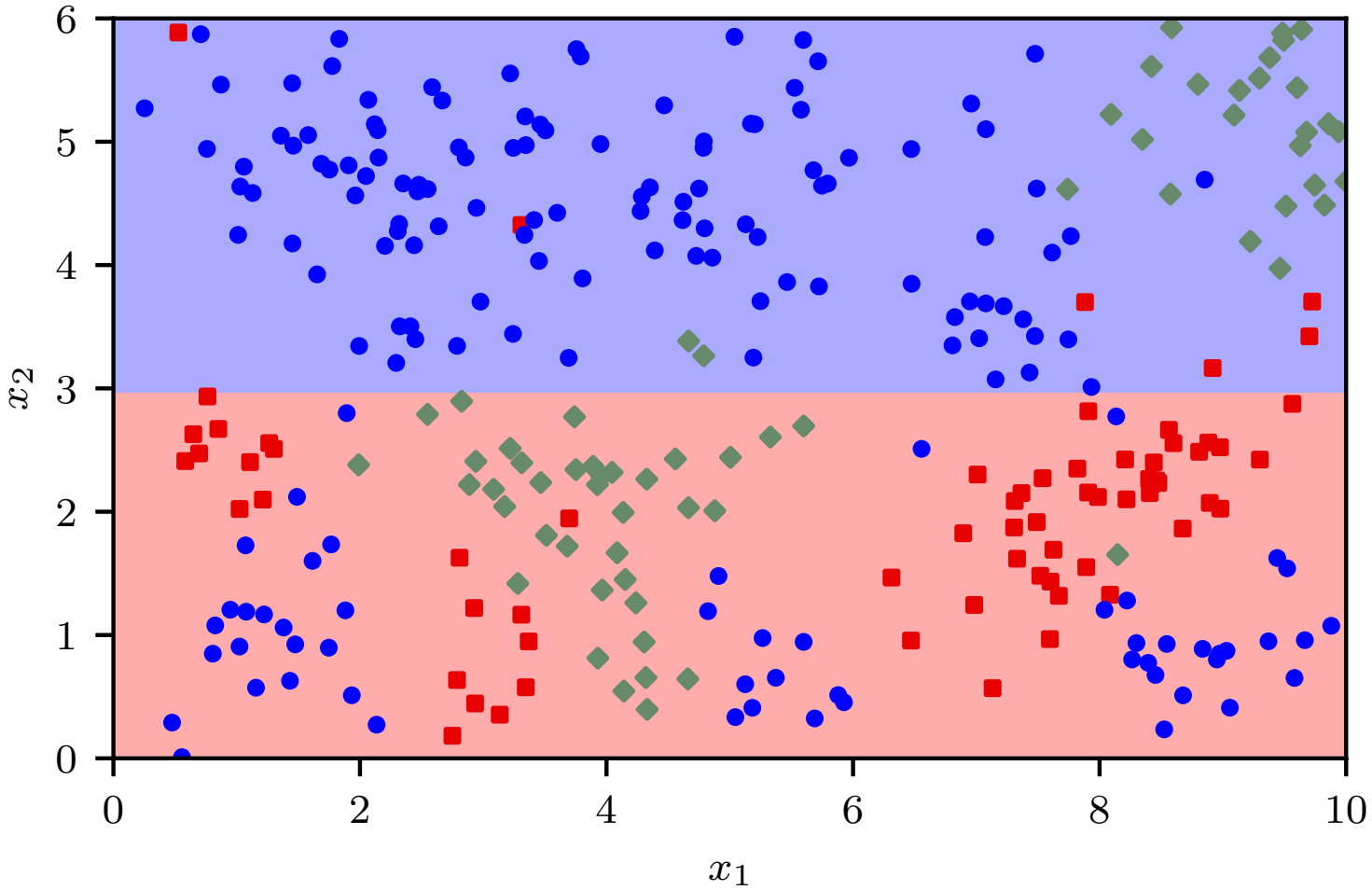
$i_G(t_L) \approx 0.6548$ and $i_G(t_R) \approx 0.3632$



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

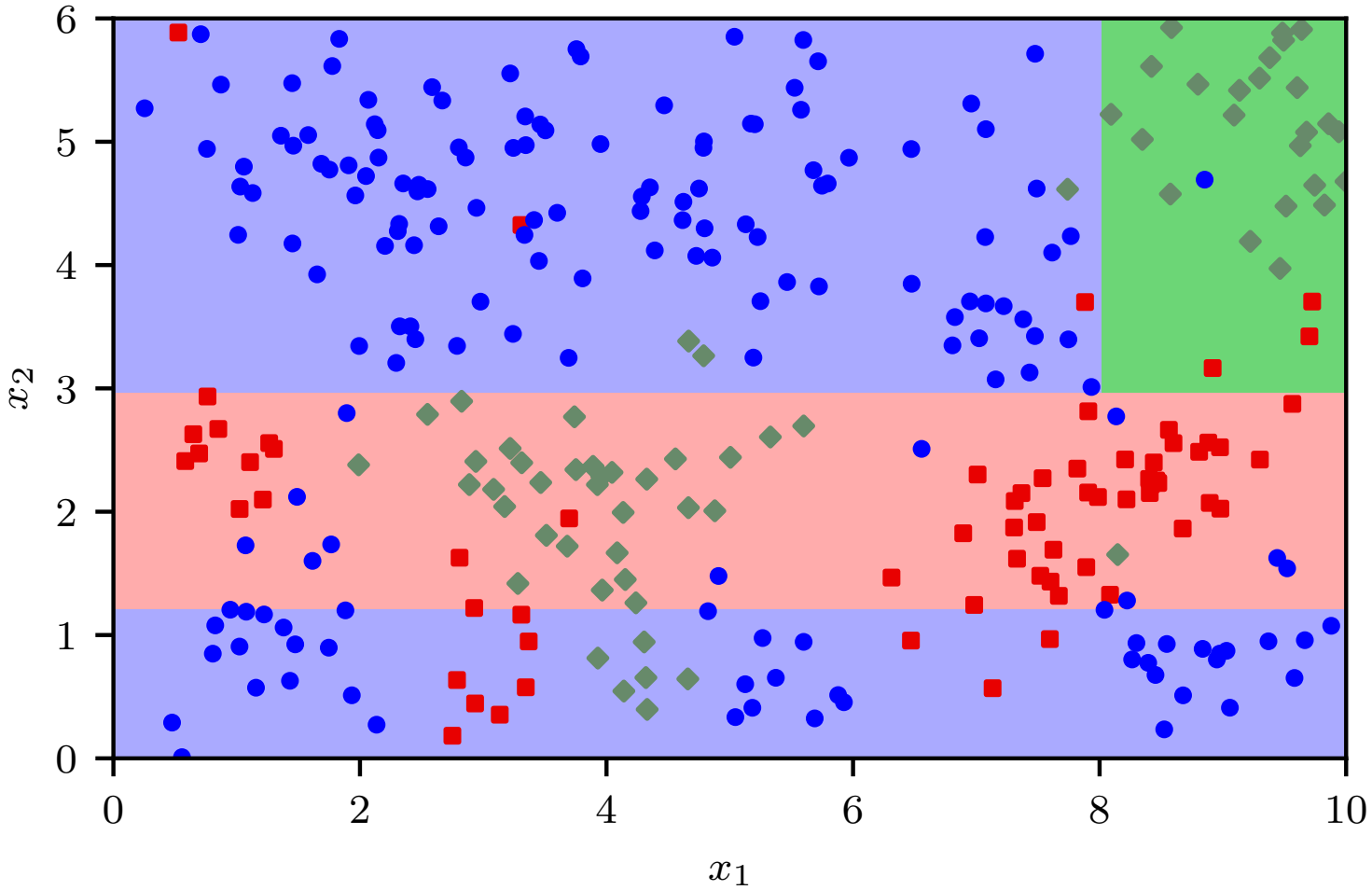

$\Delta i_G(x_1, \mathcal{D})$
0.06
0.03
0.00
x_2
0
1
2
3
4
5
0
2
4
6
8
x_1
0.00
0.06
$\Delta i_G(x_2, \mathcal{D})$

Decision boundaries at depth 1



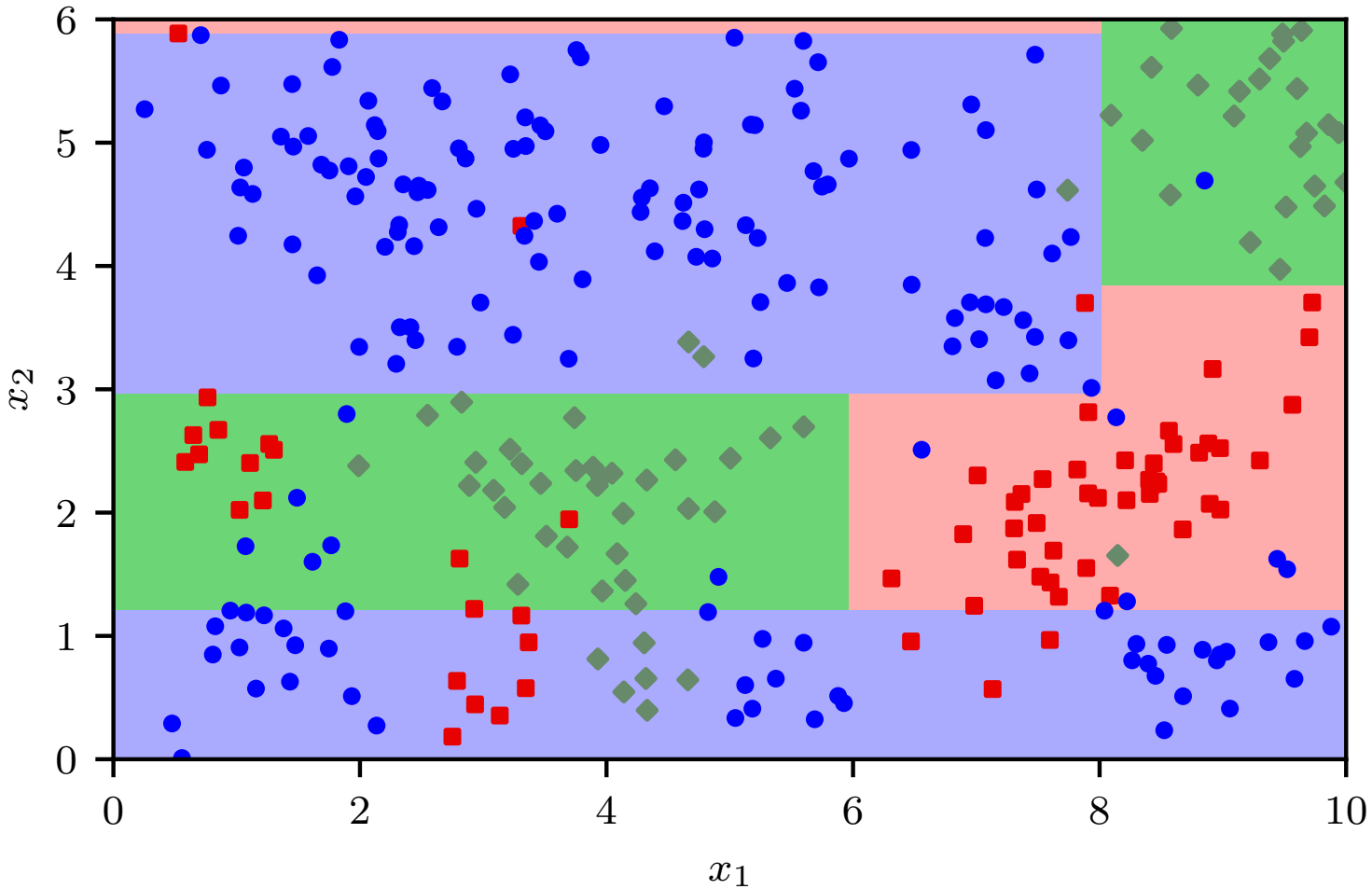
Accuracy on the whole data set: 58.3%

Decision boundaries at depth 2



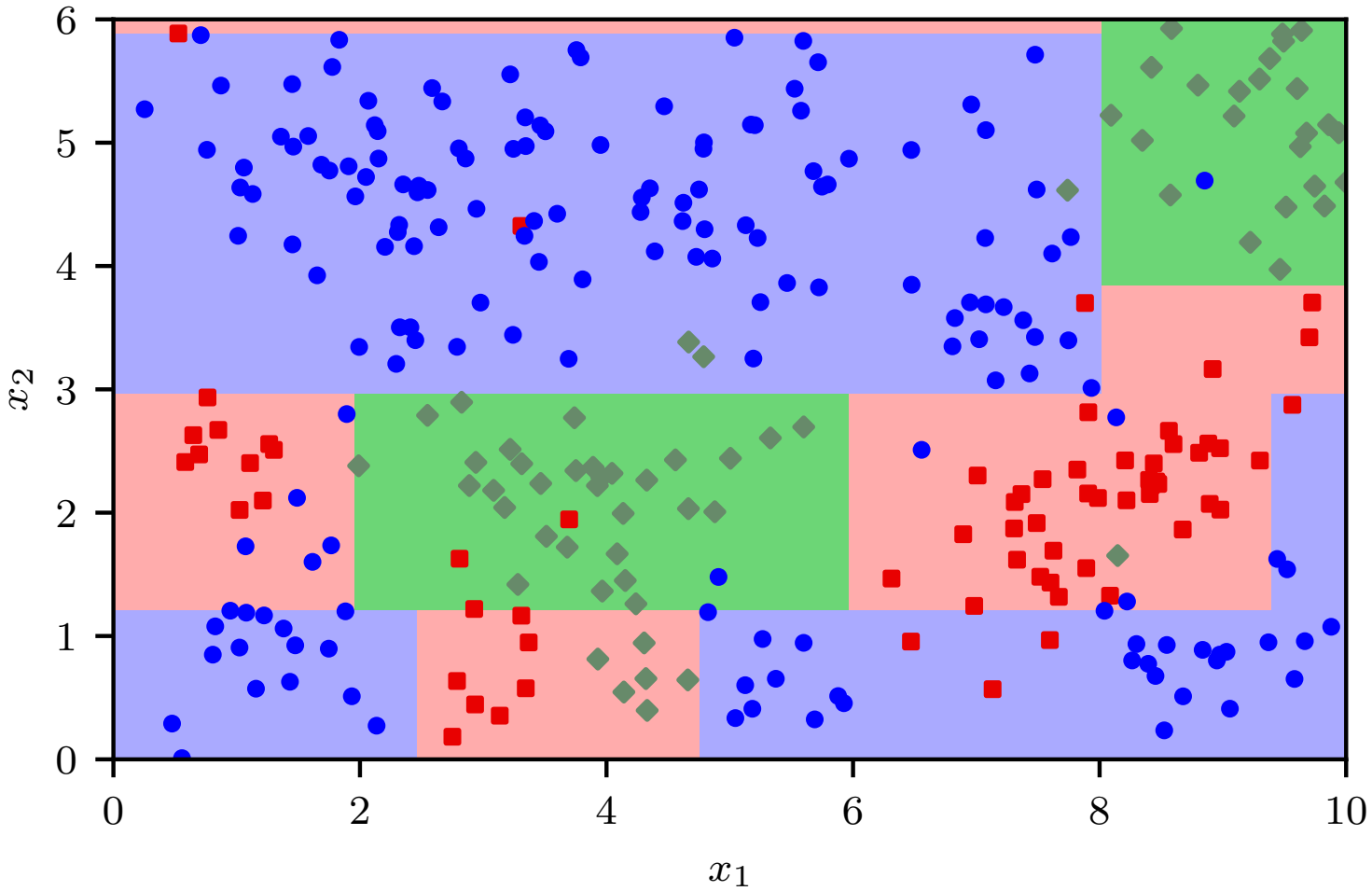
Accuracy on the whole data set: 77%

Decision boundaries at depth 3



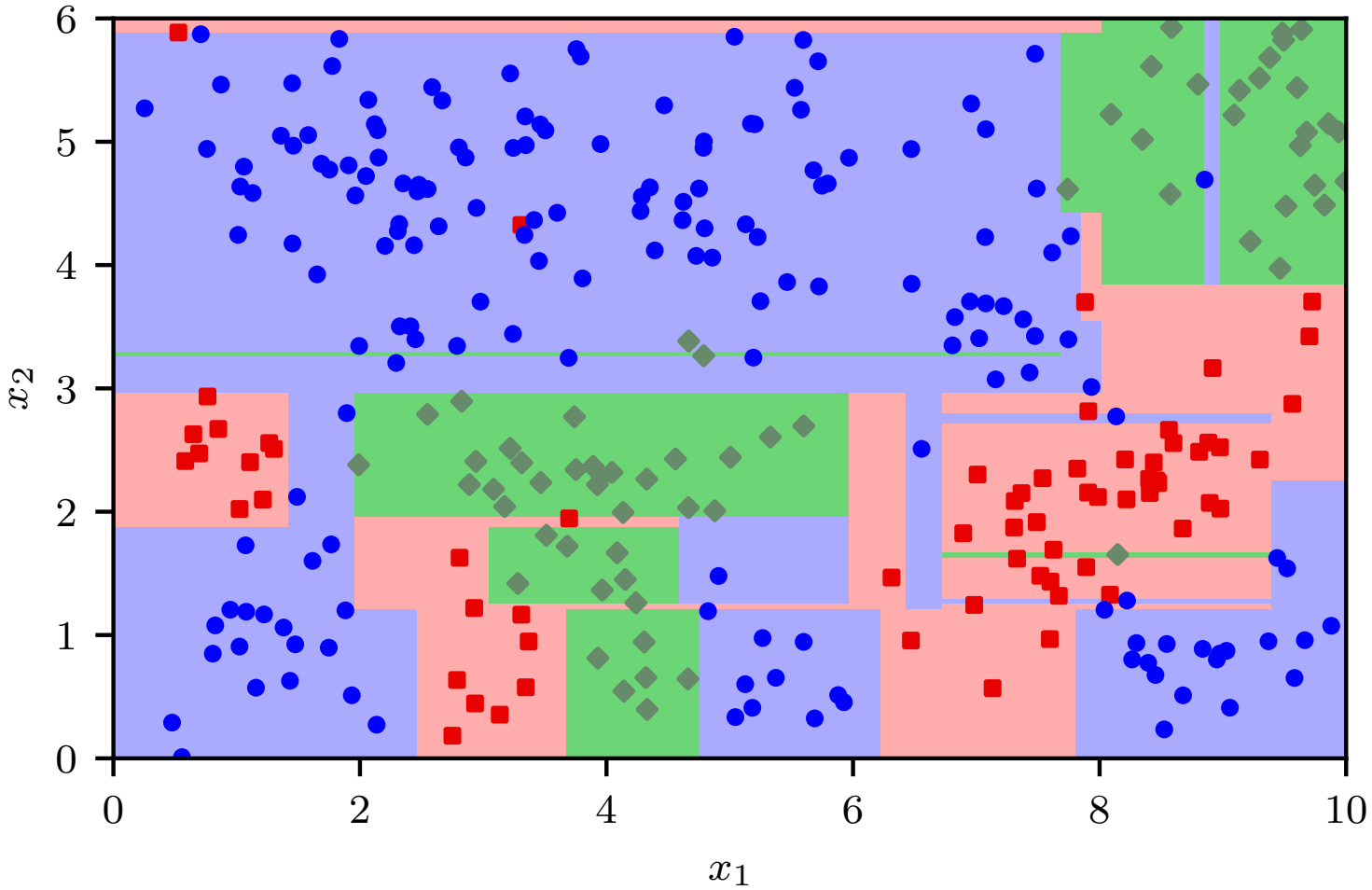
Accuracy on the whole data set: 84.3%

Decision boundaries at depth 4



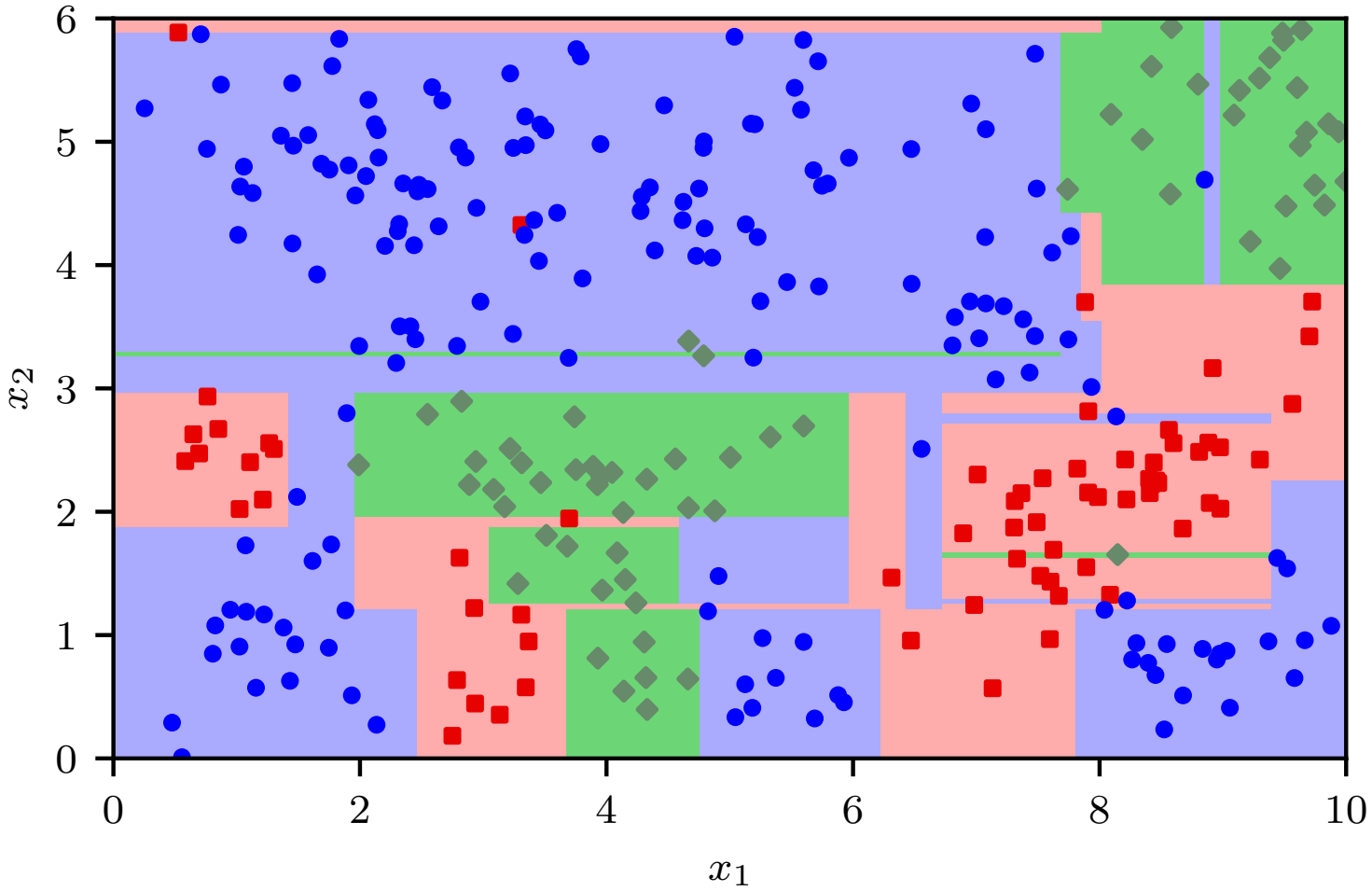
Accuracy on the whole data set: 90.3%

Decision boundaries of a maximally pure tree



Accuracy on the whole data set: 100%

Decision boundaries of a maximally pure tree



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

Overfitting

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

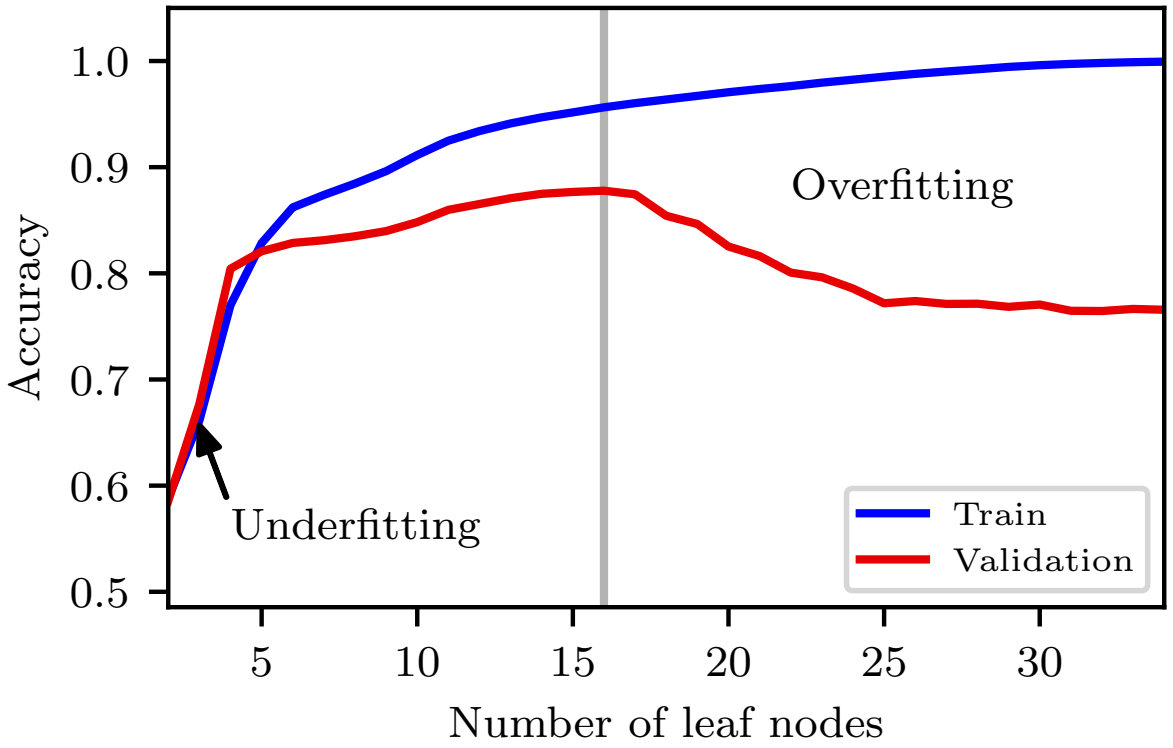
Overfitting means poor generalization!

Overfitting

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

Overfitting means poor generalization! How can we spot overfitting?

- low training error, possibly 0
- validation error is comparably high



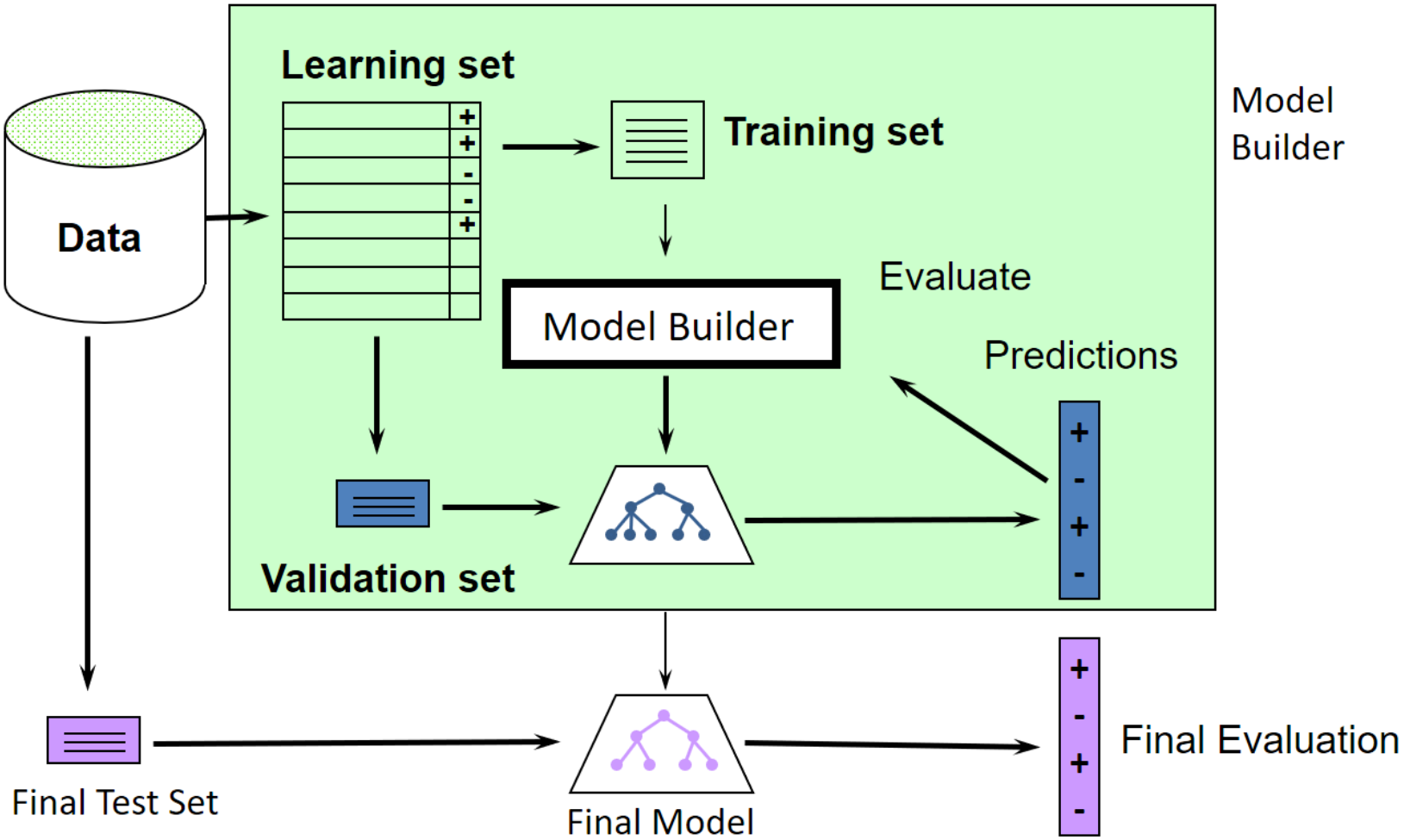
The training performance monotonically increases with every split.

The validation performance tells us how well our model generalizes, not the training performance![5]

[5] In practice, with increasing model size, data size, or training time performance first improves, then gets worse, and then improves again. This is known as the **double descent phenomenon**. It was recently observed and it is actively studied.

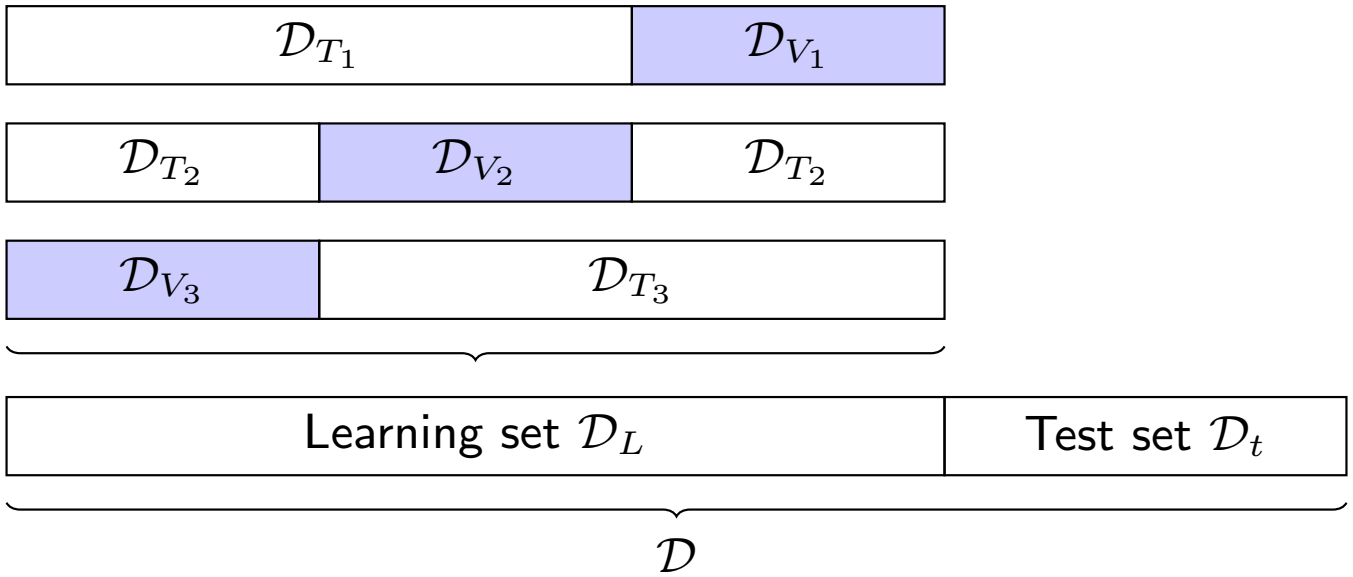
Good data science

How to do model selection / battle overfitting?

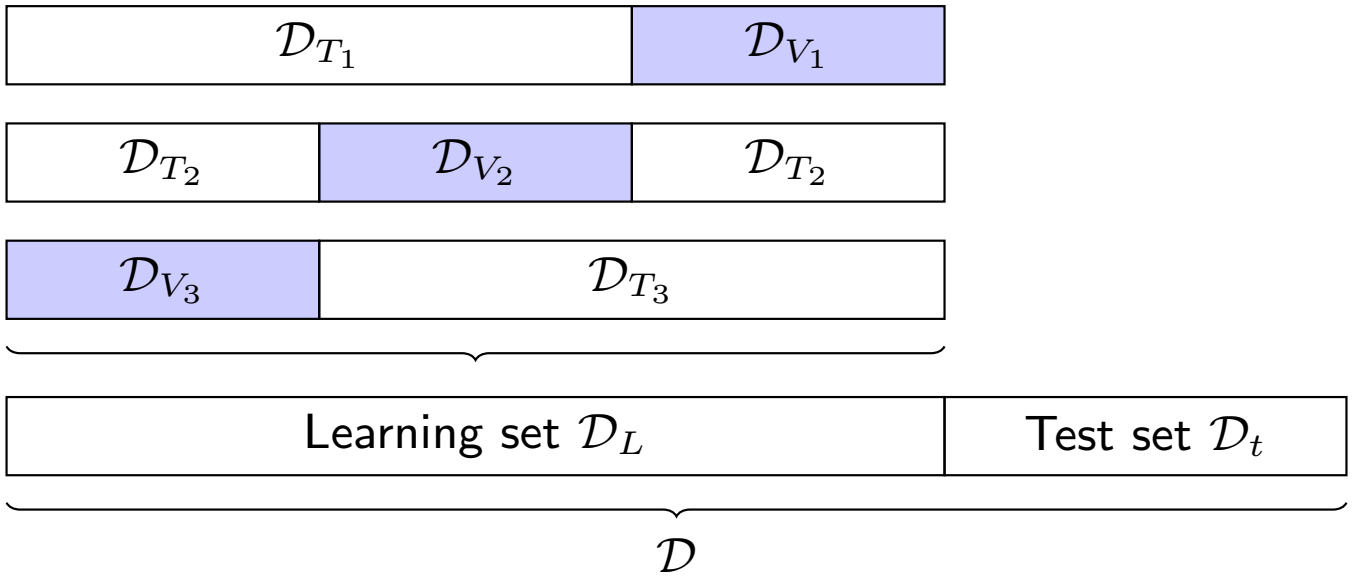


Only touch the test set once at the end to report final performance!

K-fold Cross-Validation



K-fold Cross-Validation



- Split your learning data into K folds (10-fold CV is common).
- Use $K-1$ folds for training and the remaining for evaluation.
- Average over all folds to get an estimate
  - of the error for a setting of your *hyper-parameters*
  - or the model for your model selection
- Try different settings for your hyper-parameters.
- Use all your training data and the best hyper-parameters for final training (and testing) of your model.

The extreme case - LOOCV

In *leave-one-out-cross validation* (LOOCV) we train on all but one sample.

If we have N samples, this is the same as N-fold cross-validation.

LOOCV is interesting if we do not have a lot of data and we want to use as much of it for training as possible but still get a good estimate of model performance.

The extreme case - LOOCV

In *leave-one-out-cross validation* (LOOCV) we train on all but one sample.

If we have N samples, this is the same as N-fold cross-validation.

LOOCV is interesting if we do not have a lot of data and we want to use as much of it for training as possible but still get a good estimate of model performance.

But it also means that we need to train our model N times...

If we have sufficiently large amounts of data and training our model is computationally expensive, we better stick to lower numbers of K or a single validation set.

Back to ...

... Decision Trees

Stopping criterion

We are recursively splitting the data, thereby growing the DT.

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 $\Delta i(s, t) < t_\Delta$
- accuracy on the validation set

Stopping criterion

We are recursively splitting the data, thereby growing the DT.

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 $\Delta i(s, t) < t_\Delta$
- accuracy on the validation set

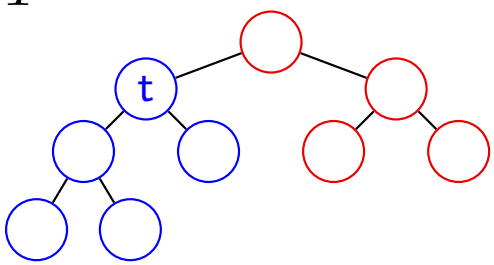
Or we can grow a tree maximally and then (post-)prune it.

Reduced error pruning

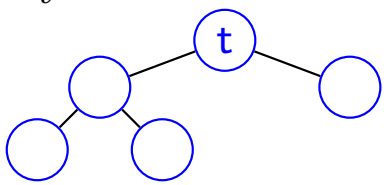
Let T be our decision tree and t one of its inner nodes.

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

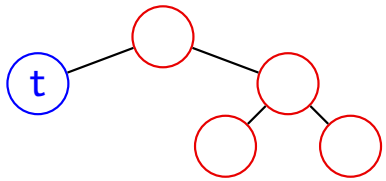
T



T_t



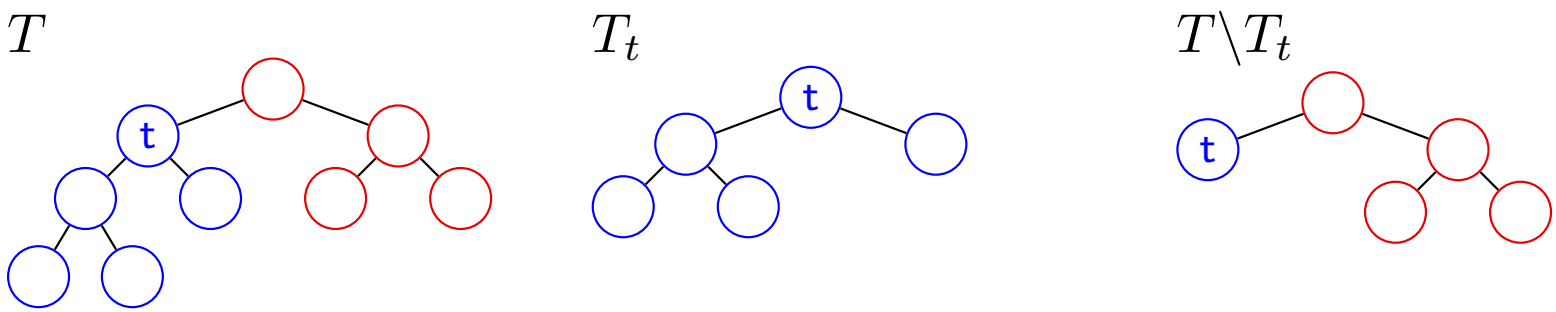
$T \backslash T_t$



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 $T \backslash T_t$.

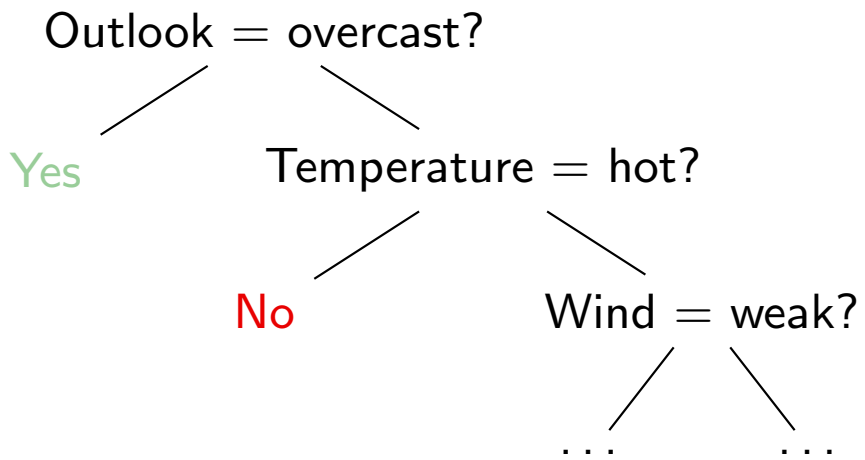


- Use validation set to get an error estimate: $\mathrm{err}_{\mathcal{D}_V}(T)$.
- For each node t calculate $\mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$
- Prune tree at the node that yields the highest error reduction.
- Repeat until for all nodes t: $\mathrm{err}_{\mathcal{D}_V}(T) < \mathrm{err}_{\mathcal{D}_V}(T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

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

Decision trees with categorical features

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

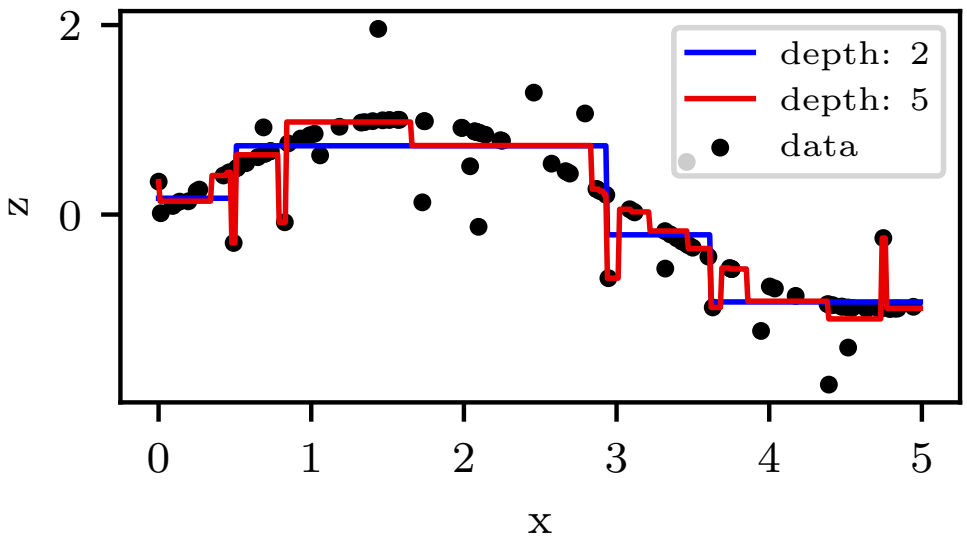


Different algorithm variants (ID3, C4.5, CART) handle these things differently.

Decision trees for regression

For regression (if y_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.



Considerations

- Human interpretable
- Can handle any combination of numerical and categorical features and targets
- Extensions (e.g. random forests, boosted trees) have very competitive performance (e.g. Kaggle competitions)
- Compared to k-NN:
  - Much better complexity w.r.t. memory/storage and inference
  - More flexible decision function

Ensembles

Main idea: Aggregate the predictions of many (diverse) classifiers to improve the performance.

Main benefit: Reduces the variance of the model by averaging.[6]

- Bagging (**b**ootstrap **agg**regat**ing**)
  - Create new datasets by sampling the training set (with replacement)
  - Train separate classifiers on each dataset
  - Combine the predictions, e.g. average or majority vote
- Boosting
  - Incrementally train (weak) classifiers that correct previous mistakes
  - Focus (give higher weight) on hard (misclassified) examples
- Stacking
  - Train a meta-classifier with the base classifiers' predictions as features
- Bucket of models; Bayesian model averaging; Bayes optimal classifier

[6] See Bias-variance tradeoff on the Linear Regression slides.

Bagging + Trees = Random Forests

- Bagged decision trees can be highly correlated
- This reduces the benefit of bagging since we need diverse classifiers

Idea: Use only a subset of randomly sampled features to learn each tree (bagging at instance level + bagging at feature level)

- Widely used in practice due to good "out-of-the-box" performance (not much tuning required)
- But less interpretable than simple decision trees
- Rule-of-thumb for the number of random features: $\log_2(d)$ for regression or $\sqrt{d}$ for classification where d is the number of features

Boosting: AdaBoost and XGBoost

Incrementally train "weak learners", e.g. decision stumps (one-level trees)

- Initialize a weight vector with uniform weights
- Loop:
  - Train weak learner on weighted examples (e.g. by weighted sampling)
  - Increase weight for misclassified examples
- Predict the (error-based weighted) majority

Adaptive (AdaBoost) and gradient (XGBoost) boosting differ mainly in how the weights are updated

- AdaBoost: closed-form weight updates based on the errors
- XGBoost: add one tree and one level in the tree at a time, greedily split based on the gradient w.r.t. a (custom) loss function

What we learned

- Interpretation and building of Decision Trees
- Impurity functions / Splitting heuristics
- Overfitting
- Good data science
- Ensembles

Reading material

Main reading

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

Extra reading

- "Pattern Recognition and Machine Learning" by Bishop [ch. 14.4]

Slides adapted from previous versions by W. Koepp & D. Korhammer. Also, some are inspired by *Understanding Random Forests* by G. Louppe.