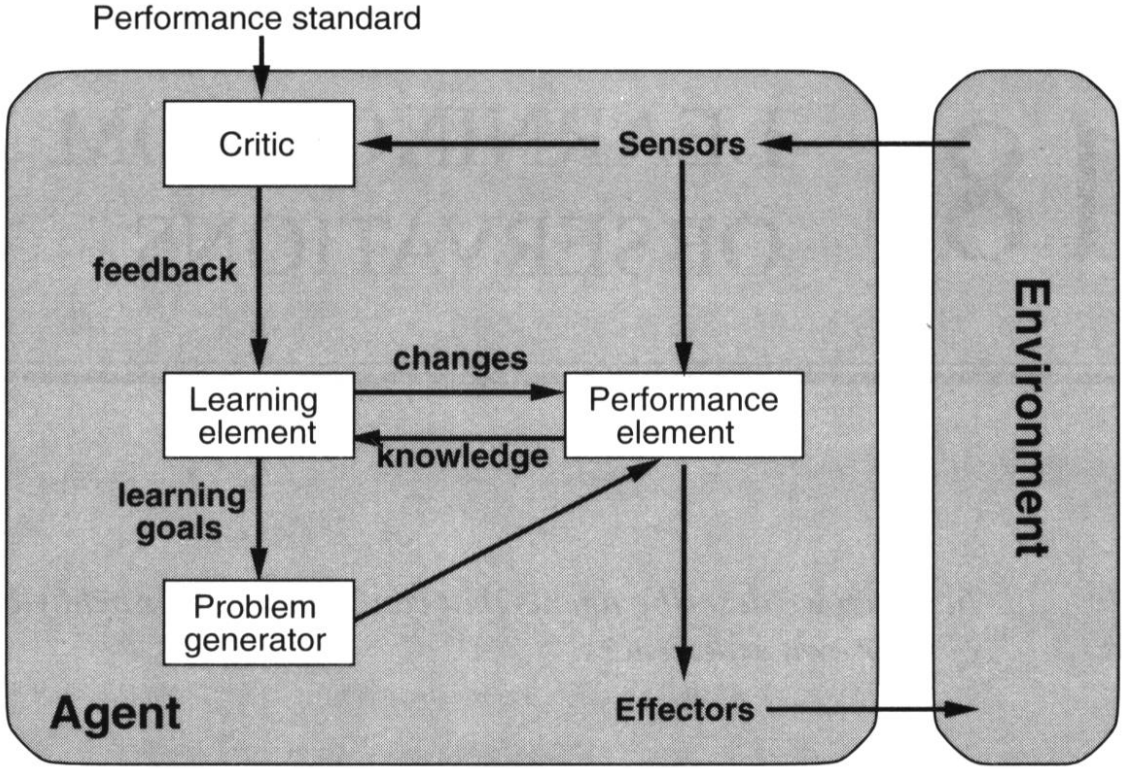
Overview

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

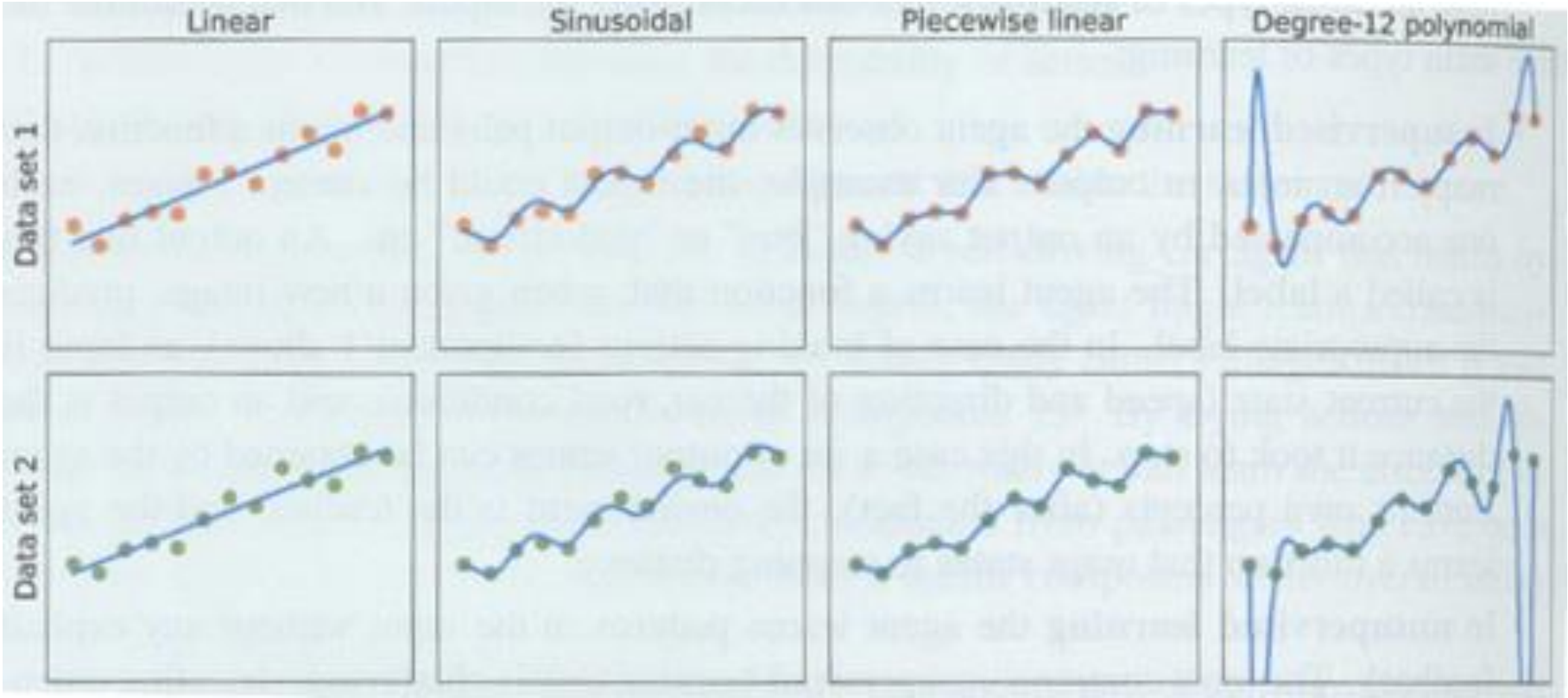
- *Given:* A training set of N example input-output pairs (x_1, y_1), (x_2, y_2), …. (x_N, y_N),
  - Where each pair was generated by an unknown function y = f(x)
- *Sought*: A function h (**hypothesis**) that approximates the true function f
  - Hypothesis is drawn from a usually constrained **hypothesis 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



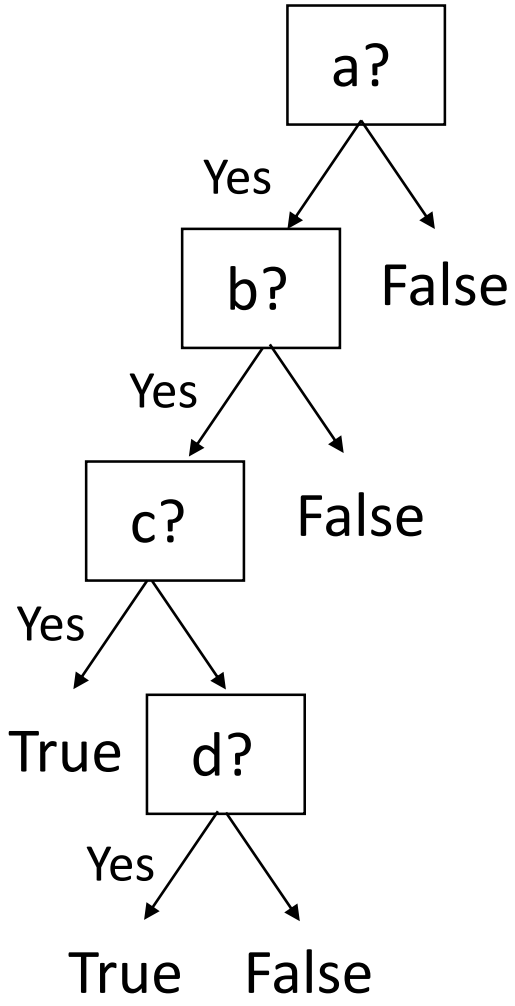
Consequences

- Learning requires some **bias:**
- **Tradeoff** beween **expressiveness** of hypothesis and **efficieny** of learning algorithm unavoidable
- **Data bias:** The available training data is not representive 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 $\Leftrightarrow$ ($Path_1 \vee Path_2 \vee \ldots$)
  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		

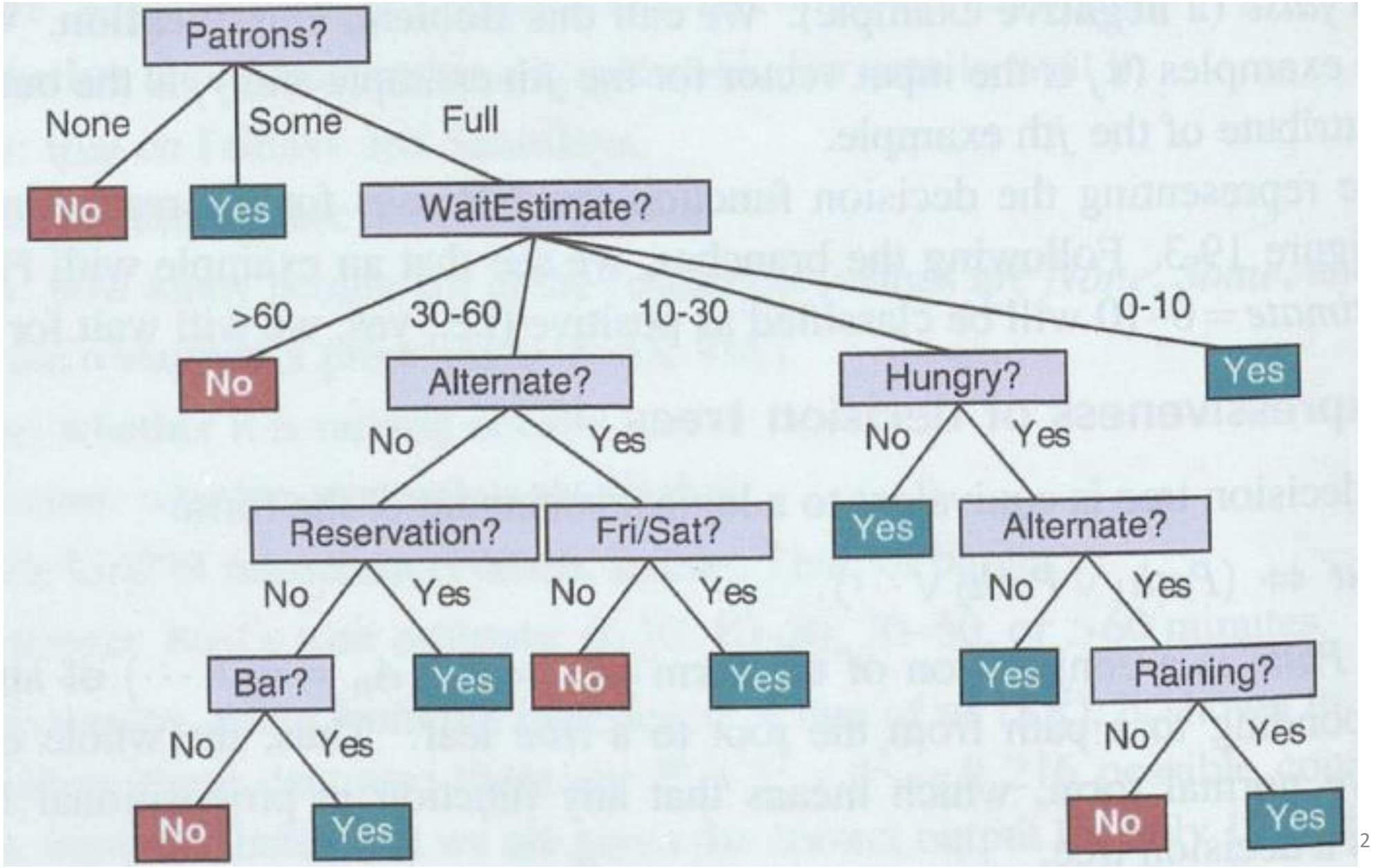
Example Restaurant domain

- **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*	$y_{12} = Yes$

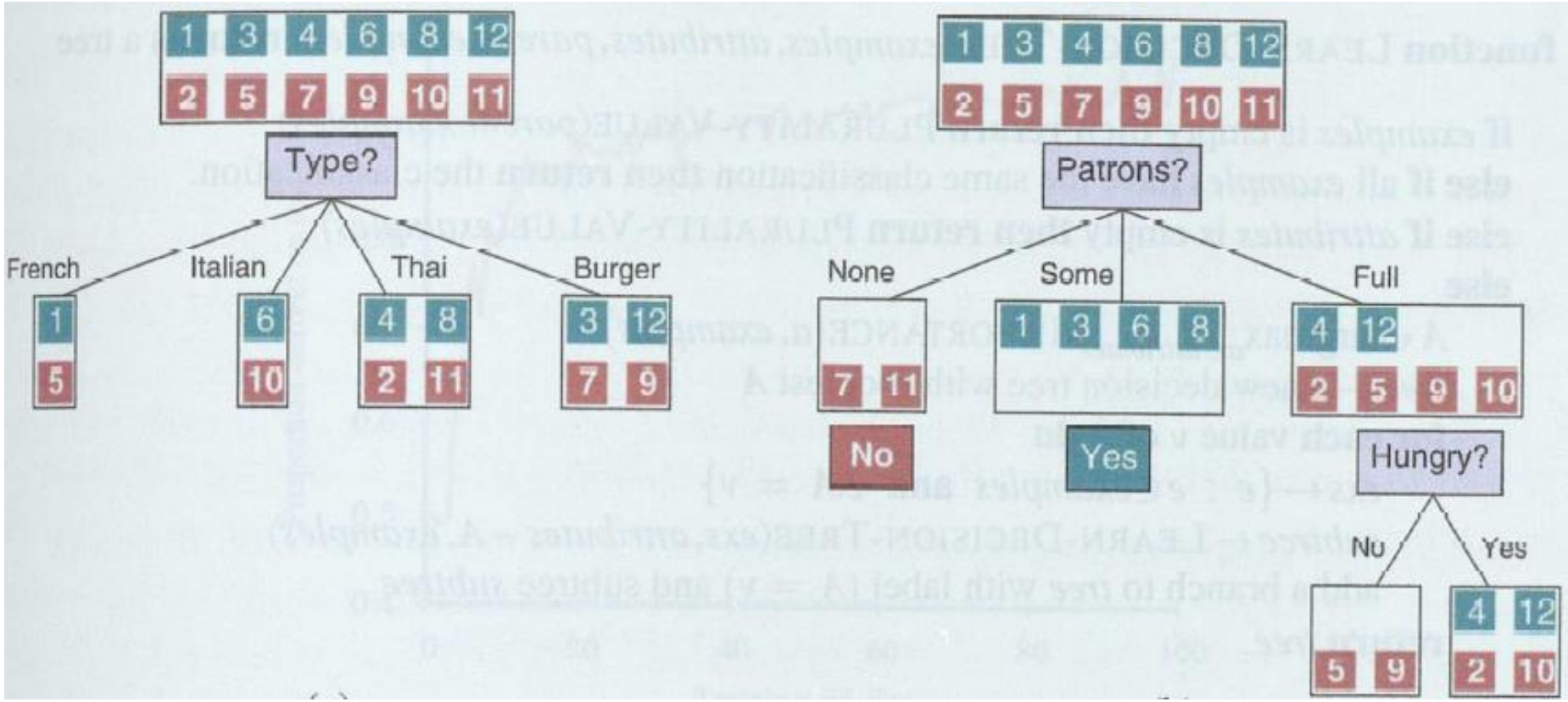
„True“ Decision Tree



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

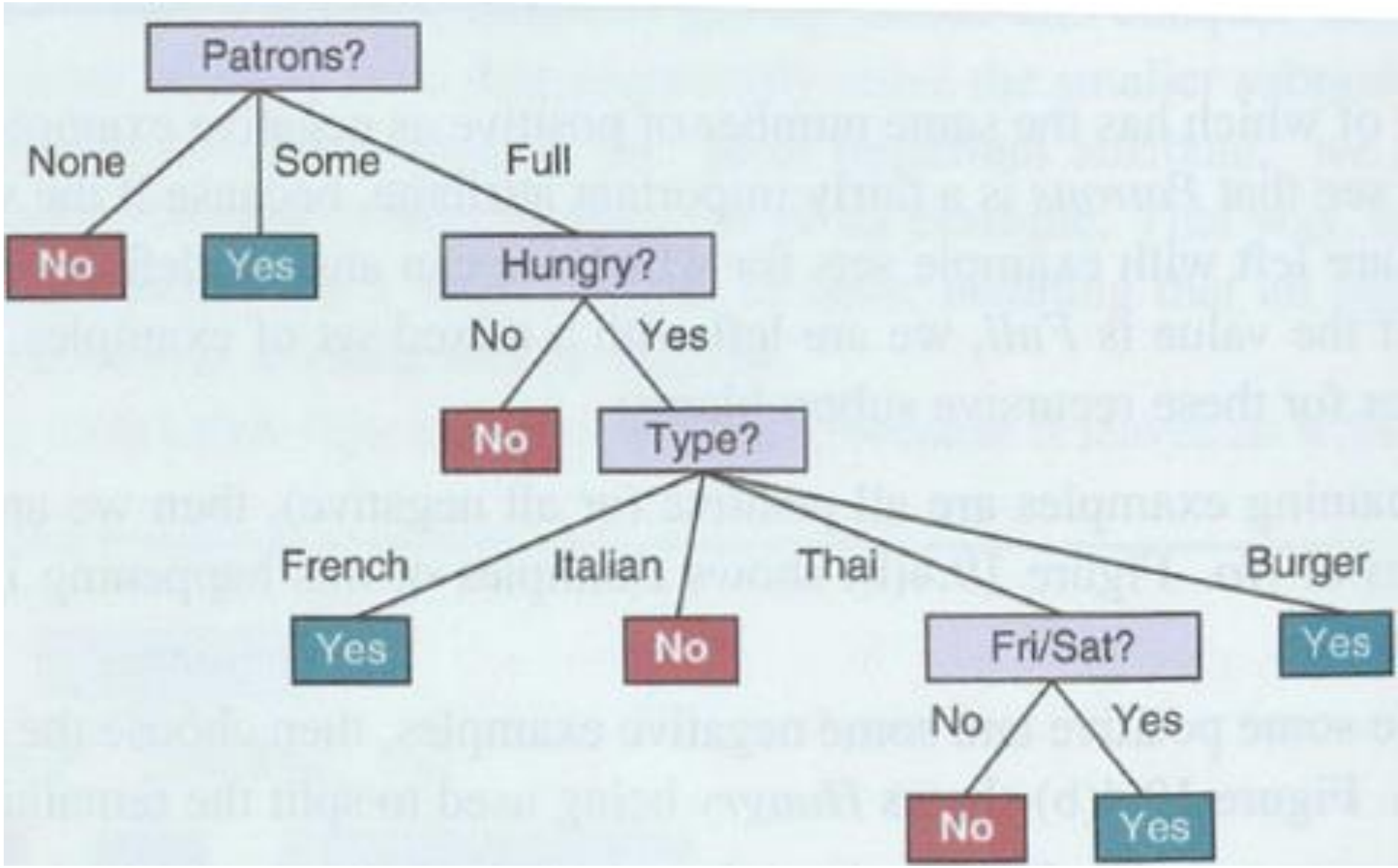
Decision tree learning algorithm

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

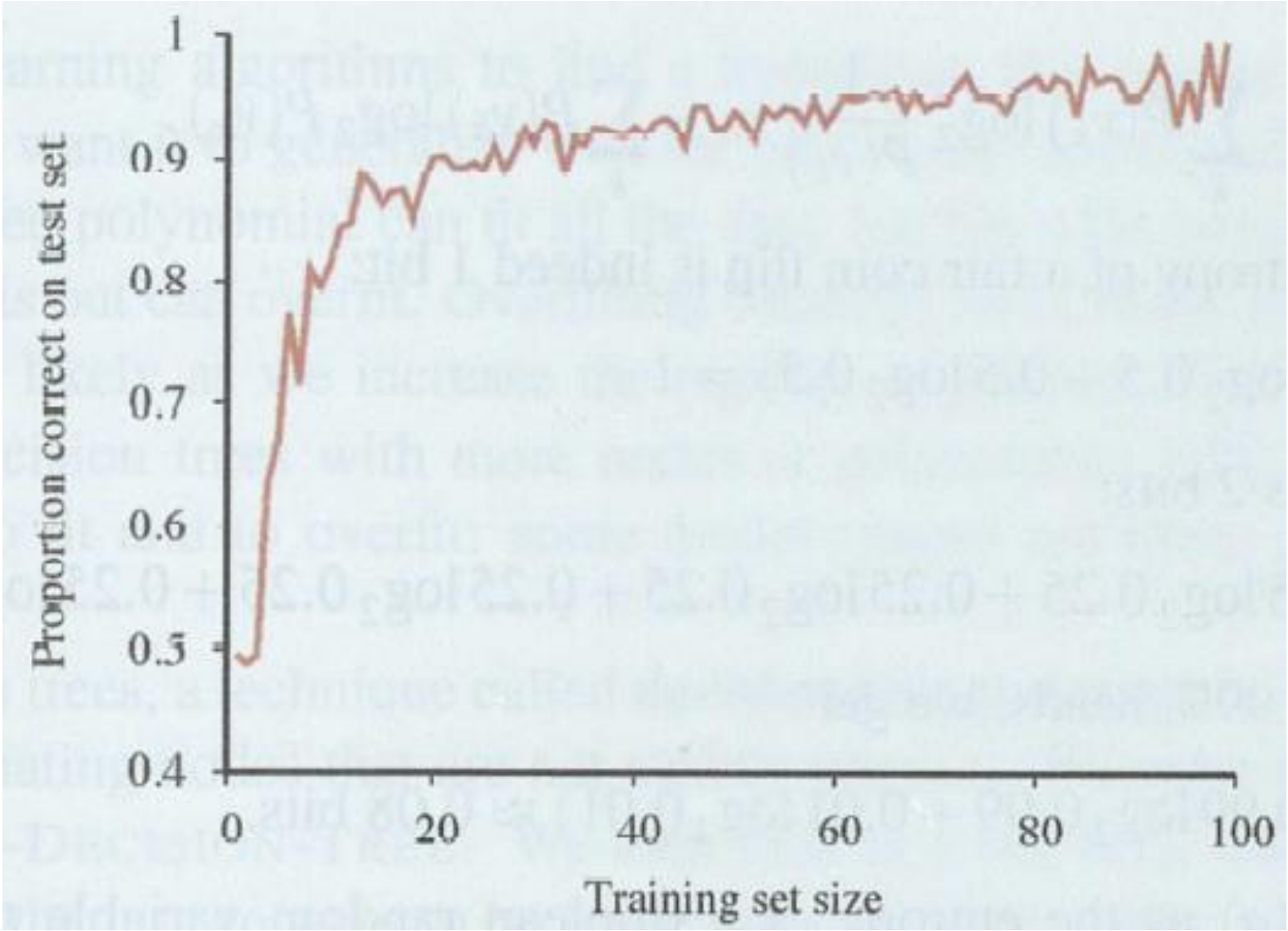
Learned Decision Tree Consistent with the 12 Examples

- 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(FairCoin) = -(½ $\log_2$ ½ + ½ $\log_2$ ½) = 1 bit; H(Die4) = 4 * (-(¼ $\log_2$ ¼)) = 2 bits
  - H(Coin99) = -(0.99 $\log_2$ 0.99 + 0.01 $\log_2$ 0.01) ≈ 0.08 bits;
  - H(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*
x_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
x_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
x_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
x_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
x_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
x_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
x_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
x_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
x_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
x_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
x_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
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*
x_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
x_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
x_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
x_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
x_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
x_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
x_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
x_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
x_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
x_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
x_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
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*
x_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
x_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
x_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
x_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
x_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
x_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
x_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
x_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
x_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
x_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
x_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
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 hyposthesis 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)
```

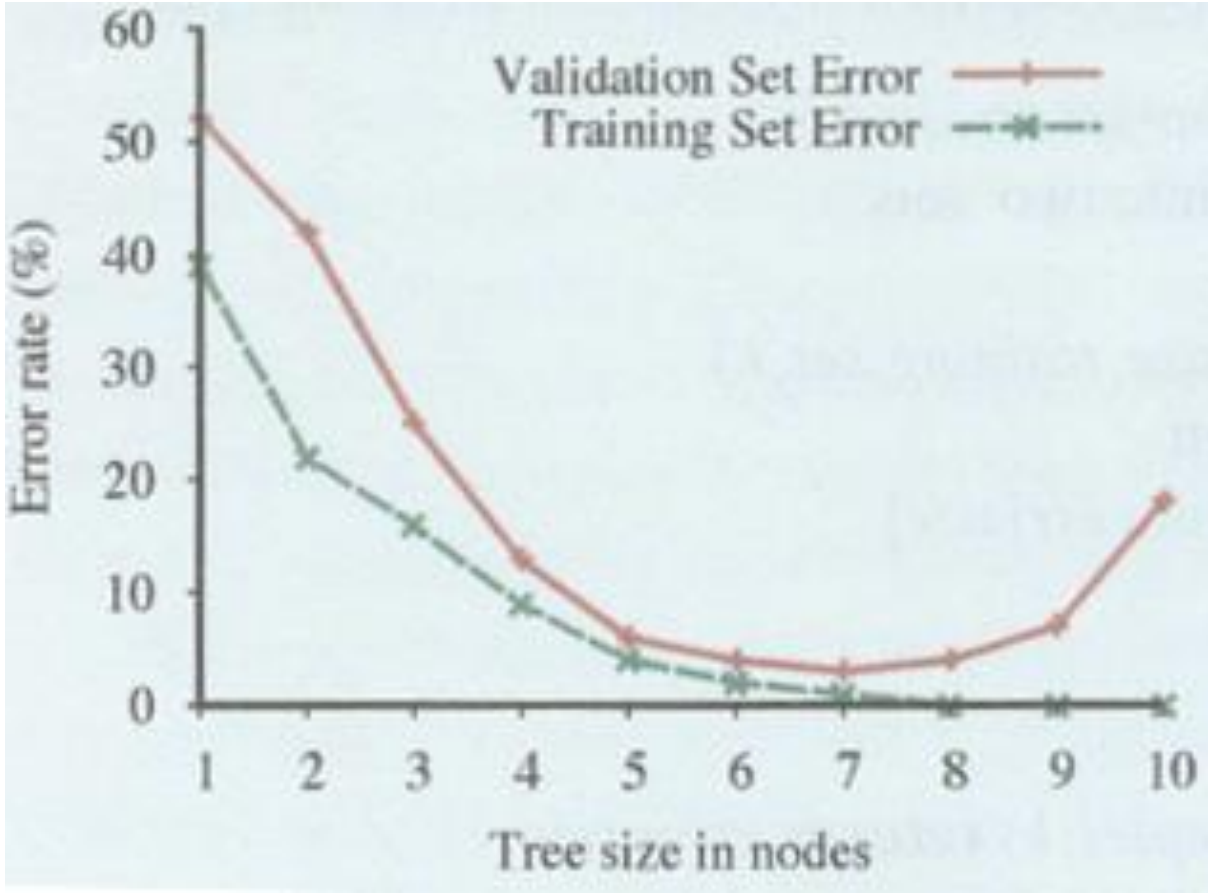
Algorithm for Cross Validation

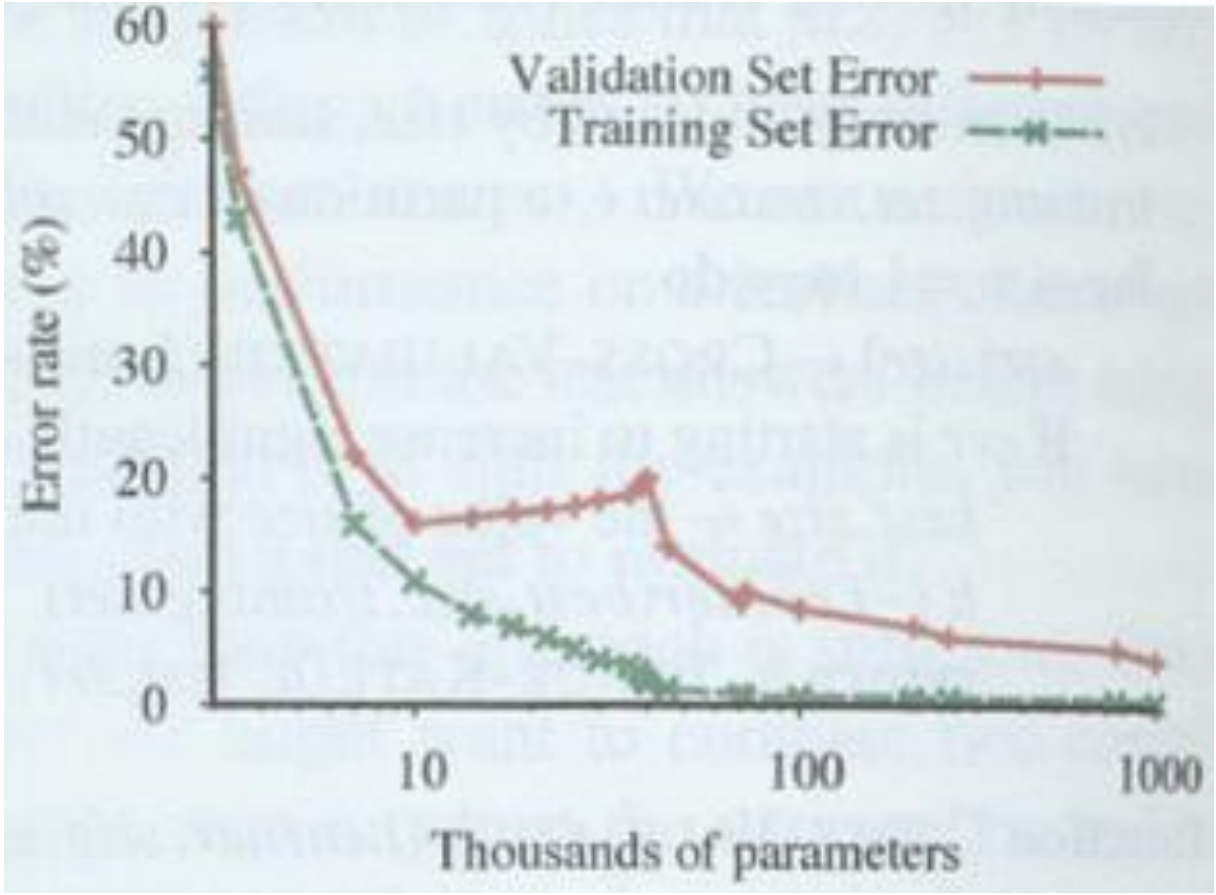
- Each iteration of the for-loop selects a different slice of the examples as validation set

```
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:** $L_1(y, \hat{y}) = |y - \hat{y}|$
  - **Squared-error loss:** $L_2(y, \hat{y}) = (y - \hat{y})^2$
  - **0/1 loss:** $L_{0/1}(y, \hat{y}) = 0$ 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:

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

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 arbitrary 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)

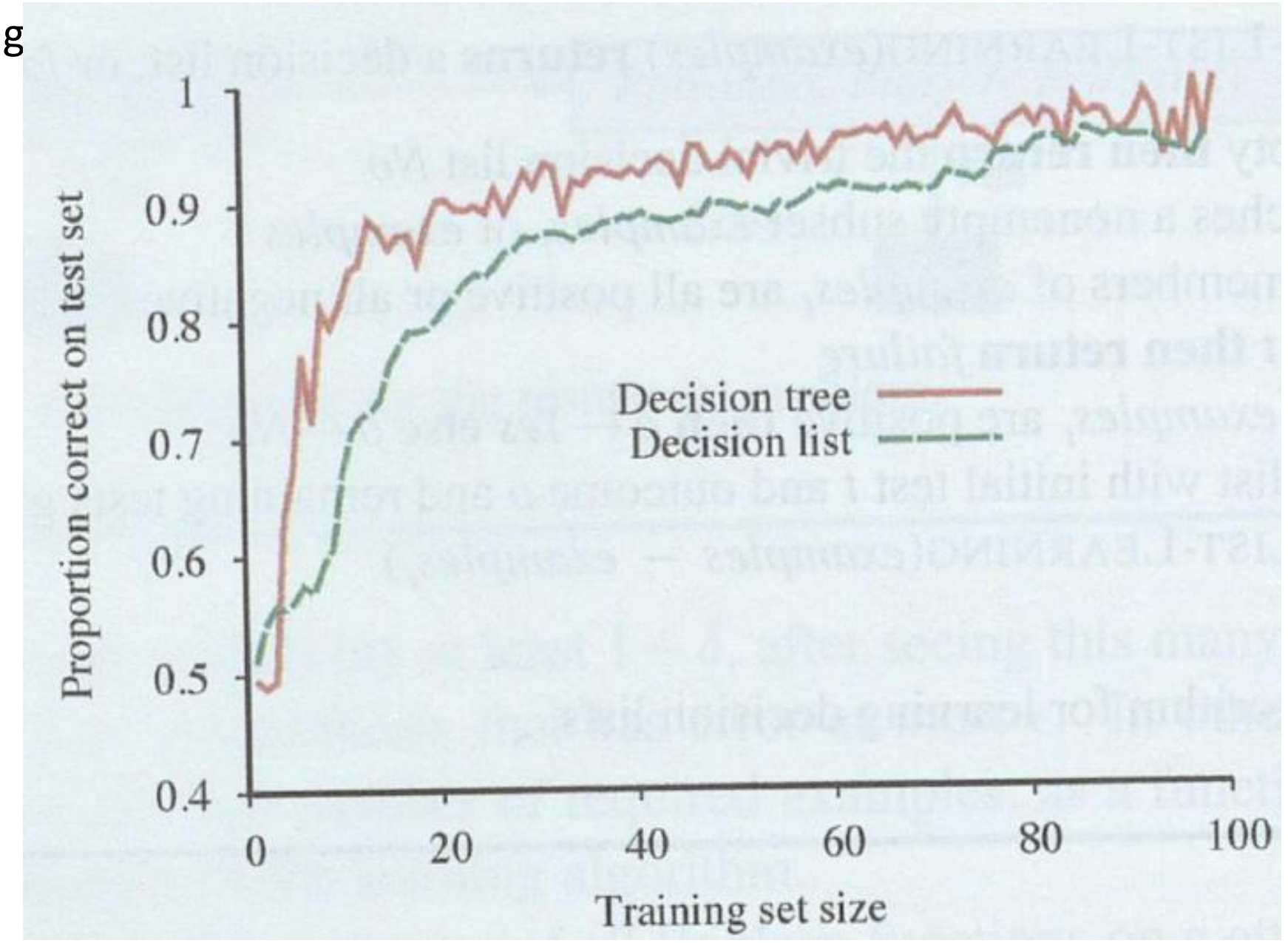
Straightforward Algorithm for Learning Decision Lists

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

if *examples* is empty **then return** the trivial decision list *No*
$t \leftarrow$ 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** $o \leftarrow Yes$ **else** $o \leftarrow No$
return a decision list with initial test t and outcome o and remaining tests given by
DECISION-LIST-LEARNING(*examples* $-$ $examples_t$)

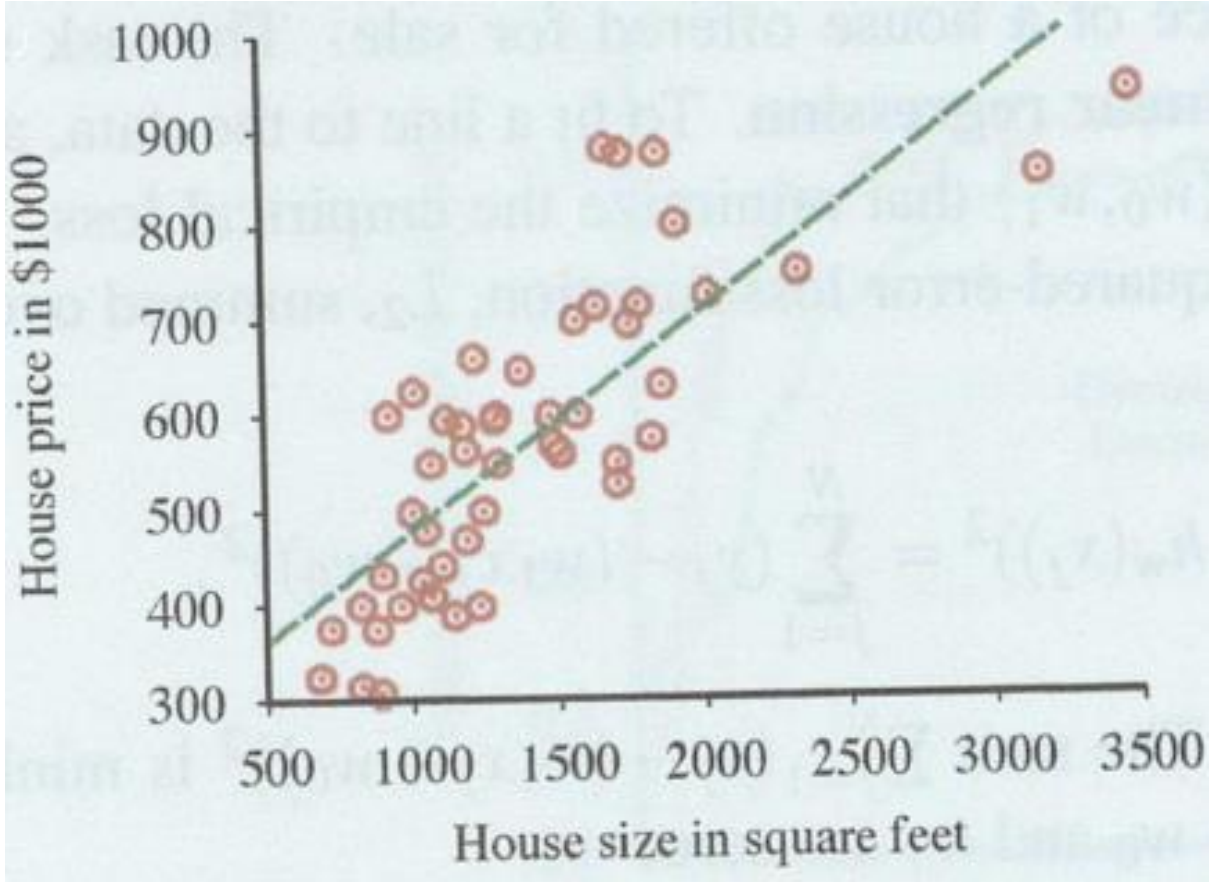
Comparison of Learning Curves

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



Linear Regression and Classification

- Task
  - Given: Data sets with numerical variables (s. figure)
  - Sought: Formula for predicition of one variable given the others
- Univariate linear regression (straight line)
  - Computed by equations with exact solution
  - Computed by gradient descent
- Generalisation to multivariate regression
- Linear classification
  - With threshold function
  - With logistic regression



Learning Straight lines (1)

- Regression task: Sought is vector $w = [w_0, w_1]$ wih $h_w(x) = w_1x + w_0$
  - With minimizing the empirical loss on the test data (e.g. house size and price)
- Empirical loss:
  - Usually squared-error loss L_2, summed over all training examples, i.e.
    - Difference between predicted value $(x_j, h_w(x_j))$ and observed value (x_j, y_j) is squared and added:

$$Loss(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

Learning Straight lines (2)

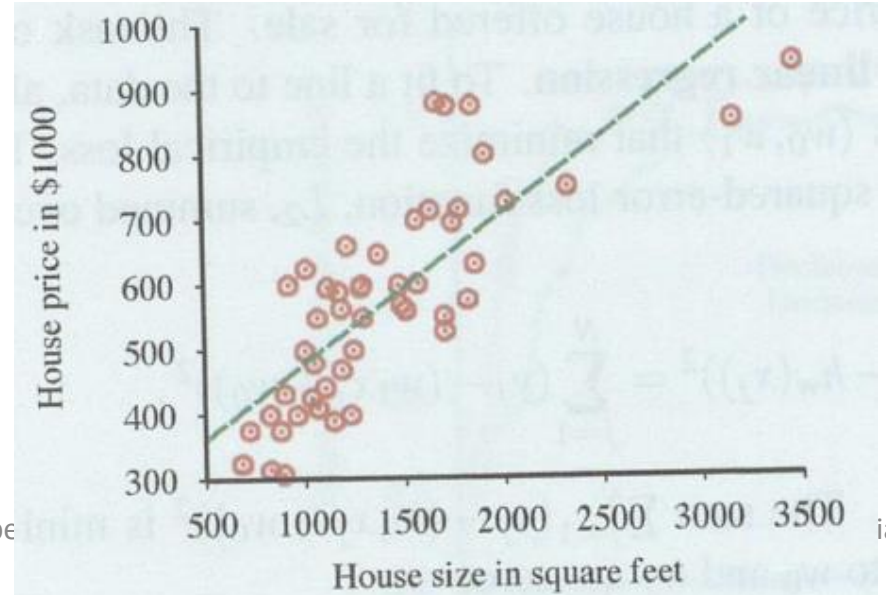
- The sum $\Sigma_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$ is minimal when its partial derivatives with respect to w_0 und w_1 are zero:

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2=0 \text{ and } \frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2=0$$

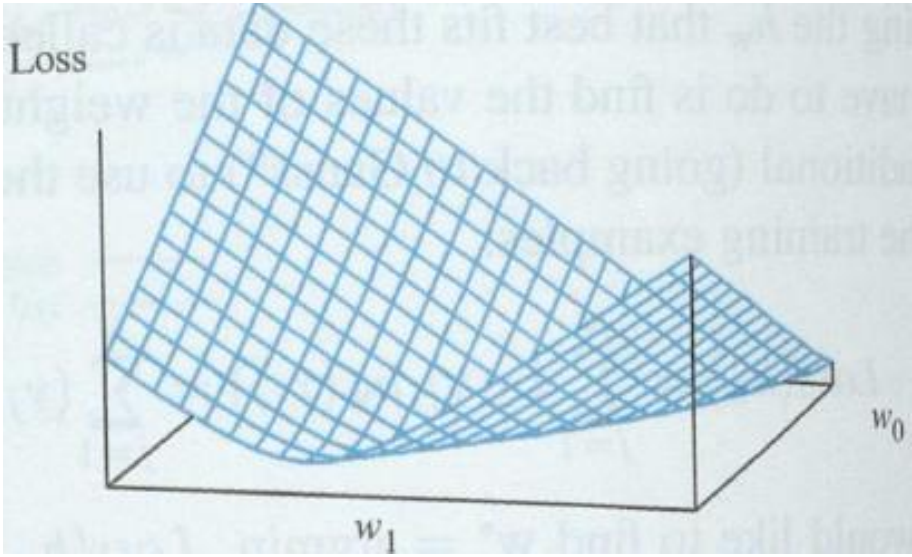
- Unique Solution:

$$w_1=\frac{N(\sum x_j y_j)-(\sum x_j)(\sum y_j)}{N(\sum x_j^2)-(\sum x_j)^2};\quad w_0=(\sum y_j - w_1(\sum x_j))/N$$

w_1= 0,232
w_0= 246



Weight space is convex yielding a unique solution:



Gradient Descent (1)

- General alternate to solving partial derivatives which can be applied to any function no matter how complex:
- Gradient descent (variant of Hill-Climbing)
  - Choose an arbritrary starting point in weight space – here (w_0, w_1)
  - Compute estimate of the gradient and move a small amount in deepest downhill direction
  - Repeat until convergence

Gradient Descent Algorithm

w ← any point in the parameter space

While not converged

for each w_i in w do $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$

(α: Learning rate (step size))

Computing the partial derivatives for w_0 and w_1 :

Chain rule $\partial\, g(f(x)/\partial x = g'(f(x)\partial f(x)/\partial x$

$$\frac{\partial}{\partial w_i} Loss(\mathbf{w}) = \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))^2 = 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(x))$$

$$= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i}(y - (w_1 x + w_0)) .$$

Applying this to w_0 and w_1 yields:

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)); \qquad \frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

with update and learning rate α : $w_0 \leftarrow w_0 + \alpha\,(y - hw(x))$; $w_1 \leftarrow w_1 + \alpha\,(y - hw(x))\,x$

Batch Gradient Descent

- An update is also possible for N training examples instead of just one training example:
  - Minimize the sum of individual losses:

$$w_0 \leftarrow w_0 + \alpha \sum_j \left(y_j - h_{\mathbf{w}}(x_j) \right); \qquad w_1 \leftarrow w_1 + \alpha \sum_j \left(y_j - h_{\mathbf{w}}(x_j) \right) \times x_j.$$

  - „Batch gradient descent learning rule“ for deterministic gradient descent
    - Rather slow
- Faster variant: **Stochastic gradient descent (SGD)**
  - Select randomly a small number of training examples „minibatch“
  - Update weights w_0 and w_1 for the sum analogous to formula for one training example
  - Repeat procedure till convergence
    - Problem: Convergence of minibatch SGD not guranteed, may oscillate
      - Improvement: Schedule of decreasing learning rate α

Multivariate Linear Regression (1)

- Extension, if each example x_j is an element vector. Hypothesis space is:

$$h_{sw}(\mathbf{x}_j) = w_0 + w_1 x_{j,1} + \ldots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}.$$

- Simplification to dot-product of weight- and input-vector (or equivalently with matrix product), if we take for w_0 a dummy input $x_{j0} = 1$:

$$h_{sw}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^\top \mathbf{x}_j = \sum_i w_i x_{j,i}.$$

- Goal: Best vector of weights w* minimizing squared-error loss over the examples:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j).$$

- Iterative Gradient descent (as before) will reach unique minimum of loss function:

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \times x_{j,i}$$

- Analytical solution with matrices: X matrix of input (one example per row) and y vector of outputs for training examples:

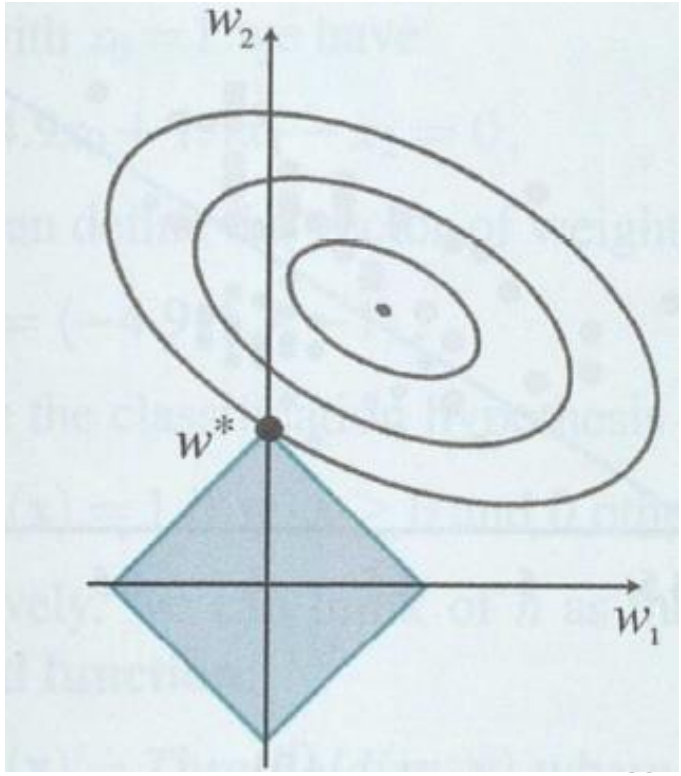
$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Multivariate Linear Regression (2)

- With multivariate regression overfitting is possible, i.e. weights may be learned for dimensions that are actually irrelevant.
- Solution: Regularization (h = hypothesis): Cost (h) = EmpLoss(h) + λ Complexity (h)
- Complexity of a linear function can be specified as a function of weights:

$$Complexity(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q$$

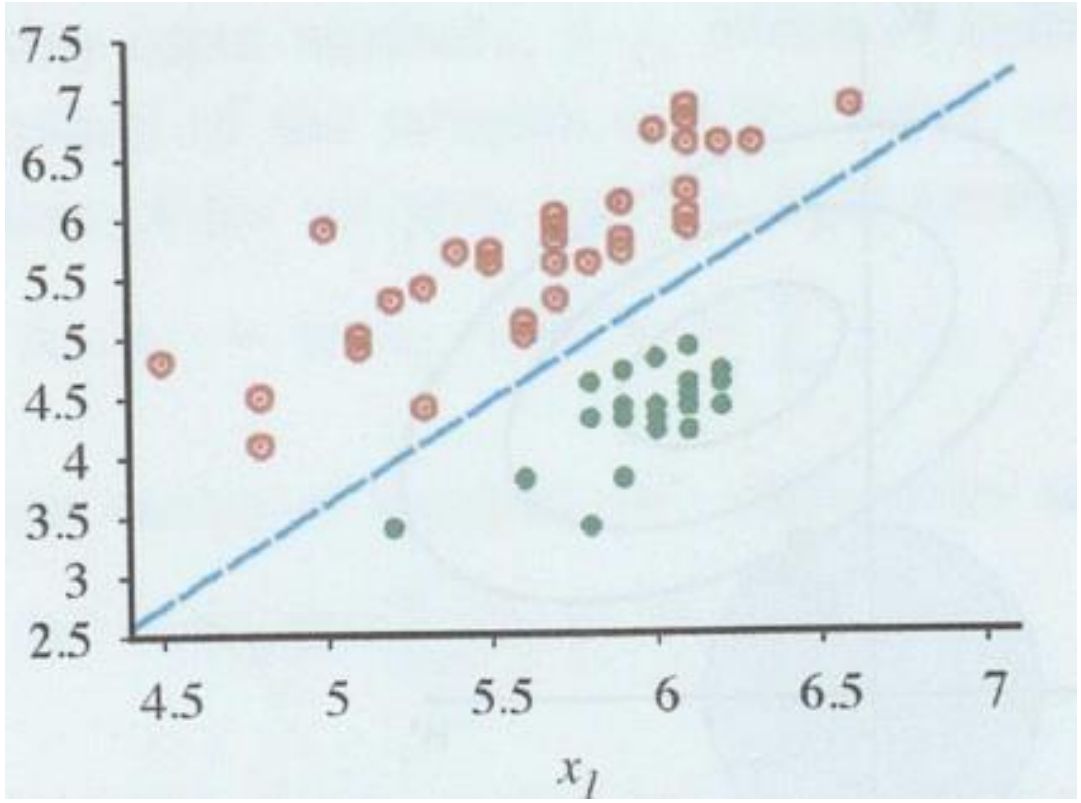
- Loss function L_q: q=1 minimizes the sum of absolute values, q=2 sum of squares
- L_1 often sets many weights to zero and is therefore preferred to L_2
- Visualization (right):
  - Concentric circles: Empirical loss (optimum is center)
  - Diamond: Complexity based on sum of weights (L_1)
  - Intersection: Solution w* with weight $w_1 = 0$



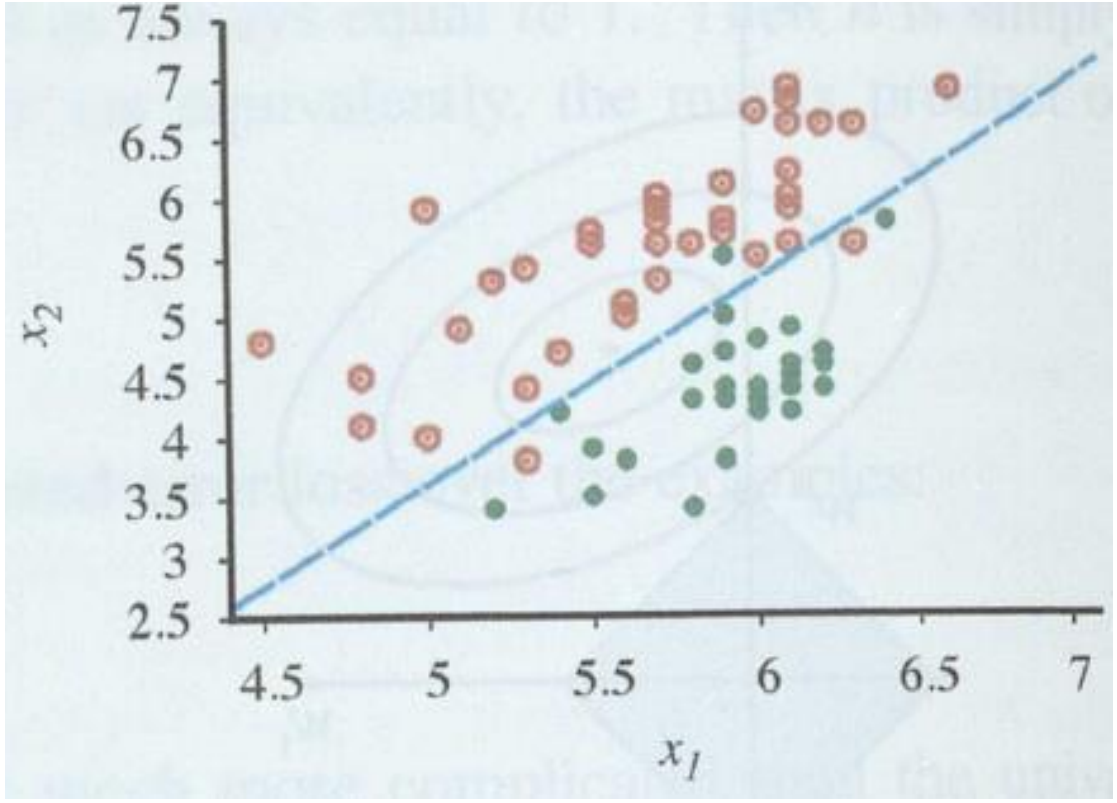
Linear Classifiers with a Hard Threshold

- Linear functions are suitable for classification as well as regression
  - Decision boundary (line or surface in higher dimensions) for separating two classes
- Example: Data for earthquakes and underground explosions with two input values x1 and x2 for body and surface wave magnitude with decision boundary $-4.9 + 1.7\ x_1 - x_2 = 0$

Simpified



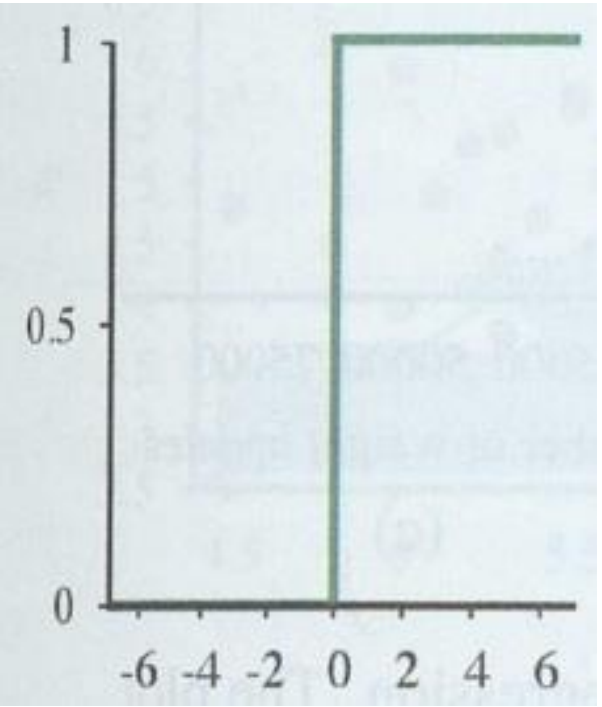
More realistic with more data



Frank

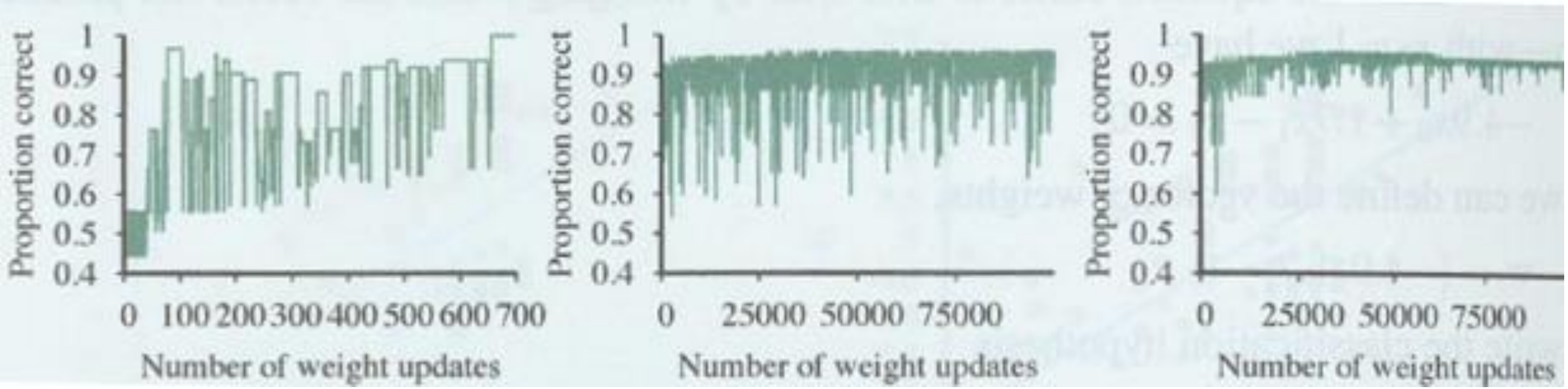
Threshold Function

- Explosions: $-4{,}9 + 1{,}7x_1 - x_2 < 0$
- Earthquakes: $-4{,}9 + 1{,}7x_1 - x_2 > 0$
- Alternate formulation with vector of weights $w = \langle -4.9, 1.7, -1 \rangle$ and $x = \langle 1, x_1, x_2 \rangle$
  - $h_w(\mathbf{x}) = 1$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$, else 0
- Alternate formulation passing the linear function $\mathbf{w} \cdot \mathbf{x}$ through a threshold function:
  - $h_w(\mathbf{x}) = \text{Threshold}(\mathbf{w} \cdot \mathbf{x})$ where Threshold $(z) = 1$ if $z \geq 0$, else 0
- Computing the threshold with gradient descent impossible
- But simple update rule availabe (perceptron rule) for a single example ($\mathbf{x}$, y)
  - $w_i \leftarrow w_i + \alpha\,(y - h_w(\mathbf{x})) \times x_i$
    - If output correct (i.e. $y = h_w(\mathbf{x})$) weights are not changed
    - If $y = 1$ and $h_w(\mathbf{x}) = 0$, w_i is increased proportional to x_i
    - If $y = 0$ and $h_w(\mathbf{x}) = 1$, w_i is decreased proportional to x_i



Convergence of Perceptron Rule

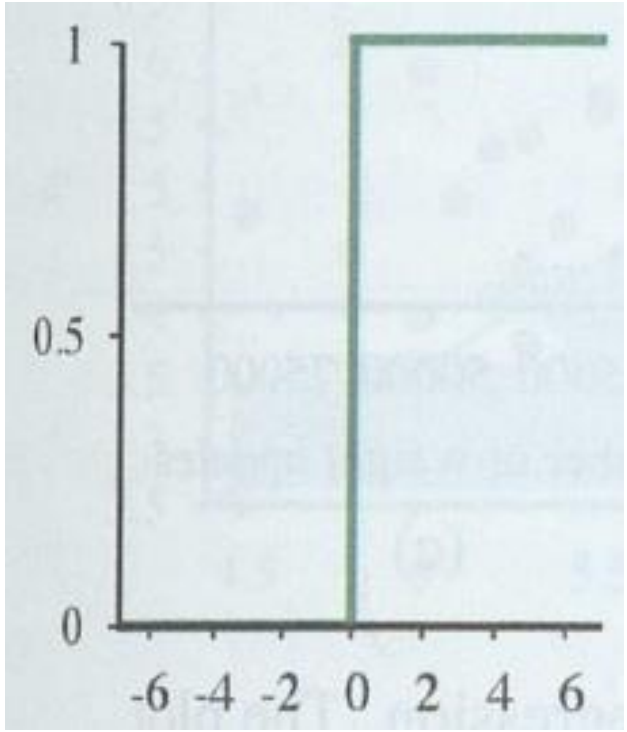
- Perceptron converges to perfect linear separator, when the data points are linearly separable
- However, it does not converge very well with noisy data
  - Can be improved with a learning rate schedule, e.g. $\alpha(t) = 1000/(1000 + t)$ (right)
- Example for earthquake data without (left) and with noisy data (middle and right):



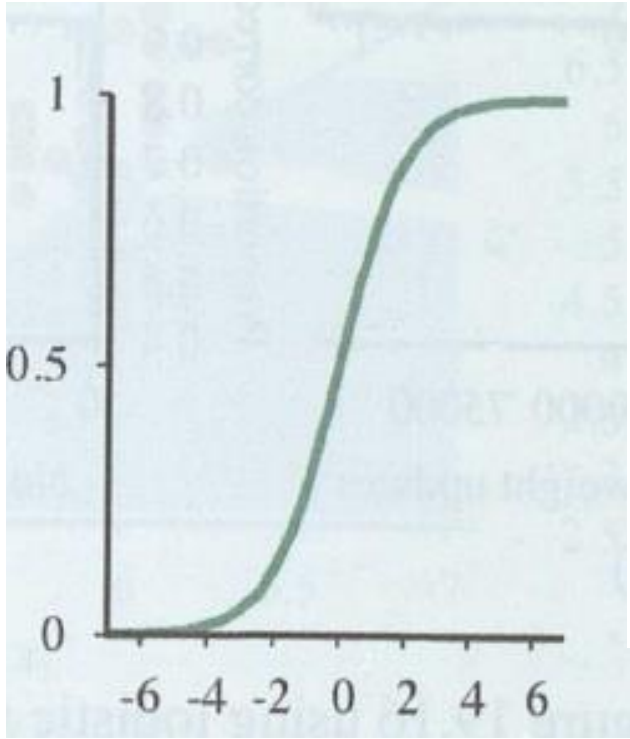
Linear Classification with Logistic Regression

- Convergence is much more effective with a differentiable function
- Candidate: logistic function: Logistic (z) = $\frac{1}{1+e^{-z}}$
- Applied to multivariate linear data: hw(**x**) = Logistic (**w** · **x**) = $\frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}$

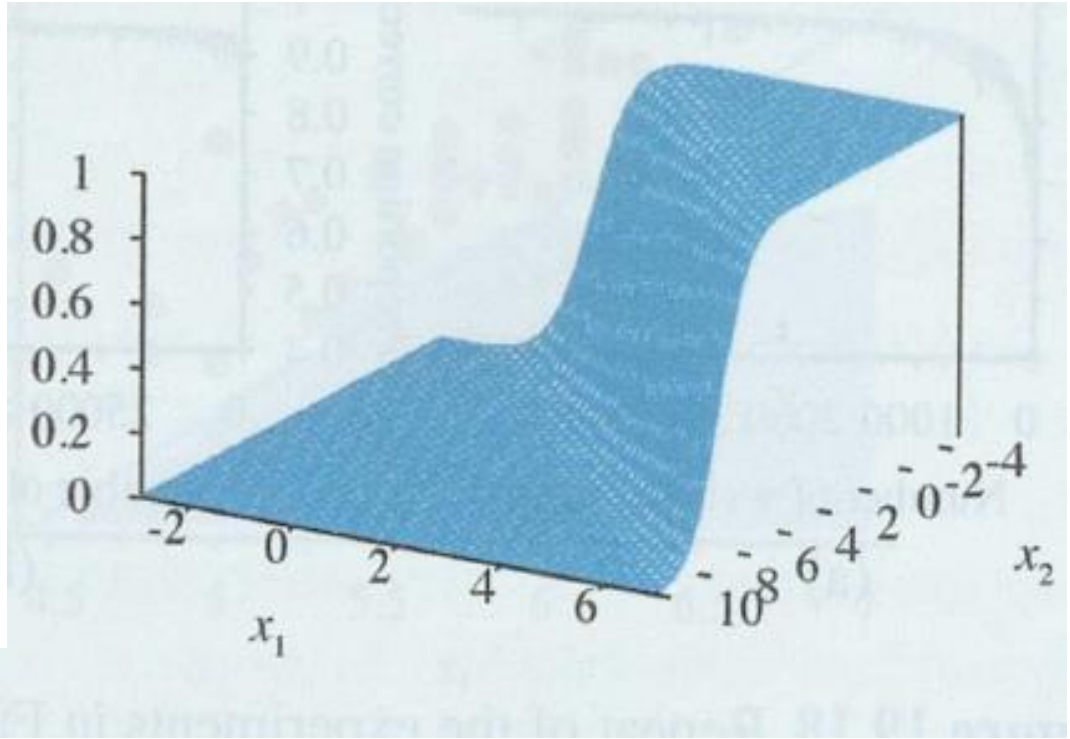
hard threshold (perceptron)



logistic threshold



plot for earthquake data (logistic)



Update of Weights with Logistic Function

Update rule: $w_i \leftarrow w_i + \alpha\,(y - h_w(\mathbf{x})) \times h_w(\mathbf{x})\,(1 - h_w(\mathbf{x})) \times x_i$

Justification:

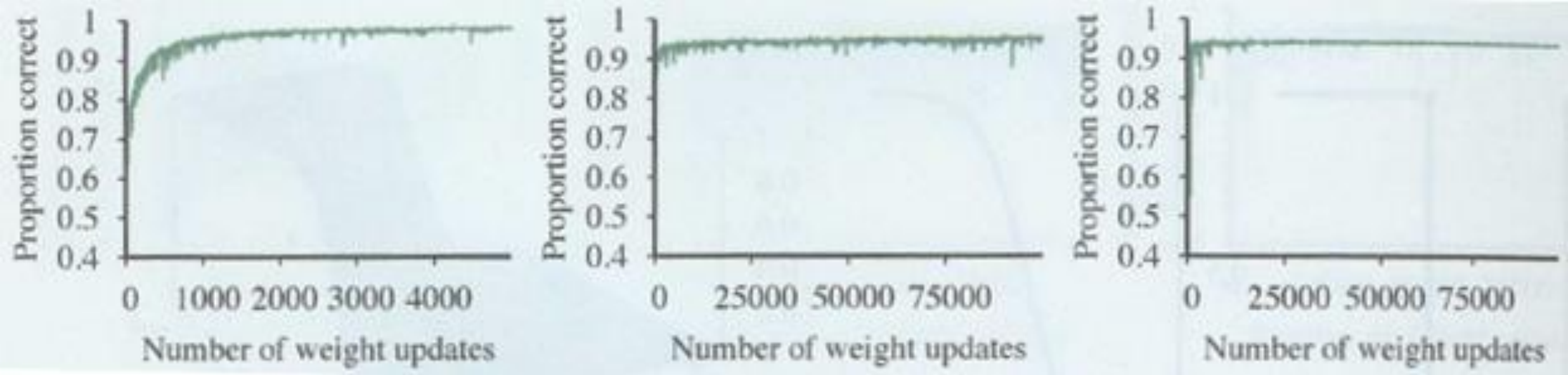
$$\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i}\mathbf{w} \cdot \mathbf{x} \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i\,.
\end{aligned}$$

g = logistic function; g‘ = derivative of g with g´(z) = g(z) (1-g(z))

g‘($\mathbf{w} \cdot \mathbf{x}$) = g($\mathbf{w} \cdot \mathbf{x}$) (1 - g($\mathbf{w} \cdot \mathbf{x}$)) = $h_w(\mathbf{x})$ (1 - $h_w(\mathbf{x})$)

Convergence of Logistic Regression

- Example for earthquake data without (left) and with noisy data (middle and right):
  - Right: With learning rate schedule, e.g. $\alpha(t) = 1000/(1000 + t)$
- Convergence on exact data (left) is slower with logistic regression compared to perceptron rule, but much better with noisy data (middle and right)



Nonparametric Models

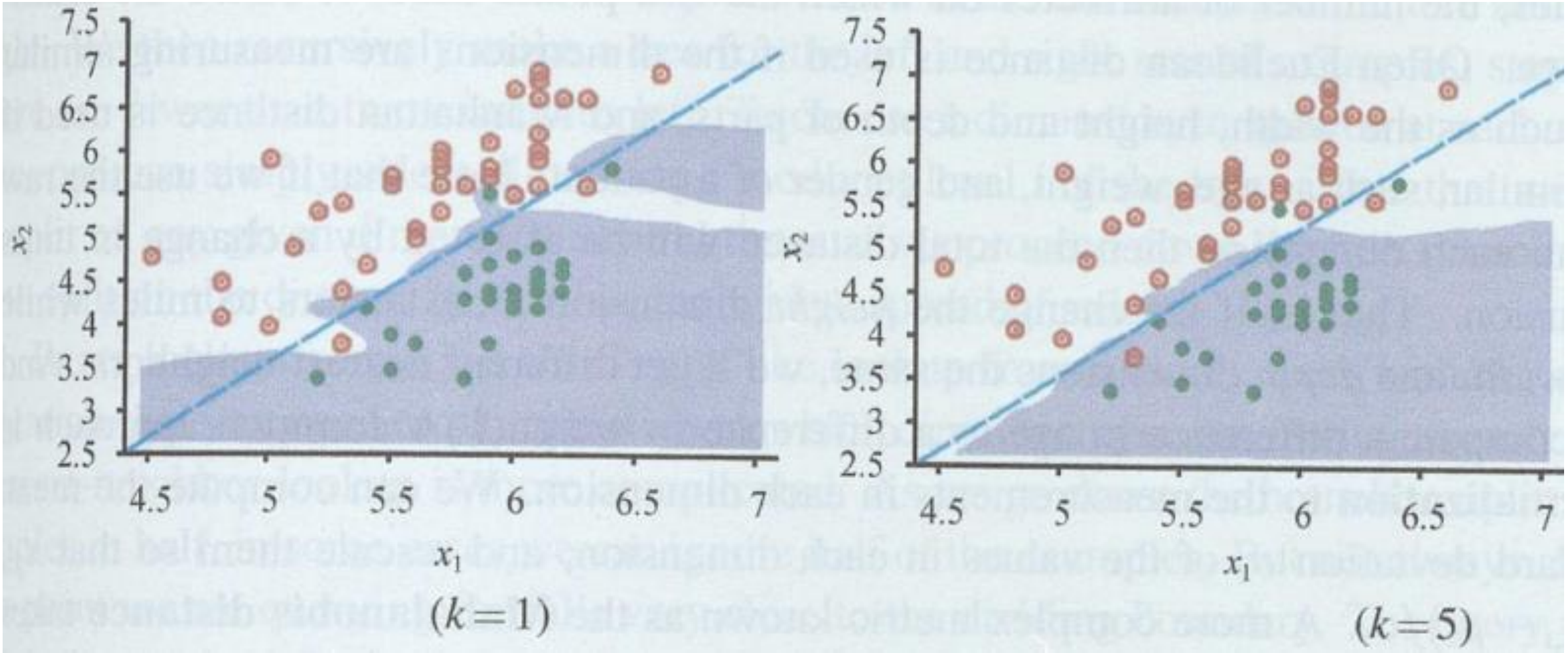
- **Parametric Learning**: All training data is summarized in a few parameters of a model
  - Once a hypothesis is trained, the training data can be throwed away
  - The complexity of the hypothesis is predefined by the model, not by the training data
- **Non-Parametric Learning**: „Let the data speak for itself“
  - Training data is used directly for prediction
  - Complexity of hypothesis grows with training data
  - ➢**Instance-based learning** or **memory based learning**
- Simplest instance-based learning method is **table look-up**
- **Better Methods:**
  - Nearest-Neighbor models
  - Non-parametric regression
  - Support vector maschines

Nearest Neighbor Models (Case-based Reasoning)

- Assumption: Similar cases have similar solutions
- Improvement over table look-up: Find similar cases (instead of identical cases)
  - Use k most similar cases instead of the one most similar (**k Nearest Neighbor** or **k-NN**)
- Problem: How to define similiarity (or distance)?
  - Continuous values:
    - **Minkoswski distance** or **L^p norm** defined as: $\mathbf{L^p(x_j,x_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{1/p}}$
      - With p=1: **Manhattan distance:** $\mathbf{\sum_i |x_{j,i} - x_{q,i}|}$
      - With p=2: **Euclidean distance:** $\mathbf{\sqrt{(\sum_i (x_{j,i} - x_{q,i})^2)}}$
    - To compare attributes with different scales: **Normalization**
      - Compute mean μ_i and standard deviation σ_i and rescale $x_{i,j}$ to $\mathbf{(x_{i,j} - \mu_i)/\sigma_i}$
  - Discrete values:
    - **Hamming-distance**: Percentage of different attributes $\mathbf{\sum_i (x_{j,i} \neq x_{q,i}) / \sum_i}$
    - Weighted (w_i) Hamming distance with partial similarities (f_i): $\mathbf{\sum_i w_i f_i(x_{j,i}, x_{q,i}) / \sum_i}$

Application of k-NN to Earthquake Data Set

- With k=1 there is some overfitting, which is avoided with k=5
- However, with few data points, there are still problems (upper right region)



Problems of Nearest Neighbor Models (k-NN)

- **Curse of dimensionality**:
  - With few attributes (dimensions) and many cases, k-NN works very well
  - With many dimensions it is rather unprobable to find similar cases
- Run-time efficiency
  - The retrieval time of naive k-NN is proportianal to number of cases
  - Improvements:
    - Hierarchical search with **k-d trees**
    - **Locality-sensitive hashing**

k-d Trees

- **k-d tree:** Balanced binary tree over data with an arbritary number of dimensions
- Recursive **construction** of k-d tree:
  - Choose a dimension d_i (attribute) randomly or by the widest spread of values
  - Split examples taking the median of dimension d_i in two branches
- **Look-up** with k-d trees
  - Standard look-up in binary trees finding one similar example
  - However, for each dimension it is necessary to check, whether the k nearest values are in the choosen branch. If not, the other branch must be included in the search.

➢K-d trees are appropriate when there are preferably at least 2^n examples for n dimensions

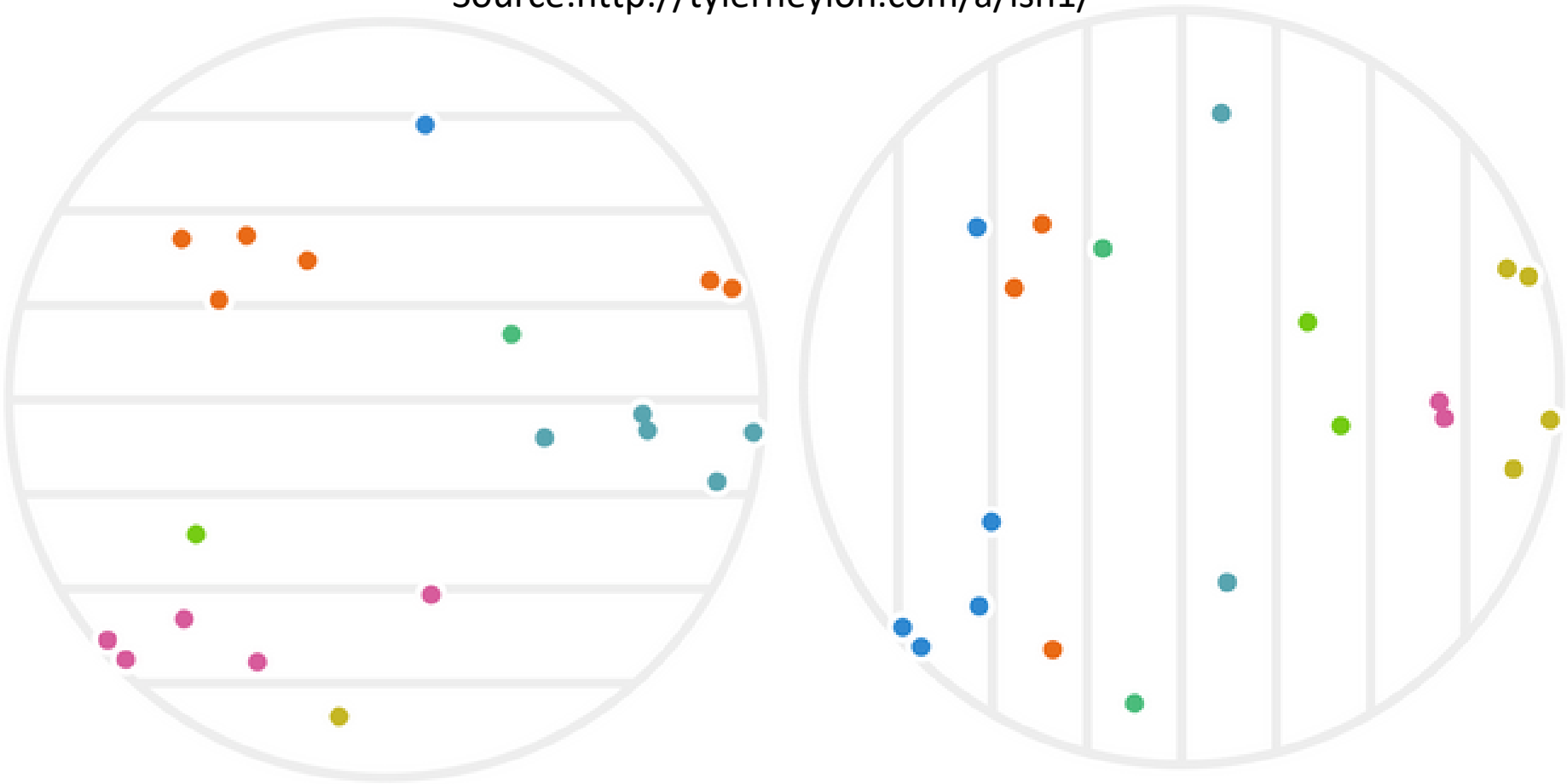
Locality-sensitive Hashing (1)

- **Hash codes** maps data or arbritary size into fixed size values (bins) allowing a fast look-up
  - Different data should be mapped to different bins
- **Locality-sensitve hashing (LSH)** tries to map similar data into the same bin
- Idea:
  - If points are close together in an n-dimensional space, they are close together in most sub-spaces (e.g. in each single dimension or a combination of some dimensions)
  - Hash all examples in many subspaces defined by random projections (functions)
    - Requires parameters about number of subspaces and how projections are generated
  - For a new case, collect all examples from all identical bins from all projections, and check how similar they are to the new case
    - Huge efficiency gain, since most dissimilar examples appear in none identical bin
    - One might require, that candidates for similar examples might appear in at least two or more identical bins, depending on the parameters

Locality-sensitive Hashing (2)

Source:http://tylerneylon.com/a/lsh1/

- Two simple projections on two-dimensial data
- Equal colors: Examples are mapped in the same bin of the projection (same hash code)

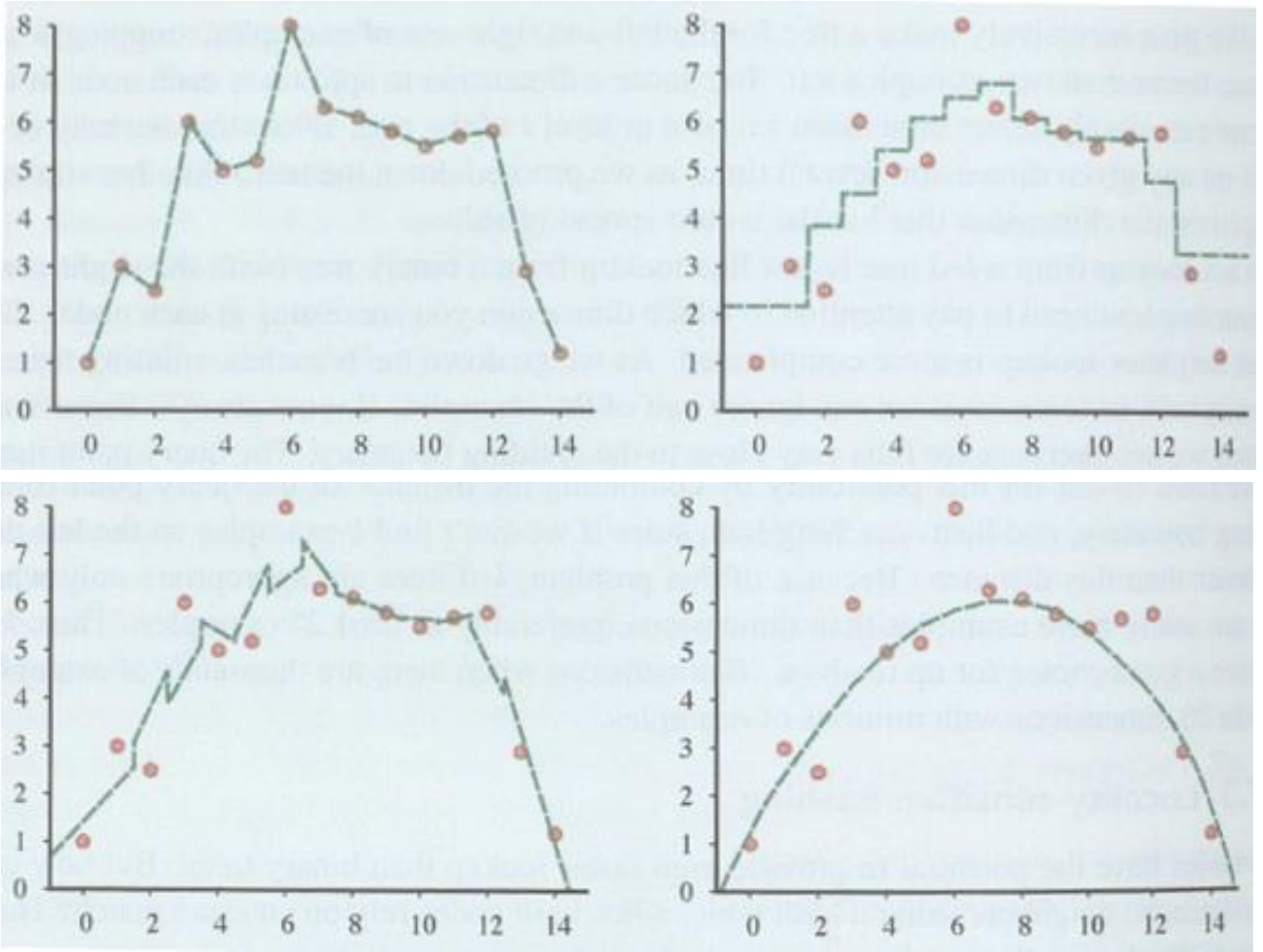


- For a large real world problems finding nearest neighbors in a dataset for 13 million images with 512 dimensions, locality-sensitive hashing had to examine only a few thousand images

Nonparametric Regression

Examples for nonparametric regression models:

- Left upper: Piecewise-linear nonparametric regression (connect-the-dots)
- Right upper: 3-nearest-neighbor average
- Left below: 3-nearest neighbor linear regression
- Right below: Locally weighted regression with a quadratic kernel



K-Nearest Neighbor Regression

- **Piecewise-linear nonparametric regression** (connect-the-dots): For a new query point x, simply interpolate the two training examples to the left and to the right
- **K-nearest-neighbor regression**: Instead of taking just one example to the left and to the right, use the k nearest neighbors
  - Take the average, i.e. the mean y-value of k points: $h(x) = \sum y_j / k$
  - Make a regression through k points, i.e. piecewise regression like connect-the-dots, but with several points
- **Locally weighted regression**: Avoid the discontinuities of the above approaches by locally weighting the examples: the nearer examples with a higher weight
  - Use a kernel function K(Distance(x_j, x_q)) which output a value for an x_j depending on the distance to the query point x_q, e.g $K(d) = \max(0, 1 - (2|d|/w)^2)$ with $w = 10$
  - For a query point x_q, the output is $h(\mathbf{x_q}) = \mathbf{w^*} \cdot \mathbf{x_q}$

    with $\mathbf{w^*} = \text{argmin}_w \sum_j K(\text{Distance}(\mathbf{x_j}, \mathbf{x_q}))\ (y_j - \mathbf{w} \cdot \mathbf{xj})^2$

Support Vector Machine (SVM)

- In early 2000s, most popular approach for supervised learning
- Currently: Deep Learning and Random Forests (ensemble of decision trees)
- Basic ideas of SVMs:
  - SVMs creates a linear separating hyperplane in a higher-dimensional space, using a transformation, the so-called kernel-trick
  - SVMs construct a maximum margin separator – a decision boundary with the largest possible distance to example points in order to generalize well
  - SVMs are nonparametric – the separating hyperplane is defined by a set of example points, not by a collection of parameter values. Instead of using all example points like k-NN, SVMs use only few examples close to the separating hyperplane
- Usually has a high computing effort (often higher than Deep Learning)

Searching for a Separating Hyperplane

Problem: Often (in particular with few dimensions), the example points are not linearly separable

Solution: Transform the features in a higher dimensional space by constructing new features

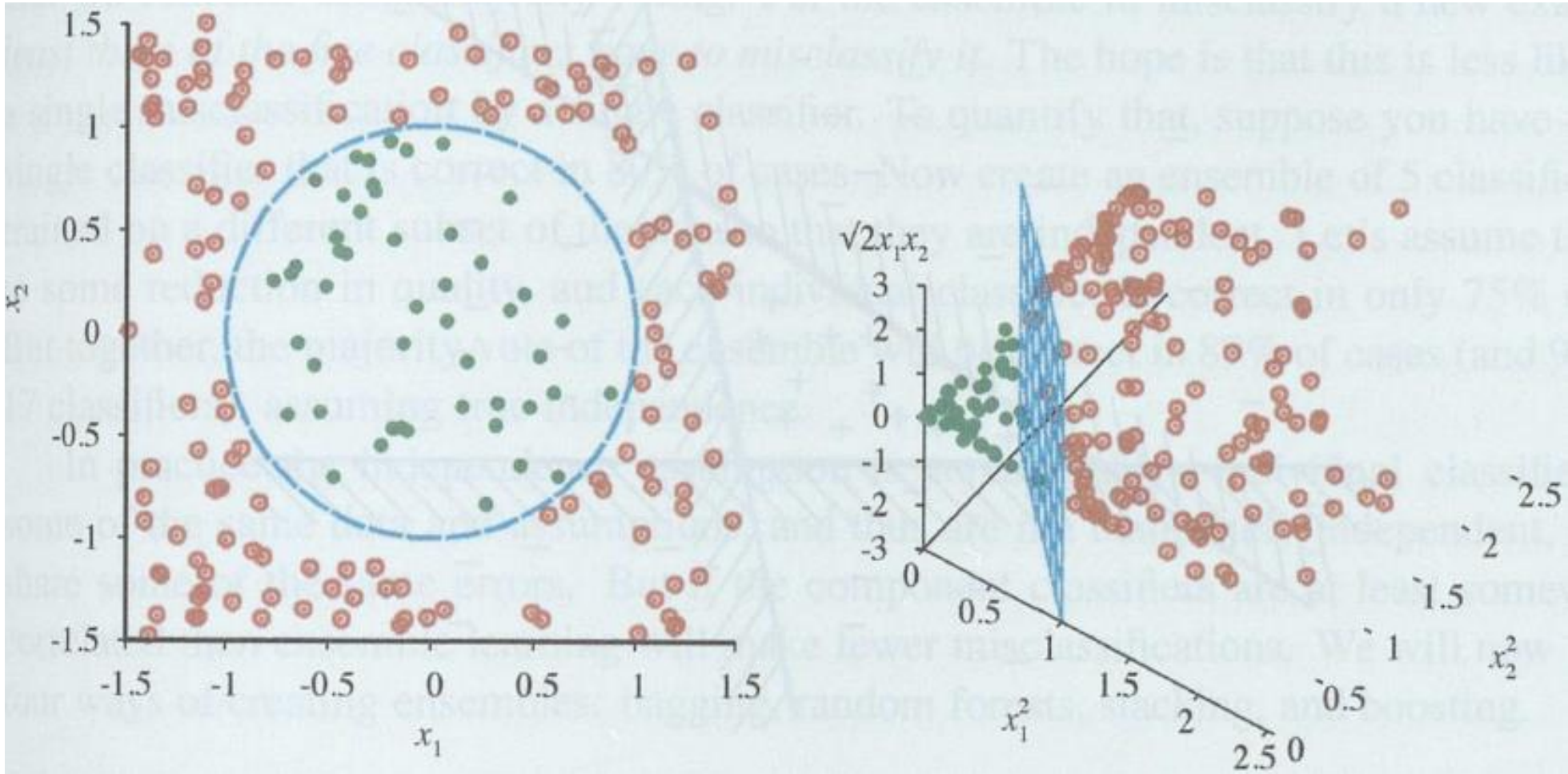
Search problem: Search for suitable transformations according to some patterns (e.g. n-grade polynoms, exponential functions etc.) for a separating hyperplane (with minimal dimensions)

Original features:

$x = (x_1, x_2)$

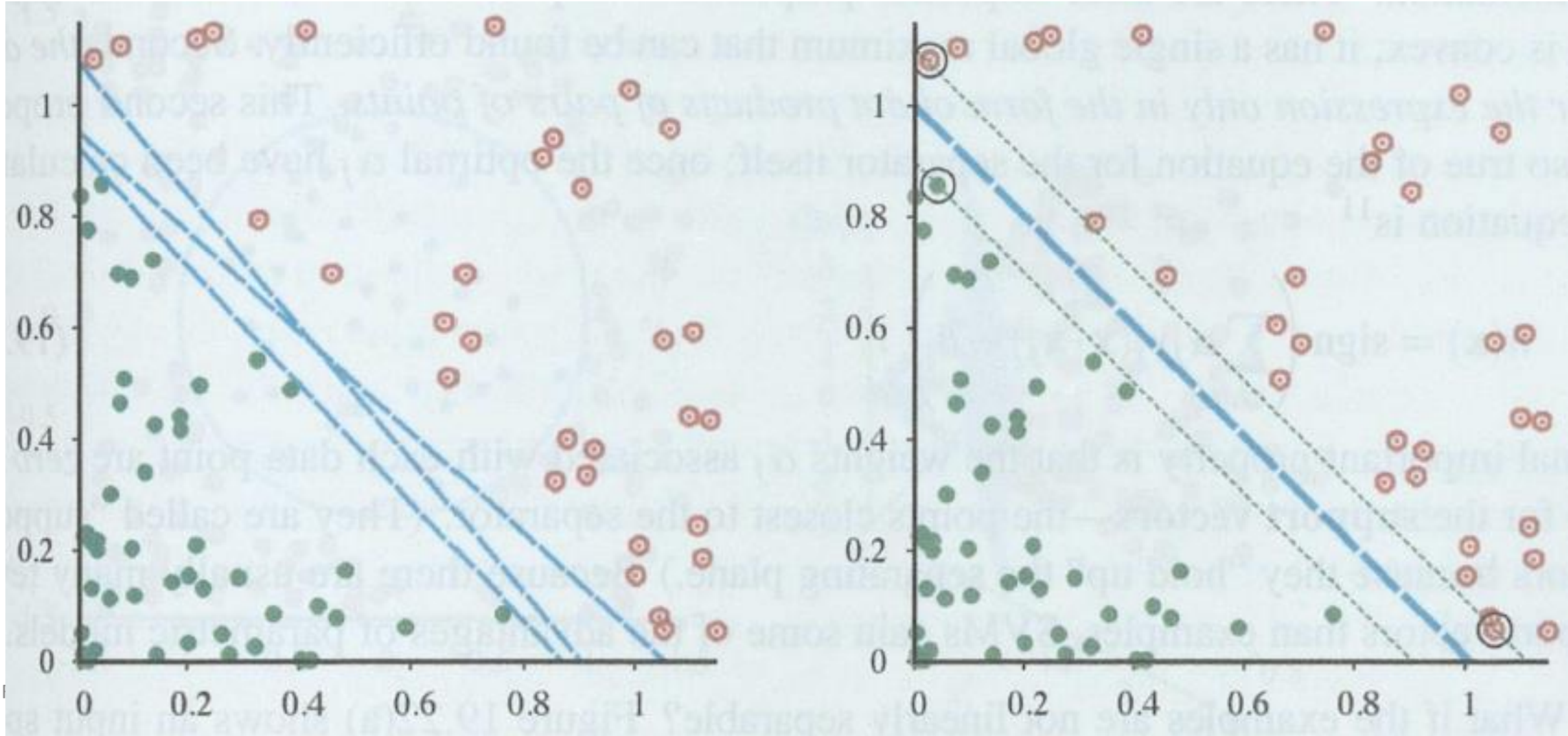
Transformed features:

$F(x)=(x_1^2, x_2^2, \sqrt{2}\, x_1x_2)$



Maximum Margin Separator

- Left: There are several candidate lines for a linear separator (searched by form of gradient descent)
- Right: SVMs choose separator with largest distance to nearest example points (**support vectors**)



SVM Optimization

- Compute parameter α_j, that maximizes the following expression (examples x_j with classification $y_j = \pm 1$):

$$\sum_j \alpha_j - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x_j} \cdot \mathbf{x_k}) \quad \text{with } \alpha_j > 0 \text{ and } \sum_j \alpha_j y_j = 0$$

  - This is a quadratic programming optimiziation problem
  - The expression has a single global maximum
  - The data is used only as dot-products (kernels are applied to dot-products)
  - The weights α_j are mostly zero and only for the support vectors ≠ 0, therefore the effective number of parameters is small (<< N)

- With the optimal α_j we can classify a new example x:

$$h(x) = \text{sign}\left(\sum_j \alpha_j y_j (\mathbf{x} \cdot \mathbf{x_j}) - b\right)$$ //b intersection point on axis (learned like α_j)

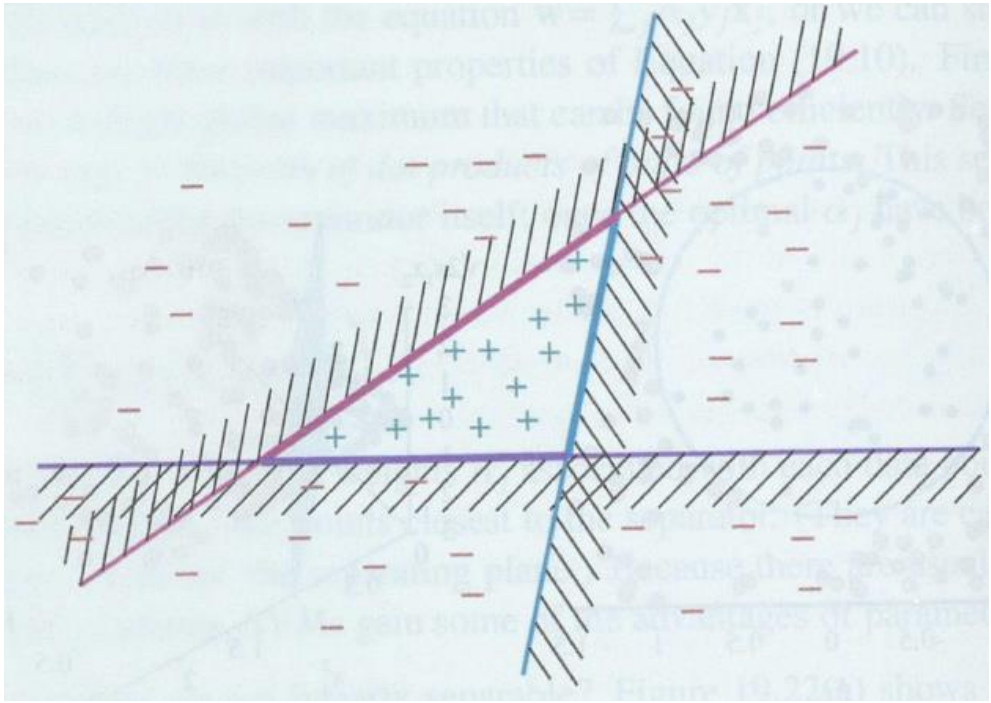
SVMs and Noisy Data

- SVM can be extended for noisy data
  - Trade-off between complexity of hypothesis and error tolerance necessary
  - Use a **soft margin** classifier allowing examples to fall on the wrong side of the decision boundary with a penalty proportional to the distance required to move them back to the correct side
  - Basic algorithm does not change

Ensemble Learning

- Idea: Select a collection or ensemble of hypotheses h_1, h_2, … , h_n and combine their predictions by averaging, voting or even by another level of machine learning
  - Individual hypotheses: **Base models**
  - Their combination: **Ensemble model**

- Advantages
  - **Reduce bias:** An ensemble might be more expressive than the base models, e.g. the ensemble of three linear classifiers can represent a nonlinear, triangular region by voting
  - **Reduce variance:** If the errors of different base classifiers are independant, the ensemble has a much smaller error for statistical reasons



Techniques for Creating Ensembles

- **Bagging:** Generate different base models by training a learning algorithm with different data
- **Random forests:** Generate different decision trees by varying attribute and split point value choices
- **Stacking:** Use multiple base models from different model classes trained on the same data
- **Boosting:** Generate different base models with the same data by increasing the weight of misclassified examples (equivalent to add identical misclassified examples) and decreasing the weight of correctly classified examples

Bagging

- **Bagging:** Generate k distinct training sets by samping with replacement from the original training set and use them for a machine learning algorithm to get K distinct hypotheses.
  - Classification: Take the plurality vote (maybe weighted by confidence)
  - Regression: Take the average: h(x) = $\frac{1}{K}\sum_{i=1}^{K} h_i(x)$

Random Forests

- **Random forests:** Bagging applied to decision trees is more effective, if the decision trees additionally differ to each other, in particular in the top attributes
  - To increase variance, vary attribute choices (rather than training examples) by selecting attributes and afterwards the split point value with a random factor
    - From n attributes choose the best attribute from a random subset of just $\sqrt{n}$ attributes
  - **Hyperparameters** (can be trained as usual by cross-validation)
    - Number of trees K
    - Number of examples used by each tree N (as percentage of complete data set)
    - Number of attributes to choose from at each split point (e.g. $\sqrt{n}$)
    - Number of random split points values tried (optional; trees constructed in this fashion are called **extremely randomized trees (ExtraTrees).**
  - Random forests need no pruning, since they are resistant to overfitting
- Random forests were most popular approach from 2011-2014 in Kaggle data science competitions and remain common (also as popular **baseline** for other learning algorithms)

Stacking

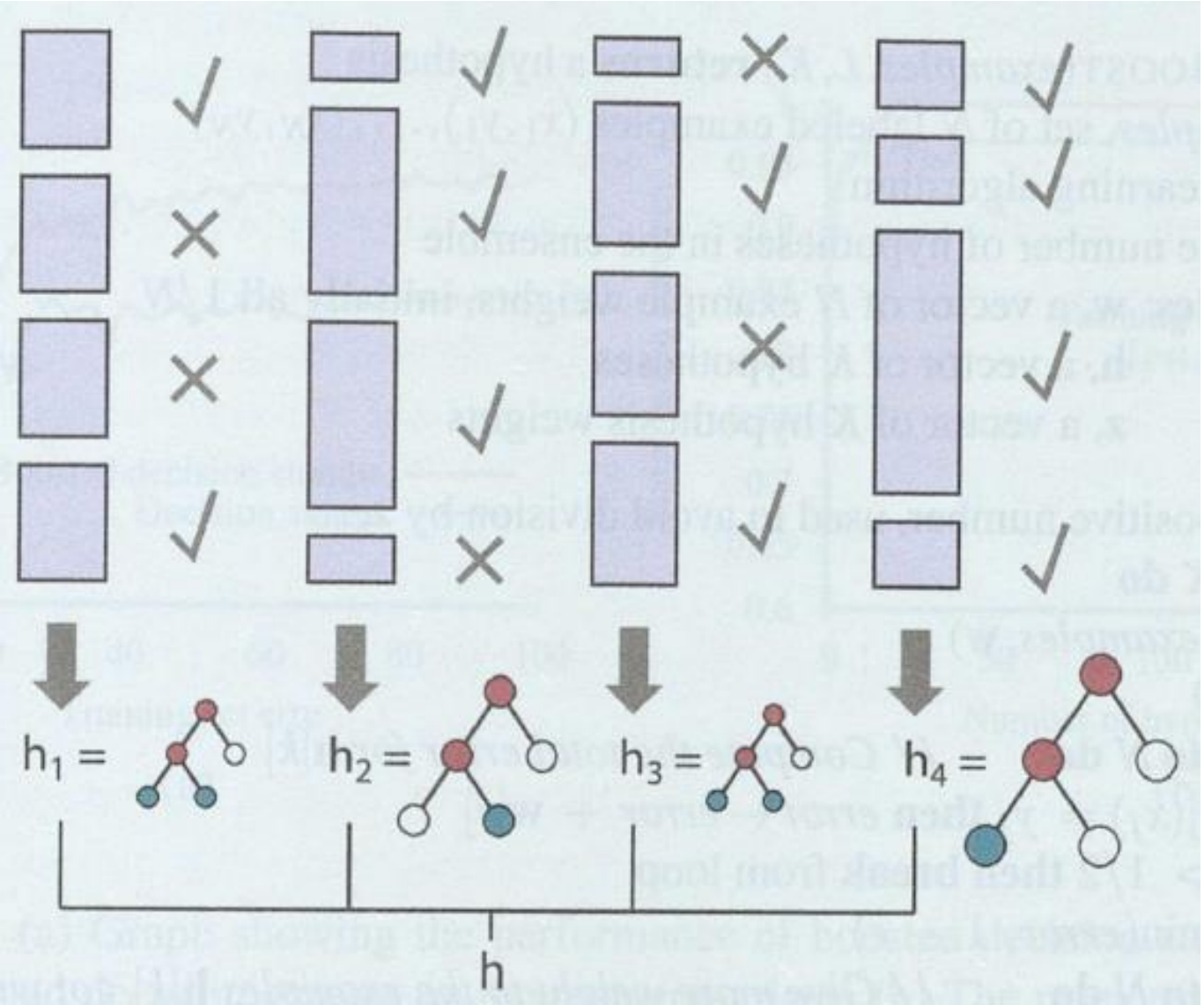
- **Stacking:** Whereas bagging combines multiple models of the same model class trained on different data, stacking combines multiple base models from different model classes (e.g. SVM, logistic regression and decision tree model) trained on the same data
  - Combination strategies:
    - Simple voting
    - Weighted voting, i.e weighted average of the base models, for example that predictions are weighted in a ratio 50%: 30%: 20% (learned as hyperparameter)
    - Learn nonlinear interactions between data and predicitions
    - Stacking on several levels (hierarchies) also possible
  - Frequently used in data science competions like Kaggle or KDD Cup, because indviduals can work independantly and finally combine their learning algorithms

Boosting

- Train different models on the same model class and the same data
- Create differences by a **weighted training set**
  - Each example gets a weight ≥ 0 that describes how much it counts during training
  - A weight of 3 might be interpreted like having 3 example copies in the training set
- Boosting starts with equal weights $w_j = 1$ for all examples and generates a first hypothesis h_1
- In next round, incorrectly classified examples increase their weights while weights of correctly classified examples decrease. From this new weighted training set, we generate hypothesis h_2
  - Examples that are difficult to classify get increasingly larger weights, until they are classified correcty
- The final ensemble lets each hypothesis vote with a weight proportional to its score on its training set

Visualization of Boosting

- Wrongly classified example (marked with an „x“) are enlarged in the next round
- The final ensemble h consists of four hypotheses h_1, .. h_4, which are weighted according to the wrongly classified examples (indicated by the size of the decision tree)
- The algorithm uses a greedy strategy and must be performed sequentially



ADABOOST Algorithm

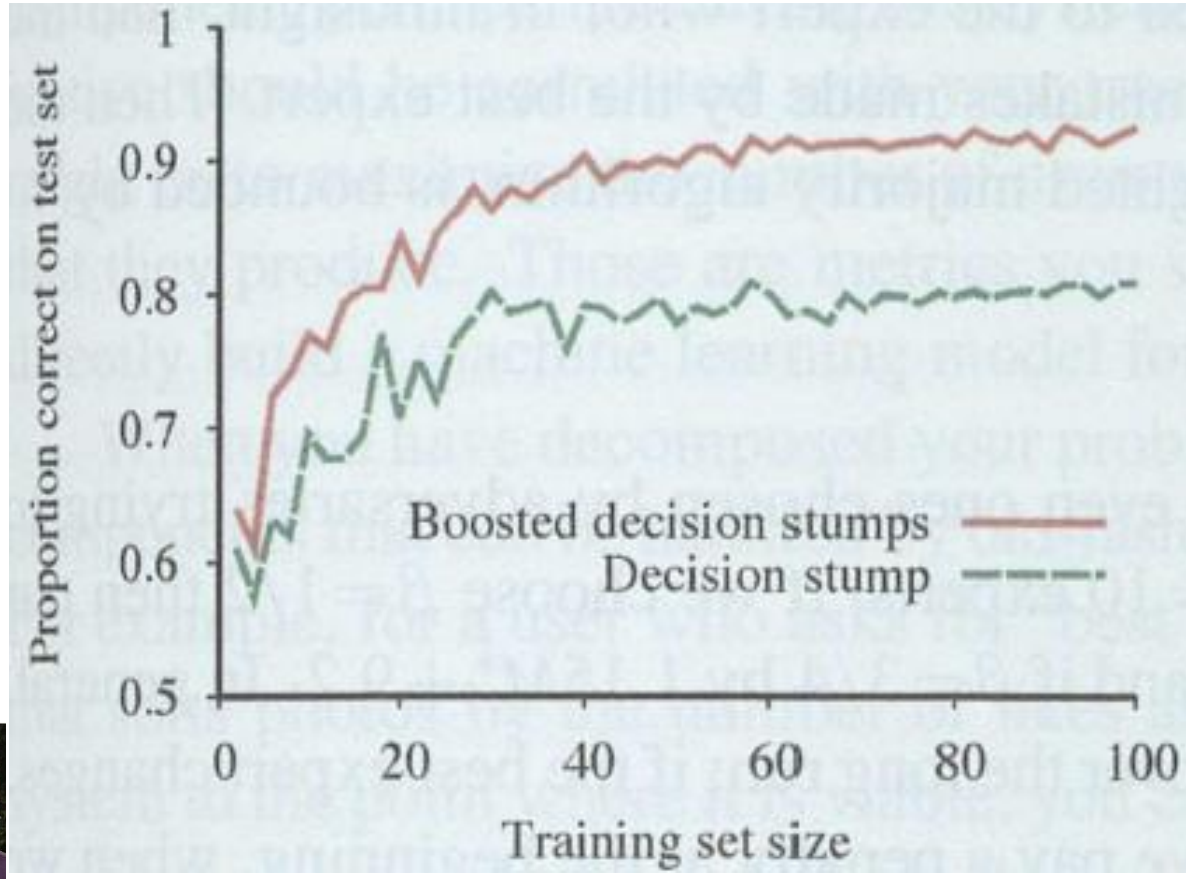
- Can be used with the function (last line) „Weighted-Majoritiy“.
- Continues to generate hypotheses, if they have an error < 0.5

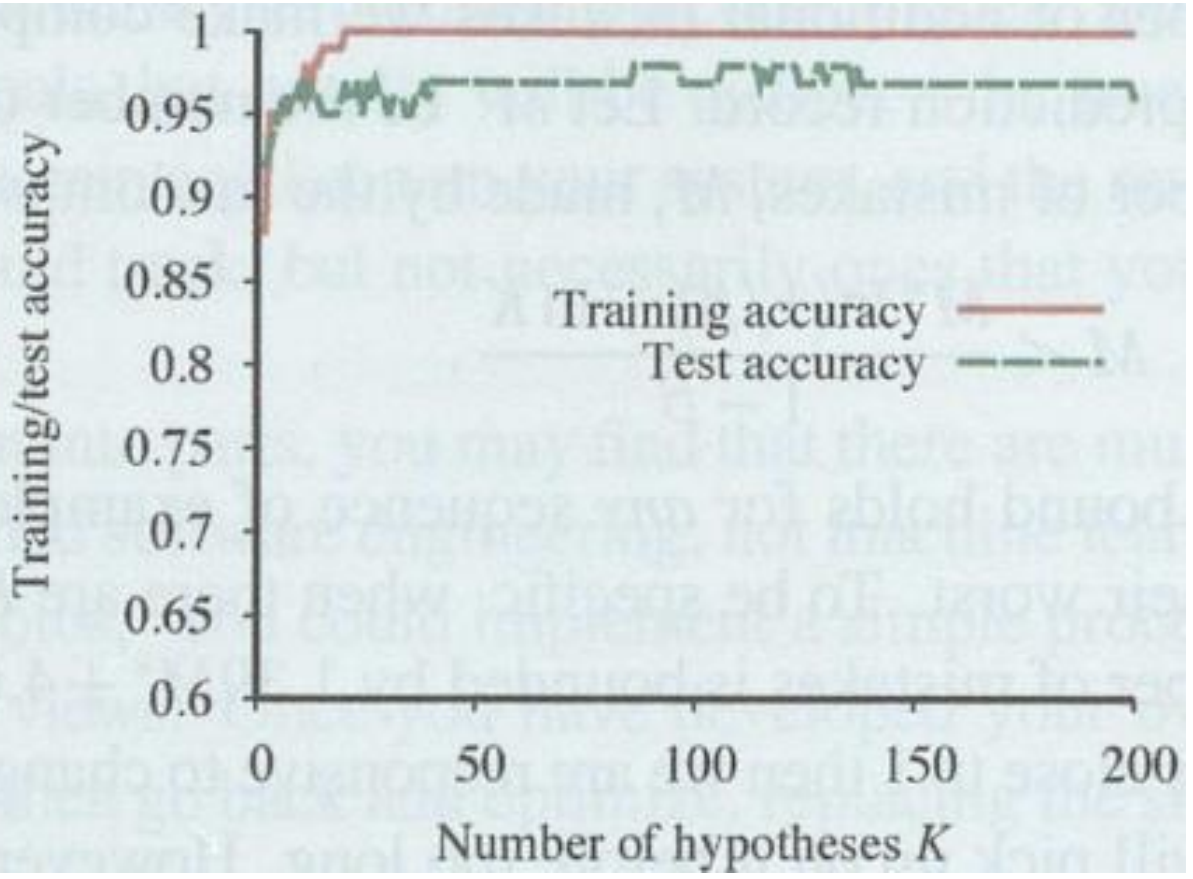
```
function ADABOOST(examples, L, K) returns a hypothesis
  inputs: examples, set of N labeled examples (x_1, y_1), ..., (x_N, y_N)
          L, a learning algorithm
          K, the number of hypotheses in the ensemble
  local variables: w, a vector of N example weights, initially all 1/N
                   h, a vector of K hypotheses
                   z, a vector of K hypothesis weights

  ε ← a small positive number, used to avoid division by zero
  for k = 1 to K do
      h[k] ← L(examples, w)
      error ← 0
      for j = 1 to N do        // Compute the total error for h[k]
          if h[k](x_j) ≠ y_j then error ← error + w[j]
      if error > 1/2 then break from loop
      error ← min(error, 1 − ε)
      for j = 1 to N do        // Give more weight to the examples h[k] got wrong
          if h[k](x_j) = y_j then w[j] ← w[j] · error/(1 − error)
      w ← NORMALIZE(w)
      z[k] ← ½ log((1 − error)/error)        // Give more weight to accurate h[k]
  return Function(x) : Σ z_i h_i(x)
```

Application of ADABOOST to Restaurant Data

- A decision stump, i.e. a decision tree with only one attribute, yields rather mediocre results (ca. 80%) even with 100 training cases (left; green line)
- Boosting decision stumps with K=5 increases the accuracy to ca. 93% (left, red line)
- Increasing K to 20 yields already 100% accuracy on training set, further increasing improves test set accuracy to maximum of 98% with K=137 (right)





Gradient boosting

- Gradient boosting uses gradient descent for boosting
  - It needs a differentiable loss function (e.g. squared error in regression or logarithmic loss in classification)
  - Instead of improving existing hypotheses (as e.g. in regression learning), it generates new hypotheses for ensemble learning (by changing parameters in the next tree)
  - Controls complexity of generated hypotheses (regularization and pruning)
- Efficient implementation in XGBOOST (eXtreme Gradient Boosting) package
  - Was used by every team in top 10 of the KDDCup 2015

Online Learning

- Standard assumption for learning from examples: the data is i.i.d. (independent and identically distributed)
- Ensemble learning is also useful for situation, where the assumption is violated, i.e. the data distribution can change over time
- Online learning algorithm „Randomized weighting majority algorithm“:
  - Adapts the weights of k experts continuously depending on the correctness of their recommendations (outputs) for each new problem

Developing Machine Learning Systems

Typical steps of using machine learning to solve practical problems:

- Problem formulation
- Data collection, assessment, and management
  - Feature engineering
  - Exploratory data analysis and visualization
- Model selection and training
- Trust, interpretability, and explainability
- Operation, monitoring, and maintenance

Problem Formulation (1)

- Identify learning problem in two steps:
  1. What problem do I want to solve for my users?
  2. What part(s) of the problem can be solved by machine learning?
- Example:
  - Ad 1: Make it easier for users to organize and access their photos → too vague
    - Help a user find all photos related to a certain topic (e.g. Paris) → better

Problem Formulation (2)

- Identify learning problem in two steps:
  1. What problem do I want to solve for my users?
  2. What part(s) of the problem can be solved by machine learning?
- Example:
  - Ad 2: Learn a function that maps a photo to a set of labels (is this really necessary for 1?)
    - Define a loss function for your machine learning component
      - e.g. measuring the system‘s accuracy at predicting a correct label
    - What kind of ground truth (labels) is available?
      - Supervised (SL), unsupervised (UL), reinforcement learning (RL)
      - Or semisupervised given a few labels with task to infer more labels
      - e.g. for recommending songs or movies:
        - Choice between SL (labels are given) and RL (continuous feedback by user)
    - Labels might be not reliable (e.g. photos of persons with age information)
      - Weakly supervised learning with noisy and imprecise labels

Data Collection, Assessment, and Management

- Most critical issue: What data is in what quality available?
  - Freely available: MNIST (60 000 digits), image net (14 000 000 photos with labels)
  - Must be created: Self manufacturing, crowd sourcing, paid or unpaid helpers
  - Data come from users
- Data provenance: Document meta data over the data (e.g. in tables)
- Problems with data:
  - Data protection issues: Consider anonymization
  - Bias in data: Consider using a subset
  - Not enough data: Consider data augmentation or transfer learning from more general data
  - Unbalanced classes in data: Consider under- or oversampling, weighted loss function
  - Dealing with outliers
  - Dealing with wrong labels: Requires to understand reasons for errors

Feature Engineering

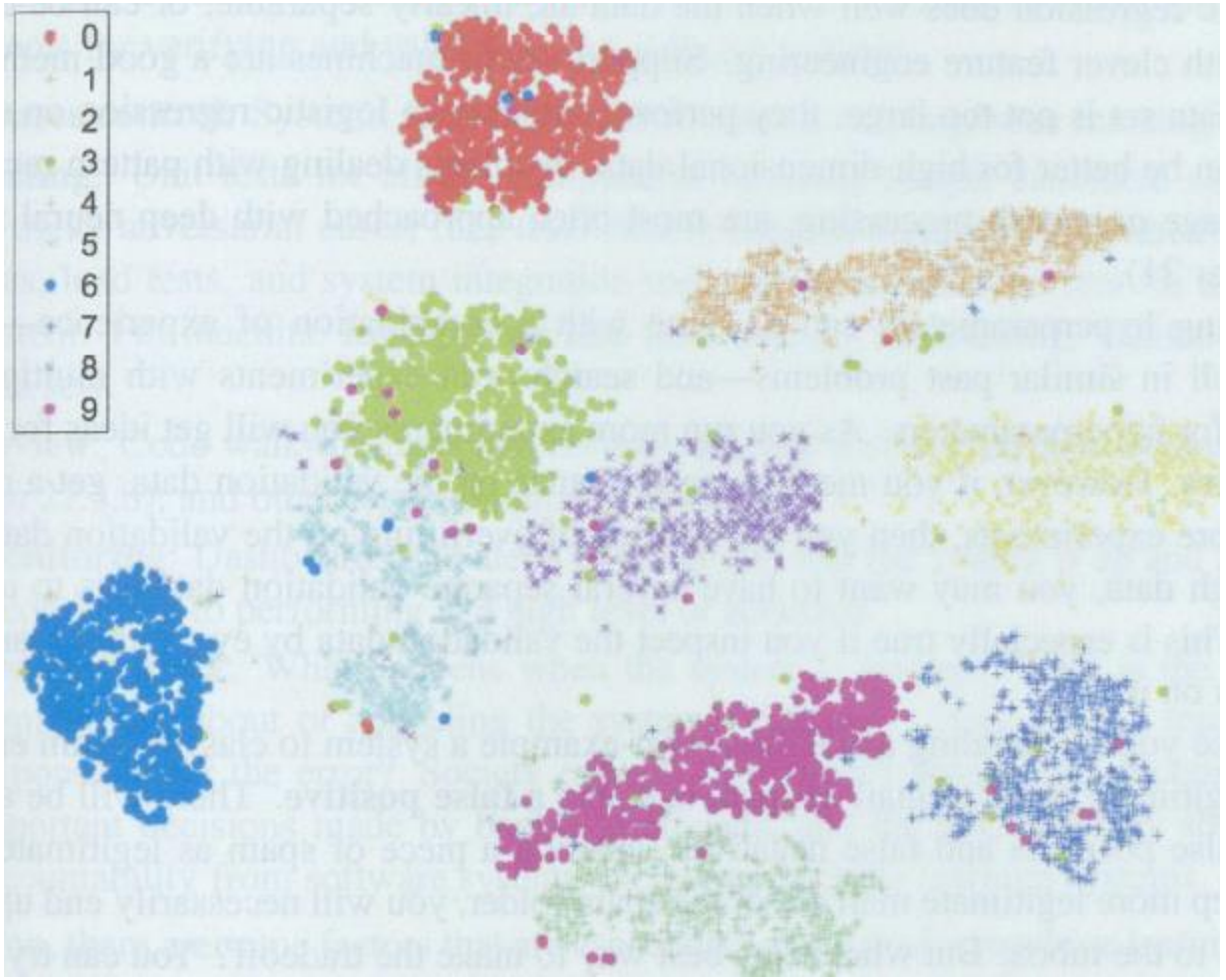
- Preprocessing
  - For numerical data:
    - Interval building (e.g. wait time: 0-10, 10-30, 30-60, >60), with domain knowledge?
    - Normalization: transformation of data to have a standard deviation of 1
  - Categorical data:
    - Transform multiple choice data in Boolean data
  - Introduction of new features based on domain knowledge
- How reliable, expressive, available are features? If key features are missing, can they be learned from other features?

- Pedro Domingos (2012): „At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily, the most important factor is features used.“

Exploratory Data Analysis and Visualization

For data understanding, visualiziations are very useful, but also summary statistics

- Looking at histograms or scatter plots can often help determine if data are missing are erroneous, normal distributed or heavy-tailed and what learning model are suitable
- Cluster data and view prototypes of each cluster and also outliers
  - Often iteratively done to find a good distance function for clustering
- Generate maps: two-dimensional or three-dimensional (projected in two dimensions)
  - Example: t-distributed stochastic neighbor embedding (t-SNE) for MNIST digit with the property, that similar points in original data set are close in the map



Model Selection and Training (1)

With cleaned data in hand and an intuitive feeling for it, it is time to build a model:

- Choose model class and optionally ensemble method
- Train model with training data
- Tune hyperparameter with validation data
- Debug the process
- Evaluate model on test data

Model Selection and Training (2)

Some recommendations for model selection:

- Random forests: Lots of categorical features with many maybe irrelevant
- Nonparametric models: Lot of data with no prior knowledge
- Logistic regression: Linear separable data (possibly with a clever conversion)
- Support vector machines: Data set not too large, similar to logistic regression for separable data, but better for high-dimensional data
- Deep neural networks: Pattern recognition like image or speech processing

Model Selection and Training (3)

Hyperparameter tuning:

- Do what worked well in similar past problems (e.g. from literature) and make experiments

Tradeoff between false negative and false positive:

- Different weights can be integrated in loss function
- **Receiver operating characteristic (ROC)** plots the tradeoff (for each value of hyperparameters)
- **Area under curve (AUC)** summarizes the ROC curve in a single number
- **Confusion matrix** for multiclassification problems shows how often a category is classified or misclassified as another category

Keep track of the general constraints for acceptance of the system

- A fast process from getting a new idea till the test results is often one of the most important factors for success in machine learning.

Trust, Interpretability, and Explainability

- Final evaluation metric is a necessary, but not sufficient condition for trust in your model
  - Related attributes: Reliability, accountability, safety
- **Trust for general software systems:**
  - Source control: Version control, build, and bug/issue tracking
  - Software testing: Unit tests, load tests, integration tests, …
  - Review: Code walk-throughs, privacy reviews, fairness reviews and other legal reviews
  - Monitoring: Dashboards and alerts
  - Accountability: What happens when the system is wrong? What is the process for complaining and appealing the system‘s decisions?
- **Interpretability:** Decision trees or linear regression models versus e.g. Neural Networks
- **Explainability:** More than interpretability (what is the case?); often better provided by a separate process to justify results, e.g. for a neural net classifying dogs, a separate explanation module might explain: has four legs, fur, a tail, floppy ears, long snout, etc.

Operation, Monitoring, and Maintenance

- Learning systems in practical use face several challenges:
  - **Long tail** of user inputs: The system might see inputs that were never tested before
    - Continuous monitoring necessary: Performance statistics, dashbord, alerts
  - **Nonstationarity:** The world changes over time
    - Example: Spam classification: Spammers react to spam classification systems, but also normal messanges might change (e.g. „sent from a smart phone“)
    - Dilemma: Well tested model built from old data versus an untested model built from latest data
      - Online learning might be associated with risks

Criteria for Tests in a Deployed Learning System

- **Tests for features and data:**
  - Features are documented; new features can be added quickly; privacy control; input feature code is tested
- **Tests for model development:**
  - Every model specification undergoes code review; every model is checked in a code repository, use actual metrics for evaluation; no better simpler models available;
- **Tests for machine learning infrastructure:**
  - Training is reproducable; models and full pipeline are tested, model allows debugging by observing step-by-step inference on a single example
- **Monitoring tests for machine learning:**
  - Quality and performance metrics in actual use correlate to training metrics; models are not out-dated,

Some Current Developments

- Explainable AI (XAI): Two kinds of explanations for machine learning systems:
  - For designers (e.g. heat maps)
  - For users
    - LIME system (2016) builds an interpretable linear model that approximates whatever machine learning system you have
    - SHAP (2018) uses Shapley value concept to determine the contribution of a feature
- Automated machine learning (AutoML): Learning to learn
  - Automated feature selection
  - Searching the space of hyperparameters
  - Tranfer learning (e.g. AutoML for deep learning models) and metalearning:
    - MAML (Model-Agnostic Meta-Learning) works with any model that can be trained by gradient descent and trains a core model, which can be fine-tuned with new data
  - Automatic Statistican (2019): Input: Some data; Output: A report mixing text, charts and calculations