

# Machine Learning

## Lecture 2: Decision Trees

---

Prof. Dr. Stephan Günnemann

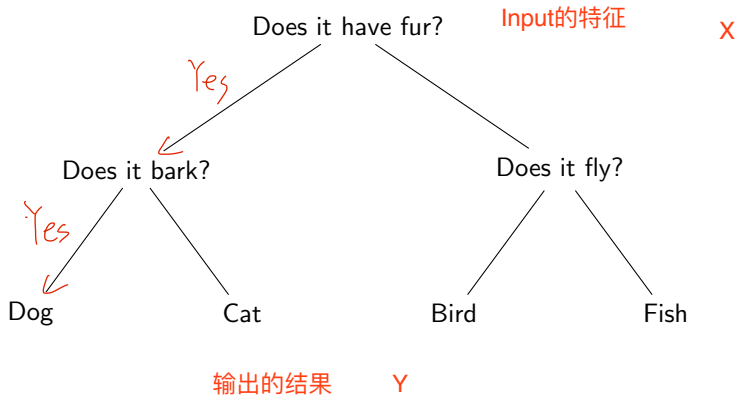
Data Analytics and Machine Learning  
Technical University of Munich

Winter term 2022/2023

# The 20-Questions Game

问问题，回答，得到答案

分类



# Tennis dataset

input feature →

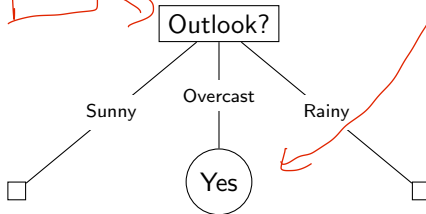
Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal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

← classed label

Goal: classification of unseen instances

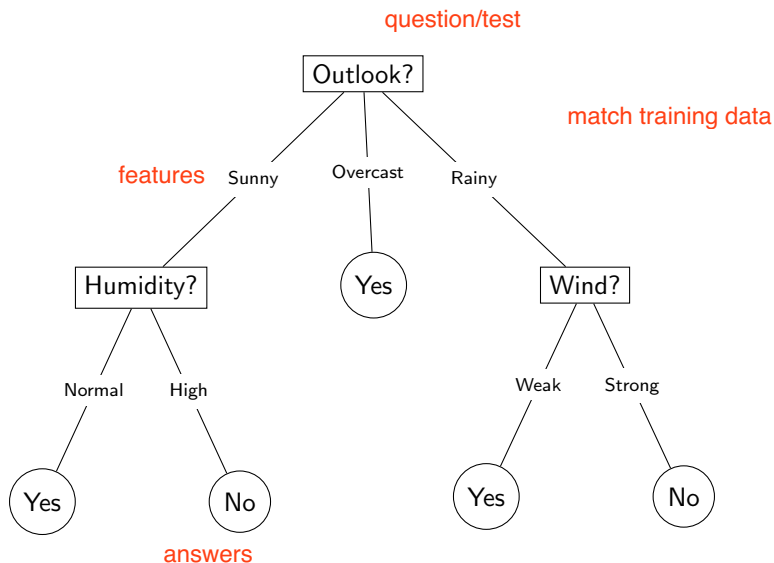
# Tennis dataset: decision tree

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

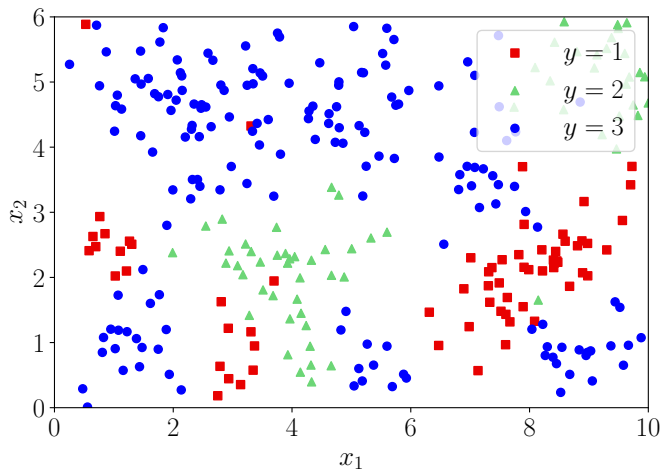


可以直接得出结果，因为Overcast

# Tennis dataset: final decision tree



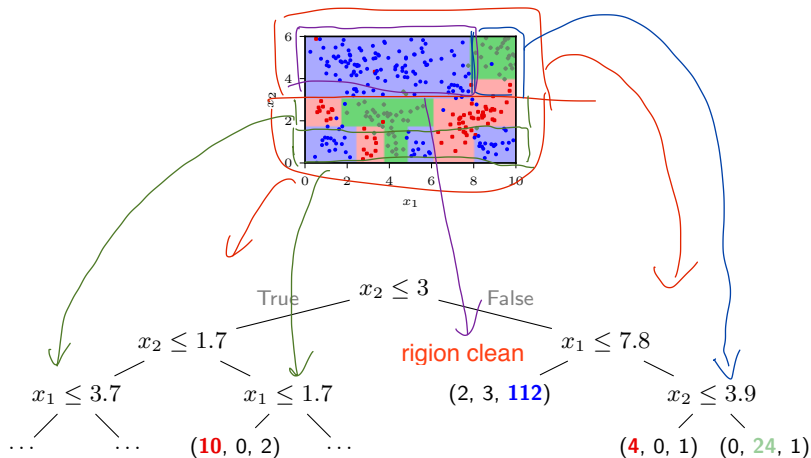
# Numerical features



Example: data  $X$  with two features  $x_1$  and  $x_2$  and class labels  $y$

# Binary split

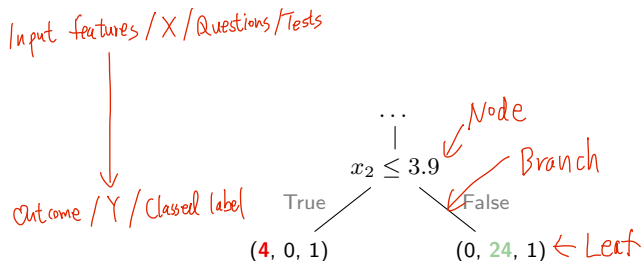
goal: find nice clean regions



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  $\hat{=}$  *feature* test  $\rightarrow$  leads to decision boundaries.
- Branch  $\hat{=}$  different outcome of the preceding feature test.
- Leaf  $\hat{=}$  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

Have a new instance, and want to predict the outcome

To classify a new sample  $x$ :

- Test the attributes of  $x$  to find the region  $\mathcal{R}$  that contains it and get the class distribution  $\mathbf{n}_{\mathcal{R}} = (n_{c_1, \mathcal{R}}, n_{c_2, \mathcal{R}}, \dots, n_{c_k, \mathcal{R}})$  for  $C = \{c_1, \dots, c_k\}$ .

Learning and training is to find a model to classify a new sample

---

<sup>1</sup>Majority label, similar to kNN

# Inference on decision trees

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

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

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

---

<sup>1</sup>Majority label, similar to kNN

# Inference on decision trees

决策树可以很好方便地预测新的数据，因为他只有一个模型，在本例子中，只需要回答三个问题就可以得到x的预测

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

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

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

Handwritten annotations for the equation above:

- Prob that a point should be in class  $c$  (pointing to  $p(y = c \mid \mathcal{R})$ )
- Number of point belonging to class  $c$  (pointing to  $n_{c, \mathcal{R}}$ )
- ← in the given region (pointing to  $n_{c, \mathcal{R}}$ )
- given the region of the input space (pointing to  $\mathcal{R}$ )
- $\sum_{c_i \in C} n_{c_i, \mathcal{R}}$  (pointing to the denominator)
- point of all classes in this region (pointing to the denominator)

- A new unseen sample  $\mathbf{x}$  is simply given the label which is most common<sup>1</sup> in its corresponding region:

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

<sup>1</sup>Majority label, similar to kNN

# Refresher: discrete probability theory

Given a jar that contains different colored balls  $\{4, 10, 6\}$ . What is the probability of randomly drawing a ball with a particular color (e.g. red)?

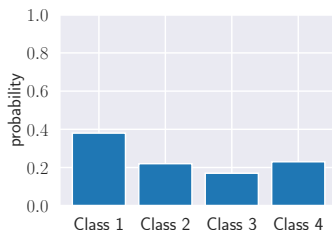
Intuitively:  $p(\text{ball} = \text{red}) = \frac{\text{number of red balls}}{\text{total number of balls}} = \frac{4}{4+10+6} = \frac{4}{20} = 0.2$

Similarly:  $p(\text{ball} = \text{green}) = 0.5$ ,  $p(\text{ball} = \text{blue}) = 0.3$

The probability mass function  $p$  assigns value to each possible outcome.

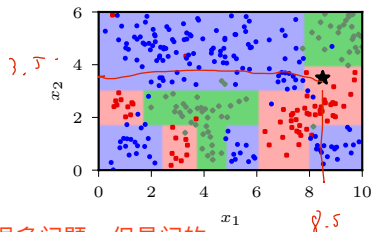
In general it has to hold:

- $\forall x, p(X = x) \geq 0$
- $\sum_x p(X = x) = 1$

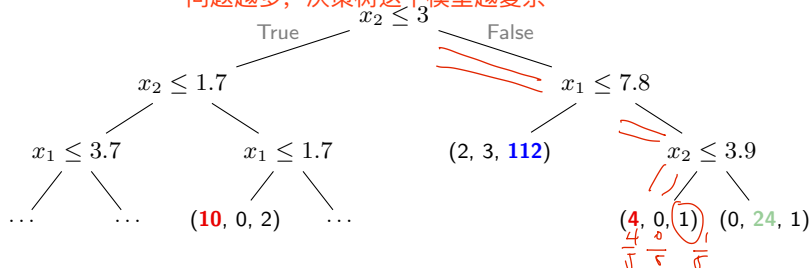


# Example prediction

Classification of  $x = (8.5, 3.5)^T$



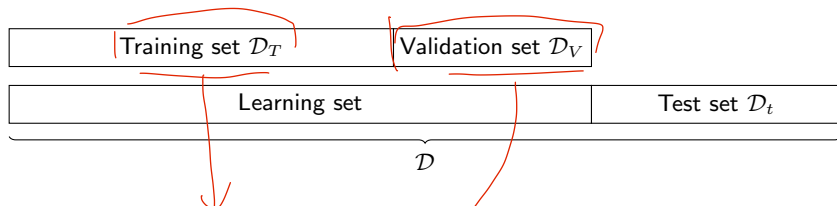
在决策树中，可以问很多问题，但是问的问题越多，决策树这个模型越复杂



# 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

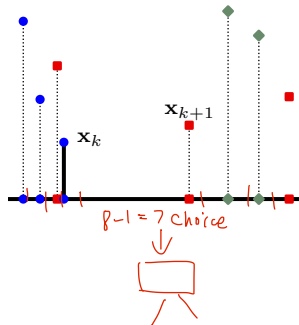
# Naive idea

too many tree

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
$x_1$	$\leq 0.36457631$
	$\leq 0.50120369$
	$\leq 0.54139549$
	$\leq \dots$
	$\leq \dots$
$x_2$	$\leq 0.09652214$
	$\leq 0.20923062$
	$\leq \dots$



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

$O(N^2)$

# 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*<sup>2</sup>.

too hard to practise use

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

分部分建立决策树

<sup>2</sup>最优是指最小化对未知样本进行分类所需的预期测试次数。即使是识别最优策略中的根节点的问题也是NP-hard。而且已知最优树构建的其他几个方面也是难以解决的。

---

<sup>2</sup>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

⇒ p24 ✓

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

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

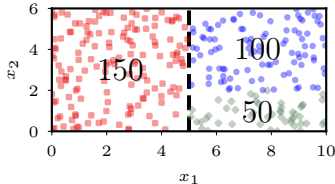
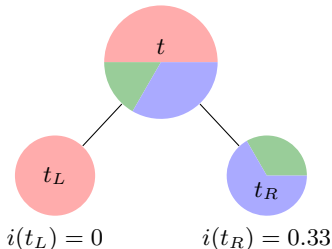
The improvement when performing a split  $s$  of  $t$  into  $t_R$  and  $t_L$  for  $i(t) = i_E(t)$  is given by

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

$\underbrace{\max_c p(y=c|t)}_{\substack{\text{volume} \\ \text{size of left and right}}} \quad \underbrace{\max_c p(y=c|t)}$

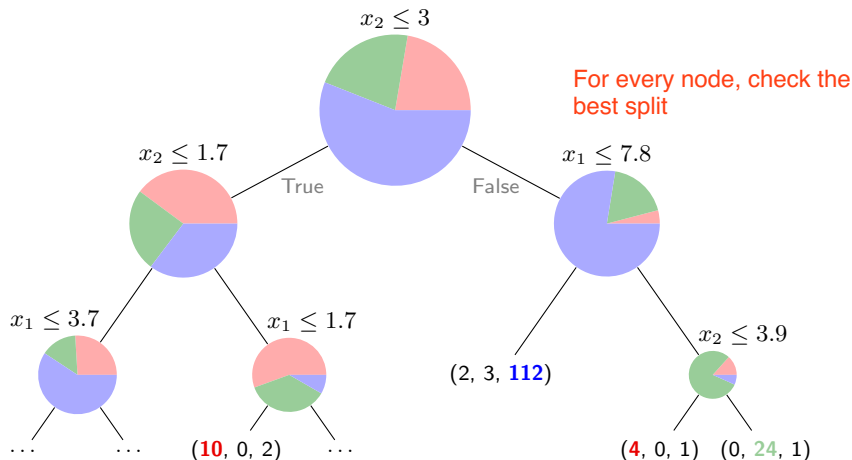
$\frac{150}{150} \quad \frac{0}{150} \quad \frac{0}{150} \quad \frac{0}{150} \quad \frac{100}{150} \quad \frac{50}{150}$

$$i(t) = 1 - 0.5 = 0.5$$



# By repeatedly applying the heuristic

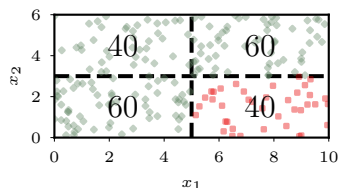
The distribution of labels becomes progressively more pure<sup>3</sup>...



<sup>3</sup>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



outcome is equal /  $\Delta = 0$  / No benefit

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:  $\{(200, 400), (200, 0)\} \rightarrow i_E(t, b) = 0.25$

$$p_L i_E(t_L) + p_R i_E(t_R) = \begin{cases} \frac{400}{800} (1 - \frac{300}{400}) + \frac{400}{800} (1 - \frac{300}{400}) = \frac{1}{4} \\ \frac{600}{800} (1 - \frac{400}{600}) + \frac{200}{800} (1 - \frac{0}{200}) = \frac{1}{4} \end{cases}$$

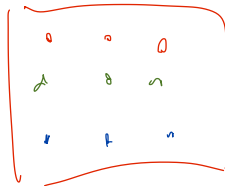
# What is a suitable criterion

looking for



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

其他两种概率相比线性误分率更倾向于划分纯净和非纯净  
由于在纯净状态向非纯净状态变化过程初期，线性误分率的斜率最低，这样错误惩罚也最低。与之相反的是，另外两种的初期惩罚非常高，因此更倾向于纯净划分

With  $\pi_c = p(y = c | t)$ :

- Misclassification rate:

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

- Entropy:

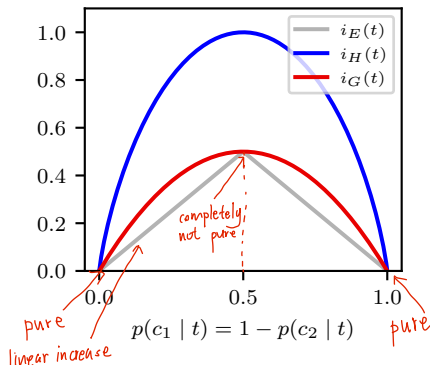
$$i_H(t) = - \sum_{c_i \in C} \pi_{c_i} \log_2 \pi_{c_i}$$

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

- Gini index:

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

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



# Shannon Entropy

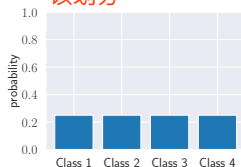
Expected number of bits needed to encode a randomly drawn value from a distribution (under most efficient code)

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

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

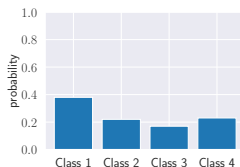
10  
11  
01  
00

需要两个字节来表现  
该划分

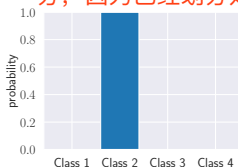


$$\mathbb{H} = 2.0$$

不需要任何字节表现划分，  
因为已经划分好了



$$\mathbb{H} = 1.9333$$



$$\mathbb{H} = 0.0$$

Higher entropy  $\rightarrow$  flatter histogram  $\rightarrow$  sampled values less predictable  
Lower entropy  $\rightarrow$  peakier histogram  $\rightarrow$  sampled values more predictable

# Detour: Information Theory

Information theory is about encoding and transmitting information

We would like to encode four messages:

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



# Detour: Information Theory

Information theory is about encoding and transmitting information

We would like to encode four messages:

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

# Detour: Information Theory

Information theory is about encoding and transmitting information

We would like to encode four messages:

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

The code above is called a Huffman Code. Happen more often so 0 is useful

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

$$i_G(t) = \sum_{c_i \in C} \underbrace{\pi_{c_i}}_{\text{probability of picking element}} \cdot \underbrace{(1 - \pi_{c_i})}_{\text{probability is misclassified}} = \sum \pi_{c_i} - \pi_{c_i}^2 = 1 - \sum \pi_{c_i}^2$$

Entropy vs Gini Index:

- It only matters in 2% of the cases which one you use.<sup>4</sup>
- Gini Index small advantage: no need to compute log which can be a bit faster

Efficient computer/ no log

<sup>4</sup>See Raileanu LE, Stoffel K. Theoretical comparison between the gini index and information gain criteria.

# Building a decision tree

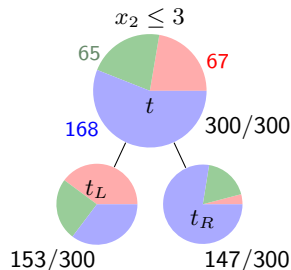
Compare all possible tests and choose the one where the improvement  $\Delta i(s, t)$  for some splitting criterion  $i(t)$  is largest

$$i_G(t) = 1 - \sum \mathcal{I}_{c_i}^2$$

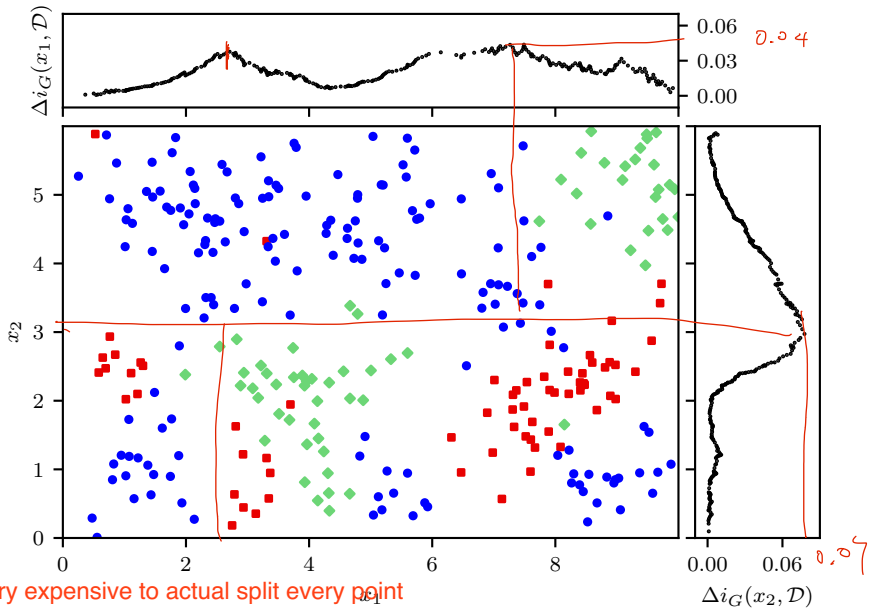
$$i_G(t) = 1 - \left(\frac{67}{300}\right)^2 - \left(\frac{65}{300}\right)^2 - \left(\frac{168}{300}\right)^2 \\ \approx 0.5896$$

After testing  $x_2 \leq 3$ :

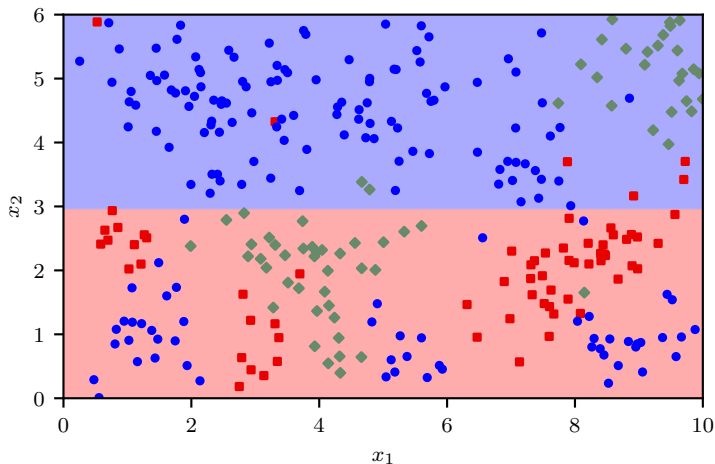
$$i_G(t_L) \approx 0.6548 \text{ and } i_G(t_R) \approx 0.3632$$



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

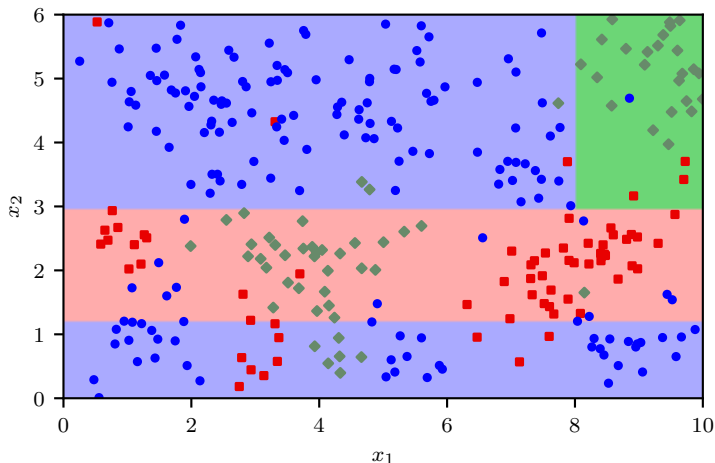


# Decision boundaries at depth 1



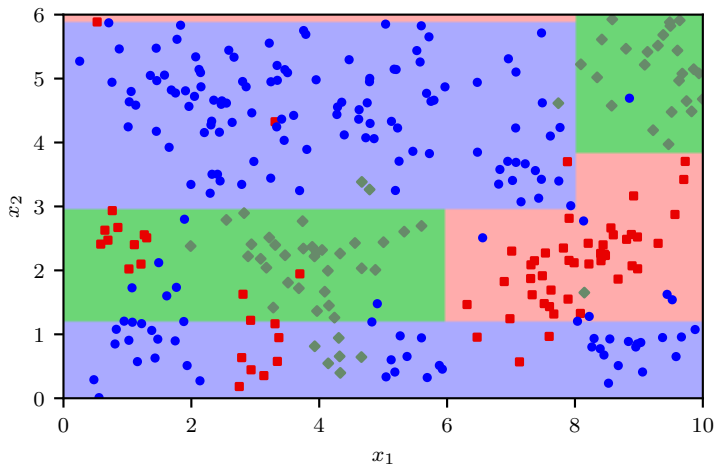
Accuracy on the whole data set: 58.3%

## Decision boundaries at depth 2



Accuracy on the whole data set: 77%

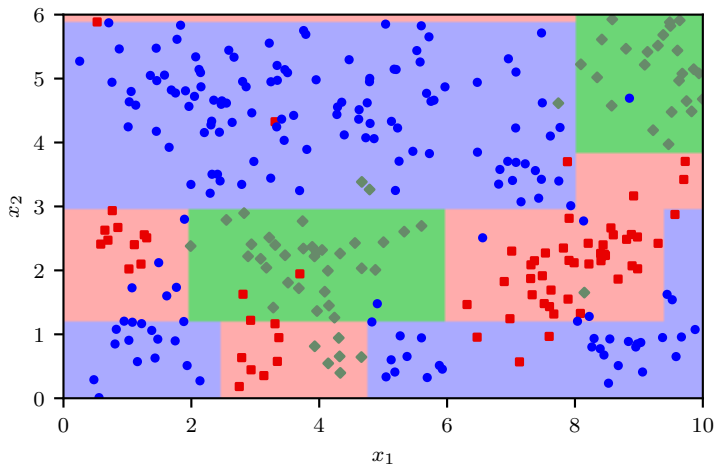
## Decision boundaries at depth 3



Accuracy on the whole data set: 84.3%

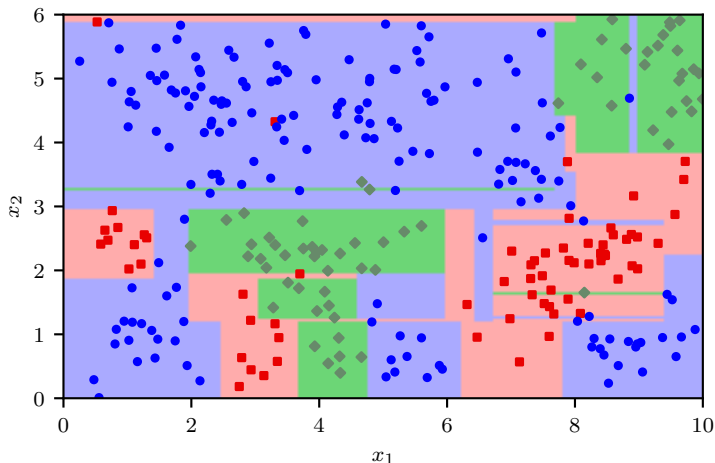


# Decision boundaries at depth 4



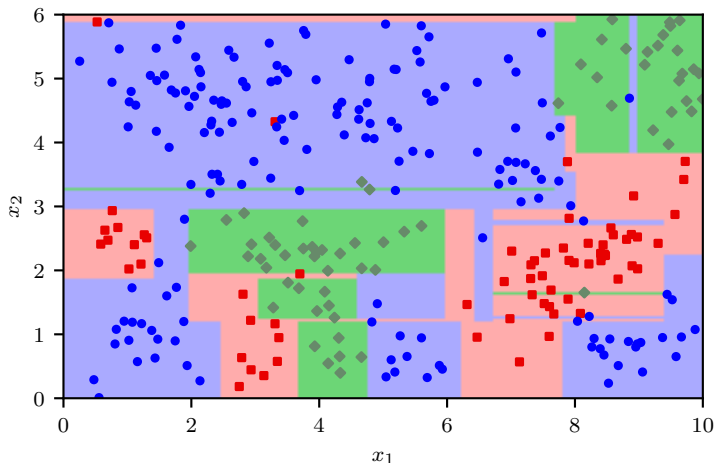
Accuracy on the whole data set: 90.3%

# 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.

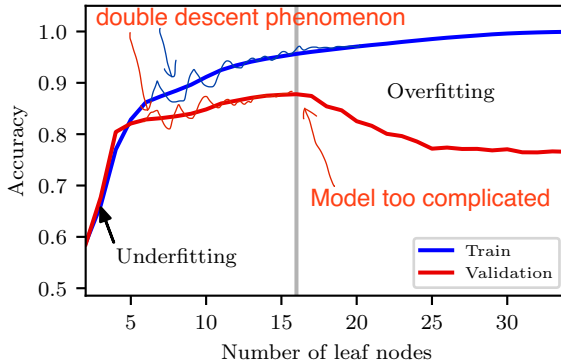
Overfitting means poor generalization!

# 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.

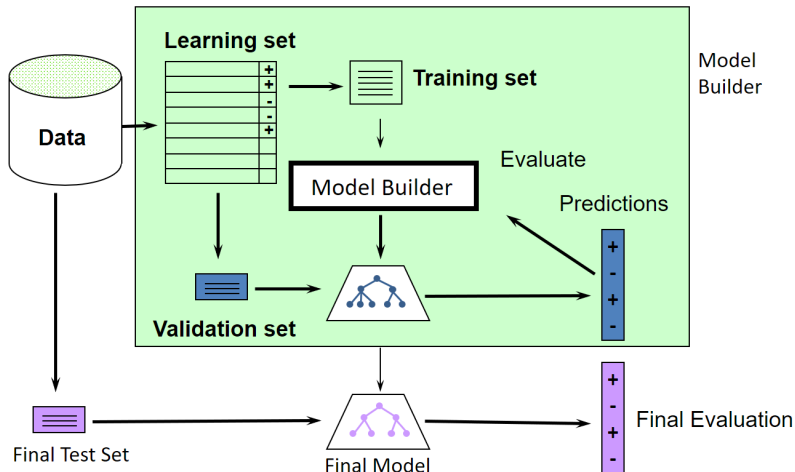
The validation performance tells us how well our model generalizes, not the training performance!<sup>5</sup>

---

<sup>5</sup>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

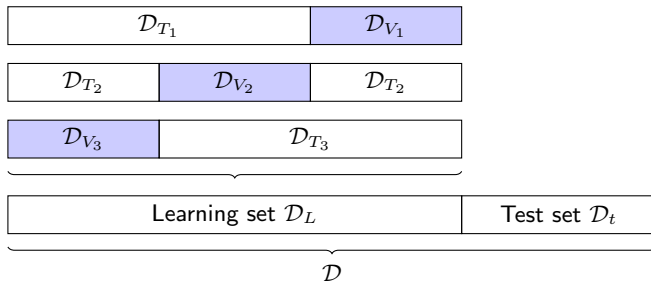
How to do model selection / battle overfitting?



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

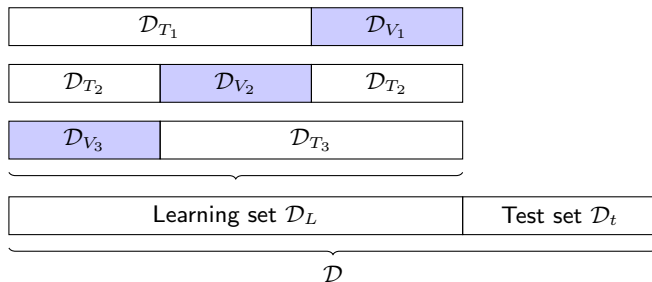
# $K$ -fold Cross-Validation

数据集不大





# 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 不在训练中变化的, 有人设计的
  - 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.  $i(t) = 0$  → might overfitting.
- 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:

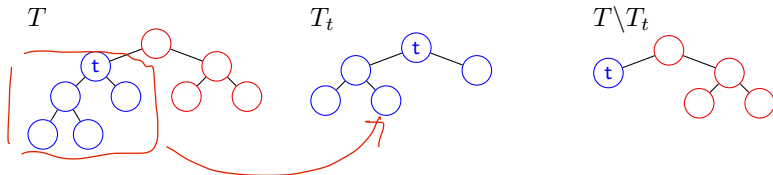
- distribution in branch is *pure*, i.e.  $i(t) = 0$
- maximum depth reached
- number of samples in each branch below certain threshold  $t_n$
- benefit of splitting is below certain threshold  $\Delta i(s, t) < t_\Delta$
- accuracy on the validation set

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  $T \setminus T_t$ .



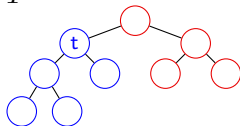
# Reduced error pruning

sklearn.

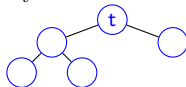
Let  $T$  be our decision tree and  $t$  one of its inner nodes.

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

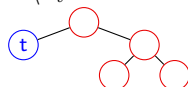
$T$



$T_t$



$T \setminus T_t$



- Use validation set to get an error estimate:  $\text{err}_{\mathcal{D}_V}(T)$ .
- For each node  $t$  calculate  $\text{err}_{\mathcal{D}_V}(T \setminus T_t)$  也可以对决策树的每一部分进行验证
- Prune tree at the node that yields the highest error reduction.
- Repeat until for all nodes  $t$ :  $\text{err}_{\mathcal{D}_V}(T) < \text{err}_{\mathcal{D}_V}(T \setminus T_t)$ . 修剪决策树

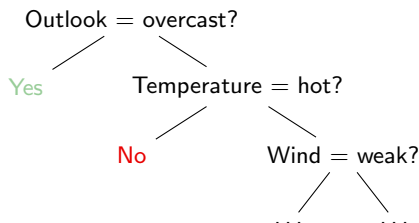
After pruning you may use both training and validation data to update the labels at each leaf.

retraining



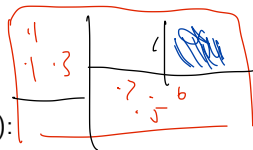
# Decision trees with categorical features

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



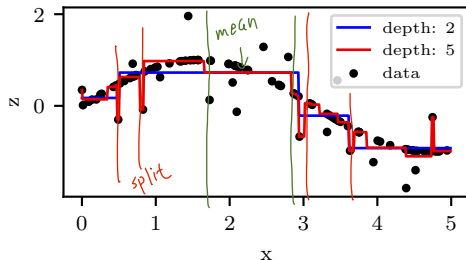
For regression (if  $y_i$  is a real value rather than a class):

- At the leaves compute the mean (instead of the mode) over the outputs.

在叶子上计算出平均数（而不是模式）。

- Use the mean-squared-error as splitting heuristic.

使用平均平方误差作为分裂启发式。



$$\sum_i (y_i - \mu)^2$$

# 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]
- biggest advantage*
- discrete and ordinal*

# Ensembles

装袋(自举式聚合)

- 通过对训练集进行抽样, 创建新的数据集 (带替换)。
- 在每个数据集上训练单独的分类器
- 合并预测, 例如, 平均数或多数票。

总计

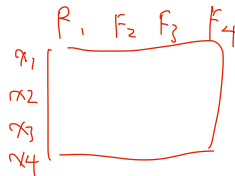
Main idea: Aggregate the predictions of many (diverse) classifiers to improve the performance.

Main benefit: Reduces the variance of the model by averaging.<sup>6</sup>

- Bagging (**bootstrap aggregating**)
  - 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 渐进式地训练 (弱) 分类器, 纠正以前的错误
  - Incrementally train (weak) classifiers that correct previous mistakes
  - Focus (give higher weight) on hard (misclassified) examples
- Stacking right
  - Train a meta-classifier with the base classifiers' predictions as features
- Bucket of models; Bayesian model averaging; Bayes optimal classifier

<sup>6</sup>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

袋装决策树可能是高度相关的  
由于我们需要不同的分类器，这就减少了袋法的好处。

想法。只使用随机抽样的特征子集来学习每棵树 (实例级的袋化+特征级的袋化)

$T_1$  •  
 $T_2$  •  
 $T_3$  •

由于良好的 "开箱即用" 性能，在实践中被广泛使用 (不需要太多的调整)

但与简单的决策树相比，可解释性较差

Decision Trees

随机特征数量的经验法则：回归的  $\log_2(d)$  或分类的  $\sqrt{d}$ ，其中  $d$  是特征的数量。

# 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

逐步训练“弱学习者”，如决策树（弱分类器）  
用统一的权重初始化一个权重向量

循环。

在加权的例子上训练弱学习者（例如通过加权抽样）。

对错误分类的示例增加权重

预测（基于错误的加权）的多数人

自适应（AdaBoost）和梯度（XGBoost）提升的不同之处主要在于权重的更新方式

AdaBoost：基于误差的闭合式权重更新

XGBoost：每次增加一棵树和树中的一个层次，贪婪地增加基于梯度和（自定义）损失函数进行分割

# 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.