Overview

I Artificial Intelligence

II Problem Solving

III Knowledge, Reasoning, Planning

IV Uncertain Knowledge and Reasoning

V Machine Learning

19. Learning from Examples

20. Learning Probabilistic Models

21. Deep Learning

22. Reinforcement Learning

VI Communicating, Perceiving, and Acting

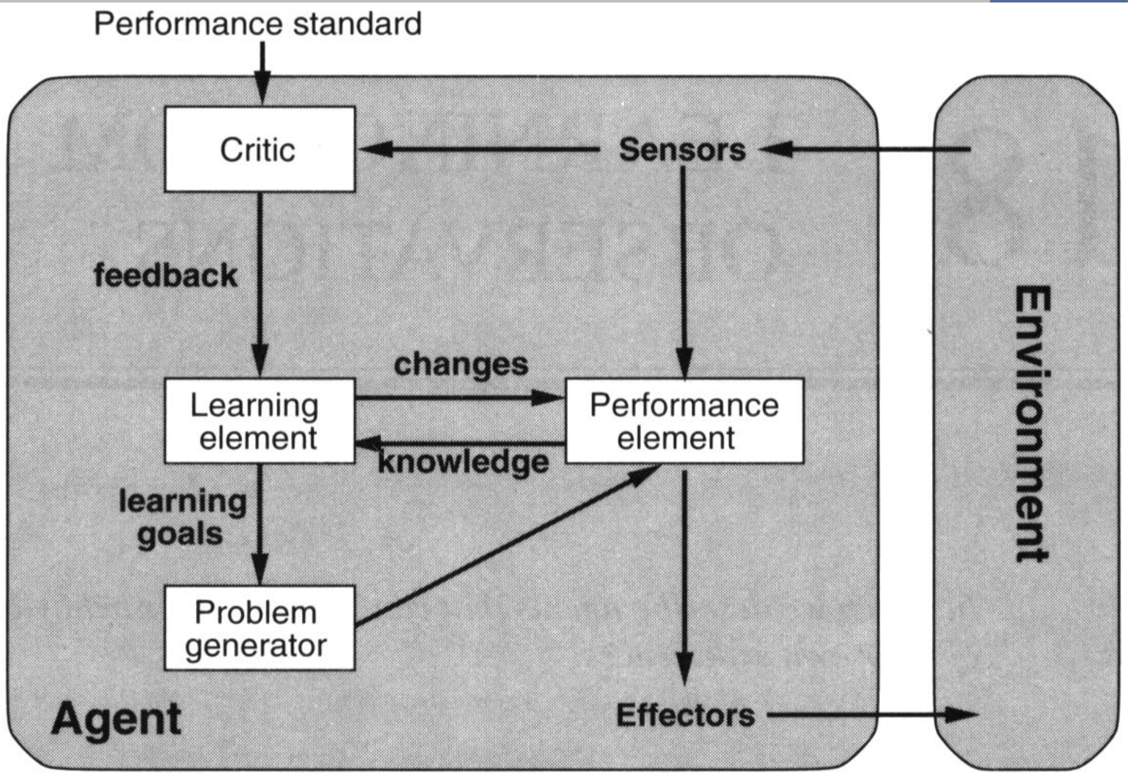
VII Conclusions

Learning from Examples

- Forms of Learning
- Supervised Learning
- Learning Decision Trees
- Model Selection and Optimiziation
- The Theory of Learning
- Linear Regression and Classification
- Nonparametric Models
- Ensemble Learning
- Developing Machine Learning Systems

Forms of Learning

- **Factors for learning**:
  - Component of agent program to be improved
  - Prior knowledge of agent
  - Knowledge representation
  - Available data and feedback on that data
- **Possible Components**:
  1. Problem generator, critic and learning element
  2. Direct mapping from condition to actions
  3. Utility information
  4. Goals describing desirable states
  5. Inference of relevant properties of the world from percept sequence
  6. Information how the world evolves and about results of actions
  7. Action-value information about desirability of actions
- **Component examples** for self-driving car learning from human driver:
  - Infer braking rules from driver brakes (2.); Learn camera images being told to be buses (5.); Try actions like braking in different situations (6.); Learn utility from passenger comments (3.)



Learning Agents

Available Feedback:

- **Supervised learning:** Agent observes input-output pairs and learns a function that maps input to output, e.g. learning the label of bus images or learning the distance to stop when braking under various conditions (speed, road conditions, etc.)
  - Opposed to logical deduction, conclusion inferred with this **induction** might be incorrect
- **Unsupervised learning**: Learning without explicit feedback like e.g. clustering
- **Reinforcement learning**: Learning from series of reinforcements (rewards or punishments), e.g. in games

Knowledge representation:

- Input is mostly **factored representation** (e.g. vector of attribute-value pairs)
- But input could be any data structure including atomic and relational

Supervised Learning

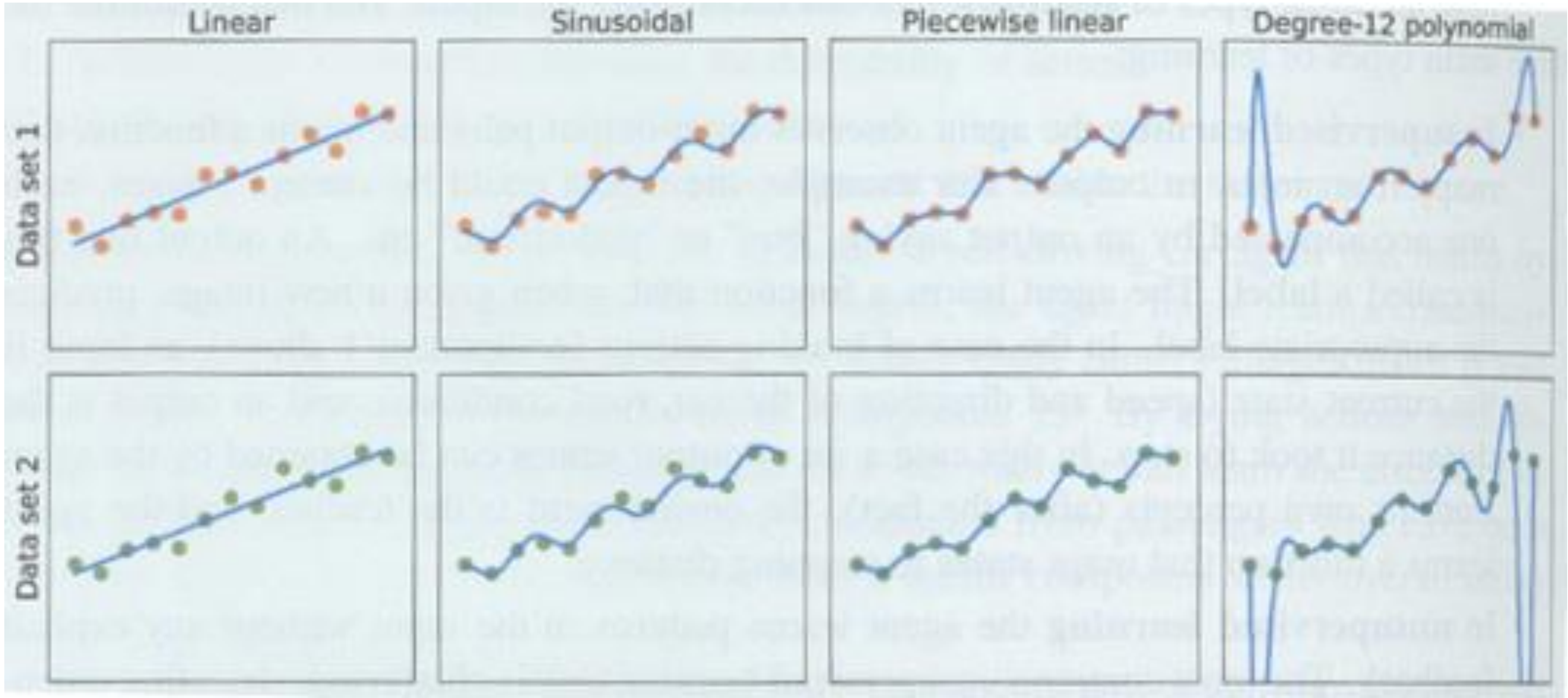
- *Given:* A training set of N example input-output pairs $(x_1, y_1), (x_2, y_2), \ldots. (x_N, y_N)$,
  - Where each pair was generated by an unknown function $y = f(x)$
- *Sought*: A function h (**hypothesis**) that approximates the true function f
  - Hypothesis is drawn from a usually constrained **hypothesis space** (e.g. 3-KNF)
    - Based on prior knowledge or an exploratory data analysis
  - Evaluation based on yet unseen examples, therefore split of examples in **training set** and **test set**

- If the output is a finite set of values (e.g. labels for pictures) the learning problem is called **classification**, if a number **regression** (numeric prediction of e.g. a score)

Supervised Learning (2)

- **Main problem: Generalization:** How to choose among different hypotheses?
  - True measure is **test set**, not **training set**
  - **Variance in training sets** (e.g. data set 1 vs. data set 2 for polynomial) also good indicator
  - **Ockham's razor**: Prefer simple hypotheses!
    - What is simple? Number of parameters of a hypothesis?
    - What is the best compromise between complexity and data fit?

- **Bias** by assumptions
- Hypothesis space causes bias: **Overfitting** vs. **underfitting**
- **Bias-variance trade-off**
- Bias by wrong assumptions



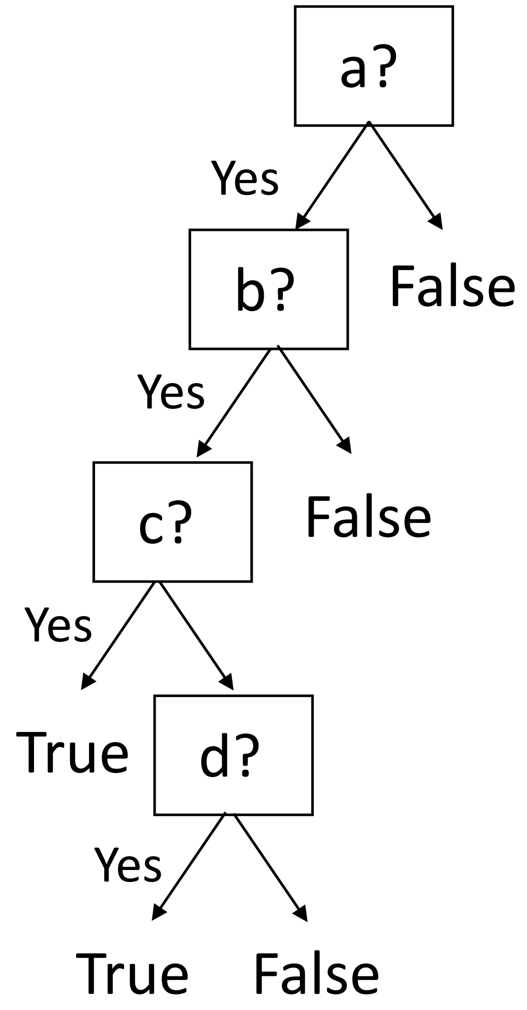
Consequences

- Learning requires some **bias:**
- **Tradeoff** beween **expressiveness** of hypothesis and **efficieny** of learning algorithm unavoidable
- **Data bias:** The available training data is not representative for the hypothesis (e.g. past data in a changing environment)
  - Incremental learning algorithms for continuous integration of new examples in data set and update of hypothesis helpful!

Decision Trees

Input: Vector of attribute values (discrete or continuous)
Output: Yes/No decision

- Decision trees represent therefore Boolean functions.
- A boolean decision tree is equivalent to a logical statement of the form
  Output $\Leftrightarrow$ (Path$_1$ $\vee$ Path$_2$ $\vee$ ...)
  It can be viewed as a compact reprentation of rules (pathes),
  e.g. instead of (a $\wedge$ b $\wedge$ c) $\vee$ (a $\wedge$ b $\wedge$ d) see right tree
- Although „How-To“ manuals are often written as decision trees, many boolean functions like parity or majority functions cannot be represented concisely.
- Are there better representations?
  - No, because there are two many functions



Number of Boolean Function with n attributes

1. A boolean function with n boolean attributes requires in general 2^n bits for its representation (truth table).
2. A set of n elements has 2^n subsets. Each subset may be a boolean function.
3. There are in total 2^{2^n} different functions (with n=2, 16 boolean functions, s. below; with n=6 already 18 446 744 073 709 551 616, with n=20 $2^{1,048,576} \approx 10^{300,000}$ functions)
4. We can‘t represent all these function realistically and need a hypothesis space bias.

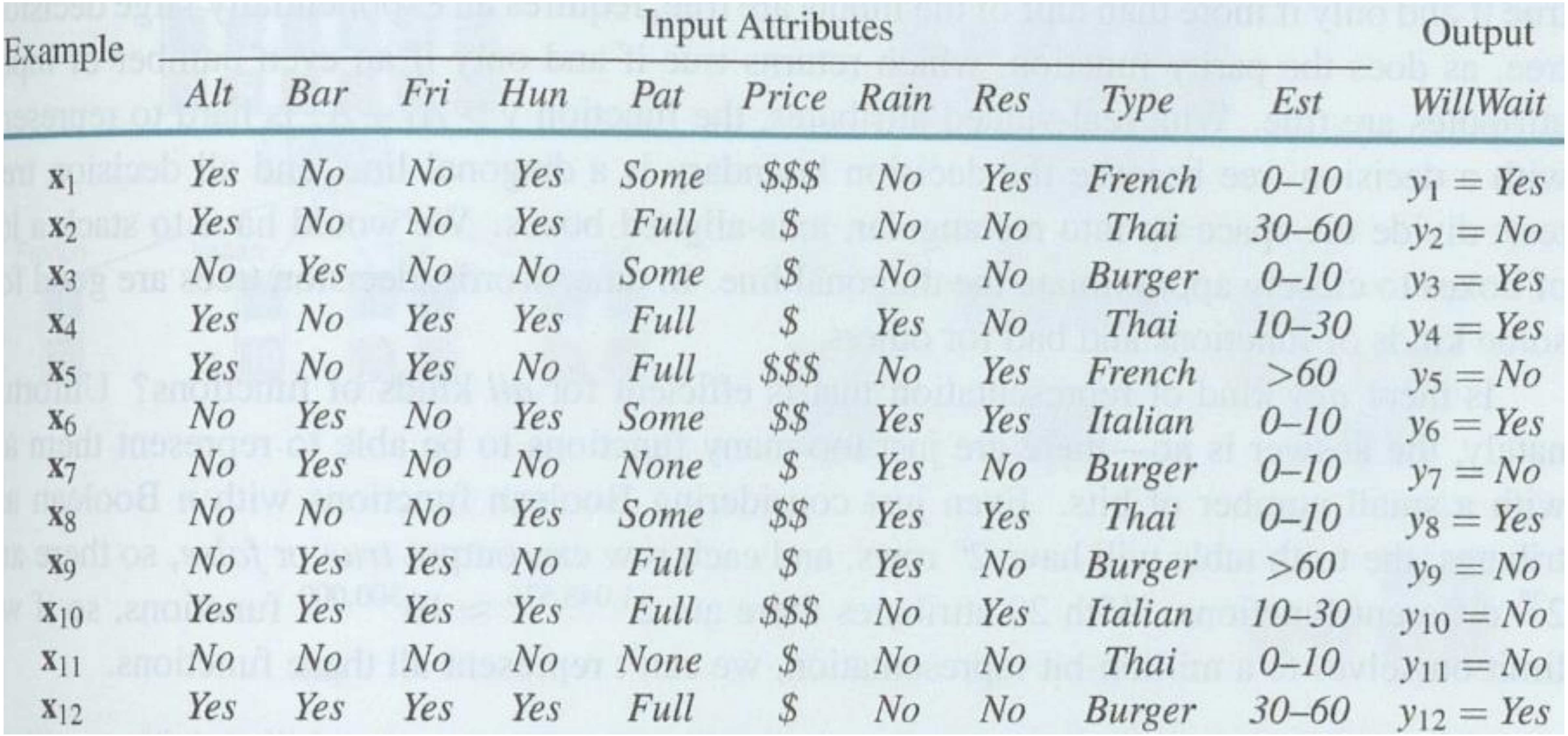
A	B		A∧B	A∨B	A⊕B	A	B	¬ A	…	
true	true		true	true	false	true	true	false		
true	false		false	true	true	true	false	false		
false	true		false	true	true	false	true	true		
false	false		false	false	false	false	false	true		

Example Restaurant domain

- **Problem of deciding whether to wait for a table at a Californian restaurant**
  - Input: Ten discrete attribute values
  - Output: Boolean variable „WillWait“

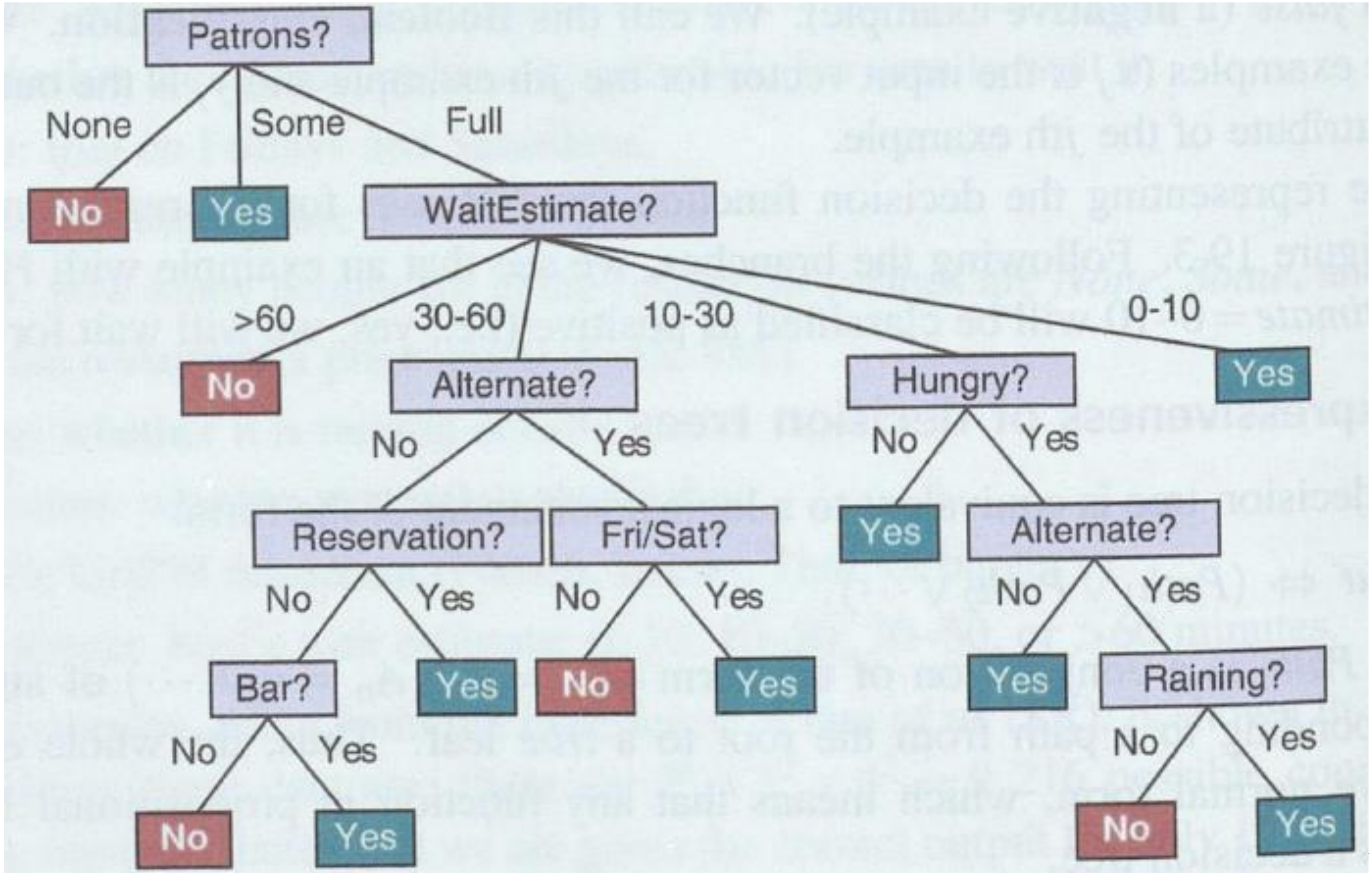
1. Alternate: Whether there is a suitable alternativ restaurant nearby
2. Bar: Whether the restaurant has a comfortable bar area to wait in
3. Fri/Sat: True on Fridays and Saturdays
4. Hungry: Whether we are hungry right now
5. Patrons: How many people are in the restaurant (None, Some, or Full)
6. Price: The restaurant‘s price range ($, $$, $$$)
7. Raining: Whether it is raining outside
8. Reservation: Whether we made a reservation
9. Type: The kind of restaurant (French, Italian, Thai, or burger)
10. WaitEstimate: Host‘s wait estimate (0-10, 10-30, 30-60, or >60 minutes)

12 Examples for the Restaurant Domain



Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

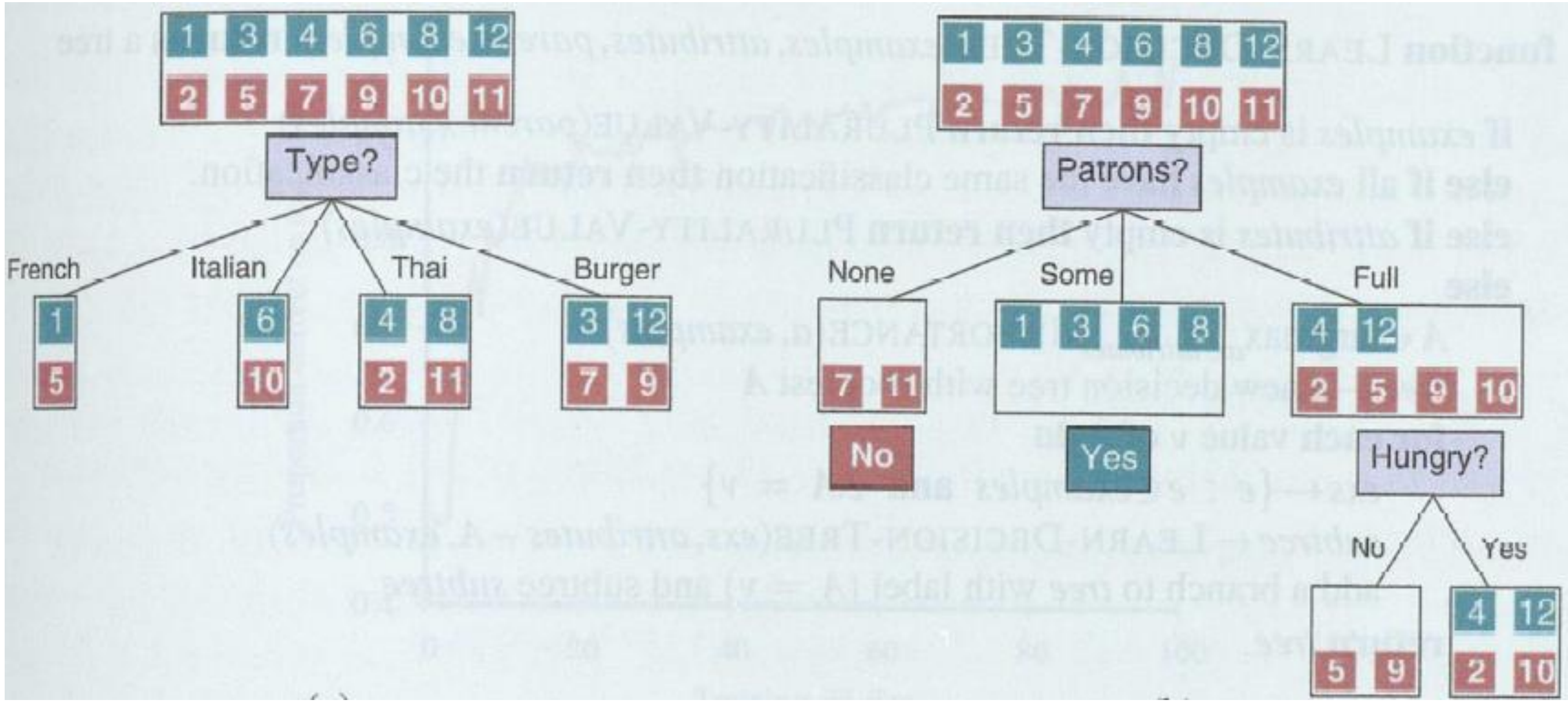
„True“ Decision Tree



Learning of Decision Trees (1)

- We want to find a tree consistent with the hypotheses.
- While finding the smallest decision tree requires exponential effort, there is a good greedy algorithm finding a tree close to the smallest:
  - Select the most important attribute first
  - Then solve recursively the smaller subproblems

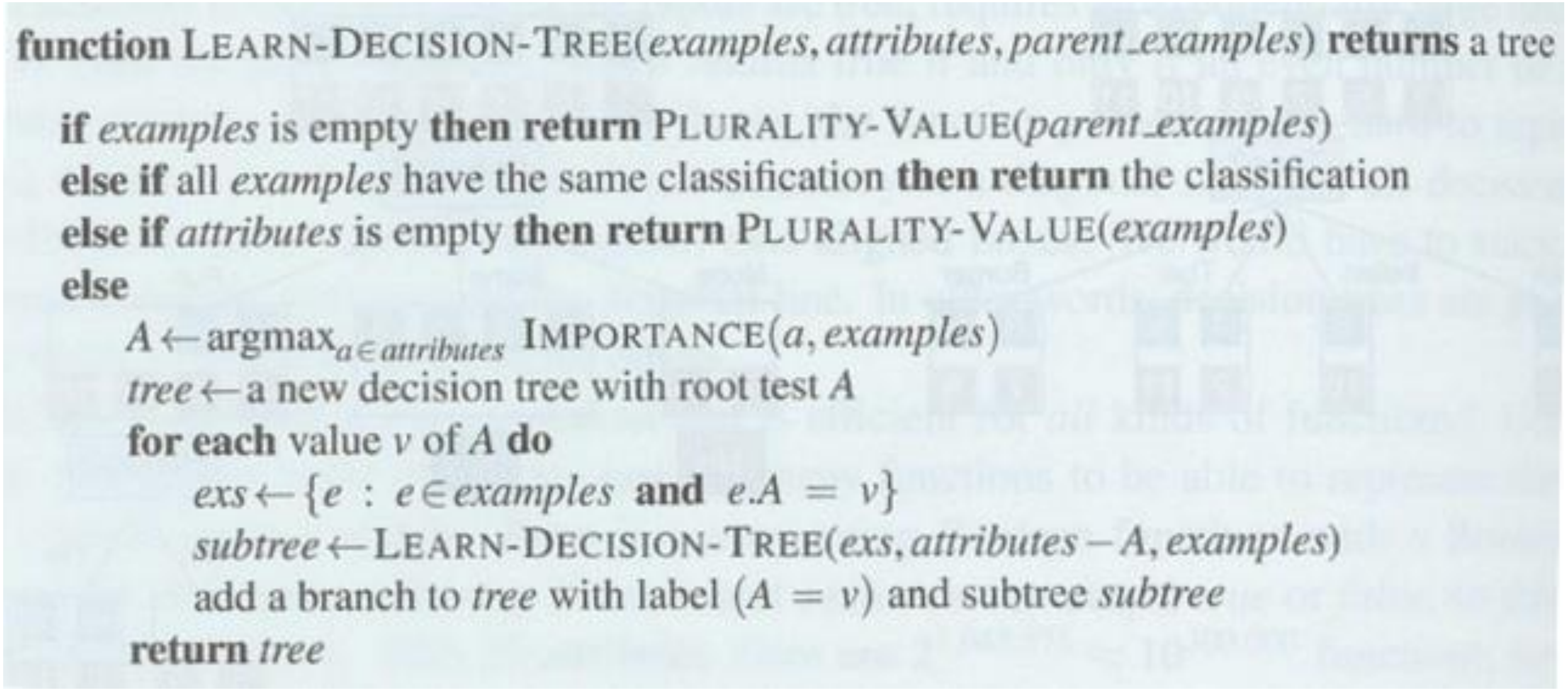
- What is the most important attribute?
  - Attribute, that splits the examples best for classification
  - e.g. „Patrons?“ splits better than „Type?“



Learning of Decision Trees (2)

- After selection of an attribute, four cases must be considered:
  - The remaining cases are all positive or negative: We are done!
  - If there are some positive and some negative examples, choose recursively the best attribute to split them
  - If there are no examples left (i.e. this combination of attributes had not been observed), return majority vote of examples in parent node
  - If there are no attributes left, but both negative and positive examples (i.e. there is noise in the data with identical examples having different classifications), return majority vote of examples

Decision tree learning algorithm

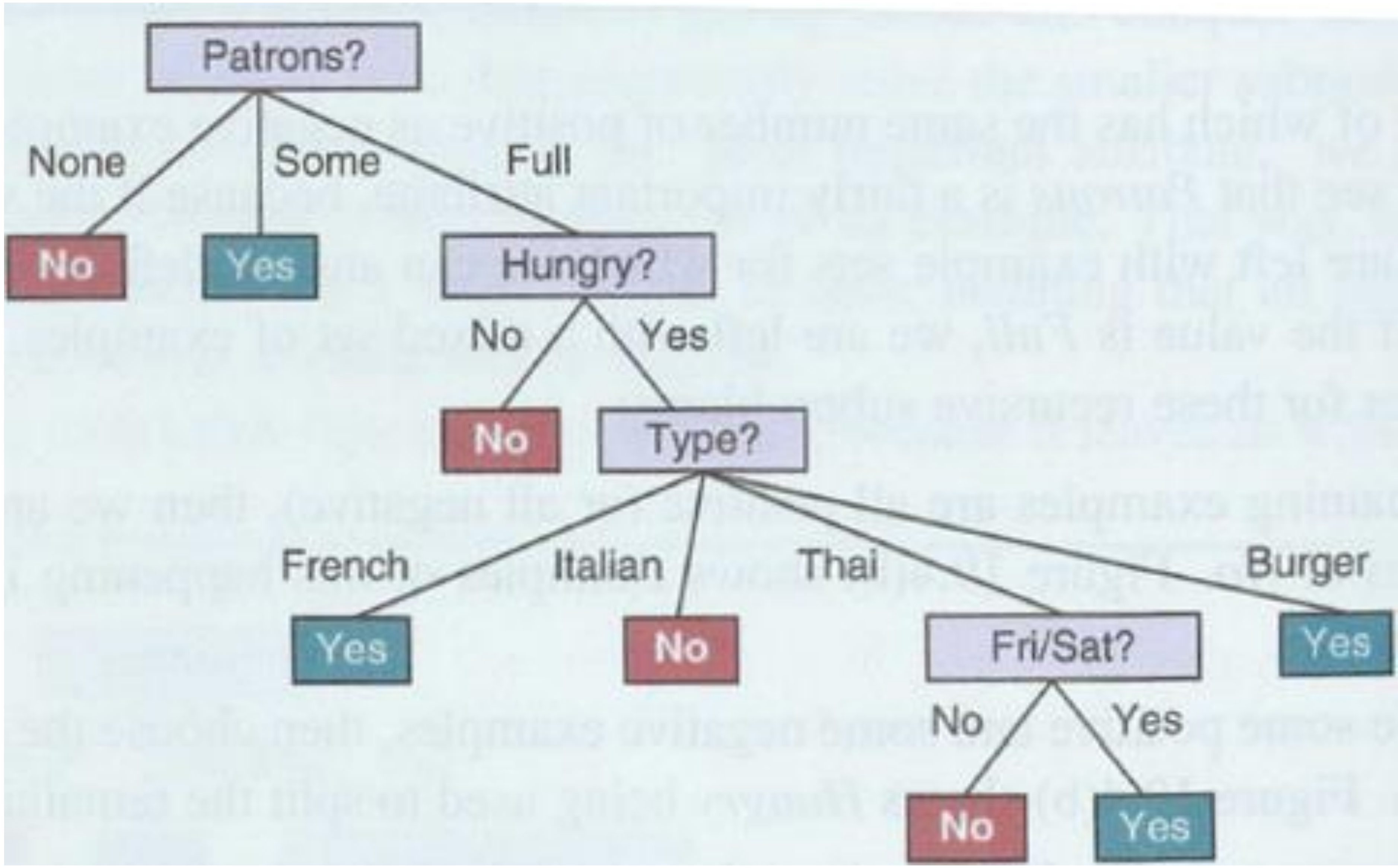


```
function LEARN-DECISION-TREE(examples, attributes, parent_examples) returns a tree

  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 ← a new decision tree with root test A
      for each value v of A do
          $exs \leftarrow \{e : e \in examples$ and $e.A = v\}$
          subtree ← LEARN-DECISION-TREE(exs, attributes − A, examples)
          add a branch to tree with label $(A = v)$ and subtree subtree
      return tree
```

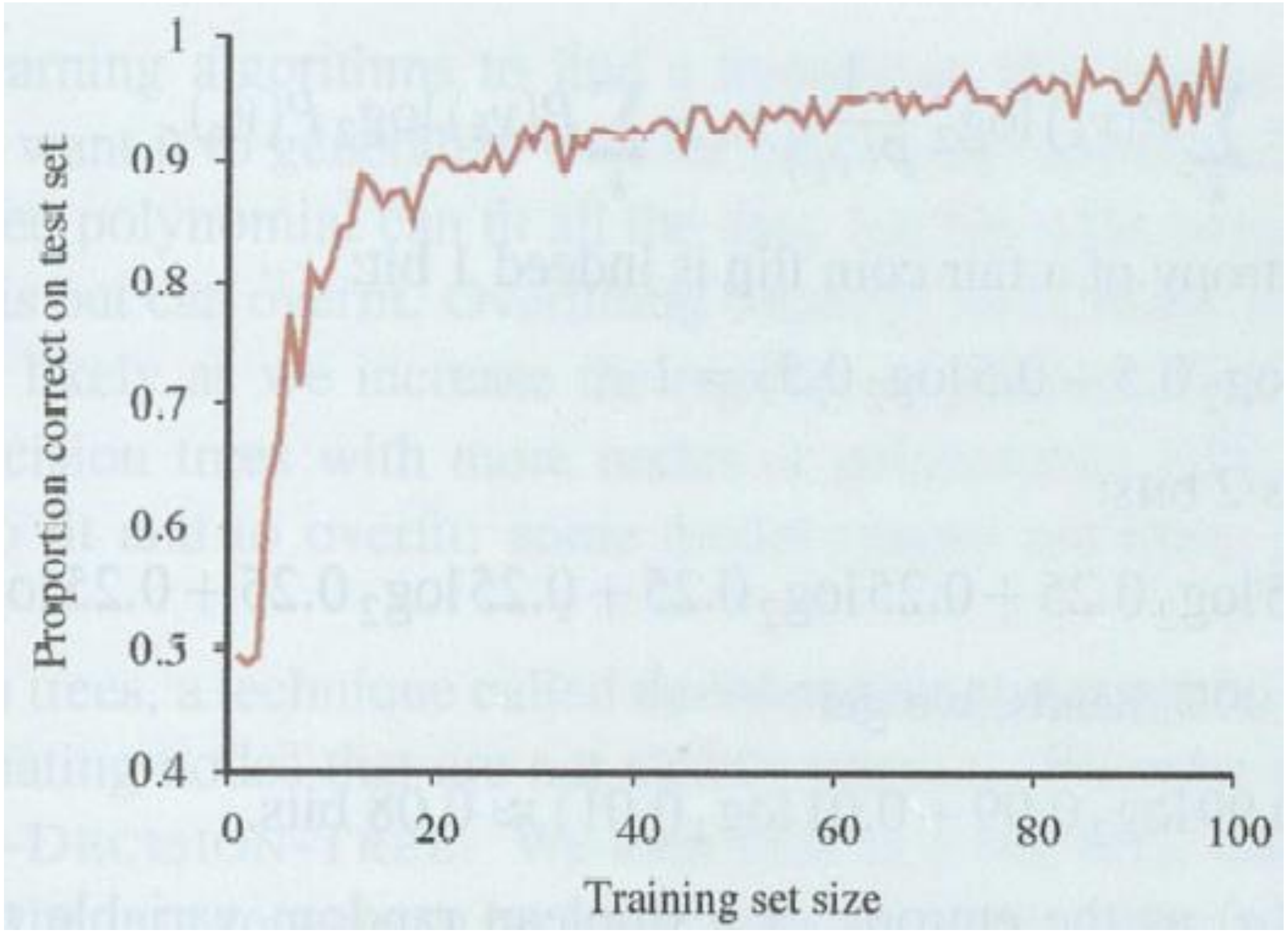
Learned Decision Tree Consistent with the 12 Examples

- Straightforward application of decision tree learning algorithm
- Learned decision tree smaller then „true“ decison tree
- According to Ockhams Razor better



Learning Curve

- Learning Curve for decision tree learning algorithm
- Training size: 0-100 generated training examples
- Each data point average of 20 trials where data is randomly split in training und test set
- Accuracy increases with training size („happy graph“)



Choosing Attribute Tests

- How should „importance“ of attribute be computed?
- Candidate: Information gain defined in terms of **entropy**
- Entroy is a measure of uncertainty of a random variable measured in bit
  - Flipping a fair coin has an entropy of 1 bit (50:50 chance of heads and tails)
  - Flipping a coin that always comes up heads has an entropy of 0
  - Rolling a 4-sided die has an entropy of 2
    - So far, entropy is the number of yes/no questions to ask
  - What is the entropy of 6-sided die resp. a coin, that comes up head in 99% of flips?
    - It should be a value between 2 and 3 resp. a positive value near zero
- **Definition of entropy:** $$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$
  - $H(\text{FairCoin}) = -(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}) = 1$ bit; $H(\text{Die4}) = 4 * (-(\frac{1}{4} \log_2 \frac{1}{4})) = 2$ bits
  - $H(\text{Coin99}) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01) \approx 0.08$ bits;
  - $H(\text{coin100}) = -(1 \log_2 1 + 0 \log_2 0)) = 0$ bits

Information Gain

- **Entropy of a Boolean variable** which is true with probability q:

  $B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$

- If a training set contains p positive examples and n negative examples, its entropy is:

  H (output variable of whole set) = $B(p/(p+n))$

  e.g. H(Restaurant training set) = $B(6/(6+6)) = B(½) = 1$

- An attribute A with d distinct values divides the training set E into subsets $E_1, \ldots, E_d$
  - Each subset E_k has p_k positive and n_k negative examples, i.e. B(subset E_K) = $B(p_k/(p_k+n_k))$
  - The probability of each subset is simply its share $(p_k+n_k) / (p+n)$
  - The expected entropy remaining after testing attribute A is:

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$$

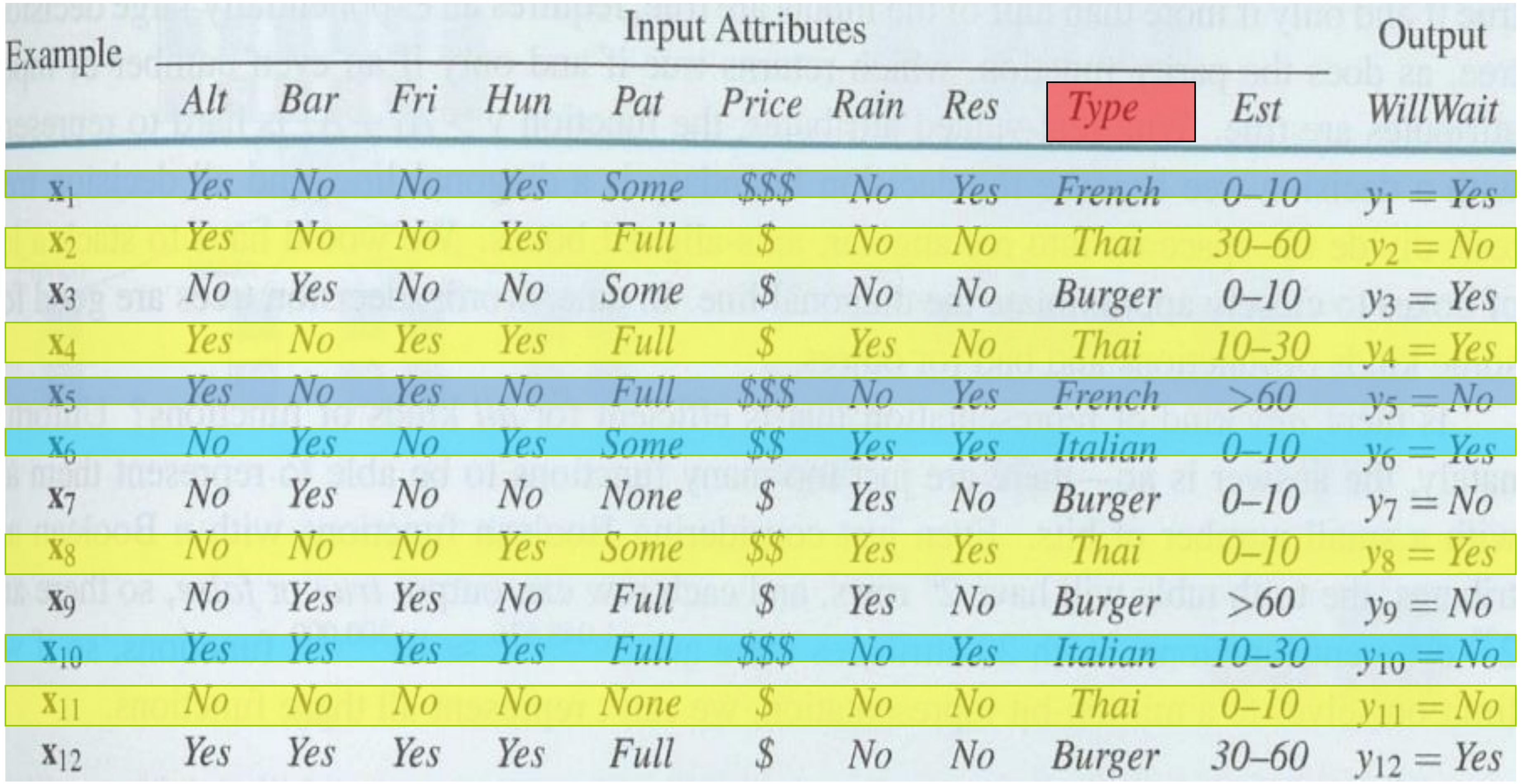
  - The information gain of attribute A is the expected reduction in entropy:

    $Gain(A) = B(p/(p+n)) - Remainder(A)$

Examples for Information Gain

- **Gain(A) = B(p/(p+n)) - Remainder (A) = B(p/(p+n)) -** $\sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$

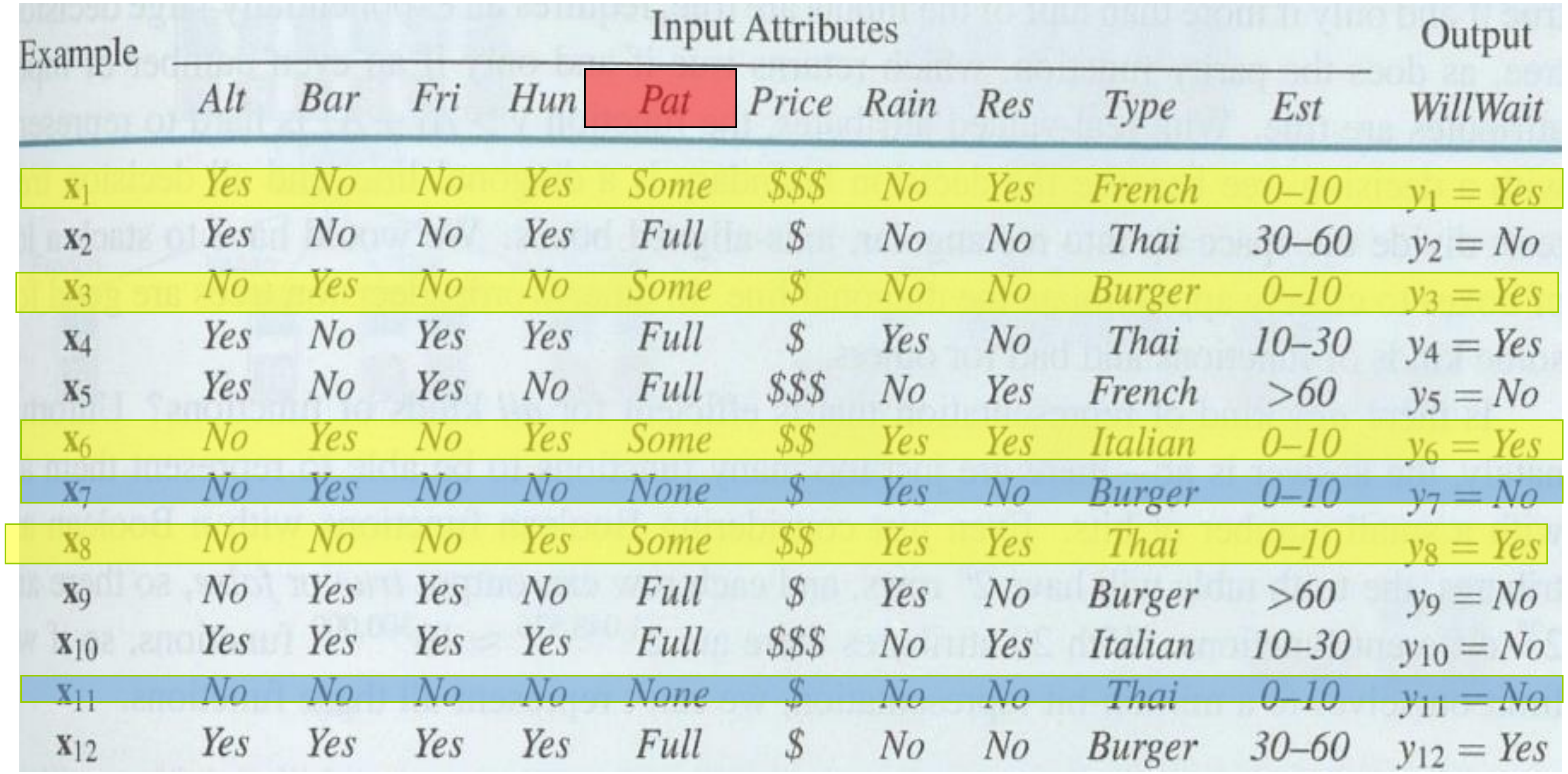
- Gain (Type) = 1 - [2/12 * B(½) + 2/12 * B(½) + 4/12 * B(½) + 4/12 * B(½))] = 0 bits



Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Examples for Information Gain

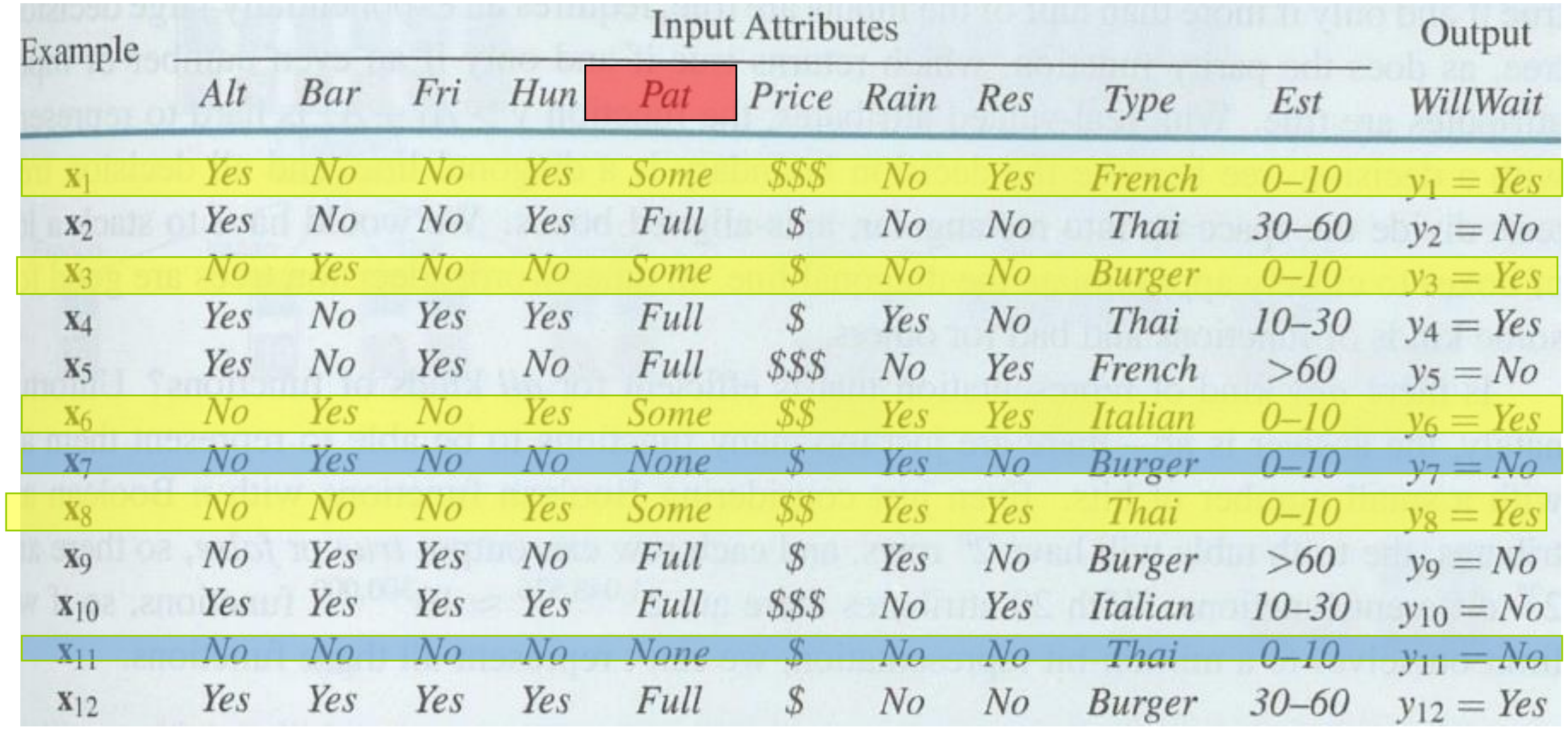
- **Gain(A) = B(p/(p+n)) - Remainder (A) = B(p/(p+n)) -** $\sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$
- Gain (Patrons) = 1 - [2/12 * B(0/2) + 4/12 * B(4/4) + 6/12 * B(2/6)] ≈ 0.541 bits



Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Course estimation of information gain:

- Attribute values with only positive or only negative examples have an entropy of zero
- If attributes with such „pure“ values exist, they have a big information gain
- A course estimation of the information gain is therefore the share of the number examples with pure values
    - e.g. Patron has two pure values: „Some“ with a share of 4/12 and „None“ with a share of 2/12 and therefore an information gain of at least 6/12, i.e. 0.5 bits



Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Generalization and Overfitting

- A decision tree has the capability to fit the data exactly, i.e. to memorize each example
- **Overfitting** is a problem of most learning algorithms
  - More likely with higher number of attributes and less likely with increasing number of training examples
  - More likely with higher hypothesis spaces (i.e. more parameters, e.g. decision trees with more nodes or polynomial functions with high degree)
- For decision trees, a standard techniques called **decison tree pruning** combats overfitting
  - Pruning eleminates irrelevant nodes in the decision tree
  - It starts with a full tree and checks bottom-up all leave nodes with a **significance test**
    - If the **null hypothesis,** that the decision was made by chance, cannot be refused, the node is deleted
    - Candidate for significance test: **Chi-Square test (χ^2)**

Rationale for Chi-Square test (χ^2)

- Decision in decision tree:
  - A set of p positive and n negative examples is divided by attribute values in subsets
  - If the attribute is irrelevant, the subsets have a similar distribution as the (super)set
- Rationale: Compute for each subset the difference between the expected numbers of positive and negative examples and the observed numbers
- **Chi-Square test:** Take the sum of the square of the differences

Chi-Square test (χ^2)

- Let N_k be the total number of examples in the subset k, and p_k and n_k the number of positive and negative examples ($N_k = p_k + n_k$)
- Let N be the total number of examples in the superset, and p and n the number of positive and negative examples (N = p + n)
- The expected numbers of positive and negative examples in each subset k is:

$$\hat{p}_k = p * \frac{N_k}{N} \quad \text{and} \quad \hat{n}_k = n * \frac{N_k}{N}$$

- The Chi-square measure of the total deviation is:

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

- The value Δ is interpreted with χ^2 statistics with d-1 degrees of freedom
  - The restaurant *Type* attribute has four values and thus 3 degrees of freedom. A value of $\Delta \geq 7.82$ would reject the null hypothesis with an error rate of 5% or less.
  - Attribute *Patrons* has three values and needs $\Delta \geq 5.99$ for significance
  - Attribute *Hungry* has two values and needs $\Delta \geq 3{,}84$ for significance

Pruning versus Early Stopping

- What is the reason to generate a decision tree with information gain and then to prune it with Chi-Square test instead of combining them in one step and stop early, when there is no significant attribute for splitting?
- It would not work for combinations of attributes:
  - If e.g. two attributes are necessary (like the XOR-function) for the decision, the first attribute is not significant, but in combination with the second attribute it is.

Broadening the Applicability of Decision Trees

- Decision trees can be made more widely useful by handling the following complications:
  - **Missing data**: Attribute values might be unknown in a case. Problematic for both classification of a new case and building the decision tree.
    - Possible solutions: **Estimating the values**; for classification: branching at missing value and computing the probability of branches
  - **Continuous input attributes** (e.g. Height, Weight, Time, etc.): Compute a split point
    - Sort values and choose split points separating positive and negative examples on each side
  - **Multivalued input attributes**: branching often not sensible (e.g. for ID- or ZIP-Attribute). If ordered may be treated like continuous attributes, otherwise group (e.g. the most important value and the rest) or ignore them
  - **Continuous-valued output attribute**: For numeric output, regression is the method of choice. But instead of just one function, it is possible to learn different regressions (e.g. linear functions) for different conditions, i.e. a **regression tree**

Discussion of Decision Trees

+ Ease of understanding
+ Scalability to large data sets
+ Versatility in handling discrete and continuous attributes
+ Explainability

– Suboptimal accuracy (largely due to greedy search)
– Problems with high degree of missing data
– Deep decision trees can be expensive to run
– Problems with online learning (e.g. adding a new case might change the root node)

- Improvement: Random Forest (see ensemble learning)

Model Selection and Optimization

- Goal in machine learning: Select a hypothesis that will optimally fit future examples
  - **Stationary assumption:** Future examples will be like the past
  - Seperation in two processes:
    - **Model selection** (better „model class selection“) to find a good hypothesis space
    - **Optimization** to find the best hypothesis within this space

- Part of model selection is qualitative and subjective (kind of preselection of plausible learning algorithms for a task) and part can be done with the same automatic process as optimiziation

Optimization

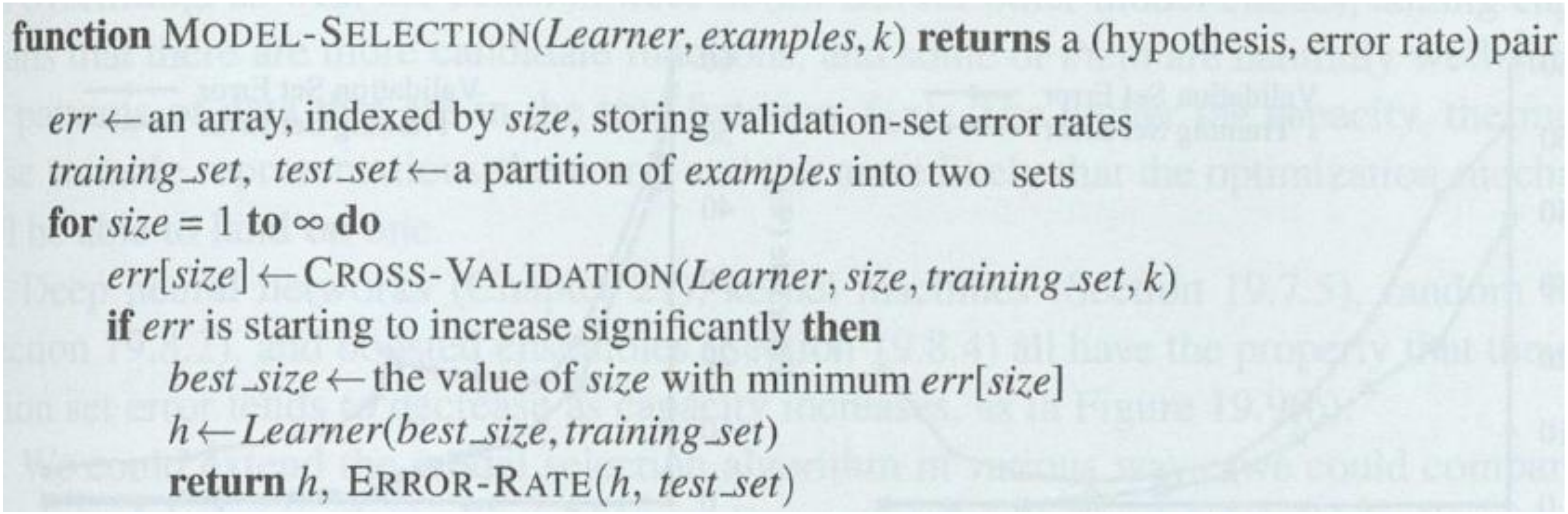
- Optimal fit: Compute **error rate** on **test set** with different examples as in **training set**
  - Often we test multiple hypotheses generated with different „knobs“ (**hyperparamters**) of a learning method, e.g. decision trees with different thresholds for pruning
  - We should reserve the test set for the final test and compare the effect of different hyperparameters with another separate **validation set** (**development set** or **dev set**)
  - In effect, we split the available data in three sets (e.g. 60:20:20):
    - **Training set** to train candidate models
    - **Validation set** to evaluate the candidate models and choose the best one
    - **Test set** to do the final unbiased evaluation of the best model
- Without validation set, the best hypothesis from a set of many hypotheses due to different combinations of hyperparameters would appear to be too good on the test set
  - e.g. from 100 random hypotheses about 5 would appear to be significant by chance
  - To avoid this problem, the test set should be used only once.

k-fold Cross Validation

- Without enough data, **k-fold cross validation** can help (increases computation time!):
  - Perform k rounds of learning: on each round 1/k of data is used for validation
  - Average score on validation set is used for choosing the best model
- Typical values for k: 5 or 10,
- Extreme case: **Leave-one-out cross validation**: n rounds with just one case for validation

Algorithm for model selection

- Builds increasingly more complex models set by the parameter „size“
- Chooses the best model with the lowest validation error rate
- Returns the best model together with its error rate on the held-out test examples

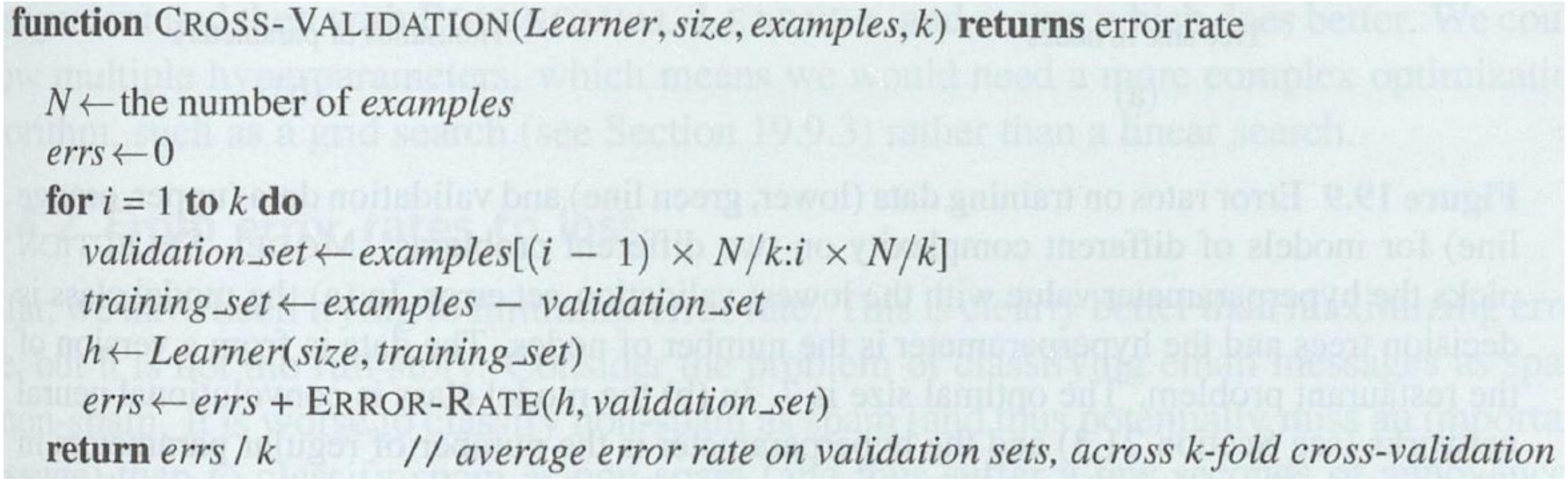


```
function MODEL-SELECTION(Learner, examples, k) returns a (hypothesis, error rate) pair

  err ← an array, indexed by size, storing validation-set error rates
  training_set, test_set ← a partition of examples into two sets
  for size = 1 to ∞ do
      err[size] ← CROSS-VALIDATION(Learner, size, training_set, k)
      if err is starting to increase significantly then
          best_size ← the value of size with minimum err[size]
          h ← Learner(best_size, training_set)
          return h, ERROR-RATE(h, test_set)
```

Algorithm for Cross Validation

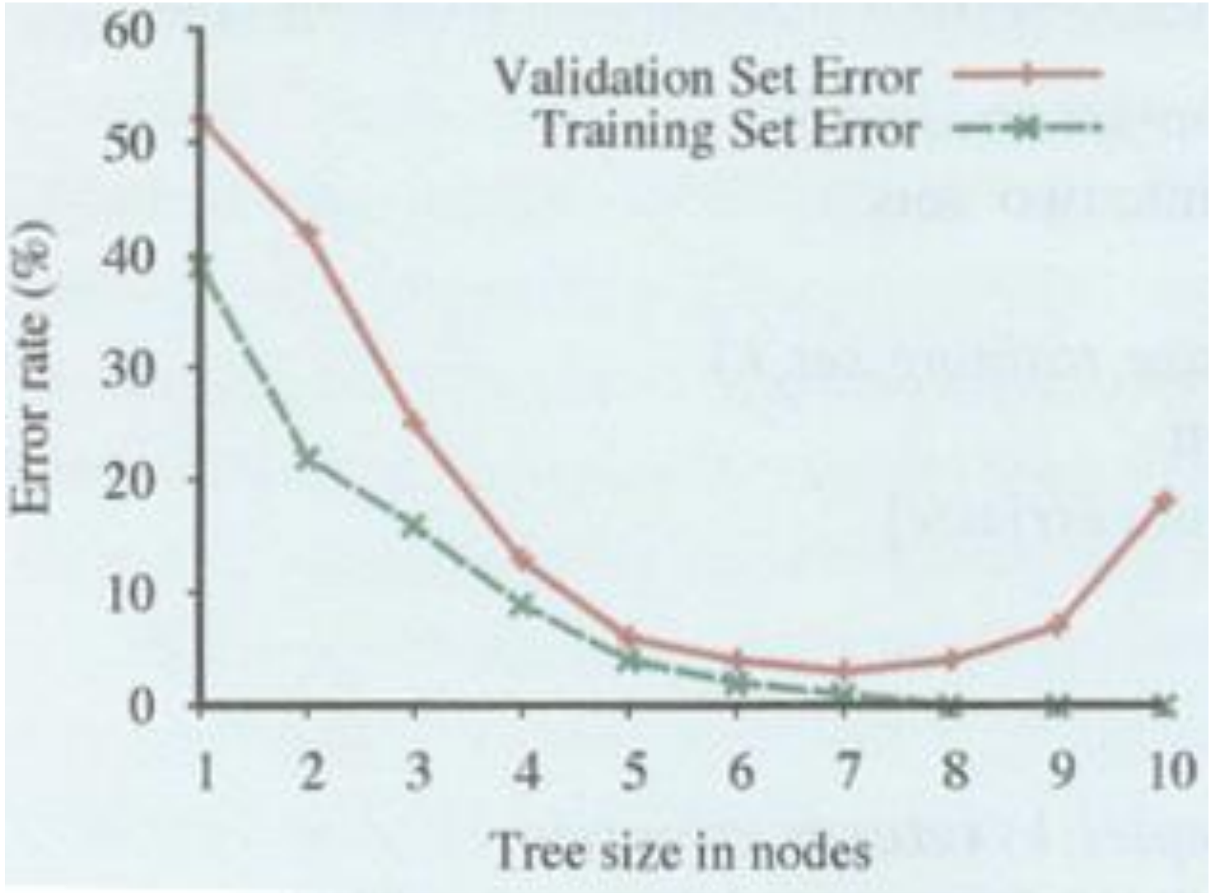
- Each iteration of the for-loop selects a different slice of the examples as validation set

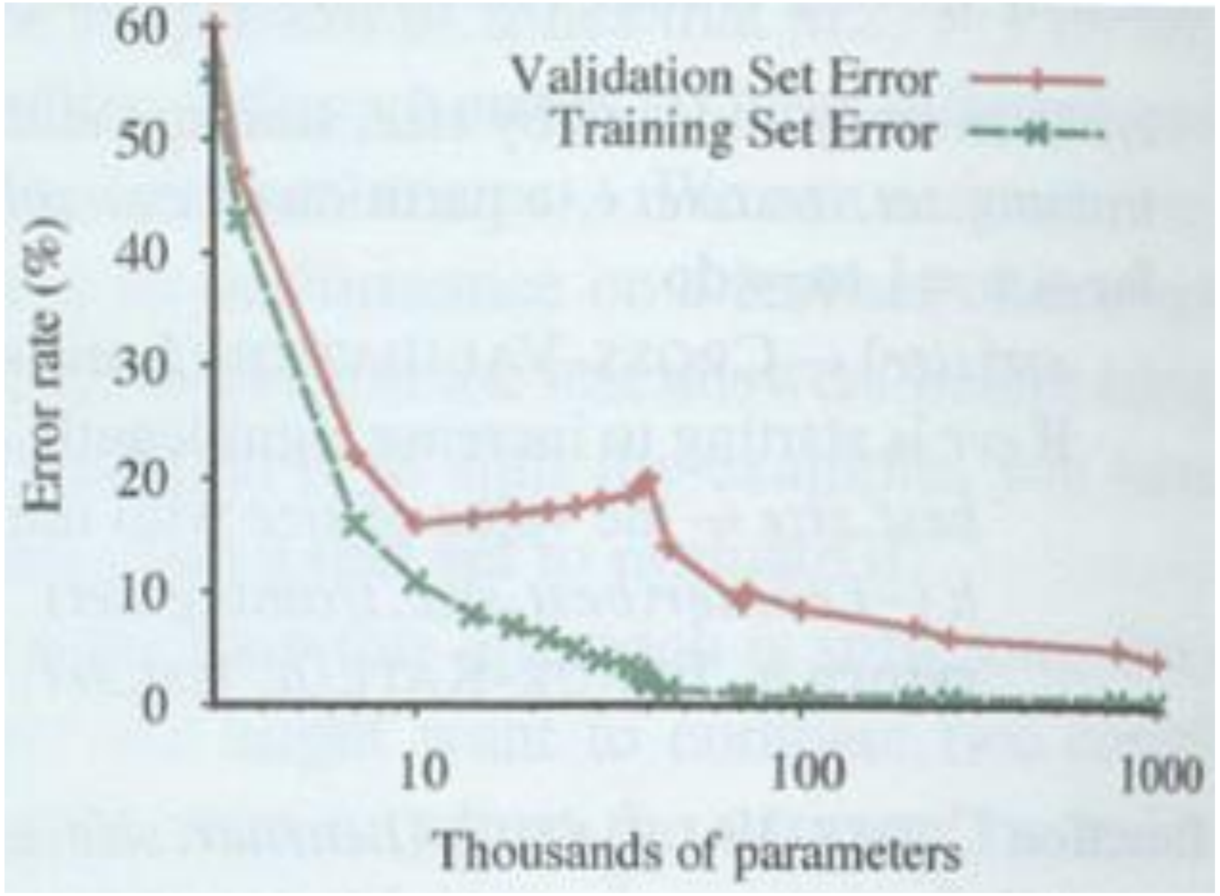


```
function CROSS-VALIDATION(Learner, size, examples, k) returns error rate

  N ← the number of examples
  errs ← 0
  for i = 1 to k do
    validation_set ← examples[(i − 1) × N/k : i × N/k]
    training_set ← examples − validation_set
    h ← Learner(size, training_set)
    errs ← errs + ERROR-RATE(h, validation_set)
  return errs / k        // average error rate on validation sets, across k-fold cross-validation
```

Error Rates on Training and Validation Data





Restaurant domain with decison tree :

Hyperparameter: Depth of decison tree

Overfitting with depth higher than 7

Image (digit) classification with CNN-Net:

Hyperparameter: # parameters (weights)

No overfitting with up to 1,000,000 parameters

From Error Rate to Loss

- Error rate assumes, that all errors are equal important
  - However, in many applications, some error types are more important than others
  - e.g. in spam classification or anonymization, classifying a relevant mail as spam or missing a name for anonymization is much more severe than the other way round
- Define a **loss function** and minimize it (equivalent to maximizing expected utility)
  - General form of loss function L(x, y, $\hat{y}$) is defined as the amount of utility lost by predicting h(x) = $\hat{y}$, where the correct answer is f(x) = y
  - Usually simplified to L(y, $\hat{y}$) independant of x
    - Example: L(spam, nospam) = 1;   L(nospam, spam) = 10
  - Loss for continuous output: (Squared) difference of y and $\hat{y}$
- Widespread loss functions:
  - **Absolute-value loss:** $\mathbf{L_1}(y, \hat{y}) = |y - \hat{y}|$
  - **Squared-error loss:** $\mathbf{L_2}(y, \hat{y}) = (y - \hat{y})^2$
  - **0/1 loss:** $\mathbf{L_{0/1}}(y, \hat{y}) = 0$ if $y = \hat{y}$, else 1

Empirical Loss

- Expected generalization loss is the sum of the loss L over all input-output examples weighted by a probability distribution over the examples
- The best hypothesis h* minimizes the expected generalization loss
- Since the probability distribution is usually not known, the **empirical loss** is estimated with

  a set of available examples E of size N:

$$EmpLoss_{L,E}(h) = \sum_{(x,y)\in E} L(y, h(x)) \frac{1}{N}$$

- minimized by the estimated best hypothesis:

$$\hat{h}^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, EmpLoss_{L,E}(h)$$

Regularization

- Complex hypotheses h tend to overfit the data and should be penalized:

  Cost (h) = EmpLoss(h) + λ Complexity (h)

  $\hat{h}*$ = hypothesis with minimal Cost(h)

  where λ is a hyperparameter balancing loss and complexity of a hypothesis
- Penalizing complex hypotheses is called **regularization** (we are looking for a hypothesis more regular)
- The choice of regularization depends on the hypothesis space (e.g. depth of a decision tree or sum of coefficiens for polynomial functions)

- Another way to simplify models is **feature selection** (discard apparently irrelevant attributes)

Hyperparameter Tuning

- With a small number of hyperparameters a systematic search with cross-validation is possible
- If there are too much combinations or they have continuous values, it is more difficult
- Approaches:
    - **Hand-Tuning**: Guess parameters on past experience (or publications) and use intuition for improvement
    - **Grid search** for systematic search (running parallel on different machines if available)
    - **Bayesian optimiziation** with a belief network over the results of the runs
    - **Population based training** (genetic algorithm): Combine the hyperparameters of successful results of the runs

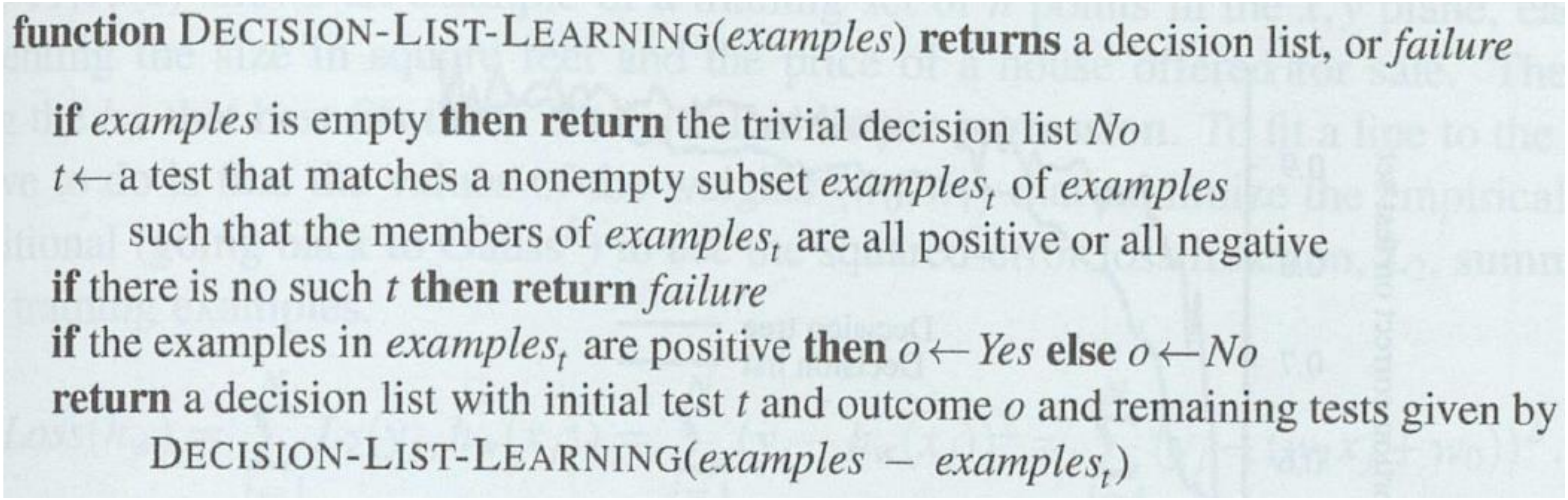
Theory of Learning

- **Stationary assumption:** Training examples and future examples are drawn from the same fixed distribution
  - Could not taken for granted in an evolving world
- How much training examples are necessary for a **probably approximately correct hypothesis (PAC)** with an error rate smaller than ε without further restricting the hypothesis space?
  - Surprising answer: all examples (e.g. for learning Boolean functions), since the hypothesis space is double exponential:
    - For n attributes, there are $2^n = c$ value combinations. A boolean function needs for each value combination c a set of truth values, with 2^c combinations resuting in 2^{2^n} functions, e.g. for there are 16 boolean functions for two variables a and b.
    - We need $\approx \log(2^{2^n}) \approx 2^n$ examples for finding a hypothesis with an error rate smaller than ε with a high probability but there exist only 2^n examples.
  - Known solution: Restrict hypothesis space or penalize complex hypotheses

PAC Learning Example: Learning Decision Lists

- Decision trees can be turned in decision lists, if each path in the decision tree is viewed separately, e.g. WillWait $\Leftrightarrow$ (Patrons = Some) $\vee$ (Patrons = Full $\wedge$ Fri/Sat)
- With decision lists of arbritrary size, any Boolean function can learned
- With decision lists restricted to at most k literals, we restrict the hypothesis space
  - The learning algorithm is forced to generalize
  - The number of necessary examples is only polynomial in the number of attributes n for small values of k (proportial to n^k)

Straightforward Algorithm for Learning Decision Lists

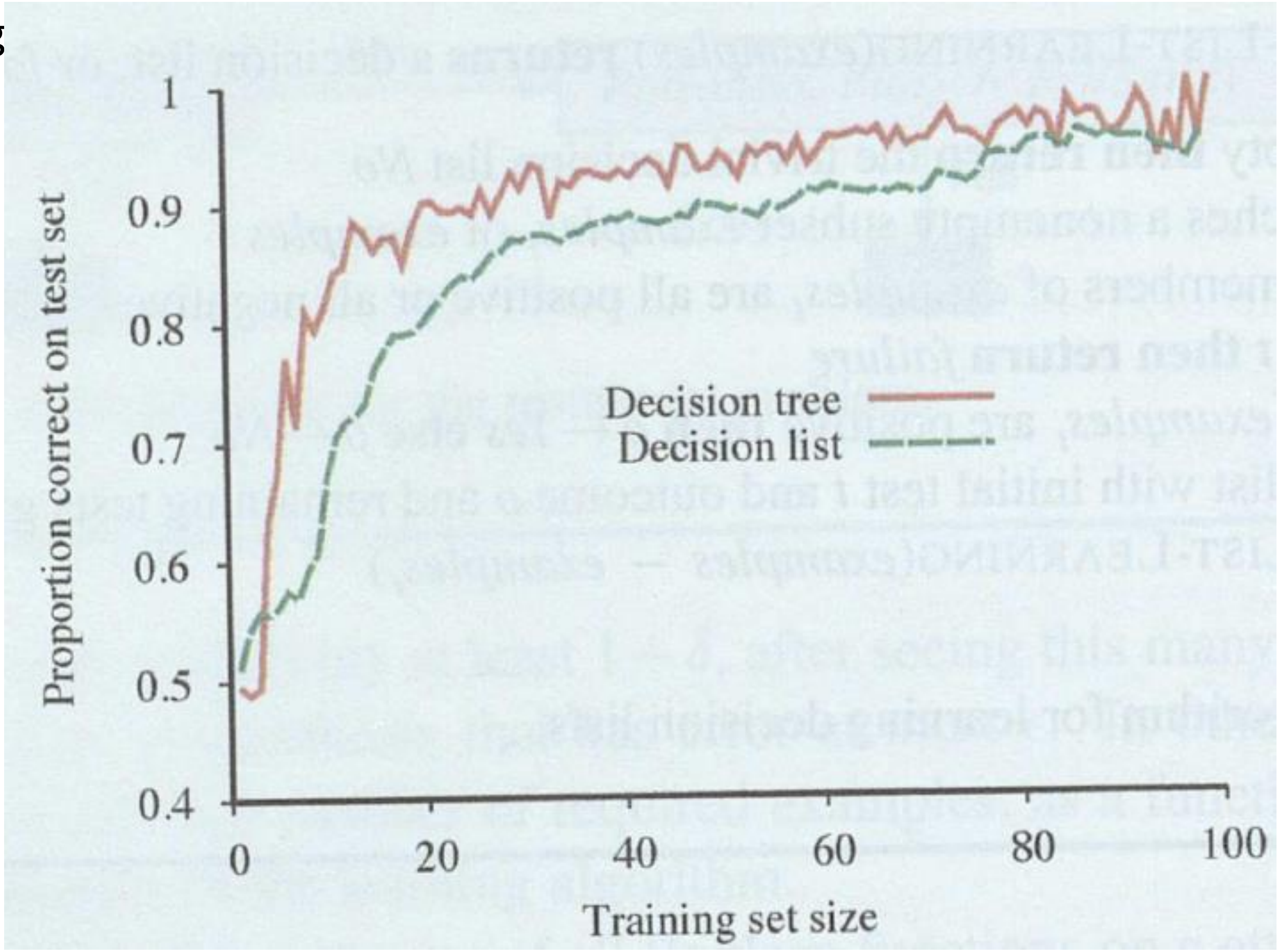


```
function DECISION-LIST-LEARNING(examples) returns a decision list, or failure

  if examples is empty then return the trivial decision list No
  t ← a test that matches a nonempty subset $examples_t$ of examples
        such that the members of $examples_t$ are all positive or all negative
  if there is no such t then return failure
  if the examples in $examples_t$ are positive then o ← Yes else o ← No
  return a decision list with initial test t and outcome o and remaining tests given by
       DECISION-LIST-LEARNING($examples - examples_t$)
```

Comparison of Learning Curves

Decision tree learning slightly better than decision list learning with up to 100 examples in training set.


Proportion correct on test set
Training set size
Decision tree
Decision list