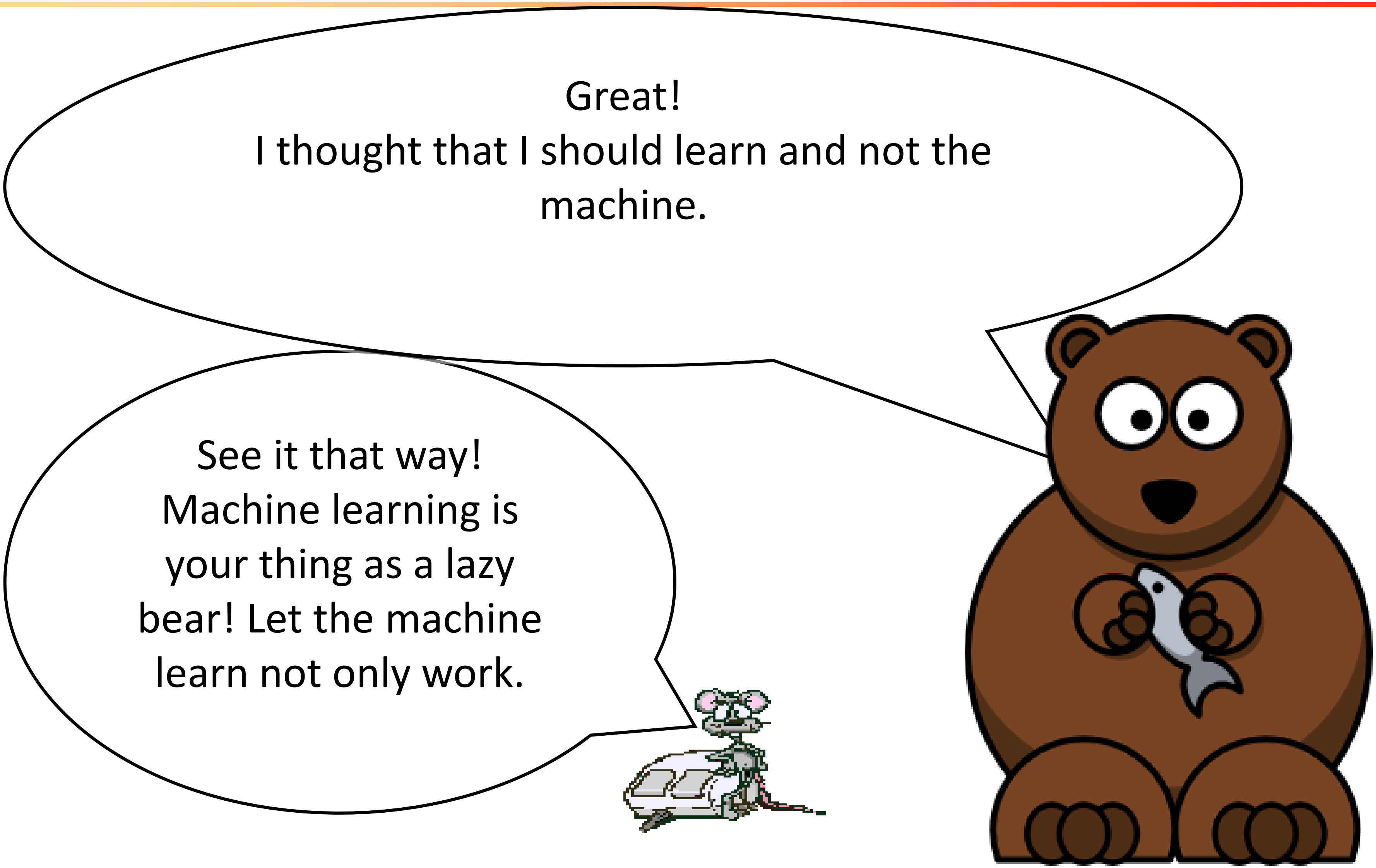
2 Machine Learning

- First, let us recall or understand what we denote as machine learning
- An engineering orientation of Tom Mitchell (1997) seems to be quite useful

  “A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.”

- Why is this thinking useful?
- On the contrary, traditional programming has to define rules, procedures, and sophisticated routines (algorithms) that determine in each detail what the computer has to do
- But is this always reasonable?

Consequences



Great!
I thought that I should learn and not the machine.

See it that way!
Machine learning is your thing as a lazy bear! Let the machine learn not only work.

2.1 Some basics

- In what follows, we try to understand some basic notations and concepts by asking
  - What is machine learning about?
  - Why do we use it?
  - How we can classify the existing systems?
- See Géron (2017, 2019)

Writing a spam filter in a traditional way

Necessary steps to do in traditional programming

1. Before writing down rules you have to analyze tons of emails in order to find out characterizing words in the header or specific sender names or domains or further significant attributes of spam mails
2. Subsequently, you write a corresponding detection algorithm that checks all these cases in order to provide a reliable identification
3. You have to test the spam filter after being launched in your real-world application. Whenever you notice identification problems or new spam attributes, you have to repeat the two steps ahead. Therefore you undergo a continuous process of testing, analyzing, and programming

But, as these tasks are not trivial, it is quite likely that the program becomes quite complex and maintaining proves to be a time-consuming inefficient task to do

Using machine learning

- On the contrary, in machine learning, the programmed spam filter is able to learn which words and phrases are promising predictors
- This learning can be done by comparing the frequency of these phrases in spam mails compared to ham mails
- Also such kind of programs have to be tuned (cycles of validation and correction) by the programmer, but are often much more compact and simpler to maintain
- Moreover, the every day usage is more efficient
  - After being validated, the system should be able to adapt the applied rules according to the experienced (modified) data dynamically coming in
  - Hence, the reprogramming of the spam filter occurs only rarely
- On top of that, there is frequently some versatility that allows to apply some basic components to different tasks

Machine learning seems to be appropriated for

- …problems for which existing solutions require a lot of tuning or huge lists of rules
- …problems for which no solution of acceptable quality is known by applying a traditional approach
- …problems for which relevant data is very fluctuating and this data directly influences the structure of the applicable algorithms
- …problems for which we have to find some insights into the structure of the problem, i.e., we use machine learning in order to understand the problem

2.1.1 Types of machine learning systems

- Today, there is an extremely increasing number of machine learning approaches that act quite differently and that are designed for various applications
- Thus, it seems to be useful to provide a useful classification
- Basically, we distinguish
  - …whether the training of the system is supervised by a teacher or not (supervised, unsupervised, semi-supervised, or reinforcement learning)
  - …whether it learns incrementally (with each new data) or not (online versus batch learning)
  - …whether the system learns by just comparing new data points with existing known ones or detect patterns in the training data and derive a predictive mathematical model (instance-based versus model-based learning)
- Note that these criteria are not complete, disjoint, or exclusive. Clearly, we may have further criteria or applications that combine different of them

Supervised learning

- Characterization
  - The algorithm is fed with training data that includes the desired solutions (called labels)
  - For instance, if a classification has to be done (e.g. by a spam filter), various examples are trained with the given class (spam or ham)
  - Another example is the prediction of a sought numeric value (realistic market price of a car depending on given features) done by regression. For this purpose, the system is trained with numerous real-world cases
  - Some regression algorithms can also be used for classification (by comparing with thresholds) and vice versa
- Examples
  - k-nearest neighbor
  - Linear regression
  - Logistic regression
  - Support vector machines (SVM)
  - Decision trees and random forests
  - Neural networks (besides semi-supervised and unsupervised neural networks)

Unsupervised learning

- Characterization
  - The training data is unlabeled (there is no teacher comparing the results)
  - For instance, you want to cluster the visitors of your online shop
  - At no time you tell (or can tell) the algorithm to which group someone belongs. Rather, the algorithm has to find it out on its own
  - Another example are visualization algorithms providing the user an overview on the data. Or anomaly detection: For instance, detecting unusual credit card transactions in order to identify fraud, the system is trained with normal instances
  - Association rule learning: Dig into large amounts of data and identify relations between the behavior of people: Who purchases potato chips also buys steaks etc.
- Examples
  - Clustering of data sets by
    - k-means clustering/k++ clustering
    - Hierarchical cluster analysis (HCA)
    - Expectation Maximization
  - Many approaches dealing with visualization and dimensionality reduction
  - Many approaches doing association rule learning

Semi-supervised learning

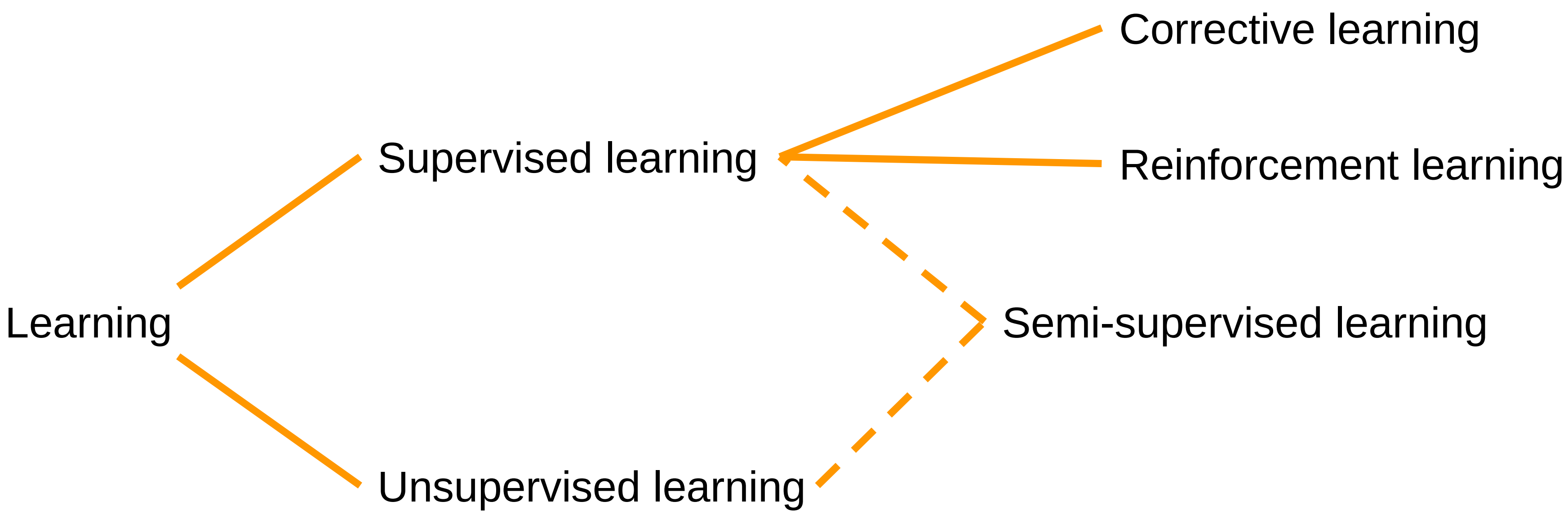
- Algorithms that deal with partially labeled training data, i.e., a lot of unlabeled data and a little bit of labeled data
- Note that this definition is quite ambiguous
- Therefore, let us consider a photo hosting service as an appropriate example
  - Usually you upload your family photos
  - Then, the service starts clustering these photos with the possible result that it identifies the same person on different photos (this is obviously the unsupervised part of the system)
  - After that, you are asked to label these persons by name and possibly correct some results. This naming simplifies future search processes
  - Many semi-supervised learning algorithms are just such combinations of unsupervised and supervised methods

Reinforcement learning

- In reinforcement learning system the system is frequently denoted as an agent
- This agent continuously observes the environment, select and perform actions while getting feedbacks for them
  - Rewards for intended, i.e., pursued actions and
  - Negative rewards, i.e., penalties
- The agent has to find the best strategy, denoted as a policy, to maximize the total received reward
- Robots often implement reinforcement learning algorithms to learn how to walk
- DeepMind’s AlphaGo program is another example for reinforcement learning. It learned its strategy by analyzing millions of games. It beats in May 2017 the world champion Ke Jie at the game of Go

Summary – Classes of learning algorithms

- Basically, we can depict the following classification of learning algorithms (see Rojas (1996))



Batch learning

- The system is incapable of incrementally learning
- It needs all available data at once
- As this takes some time, it is often done by an offline step (offline learning) and not during the ordinary application of the system
- Therefore, if new data is available and has to be considered, the system is trained with all data (new and old) from scratch. Subsequently, the new version replaces the former one
- As the training cycles are time consuming, such steps should be scheduled with some care

Online learning

- In contrast to batch learning systems, online learning systems are trained incrementally by feeding them in small batches or individually
- The effort of those learning steps is insignificant (learning the new data on the fly)
- One important aspect is the learning rate, i.e., how fast adapts the system its behavior to the specifics of the new training data?
- However, fast adaption is not always preferable if the system is fed with bad data (i.e., data that is not representative or distorted)
- Thus, online learning needs to be continuously controlled by checking its ongoing performance results

Instance-based learning

- These systems learn from each individual case
- In an extreme scenario each case is stored and if this case occurs again it is handled accordingly
- More sophisticated instance-based learning systems mathematically measure the *similarity between incoming instances* in order to decide whether a current case coincides with the learned ones
- For instance, spam filters frequently use instance-based learning
  - Spam filters are learned by heart
  - Then, similarity measures are applied as the number of relevant words two emails have in common

Model-based learning

- Model-based learning approaches generalize from a set of examples by building a model that is based on these examples. Then, this model is used to make predictions
- For instance, consider a scientific study that deals with the question *whether money makes people happy*
  - It may use the Better Life Index data from the OECD's website as well as stats about GDP per capita from the IMF's website
  - Then, both data is mixed together in order to define happiness in dependence of income
  - As you identify a trend in this mixed data (it actually looks like life satisfaction goes up more or less linearly as the country's GDP per capita increases), you decide to apply linear regression in order to model satisfaction as a linear function of income
  - This is the model selection step
- In summary: Studying the data, selecting a model, Training it on the data (i.e., the learning algorithm searched for the model parameter values that minimize a cost function), and finally applying the model to make predictions on new cases (this is called inference), hoping that this model will generalize well

2.1.2 Main challenges and pitfalls

The methods introduced in this section (and throughout the entire course) do not work by themselves, but need to be programmed and designed in a sophisticated way. Particularly, the efficient use of machine learning approaches may be endangered by the following challenges:

- Insufficient quantity of training data
- Non-representative training data
- Poor-quality training data
- Irrelevant Features
- Overfitting the training data
- Underfitting the training data

Insufficient Quantity of Training Data

- A toddler learns what an apple is …
    - by just seeing (and touching) one and hearing the word “apple” (maybe several times)
    - Afterwards, this child is able to recognize apples in all sorts of colors and shapes. This is amazing out of our view as ML-scientists
- In machine learning, however, it takes much more data to reach this point
- Finding sophisticated learning procedures is for sure one decisive side of the coin
- However, on the other, not less decisive side, there are studies that reveals that completely different learning algorithms (including fairly simple ones) may attain a comparable solution quality if the quantity of the training data is sufficiently high
- This underlines that insufficient quantity of training data is a serious shortcoming

Nonrepresentative Training Data

- Clearly, it is crucial that your training data be representative of the new cases you want to generalize to
- This is true whether you use instance-based learning or model-based learning
- For instance, if our formerly mentioned scientific study that deals with the question whether *money makes people happy* does the linear regression without richer countries the results may be not appropriate for countries like Switzerland or Luxembourg

Poor-Quality Data

- Clearly, if the training data is full of errors, outliers, and noise (reasons for that are often poor quality measurements or false channels of data acquisition), it is hard for the system to exploit the underlying pattern (it is gone due to the poor quality)
- Hence, it is worth the effort, to spend time cleaning up the data, i.e., delete outliers, decide about missing or corrupted values of some attributes etc.

Irrelevant Features

- In order to derive the existing pattern in the data, it is useful to condense the features to the relevant ones
- Feature engineering comprises
  - Feature selection: selecting the most useful features to train on among existing features.
  - Feature extraction: combining existing features to produce a more useful one (e.g., dimensionality reduction may help)
  - Creating new features by gathering new data

Overfitting

I am overfitted! Great!
I am the best.
I get every fish within a second.

No, no. OVERfitted means that you are too specialized by the training data and this is not useful for a general real-world case

What is overfitting?

- Roughly speaking, overfitting pictures the situation when the learning algorithm overgeneralizes its training data with the consequence that it performs well for the specific training data, but not for the general case
- Overfitting frequently happens when the applied model is too complex relative to the amount and noisiness of the training data. Thus, possible solutions may be
  - Simplify the model by reducing the integrated parameters (e.g., a linear model rather than a high-degree polynomial model),
  - Simplify the model by reducing the number of attributes in the training data or by constraining the model
  - Gather more training data
  - Reduce the existing noise in the training data (e.g., fix data errors and remove outliers)

The opposite: Underfitting

- Occurs when the model is to simple to learn the structure of the data
- For instance, if a linear model is applied to a phenomenon with exponential dependencies this would lead to underfitting

2.2 Decision trees

- Trees are very common data structure in computer science
- In a approximately full tree with node degree c we have an asymptotic depth of $\mathcal{O}(log_c(N))$
- Hence, access and update operations can be done quite efficiently
- Therefore, trees are promising data structures for efficiently storing larger data sets
- In what follows, we want to learn from these data set while storing this exploited knowledge in an efficient way
- As a consequence, despite its huge size, stored knowledge can be accessed very fast

2.2.1 Learning with decision trees – The problem

- Specifically, in what follows we consider a list of items that possess individual attributes. Moreover, each item belongs to a specific class that is given by the respective data set
  - Our task is to derive a tree that classifies the items by some of their stored attribute values
  - While starting from the root node with all items each inner node clusters the respective items according to their values for a specific attribute considered at this node
  - At a leaf node a final classification is provided
  - Therefore, after starting at the root node, each item is iteratively classified by following the inner nodes according to the respective attribute values of the item until the classification is done by the reached leaf

Preparing the mathematical definition

2.2.1.1 **Definition**

Let $n, k \in \mathbb{N}$ be natural numbers. A **set of $\boldsymbol{n}$ attributes** is given by $A = \{A_1, \dots, A_n\}$, while for each $i \in \{1, \dots, n\}$ the attribute A_i is defined as a set of possible values. A **classification into $\boldsymbol{k}$ classes** is a set $C = \{C_1, \dots, C_k\}$, while for each $j \in \{1, \dots, k\}$ C_j is a class that is finally assigned to each item (or case) that possesses individual values for each attribute.

Mathematically, we consider the following

2.2.1.2 **Definition**

Given n attributes $A = \{A_1, \dots, A_n\}$ and a classification $C = \{C_1, \dots, C_k\}$ into k classes as defined in Definition 2.2.1.1.

A **data set** M with m items possessing n attributes **to be learned by a decision tree** is defined as follows

$$M = \left(\begin{array}{ccc|c} m_{1,1} & \dots & m_{1,n} & m_{1,n+1} \\ \vdots & \ddots & \vdots & \vdots \\ m_{m,1} & \dots & m_{m,n} & m_{m,n+1} \end{array}\right)$$

The **value** of the ith item for the jth attribute is defined by entry $m_{i,j} \in A_j$, with $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, n\}$.

The **classification** of the ith item is defined by entry $m_{i,n+1} \in C$, with $i \in \{1, \dots, m\}$

Attributes of a data set

2.2.1.3 **Definition**

Let M be a data set with m items and n attributes as defined in Definition 2.2.1.2.

M is denoted as **non-trivial** if and only if it holds that

$\exists i \neq j \in \{1, \dots, m\}$: $m_{i,n+1} \neq m_{j,n+1}$

M is denoted as **consistent** if and only if it holds that

$\exists i \neq j \in \{1, \dots, m\}$:

$\left(\forall h \in \{1, \dots, n\}\text{: } m_{i,h} = m_{j,h}\right) \Rightarrow m_{i,n+1} = m_{j,n+1}$

Decision tree – Interpretation

2.2.1.4 **Definition**

A **decision tree** T **for a data set** M as defined in Definition 2.2.1.2 with an attribute set $A = \{A_1, \dots, A_n\}$ and a classification $C = \{C_1, \dots, C_k\}$ is a tree with leafs $C_i \in C$ and inner nodes $A_j \in A$. The edges that emerge from such an inner node $A_j \in A$ possess edge values $v \in A_j$

A **decision tree** T **classifies an item** $x \in A_1 \times \cdots \times A_n \times C$ into class $C_i \in C$ if there exists an edge path $(v_1, \dots, v_l)$ in tree T such that each edge value v_c of the attribute $a(c)$ visited by the path at depth c coincides with the corresponding entry of item x, i.e., it holds that $x_{a(c)} = v_c$ and the path finally reaches leaf $C_i \in C$.

Correctness of classification

2.2.1.5 **Definition**

A **decision tree** T **for a data set** M (as defined in Definition 2.2.1.2) with an attribute set $A = \{A_1, \dots, A_n\}$ and a classification $C = \{C_1, \dots, C_k\}$ **classifies an item** $x \in A_1 \times \dots \times A_n \times C$ **into class** $C_i \in C$ **correctly** if and only if it holds that $x_{n+1} = C_i$.

By storing the information in a decision tree T that is given in data set M, we demand, if M is consistent, that T classifies all items of M correctly. Moreover, we want to store the information (learned knowledge) in a most efficient way.

For this purpose, we have to define suitable performance measures.

Performance measure

2.2.1.6 **Definition**

Let T be a **decision tree for a data set** M as defined in Definition 2.2.1.2 with an attribute set $A = \{A_1, \dots, A_n\}$ and a classification $C = \{C_1, \dots, C_k\}$. Then, we define the following performance measures:

1. For each leaf $C_i \in C$ of T we define $l(C_i)$ as the **length** of the path starting from the root and ending at C_i that is measured by the transferred edges
2. The **external path-length of** T is defined by $\sum_{i=1}^{k} l(C_i)$
3. For each item $x = (x_1, \dots, x_n, x_{n+1}) \in M$ the function $l^M(x)$ gives the number of edges of the path in T that correctly classifies x
4. The **weighted total external path-length** of T is defined by $\sum_{x \in M} l^M(x)$

Interpreting the performance measure

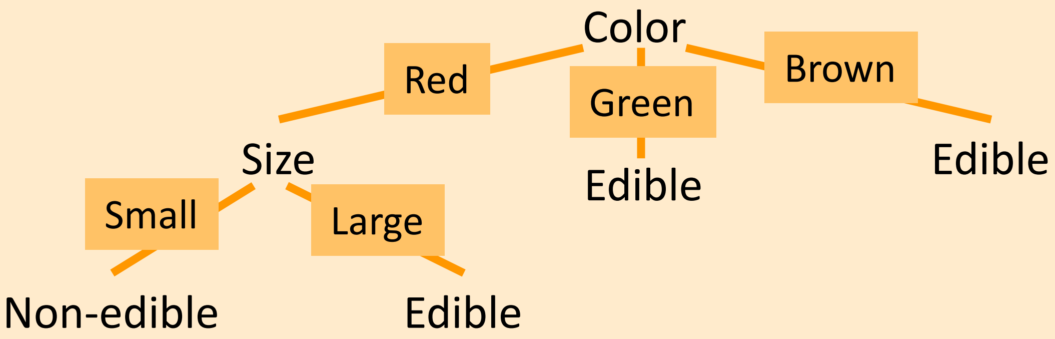
The performance measures of the generated decision tree T, as defined in Definition 2.2.1.6, can be interpreted as follows:

- The number of leafs gives the **number of rules stored in** T
- The external path-length is the **total storage consumption of decision tree** T
- Depth of T is the **maximal length of some stored rule**
- The weighted total external path-length of T is the **total time effort needed for classifying the data set** M

2.2.1.7 Example – Mushroom classification

Color	Size	Dots?	Edible?
Red	Small	Yes	No
Brown	Small	No	Yes
Brown	Large	Yes	Yes
Green	Small	No	Yes
Red	Large	No	Yes

Decision tree



Color

Red

Green

Brown

Size

Edible

Edible

Small

Large

Non-edible

Edible

External path-length amounts to $1 + 1 + 2 + 2 = 6$

Weighted total external path-length amounts to $2 + 1 + 1 + 1 + 2 = 7$

2.2.2 Excursion: Entropy in information theory

- The information theory was originated in 1948 by Shannon
- He published the paper named “A Mathematical Theory of Communication”
- In this paper, he proposed the measure of information entropy, which describes the amount of impurity in a set of features
- The entropy H of a set p comprising n possible events with an occurrence probability of p_i with $0 < p_i \leq 1$ and $\sum_{i=1}^{n} p_i = 1$ is given through

$$H(p_1, \dots, p_n) = -\sum_{i=1}^{n} p_i \cdot log_2(p_i)$$

- Note that, in this computation, *events with a zero probability are ignored*, i.e., they are put aside before computing the entropy value and therefore their contribution is set to zero

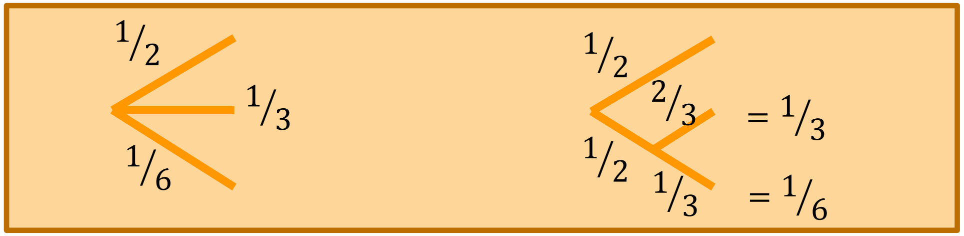
2.2.2 Excursion: Entropy in information theory

- The base 2 represents a coding into a binary digits, i.e., into bits
- These are the smallest information units as they can have only two distinct values
- The entropy measures the average information attained by executing one random choice according to the given occurrence probabilities

The assumptions made by Shannon

Shannon (1948) derived the entropy measure $H(p_1, \dots, p_n)$ on the basis of the following requirements that

1. H should be continuous in the p_i
2. If all the p_i are equal, i.e., $p_i = \frac{1}{n}, \forall i \in \{1, \dots, n\}$, then H should be a monotonic increasing function of n. With equally likely events there is more choice, or uncertainty, when there are more possible events
3. If a choice be broken down into two successive choices, the original *H* should be the weighted sum of the individual values of *H*. The meaning of this is illustrated in the figure below. At the left we have three possibilities $p_1 = \frac{1}{2}, p_2 = \frac{1}{3}, p_3 = \frac{1}{6}$. On the right we first choose between two possibilities each with probability $\frac{1}{2}$, and if the second occurs make another choice with probabilities $\frac{2}{3}, \frac{1}{3}$. The final results have the same probabilities as before.



The assumptions made by Shannon

3. Continuation: As there are identical probabilities at the end, we require, in this special case, that it holds $H\left(\frac{1}{2},\frac{1}{3},\frac{1}{6}\right) = H\left(\frac{1}{2},\frac{1}{2}\right) + \frac{1}{2} \cdot H\left(\frac{2}{3},\frac{1}{3}\right)$. The coefficient $\frac{1}{2}$ is because this second choice only occurs half the time.

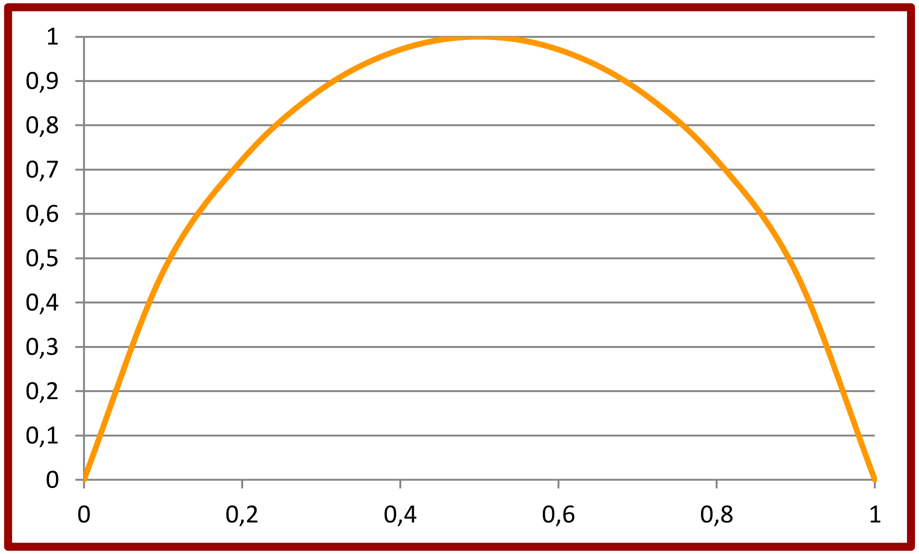
Shannon (1948) proves that the only entropy measure $H(p_1, \dots, p_n)$ that satisfies these three assumptions possesses the form:

$$H(p) = -K \sum_{i=1}^{n} p_i \cdot log_2(p_i)$$

while K is a positive constant.

Observations

- The entropy $H(p_1, \dots, p_n)$ is **maximal** if we have equal probabilities (minimal knowledge of the given distribution), i.e., if we have $\forall i \in \{1, \dots, n\}: p_i = \frac{1}{n}$ it holds that $H(p_1, \dots, p_n) = -\sum_{i=1}^{n} p_i \cdot log_2(p_i) = -\sum_{i=1}^{n} \frac{1}{n} \cdot log_2\left(\frac{1}{n}\right) = -\frac{1}{n}\sum_{i=1}^{n} log_2\left(\frac{1}{n}\right) = -\,log_2\left(\frac{1}{n}\right) =$ $-\,log_2(1) + log_2(n) = log_2(n)$
- On the contrary, the entropy is **minimal** (equal to zero) if there is a safe event, i.e., $\exists i \in \{1, \dots, n\}: p_i = 1$ and have perfect knowledge of the given distribution
- It holds that $0 \le H(p_1, \dots, p_n) \le log_2(n)$
- For illustration purposes, the entropy $H(p, 1-p)$ is plotted below



2.2.2 ID3/C4.5/C5

- In what follows, we consider the problem of finding an optimal decision tree
- However, Hyafil and Rivest (1976) show that this problem is strongly NP-hard
- Therefore, in what follows, we will design a heuristic approach
- By considering the small example 2.2.1.7, we observe that the values specific items possess for a certain attribute are not independent from values occurring for other attributes
- Hence, we see that the construction of a corresponding decision tree has to chose the attributes positioned at the inner nodes in a sophisticated way in order to minimize the resulting weighted total external path-length of the decision tree

Basic structure of the decision tree construction

Given: Data set M with m item belonging to k classes while possessing n attributes

Function $Decision\_tree\_construction(M)$:

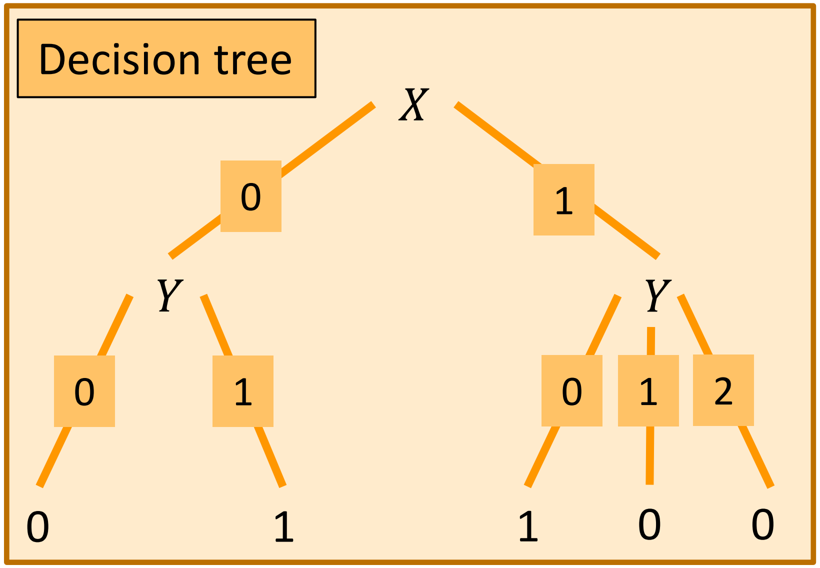
1. Delete all attributes that have only a single value in M (all items in M are not distinguishable by that attribute). IF M is trivial THEN build a tree T with one node that gives the single class $C = C_i$. Moreover, if there is no attribute left assign the classification $C_j \in C$ that the most items in M posses.
2. IF M is NOT trivial THEN select heuristically (by following a predetermined criterion) an attribute $A_j \in A$ and consider the values that items in M possess for attribute A_j. Let $\{O_1, O_2, \ldots, O_r\}$ be the set of r occurring attribute values for attribute A_j. Build the disjoint separation of $M = M_1 \cup M_2 \cup \cdots \cup M_r$ with $M_i = \left\{ x \in M \mid x_j = O_i \right\}$.
3. For $i = 1$ TO r: Call recursively $Decision\_tree\_construction(M_i)$

Lacking attribute value combinations in data sets

- Above procedure correctly classifies all items of the initial dataset M
- This can not be guaranteed for new items, even if they only contain Attribute values encountered in the initial Dataset
- Example:

Dataset

X	Y	C
0	0	1
0	1	0
0	1	0
1	0	1
1	1	0
1	2	0



- What about an item with attribute values $X = 0$ and $Y = 2$?

Lacking attribute value combinations in data sets

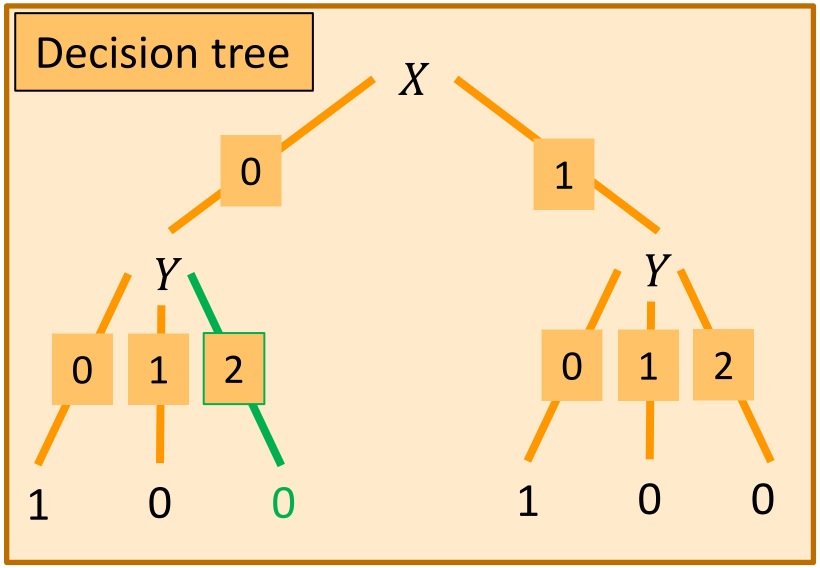
- In step 2 of the decision tree construction procedure a disjoint separation was built solely based on the attribute values occurring in the considered dataset, but not by considering all possible attribute value combination of a theoretical dataset
- The description of the separation procedure given by Quinland (1993) slightly differs from the one given above according to the creation of a branching step. In the description provided by Quinland (1993) the disjoint separation is based on **all** possible attribute values
- Moreover, empty disjoint separations M_i that form a leaf node with classification ‚null', represent a no classification case
- Obviously, a classification query can result in such an answer whenever no feasible edge exists for classifying a new item
- Quinland (1993) suggests that a better solution would be to generalize from the dataset of the parent node and assign this leaf the most frequent class, although this is not implemented in ID3

Lacking attribute value combinations in data sets

- Let us return to our small example: Here, we now follow the suggested approach of Quinland (1993) and include nodes for lacking attribute values
- The classification is determined by choosing the most frequent class of items at the parent node
- Thus, items with the (currently) not occurring combination $X = 0$ and $Y = 2$ are classified as 0 since the majority of items with $X = 0$ are classified as 0

Dataset

X	Y	C
0	0	1
0	1	0
0	1	0
1	0	1
1	1	0
1	2	0



Special cases for a subtree T and their handling

- Special case 1: T contains one or more cases, but all belong to a single class C_j

  The decision tree for T contains a single leaf that identifies class C_j

- Special case 2: T contains no cases at all
  - The decision tree is again just a leaf, but the class to be associated with the leaf must be determined from information other than T
  - For instance, the leaf might be chosen in accordance with some background knowledge of the domain, such as the overall majority class
  - C4.5 uses the most frequent class at the parent of this node

Gain criterion

- As the basic structure of the decision tree construction procedure reveals, we have to decide for the next attribute to be assigned to the next inner node only according to the immediate partition of the considered data set into subtrees
- Hence, in this decision, we do not integrate the consequences caused by subsequent divisions in further subsets
- We consider a data set M with n items, k classes, and m attributes
  - $freq(C_i, M)$: Number of items x in data set M with classification C_i, with $i \in \{1, \dots, k\}$
  - $freq(M) = |M| = n$: Size of data set M
  - $prob(C_i, M)$: Probability of randomly drawing an item with classification C_i in data set M
  - $info(M)$: Expected information received by drawing an item from data set M
- It holds that $prob(C_i, M) = \frac{freq(C_i,M)}{freq(M)} = \frac{freq(C_i,M)}{|M|}$
- I.e., such a draw of an item in data set M with classification C_i conveys the information of $-log_2\left(\frac{freq(C_i,M)}{|M|}\right)$ bits

Expected information of a data set S

- We compute $info(M) = -\sum_{i=1}^{k} \frac{freq(C_i,M)}{freq(M)} \cdot log_2\left(\frac{freq(C_i,M)}{freq(M)}\right)$ as the number of bits that such a drawing of an item from M conveys
- Clearly, it is just the entropy of M
- We apply this measurement before and after using a chosen attribute A_j for partitioning a considered data set M into subtrees $T_1, \ldots, T_l$
- Hence, we derive the information gain by computing

$$info(M, A_j) = \sum_{i=1}^{l} \frac{freq(T_i)}{freq(M)} \cdot info(T_i) \text{ and}$$

$$gain(M, A_j) = info(M) - info(M, A_j)$$

- The abbreviation $gain(M, A_j)$ provides us with the information we perceived by conducting the partition of the set M pertaining the values of items in M according the attribute A_j
- A reduced entropy reveals that we know more about the structure of the current distribution pertaining the classification of the items

Message of the gain criterion

- Consequently, in order to maximize the gained information in each separation, the gain criterion chooses the attribute for separation that maximizes the information gain
- Hence, for data set M and a classification $C = \{C_1, \ldots, C_k\}$, choose C_j fulfilling $gain\left(M, A_j\right) = max\left\{gain(M, A_l) \mid l \in \{1, \ldots, k\}\right\}$

Continuing the small mushroom example

- We come back to the mushroom example
- Hence, we consider again the following data set

Color	Size	Dots?	Edible?
Red	Small	Yes	No
Brown	Small	No	Yes
Brown	Large	Yes	Yes
Green	Small	No	Yes
Red	Large	No	Yes

- There are three attributes: color, size, and dots. We compute the entropy values
- We start with the given data set M and obtain

$$info(M) = -\frac{1}{5} log_2 \left(\frac{1}{5}\right) - \frac{4}{5} log_2 \left(\frac{4}{5}\right) = 0{,}72192809$$

Mushroom example – Attribute color

- We obtain three subtrees with the respective values red, brown, and green
- This can be illustrated by the following sub data sets

Color	Edible	Non-edible	Sum
Red	1	1	2
Brown	2	0	2
Green	1	0	1

- We compute the entropy of the three resulting subtrees
  - Red: $-\frac{1}{2} log_2\left(\frac{1}{2}\right) - \frac{1}{2} log_2\left(\frac{1}{2}\right) = 1 = log_2(2)$
  - Brown: $-\frac{2}{2} log_2\left(\frac{2}{2}\right) = 0$
  - Green: $-\frac{1}{1} log_2\left(\frac{1}{1}\right) = 0$
  - Thus, we obtain $info(M, color) = \frac{2}{5} \cdot 1 + \frac{2}{5} \cdot 0 + \frac{1}{5} \cdot 0 = \frac{2}{5} = 0{,}4$
  - $gain(M, color\ ) = info(M) - info(M, color) = 0{,}72192809 - 0{,}4 = 0{,}32192809$

Mushroom example – Attribute size

- We obtain two subtrees with the respective small and large
- This can be illustrated by the following sub data sets

Size	Edible	Non-edible	Sum
Small	2	1	3
Large	2	0	2

- We compute the entropy of the two resulting subtrees
  - Small: $-\frac{2}{3}log_2\left(\frac{2}{3}\right)-\frac{1}{3}log_2\left(\frac{1}{3}\right)=0{,}91829583$
  - Large: $-\frac{2}{2}log_2\left(\frac{2}{2}\right)=0$
  - Thus, we obtain $info(M,size)=\frac{3}{5}\cdot 0{,}91829583+\frac{2}{5}\cdot 0=0{,}5509775$
  - $gain(M,size)=info(M)-info(M,size)=0{,}72192809-0{,}5509775=0{,}17095059$

Mushroom example – Attribute dots

- We obtain two subtrees with the respective values with and without dots
- This can be illustrated by the following sub data sets

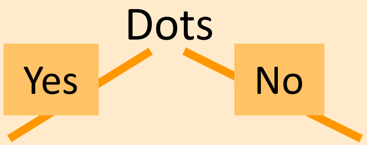
Size	Edible	Non-edible	Sum
With dots	1	1	2
Without dots	3	0	3

- We compute the entropy of the two resulting subtrees
  - With dots: $-\frac{1}{2} log_2\left(\frac{1}{2}\right) - \frac{1}{2} log_2\left(\frac{1}{2}\right) = 1 = log_2(2)$
  - Without dots: $-\frac{3}{3} log_2\left(\frac{3}{3}\right) = 0$
  - Thus, we obtain $info(M, dots) = \frac{2}{5} \cdot 1 = 0{,}4$
  - $gain(M, dots\ ) = info(M) - info(M, color) = 0{,}72192809 - 0{,}4 = 0{,}32192809$

We choose the attribute dots

- As it possesses less attribute values
- We therefore obtain with $info(M) = 0{,}4$

Decision tree



Color	Size	Edible
Brown	Large	Yes
Red	Small	No

Further distinctions are necessary

Color	Size	Edible
Brown	Small	Yes
Green	Small	Yes
Red	Large	Yes

Trivial case as all items are edible
No further distinctions are necessary

Mushroom example – Attribute color

- We obtain two subtrees with the respective values red and brown
- This can be illustrated by the following sub data sets

Color	Edible	Non-edible	Sum
Red	1	0	1
Brown	0	1	1

- We compute the entropy of the two resulting subtrees
  - Red: $-\frac{1}{1} log_2\left(\frac{1}{1}\right) = 0$
  - Brown:$-\frac{1}{1} log_2\left(\frac{1}{1}\right) = 0$
  - Thus, we obtain $info(M, color) = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 0 = 0$
  - $gain(M, color\ ) = info(M) - info(M, color) = 0{,}4 - 0 = 0{,}4$

Mushroom example – Attribute size

- We obtain two subtrees with the respective values large and small
- This can be illustrated by the following sub data sets

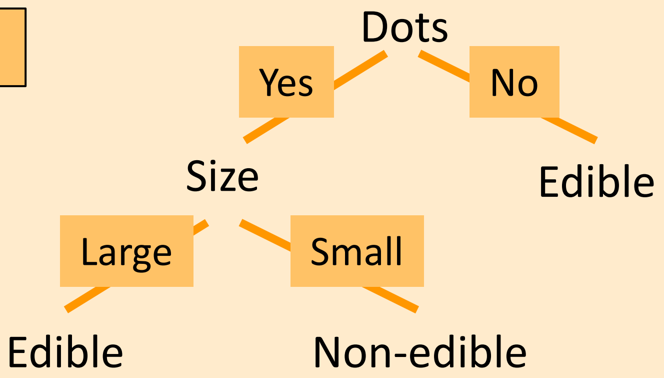
Size	Edible	Non-edible	Sum
Large	1	0	1
Small	0	1	1

- We compute the entropy of the two resulting subtrees
  - Large: $-\frac{1}{1} log_2 \left(\frac{1}{1}\right) = 0$
  - Small:$-\frac{1}{1} log_2 \left(\frac{1}{1}\right) = 0$
  - Thus, we obtain $info(M, size) = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 0 = 0$
  - $gain(M, size\ ) = info(M) - info(M, size) = 0{,}4 - 0 = 0{,}4$

We choose the attribute size

- We obtain the following complete decision tree
- It has the external path length $1 + 2 + 2 = 5$
- It has a total weighted path length $1 + 1 + 1 + 2 + 2 = 7$

Resulting decision tree



- Note that by firstly choosing the attribute color instead (this was also possible as the attainable gain is identical) we obtain the first tree with longer external path length
- The reason for this is that color possesses three occurring values and dots only two
- This aspect will be considered in more detail next

Gain ratio criterion – Motivation

- The first version of the ID3 algorithm solely applies the gain criterion
- Quinland (1993) reports that he noticed that there is a bias in this procedure for attributes with many occurring values as, due to more values, the information gain is larger
- However, as we do not want to give an incentive to this aspect, Quiland (1993) proposes to modify the gain criterion by relating it to the entropy of the value distribution

Gain ratio criterion – Mathematical definition

- We consider a data set M with n items, k classes, and m attributes
- All abbreviations are given as before
- We consider a chosen attribute A_j for partitioning a considered data set M into subtrees $T_1, \dots, T_l$
- The abbreviation $split$ provides the information given in the separation according to attribute A_j

$$split\ info(M, A_j) = -\sum_{i=1}^{l} \frac{freq(T_i)}{freq(M)} \cdot log_2\left(\frac{freq(T_i)}{freq(M)}\right)$$

- The gain ratio expresses the attained information gain relative to the effort of the performed separation to attribute A_j

$$gain\ ratio(M, A_j) = \frac{gain(M, A_j)}{split\ info(M, A_j)}$$

How to apply the gain ratio criterion

- Based on these values, Quinland (1993) proposes to follow the gain ratio criterion such that an attribute is selected that maximizes the gain ratio, subject to the constraint that the information gain must be large, i.e., at least as great as the average gain over all attributes examined
- On the next slide, we apply the gain ratio criterion to our mushroom example

Applying the gain ratio to the mushroom example

- During the first choice of an attribute we obtain

$$split\ info(M,color) = -\frac{2}{5}log_2\left(\frac{2}{5}\right) - \frac{2}{5}log_2\left(\frac{2}{5}\right) - \frac{1}{5}log_2\left(\frac{1}{5}\right)$$
$$= -\frac{4}{5}\cdot(-1{,}3219280949) - \frac{1}{5}\cdot(-2{,}3219280949) = 1{,}5219280949$$

- Therefore, it holds that

$$gain\ ratio(M,color) = \frac{0{,}32192809}{1{,}5219280949} = 0{,}2115264782$$

- During the first choice of an attribute we obtain

$$split\ info(M,size) = -\frac{3}{5}log_2\left(\frac{3}{5}\right) - \frac{2}{5}log_2\left(\frac{2}{5}\right) = 0{,}97095059$$

- Therefore, it holds that

$$gain\ ratio(M,size) = \frac{0{,}17095059}{0{,}97095059} = 0{,}17606518$$

- During the first choice of an attribute we obtain

$$split\ info(M,dots) = -\frac{2}{5}log_2\left(\frac{2}{5}\right) - \frac{3}{5}log_2\left(\frac{3}{5}\right) = 0{,}97095059$$

- Therefore, it holds that

$$gain\ ratio(M,dots) = \frac{0{,}32192809}{0{,}97095059} = 0{,}3315597$$

Result

- This time, we do not have a choice and have to choose the dot-attribute in the first place
- This underlines the motivation for introducing the gain ratio criterion

Observations

- Quinland (1993) reports that, in his experience, the gain ratio criterion is robust and typically gives a consistently better choice of test than the gain criterion
- It even appears advantageous when all tests are binary but differ in the proportions of cases associated with the two outcomes

From ID3 to C4.5/C5.0

- Until now, we have considered ID3 as a decision tree generation procedure
- Quinland (1993) proposes some extensions leading to the program C4.5
  - Handling of continuous attributes
  - Handling training data with missing attribute values
  - Pruning trees after creation
- Further improvements were implemented in C5 (Link, Link)
  - Speedup – C5.0 is significantly faster than C4.5
  - Improved memory usage
  - Generated decision trees are of reduced size
  - Provision of using case-depending weights (item dependent)
  - Attribute winnowing

Tests

- Quinland (1993) uses the generalized term *test* instead of using attributes for generating the next node in the decision tree
- This is reasonable as in C4.5/C5 each node does not necessarily coincide with an attribute as we have defined it in the so-called "Basic structure of the decision tree construction"
- In contrast to this, the decision tree generation procedure C4.5 contains mechanisms for proposing three types of tests
  - The **standard test on a discrete attribute**, with one outcome and branch for each possible value of that attribute (this was solely considered before)
  - A more complex test, based on a discrete attribute, in which the possible values are allocated to a variable number of groups (that have to be generated) with one outcome for each group (rather than each value (**condensing the values**)

Tests

  - If attribute A has **continuous numeric values**, a binary test with outcomes $A \leq Z$ and $A > Z$, based on comparing the value of A against a threshold value Z
- All these tests are evaluated in the same way
  - The gain ratio (alternatively the gain) criterion is applied that arises from the produced partitions, respectively
  - It turns out to be useful to require for each partition (test) that at least two of the resulting subsets contain a reasonable number of items. Specifically, this additional restriction should avoid near-trivial splits. Note that the minimum number can be adjusted application-dependent

Continuous attributes

- If an attribute is known to be continuous we may face the problem of arbitrary thresholds
- However, this is not the case as we can use the following procedure (see Paterson and Niblett (1982) or Breiman et al. (1984)) for finding appropriate thresholds against which to compare the values of continuous attributes
- As before, we assume to consider a continuous attribute $A_i \in A$ within a data set M that comprises m items with n attributes
- Hence, for the considered attribute A_i, we have the ordered values $\{v_1, \dots, v_m\}$

Continuous attributes

- Consider an arbitrary threshold v between v_i and v_{i+1}. Clearly, independent of its specific value v separates the items into the ones whose values for A_i are in $\{v_1, \dots, v_i\}$ and those whose values are in $\{v_{i+1}, \dots, v_m\}$
- Hence, we have only $m - 1$ possible splits of the items in M
  - After sorting the existing values, the separation can be carried out in one pass, updating the distributions to the left and right of the threshold on the fly
  - For this purpose, one should use the midpoint $\frac{v_i + v_{i+1}}{2}$ as the threshold
- One possibility is to build a binary split and test the gain and gain ratio for testing all reasonable threshold values $\frac{v_i + v_{i+1}}{2}$, $i = 1, \dots, m - 1$ (reasonable means that in both groups are more that one element)

Unknown attribute values

- So far, the introduced decision tree generation procedure assumes that, in the considered data set, all items have well-defined values for all listed attributes
- Unfortunately, in real-world applications, this does not has to be the case. Rather, it is quite common that data is not complete
- We are facing the choice of discarding the data or amending the procedure accordingly. For the latter, the literature provides various proposals while, in what follows, we consider the handling of C4.5/C5.0
  - As a tested attribute can provide no information about the class membership of items whose value of the test attribute is unknown, this item is left out when respective $info$ values are computed
  - This applies to both needed values $info(M)$ and $info(M, A)$

Unknown attribute values

- Hence, we compute the values $info(M)$ and $info(M, A)$ as before, except that only cases with known values of A are taken into account
- But, as we have left out items, the attained gain has to be weighted with the proportion of participating items (probability P that the value of attribute A is known in M) that in data set M
- Thus, we define $gain(M, A) = P \cdot (info(M) - info(M, A))$
- Furthermore, the definition of $split\ info(M, A)$ can be altered by regarding the cases with unknown values as an additional group.
- Consequently, if the considered attribute A has n outcomes in M, this leads to an entropy values basing on $n + 1$ subsets

Example – Item 6 is ignored for attribute outlook

Item	Outlook	Temp (°F)	Humidity	Windy?	Class – Play?
1	Sunny	75	70	True	Yes
2	Sunny	80	90	True	No
3	Sunny	85	85	False	No
4	Sunny	72	95	False	No
5	Sunny	69	70	False	Yes
6	?	72	90	True	Yes
7	Overcast	83	78	False	Yes
8	Overcast	64	65	True	Yes
9	Overcast	81	75	False	Yes
10	Rain	71	80	True	No
11	Rain	65	70	True	No
12	Rain	75	80	False	Yes
13	Rain	68	80	False	Yes
14	Rain	70	95	False	Yes

Computed results

The $info$ and $gain\ values$ are computed while ignoring the sixth item:

- $info(M) = -\frac{8}{13} \cdot log_2\left(\frac{8}{13}\right) - \frac{5}{13} \cdot log_2\left(\frac{5}{13}\right) = 0.961$
- $info(M, outlook) =$ $\frac{5}{13} \cdot \left(-\frac{2}{5} \cdot log_2\left(\frac{2}{5}\right) - \frac{3}{5} \cdot log_2\left(\frac{3}{5}\right)\right) + \frac{3}{13} \cdot \left(-\frac{3}{3} \cdot log_2\left(\frac{3}{3}\right)\right) + \frac{5}{13} \cdot \left(-\frac{3}{5} \cdot log_2\left(\frac{3}{5}\right) - \frac{2}{5} \cdot log_2\left(\frac{2}{5}\right)\right) = 0{,}747$
- $gain(outlook, M) = \frac{13}{14} \cdot \left(info(M) - info(M, outlook)\right) = 0{,}199$

The $split\ info$ computation considers an additional subset:

- $split\ info(M, outlook) = -\frac{5}{14} \cdot log_2\left(\frac{5}{14}\right) - \frac{3}{14} \cdot log_2\left(\frac{3}{14}\right) - \frac{5}{14} \cdot log_2\left(\frac{5}{14}\right) - \frac{1}{14} \cdot log_2\left(\frac{1}{14}\right) = 1{,}809$
- $gain\ ratio(outlook, M) = 0{,}110$

Note that by changing the outlook value of item 6 from “?” to “overcast” would increase this $gain\ ratio$ value to 0,156

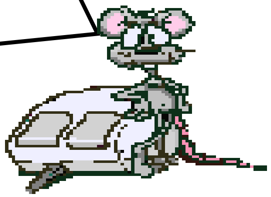
Pruning

- Pruning pursues the reduction of a generated decision tree for efficiency or correctness reasons, i.e., it intends to replace certain subtrees with leaves
- Therefore, pruning reduces specific parts of the current decision tree
- Basically, there are two alternative ways for pruning
  - Prepruning: During the tree generation process it is decided that a currently considered data set is not further divided, i.e., we have to integrate such kind of stopping criterion
  - Pruning after tree generation: removing retrospectively some of the subtrees built during the preceding tree generation process

Consequences

Why do we need pruning? I thought that the found tree was efficiently generated for the data base…

Clearly, if the data base is consistent, there is no erroneously classified item. But, this does not have to be the case and please note that in many applications, it is only training data!



Small example – Overfitting

- The partitioning procedure used for decision tree generation introduced so far assumed that the data set was
  - More or less consistent and
  - Does not lead to a tree structure that overfits the data
- However, in real-world applications the items in the data set may be not consistent
- Furthermore, after being generated, the tree may be used for further items with modified interdependencies between the attribute values and their classification
- For illustration purposes, we consider the extreme case of random data
  - Let us assume we have two classes while one class (class 1) has the dominating probability $p \geq 0.5$
  - If a **most simple classifier** assigns all cases to this first class (corresponding decision tree consists only of one node, i.e., it is a leaf with class 1 identification) its expected error rate is obviously $1 - p$

More can be less

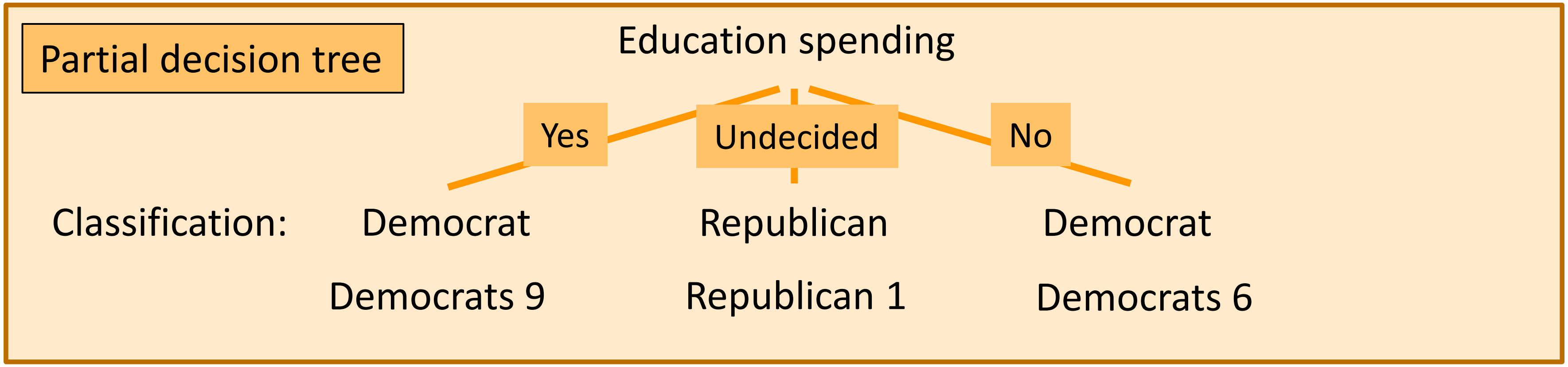
- We also consider an alternative **more complex classifier** that assigns an item with probability p to class 1 and with probability $1-p$ to class 2
- The error rate is now the sum of
  - the probability that a case assigned to class 2 belongs to class 1, i.e., $p \cdot (1-p)$, and
  - the probability that a case assigned to class 1 belongs to class 2, i.e., $(1-p) \cdot p$
  - Just to check: the remaining cases have a total probability of $p^2 + (1-p)^2 = 2p^2 - 2p + 1$ and thus we obtain $2p^2 - 2p + 1 + 2p \cdot (1-p) = 1$
- Thus, we obtain $2p \cdot (1-p)$ and since $p \geq 0.5$ this error rate is at least $1-p$
- Therefore, the **simple classifier outperforms the second more complex one for the expected error rate**
- Clearly, the most simple classifier benefits from the fact that the whole classification game is guessing and therefore, on the long run, it is best to go for the majority of cases (i.e., it uses the entire knowledge)
- But in real-world applications, data sets are at least partly indeterminate because the attributes do not capture all information relevant to classification

Error-based pruning

- C4.5/C5.0 applies pruning after the tree generation
- Specifically, a subtree is replaced by a leaf (pruned to it) if the expected error rate of this leaf (this has to chose a majority classification) is smaller than an upper limit derived for the respective subtree
- For this purpose, a confidence level CF is defined (in C4.5 the default confidence level is 25%) and the upper limit is defined by the confidence limits for the binomial distribution
- It is abbreviated as $U_{CF}(E, N)$ with
  - N: number of items in the data set
  - E: number of erroneously classified items in the considered data set
- Hence, the correct/non-correct classification is interpreted as a Bernoulli experiment

Example – Democrats and Republicans

- We consider a subtree of a decision tree derived from congressional voting data in the United States of America
- It classifies the members to the respective parties
- The attribute is education spending with the values (n/y/u) and the classification for the items in the data set



Partial decision tree

Education spending

Yes

Undecided

No

Classification: Democrat Republican Democrat

Democrats 9 Republican 1 Democrats 6

- Hence, there occurs no classification error for the training data (i.e., for the current data set)

Example – Democrats and Republicans

- However, in order to estimate errors for future cases, Quinland (1993) proposes to apply $U_{CF}(E,N)$ with $CF = 25\%$
- In our example, we go through the three classification cases and obtain $U_{25\%}(0{,}9) = 0.143$, $U_{25\%}(0{,}1) = 0.750$, and $U_{25\%}(0{,}6) = 0.206$
- Hence, based on these assumptions, the total number of predicted errors of the considered subtree amounts to $9 \cdot 0.143 + 6 \cdot 0.206 + 1 \cdot 0.750 = 3.273$
- If we replace this subtree by a leaf with the classification “democrat”, we obtain for the training data 15 correct classifications and 1 erroneous classification
- Hence, we obtain $U_{25\%}(1{,}16) = 0.157$ and $16 \cdot U_{25\%}(1{,}16) = 2.512$ predicted errors of this leaf
- Consequently, C4.5 prunes the subtree to a leaf

Windowing

- Windowing is a technique in decision tree generation that processes a large data set by separating it into smaller pieces that are iteratively processed while the successively constructed tree is adapted in each step
- Specifically, typical steps are
  - Start with a subset of items (denoted as the *window*) and generate the corresponding decision tree
  - Subsequently, this tree is used to classify (one by one) the remaining items
  - Usually, some of these remaining items will now be misclassified (exceptions)
  - Hence, add a selection of these exceptions to the initial window and update the decision tree that is in turn tested with the remaining cases
  - This cycle is repeated until all cases are classified correctly (if the entire data set is consistent)

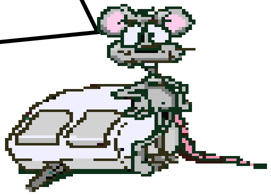
Windowing

- It is quite common that the window ended up containing only a small fraction of the training cases
- This final window represents a screened set of training cases that comprises the “interesting ones” together with a sufficiently large variety of “ordinary cases”
- Note that rather than picking training cases randomly to form the initial window, C4.5 biases the choice so that the distribution of classes in the initial window is as uniform as possible
- Moreover, the process of adding exceptions is controlled
  - While ID3 strongly limits the number of exceptions to be added, C4.5 always adds at least the half of these cases in each iteration, thereby attempting to speed convergence on a final tree
  - C4.5 may also stop before the tree correctly classifies all cases outside the window if it appears that the generated trees do not become more accurate
  - For domains in which classification is not correctly possible due to noise or indeterminacy, early termination is meant to prevent the growth of the window until it includes almost all the training cases

Windowing – A modern technique?

Windowing was necessary in former times due to strong memory limitations. But today?

There are still some interesting reasons to be considered. Therefore, it may be useful to retain windowing.



Why retain windowing?

- Frankly speaking, windowing was invented due to limited memory resources in former times
- Therefore, it was necessary to analyze larger data sets with significant numbers of attributes and items
- But today?
- Even inexpensive computers run processes with significant memory consumptions (larger than the physical memory)
- Since windowing was introduced to overcome memory limits that no longer pose any problems, its retention in the system needs some justification

Reasons for windowing

- Faster construction of trees for useful data sets
  - If the data set is consistent and free of noise and indeterminism, windowing can quickly converge on a final tree and so lead to computational speedup
  - Quinland (1993) reports a speedup of 15% for a collection of 8,124 mushroom descriptions, each classified as poisonous or edible. When this data is used with the default windowing parameters, the initial window gives a tree that correctly classifies all the other cases, so the final tree is arrived at on the first cycle
  - However, note that the opposite is true for real-world data with unreliable data sets. Here, slow downs are observed

Reasons for windowing

- More accurate or specialized trees
  - Empirical studies report that some samples can help for dealing with continuous variables and finding better thresholds
  - The generation of various decision trees by using different starting samples provides the basis for new features, namely
    - Growing several alternative trees and selecting as "the" tree the one with the lowest predicted error rate
    - Growing several trees, generating production rules from all of them, then constructing a single production rule classifier from all the available rules
  - However, in these positive cases, the downside is almost always a considerably higher computational effort

Example – The pathological multiplexer

- In the pathological multiplexor task, a case is described by a series of bits
  - The first a bits constitute an address (from 0 to $2^a - 1$)
  - Then, there are 2^a data bits $d_1, \dots, d_{2^a}$ while the binary classification in yes or no is determined by the $a - 1$th data bit d_{a-1}
  - Let a=3: An address is defined by $a_0 a_1 a_2$ and we have eight data bits $d_0 d_1 d_2 d_3 d_4 d_5 d_6 d_7$. Thus, $d_a = d_2$ gives the classification
  - For instance, 01001101001 belongs to the class 1, i.e., yes, due to $d_2 = 1$
- For this 11-bit multiplexor, five training set sizes 100, 200, ... , 500 were chosen and five training sets of each size generated randomly (with replacement)
- A large set of 1,000 test cases was also generated randomly and used for evaluation issues (these case were unseen)

Results of the pathological multiplexer

Training cases	No windowing	Single tree windowing	Ten trees windowing
100	35.4 %	36.0 %	34.4 %
200	24.4 %	24.6 %	16.9 %
300	18.5 %	13.9 %	11.6 %
400	17.9 %	9.4 %	5.7 %
500	13.2 %	8.0 %	6.3 %

- The table provides the measured error rates
- The higher accuracy on unseen cases was achieved at additional running time
  - Developing a single tree by windowing takes almost twice as long as generating a tree from all the training cases in one pass
  - And producing ten trees takes ten times as long

2.2.3 CART

- CART stands for “classification and regression trees”
- It is a further well-known and widely used algorithm for decision tree generation
- It was originated by Breimann, Friedman, Olshen, and Stone (1984)
- The procedure generates **only binary decision trees** and conducts the choice of the separation steps by applying the **gini impurity measure**
- Although the binary tree attribute seems to be fairly restrictive, it is not a real limitation
- Note that for each occurring value a_l of some attribute A_j (the jth attribute) we can check every item x in a binary way, i.e., whether it holds that $x_j \leq a_l$ or not
- Thus, possible branches are combinations of attributes and value sets that are combined for the left branch and for the right branch, respectively

Gini impurity

- Given a data set M comprising m items with values for n attributes $A = \{A_1, \dots, A_n\}$ and a classification $C = \{C_1, \dots, C_k\}$ into k classes
- The attributes and their values are transformed into o suitable branching candidates $B = \{b_1, b_2, \dots, b_o\}$
- The **impurity** suggests the aim of the decision tree to have each leaf representing only items of the same class (causing no classification error)
- The impurity measure is mathematically defined as follows
  - $N(i, b_l, r/l)$: Fraction of the number of items of a considered data set belonging to the right/left branch of b_l possessing the classification $C_i \in C$
  - $\#(b_l, r/l)$: Number of items (of the m ones) belonging to the right/left branch of b_l
  - The **Gini impurity of the branching candidate** $b_l \in B$ is defined by $G(b_l) =$

$$\frac{\#(b_l, l)}{m} \cdot \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} N(i, b_l, l) \cdot N(j, b_l, l) + \frac{\#(b_l, r)}{m} \cdot \sum_{i=1}^{k} \sum_{j=1, j \neq i}^{k} N(i, b_l, r) \cdot N(j, b_l, r)$$

$$= \frac{\#(b_l, l)}{m} \cdot \sum_{i=1}^{k} N(i, b_l, l) \cdot \sum_{j=1, j \neq i}^{k} N(j, b_l, l) + \frac{\#(b_l, r)}{m} \cdot \sum_{i=1}^{k} N(i, b_l, r) \cdot \sum_{j=1, j \neq i}^{k} N(j, b_l, r)$$

Gini impurity – Simplifying the formula

$$= \frac{\#(b_l, l)}{m} \cdot \sum_{i=1}^{k} N(i, b_l, l) \cdot \big(1 - N(i, b_l, l)\big) + \frac{\#(b_l, r)}{m} \cdot \sum_{i=1}^{k} N(i, b_l, r) \cdot \big(1 - N(i, b_l, r)\big)$$

$$= \frac{\#(b_l, l)}{m} \cdot \sum_{i=1}^{k} (N(i, b_l, l) - N(i, b_l, l)^2) + \frac{\#(b_l, r)}{m} \cdot \sum_{i=1}^{k} (N(i, b_l, r) - N(i, b_l, r)^2)$$

$$= \frac{\#(b_l, l)}{m} \cdot \left(1 - \sum_{i=1}^{k} N(i, b_l, l)^2\right) + \frac{\#(b_l, r)}{m} \cdot \left(1 - \sum_{i=1}^{k} N(i, b_l, r)^2\right)$$

CART – A simple example

- We consider the following data set

Item	1	2	3	4	5	6	7	8	9	10
A_1	0.6	1.8	0.7	0.2	1.1	2.9	2.5	2.2	2.8	2.5
A_2	0.2	0.3	1.1	1.3	2.3	0.4	1.2	1.6	2.0	3.0
Class?	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes

- The classification is binary ($k = 2$), i.e., just yes or no
- By testing all thresholds for the two attributes, we obtain 20 theoretical branching alternatives
- However, we focus here on the most promising one and an alternative one for comparison reasons
- First, we consider for attribute A_2 the threshold 1.15 (say b_1)
- Hence, the left branch contains the items 1,2,3,and 6. The right 4,5,7,8,9, and 10

Branch alternative b_1 – Gini impurity

- Left branch: $N(no, b_1, l) = \frac{3}{4}$, $N(yes, b_1, l) = \frac{1}{4}$, and $\frac{\#(b_1,l)}{m} = \frac{4}{10}$
- Right branch: $N(no, b_1, r) = \frac{2}{6}$, $N(yes, b_1, r) = \frac{4}{6}$, and $\frac{\#(b_1,r)}{m} = \frac{6}{10}$
- Hence, we obtain

$$G(b_1) = \frac{\#(b_1,l)}{m} \cdot \left(1 - \sum_{i=1}^{k} N(i, b_1, l)^2\right) + \frac{\#(b_1,r)}{m} \cdot \left(1 - \sum_{i=1}^{k} N(i, b_1, r)^2\right)$$

$$\frac{4}{10} \cdot \left(1 - \frac{10}{16}\right) + \frac{6}{10} \cdot \left(1 - \frac{20}{36}\right) = \frac{2}{5} \cdot \left(\frac{3}{8}\right) + \frac{6}{10} \cdot \left(\frac{4}{9}\right)$$

$$= \frac{3}{20} + \frac{4}{15} = \frac{9}{60} + \frac{16}{60} = \frac{25}{60} = \frac{5}{12} = 0.41\overline{6}.$$

Branch alternative b_2 – Gini impurity

- Second, we consider for attribute A_1 the threshold 2.0 (say b_2)
- Hence, the left branch contains the items 1,2,3,4, and 5. The right 6,7,8,9, and 10
- Left branch: $N(no, b_2, l) = \frac{5}{5} = 1$, $N(yes, b_2, l) = \frac{0}{5} = 0$, and $\frac{\#(b_2,l)}{m} = \frac{5}{10}$
- Right branch: $N(no, b_2, r) = \frac{0}{5} = 0$, $N(yes, b_2, r) = \frac{5}{5} = 1$, and $\frac{\#(b_2,r)}{m} = \frac{5}{10}$
- Hence, we obtain

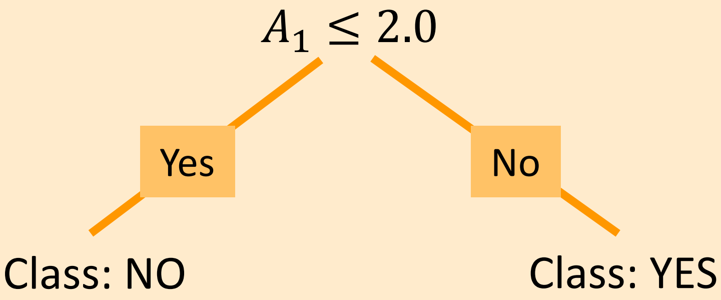
$$G(b_1) = \frac{\#(b_2,l)}{m} \cdot \left(1 - \sum_{i=1}^{k} N(i, b_2, l)^2\right) + \frac{\#(b_2,r)}{m} \cdot \left(1 - \sum_{i=1}^{k} N(i, b_2, r)^2\right)$$

$$\frac{5}{10} \cdot (1-1) + \frac{5}{10} \cdot (1-1) = \frac{5}{10} \cdot (0) + \frac{5}{10} \cdot (0) = 0 + 0 = 0$$

Branching alternative b_2 is chosen

- The branching alternative b_2 provides the perfect value 0.0 as there is no impurity left and we have found the decision tree for the training data

Resulting decision tree



2.3 Statistical methods for data analysis

- In what follows, we consider some basic statistical methods that are quite useful for data analysis
- The following depictions are based on Section 8.1 of the book of Ertel (2016)
- Analogously, in what follows, we use, only for illustration purposes, some data available from the so-called LEXMED project
  - LEXMED is a machine learning system for diagnostic appendicitis purposes
  - It is an expert system that uses reasoning with probabilities and maximum entropy
- However, at this point, we only consider a generated data set of $N = 473$ patients with collected data for 15 attributes and a respectively derived diagnosis (see next slide)
- Hence, each patient is defined by a vector x with the respective 16 values

Data from Ertel (2016)

Var.-Nr.	Description	Values of domain
1	Age	Continuous
2	Sex (1 D male, 2 D female)	1, 2
3	Dolor Quadrant 1	0, 1
4	Dolor Quadrant 2	0, 1
5	Dolor Quadrant 3	0, 1
6	Dolor Quadrant 4	0, 1
7	Muscular defense (local)	0, 1
8	Muscular defense (general)	0, 1
9	Dolor during leaving hold of	0, 1
10	Agitation	0, 1
11	Dolor during rectal examination	0, 1
12	Temperature axial	Continuous
13	Temperature rectal	Continuous
14	Leucocytes	Continuous
15	Diabetes mellitus	0, 1
16	Diagnosis: appendicitis (yes or no)	0, 1

Applying basic statistical values

- Mean

$$\bar{x}_i = \frac{1}{N} \cdot \sum_{p=1}^{N} x_i^p$$

- The standard deviation gives the average deviation from the mean

$$\sigma_i = \sqrt{\frac{1}{N} \cdot \sum_{p=1}^{N} \left(x_i^p - \bar{x}_i\right)^2}$$

- The covariance gives us information concerning a possible correlation of two attribute values over all patients, i.e., it holds that

$$\sigma_{i,j} = \frac{1}{N} \cdot \sum_{p=1}^{N} \left(x_i^p - \bar{x}_i\right) \cdot \left(x_j^p - \bar{x}_j\right)$$

Clearly, if for two attributes, patients have related values, this leads to positive contributions as both products are positive or negative.

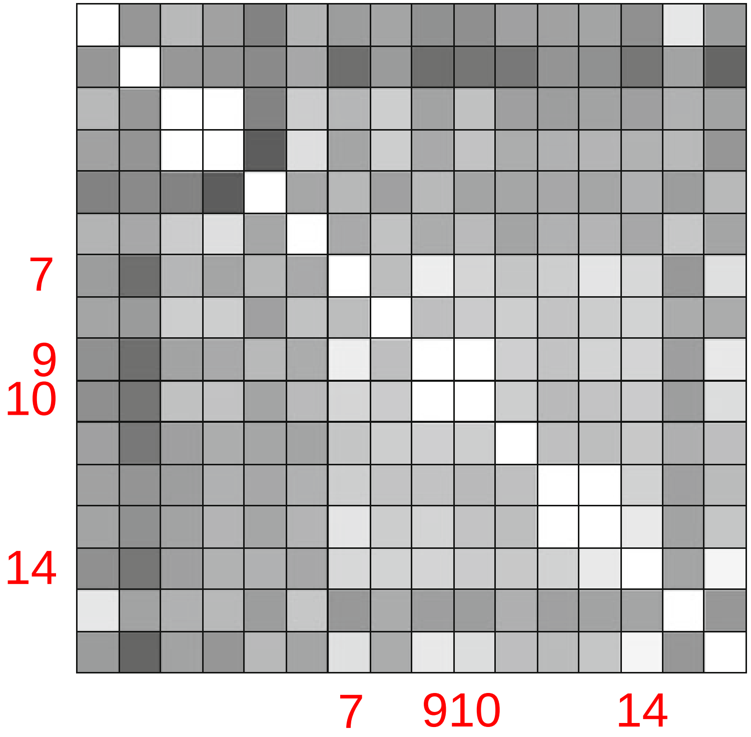
Applying basic statistical values

- However, as the covariance is significantly triggered by the absolute values of the respective variables, we normalize the covariance by the product of standard deviations. Hence we get Pearson's correlation coefficient

$$\rho_{i,j} = \frac{\sigma_{i,j}}{\sigma_i \cdot \sigma_j} = \frac{1}{N \cdot \sigma_i \cdot \sigma_j} \cdot \sum_{p=1}^{N} \left(x_i^p - \bar{x}_i\right) \cdot \left(x_j^p - \bar{x}_j\right)$$

- For the example, we can generate the matrix of all Pearson’s correlation coefficients for the 16 variables
- These correlations can be best illustrated by graphical density profiles as done on the next slide

Density profiles of the correlation coefficients



= −1: is black

= 1: is white

See Ertel (2016) p.199

- By considering the density profiles (detailed values can be found in Ertel (2016) p.199), it becomes obvious that the attributes 7,9,10, and 14 possess the strongest correlation (0.33, 0.38, 0.32, and 0.44) with the sought classification (attribute 16)
- However, the attributes 9 and 10 are also highly correlated (0.53). Therefore, one of the two values may be sufficient

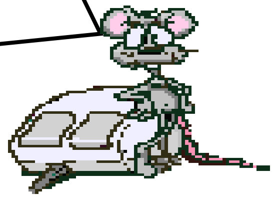
2.4 The perceptron – A linear classifier

- Again, we consider a data base with N items that are characterized by n attribute values and have to be **clustered into two classes**, i.e., a binary classification is sought
- To be able to **separate a set of N items** that are given as vectors x in the $\mathbb{R}^n$ with an additional classification bit $y \in \{0,1\}$ in a **linear way** means that we are able to define a **hyperplane** that divides the considered vector space $\mathbb{R}^n$ into two **half spaces** such that **all items with a classification 0** are in the one half space while **all other items (with classification 1)** are in the other half space
- If this is the case, the hyperplane provides us with a fast computable method that decides to which class a vector belongs

Hyperplanes, half spaces are …?

Please stop! What the hell are hyperplanes and half spaces? I am happy to know what the $\mathbb{R}^n$ is.

No problem.
Both terms come from Linear Algebra and are really important.
Particularly in Operations Research.



Hyperplanes

2.4.1 **Definition**

Let $a \in \mathbb{R}^n \setminus \{0\}$ and $\alpha \in \mathbb{R}$. Then, the vectors of the $\mathbb{R}^n$ that belong to the set

$$H = \{ x \in \mathbb{R}^n \mid a \cdot x = \alpha \}$$

constitute a **hyperplane**

Observations

- Due to the linear restriction to be fulfilled, such a hyperplane in an n-dimensional space has the dimension $n - 1$
- A hyperplane defines two separated half spaces, i.e., it divides the space into two parts

Half spaces

2.4.2 **Definition**

Given the hyperplane $H = \{\, x \in \mathbb{R}^n \mid a \cdot x = \alpha \,\}$ as defined in Definition 2.4.1.

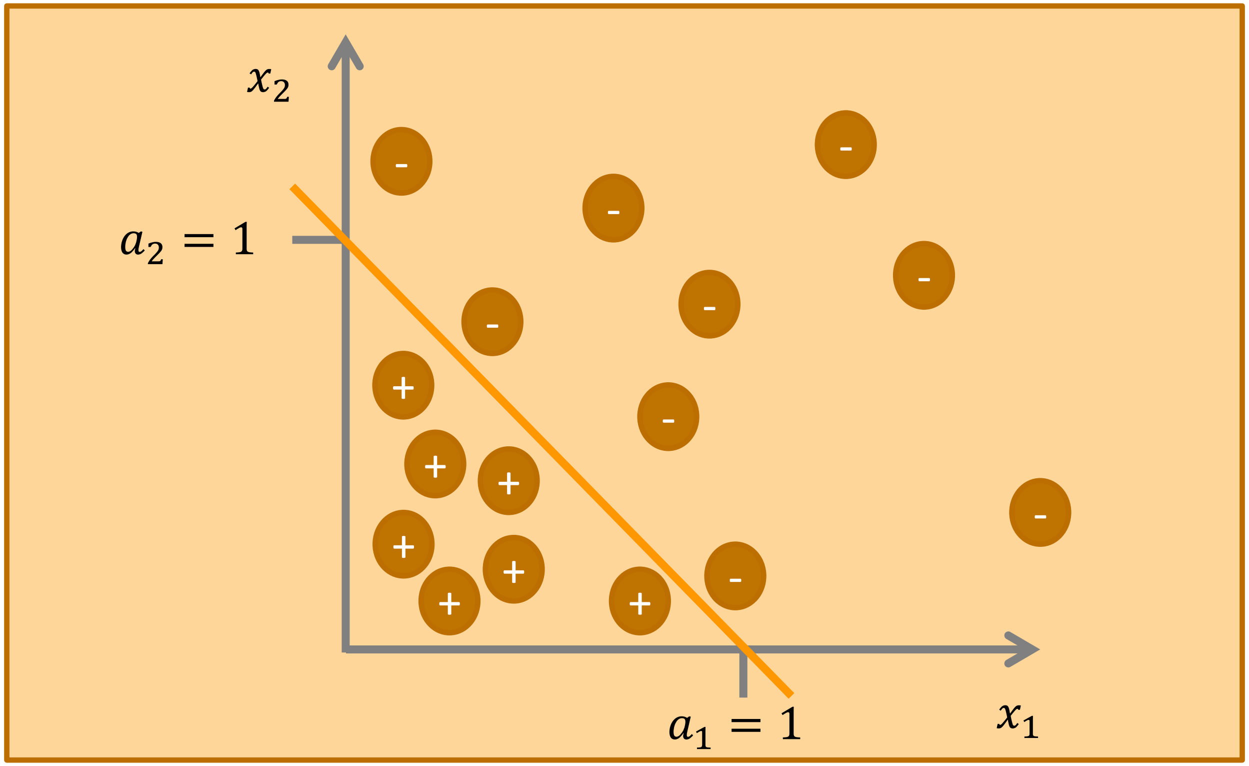
Then, this hyperplane determines the following two half spaces in the $\mathbb{R}^n$:

First half space: $H_1^{\geq} = \{\, x \in \mathbb{R}^n \mid a \cdot x \geq \alpha \,\}$

Second half space: $H_2^{\geq} = \{\, x \in \mathbb{R}^n \mid -a \cdot x \geq -\alpha \,\}$

Illustration

- We illustrate the linear separation for the two-dimensional space
- Here, the hyperplane $H = \left\{ x \in \mathbb{R}^n \mid \binom{1}{1} \cdot x = 1 \right\}$ separates the two-dimensional space $\mathbb{R}^2$



Linear separability

Hence, we can now define what we understand under a linearly separable data set:

2.4.3 **Definition**

Two sets $M_{-}, M_{+} \subseteq \mathbb{R}^{n}$ are denoted as **linearly separable** if and only if there exist $n+1$ real numbers $a_1, a_2, \dots, a_n \in \mathbb{R}$ and $\theta \in \mathbb{R}$ such that it holds that $\sum_{i=1}^{n} a_i \cdot x_i \geq \theta, \forall x \in M_{+}$ and $\sum_{i=1}^{n} a_i \cdot x_i < \theta, \forall x \in M_{-}$. The parameter θ is denoted as the **threshold value**.

Example – The AND function

- We are looking for the weights and threshold needed to implement the AND function for the $\{0,1\}^2$ with a perceptron
- Thus, the mapped items are the following

$x_0 = \binom{0}{0}, x_1 = \binom{0}{1}, x_2 = \binom{1}{0}, x_3 = \binom{1}{1}$, with $M_- = \{x_0, x_1, x_2\}$ and $M_+ = \{x_3\}$

The sets M_- and M_+ are linearly separable by using the hyperplane $H = \left\{ x \in \mathbb{R}^n \mid \binom{1}{1} \cdot x = \theta = 2 \right\}$ as we have for $x_0 = \binom{0}{0}$: $\binom{1}{1} \cdot \binom{0}{0} = 0 < \theta = 2$, for $x_1 = \binom{0}{1}$: $\binom{1}{1} \cdot \binom{0}{1} = 1 < \theta = 2$, and for $x_2 = \binom{1}{0}$: $\binom{1}{1} \cdot \binom{1}{0} = 1 < \theta = 2$, while for $x_3 = \binom{1}{1}$: $\binom{1}{1} \cdot \binom{1}{1} = 2 \geq \theta$ as claimed by Definition 2.4.3.

Absolute linear separability

Analogous to the linear separability, we introduce the absolute linear separability. Here, we demand in both cases non-equality, i.e., no item belongs to the separating hyperplane

2.4.4 **Definition**

Two sets $M_-, M_+ \subseteq \mathbb{R}^n$ are denoted as **absolutely linearly separable** if and only if there exist $n+1$ real numbers $a_1, a_2, \dots, a_n \in \mathbb{R}$ and $\theta \in \mathbb{R}$ such that it holds that $\sum_{i=1}^{n} a_i \cdot x_i > \theta, \forall x \in M_+$ and $\sum_{i=1}^{n} a_i \cdot x_i < \theta, \forall x \in M_-$.

Coming back to the AND function

The sets M_- and M_+ are absolutely linearly separable by using the hyperplane

$H = \left\{ x \in \mathbb{R}^n \;\middle|\; \binom{1}{1} \cdot x = \theta = 1.5 \right\}$ as we have for $x_0 = \binom{0}{0}$: $\binom{1}{1} \cdot \binom{0}{0} = 0 < \theta$, for $x_1 = \binom{0}{1}$: $\binom{1}{1} \cdot \binom{0}{1} = 1 < \theta$, and for $x_2 = \binom{1}{0}$: $\binom{1}{1} \cdot \binom{1}{0} = 1 < \theta$, while for $x_3 = \binom{1}{1}$: $\binom{1}{1} \cdot \binom{1}{1} = 2 > \theta$ as claimed by Definition 2.4.4.

Consequence

2.4.5 **Lemma**

Two finite sets of points, M_- and M_+, in n-dimensional space are linearly separable if and only if there are also absolutely linearly separable. Hence, linear separability and absolute linear separability are equivalent.

Proof of Lemma 2.4.5

- We assume that M_- and M_+ are linearly separable
- Hence, there exist weights $w_1, \dots, w_n, w_{n+1}$ such that it holds

  $\sum_{i=1}^{n} w_i \cdot x_i \geq w_{n+1}, \forall x \in M_+$ and $\sum_{i=1}^{n} w_i \cdot x_i < w_{n+1}, \forall x \in M_-$
- Let $\epsilon = max\{\sum_{i=1}^{n} w_i \cdot x_i - w_{n+1} \mid x \in M_-\}$
- Then, we have $\epsilon < \frac{\epsilon}{2} < 0$
- Moreover, we set $v = w_{n+1} + \frac{\epsilon}{2}$. Then, for all points $x \in M_+$, it holds that $\sum_{i=1}^{n} w_i \cdot x_i \geq w_{n+1}$ and therefore, by setting $w_{n+1} = v - \frac{\epsilon}{2}$, we obtain $\sum_{i=1}^{n} w_i \cdot x_i \geq v - \frac{\epsilon}{2}$. This implies $\sum_{i=1}^{n} w_i \cdot x_i - \left(v - \frac{\epsilon}{2}\right) \geq 0$
- Therefore, we conclude that $\sum_{i=1}^{n} w_i \cdot x_i - v \geq -\frac{\epsilon}{2} > 0$.
- This implies $\sum_{i=1}^{n} w_i \cdot x_i > v$

Proof of Lemma 2.4.5

- Analogously, we consider $x \in M_-$
- Since $\epsilon = max\{\sum_{i=1}^{n} w_i \cdot x_i - w_{n+1} \mid x \in M_-\}$, we conclude that $\sum_{i=1}^{n} w_i \cdot x_i - w_{n+1} \le \epsilon$
- By using $w_{n+1} = v - \frac{\epsilon}{2}$, it holds that $\sum_{i=1}^{n} w_i \cdot x_i - \left(v - \frac{\epsilon}{2}\right) \le \epsilon$
- Thus, we deduce $\sum_{i=1}^{n} w_i \cdot x_i - v \le \frac{\epsilon}{2} < 0$
- Therefore, it holds that $\sum_{i=1}^{n} w_i \cdot x_i < v$
- Hence, M_+ and M_- are absolutely linearly separable
- Clearly, the inverted direction is trivial, i.e., if M_+ and M_- are absolutely linearly separable, they are, by definition, also linearly separable

Perceptron

2.4.6 **Definition**

Given a weight vector $\omega = (w_1, w_2, \dots, w_n) \in \mathbb{R}^n$ and $x \in \mathbb{R}^n$ an input vector.

A **perceptron** is a mapping $P{:}\, \mathbb{R}^n \mapsto \{0,1\}$ such that

$$P_\omega(x) = \begin{cases} 1 & if \omega \cdot x = \sum_{i=1}^{n} w_i \cdot x_i > 0 \\ 0 & otherwise \end{cases}$$

In order to additionally consider the threshold value θ, both vectors are extended by adding $w_{n+1} = -\theta$ and $x_{n+1} = 1$, respectively.

Then, we obtain $\omega \cdot x - \theta \cdot 1 = -\theta + \sum_{i=1}^{n} w_i \cdot x_i$ and if $P_\omega(x) = 1$ holds, we have $\sum_{i=1}^{n} w_i \cdot x_i > \theta$

Perceptron training

- In what follows, we generate a training procedure that iteratively generates a perceptron for a given data set
- It should absolutely separate the two sets $M_-, M_+ \subseteq \mathbb{R}^n$

Perceptron learning algorithm

```
**Input**: $M_+, M_- \subseteq \mathbb{R}^n$ as sets of items with positive (“1”) and negative (“0”) classification
Set $\omega \in \mathbb{R}^n$ arbitrarily such that $\omega \neq 0$ holds
Set $t := 0$ /* Counter of conducted updates */
REPEAT
    FOR all $x \in M_+$
        IF $\omega \cdot x \leq 0$ THEN $\omega := \omega + x; t := t + 1;$
    END FOR
    FOR all $x \in M_-$
        IF $\omega \cdot x \geq 0$ THEN $\omega := \omega - x; t := t + 1;$
    END FOR
UNTIL all $x \in M_+ \cup M_-$ are classified correctly by $P_\omega(x)$
```

Convergence

2.4.7 **Theorem**

We consider the perceptron learning algorithm and assume that the sets $M_+, M_- \subseteq \mathbb{R}^n$ are finite and linearly separable. Then, the perceptron learning algorithm updates the weight vector $\omega \in \mathbb{R}^n$ a finite number of times, i.e., the algorithm will terminate with a perceptron that separates the elements of the two sets $M_+, M_- \subseteq \mathbb{R}^n$.

Proof of Theorem 2.4.7

- We give the proof that can be found in Rojas (1996)
- First of all, we make three simplifications without losing generality
  1. The sets $M_+, M_- \subseteq \mathbb{R}^n$ can be joined together in a single set named M. In order to enable an equal treatment of all vectors in M, we negate all vectors $M_- \subseteq \mathbb{R}^n$
  2. Subsequently, we normalize all vectors of M. This does not change the decision criterion of the algorithm (and therefore the termination) as if we have $\omega \cdot x < 0$ for some $x \in M$, this also applies after multiplying a scalar η, i.e., this also applies to $\eta \cdot x$. The same is true for $\omega \cdot x > 0$, $\forall x \in M$. Thus, in what follows all vectors in M are normalized, i.e., $\|x\| = 1, \forall x \in M$
  3. The weight vector can be also normalized. As we assume that the considered problem is linearly separable there exists such a solution vector $w^o \in \mathbb{R}^n$ that we normalize and obtain $w^* \in \mathbb{R}^n$ fulfilling $\|w^*\| = 1$

Proof of Theorem 2.4.7

- We consider the step of the algorithm that updates the weight vector and this was the $t + 1$th update. The result is the weight vector w_{t+1} that was built by setting $w_{t+1} \coloneqq w_t + x$
- The reason for this update is that there is a vector $x \in M$ with $w_t \cdot x \leq 0$
- We start with the following trigonometric result

  For two vectors $A, B \in \mathbb{R}^n$ and the angle ρ between them, it holds that $cos(\rho) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$
- Hence, we compare the current (just updated) weight vector w_{t+1} and the normalized solution vector w^*
- It holds that (note that $\|w^*\| = 1$)

$$cos(\rho) = \frac{w_{t+1} \cdot w^*}{\|w_{t+1}\| \cdot \|w^*\|} = \frac{(w_t + x\,) \cdot w^*}{\|w_{t+1}\|} = \frac{w_t \cdot w^* + x \cdot w^*}{\|w_{t+1}\|}$$

- We compute $\delta = min\{\, w^* \cdot \tilde{x} \mid \tilde{x} \in M\,\} > 0$ and obtain

$$cos(\rho) = \frac{w_t \cdot w^* + x \cdot w^*}{\|w_{t+1}\|} \geq \frac{w_t \cdot w^* + \delta}{\|w_{t+1}\|}$$

Proof of Theorem 2.4.7

- Note that $\delta = min\{ w^* \cdot \tilde{x} \mid \tilde{x} \in M \} > 0$ due to the fact that w^* is a solution and therefore correctly separates all elements of set M
- By induction of all updating steps, we obtain for the initial weight vector w_0

$$cos(\rho) \geq \frac{w_0 \cdot w^* + (t+1) \cdot \delta}{\|w_{t+1}\|} \quad (*)$$

- Furthermore, after modifying the numerator, we consider the denominator. It holds that

$$\|w_{t+1}\|^2 = (w_t + x) \cdot (w_t + x) = \|w_t\|^2 + 2w_t \cdot x + \|x\|^2$$

- Since, by assumption, $w_t \cdot x \leq 0$, we have $2w_t \cdot x \leq 0$ and therefore

$$\|w_{t+1}\|^2 = \|w_t\|^2 + 2w_t \cdot x + \|x\|^2 \leq \|w_t\|^2 + \|x\|^2$$

- Moreover, by assumption, all vectors $x \in M$ are normalized and we obtain

$$\|w_{t+1}\|^2 \leq \|w_t\|^2 + 1$$

- Again, we get by induction of the all updating steps

$$\|w_{t+1}\|^2 \leq \|w_0\|^2 + t + 1 \Longrightarrow \|w_{t+1}\| \leq \sqrt{\|w_0\|^2 + t + 1} \quad (**)$$

Proof of Theorem 2.4.7

- By substituting $(**)$ in $(*)$, we derive

$$cos(\rho) \geq \frac{w_0 \cdot w^* + (t+1) \cdot \delta}{\|w_{t+1}\|} \geq \frac{w_0 \cdot w^* + (t+1) \cdot \delta}{\sqrt{\|w_0\|^2 + t + 1}}$$

- The right hand side comprises the values $\delta, w_0 \cdot w^*$, and $\|w_0\|^2$ that are **constant and positive** during the computation steps for a given data set
- However, the number of conducted steps increases and the **right hand side grows proportionally to $\sqrt{t}$**
- But, as the **left hand side is upper bounded by 1**, t is bounded by a maximum value
- This proves the termination and therefore convergence of the perceptron learning algorithm
- Clearly, the proof underlines that the algorithm works by bringing the initial vector w_0 iteratively sufficiently close to the solution vector w^* as $\cos(\rho)$ becomes larger due to a proportionally smaller angle ρ

Example – The OR function

$x_0 = \begin{pmatrix}0\\0\end{pmatrix}, x_1 = \begin{pmatrix}0\\1\end{pmatrix}, x_2 = \begin{pmatrix}1\\0\end{pmatrix}, x_3 = \begin{pmatrix}1\\1\end{pmatrix}$, with $M_- = \{x_0\}$ and $M_+ = \{x_1, x_2, x_3\}$

In order to compute a linear separator, we have to extend the vectors to

$x_0 = \begin{pmatrix}0\\0\\1\end{pmatrix}, x_1 = \begin{pmatrix}0\\1\\1\end{pmatrix}, x_2 = \begin{pmatrix}1\\0\\1\end{pmatrix}$, and $x_3 = \begin{pmatrix}1\\1\\1\end{pmatrix}$ and start the calculation with

$w_0 = \begin{pmatrix}0\\0\\0\end{pmatrix}$. Then, we obtain:

1. $w_0 \cdot x_0 = 0$: This is not correctly classified and we update to $w_1 := w_0 - x_0 = \begin{pmatrix}0\\0\\0\end{pmatrix} - \begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\-1\end{pmatrix}$

2. $w_1 \cdot x_1 = -1$: This is not correctly classified and we update to $w_2 := w_1 + x_1 = \begin{pmatrix}0\\0\\-1\end{pmatrix} + \begin{pmatrix}0\\1\\1\end{pmatrix} = \begin{pmatrix}0\\1\\0\end{pmatrix}$

Example – The OR function

3. $w_2 \cdot x_2 = 0$: This is not correctly classified and we update to $w_3 \coloneqq w_2 + x_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$
4. $w_3 \cdot x_3 = 1$: This is correctly classified and we have no update
5. $w_3 \cdot x_0 = 1$: This is not correctly classified and we update to $w_4 \coloneqq w_4 - x_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$
6. $w_4 \cdot x_1 = 1$: This is correctly classified and we have no update
7. $w_4 \cdot x_2 = 1$: This is correctly classified and we have no update
8. $w_4 \cdot x_3 = 2$: This is correctly classified and we have no update
9. $w_4 \cdot x_0 = 0$: This is not correctly classified and we update to $w_5 \coloneqq w_4 - x_0 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$

Example – The OR function

10. $w_5 \cdot x_1 = 0$: This is not correctly classified and we update to $w_6 \coloneqq w_5 + x_1 = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}$
11. $w_6 \cdot x_2 = 1$: This is correctly classified and we have no update
12. $w_6 \cdot x_3 = 3$: This is correctly classified and we have no update
13. $w_6 \cdot x_0 = 0$: This is not correctly classified and we update to $w_7 \coloneqq w_6 - x_0 = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$
14. $w_7 \cdot x_1 = 1$: This is correctly classified and we have no update
15. $w_7 \cdot x_2 = 0$: This is not correctly classified and we update to $w_8 \coloneqq w_7 + x_2 = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}$
16. $w_8 \cdot x_3 = 4$: This is correctly classified and we have no update

Example – The OR function

17. $w_8 \cdot x_0 = 0$: This is not correctly classified and we update to $w_9 := w_8 - x_0 = \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}$
18. $w_9 \cdot x_1 = 1$: This is correctly classified and we have no update
19. $w_9 \cdot x_2 = 1$: This is correctly classified and we have no update
20. $w_9 \cdot x_3 = 3$: This is correctly classified and we have no update
21. $w_9 \cdot x_0 = -1$: This is correctly classified and we have no update

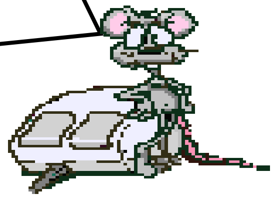
Termination

Output is: $\omega = \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}$

Is everything fine?

OK. It terminates. That is nice. But, is this convergence process fast? The example does not suggest this…

Unfortunately, not always as you see. It depends on some prerequisites. This is considered next.



Accelerating the convergence

- Although the perceptron learning algorithm converges to a solution, the **number of iterations can be very large** if the input vectors are not normalized and are arranged in an unfavorable way
- Note that in an update step the respective vector x gives the weight vector a new direction in order to correct the misclassification of x. But, in order to correct this failure directly it is reasonable to do just this in one step (and not in many) while doing not more than this (causing new failures)
- Therefore, the following modification seems to be reasonable

Improved update handling

- If at iteration t the vector $x \in M_+$ is classified erroneously and we have $w_t \cdot x \leq 0$ the resulting error δ is determined by $\delta = -w_t \cdot x$
- By using a small positive value $\epsilon > 0$, the new weight vector w_{t+1} is calculated as follows

$$w_{t+1} = w_t + \frac{\delta + \epsilon}{\|x\|^2} \cdot x$$

- Note that this corrects the misclassification in one update step as it holds

$$w_{t+1} \cdot x = \left(w_t + \frac{\delta + \epsilon}{\|x\|^2} \cdot x \right) \cdot x = w_t \cdot x + \delta + \epsilon = -\delta + \delta + \epsilon = \epsilon > 0$$

- Hence, the usage of ϵ guarantees that the new weight vector just barely skips over the border of the region with a higher error
- Therefore, ϵ should be made small enough to avoid skipping to another region whose error is higher than the current one
- When $x \in M_-$ holds, the correction step is similarly, but using the factor $\delta - \epsilon$ instead of $\delta + \epsilon$

Observations

- As mentioned in Rojas (1996) it can be stated that the accelerated algorithm is an example of corrective learning
  - The weight vector is not just enforced, but **completely corrects the currently observed error**
  - A **variant of this rule** is correction of the weight vector using a proportionality constant γ as the learning factor in so far that at each update the vector $\gamma \cdot (\delta + \epsilon) \cdot x$ is added to the current weight vector w. In this updating the learning constant falls to zero when learning progresses

Example – The OR function

$x_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, x_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, x_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, x_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, with $M_- = \{x_0\}$ and $M_+ = \{x_1, x_2, x_3\}$

In order to compute a linear separator, we have to extend the vectors to

$x_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, x_1 = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}, x_2 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$, and $x_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ and start the calculation with

$w_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ and set $\epsilon = 0.1$. Then, we obtain with update formula $w_{t+1} = w_t + \frac{\delta \pm \epsilon}{\|x\|^2} \cdot x$ and $\delta = -w_t \cdot x$:

1. $w_0 \cdot x_0 = 0$: This is not correctly classified and we update with $\delta = 0$ to

   $$w_1 := w_0 + \frac{0-0.1}{1} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -0.1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -0.1 \end{pmatrix}$$

2. $w_1 \cdot x_1 = -0.1$: This is not correctly classified and we update with $\delta = 0.1$ to

   $$w_2 := w_1 + \frac{0.1+0.1}{2} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -0.1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.1 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.1 \\ 0 \end{pmatrix}$$

Example – The OR function

3. $w_2 \cdot x_2 = 0$: This is not correctly classified and we update with $\delta = 0$ to

$$w_3 := w_2 + \frac{0+0.1}{2} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0.05 \\ 0.05 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.15 \\ 0.05 \end{pmatrix}$$

4. $w_3 \cdot x_3 = 0.2$: This is correctly classified and we have no update

5. $w_3 \cdot x_0 = 0.05$: This is not correctly classified and we update with $\delta = -0.05$ to

$$w_4 := w_3 + \frac{-0.05-0.1}{1} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.15 \\ 0.05 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -0.15 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.15 \\ -0.1 \end{pmatrix}$$

6. $w_4 \cdot x_1 = 0.05$: This is correctly classified and we have no update

7. $w_4 \cdot x_2 = -0.1$: This is not correctly classified and we update with $\delta = 0.1$ to

$$w_5 := w_4 + \frac{0.1+0.1}{2} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.15 \\ -0.1 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.15 \\ 0 \end{pmatrix}$$

8. $w_5 \cdot x_3 = 0.25$: This is correctly classified and we have no update

9. $w_5 \cdot x_0 = 0$: This is not correctly classified and we update with $\delta = 0$ to

$$w_6 := w_5 + \frac{-0.1}{1} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.15 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -0.1 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.15 \\ -0.1 \end{pmatrix}$$

Example – The OR function

10. $w_6 \cdot x_1 = 0.05$: This is correctly classified and we have no update
11. $w_6 \cdot x_2 = 0$: This is not correctly classified and we update with $\delta = 0$ to $$w_7 := w_6 + \frac{0.1}{2} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.15 \\ -0.1 \end{pmatrix} + \begin{pmatrix} 0.05 \\ 0 \\ 0.05 \end{pmatrix} = \begin{pmatrix} 0.15 \\ 0.15 \\ -0.05 \end{pmatrix}$$
12. $w_7 \cdot x_3 = 0.25$: This is correctly classified and we have no update
13. $w_7 \cdot x_0 = -0.05$: This is correctly classified and we have no update
14. $w_7 \cdot x_1 = 0.1$: This is correctly classified and we have no update
15. $w_7 \cdot x_2 = 0.1$: This is correctly classified and we have no update

Termination

Output is: $\omega = \begin{pmatrix} 0.15 \\ 0.15 \\ -0.05 \end{pmatrix}$

Initial vector

- Aside from the update formula, the convergence speed significantly depends on the initial vector w_0
- Ertel (2016) proposes to use

$$w_0 = \sum_{x_i \in M_+} x_i - \sum_{x_i \in M_-} x_i$$

- It can be observed that this initial vector may accelerate the convergence of the perceptron learning algorithm

Example – The OR function

$x_0 = \binom{0}{0}, x_1 = \binom{0}{1}, x_2 = \binom{1}{0}, x_3 = \binom{1}{1}$, with $M_- = \{x_0\}$ and $M_+ = \{x_1, x_2, x_3\}$

In order to compute a linear separator, we have to extend the vectors to

$x_0 = \begin{pmatrix}0\\0\\1\end{pmatrix}, x_1 = \begin{pmatrix}0\\1\\1\end{pmatrix}, x_2 = \begin{pmatrix}1\\0\\1\end{pmatrix},$ and $x_3 = \begin{pmatrix}1\\1\\1\end{pmatrix}$ and start the calculation with $w_0 = x_1 + x_2 + x_3 - x_0 = \begin{pmatrix}2\\2\\2\end{pmatrix}$ and set $\epsilon = 0.1$. Then, we obtain with update formula $w_{t+1} = w_t + \frac{\delta \pm \epsilon}{\|x\|^2} \cdot x$ and $\delta = -w_t \cdot x$:

1. $w_0 \cdot x_0 = 2$: This is not correctly classified and we update with $\delta = -2$ to $w_1 \coloneqq w_0 + \frac{-2-0.1}{1} \cdot \begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}2\\2\\2\end{pmatrix} + \begin{pmatrix}0\\0\\-2.1\end{pmatrix} = \begin{pmatrix}2\\2\\-0.1\end{pmatrix}$
2. $w_1 \cdot x_1 = 1.9$: This is correctly classified and we have no update
3. $w_1 \cdot x_2 = 1.9$: This is correctly classified and we have no update
4. $w_1 \cdot x_3 = 3.9$: This is correctly classified and we have no update

Example – The OR function

5. $w_1 \cdot x_0 = -0.1$: This is correctly classified and we have no update

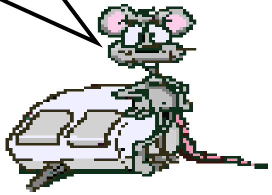
Termination

Output is: $\omega = \begin{pmatrix} 2 \\ 2 \\ -0.1 \end{pmatrix}$

Is everything fine?

Impressive. By solely changing the initial vector, we only need a single update of the weight vector instead of seven as before.

…and it underlines that it may be reasonable to directly integrate the impact of the vectors of both sets M_- and M_+.



What if the learning set is not linearly separable?

- In that case there is no termination possible and we do not obtain a solution
- Moreover, if we stop the computation after an arbitrary step the quality of the generated weight vector is undefined
- Therefore, Gallant (1990) proposed a very simple variant of the perceptron learning algorithm capable of computing a good approximation of an ideal, but not attainable, linear separation
- The main idea of the algorithm is to store the best weight vector found so far by perceptron learning (in a “pocket”) while continuing to update the weight vector itself
- If a better weight vector is found, it supersedes the one currently stored and the algorithm continues to run

The pocket algorithm

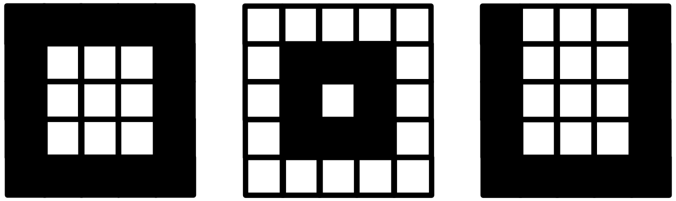
- Initialize the weight vector w randomly
- Set $w_s := w$;
- Set $h_s := 0$;
- Iterate:
  - Update w using a single iteration of the perceptron learning algorithm;
  - Keep track of the number h of consecutively successfully tested vectors.
  - If at any moment $h > h_s$ THEN set $w_s := w$; $h_s := h$;
  - Go to iterate

Observations

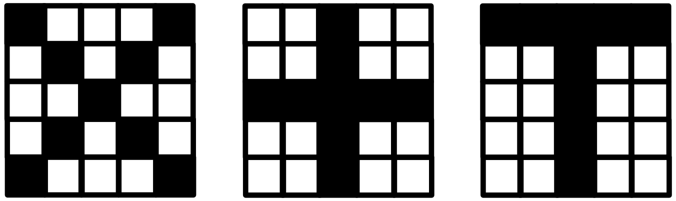
- The algorithm can occasionally change a good stored weight vector for an inferior one, since only information from the last run of selected examples is considered
- The probability of this happening, however, becomes smaller and smaller as the number of iterations grows
- If the training set is finite and the weights and vectors are rational, it can be shown that this algorithm converges to an optimal solution with probability 1 (see Gallant (1990))

Possible application – Pattern recognition

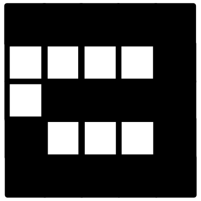
- Ertel (2016) gives the following further example for a possible application of the linear perceptron learning algorithm
- The application deals with pattern recognition of specific letters that may be modified by inverted bits
- Hence, a specific fault tolerance (adaptability) is needed in order to attain a reliable recognition of partly falsely transferred patterns



Patterns of set M_+



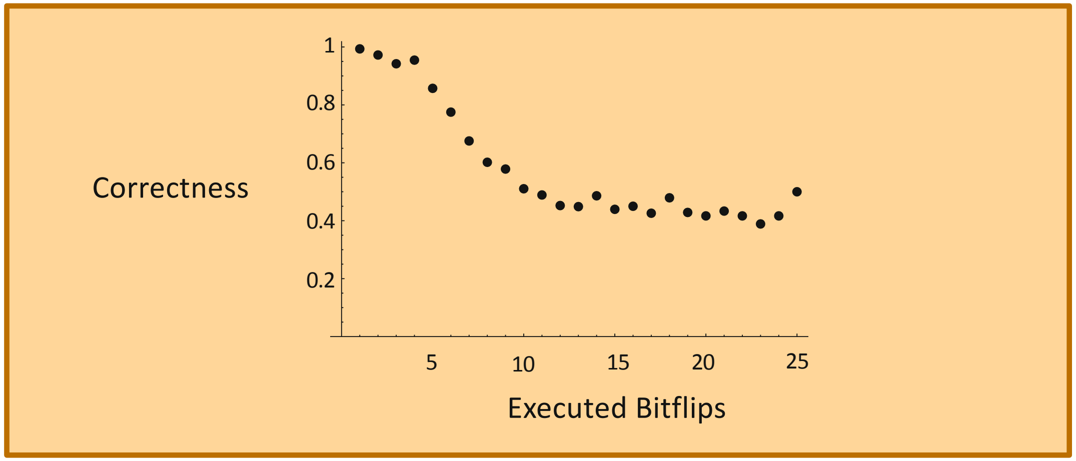
Patterns of set M_-



Candidate to be classified

Attained correctness

- Clearly, the number of correctly classified patterns significantly depends on the number of inverted bits
- The more bits are inverted the lower is the relative correctness of the algorithm
- Specifically, for the considered application, Ertel (2016) reports the following correctness values in dependence of the number of inverted bits:



(See Ertel (2016) p.205)