Data Mining: Foundations

1. Introduction

1.1. What is Data Mining?

Knowledge discovery in databases (KDD) is the process of (semi-)automatic extraction of knowledge from databases, which is:

- Valid
- Previously known
- And potential useful

Remarks:

- (semi-)automatic: different from manual analysis. Often some user interaction is necessary
- Valid: in the statistical sense (e.g. 99.98%)
- Previously unknown: not explicit, no "common sense knowledge" (Pregnant → Female)
- Potentially useful: for a given application

Examples:

- Super Market Basket Analysis (customer Intelligence)
  - Association discovery in gigantic spaces (Recommendations)
  - (Similar) pattern discovery (customer segmentation)
- Discriminative Fragment Mining (Biology)
  - Analysis of highly structured data
- Stock Market Prediction (Finance)
  - Analysis of time series (and incomplete) data
- Credit Card/Customer Churn Analysis (marketing)
  - Prediction of customer behavior

ETL: Extraction, Transformation, Loading

- Getting the data is not always easy:
  - Different resources: flat files, different data bases, excel, spreadsheets, …
  - Integration cumbersome: Missing/not unique IDs, wrong entries, …
  - Sometimes also privacy concerns (not all data in one location)
- Data needs to be transformed:
  - Type conversions
  - Missing value correction/clean-up/ imputation (Anrechnung?)
  - Generation of new values (e.g. convert year of birth into age)
- Three files: customer, products, shopping baskets.
- Over 80% of data miners' time is spent on loading and cleaning data

1.2. Looking at Data

- Familiarize yourself with the data
  - Identify trends
  - Strange patterns
  - Outliers, …
- Types of views:
  - 1D: Histograms
  - 2D: Scatterplots, Scatter Matrix, Multi-Dimensional Scaling
  - 3D: Scatterplots
  - ≥ 3D: Parallel Coordinates
- Visualizations are a good way for first sanity checks
- Interactivity (Brushing) very helpful

1.3. Describing your data

- Simple descriptors, such as:
  - Range
  - Mean /median
  - Standard deviation
  - Nominal values and their frequencies, …

1.4. Finding Patterns

Finding (significant?) patterns in data may reveal interesting connections:

- Global patterns: groups of customers or products
  - Clusters
- Local Patterns: connections between products, sub populations of customers (recommendation engines!)
  - Subgroups
  - Association Rules
- Similarity:
  - Finding the right similarity matric is an art
  - Distance based methods in high dimensions offer all sorts of interesting surprises …

1.5. Finding Models

Deriving models, that describe (aspects of) the data:

- Rules
- Trees
- Typical (or really odd!) examples

Models attempt to describe what is going on in the system, which “generated” the data

1.6. Finding Predictors

Sometimes we want to find a model, which we can use to later predict the target variable(s):

- Predict future shopping behavior
- Predict credit risk
- Predict activity of chemical compound

And we may not care too much about actually understanding the model itself

Brute Force Predictors

- Very simple: look at your closest neighbor
  - Case based reasoning works that way
  - Depends heavily on your distance function
  - Does not work well with outliers/noise
- Slightly better: look at a few of your neighbors
  - K Nearest Neighbor
  - Works pretty well
  - Bet pretty expensive to compute…
- Even better: look at all neighbors, but weight them
  - Weighted K Nearest Neighbor
  - Works even better
  - Even more expensive

But: those are rather good baseline predictors!

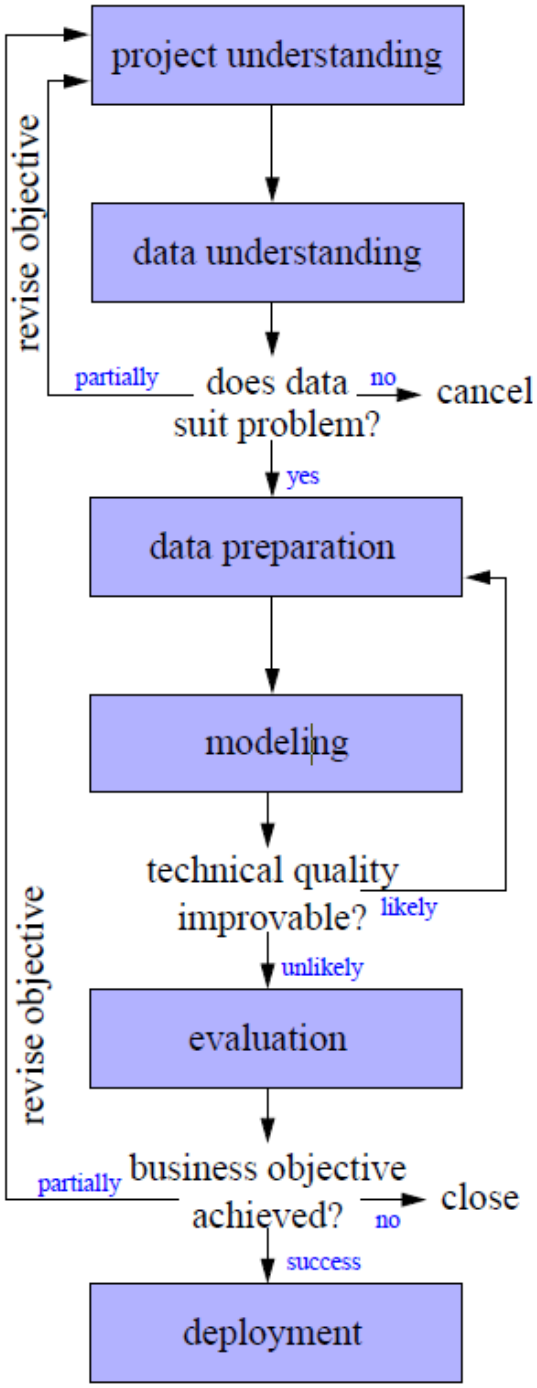
Other Predictors

- Decision Trees, Rules , … (all of our models!)
- (Naïve) Bayes Classifiers
- Regression
- (Artificial) Neural Networks
- Support Vector Machines (Kernel Methods)

1.8. Data Mining Tools

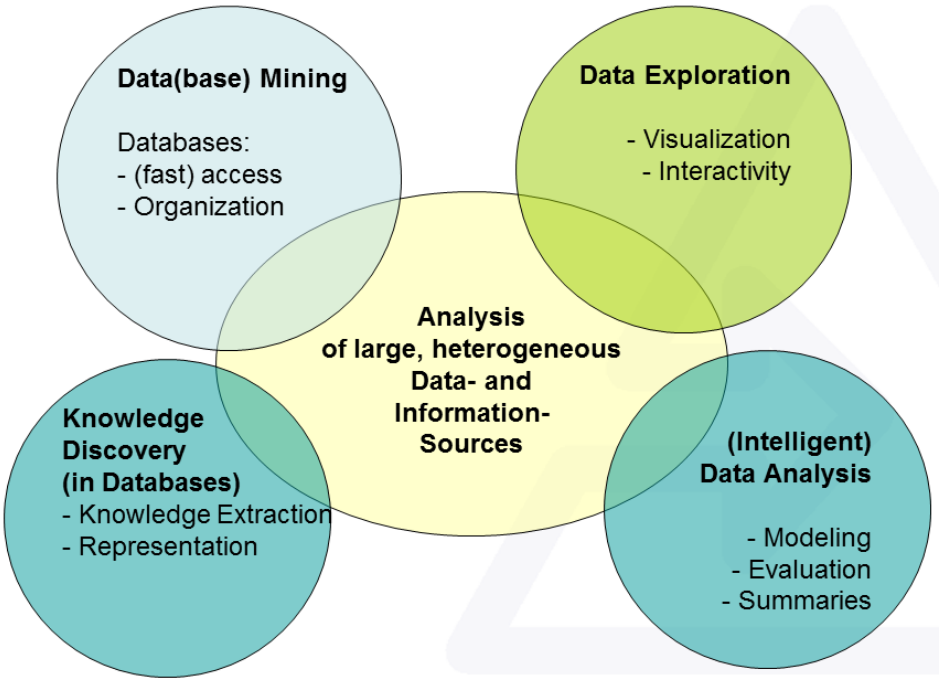
- Programmatic tools
  - Allow code analysis procedure
  - Examples: S (an open source R!)
  - Pro: flexibility (everything can be coded)
  - Con: maintenance (what of your R-Expert leaves?)

1.7. Data Mining Cycle



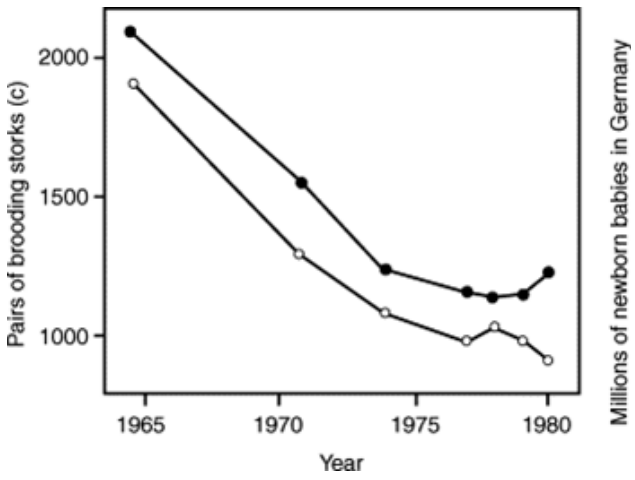
- Tables based tools
  - Spreadsheet based tools, easy to manipulate
  - Examples: Excel, Calc, …
  - Pro: intuitive to use, quick results
  - Con: Repeatability (what if you need that report every month?)
- Data flow based tools
  - Visually assemble analysis flow step by step
  - Examples: IBM Miner, Clementine, KNIME
  - Pro: easy to use, communicate and document, repeatable
  - Con: not quite as flexible (but script integration helps)

1.9. Terminology



1.10. Correlation ⇏ Causality

Hypothesis: stork brings babies ⇒ correlation is significant and positive!



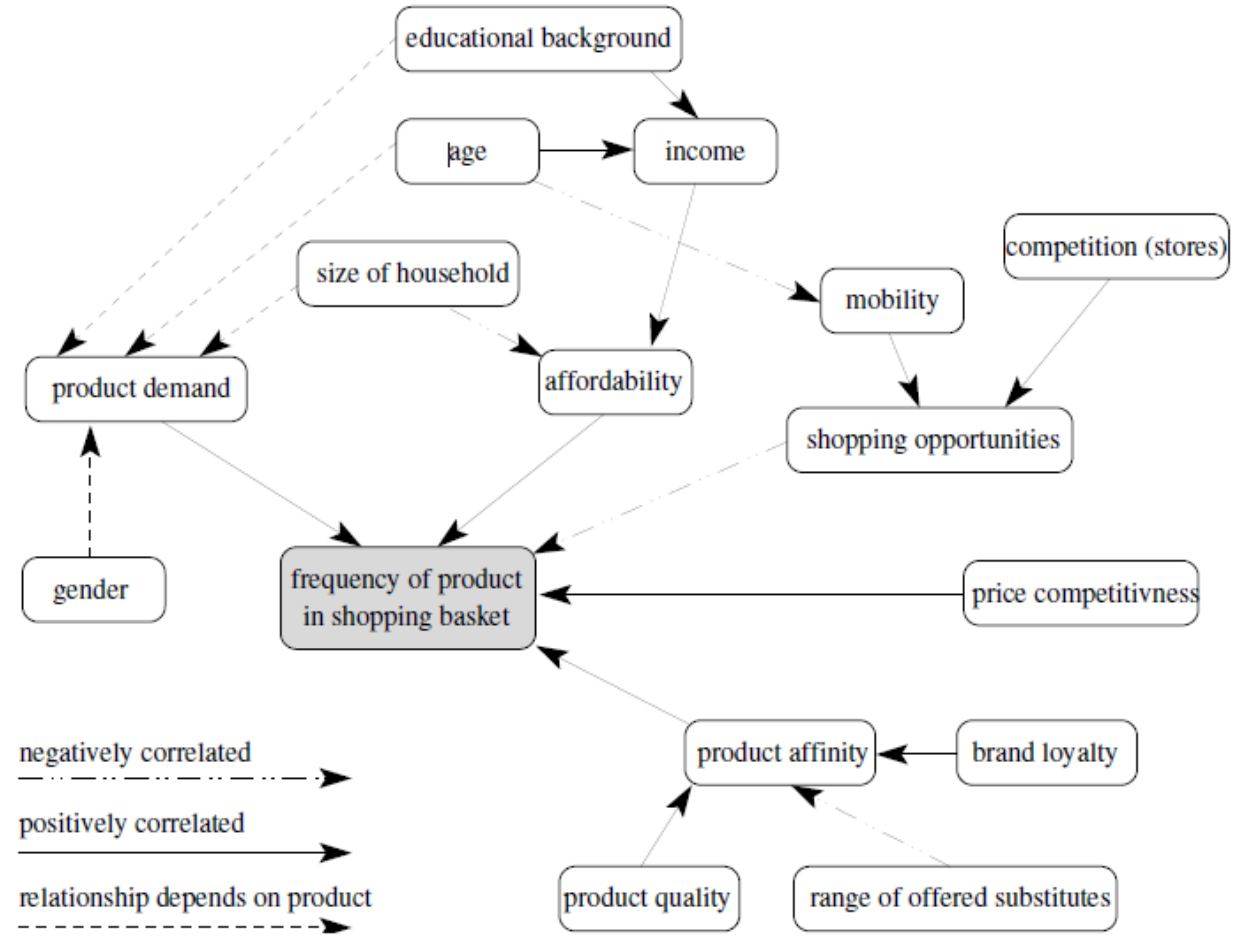
2. Project Understanding

2.1. General

- Problem formulation
- Mapping the problem formulation to a data analysis taks
- Understanding the situation (available data, suitability of the data, …)
- The 80-20 Rule:
  - Average time spent for project and data understanding: 20%
  - Importance for success: 80%

2.2. Determine the project objective

Cognitive Map



Assess the situation

- Requirements and constraints
  - Model requirements, e.g. model has to be explanatory
  - Ethical, political, legal issues, e.g. variables such as gender, age, race must not be used
  - Technical constraints, e.g. time limits

- Assumptions
  - Representativeness:
    The sample represents the whole
  - Informativeness:
    The model includes all important information.
  - Good data quality
  - Presence of external factors:
    The external world is not changing

2.3. Determine analysis goals

- Determine data mining tasks
  - Classification, regression, cluster analysis
  - Finding associations, deviation analysis, …
- Specify the requirements for the models
- Determine analysis goals
  - Interpretability
  - Reproducibility/stability
  - Model flexibility/adequacy
  - Runtime
  - Interestingness and use of expert knowledge

3. Data Understanding

3.1. Questions in Data Understanding

Goal: Gain insight in your data

1. With respect to your project goals
2. And general

Find answers to questions

1. What kind of attributes do we have?
2. How is the data quality?
3. Does visualization helps?
4. Are attributes correlated?
5. What about outliers?
6. How are missing values handled?

3.2. Attribute Understanding

We often assume that the data set is provided in form of a simple table:

- The rows of the table are called **instances, records** or **data objects.**
- The columns of the table are called **attributes, features** or **variables.**

Types of attributes

- **Categorical (nominal):**
  - finite domain
  - often called *classes* or *categories*
  - Examples: {female, male}, {ordered, sent, received}
- **Ordinal**:
  - finite domain with a linear ordering on the domain
  - Examples: {B.Sc., M.Sc., Ph.D.}
- **Numerical:** values are numbers
- **Discrete:** categorical/numerical attribute, whose domain is a subset of the integer numbers
- **Continuous:** Numerical attribute with values in the real numbers or an interval

3.3. Data Quality

Low data quality makes it impossible to trust analysis results: "Garbage in, garbage out."

Accuracy: Closeness between the value in the data and the true value

- Reason of low accuracy of **numerical attributes**:
  - Noisy measurements, Limited precision
  - Wrong measurements, Transposition of digits (when entered manually)
- Reason of low accuracy of **categorical attributes**:
  - erroneous entries, typos

- **Syntactic accuracy:**
  - Entry is not in the domain
  - Examples: fmale in gender, text in numerical attributes, …
  - Can be checked quite easy
- **Semantic accuracy:**
  - Entry is in the domain but not correct.
  - Example: John Smith is female
  - Needs more information to be checked
- **Completeness:**
  - Is violated if an entry is not correct although it belongs to the domain of the attribute
  - Example: Complete records are missing, the data is biased
- **Unbalanced data**:
  - The data set might be biased extremely to one type of records
  - Example: Defective goods are a very small fraction of all
- **Timeliness**:
  - Is the available data up to date?

3.4. Data Visualization

Bar chart: simple way to depict the frequencies of the values of a categorical attribute

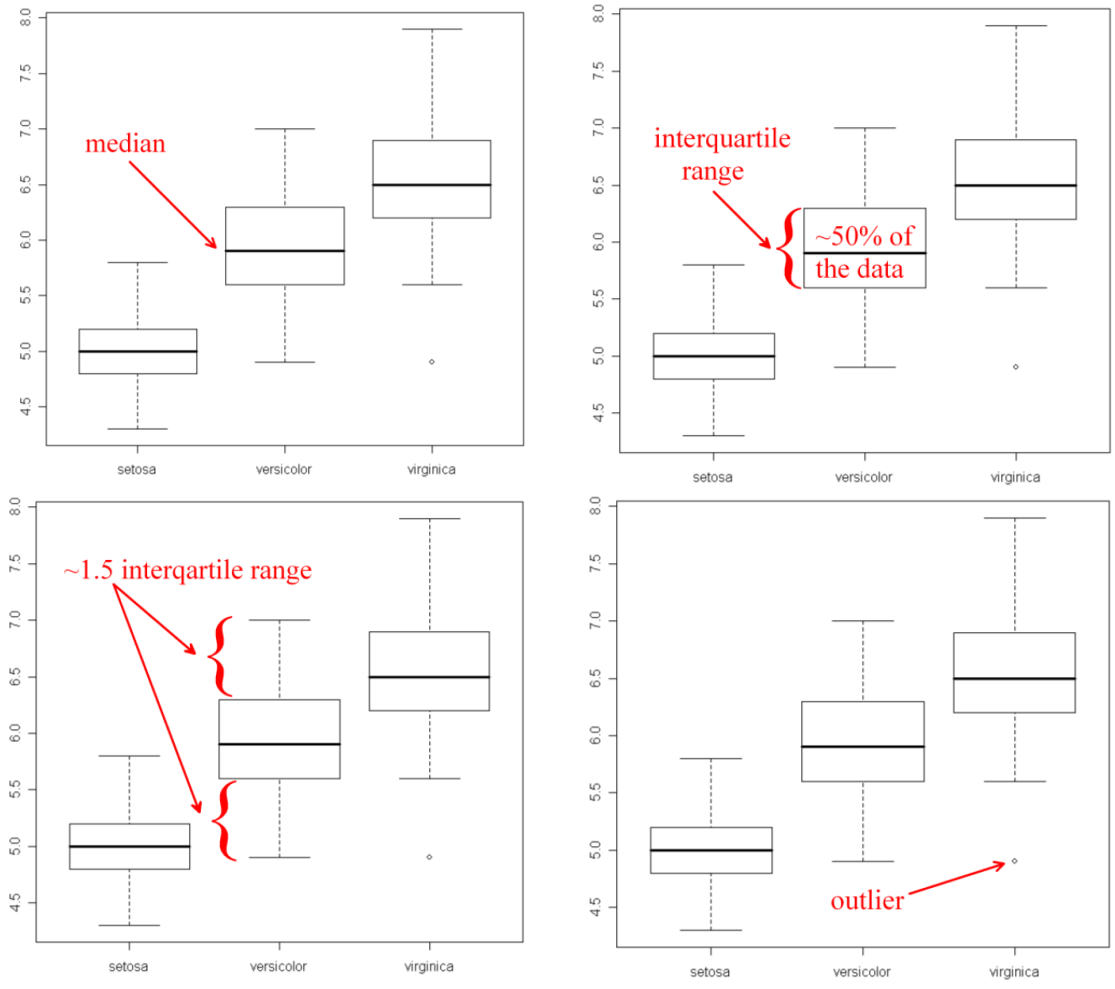
Histogram

- Shows the frequency distribution for a numerical attribute.
- Range of the numerical attribute is discretized into a fixed number of intervals (bins), usually of equal length
- For each interval the (absolute) frequency of values falling into it is indicated by the height of a bar
- Number of bins according to ***Sturges' rule***: $k = \lceil log_2(n) + 1 \rceil$, where n is the sample size. The rule is suitable for data from normal distributions and from data sets of moderate size

Reminder:

- Median: Value in the middle (for given values in increasing order)
- q%-quantile (0 < q < 100): The value for which q% of the values are smaller and 100-q% are lager. Median is the 50%-quantile
- Quartiles: 25% -quantile (1st quartile) median (2nd quantile), 75%-quantile (3rd quartile)
- Interquartile range (IQR): 3rd-1st quantile

Boxplots



Scatterplot

- Visualizes two variables in a two-dimensional plot.
- Each ax corresponds to one variable
- Can be enriched with additional information: color or different symbols to incorporate a third attribute

3.5. Correlation Analysis

= similar behavior of two attributes

Pearson's correlation coefficient

- Is a measure for a *linear relationship* between two *numerical* attributes X and Y
- $r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y}$, where $\bar{x}$ and $\bar{y}$ are the mean values of X and Y**.** s_x and s_y are the corresponding (sample) standard derivations (Standardabweichung)
- $-1 \leq r_{xy} \leq 1$
- The larger the absolute value of the Pearson correlation coefficient, the stronger the linear relationship between the two attributes.
- For $|r_{xy}| = 1$ the values of X and Y lie exactly on a line
- Positive (negative) correlation indicates a line with positive (negative) slope
- For monotone functional, but non-linear relationship coefficient will not be -1 or 1
- Can be close to zero despite (trotz) a monotone functional relationship

Rank correlation coefficients

- Ignoring the exact numerical values of the attributes and considering only the ordering of the values
- Intended to measure monotonous correlations, where the monotonous function does not have to be linear

- **Spearman's rank correlation coefficient (Spearman's rho):**
  - $p = 1 - 6\,\frac{\sum_{i=1}^{n}(r(x_i) - r(y_i))^2}{n\,(n^2-1)}$, where $r(x_i)$ is the rank of value x_i when we sort the list $(x_1, \ldots, x_i)$ in increasing order. $r(y_i)$ is defined analogously.
  - When the rankings of the x- and y-values are exactly in the same order, Spearman's rho will yield the value 1
  - In reverse order, the value -1

3.6. Outlier Detection

Outlier = value or data object, that is far away or very different from all or most of the other data

Causes for outliers

- Data *quality* problems (erroneous data coming from wrong measurements or typing mistakes)
- Exceptional or unusual *situations/data objects*

Outlier handling

- Outliers coming from erroneous data should be *excluded*
- Even if outliers are correct (exceptional data), it is sometime useful to exclude them. E.g. a single extremely large outlier can lead to completely misleading values for the mean value

Outlier detection: Single attributes

- **Categorical attributes:**
  - an outlier is a value that occurs with a *frequency extremely lower* than the frequency of all other values
  - Sometimes outliers are the target objects of the analysis
  - *Example:* Automatic quality control system
  - *Goal:* Train as classifier, classifying the parts as correct or with failures based on measurements of the produced parts.
  - The frequency of the correct party will be so high, that the parts with failure might be considered as outliers.

- **Numerical attributes:**
  - Outliers in boxplots
  - Statistical tests, for example *Grubb's test*

3.7. Missing values

Causes for missing values

- Broken sensors
- Refusal to answer a question
- Irrelevant attribute for the corresponding object

Missing values might not necessarily be indicated as missing (instead: zero und default values).

3.8. A checklist for data understanding

- Determine the quality of the data (e.g. syntactic accuracy)
- Find outliers. (e.g. using visualization techniques)
- Detect and examine missing values. Possible hidden by default values
- Discover new or confirm expected dependencies or correlations between attributes
- Check specific application dependent assumptions (e.g. the attribute follows a normal distribution)
- Compare statistics with the expected behavior
- Check the **distribution for each attribute** (unexpected properties like outliers, correct domains, correct medians)
- Check **correlations or dependencies** between pairs of attributes

3.9. Methods for higher-dimensional data

Visualizing multidimensional data

- **3D** techniques can be used to incorporate three axes (attributes)
- For m attributes there are $\binom{m}{2} = m\ (m-1)$ possible **3D scatter plots**
- **Parallel Coordinates:**
  - Draw coordinate axes parallel to each other
  - Not limitation for the number of axes to be displayed
  - For each data object, a polyline is drawn connecting the values of the data object for the attributes on the corresponding axes
- **Radar plots:**
  - Coordinate axes drawn in a star-like fashion intersecting in one point
- **Star plot:**
  - Similar to radar plots, but each object is drawn separately

Outlier Detection

- Scatter plots for (visually detecting) outliers w.r.t. two attributes
- PCA or MDS plots for (visually detecting) outliers
- Cluster analysis techniques: Outliers are those points which cannot be assigned to any cluster

Principle approach for incorporating all attributes in a plot:

- Preserve as much of the “structure”
- Define measurement that evaluates lower-dimensional representations (plots) of the data in terms of how well a representation preserves the original “structure” of the high-dimensional data set
- Find the plot that gives the best values for the defined measure
- There is no unique measure for “structure” preservation

Multidimensional scaling (MDS)

- Is not constructing an explicit mapping from the high-dimensional space to the low-dimensional space
- Only positions the data points in the low-dimensional space
- Uses the distances between the high-dimensional data points as structure preservation criterion

- **Input:** $(x_1, \dots, x_n)$ with $x_i \in \mathbb{R}^d =: X$ (input space)
- **Output:** $(p_1, \dots, p_n)$ with $p_i \in \mathbb{R}^q =: Y$ (output space, usually q= 2, sometimes also q= 3)
- **Goal:** define a point p_i for each object x_i such that the distances $d_{ij}^{(Y)}$ between the points p_i and p_j are (roughly) the same as the distances $d_{ij}^{(X)}$ between the original data objects x_i and x_j
  Usually $d_{ij}^{(X)} = \|p_i - p_j\|$

- **Baseline:** distance matrix $\left[d_{ij}^{(X)}\right]_{1 \le i,j \le n}$ where $d_{ij}^{(X)}$ is the distance between object I and object j
- **Distance requirement:**
  - Non-negative: $d_{ij}^{(X)} \geq 0$
  - Symmetric: $d_{ij}^{(X)} = d_{ij}^{(X)}$
  - Diagonal should be zero: $d_{ij}^{(X)} = 0$ (Each data object has distance zero to itself)
  - Usually the distances are the Euclidean distances of the data object (after normalization) in the high-dimensional space

- **Objective functions:**
  - Energy functions define the quality of a solution
    - E0: Absolute squared error (TODO Equation)
    - E1: Normalized absolute squared error (TODO Equation)
    - E2: Relative squared error
    - E3: mixture between relative and absolute squared error (called stressed)
  - Normalization factor doesn't has influence on the location of the minimum of the objective function
  - In contrast to E0, the value of E1 does neither depend on the number of data objects nor the magnitude of the original distances
  - MDS based on E3 is called Sammon sampling

- Non-linear optimization problem with q*n (2n for q=2) parameters to be optimized
- Even for small data set a two-dimensional MDS representation requires the optimization of hundreds of parameters (searching a x and y position for each data point)
- A gradient descent method is used to minimize the objective function for MDS

- **Gradient descent method**
  - Given a differentiable objective function $f: \mathbb{R}^k \rightarrow \mathbb{R}$ with k parameters for which we want to find the minimum
  - f defines a (k+1)-dimensional "landscape" in which we want to find the lowest point

1. **Start with an arbitrary initial solution $\boldsymbol{x_0}$**
2. **Compute the gradient** of f at $\boldsymbol{x_0}$
   The gradient is the vector of partial derivatives.
   The gradient points in the direction of steepest ascent.
3. From $\boldsymbol{x_0}$ go a fixed (or variable) step width in the **opposite direction of the gradient** (the direction of the steepest descent) to find the next solution $\boldsymbol{x_1}$.
4. Continue in the same way with $\boldsymbol{x_1}$ (steps 2 and 3), $\boldsymbol{x_2}$, … until a stop criterion is satisfied. ((almost) no change of the $\boldsymbol{x_i}$, maximum number of iteration steps, …)

4. Version Space

4.1. Inductive Learning

- Data are instances (records) from an instance space X often $X \in D_1 \times \ldots \times D_d$ where D_i: domain of attribute i
- Given a (relatively small) sample of data from X (training data)
- Given a *target function* specifying the learning goal
- We want to induce *general* hypothesis approximating the target function on the whole instance space from the *specific* training data

Fundamental Assumption: Any hypothesis approximating the target function well over the training data, will also approximate the target function well over the unobserved instances of X.

4.2. Concept Learning

- Concept C: subset of X, $c: X \rightarrow 0{,}1$ is the characteristic function of C
- Task: approximate the target function c using the attributes of X in order to distinguish instances (not) belonging to C
- Training data D: positive and negative examples of the concept: $\langle x_1, c(x_1)\rangle, \ldots, \langle x_n, c(x_n)\rangle$

Example for the following algorithms:

- **Concept:** days on which Jane enjoys her favorite water sports
- **Task:** predict the value of "Enjoy Sport" for an arbitrary day based on other attributes

Sky	Temp	Humid	Wind	Water	Forecast	Enjoy Sport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

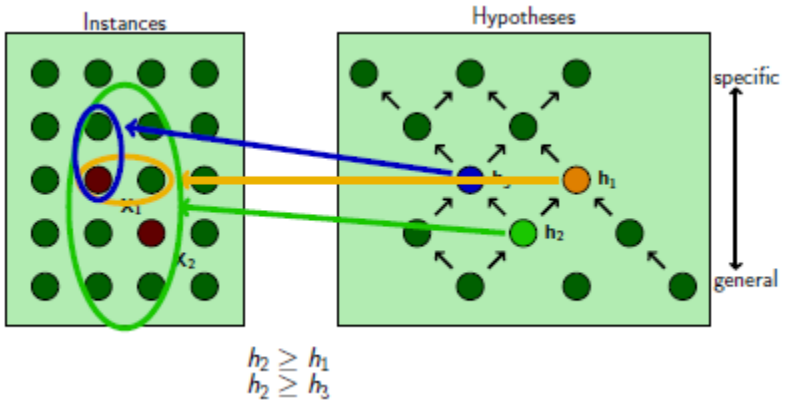
- Task more formally: want to induce hypothesis $h: X \in \{0, 1\}$ from a set of (possible) hypothesis H such that $h(x) = c(x), \forall x \in D$
- Hypothesis h is a conjunction of constraints on attributes
- Each constraint can be:
  - A specific value: e.g. Water = Warm
  - A don't care value: e.g. Water = ?
  - No value allowed (null hypothesis): e.g. Water = ∅

- Six (nominal) Attributes:
  - Sky: Sunny, Cloudy, Rainy
  - AirTemp: Warm, Cold
  - Humidity: Normal, High
  - Wind: Strong, Weak
  - Water: Warm, Cold
  - Forecast: Same, Change
- Distinct instances: 3*2*2*2*2*2 = 96
- Distinct concepts: 2^{96}
- Syntactically distinct hypotheses: 5*4*4*4*4*4 = 5120
- Semantically distinct hypotheses: 1 + 4*3*3*3*3*3 = 973

Ordering the Hypothesis Space

- Example:
  $h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$
  $h_1 = \langle Sunny, ?, ?, ?, ?, ? \rangle$
- Sets of instances covered by h_1 and h_2:
  h_2 imposes fewer constraints than h_1 and therefore classifies more instances x as positive than h_1
- Let h_j and h_k be hypotheses from H, i.e. Boolean-valued functions defined ofer X
  Then h_j is more general than or equal h_k, $(h_j \geq h_k)$ if ∀ x ∈ X:
  $$[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$
- The relation ≥ imposes a partial order over the hypothesis space H (general-to-specific ordering)

Relationship between Instance and Hypotheses



$x_1 = \langle Sunny, Warm, High, Strong, Cool, Same \rangle$
$x_2 = \langle Sunny, Warm, High, Light, Warm, Same \rangle$

$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$
$h_2 = \langle Sunny, ?, ?, ?, ?, ? \rangle$
$h_3 = \langle Sunny, ?, ?, ?, Cool, ? \rangle$

Searching the Hypothesis Space

- Exhaustive search is infeasible in rl applications
- Exploit the ordering:
  - *Bottom-up:* start with general hypotheses and keep specializing
  - *Top-down*: start with specialized hypothesis and keep generalizing

Find-S Algorithm

Finds one maximally specific hypothesis

- Initialize h to the most specific hypothesis in H
- **For each** positive training instance x
  For each attribute constraint a_i in h
  If the constraint a_i in h is satisfied by x
  Then do nothing
  Else generalize a_i with respect to "≥" until a_i is satisfied by x
- Output hypothesis h.

x_1 =<Sunny,Warm,Normal,Strong,Warm,Same> + $h_0 = < \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$

x_2 =<Sunny,Warm,High,Strong,Warm,Same> + h_1 =<Sunny,Warm,Normal,Strong,Warm,Same>

x_3 =<Rainy,Cold,High,Strong,Warm,Change> − $h_{2,3}$ =< Sunny,Warm,?,Strong,Warm,Same>

x_4 =<Sunny,Warm,High,Strong,Cool,Change> + h_4 =<Sunny,Warm,?,Strong,?,?>

- Algorithm is very efficient!
- Ignores negative training examples

Version Space

- A hypothesis h is **consistent** with a set of training examples D of the target concept C if $h(x) = c(x)$ for each $< x, c(x) >$ in D
- The **version space**, $VS_{H,D}$, with respect to hypothesis space H and the training set D is the subset of hypothesis from H consistent with all training examples
- The **general boundary,** G of version space $VS_{H,D}$ is the set of its maximally general members
- The **specific boundary**, S of version space $VS_{H,D}$ is the set of maximally specific members
- Every member of the version space lies between these boundaries
- *G&S* are the compact representation of the version space

Candidate Elimination Algorithm

- **For each** training example $d = <x, c(x)>$
  - **If** *d* is a positive example
    - Remove from G any hypothesis that is inconsistent with *d*
    - **For each** hypothesis s in S that is not consistent with *d* remove s from S
    - Add to S all minimal generalizations h of s such that
      - (1) h is consistent with d and
      - (2) Some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S
  - **If** *d* is a negative example
    - Remove from S any hypothesis that is inconsistent with *d*
    - **For each** hypothesis g in G that is not consistent with *d* remove g from G
    - Add to G all minimal specializations h of g such that
      - (1) h is consistent with d and
      - (2) Some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

Initialization

S: $\{< \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >\}$
G: $\{<?, ?, ?, ?, ?, ? >\}$

$x_1 =< Sunny, Warm, Normal, Strong, Warm, Same > \ +$

S: $\{< Sunny, Warm, Normal, Strong, Warm, Same >\}$
G: $\{<?, ?, ?, ?, ?, ? >\}$

$x_2 =< Sunny, Warm, High, Strong, Warm, Same > \ +$

S: $\{< Sunny, Warm, ?, Strong, Warm, Same >\}$
G: $\{<?, ?, ?, ?, ?, ? >\}$

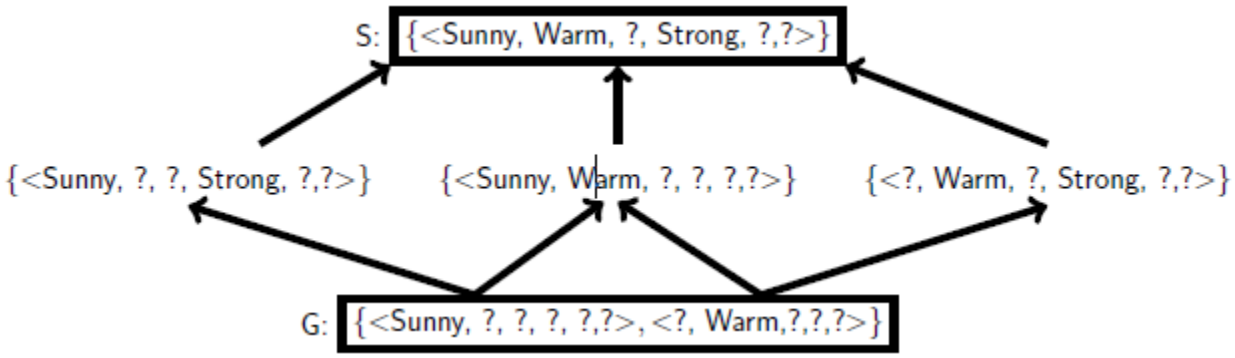
$x_3 =< Rainy, Cold, High, Strong, Warm, Change > \ -$

S: $\{< Sunny, Warm, ?, Strong, Warm, Same >\}$
G: $\{< Sunny, ?, ?, ?, ?, ? >, <?, Warm, ?, ?, ?, ? >, <?, ?, ?, ?, ?, Same >\}$

$x_4 =< Sunny, Warm, High, Strong, Cool, Change > \ +$

S: $\{< Sunny, Warm, ?, Strong, ?, ? >\}$
G: $\{< Sunny, ?, ?, ?, ?, ? >, <?, Warm, ?, ?, ?, ? >\}$

- **The complete Version Space**



- **Classification of new data**

$x_5 = < Sunny, Warm, Normal, Strong, Cool, Change > \quad + 6/0$

$x_6 = < Rainy, Cold, Normal, Light, Warm, Same > \quad - 0/6$

$x_7 = < Sunny, Warm, Normal, Light, Warm, Same > \quad ? 3/3$

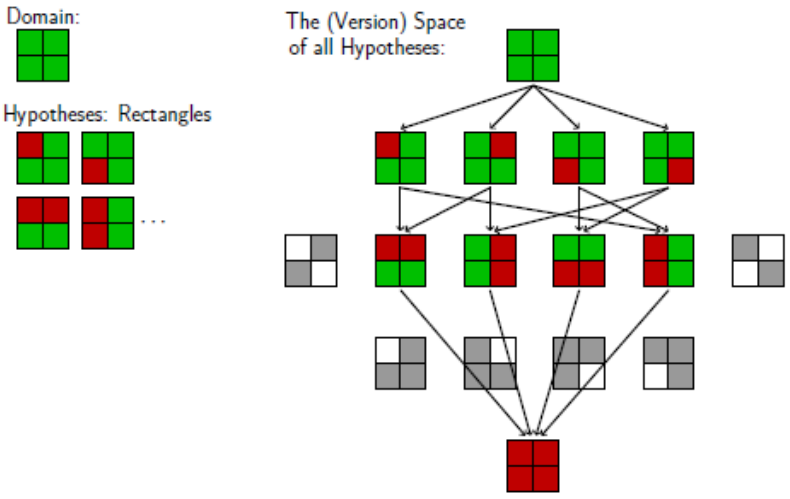
$x_8 = < Sunny, Cold, Normal, Strong, Warm, Same > \quad ? 2/4$

- Exploits negative training examples
- Finds all consistent hypothesis from H
- Can determine confidence of classification of new data
- Can detect inconsistencies in training data
- Algorithm is not very efficient

Inductive Bias

- Hypothesis space is unable to represent a simple disjunctive target connect:
  (Sky = Sunny) v (Sky = Cloudy)
- **Unbiased learner:**
  - Idea: Choose H that expresses every teachable concept, i.e. H is the set of all subsets of X
  - H: conjunctions, disjunctions, negations of constraints on attrobutes
    e.g. <Sunny, Warm, Normal, ?, ?, ?> v <?, ?, ?, ?, ?, Change>
  - Example for G & S:
    - Positive examples: x_1, x_2, x_3
    - Negative examples: x_4, x_5
    - S: (x_1 v x_2 v x_3)
    - G: ¬ (x_4 v x_5)
  - No generalization beyond the training examples
    - Can classify only the training examples themselves.
    - Needs every single instance in X as a training example

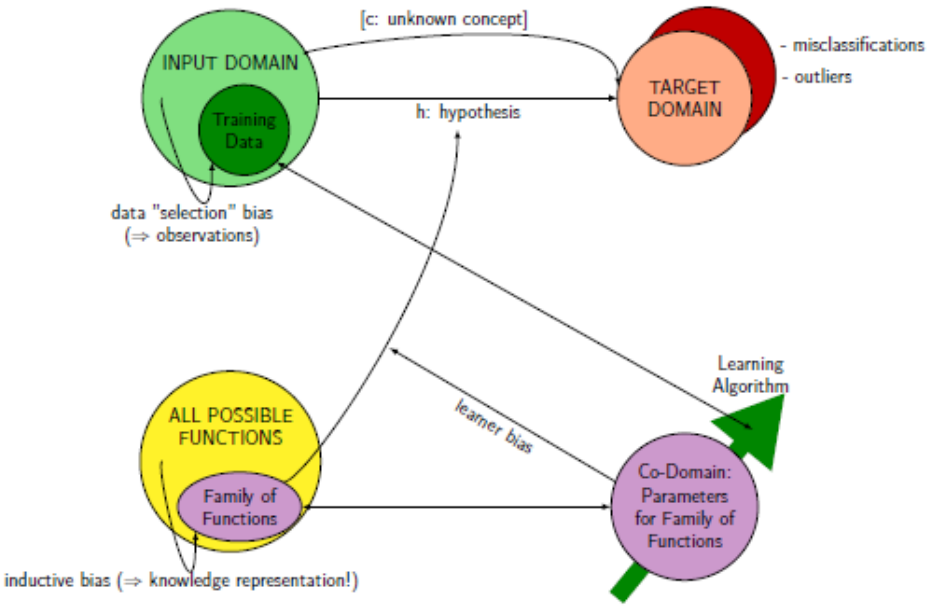
- A learner that makes no prior assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances
- Inductive bias: set of assumptions that justify the inductive inferences as deductive inferences
- Use domain of application to choose appropriate inductive bias
- Too vague inductive bias: connate generalize well
- Too strict inductive bias: no consistent hypothesis



Discussion of Concept Learners

All concept learners suffer from the following limitations:

- Cannot handle inconsistent training data (noise)
  - Modification possible (how?)
- One rule to describe training data
  - Not expressive enough
- Overfit the training data
  - Because of the data driven search strategy (bottom-up)
- Need more sophisticated methods for real life problems



5. Data Preparation

5.1. Data understanding vs. Data Preparation

Data understanding provides general information about the data like

- The existence and partly also about the character of massing values
- Outliers
- The characteristics of attributes
- Dependencies between attributes

Data preparation uses this information to

- Select attributes,
- Reduce the dimensionality of the data set
- Select records
- (try to) handle missing values
- (try to) handle outliers
- Integrate, unify and transform data
- Improve data quality

5.2. Feature(s)

Construction refers to generating new features from given attributes.
Selection

- … refers to techniques to choose a subset of the features (attributes) that is as small as possible and sufficient for the data analysis (=still informative!)
- Includes
  - Removing (more or less) **irrelevant features**
  - Removing **redundant features**

- **Techniques**
  - *Selecting the top-ranked features*
    Choose the features with the best evaluation when single features are evaluated
  - *Selecting the top-ranked subset*
    Choose the subset of features with the best performance. Requires a exhaustive search and is impossible for larger numbers of features
  - *Forward selection*
    Start with the empty set of features and add features one by one. In each step, add the feature that yields the best improvement of the performance
  - *Backward elimination*
    Start with the full set of features and remove features one by one. In each step, remove the feature that results in the smallest decrease in performance.

- **Reasons for using only a subsample:**
  - *Faster computation*
  - *Cross-Validation with training and test set*
  - *Timeliness:* Data which is outdated can be removed.
  - *Representativeness:* Is the given sample matching the whole population?
    If not and there are information about the true distribution select representative subsample. (e.g. there are more women than men in a IT-questionaire)
  - *Rare events:* Specifically select rare events to model them better (over sampling)

5.3. Data cleansing

... or **data scrubbing** refers to detecting/correcting/removing

- Inaccurate
- Incorrect
- Incomplete

records from a data set.

5.4. Improve data quality

- Turn all characters into capital letters to level case sensitivity
- Removes spaces and nonprinting characters
- Fix the format of numbers, date and time (including decimal point)
- Split fields that carry mixed information into two separate attributes (**field overloading**)
- Use spell-checker or stemming to normalize spelling in free text entries
- Replace abbreviations by their long form (with the help of a dictionary). Or vice versa.
- Normalize the writing of addresses and names, possibly ignoring the order of title, surname, forename, etc. to ease their re-identification
- Convert numerical values into standard units, especially if data from different sources are used (e.g. different countries)
- Use dictionaries containing all possible values of an attribute to assure that all values comply with the domain of knowledge

5.5. Missing value

- *Ignorance/detection:* Delete the whole record
- *Imputation:* The missing values may be replaced by some estimate (mean, median, …)
- *Explicit value:* Use a specific value as missing for the model. (e.g. -1, if only positive numbers on the domain)

5.6. Transformation of data

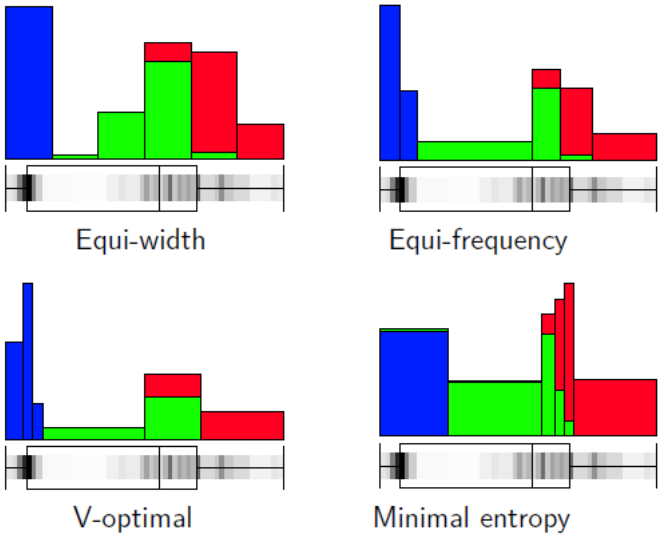
Some model can only handle numerical attributes, other models only categorical attributes.

Categorical ⇒ Numerical

- *Binary attribute:* numerical attribute with the values 0 and 1
- *Ordinal attributes* (“sortable”): enumerate in the correct order 1, …, k
- *Categorical attribute* (not ordinal) with more than two values (a_1, …, a_k) should *not be converted into a single numerical attribute.*
  Instead: convert to k attributes A_1, …, A_k with values 0 and 1. a_i is represented by $A_i = 1$ and $A_j = 0$ for $i \neq j$ (1-of-n encoding)

Numerical ⇒ Categorical

- **Equi-width discretization:** Split the range into intervals (bins) of the same length
- **Equi-frequency discretization:** Split the range into intervals such that each interval (bin) contains (roughly) the same number of records
- **V-optimal discretization:** Minimize $\sum_i n_i V_i$ where n_i is the number of data objects in the i-th interval and V_i is the sample variance of the data in this interval
- **Minimal entropy discretization:** Minimizes the entropy (Only applicable in the case of classification problems)



5.7. Normalization/Standardization

- … for analysis techniques in which the influence of an attribute depends on the scale or measurement unit (e.g. PCA, MDS; cluster analysis).

For a numerical attribute X:

- **Min-max normalization:**
  $n: dom(x) \to [0,1], \quad x \mapsto \frac{x - min_X}{max_X - min_X}$
- **Z-score standardization:**
  Sample mean: $\hat{\mu}_X$ and empirical standard deviation: $\hat{\sigma}_X$
  $s: dom\ (X) \to \mathbb{R}, \quad x \mapsto \frac{x - \hat{\mu}_X}{\hat{\sigma}_X}$
- **Decimal scaling:**
  s is the smallest integer value larger than $log_{10}(max_X)$
  $d: dom\ (X) \to [0,1], \quad x \mapsto \frac{x}{10^s}$

Principal component analysis (PCA)

- Constructs a **projection** from the high-dimensional space to a lower-dimensional space (plane or hyperplane)
- Uses the **variance** in the data as structure preservation (Strukturerhaltung) criterion
- Tries to **preserve as much of the original variance** of the data when projected to a lower-dimensional space
- Variance of a multidimensional data set: Sum of the variances of the attributes

- The data points are first centered around the origin by subtracting the mean values
- **Goal:** Find a projection in the form of a linear mapping $\mathbb{R}^m \to \mathbb{R}^q$ (for visualization purposes choose q = 2 or 3 according to dimensions) given by

$$\boldsymbol{y} = \boldsymbol{M} * (\boldsymbol{x} - \bar{\boldsymbol{x}})$$

where M is a *q x m* matrix such that the variance of the projected data $y_i = M * (x_i - \bar{x})$ is as large as possible

- **Problem:** Without restriction for the matrix M, the entries in M can be chosen arbitrary large, so that the data are **not only projected but also stretched**, leading to an arbitrary large variance of the projected data.
  Therefore: Introduce constraints such that the matrix M is only a projection.

- **Constraints:** The row $\mathbf{v_i}$ of the matrix M = (v_1, …, v_q) must be normalized i.e. $\|v_i\| = 1$

  The projection matrix M for PCY is given by **M = ($\mathbf{v_1}$, …, $\mathbf{v_q}$)** where the **principal components $\mathbf{v_1}$, …, $\mathbf{v_q}$** are the normalized eigenvectors of the **covariance matrix** C of the date for the q largest eigenvalues $\lambda_1 \geq \ldots \geq \lambda_q$.
  λ is called an eigenvalue of a matrix A, if there is a non-zero vector v such that Av = λv holds. The vector v is called eigenvector tot the eigenvalue λ.

- **Dimension reduction:**
  - Let $\lambda_1 \geq \ldots \geq \lambda_m$ be the eigenvalues of the covariance matrix.
  - When we project the data to the first q principal components v_1, …, v_q corresponding to the eigenvalues λ_1, …, λ_q, this projection will preserve a fraction of

$$\frac{\lambda_1 + \ \ldots + \lambda_q}{\lambda_1 + \ \ldots + \ \lambda_m}$$

    Of the variance of the original data.

6. Modeling

6.1. Model class

- Model = the form or structure of the analysis result
- Here the parameters are **not** defined only the type is selected

Requirements

- **Simplicity**
  - Occam's razor: Choose the simplest model that still "explains" the data
  - Easier to understand
  - Lower complexity
  - Avoid overfitting

- **Interpretability**
  - Black-Boxes are mostly not a proper choice
  - But: they often produce very good accuracy (e.g. neural networks)

Global vs. local models

- **Global models** provide a (not necessarily good) description for the whole data set, e.g. Regression line
- **Local models** or **patterns** provide a description for only a part or subset of the data set, e.g. Association rules

6.2. Score function

- Find an **objective function** $f: M \rightarrow \mathbb{R}$
- Which evaluates the quality of your model
- Used to determine the "best" model
- Does not tell how to find the best or a good model
- Only provides a means for comparing models

Error functions for classification problems

- Very common **misclassification rate** = $\frac{\#\, wrong\ classified}{\#\, total\ classified}$
- A low misclassification rate does not necessarily tell anything about the quality of a classifier
- When classes are **unbalanced**

- **Cost Matrix**
  - More general approach than the misclassification rate
  - The consequences (costs) for misclassification for one class might be different than for another class
  - The **expected loss** should be minimized

Example

(Hypothetical) cost matrix for the tea cup production problem

true class	OK	broken
OK	0	1
broken	10	0

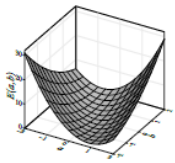
A classifier might classify a specific cup with 80% to the class ok and with 20% to the class broken.

- Expected loss for choosing ok: $0.8 \cdot 0 + 0.2 \cdot 10 = 2$.
- Expected loss for choosing broken: $0.8 \cdot 1 + 0.2 \cdot 0 = 0.8$.

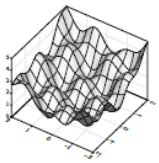
Choose broken in this case to minimize the expected loss!

6.3. Modeling algorithms

Algorithms for model fitting



$k = 2$
A landscape for the mean squared error.



$k = 2$
A landscape with many local minima and maxima

- For differentiable score functions, **gradient methods** can be applied
- **Problems:**
  - Will only find local optima
  - Parameters)step width) must be adjusted or computed in each iteration step
- For discrete problems with a finite search space (like finding association rules), **combinatorial optimization** strategies are needed
- In principle, an **exhaustive search** of the finite domain M is possible, but in most cases not feasible, since M is much too large

- **Heuristic strategies**
  - **Random search:** Create random solutions, choose the best among them (**very inefficient**)
  - **Greedy strategies:** Formulate algorithm that tries to improve the solution in each step
    - *Example:* Gradient method
    - *Example:* Hill-climbing
      Start with a random solution, generate new solutions in the "neighborhood" of the solution. If a new solution is better than the old one, generate new solutions in its "neighborhood".
    - **Can find a solution quick, but can get stuck in local optima.**
  - Simulated annealing
  - Evolutionary algorithms
  - Alternating optimization

- **Overfitting**
  - Fitting the noise rather than fitting the underlying relationship.
  - Typical indicator: "Perfect fit" e.g. error gets near zero
  - Possible solution: choose a less flexible model

Model error

- **Experimental error**
  - Also *Pure error* or *intrinsic error* is inherent in the data and due to noise, random variations, imprecise measurements or the influence of hidden variables that cannot be observed
  - Impossible to overcome this error by the choice of a suitable model
  - In context of classification problems also called *Bayes error*

- **ROC curves**
  - = receiver operating characteristic curve
  - Used to judge a classifier without a cost matrix or the misclassification rate
  - Draws false-positive rate against true-positive rate (depending on threshold)
  - Ideal case: only true positives and no false negatives
  - Area under curve (AUC) is indicator how well the classifier solves the problem. The larger the area, the better the solution for the classification problem.

- **Confusion Matrix**
  - Table where rows represent true classes and the columns predicted classes
  - Entries = how many objects are classified into class of corresponding column

- **Sample error**
  - data is not a perfect representation of the underlying data
  - the smaller the sample the smaller the probability for a perfect model
  - Example: Throw a dice. What if the dice was manipulated?

- **Model error**
  - simpler model ⇒ bigger error
  - more complex model ⇒ overfitting and larger error on new data
  - type of model ⇒ different "fit" to data

- **Algorithmic error**
  - Based on selected algorithm
  - E.g. gradient descent ⇒ local minima
  - Can often be estimated (several runs are similarly biased)
  - Reality: assuming that the algorithm is good enough

- **Machine Learning Bias:** Model and Algorithmic Error
- **Variance:** Intrinsic (Experimental) and Sample Error

- **Learning without a bias is not possible!**

6.4. Model validation

- Error for unseen data will most probably always be bigger for the data used for training

Training and Test Data

- Split into two subsets: training and test data
- Train model on training data and measure quality of model on the test data
- Typically 2/3 training, 1/3 test

- Splitting strategies:
  - Random (distribution in both sets should be roughly the same)
  - Stratification (i.e. the distribution of one class should remain)
- Split into training, test and validation
  - Choose for each model kind the best based on the test data
  - Test the best models on the validation data set

Cross-Validation

- Split data multiple times to validate the results, to reduce the effect of single estimation
- Use combination of the received models, or the best
- **K-fold Cross-Validation** (e.g. k = 10):
  - Divide into k subsets
  - Test data is always another subset, training data the rest
  - Average of the k-model error is supposed as the model error
- **Leave-One-Out Method:**
  - For very small data sets
  - Use everything except of one data point for training
  - This single point is used for testing

Bootstrapping

- **Overall goal**: Draw sample multiple times to underline your model (e.g. with the variance of the parameters)
- Pseudo algorithm:
  - Draw k bootstrap samples from the data
  - Learn the model on each sample
  - Calculate the mean and standard deviation
  - Small standard deviation support the model
- Final parameters can be achieved by averaging the k sets of the parameters
- Bagging (Use Bootstrapping to improve the results) see later

Measure for model complexity

- The goal is to fulfill Occam's razor:
  Choose the simplest model that still "explains" the data.

- **Minimum Description Length Principle (MDL)**
  - Basic Idea: Regard the model as a way of data compression
  - To create the data, two things are needed:
    - The decompression rule
    - The compressed data
  - Quality is than measured by the number of bits needed to code these two

- Simplest cases:
  - Compressed data of size 1, by adding the data to the rule (data = original)
  - Decompressed rule of size 1, by saving the real data set as the compressed
- **Goal:** find a solution inbetween
- Example: P. 257 ff.

- **Akaike's Information Criterion measures:**
  - Model complexity by number of parameters (k)
  - Fit to data by probability of data generated by model (L)
  - AIC = 2k – 2 ln(L)

7. Finding Patterns: Clustering

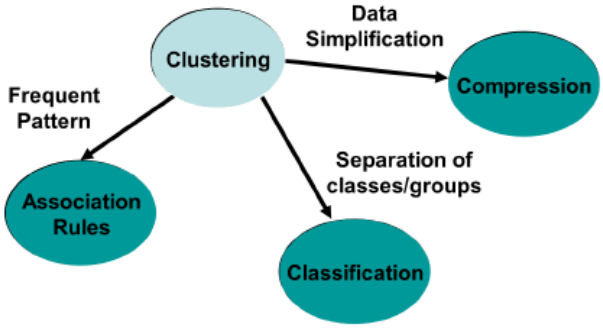
Definition of Finding patters:

- **Patterns describe or summarize** the data set or parts of it
- **No specific target attribute**, whose values should be predicted
- **Visualization** techniques like scatterplots or MDS are methods for finding patterns by visual inspection
- **Correlation analysis** can find patterns in the form of dependencies of pairs of variables

7.1. Definitions to Clustering

Clustering

- Identifying groups of "similar" data objects
- … identifies a finite set of categories, classes or groups (clusters) in the dataset such that objects within the same cluster shall be as similar as possible
- Objects of different clusters shall be as dissimilar as possible
- **Properties:**
  - Clusters may have different sizes, shapes and densities
  - … may form a hierarchy
  - … may be overlapping or disjoint
- … is a basic technique for Knowledge discovery
- Before clustering data should be normalized

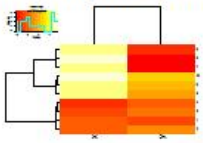


Goal of cluster analysis

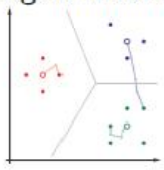
- **Find "natural" clusters and des**cribe their properties ⇒ **Data Understanding**
- Find useful and suitable groupings ⇒ **Data Classification**
- Find representatives for homogenous groups ⇒ **Data Reduction**
- Find unusual data objects ⇒ **Outlier Detection**
- Find random perturbations of the data ⇒ **Noise Detection**

Types of Clustering Approaches

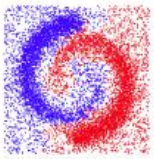
Linkage Based,
e.g. Hierarchical Clustering



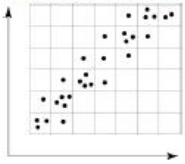
Clustering by Partitioning,
e.g. k-Means



Density Based Clustering,
e.g. DBScan

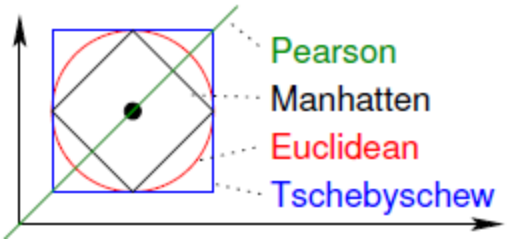


Grid Based Clustering



7.2. Measure Similarity

Notion of (dis-)similarity



- **Numerical attributes:**
  - Dissimilarities between two numerical vectors (Pearson, Manhattan, Euclidean, …)
- **Binary attributes:**
  - The two values can be interpreted as some property being absent or present (e.g. 0,1)
  - A vector can be interpreted as a set of properties that the corresponding object has

Example

- The binary vector $(0, 1, 1, 0, 1)$ corresponds to the set of properties $\{a_2, a_3, a_5\}$.
- The binary vector $(0, 0, 0, 0, 0)$ corresponds to the empty set.
- The binary vector $(1, 1, 1, 1, 1)$ corresponds to the set $\{a_1, a_2, a_3, a_4, a_5\}$.

- **Nominal attributes:**
  - … may be transformed into a set of binary attributes, each of them indicating one particular feature of the attribute (1-of-n coding)

Example

Attribute *Manufacturer* with the values *BMW, Chrysler, Dacia, …* .

manufacturer	…		binary vector
Volkswagen	…	→	00001
Dacia	…		01000
Ford	…		00100

Then one of the dissimilarity measures for binary attribute can be applied.

  - OR compute the proportion of attributes, where both vectors have the same value (Russel and Rao dissimilarity measure)

7.3. Hierarchical Clustering

- … builds clusters step by step
- to decide which data objects belong to same cluster a (dis-)similarity measure is needed
- access to features is not needed; needed is a n x n-matrix with the (dis-)similarity of data objects, with the following conditions:
    - values cannot be negative
    - data objects are similar to themselves
    - two data objects are (dis-)similar to same degree to each other

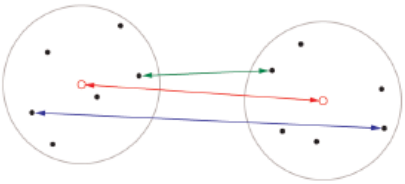
Agglomerative hierarchical clustering (AGNES = Agglomerative Nesting)

Input: *n x n* dissimilarity matrix

1. Start with n clusters, each data object forms a single cluster
2. Reduce the number of clusters by joining those two clusters that are most similar (least dissimilar)
3. Repeat step 2 until there is only one cluster left containing all data objects

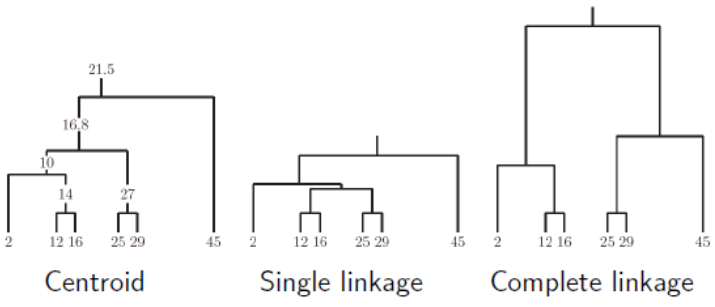
Measuring dissimilarity between clusters

- **Centroid:** Distance between the centroids (mean value vectors) of the two clusters
- **Average Linkage:**
    - Average dissimilarity between all pairs of points
    - Also tends to clearly towards compact clusters
- **Single Linkage:**
    - dissimilarity between most *similar* data objects
    - can "follow chains" in the data
- **Complete Linkage:**
    - Dissimilarity between the two most *dissimilar* data objects
    - Leads to very compact clusters



Dendrograms

- Cluster merging process arranges the data points in a binary tree
- Draw the data tuples at the bottom or left (equally spaced if they multidimensional)
- Draw connection between clusters, that are merged; with the distance to the data points representing the distance between the clusters
- Example: Clustering of the 1-dimensional data set {2, 12, 16, 25, 29, 45}



Choosing the right clusters

- **Simplest Approach:**
  - Specify a minimum desired distance between clusters
  - Stop merging clusters if the closest two clusters are farther apart than this distance
- **Visual Approach:**
  - Merge clusters until all data points are combined into one cluster
  - Draw dendrogram and find a good cut level
  - Advantage: cut needs to be strictly horizontal
- **More sophisticated Approaches:**
  - Analyze the sequence of distances in the merging process
  - Try to find a step in which the distance between the two clusters merged is considerably larger than the distance of the previous step
  - Several heuristic criteria exist for this step selection

Heatmaps

… combine

- A dendrogram resulting from clustering the data
- A dendrogram resulting from clustering the attributes
- Colors to indicate the values of the attributes

Divisive hierarchical clustering (DIANE = Divisive Analysis)

- starts with the whole data set as a single cluster and then divides clusters step by step into smaller ones (top – down)
- *agglomerative* (AGNES) clustering the minimum of the pairwise dissimilarities has to be determined, leading to a *quadratic complexity* in each step
- In *divisive* clustering for each cluster all possible splits would have to be considered
- In the first step, there are 2^{n-1}-1 *possible splits*, where n is the number of data objects
- Bad complexity leads to rarely use of this approach

7.4. Prototype Based Clustering

- **Given:** data set of size *n*
- **Return:** set of typical examples of size *k* << *n*

k-Means clustering

- Choose a number of *k* of clusters to be found (user input)
- Initialize the cluster centers randomly (e.g. by randomly selecting *k* data points)
- **Data point assignment:** Assign each data point to the cluster center that is closest to it (i.e. closer than any other cluster center)

- **Cluster center update:** Compute new cluster centers as the mean vectors of the assigned data points (Intuitively: center of gravity if each data point has unit weight)
- Repeat last two steps until the clusters centers do not change anymore
- Scheme converges, i.e. the update of the cluster centers cannot go on forever
- The clustering result is fairly **sensitive to the initial positions** of the cluster centers (with a bad initialization clustering may fail, stuck in **local minima**)

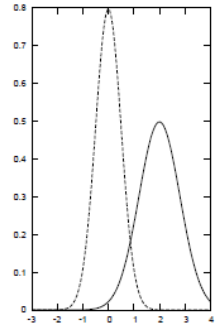
- **Advantages:**
  - Relatively efficient: *O(tkn)* where n is # objects, k is # clusters, and t is # iterations Normally *k, t* << *n* (t typically 5-10)
  - Simple implementation ⇒ most popular partitioning clustering method

- **Weakness:**
  - Often terminates at local optimum. Better local optimum may be found using other techniques like: deterministic annealing an genetic algorithms
  - Applicable only when mean is defined (categorical data?)
  - Need to specify k
  - Unable to handle noisy data and outliers
  - Not suitable to discover clusters with non-convex shapes

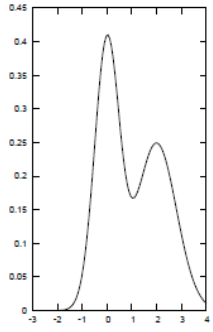
7.5. Gaussian Mixture Models

EM-Clustering

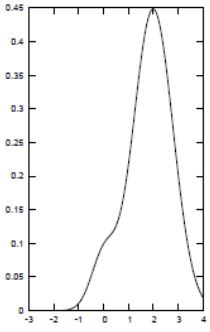
- **Assumption:** Data was generated by sampling a set of normal distributions (The probability density is a mixture of normal distributions.)
- **Aim:** Find the parameters for the normal distributions and how much each normal distribution contributes to the data.



Two normal distributions.



Mixture model (both normal distrubutions contribute 50%).



Mixture model (one normal distrubutions contributes 10%, the other 90%).

- **Algorithm:** EM clustering (expectation maximization)
  Alternating scheme in which the parameters of the normal distributions and the likelihoods of the data points to be generated by the corresponding normal distributions are estimated

7.6. Density based Clustering

- For numerical data, density-based clustering algorithm often yield the best results
- Principle: A connected region with high data density corresponds to one cluster

DBScan

Principle Idea:

1. Find a data point where the data density is high, i.e. in whose ε-neighborhood is at least l other points (ε and l are parameters of the algorithm to be chosen by the user.)
2. All the points in the ε-neighborhood are considered to belong to one cluster
3. Expand this ε-neighborhood (the cluster) as long as the high density criterion is satisfied
4. Remove the cluster (all data points assigned to the cluster) from the data set and continue with 1. As long as data points with a high data density around them can be found

8. Finding Patterns: Item Sets

8.1. Association Rules: Motivation

- Association rule induction: Originally designed for market basket analysis
- Aims at finding patterns in the shopping behavior of customers
- More specifically: Find sets of products that are frequently bought together

- Assessing the quality of association rules:
  - **Support of an item set:**
    Fraction of transactions that contain the item set
  - **Support of an association rule $\boldsymbol{X \rightarrow Y}$:**
    Either: Support of $X \cup Y$ (more common: rule is correct)
    Or: Support of X (more plausible: rule is applicable)
  - **Confidence of an association rule $\boldsymbol{X \rightarrow Y}$:**
    Support of $X \cup Y$ divided by support of X (estimate of $P(Y|X)$)

- Two step implementation of the search for association rules:
  - Find the **frequent item sets** (also called *large item sets)*, i.e. the item sets that have at least a user-defined **minimum support**
  - From rules using the frequent item sets found and select those that have at least a user-defined **minimum confidence**

8.2. Frequent Item Set Mining

Given:

- a set $B = \{i_1, \ldots, i_m\}$ of items, the **item base**,
- a vector $T = (t_1, \ldots, t_n)$ of transactions over B, the **transaction database**,
- a number $s_{\min} \in \mathbb{N}$, $0 < s_{\min} \leq n$, or (equivalently)
  a number $\sigma_{\min} \in \mathbb{R}$, $0 < \sigma_{\min} \leq 1$, the **minimum support**.

Desired:

- the set of **frequent item sets**, that is,
  the set $F_T(s_{\min}) = \{I \subseteq B \mid s_T(I) \geq s_{\min}\}$ or (equivalently)
  the set $\Phi_T(\sigma_{\min}) = \{I \subseteq B \mid \sigma_T(I) \geq \sigma_{\min}\}$.

Note that with the relations $s_{\min} = \lceil n\sigma_{\min} \rceil$ and $\sigma_{\min} = \frac{1}{n} s_{\min}$ the two versions can easily be transformed into each other.

Searching for Frequent Item Sets

- Standard search procedure is an **enumeration approach,** that enumerates candidate item sets and checks their support
- Improves over the brute force approach by exploiting the **apriori property** to skip item sets that cannot be frequent because they have an infrequent subset
- The **search space** is the **partially ordered set**
- Structure of partially ordered set helps to identify those item sets that can be skipped due to the apriori property. ⇒ **top-down search** (from empty set/one-element sets to larger sets)

Breadth First Searching: The Apriori Algorithm

1. Determine the support of the one element item sets and discard the infrequent items
2. From candidate item sets with two items (both items must be frequent), determine their support, and discard the infrequent item sets
3. From candidate item sets with three items (all pairs must be frequent), determine their support, and discard the infrequent item sets
4. Continue by forming candidate item sets with four, five, etc. items until no candidate item set is frequent

- Scheme is based on two main steps: **candidate generation** and **pruning**
- all enumeration algorithms are based on these steps in some form

Canonical Forms

- A word is in canonical form, if the items (letters) are lexicographical sorted (e.g. abc but not bca)
- **Prefix Property:**
  - The longest proper prefix of the canonical code word of any item set is a canonical code word itself.
  - With the longest proper prefix of the canonical code word of an item set we now only know the canonical parent, but also its canonical code word.
  - Due to the prefix property the canonical code words of all child item sets that have to be explored in the recursion *with the exception of the last letter* (the added item) are already proofed as canonical code words. In a search scheme, only the new built words have to be proofed instead of each containing item set step by step.

Depth First Searches and Conditional Databases

- … can also be seen as a **divide-and-conquer scheme:**
  First find all frequent item sets that contain chosen item, then all frequent item sets that do not contain it.
- General search procedure:
  - Restrict the transaction database to those containing *a*. This is the **conditional database for the prefix** *a.*
  - Recursively search this conditional database for frequent item sets and add the prefix *a* to all frequent items sets found in the recursion.
  - Remove *a* from transactions in the full transaction database. This is the **conditional database for item sets without** *a*.
  - Recursively search this conditional database for frequent item sets.
- With this scheme only frequent one-element item sets have to be determined. Larger item sets result from adding possible prefixes.

Global and Local Item Order
P. 428?

The Eclat Algorithm

Maximal and Closed Item Sets

- An item set cannot have a lower support than any of its supersets (more specific set).
- An item set is **closed** if it is frequent, but none of its proper supersets has the same support.
- Every frequent item set has a **closed superset (with the same support)**.
- An item set is **maximal**, if it is frequent, but none of its proper supersets is frequent.
- Every frequent item set has a **maximal superset**.
- **All maximal item sets are closed**

9. Finding Explanations

Finding Explanations

- … for an unknown dependency within the data, instead of finding structure in a data set.
- Given Dataset $D = \{(x_i, Y_i) | i = 1, \dots, n\}$ with n tuples
  - x: Object description
  - Y: target attribute
    - Nominal: **classification problem**
    - Numerical: **regression problem**
- Data analysis
  - **Supervised** (because we know the desired outcome)
  - **Descriptive** (because we care about explanation)

(Un-) Supervised learning

- **Unsupervised learning:**
  - No class information given
  - Goal: detect unknown patterns
  - Cluster analysis and association rules (no specific target attribute)
- **Supervised learning:**
  - Class information exists/ is provided by supervisor
  - Goal: learn class structure for future unclassified/unknown data
  - a target attribute should be predicted based on other attributes

Eager vs. Lazy Learners

- **Lazy:** Save all data from training, use it for classifying (The learner was lazy, classifier has to do the work)
- **Eager:** Builds a (compact) model/structure during training, use model for classification. (The learner was eager/worked harder, classifier has a simple life)

9.1. Bayes Classifiers

- Classification problem because of a **nominal target attribute**
- … express their model in terms of simple probabilities
- Any other model should at least perform as well as the naïve Bayes classifier

Bayes' Theorem

$$P(h|E) = \frac{P(E|h) * P(h)}{P(E)}$$

- The probability *P(h|E)* that a hypothesis *h* is true given event *E* has occurred, can be derived from
  - *P(h)* the probability of the hypothesis *h* itself
  - *P(E)* the probability of the event *E*
  - *P(E|h)* the conditional probability of the event *E* given the hypothesis *h*

Bayes classifiers

- If we assume that every hypothesis $h \in H$ is equally probable a priori we can get the **maximum likelihood** hypothesis:

$$h_{ML} = \arg\max P(E|h)$$

- The probability *P(h)* can be estimated easily based on a given data set *D*:

$$P(h) = \frac{\#\text{ data from class } h}{\#\text{ data}}$$

- In principle, the probability *P(E|h)* could be determined analogously based on the values of the attributes $A_1, \dots, A_m$ i.e. the attribute vector $E = (a_1, \dots, a_m)$.

$$P(E|h) = \frac{\#\text{ data from class } h \text{ with values } (a_1, \dots, a_m)}{\#\text{ data from class } h}$$

- Naïve assumption: attributes $A_1, \dots, A_m$ are independent given the class, i.e.

$$P(E = (a_1, \dots, a_m)|h) = P(a_1|h) * \dots * P(a_m|h) = \prod_{a_i \in E} P(a_i|h)$$

- $P(a_i|h)$ can be computed easily:

$$P(a_i|h) = \frac{\#\text{ data from class } h \text{ with } A_i = a_i}{\#\text{ data from class } h}$$

Naïve Bayes classifier

- Is called **naïve** because of the (conditional) independence assumption for the attributes.
- Assumption is unrealistic, but the classifier yields good results, when not too many attributes are correlated.
- Based on the values $(a_1, \dots, a_m)$ of the attributes $A_1, \dots, A_m$ a prediction for the value of the attribute *H* should be derived:
- For each class $h \in H$ compute the likelihood $L(h|E)$ under the assumption, that the $A_1, \dots, A_m$ are independent given the class

$$L(h|E) \prod_{a_i \in E} P(a_i|h) * P(h)$$

- Assign E to the class $h \in H$ with the highest likelihood

$$pred(E) = \arg\max L(h|E)$$

- If a data set *D* does not contain any object with a given combination of values, a full Bayes classifier would not be able to classify this object

- **Treatment of missing values:**
  - During learning: The missing values are simply not counted for the frequencies of the corresponding attribute.
  - During classification: Only the probabilities (likelihoods) of those attributes are multiplied for which a value is available.

Laplace correction

- If a single likelihood is zero, then the overall likelihood is zero automatically, even when the other likelihoods are high.
- Solution: usage of Laplace correction γ:

$$P(y) = \frac{n_y}{n} \Rightarrow \hat{P}(y) = \frac{\gamma + n_y}{\gamma * |dom(Y)| + n}$$

$$P(y) = \frac{n_{yx}}{n_y} \Rightarrow \hat{P}(x|y) = \frac{\gamma + n_{yx}}{\gamma * |dom(X)| + n_y}$$

  - n # data
  - n_y # data from class *y*
  - n_{yx} # data from class *y* with value *x* for attribute *X*
  - $dom(X)$ # distinct values in *X*
  - Common choices for $\gamma = 1$ *or* $\gamma = \frac{1}{2}$

Full Bayes classifier

- Naïve Bayes classification for numerical data are equivalent to full Bayes classifier with diagonal covariance matrices
- Restricted to metric/numeric attributes (only the class is nominal/symbolic)
- Assumption: Each class can be described by a multivariable normal distribution
- Intuitively: Each class has a bell-shaped probability density

Pros

- Gold standard for comparison with other classifiers
- High classification accuracy in many applications
- Classifier cab easily be adapted to new training objects
- Integration of domain knowledge

Cons

- The conditional probabilities may not be available
- Independence assumptions might not hold for data set

9.2. Regression

- Find a function $f: \text{dom}(X_1) \times \ldots \times \text{dom}(X_k) \rightarrow Y$ minimizing the error $e(f(x_1, \ldots, x_k), y)$ for all given data objects $(x_1, \ldots, x_k, y)$

Regression line

- Given: A data set for two continuous attributes *x* and *y*.
- It is assumed that there is an approximate linear dependency between *x* and *y*:
$$y \approx a + bx$$
- Find a regression line (i.e. determinate the parameters a and b) such that the line fits the data as good as possible.

Cost functions

- Usually the **sum of errors in y-direction** is chosen as cost function (to be minimized)
- **Construction:** Given data $(x_i, y_i)(i = 1, \ldots, n)$, the least squares cost function is:
$$F(a, b) = \sum_{i=1}^{n} ((a + bx_i) - y_i)^2$$
  - Goal: The y-values that are computed with the linear equation should (in total) deviate as little as possible from the measured values.
- A necessary condition for a **minimum of the cost function** is that the partial derivatives of this function w.r.t. the parameters a and b vanish.

Least squares and MLE

- A regression line can be interpreted as a **maximum likelihood estimator (MLE)**
- Assumption: The data generation process can be described well by the model $y = a + bx + \varepsilon$, where ε is a normally distributed random variable with mean 0 and (unknown) variance σ^2
- The parameter that minimizes the sum of squared deviations (in y-direction) from the data points maximizes the probability of the data given this model class.

Nonlinear regression

- Minimization of the error function based on partial derivatives w.r.t. the parameters does not work in the other examples of error functions since
  - The absolute value and the maximum are not everywhere differentiable
  - The distance in the case of the Euclidean distance leads to system of nonlinear equation for which no analytical solution is known.
- Leading to nonlinear equations like: $y = a\, e^{bx}$
- Possible Solutions:
  - Iterative Methods (e.g. gradient descent)
  - Transformation of the regression function to a regression line or regression polynomial

Generalization

- Considering a data set as a collection of examples, describing the dependency between the predictor variables and the dependent variable, the regression function should "learn" this dependency from the data to **generalize** it in order to make correct predictions on new data.
- To achieve this, the regression function must be universal (flexible) enough to be able to learn the dependency. (does not mean, that a more complex function leads to better results than a simple one)

Overfitting

- function "learns" description of the data, not the structure inherent in the data
- The prediction using a complex function can be worse than for a simpler one.

Regression & nominal attributes

- If most of the predictor variables are numerical and few nominal attributes have small domains, regression function can be constructed for each possible combination of the values of the nominal attributes. (given that the data set is sufficiently large and covers all combinations)

- **Encode nominal attributes as numerical attributes:**
    - Binary attributes can be encoded as 0/1 or -1/1
    - Attributes with more than two values, a 0/1 or -1/1 numerical attribute should be introduced for each possible value of the nominal attribute
    - Do not encode nominal attributes with more than two values in one numerical attribute, unless the nominal attribute is actually ordinal

Classification as regression

- A two-class classification problem (with classes 0 and 1) can be viewed as regression problem
- Regression function will usually not yield exact outputs 0 and 1, but the classification decision can be made by considering 0.5 as a cut-off value
- Problem: objective functions aims at minimizing the function approximation error (e.g. the mean squared error), but not misclassifications

Logistic regression: Kernel estimation

- Given: set of data points $\{x_1, \dots, x_n\}$, each belongs to one the two classes c_1 and c_2
- Desired: simple description of the probability function p(x) for the given data set X
- For small data space with sufficient realizations for every possible point, the class probability can be estimated by the relative frequencies of the classes

- If this is not the case: **Kernel estimation**:
    - Idea: Define a "influence function" (kernel), which describes how strongly a data point influences the probability estimate for neighboring points

Pros

- Strong mathematical foundation
- Simple to calculate and to understand (for a moderate number of dimensions)
- High predictive accuracy

Cons

- Many dependencies are non-linear
- Global model does not adapt to locally different data distributions (Locally weighted regressions)

9.3. Decision Trees

- Attributes: Class C, other attributes $A^{(1)}, \dots, A^{(m)}$
- Data: $S = \{ (x_i, c_i) | i = 1, \dots, N \}$
- Find a interpretable model to understand dependency target attribute c_i and the input vectors x_i
- Model will not express necessarily the causal relationship, but only numerical correlations

Example

The Cupfactory

Given: Measurements about cup (density, production time, ...) (attributes)
and if the cup is broken or not (class attribute)
Goal: How can we know if the cup is broken based from the measurements?
e.g. if density is low than cup is broken

Decision Tree

- Find **hierarchical structure** to explain how different areas in the input space correspond to different outcomes
- Useful for data with a lot of unknown attributes of unknown importance

Classification with decision trees

- **Recursive Descent:**
  1. Start at the root node
  2. If the current node is an **leaf node**
     - Return the class assigned to the node
  3. If the current node is an **inner node**
     - Test the attribute associated with the node
     - Follow the branch labeled with the outcome of the test
     - Apply the algorithm recursively
- Intuitively: Follow the path corresponding to the case to be classified

Learning of decision trees

- **Top-down approach:**
  - Build the decisions tree from top to bottom (from root to leaves)
- **Greedy selection of a test attribute**
  - Compute an evaluation measure for all attributes
  - Select the attribute with the best evaluation
- **Divide and conquer/recursive descent**
  - Divide the example cases according to the values of the test attribute
  - Apply the procedure recursively on the subsets
  - Terminate the recursion if
    - All cases belong to the same class or
    - No more test attributes are available

Evaluation Measures

- **Rate of correctly classified example cases**
  - Advantage: simple to compute, easy to understand
  - Disadvantage: works well only for two classes
- If there are more than two classes, the rate of misclassified example cases **neglects a significant amount of the available information**
  - Only the majority class, that is the class occurring most often in a subset of example cases, is really considered
  - Distribution of the other classes has no influence.
  - Good choice of class can be important for deeper levels of the decision tree

- **Information gain** (based on Shannon entropy $H = -\sum_{i=1}^{n} p_i log_2 p_i$

$$I_{gain}(C, A) = H(C) - H(C|A)$$
$$= \sum_{i=1}^{n_C} p_i log_2 p_i - \sum_{j=1}^{n_A} p_j \left(\sum_{i=1}^{n_C} p_{i|j} \ log_2 p_{i|j} \right)$$

  - *H(C)* Entropy of the class distribution (C: class attribute)
  - *H(C|A)* expected entropy of the class distribution if the value of the attribute A becomes known
  - ... is biased towards many-valued attributes i.e., if two attributes have about the same information content it tends to select the one having more values
  - Normalization removes/reduces this bias: **Information gain Ratio**

- **Gini Index**
  - Rate of wrong classifications of a random data point (training set) on the basis of the distribution of the labels (training set)
  - Can be interpreted as expected error rate
- χ^2 **– measure**
  - Compares the actual joint distribution with *hypothetical independent distribution*
  - Uses absolute comparison
  - Can be interpreted as a difference measure

Treatment of numerical attributes (Greedy Approach)

- General Approach: Discretization
- **Preprocessing I:**
  - Form equally sized or equally populated intervals (binning)
- **Preprocessing II / Multisplits during tree construction:**
  - Build a decision tree using only the numeric attribute
  - Flatten the tree to obtain a multi-interval discretization
- Splits at boundary points minimize entropy
- For binary splits (only one cut point) all boundary points are considered and the one with the smallest entropy is chosen
- For multiple splits a recursive procedure is applied

Missing Values

- Replace with most frequent value
- Replace with most frequent value of the same class

- Replace with all possible values and propagate examples with the corresponding weights

Pruning decision trees

- … **serves the purpose:**
  - To simplify the tree (improve interpretability)
  - To avoid Overfitting (improve generalization)

- **Basic idea:**
  - Replace “bad” branches (subtrees) by leaves
  - Replace subtree by its largest brand if it is better

- **Reduced error pruning**
  - Classify a set of **new example cases** with the decision tree (These cases must not have been used for learning!)
  - Determine the **number of errors** for all leaves
  - The number of errors of a subtree is the sum of the errors of all of its leaves
  - Determine the number of errors for leaves that replace subtrees
  - If such a leaf leads to the same or fewer errors than the subtree replace the subtree by the leaf
  - If a subtree has been replaced, recompute the number of errors of the subtrees it is part of
  - **Advantage:**
    - Very good pruning
    - Effective avoidance of overfitting
  - **Disadvantage:**
    - Additional example cases needed
    - Number of cases in a leaf has no influence

- **Pessimistic pruning**
  - Classify a set of example cases with the decision tree. (These cases may or may not have been used for the learning)
  - Determine the number of errors for all leaves and **increase this number by a fixes, user-specified amount *r***
  - The number of errors of a subtree is the sum of the errors of all of its leaves
  - Determine the number of errors for leaves that replace subtrees (also increased by *r*)
  - If such a leaf leads to the same or fewer errors than the subtree, replace the subtree by the lead and recompute subtree errors
  - **Advantage:**

        - No additional example cases needed
    - **Disadvantage:**
        - Number of cases in a leaf has no influence

- **Confidence level pruning**
    - Like pessimistic pruning, but sees classification in a lead as a Bernoulli experiment (error / no error)
    - **Advantage:**
        - No additional example cases needed
        - Good pruning
    - **Disadvantages:**
        - Statistically dubious foundation

Predictive vs. descriptive tasks

- Predictive tasks:
  The decision tree (or more generally, the classifier) is constructed in order to apply it to new unclassified data
- Descriptive tasks:
  The purpose of the tree construction is to understand, how the classification has been carried out so far.

Regression Trees

- Like decision trees, but target variable is not a class, but a numeric quantity
- Simple regression trees: predict constant values in leaves
- More complex regression trees: predict linear functions in leaves

9.4. Rules

Propositional Rules

- Also: truth bearer or statement rules
- … are either true or false (under an interpretation)
- Only atomic facts
    - Constraints on numerical attributes, e.g. $<, \leq, =, \ldots$
    - Constraints on nominal attributes, e.g. $=, \in set$
    - Constraints on ordinal attributes, e.g. $<, \in set, \in range$
- Combinations of facts using only logical operators

- No variables (in contrast to first order rules)

Learning Propositional Rules

Categorization of more general Propositional Rule Learners:

- Supported attribute types
  - Nominal only ⇒ relatively small hypothesis space
  - Numerical only ⇒ "geometrical" rule learners
  - Mixed attributes ⇒ more complex heuristic needed
- Learning strategy
  - Specializing
    - Start with very general rules
    - Iteratively specialize the rule
  - Generalizing:
    - … existing rule to cover one more pattern
    - Merge two existing rules (the more general case)
    - Training algorithm: greedy (complete search of merge tree infeasible)
    - Training algorithms differ in (… choice of rules/patterns to merge; … used stopping criteria)
  - Most real world data sets are too complex to be explained by one rule only
  - Many rule learning algorithms wrap the learning of one rule into an outer loop based on set covering strategy (**sequential covering**):
    - Attempts to build most important rules first
    - Iteratively adds smaller/less important rules

Issues with most propositional rule learners:

- Heuristics for real world data often fail
  - Too many dimensions make greedy constraint picking difficult
  - In real world feature spaces often patterns of same class not "axes parallel"
- Sharp boundaries for noisy data unsuited
  - Fuzzy Rules
  - Hierarchical Rule Systems
- Propositional rules not very expressive

Geometrical rule learners

- Limited to numerical attributes (comparable ranges help, too)
- Goal:

- Fin rectangular (axes parallel) area(s) one by one that are occupied only by patterns of one class
- Each such area represents a rule

- Creates one rule after the other until no more useful rules can be built. To find one rule:
  - Draw random starting point
  - From most specific rectangular hypothesis covering this point
  - While possible
    - Find nearest neighbor of same class
    - Generalize hypothesis (rectangle) to include this point (the latter may not be possible for all neighbors of the same class)

- RecBF learner
  - Goal: find specific and general rules (allows to estimate classification certainy)
  - Algorithm motivated by neural network learning method
  - Finds most specific within each (locally) most general rule
  - Iterative algorithm
  - (much) faster than rule-by-rule approach
  - Problems:
    - Depends on order of training examples
    - Shrink-procedure based on heuristics
  - Properties:
    - Explains all training pattern correctly
    - Most general rules: depend on a few attributes only
    - Most specific rules: depend on all attributes

Decision tree based rule finding

Rules from Decision Tree are:

- Mutually exclusive (and therefore also conflict free)
- Unordered
- Complete
- Instable (inherited from the heuristic decision tree algorithms)
- Redundant
  - due to recursive nature of the trees
  - constraints on splits appear in several rules)
- More balanced tree results in nested ordering of rules.

Good rule finding

- Following (more than) one branch resembles a general-to-specific beam search:
  - Hypothesis (rule) is **increasingly specified**
  - At each branching point **all possible specializations** are considered
  - Choice of best branch can be made based on **information gain, coverage,** or mix of both
  - Class (rule post condition) depends on **majority class** at each point
- **Greedy algorithm** can produce sub-optimal solution
  - beam search produces k candidate solution
  - at each level only the k best branches are kept
- Sometimes only learning rules of one class is sufficient (e.g. to model a minority class), default class = majority class

First order rules

.. differ from propositional logic by their use of variables. FOL is also based on a number of base constructs:

- **Constants**, e.g. Bob, Luise, red, green
- **Variables**, e.g. x and y
- **Predicates**, e.g. is_father(x,y), which produces truth values as a result
- **Functions**, e.g. age(x), which produce constants

Using constants variables, predicates and functions we can construct:

- **Terms:** constants, variables or functions applied to a term
- **Literals:** predicates (nor negation of predicates) applied to any set of terms
- **Ground literals:** literals that do not contain a variable
- **Clauses:** disjunctions of literals whose variables are universally quantified
- **Horn clauses:** clauses with at most one positive literal

- A rule body is satisfied if a least one binding exists that satisfies all literals.
- Binding of variables = substitution of variables by appropriate constants

Learning first order rules: FOIL

- FOIL (First Order Inductive Learning method) was one the first algorithms published
- One (dramatic) restriction: no functions!
- One (easy) extension: literals in body can be negated

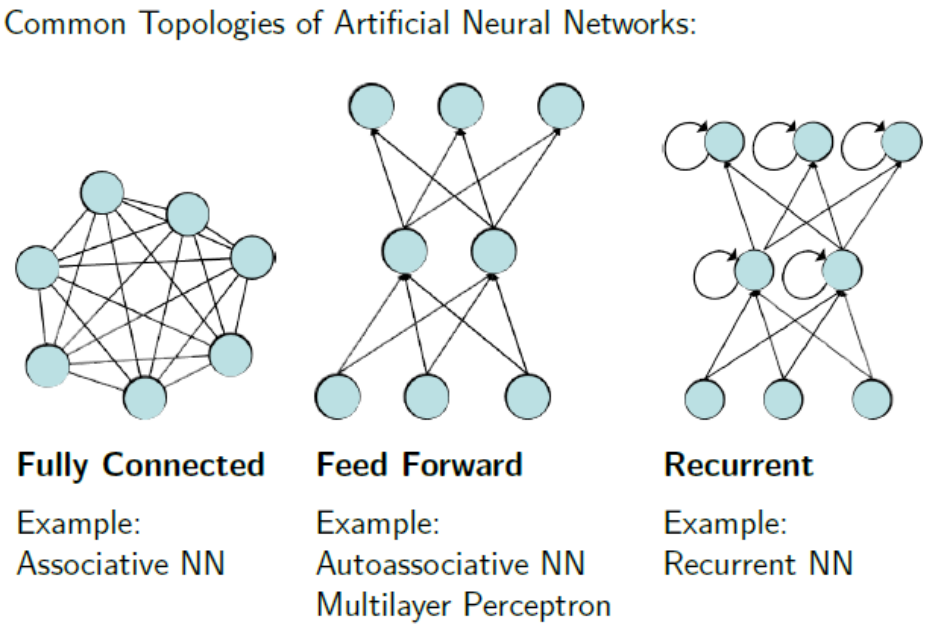
- Combines:
  - Outer specific-to-general search (additional rules add positive coverage)
  - Inner general-to-specific search (rules are specialized by adding literals)
- … can also learn recursive concepts (somewhat tricky since infinite recursion needs to be avoided)
- Extensions of FOIL also handle noisy data (Stopping criteria considers description-length principle to avoid adding lengthy rules to explain few examples)

- Inductive Logic Programming allows to learn concepts from multi relational data bases (no prior integration necessary)
- Most existing algorithms scale poorly (FOIL is a good example)
- Applicable for special types of data (e.g. structured or mainly nominal values)

10. Finding Predictors

10.1. Artificial Neural Networks

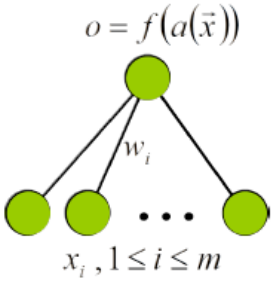
Neural Networks: big but slow



- **A Neuron:**
  - … binary switch, being either active or inactive
  - Each neuron has a fixed threshold value
  - … receives input signals from excitatory (positive) or inhibitory (negative) synapses (connections to other neuron)

- The inputs are accumulated (integrated) for a certain time. When the threshold value of the neuron is exceeded, the neuron becomes active and sends signals to neighboring neurons via its synapses.

- **The perceptron (by Rosenblatt):**
  - Aim: Automatic learning of weights and the threshold to classify objects, which have been shown on the retina correctly.
  - A simplified retina is equipped with receptors (input neurons), which are activated by an optical stimulus
  - The stimulus is passed on to an output neuron via a weighted connection (synapse).
  - When the threshold θ of the output neuron is exceeded, the output is 1, otherwise 0.



$$a(\mathbf{x}) = \sum_{i=1}^{m} w_i \cdot x_i \qquad o = f(a) = \begin{cases} 1 & \text{if } a > \theta \\ 0 & \text{otherwise} \end{cases}$$

  - (Numerical) input attributes A_i, output class (0 or 1)
  - Classifier for two-class problem
  - For multiclass problems use one perceptron per class

Perceptron learning algorithm

- Initialize the weights and the threshold value randomly
- For each data object in the training data set, check whether the perceptron predicts the correct class.
- If the perceptron predict the wrong class, adjust weights and threshold value
- Repeat this until no changes occur.
- **Delta Rule:** Whenever the perceptron makes a wrong classification change weights and threshold in "appropriate direction".
  - If the desired output is 1 and the perceptron's output is 0, the threshold

$$\Delta w_i = \begin{cases} 0 & \text{if } t_p = t \\ +\sigma a_i & \text{if } t_p = 0 \text{ and } t = 1 \\ -\sigma a_i & \text{if } t_p = 1 \text{ and } t = 0 \end{cases}$$

$$\Delta \theta = \begin{cases} 0 & \text{if } t_p = t \\ -\sigma & \text{if } t_p = 0 \text{ and } t = 1 \\ +\sigma & \text{if } t_p = 1 \text{ and } t = 0 \end{cases}$$

- w_i: A weight of the perceptron
- θ: The threshold value of the output neuron
- $(a_1, \ldots, a_n)$: An input
- t: desired output for input $(a_1, \ldots, a_n)$
- t_p: The perceptron's output for input $(a_1, \ldots, a_n)$
- $\sigma > 0$: Learning rate

is not exceeded, although it should be. Therefore, lower the threshold and adjust weights depending on the sign and magnitude of the inputs.
  - If the desired output is 0 and the perceptron's output is 1, the threshold is exceeded, although it should not be. Therefore increase the threshold and adjust the weights depending on the sign and magnitude of the inputs.

- **Perceptron convergence theorem:**
  If, for a given data set with two classes, there exists a perceptron that can classify all patterns correctly, then the delta rule will adjust the weights and the threshold after a finite number of steps in such way that all patterns are classified correctly.

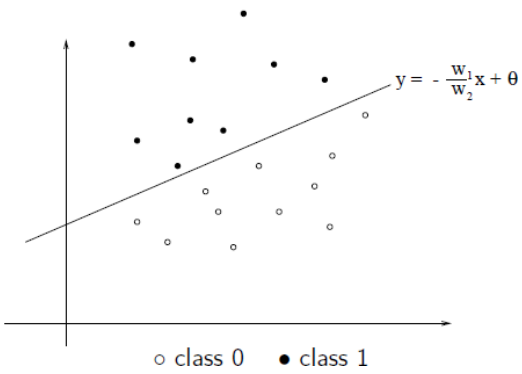
Linear separability

- Consider a perceptron with two input neurons. Let t_p be the output of the perceptron for input (a_1, a_2). Then:

$$t_p = 1 \Leftrightarrow w_1 * a_1 + w_2 * a_2 > \theta$$
$$\Leftrightarrow a_2 > -\frac{w_1}{w_2} a_1 + \frac{\theta}{w_2}$$

- The perceptron's output is 1 if the input pattern (a_1, a_2) is above the line

$$y = -\frac{w_1}{w_2} x + \frac{\theta}{w_2}$$



- The parameters w_1, w_2, θ determine the line. All input patterns above this line are assigned to class 1, the input patterns below the line to class 0.

- **In hyperspace:** Perceptrons represent hyperplanes in feature space
- A perceptron with n input neurons can classify all examples from a data set with n variables and two classes correctly if there exists a hyperplane separating the two classes. (linearly separable)

Perceptron with a hidden layer of neurons

- The hidden layer carries out a transformation. The output neuron can solve the linearly separable problem transformed space.
- Adjust weights and threshold with help of multilayer perceptrons with gradient descent
  - Does not work with binary (non-differentiable) threshold function as activation function for the neuron
  - Must be replaced by a differentiable function

- **Multilayer perceptrons (MLP):**
  - … neural network with an input layer, one or more hidden layers and an output layer
  - Connections exist only between neurons from one layer to the next layer.
  - Activations functions for neurons are usually sigmoidal functions

Backpropagation

- **Error Backpropagation:**
  Recursive equation for updating the weights: Update the weights to the neuron in the output layer first and then go back layer by layer and update the corresponding weights.
  … in four steps (for MLP):
  1. Forward Phase: Calculate output for all neurons (including hidden ones) from input to output layer.
  2. Backward Phase: calculate deltas for all neurons from output to input layer.
  3. Weight update: Use deltas and output values to calculate the weight updates from output to input layer.
  4. Batch weight update: Update all weights at once.

- ... as a gradient descent technique can only find a local minimum. Training the networks with different random initializations will usually lead to different results.
- Usually, weights are not updated after a whole epoch, i.e. after all patterns have been presented once (called offline training), but after the presentation of each input pattern (online training). This is usually faster, although not formally the same and potentially risky (oscillations).
- There is no general rule, how to choose the number of hidden layers and the size of the hidden layers. Small neural networks tend to overfitting.
- The steepness of the activation function is usually fixed and is not adjusted.
- The multilayer perceptron learns only in those regions where the activation function is not close to zero or one, otherwise the derivate is almost zero.

10.2. Support Vector Machines (Kernel Machines)

Motivation

- **Main Idea:**
  - Embed data into suitable vector space
  - Find linear classifier (or other linear pattern of interest) in new space

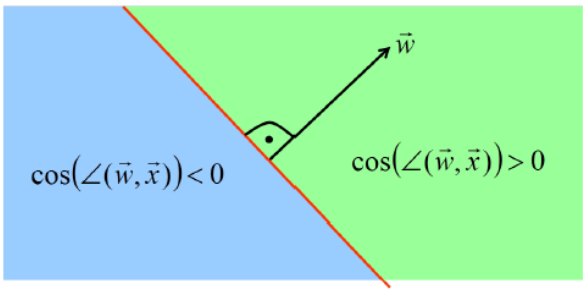
- **Needed:** a Mapping $\Phi: x \in X \mapsto \Phi(x) \in F$
- **Key assumptions:**
  - Information about relative position is often all that is needed by learning methods
  - The inner products between points in the projected space can be computed in the original space using special functions (kernels)

Linear Discriminant

Simple linear, binary classifier:
$f(x) = \langle x, w \rangle + b = \sum_{i=1}^{n} x_i w_i + b$

- Class A if *f(x)* positive
- Class B if *f(x)* negative

$$f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b = \sum_{i=1}^{n} x_i w_i + b = b + |\mathbf{x}|\,|\mathbf{w}| \cos\left(\angle(\mathbf{w}, \mathbf{x})\right)$$



- … represents hyperplanes in feature space

Rosenblatt Algorithm

- On-line (pattern by pattern approach)
- Mistake driven (updates only in case of wrong classification)
- … converges guaranteed if a hyperplane exists which classifies all training data correctly (data is linearly separable)
- Learning rule:

$$\text{IF } y_j \cdot (\langle \mathbf{w}, \mathbf{x}_j \rangle + b) < 0 \text{ THEN } \begin{cases} \mathbf{w}(t+1) = \mathbf{w}(t) + y_j \cdot \mathbf{x}_j \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

- Weight vector (if initialized properly) is simply a weighted sum of input vectors (b is even more trivial)

Dual Representation

- Weight vector is a weighted sum of input vectors:

$$w = \sum_{j=1}^{m} \alpha_j y_j x_j$$

- "difficult" training patterns have larger alpha
- "easier" ones have smaller or zero alpha

- Learning rule:

$$\text{IF } y_j \cdot \left(\sum_{j=1}^{n} \alpha_j y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b \right) < 0 \text{ THEN } \begin{cases} \alpha_i(t+1) = \alpha_i(t) + 1 \\ b(t+1) = b(t) + y_j \cdot R^2 \end{cases}$$

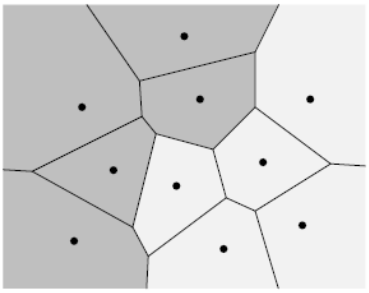
- Harder to learn examples having larger alpha (higher information content)
- Information about training examples enters algorithm only through the inner products (which we could pre-compute)

Kernels

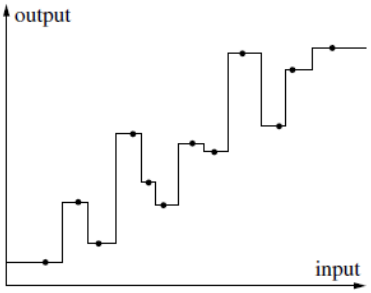
- We do not need to know the projection Φ It is sufficient to prove that K is a Kernel.
- … are modular and closed: we can compose new Kernels based on existing ones
- … can be defined over non numerical objects
    - Text: e.g. string matching kernel
    - Images, trees, graphs, …
- A good Kernel is crucial
    - Gram matrix diagonal: classification easy and useless
    - No free Kernel: too many irrelevant attributes (Gram matrix diagonal)

10.3. Nearest Neighbor

- Nearest neighbor predictors are special case of **instance-based learning**
- Instead of constructing a model that generalizes beyond the training data, the training examples are merely stored
- Predictions for new cases are derives directly from these stored examples and their (known) classes or target values



Classification



Regression

- For a new instance use the target value of the closes neighbor in the training set
- Problems with noisy data

k-nearest neighbor predictor

- Instead of relying for the prediction on only one instance, the (single) nearest neighbor, usually the *k (k > 1)* are taken into account, leading to the k-nearest neighbor predictor
- **Classification:** Choose the *majority class* among the k nearest neighbors for prediction
- **Regression:** take the *mean value* of the k nearest neighbors for predictors
- **Disadvantage:**
    - All *k* nearest neighbors have the same influence on the prediction
    - Closer nearest neighbor should have higher influence on the prediction

- **Distance Metric:**
  The distance metric, together with a possible task-specific scaling or weighting of the attributes, determines which of the training examples are nearest to a query data point and this selects the training example(s) used to produce a prediction.
    - Problem dependent
    - Often Euclidean distance (after normalization)
- **Number of Neighbor:**
  The number of neighbors of the query point that are considered can range from only one (the basic nearest neighbor approach) through a few (like k-nearest neighbor approaches) to, in a principle, all data points in an extreme case.
    - Very often chosen on the basis of cross validation
    - Choose k that leads to the best performance for cross-validation
- **Weighting function for the neighbors:**
  Weighting function defined on the distance of a neighbor from the query point, which yields higher values for smaller distances.
    - E.g. tricubic weighting function
- **Prediction function:**
  For multiple neighbors, one needs a procedure to compute the prediction from the (generally differing) classes of target values of these neighbors, since they may differ and thus may not yield a unique prediction directly.
    - **Regression:**
        - Compute the weighted average of the target values of the nearest neighbors.
    - **Classification:**
        - Sum up the weights for each class among the nearest neighbors
        - Choose the class with the highest value

Adjusting the distance function

- choice of the distance function is crucial for the success of a nearest neighbor approach
- one way to adept the distance function is **feature weights** to put a stronger emphasis on those feature that are more important
- A configuration of feature weights can be evaluated based on crass-validation
- The optimization of the feature weights can be carried out based on some heuristic strategy like hill climbing, simulated annealing or evolutionary algorithms.

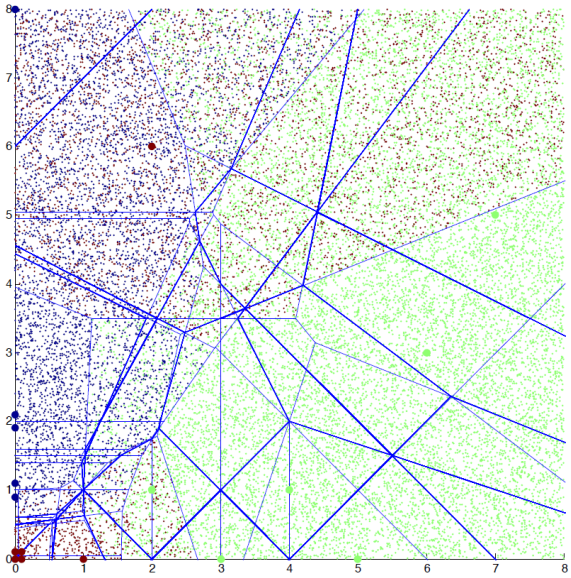
Data set reduction, prototype building

- **Advantage** of the nearest neighbor approach:
  No time for training is needed - at least when no feature weight adaption is carried out or the number of nearest neighbors is fixed in advance
- **Disadvantage** of the nearest neighbor approach:
  Calculation of the predicted class or value can take ling when the data set is large

Possible solutions:

- Finding a **smaller subset of the training set** for the nearest neighbor predictor
- Building prototypes by **merging (close)** instances, for instance by averaging

Choice of parameter k



- k = 1 yields y = piecewise constant labeling
- “too small” k: very sensitive to outliers
- “too large” k: many objects from other clusters (classes) in the decision set
- k = N predicts y = globally constant (majority) label
- The selection of k depends from various input “parameters”:
  - The size of the data set
  - The quality if the data
- k = 1: highly localized classifier, perfectly fits separable training data
- k > 1:
  - the instance space partition refines
  - More segments are labeled with the same local models.

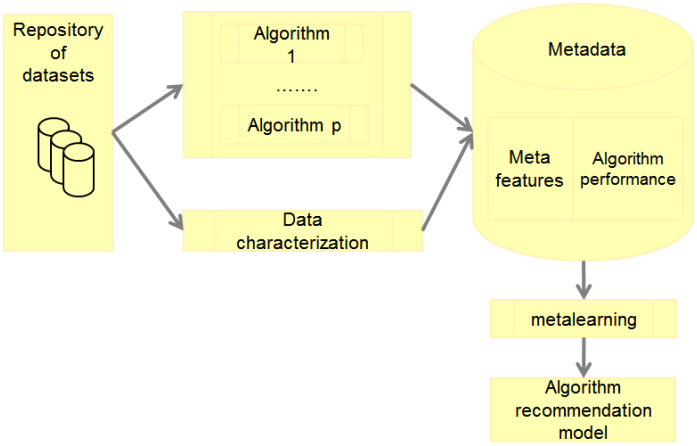
10.4. Meta Learning (machine Learning)

- Exploit **Wisdom of crowd** ideas for specific tasks:
  - combining (classifier) predictions
  - aim to combine independent and diverse predictors (Classifiers)

- Use of labeled training data:
  - To identify expert classifiers in the pool
  - To identify complementary classifiers
  - To indicate how to best combine them

Basic terms

- Bias = model error + algorithmic error
  - Model error: the error we get by selecting a model
  - Algorithmic error: by selecting the algorithm itself and parameters of the algorithm
- Base Learning: Fixed Bias / User parameterized
- Meta-Learning: Dynamic bias selection using meta knowledge
- Meta-Knowledge: Knowledge achieved during learning process



Bagging and Boosting

- Best known techniques
- Based on selection of multiple data sub sets
- Meta model is created by combining the base models
- Always uses the same base learner
- Advantages:
  - Reduces overfitting
  - Most efficient when the base learner is highly sensitive to data
  - Typically increases accuracy
- Disadvantages:
  - Interpretability of interpretable base learners is lost

- **Bagging:**
  - Select N independent samples of the training data
  - Learn one model on each of the samples
  - Classification: use the class most predicted by all classifiers
  - Regression: use the mean of all predictions

- **Boosting:**
  - Tries to learn a weighting for the models
  - Later weak learners focus more on the examples that previous weak learners misclassified
  - There is no single "best" boosting method
  - One boosting method after Schapire:
    - Training:
      - Create c_1: weak learner on a sample t_1 of the data
      - Create t_2: sample which is 50% miss classified by c_1
      - Create c_2: weak learner on the sample t_2
      - Create t_3: subset of the data where c_1 predicts differently than c_2
      - Create c_3: weak learner on the sample t_3
    - Classification:
      - Classify with c_1 and c_2
      - If unequal, use c_3 as final classification

Stacking and Cascade Generalization

- **Stacking:**
  - … uses differences among learners
  - Two levels of learning:
    1. Base learners are trained, each on the whole data set
    2. Meta learners are created on meta data (e.g. predicted class) obtained in level 1
  - Two levels of classifying
    1. Base learners are used on data point
    2. Meta learners are applied on base learner predictions

- **Cascade Generalization:**
  - Base learners are used in a sequence with "partial" meta learners
  - Knowledge from previous classifiers can be used in later ones
  - After each base learner the data set is adjusted using new information
  - For classification:

        - Only the last model is used
        - Which incorporates the knowledge of previous models (all base methods are used)

Cascading and Delegating

- Until now: all base classifiers are used for classification
- Here: multistage classifiers, not all are required for classification
- Main advantage: faster classification

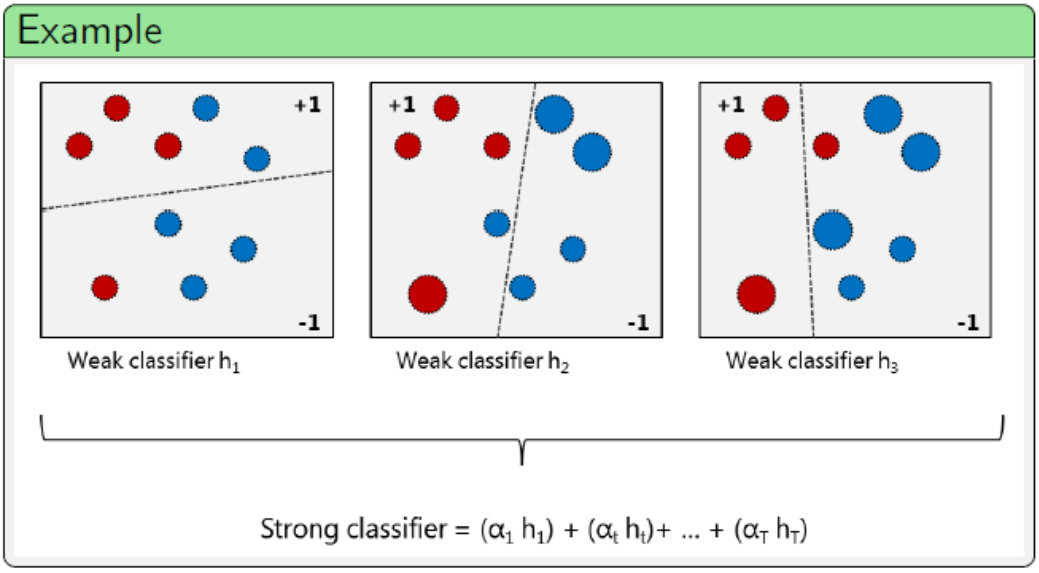
- **Cascading:**
    - Multilearner version of boosting
    - Uses learned confidence of previous models
    - Train base learner using knowledge from previous base learner…
    - … on data, which was most probably misclassified by previous learners
    - Classify: go through all base models, stop and use as classification if the model has a confidence greater than epsilon

- **Delegating:**
    - Cascading: all instances are used in each step
    - Delegating: only instances below confidence threshold are processed in the next step
    - Idea:
        - Use everything and test for which data points you are good enough
        - Pass the remaining work to someone else
        - If there is no someone else, … guess
    - Advantages:
        - Still understandable (no model combination)
        - Improved efficiency due to the decreasing number of examples

Ada Boost

- Is a linear classification algorithm
- Has good generalization properties (Avoids overfitting as long as the training data is not too noisy)

- Is a feature selector



- Advantages:
  - Very simple to implement
  - Feature selection on very large feature spaces
  - Fairly good generalization
- Disadvantages:
  - Can overfit in presence of noise
  - Unclear which weak-learning algorithm fits best for a given problem