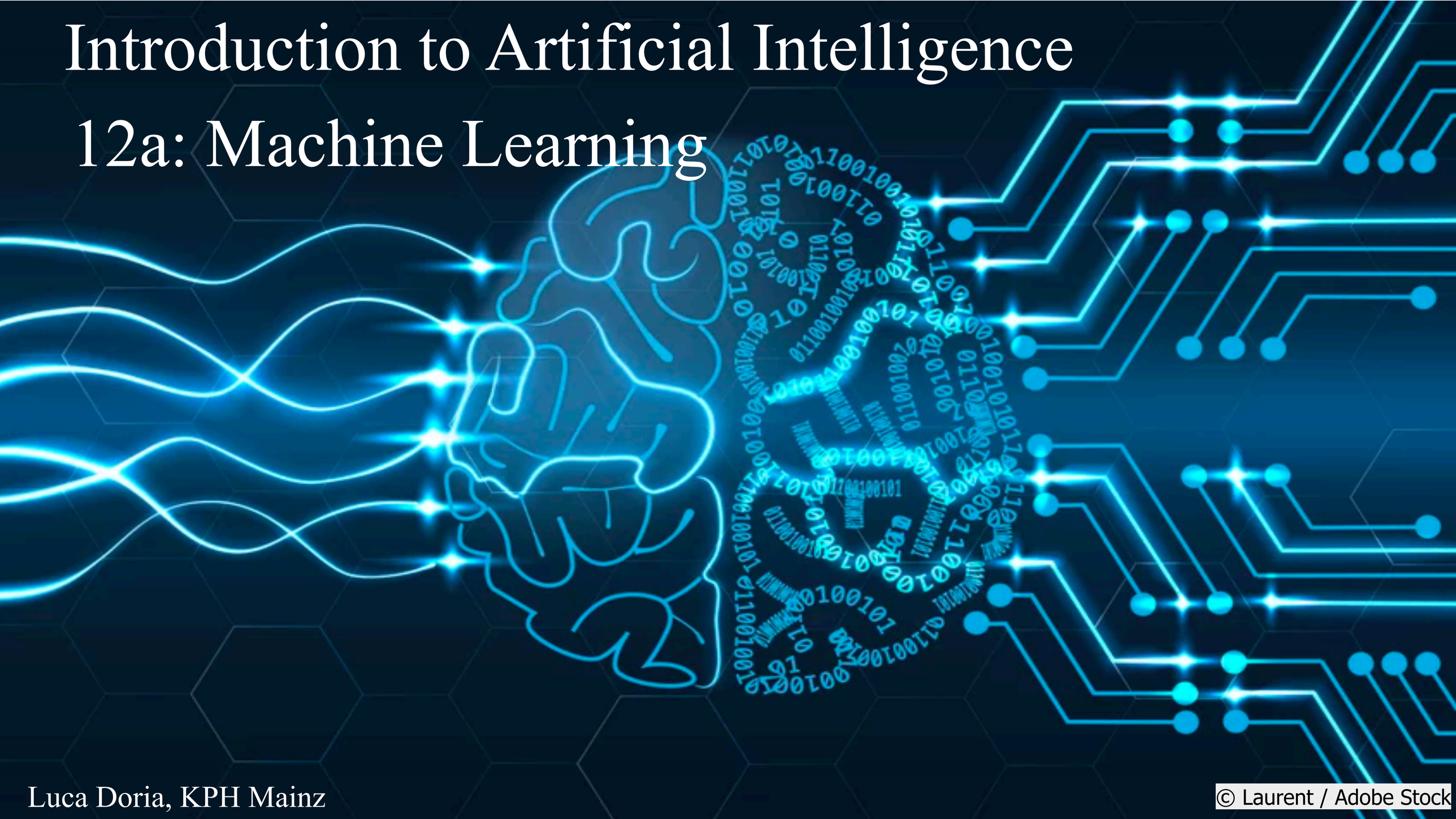
Introduction to Artificial Intelligence

12a: Machine Learning



Luca Doria, KPH Mainz

Introduction

- Machine learning is getting a lot of attention since some years
- Many applications, many different methods
- Since you will have specialised classes about that:
  - We will just give an introduction to this field
  - Some historical developments
  - Early methods which lead to the present ones

Overview

- Intelligent agents can improve their performance through learning.
- If the agent is a computer: machine learning.

- Supervised learning:
  the agent observes input-output pairs (examples) and learns from them.
- Unsupervised learning:
  the agent learns patterns just observing input examples.
- Reinforcement learning:
  the agent learns through rewards (or “punishments”!) what is correct or not.

- Tasks:
  - Classification
  - Regression

Classification

General Classification Problem:

Given a dimension N input vector of features $\vec{x}$, find a function f such that

$$f : \mathbb{R}^N \rightarrow C \subset \mathbb{N}^M$$

where the output is in a subset C of the integer numbers.
C are the "classes" (e.g. 0, 1, 2, 3, …) into which the input vectors can be divided (classified).

A special common case is boolean classification, where the output of f is restricted to two numbers (the input belongs to a class or not, e.g. 0,1).

In some cases, $f : \mathbb{R}^N \rightarrow \mathbb{R}$ and the output can be seen as the probability of belonging to a class (or not).

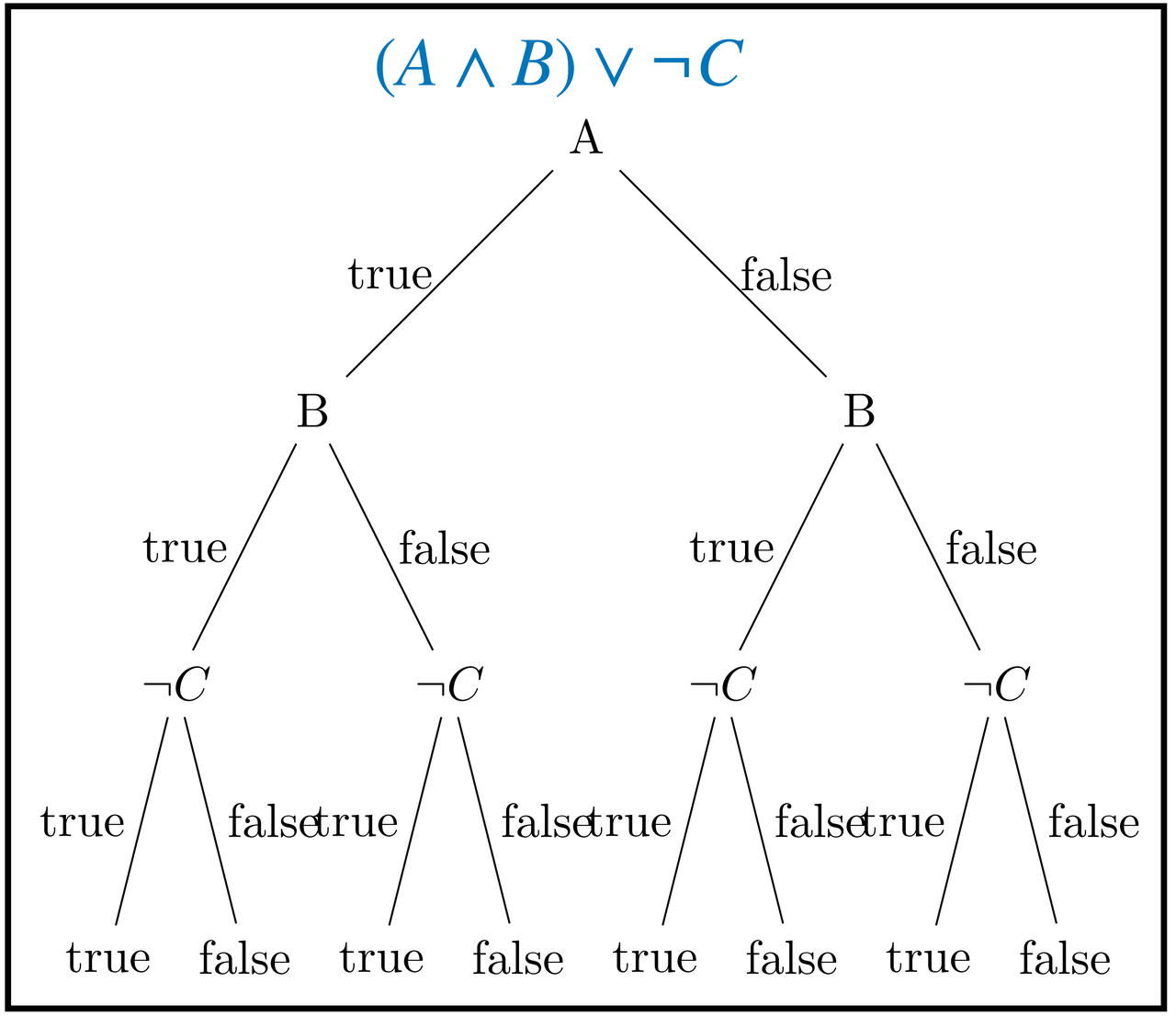
Classification: Decision Trees

A sequence of 0-1 decisions can be represented as a propositional logic sentence.

Every sentence can be viewed as a decision tree.

The nodes correspond to "attributes".

Warning: In general, trees become exponentially large.



Example

Russell&Norvig

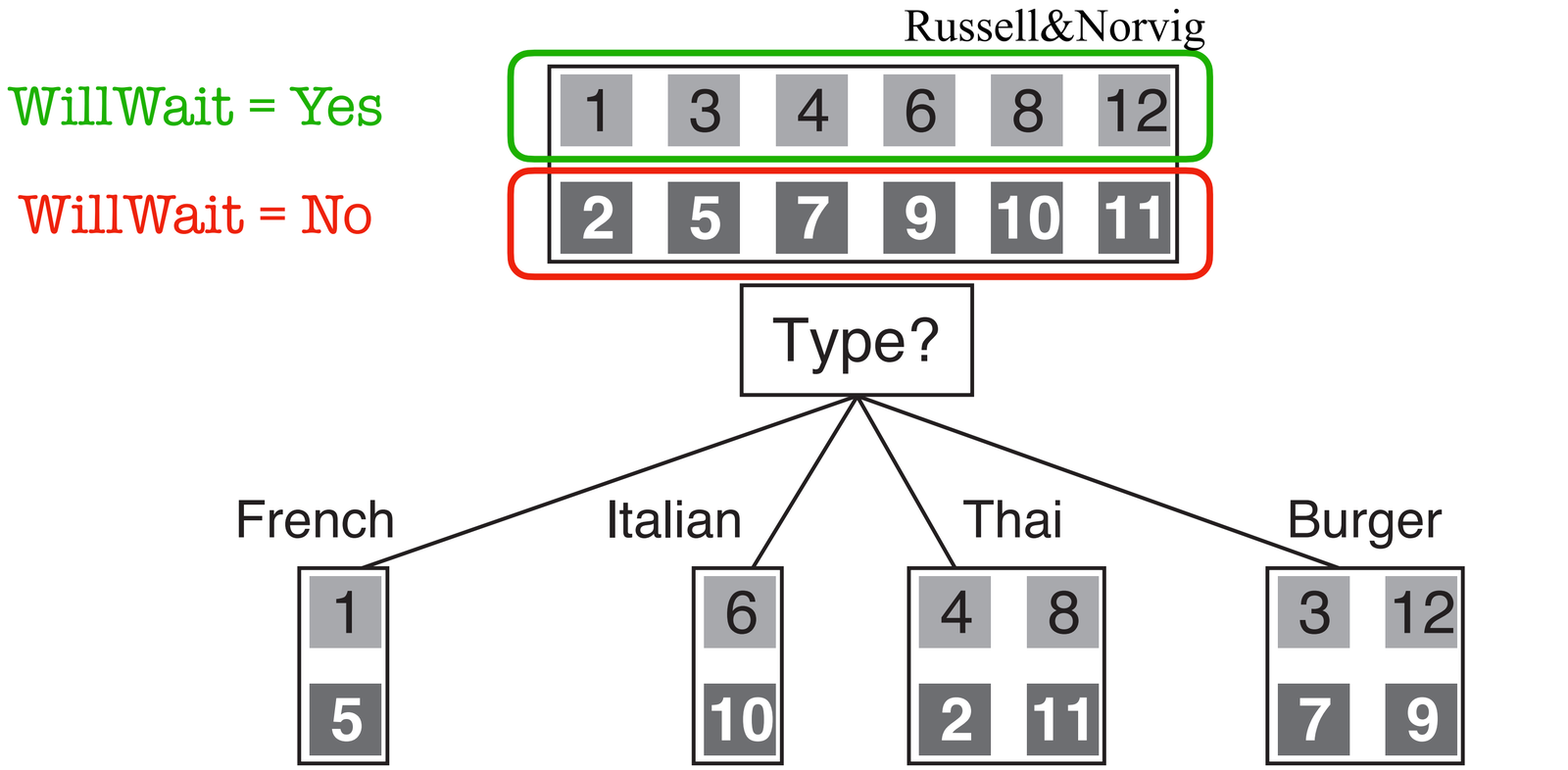
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*60*	$_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*60*	$_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$_{12} = Yes$

$\vec{x}$ → 0,1

Decisions

A decision splits the training dataset into classes.

Relative to the previous dataset, a decision over the **`Type`** attribute leads to:



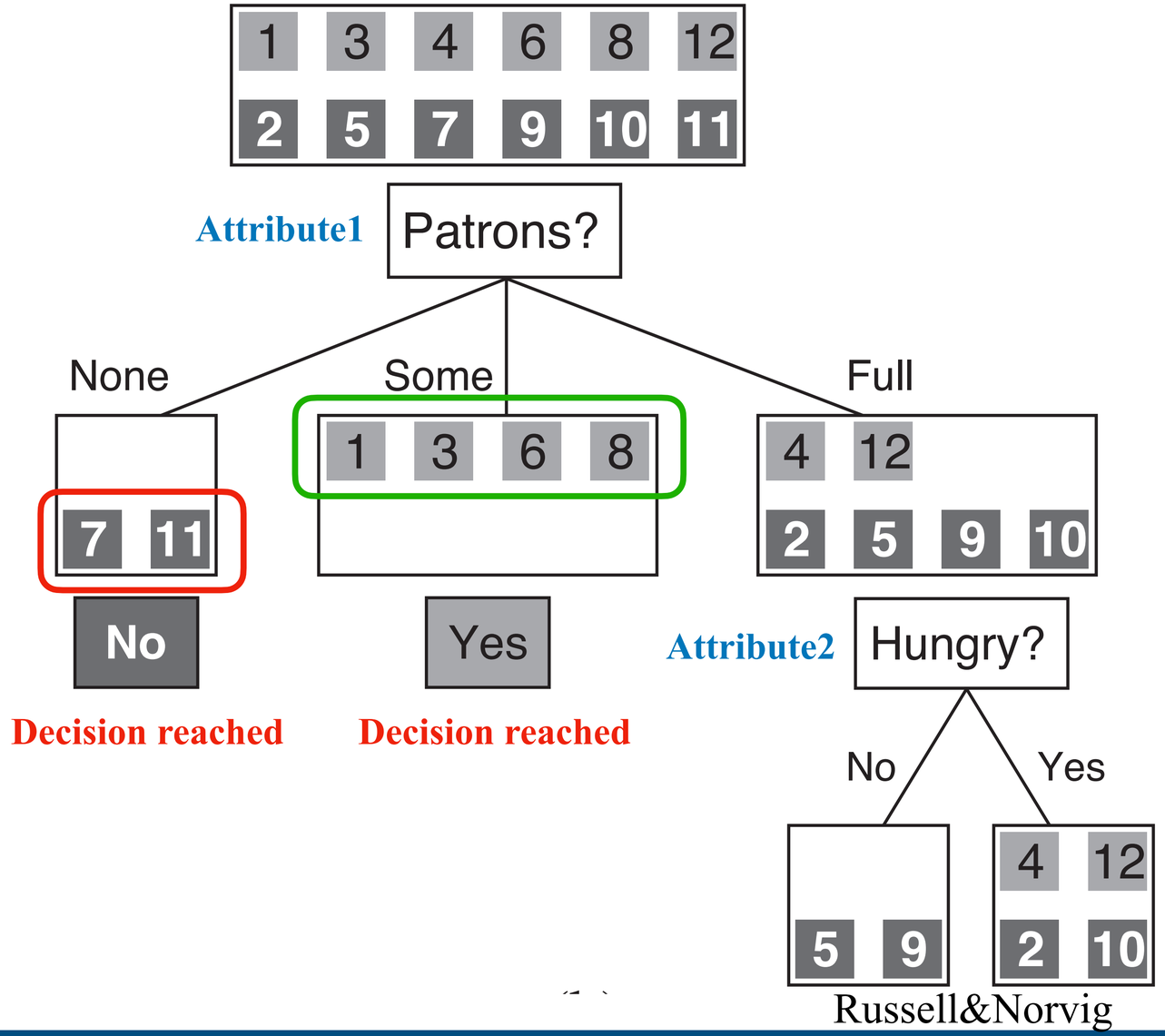
Split again each leaf according to another attribute….

Which attribute to use? Or: how to split?

We split first according to Patrons: found two cases where we can take a decision.

In the case (Patrons=Full) we cannot take a decision and split again based on Hungry?

How can we decide which attribute to use at each step?



Entropy

We introduce the concept of **entropy** from information theory (C. Shannon, 1949).

We can associate an entropy to a random variable: the more information it contains, the smaller is the entropy. Often the entropy is measured in bits of information.

Examples:

1) A random variable which returns always the same value must have entropy = 0
2) A fair coin has entropy = 1bit (the information is only one: head or tail).

Definition:

If a random variable R has values r_k with probability $P(r_k)$, the entropy H is

$$H(R) = -\sum_k P(r_k)\log_2 P(r_k)$$

Entropy

Particular case: Entropy for a boolean variable:

$$H = -p\log_2 p - (1-p)\log_2(1-p)$$

Example: fair coin:

$$H = -0.5\log_2 0.5 - 0.5\log_2 0.5 = -\log_2 0.5 = 1 \text{ bit}$$

If a training set contains P positive cases and N negative cases, the (estimated) probability of a positive case is $p_{PN} = P/(P+N)$ and thus the entropy is:

$$H = H(p_{PN})$$

Information Gain (1)

- The result of a test on a random variable will result in some acquired knowledge and this leads to an entropy reduction.
- If an attribute A has d different values, the training set is divided in subsets E_k each of which has probability $p_k = P_k/(P_k+N_k)$, where P and N are like before the positive and negative cases, respectively.
- If we follow the k-branch of the tree, then the entropy of the dataset in the node is $H=H(p_k)$ while the entropy of the whole dataset is $H=H(p)$.
- A randomly chosen example from the dataset has a probability $(P_k+N_k)/(P+N)$ to belong to the subset E_k.

Information Gain (2)

We define the **information gain** as:

$$Gain(D, A) = H(A) - \sum_{k \in Val(D)} \frac{p_k + n_k}{p + n} H\left(\frac{p_k}{p_k + n_k}\right)$$

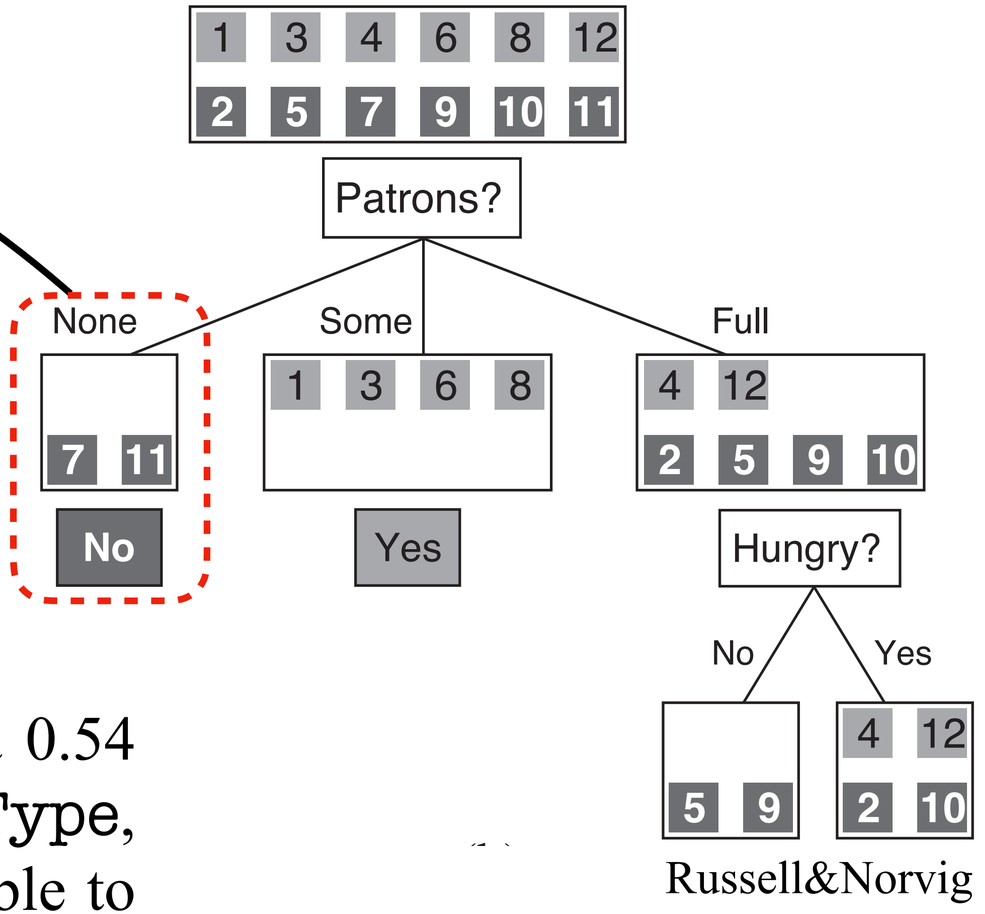
- $Gain(D, A)$: Dataset
- $H(A)$: Entropy relative to the attribute A
- $\frac{p_k + n_k}{p + n}$: Probability of a randomly chosen example. Or: probability to be in E_k. Or: proportion of examples in A with value k.
- $H\left(\frac{p_k}{p_k + n_k}\right)$: Entropy of the subset E_k

The IG quantifies the reduction of entropy (or the information gain) for each choice of the attribute value (or: splitting the dataset according to A).

Information Gain (3)

Example from the restaurant choice (see before):

$$\text{Gain(Patrons)} = 1 - \left[\frac{2}{12}H(\frac{0}{2}) + \frac{4}{12}H(\frac{4}{4}) + \frac{6}{12}H(\frac{2}{6})\right] \approx 0.54 \text{ bits}$$



Russell&Norvig

Splitting the sample according to Patrons will lead to a 0.54 bits of information gain. If you repeat the calculation for Type, you will get zero (check!). —> Patrons is a better variable to use first (maximum information gain).

Training a Decision Tree

function DECISION-TREE-LEARNING($examples$, $attributes$, $parent\_examples$) **returns** a tree

  if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
  else if all $examples$ have the same classification **then return** the classification
  else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
  else
    $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, $examples$)
    $tree \leftarrow$ a new decision tree with root test A
    for each value v_k of A **do**
      $exs \leftarrow \{ \mathsf{e} : \mathsf{e} \in examples$ **and** $\mathsf{e}.A = v_k \}$
      $subtree \leftarrow$ DECISION-TREE-LEARNING(exs, $attributes - A$, $examples$)
      add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
    return $tree$

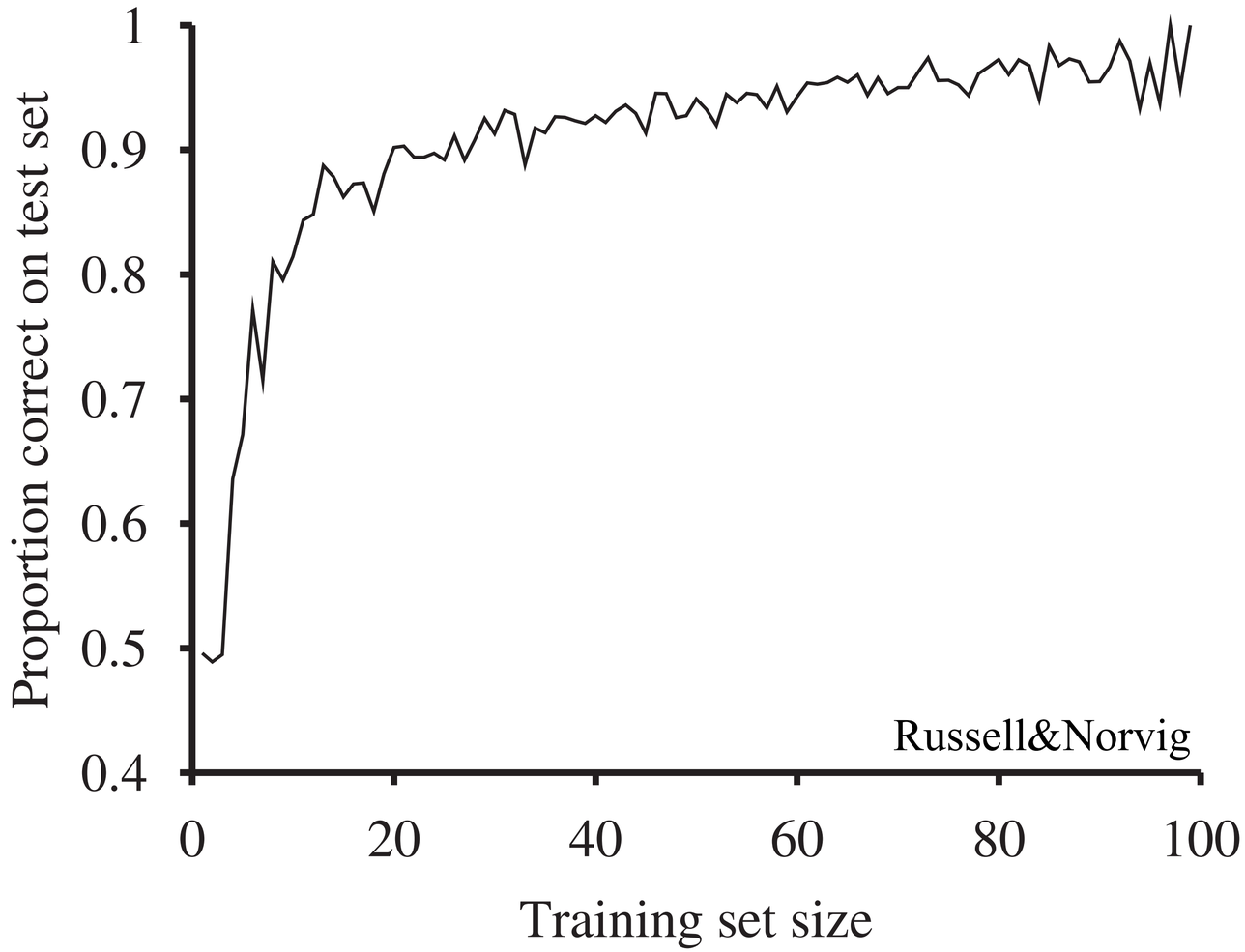
Selects most common output value among an examples set, breaking ties randomly.

Max Gain

Recursion

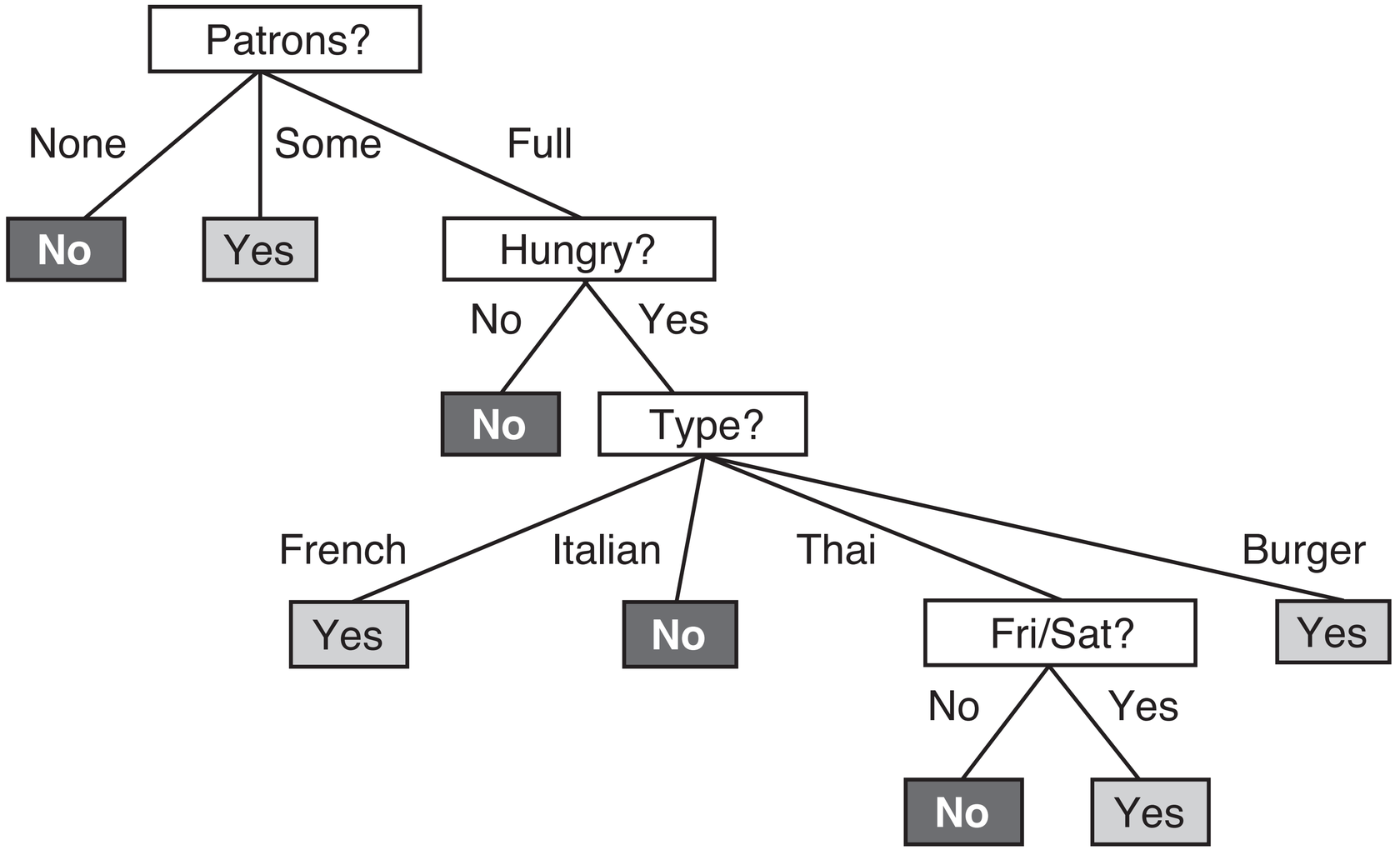
The output of the training will be a sequence of splittings induced by a sequence of attributes, chosen with the criterium of maximum information gain. Ideally, the tree resulting from the training should be the smallest one. In its generality, the problem is intractable, but with the previous heuristics (splitting first with attributes giving the maximum information gain), a good result can be found.

Training a Decision Tree



Learning curve for 100 randomly generated examples

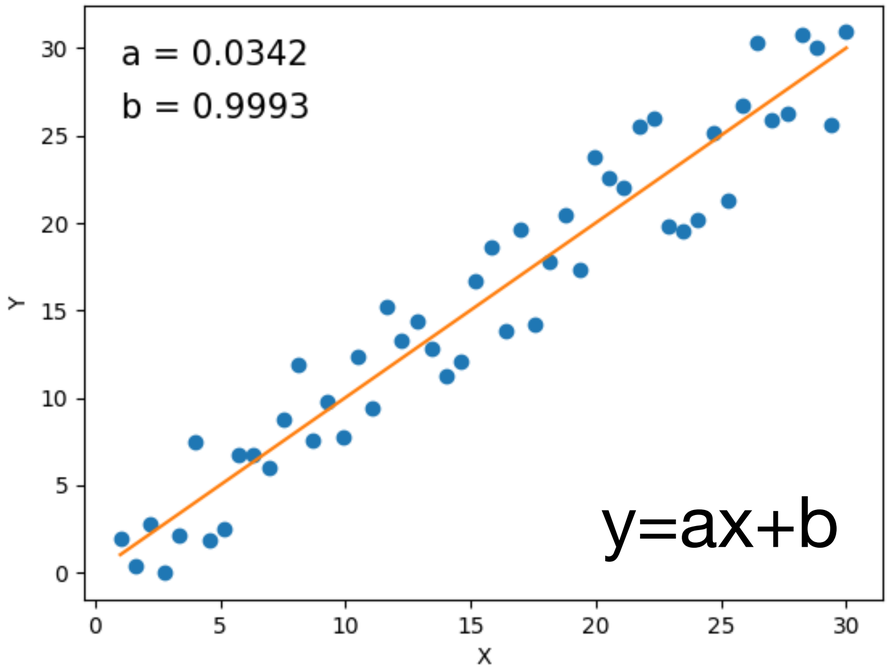
Output Example



Russell&Norvig

Fitting

General problem: find a function which fits data points (the “examples”).
Example: linear fit:

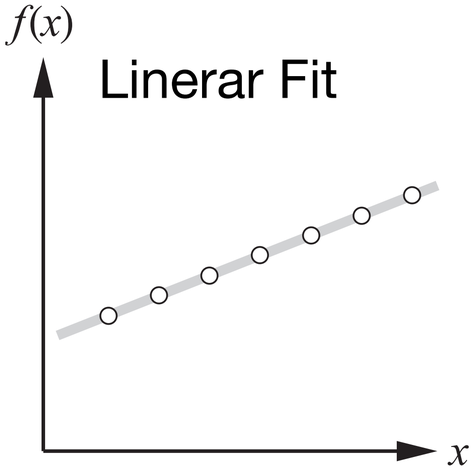


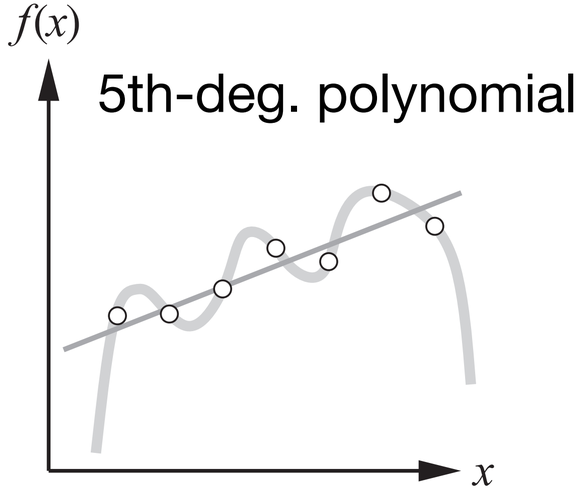
$$a = \frac{\sum_i y_i \sum_i x_i^2 - \sum_i x_i \sum_i x_i y_i}{N \sum_i x_i^2 - \left(\sum_i x_i\right)^2}$$

$$b = \frac{N \sum_i x_i y_i - \sum_i x_i \sum_i y_i}{N \sum_i x_i^2 - \left(\sum_i x_i\right)^2}$$

Problem: did we choose the correct function?
Is the function too simple or too complex?

Overfitting





In the left-case, the line seems to describe the points.
In the right case, the line generically describes the points but a more complex function interpolates them.
Problem: are the points on a straight line + “noise” or are they really coming from a 5th-order polynomial.
If not, the 5th-order polynomial is just “learning noise”, or **overfitting**!

Avoiding Overfitting in DTs

In general:

- Likelihood of overfitting increased with more attributes (parameters)
- Likelihood of overfitting decreases with more examples.

For DTs: Pruning Technique

- Eliminate a tree's node if it splits the sample in subsets with similar proportions of positives.
- This means a small Gain.
- How small?

DT Pruning (1)

- Use a statistical significance test.
- Null hypothesis: the attribute is irrelevant and the gain is zero (for an infinite set).
- Can we reject statistically the null hypothesis?
- We would like to calculate the probability that under the null hypothesis a sample with size N=p+n exhibits the observed deviation from the expected distribution of positive and negative examples.
- Expected positives and negatives:

$$\hat{p} = p\frac{p_k + n_k}{p + n} \qquad \hat{n} = n\frac{p_k + n_k}{p + n}$$

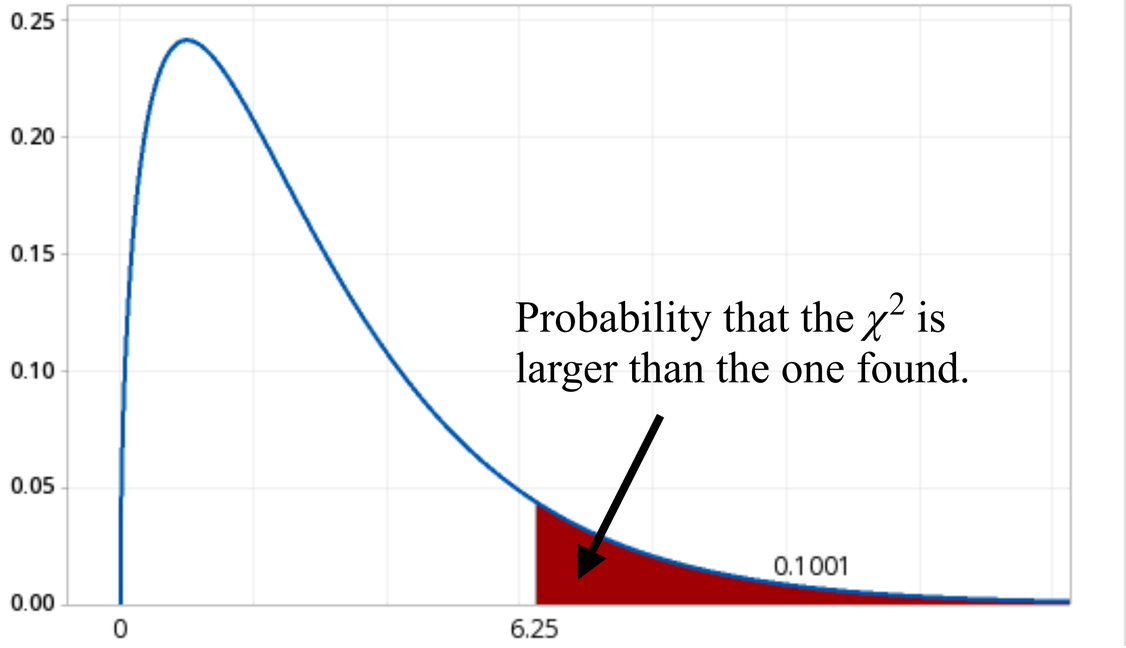
These are the expected numbers assuming the null hypothesis, i.e. irrelevance of the attribute.

DT Pruning (2)

For assessing the deviation from the hypothesis we can use the χ^2 measure:

$$\chi^2 = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k} \Leftrightarrow \sum_i \frac{(O_i - E_i)^2}{E_i}$$

The last variable is distributed following the χ^2 distribution with d-1 degrees of freedom which can be used for calculating the significance.

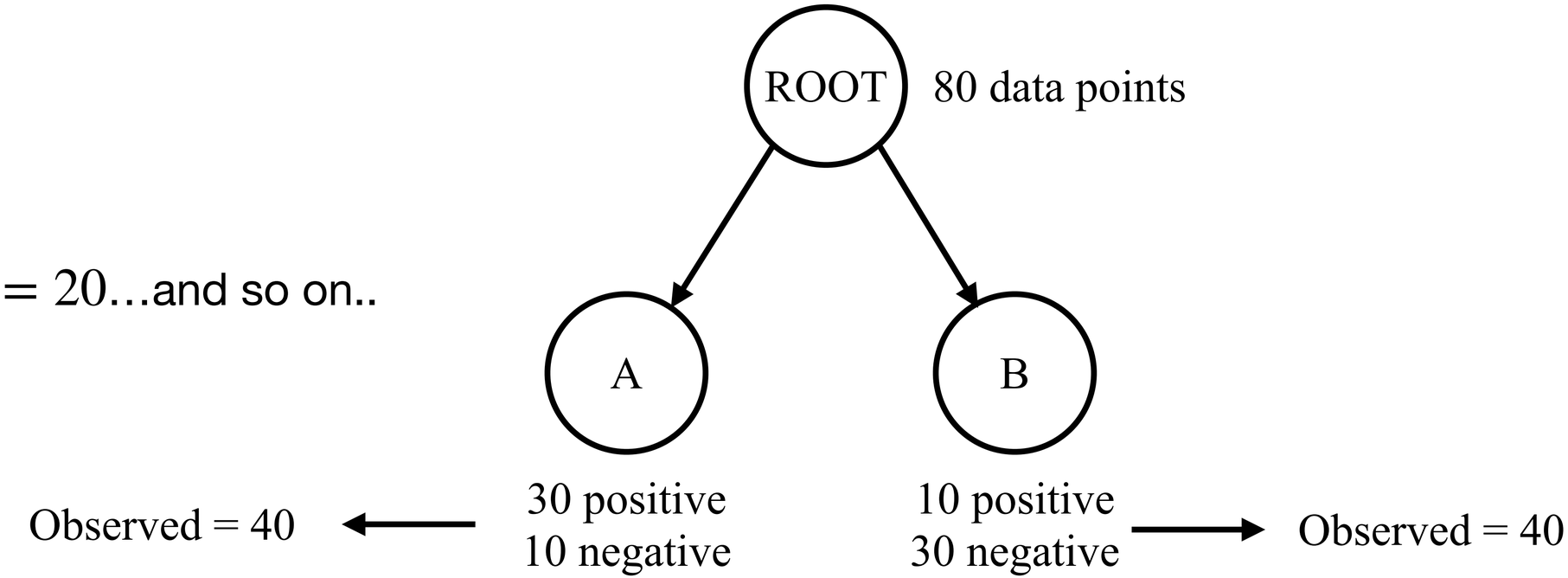


DT Pruning (3)

Example

Expected =

$$\hat{p} = p\frac{p_k + n_k}{p + n} = 40 \times \frac{40}{80} = 20 \text{...and so on..}$$



$$\chi^2_A = \frac{(30-20)^2}{20} + \frac{(10-20)^2}{20} = 10$$

$$\chi^2_B = \frac{(10-20)^2}{20} + \frac{(30-20)^2}{20} = 10$$

d=2 —> Integrate a chi-square distribution for 1 degree of freedom. Integral = 3.8 < 10 : no puning.
If the integral were > 3.8: prune the node.

Boosting: Boosted Decision Trees

Basic idea of boosting:
Examples that are wrongly classified by the current model are given more weight, so another model will focus more on these hard cases.

A boosting algorithm: **AdaBoost** (Adaptive Boosting, **Ensemble Learning Algorithm**)

1) Initialize Weights**:** Each observation in the training dataset is assigned an equal weight initially.
2) Train a "weak learner" (small tree) on the weighted dataset.
3) Calculate error: total weight of the misclassified examples.
4) Compute the weight of the weak learner based on its error (lower error, higher weights)
5) Update weights: increase weights of misclassified examples.
6) Combine the weak learners: the model is a weighted sum of the learners.

AdaBoost (more details)

- Assign weight w_i to the i-th example in the dataset.
- Train the weak learner h_t on the weighted dataset (often a tree with 1 level).
- Calculate the error of the misclassified examples: $e_t = \sum_i w_i \cdot I(y_i \neq h_t(x_i))$

- Weight for the weak learner: $\alpha_i = \frac{1}{2} \log\left(\frac{1 - e_t}{e_t}\right)$: low error → high weight

- Update weights (**BOOSTING**): $w_i^{t+1} = w_i^t e^{\alpha_t \cdot I}$ and normalise: $w_i^{t+1} = w_i^{t+1} / (\sum_i w_i^{t+1})$

- Final model (weighted sum): $H(x) = sign\left(\sum_t \alpha_t h_t\right)$

Boosting Algorithms

- AdaBoost is powerful because effectively reduces bias and variance.
- It can be sensitive to noisy data and outliers since these can receive very high weights.
- Boosting can significantly improve the predictive accuracy of the model.
- Boosting can overfit, but in general it is more robust to overfitting compared to other methods.
- Other boosting procedures exist (e.g. Gradient Boosting)

Support Vector Machines

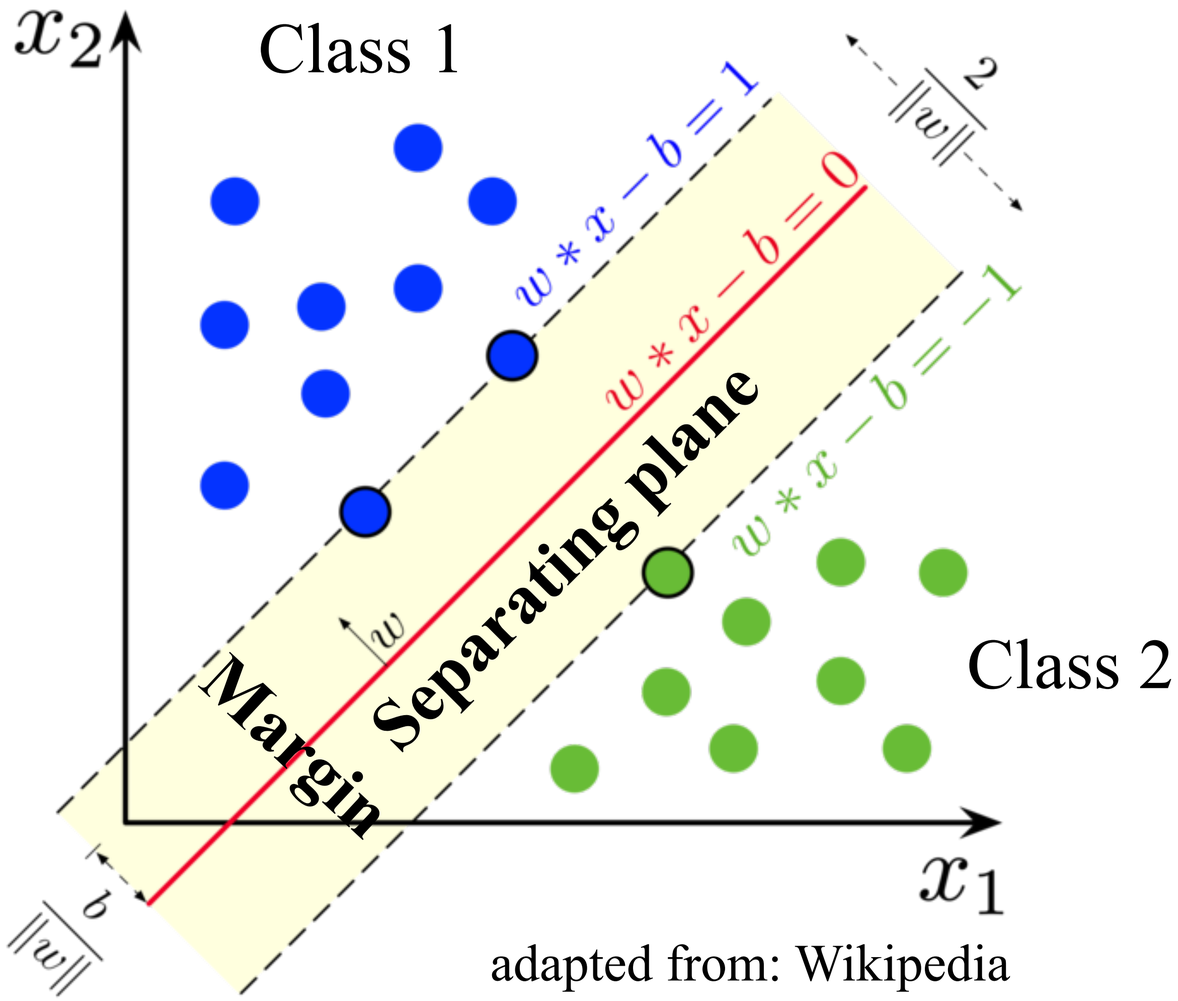
Before a suitable HW was available to train large neural networks (~2000s), support vector machines enjoyed big popularity.

Advantages:

- Can work on non-linear separation problems
- Computationally simple and fast to train
- Avoid/reduce the multiple-minima problem typical of NNs.

Idea: margin maximisation

Support Vector Machines



adapted from: Wikipedia

- In this example, a linear separation is possible.
- We define the plane $\vec{w} \cdot \vec{x} = b$
- Find the vector **w** that maximises the margin.
- Commonly, the output of an SVM is +1 or -1, corresponding to the two classes:
  Class 1: wx-b=+1 (Plane 1)
  Class 2: wx-b=-1 (Plane 2)

Distance between the planes: 2/||w|| .
We would like to maximise it.

Support Vector Machines

Starting point: decision function $f(x) = wx + b$

Minimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

$$y_i(w \cdot x_i + b) \geq 1 \quad \text{for all } i$$

"Soft-margin" case:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \text{for all } i$$

Support Vector Machines

Lagrangian formulation:

$$L(w, b, \xi, \alpha, \beta) = \underbrace{\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i}_{\text{Function to minimise}} - \sum_{i=1}^{n}\alpha_i[y_i(w \cdot x_i + b) - 1 + \xi_i] - \sum_{i=1}^{n}\beta_i\xi_i$$

Lagrange multipliers

Lagrange multipliers

Constraints

Support Vector Machines

Minimization:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \quad \Rightarrow \quad \alpha_i = C - \beta_i$$

Support Vector Machines

Substituting in the original lagrangian we get the dual representation:

$$L(w, b, \xi, \alpha, \beta) = \frac{1}{2}\left(\sum_{i=1}^{n} \alpha_i y_i x_i\right) \cdot \left(\sum_{j=1}^{n} \alpha_j y_j x_j\right) - \sum_{i=1}^{n} \alpha_i [y_i (w \cdot x_i + b) - 1 + \xi_i]$$

$$= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{n} \alpha_i$$

Notice the dot-product: relevant for the “Kernel trick”, see next slides.

Support Vector Machines

Dual minimisation problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

with constraints:

$$0 \leq \alpha_i \leq C \quad \text{for all } i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

This is a quadratic optimisation problem: many algorithms exist for solving it.

Support Vector Machines

Once the previous QOP is solved:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$b = y_k - \sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x_k)$$

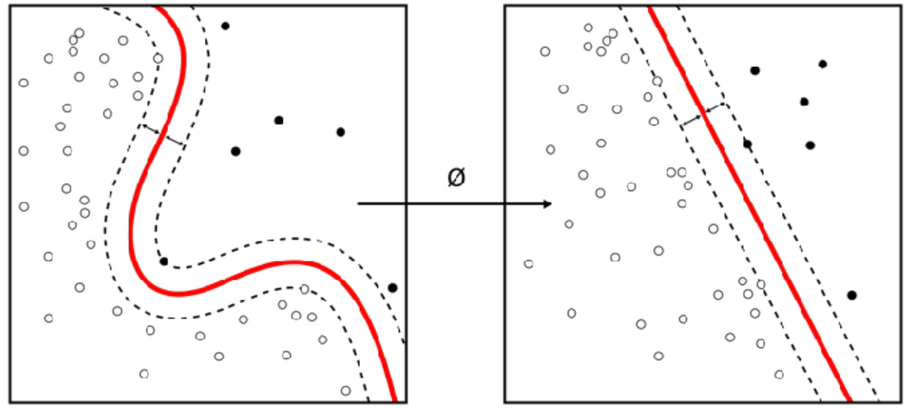
Decision function:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i (x_i \cdot x) + b \right)$$

Support Vector Machines

The “Kernel Trick”:

When the data are not linearly separable, we can transform it into an higher-dimensional space applying a kernel to the input vectors. In the higher-dimensional space, the data might be (better) separable.
Commonly used kernes are:



adapted from: Wikipedia

- **Linear Kernel:** $K(x_i, x_j) = x_i \cdot x_j$
- **Polynomial Kernel:** $K(x_i, x_j) = (x_i \cdot x_j + 1)^d$
- **Radial Basis Function (RBF) Kernel:** $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
- **Sigmoid Kernel:** $K(x_i, x_j) = \tanh(\alpha x_i \cdot x_j + c)$

Summary

Two machine learning algorithms: Decision trees and Support Vector Machines

- The notion of information gain and entropy can be used for decision trees.
- Branch pruning and boosting can significantly improve DTs.
- SVMs are efficient classifiers.
- Computationally affordable (with limits).
- Based on a well-known quadratic optimisation problem.
- Quite good for high-dimensional problems (many features).
- The application to a kernel instead of a dot-product can generalize them to non-linearly separable problems.