Selected Topics of Pervasive Computing

Stephan Sigg

Georg-August-University Goettingen, Computer Networks

06.11.2013

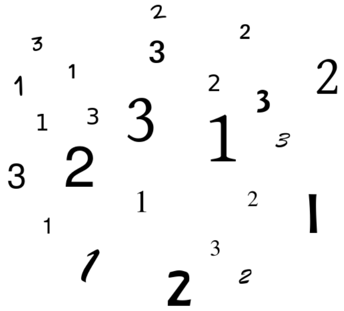
Overview and Structure

30.10.2013	Organisational
30.10.3013	Introduction
06.11.2013	Classification methods (Feature extraction, Metrics, machine learning)
13.11.2013	Classification methods (Basic recognition, Bayesian, Non-parametric)
20.11.2013	–
27.11.2013	–
04.12.2013	–
11.12.2013	Classification methods (Linear discriminant, Neural networks)
18.12.2013	Classification methods (Sequential, Stochastic)
08.01.2014	Features from the RF channel (Effects of the mobile radio channel)
15.01.2014	Security from noisy data (Error correcting codes, PUFs, Applications)
22.01.2014	Context prediction (Algorithms, Applications)
29.01.2014	Networked Objects (Sensors and sensor networks, body area networks)
05.02.2014	Internet of Things (Sensors and Technology, vision and risks)

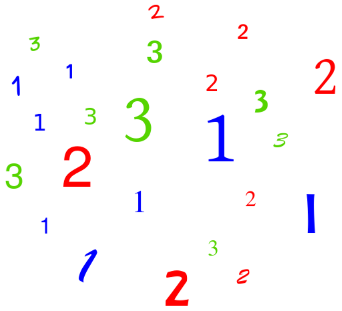
Outline

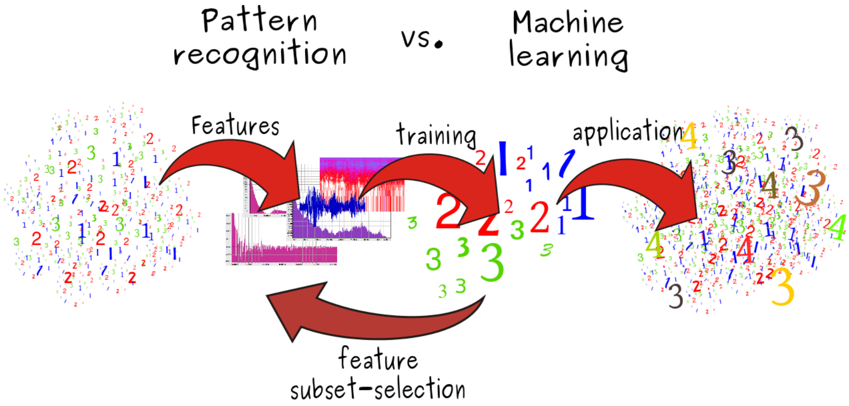
Pattern recognition vs. Machine learning

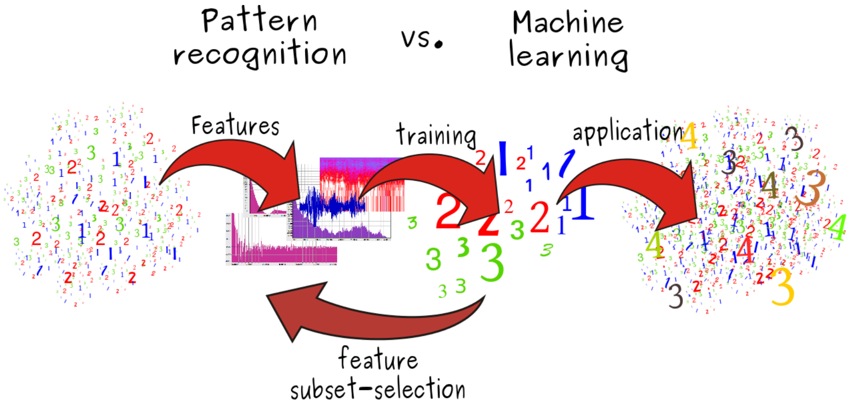
Pattern recognition vs. Machine learning



Pattern recognition vs. Machine learning



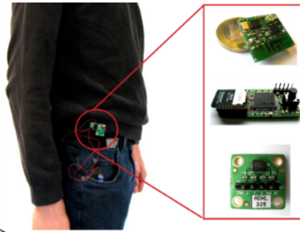

Pattern recognition
vs.
Machine learning
Features
training
application
feature subset-selection



- Mapping of features onto classes by using prior knowledge
- What are characteristic features?
- Which approaches are suitable to obtain these features?

Data sampling

- Record sufficient training data
  - Annotated! (Ground-truth)
  - Multiple subjects
  - Various environmental conditions (time of day, weather, ...)

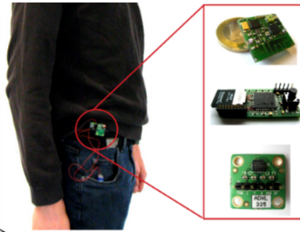


Data sampling

- Record sufficient training data
  - Annotated! (Ground-truth)
  - Multiple subjects
  - Various environmental conditions (time of day, weather, ...)

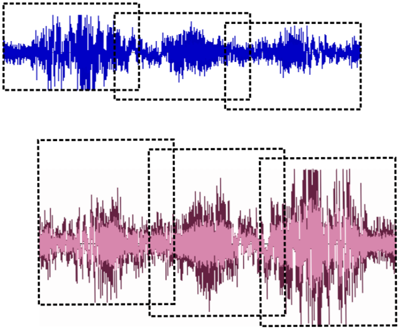
Example

- Electric supply data over 15 years covers 5000 days but only 15 christmas days
- Especially critical events like accidents (e.g. plane, car, earthquake) are scarce



Feature subset-selection

- Pre-process data
  - Framing
  - Normalisation



Feature subset-selection

- Pre-process data
  - Framing
  - Normalisation

Domain knowledge?
$\rightarrow$ better set of ad-hoc features

Features commensurate?
$\rightarrow$ normalise

Pruning of input required?
$\rightarrow$ if no, create disjunctive features or weithted sums of features

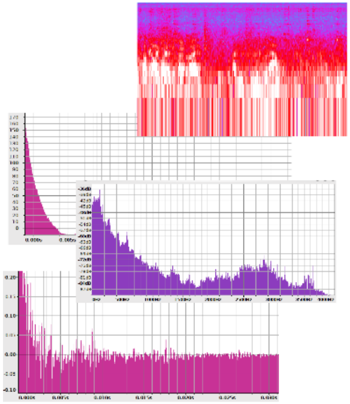
Independent features?
$\rightarrow$ construct conjunctive features or products of features

Is the data noisy?
$\rightarrow$ detect outlier examples

Do you know what to do first?
$\rightarrow$ If not, use a linear predictor

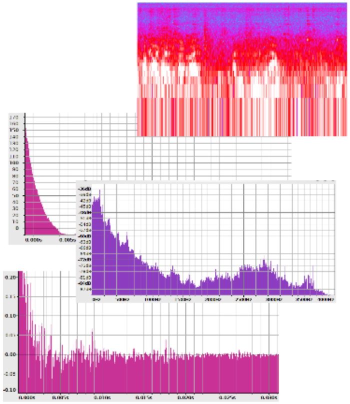
Feature extraction

- Identify meaningful features
    - remove irrelevant/redundant features



Feature extraction

- Identify meaningful features
  - remove irrelevant/redundant features
- Features can be contradictory!



Feature subset-selection

Simple ranking of features with correlation coefficients

Example: Pearson Correlation Coefficient

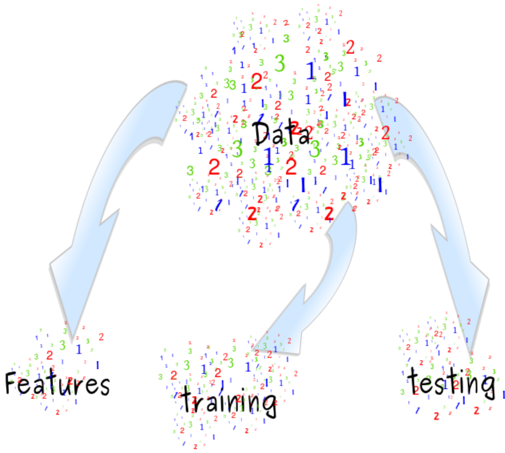
$$\varrho(X, Y) = \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}} \tag{1}$$

- Identifies linear relation between input variables x_i and an output y

Feature subset-selection

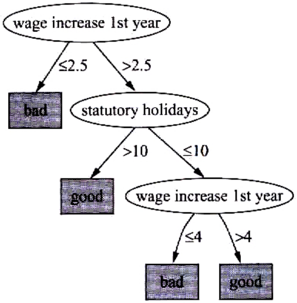
How to do reasonable feature selection

- Utilise dedicated test- and training- data-sets
- Pay attention that a single raw-data sample could not impact features in both these sets
- Don't train the features on the training- or test- data-set

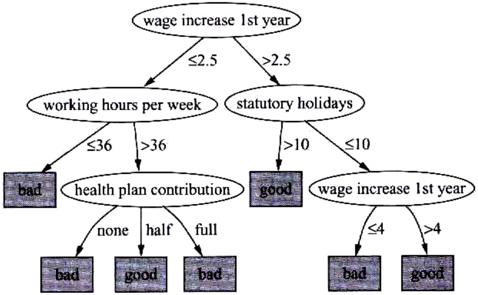


Training of the classifier

A decision tree classifier



(a)



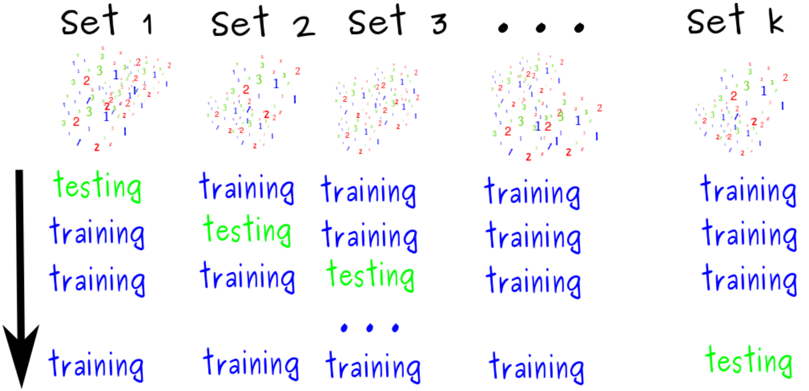
(b)

Training of the classifier

Evaluation of classification performance

k-fold cross-validation

- Standard: k=10

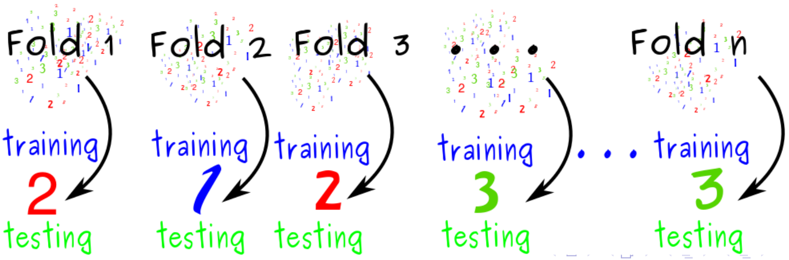


Training of the classifier

Evaluation of classification performance

Leave-one-out cross-validation

- n-fold cross validation where n is the number of instances in the data-set
- Each instance is left out once and the algorithm is trained on the remaining instances
- Performance of left-out instance (success/failure)

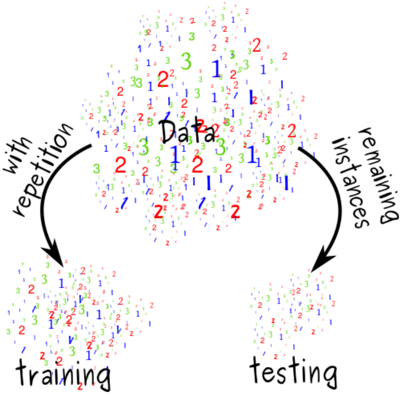


Training of the classifier

Evaluation of classification performance

0.632 Bootstrap

- Form training set by choosing n instances from the data-set <u>with replacement</u>
- All not picked instances are used for testing
- Probability to pick a specific instance: $1 - \left(1 - \frac{1}{n}\right)^n \approx 1 - e^{-1} \approx 0.632$

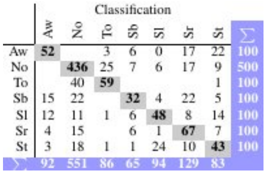


Training of the classifier

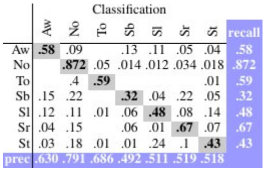
Evaluation of classification performance

Classification accuracy

- Confusion matrices
- Precision
- Recall



	Classification							
	Aw	No	To	Sb	Sl	Sr	St	∑
Aw	**52**		3	6	0	17	22	100
No		**436**	25	7	6	17	9	500
To		40	**59**				1	100
Sb	15	22		**32**	4	22	5	100
Sl	12	11	1	6	**48**	8	14	100
Sr	4	15		6	1	**67**	7	100
St	3	18	1	1	24	10	**43**	100
∑	92	551	86	65	94	129	83	



	Classification							
	Aw	No	To	Sb	Sl	Sr	St	recall
Aw	**.58**	.09		.13	.11	.05	.04	.58
No		**.872**	.05	.014	.012	.034	.018	.872
To		.4	**.59**				.01	.59
Sb	.15	.22		**.32**	.04	.22	.05	.32
Sl	.12	.11	.01	.06	**.48**	.08	.14	.48
Sr	.04	.15		.06	.01	**.67**	.07	.67
St	.03	.18	.01	.01	.24	.1	**.43**	.43
prec	.630	.791	.686	.492	.511	.519	.518	

Training of the classifier

Evaluation of classification performance

Information score

Let C be the correct class of an instance and $\mathcal{P}(C)$, $\mathcal{P}'(C)$ be the prior and posterior probability of a classifier
We define:[1]

$$I_i = \begin{cases} -\log(\mathcal{P}(C)) + \log(\mathcal{P}'(C)) & \text{if } \mathcal{P}'(C) \geq \mathcal{P}(C) \\ -\log(1-\mathcal{P}(C)) + \log(1-\mathcal{P}'(C)) & \text{else} \end{cases} \tag{2}$$

The information score is then

$$\text{IS} = \frac{1}{n}\sum_{i=1}^{n} I_i \tag{3}$$

[1] I. Kononenko and I. Bratko: Information-Based Evaluation Criterion for Classifier's Performance, Machine Learning, 6, 67-80, 1991.

Training of the classifier

Evaluation of classification performance

Brier score

The Brier score is defined as

$$\text{Brier} = \sum_{i=1}^{n} (t(x_i) - p(x_i))^2 \tag{4}$$

where

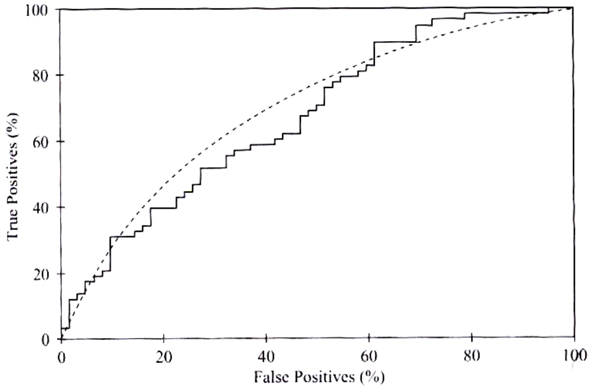
$$t(x_i) = \begin{cases} 1 & \text{if } x_i \text{ is the correct class} \\ 0 & \text{else} \end{cases} \tag{5}$$

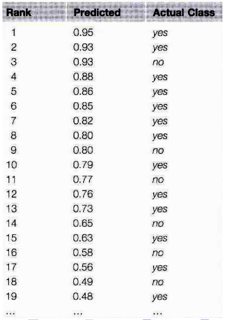
and $p(x_i)$ is the probability the classifier assigned to the class x_i.

Training of the classifier

Evaluation of classification performance

Area under the receiver operated characteristic (ROC) curve (AUC)





Rank	Predicted	Actual Class
1	0.95	*yes*
2	0.93	*yes*
3	0.93	*no*
4	0.88	*yes*
5	0.86	*yes*
6	0.85	*yes*
7	0.82	*yes*
8	0.80	*yes*
9	0.80	*no*
10	0.79	*yes*
11	0.77	*no*
12	0.76	*yes*
13	0.73	*yes*
14	0.65	*no*
15	0.63	*yes*
16	0.58	*no*
17	0.56	*yes*
18	0.49	*no*
19	0.48	*yes*
...	...	...

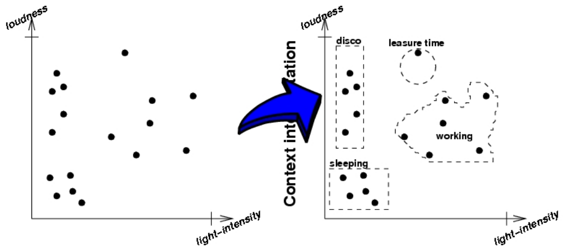
Pattern recognition and classification

Data mining frameworks

- Orange Data Mining (http://orange.biolab.si/)
- Weka Data Mining (http://www.cs.waikato.ac.nz/ml/weka/)

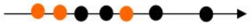
Pattern recognition and classification

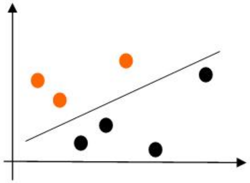
- From features to context
  - Measure available data on features
  - Context reasoning by appropriate method
    - Syntactical (rule based – e.g. RuleML)
    - Bayesian classifier
    - Non-parametric
    - Linear discriminant
    - Neural networks
    - Sequential
    - Stochastic



Pattern recognition and classification

- Allocation of sensor value by defined function
  - Correlation of various data sources
  - Several methods possible – simple approaches
  - Template matching
  - Minimum distance methods
  - 'Integrated' feature extraction
    - Nearest Neighbour
    - Neural Networks
- Problem
  - Measured raw data might not allow to derive all features required
  - Therefore often combination of sensors


Not enough features


Sufficient feature count

Pattern recognition and classification

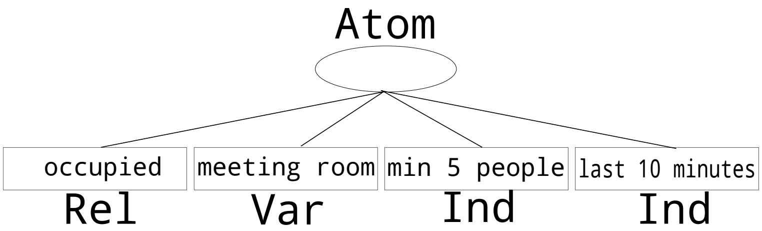
- Methods – Syntactical (Rule based)
  - Idea: Description of Situation by formal Symbols and Rules
  - Description of a (agreed on?) world view
  - Example: RuleML
- Comment
  - Pro:
    - Combination of rules and identification of loops and impossible conditions feasible

    Contra:
    - Very complex with more elaborate situations
    - Extension or merge of rule sets typically not possible without contradictions

Pattern recognition and classification

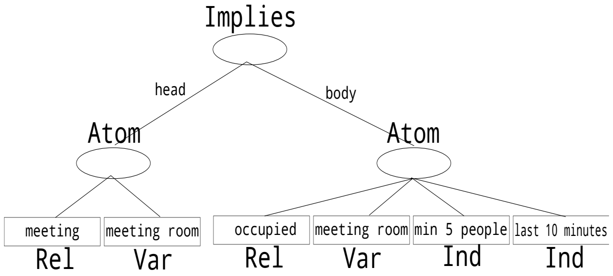
- Rule Markup Language: Language for publishing and sharing rules
- Hierarchy of rule-sub-languages (XML, RDF, XSLT, OWL)
- Example:
  - A meeting room was occupied by min 5 people for the last 10 minutes.

```
<Atom>
     <Rel> occupied </Rel>
     <Var> meeting room </Var>
     <Ind> min 5 persons </Ind>
     <Ind> last 10 minutes </Ind>
</Atom>
```



Pattern recognition and classification

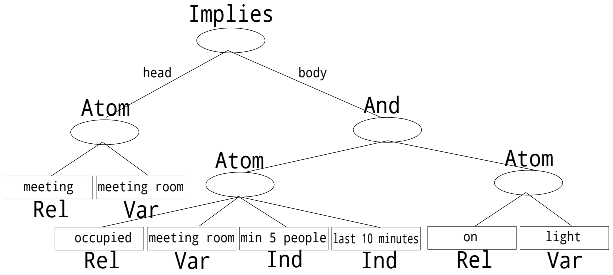
- Also conditions can be modelled
  - A Meeting is taking place in a meeting room when it was occupied by min 5 people for the last 10 minutes.



```
<Implies>
  <head>
     <Atom>
            <Rel> meeting </Rel>
            <Var> meeting room </Var>
     </Atom>
  </head>
  <body>
     <Atom>
            <Rel> occupied </Rel>
            <Var> meeting room </Var>
            <Ind> min 5 persons </Ind>
            <Ind> last 10 minutes </Ind>
     </Atom>
  </body>
</Implies>
```

Pattern recognition and classification

- Logical combination of conditions
  - A Meeting is taking place in a meeting room when it was occupied by min 5 people for the last 10 minutes and the light is on.



```
<Implies>
   <head>
      <Atom>
         <Rel> meeting </Rel>
         <Var> meeting room </Var>
      </Atom>
   </head>
   <body>
      <And>
         <Atom>
            <Rel> on </Rel>
            <Var> light </Var>
         </Atom>
         <Atom>
            <Rel> occupied </Rel>
            <Var> meeting room </Var>
            <Ind> min 5 persons </Ind>
            <Ind> last 10 minutes </Ind>
         </Atom>
      </And>
   </body>
</Implies>
```

Outline

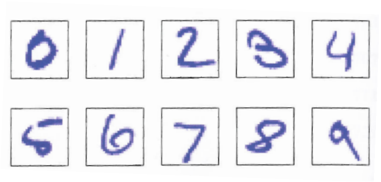
Recognition of patterns

Patterns can be described by a sufficient number of rules

Samples are inaccurate

Tremendous amount of rules to model all variations of one class

Therefore: Consider machine learning approaches

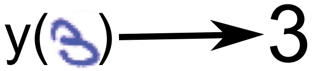


Recognition of patterns

Training set $x_1 \ldots x_N$ of a large number of N samples is utilised

Classes $t_1 \ldots t_N$ of all samples in this set known in advance

Machine learning algorithm computes a function $y(x)$ and generates a new target t‘

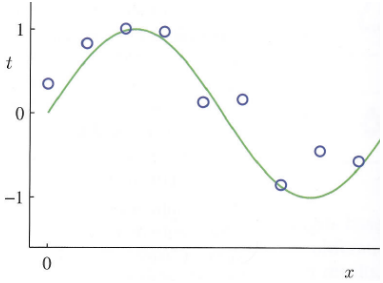


Polynomial curve fitting

Example

A curve shall be approximated by a machine learning approach

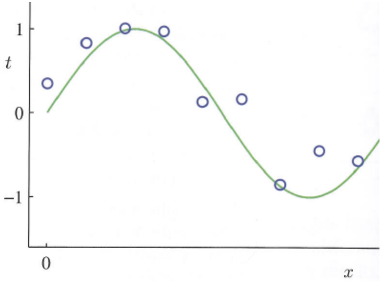
Sample points are created for the function $\sin(2\pi x) + \mathcal{N}$ where $\mathcal{N}$ is a random noise value



Polynomial curve fitting

We will try to fit the data points into a polynomial function:

$$y(x, \vec{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$



Polynomial curve fitting

We will try to fit the data points into a polynomial function:

$$y(x, \overrightarrow{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

This can be obtained by minimising an error function that measures the misfit between $y(x, \overrightarrow{w})$ and the training data set:

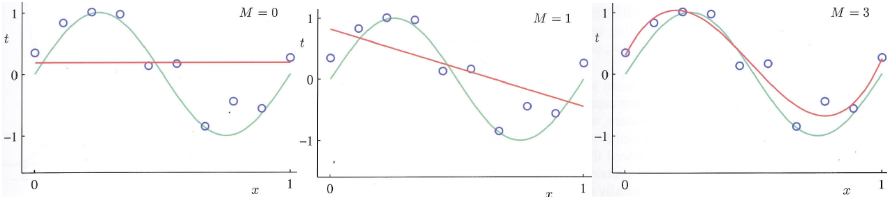
$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{i=1}^{N} \left[y(x_i, \overrightarrow{w}) - t_i \right]^2$$

$E(\overrightarrow{w})$ is non-negative and zero if and only if all points are covered by the function

Polynomial curve fitting

One problem is the right choice of the dimension M

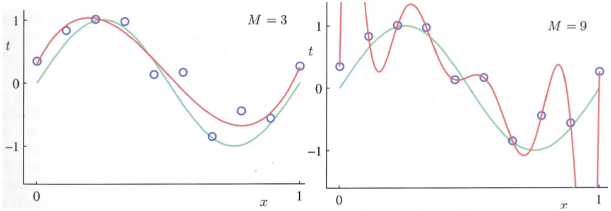
When M is too small, the approximation accuracy might be bad



Polynomial curve fitting

However, when M becomes too big, the resulting polynomial will cross all points exactly

When M reaches the count of samples in the training data set, it is always possible to create a polynomial of order M that contains all values in the data set exactly.



Polynomial curve fitting

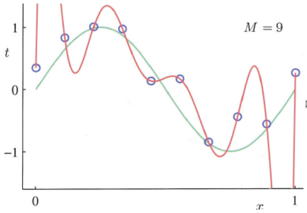
This event is called overfitting

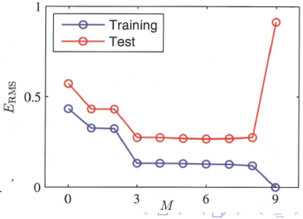
The polynomial now trained too well to the training data

It will therefore perform badly on another sample of test data for the same phenomenon

We visualise it by the Root of the Mean Square (RMS) of $E(\vec{w})$

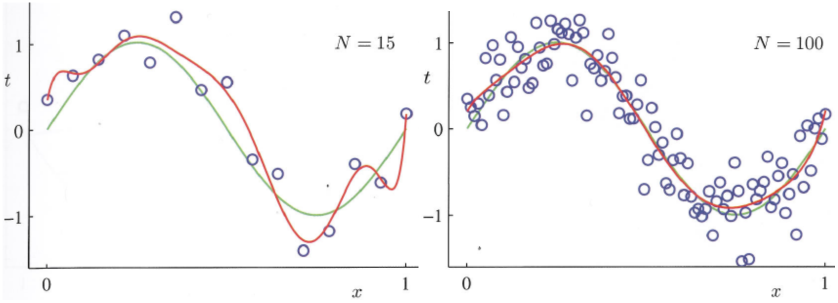
$$E_{RMS} = \sqrt{\frac{2E(\vec{w})}{N}}$$





Polynomial curve fitting

With increasing number of data points, the problem of overfitting becomes less severe for a given value of M



Polynomial curve fitting

One solution to cope with overfitting is regularisation

A penalty term is added to the error function

This term discourages the coefficients of $\overrightarrow{w}$ from reaching large values

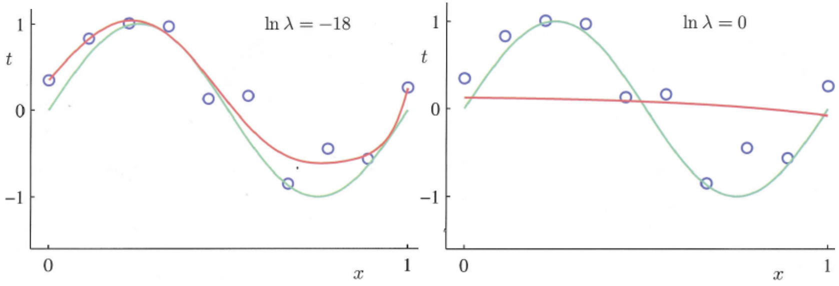
$$\overline{E}(\overrightarrow{w}) = \frac{1}{2}\sum_{i=1}^{N}\left[y(x_i, \overrightarrow{w}) - t_i\right]^2 + \frac{\lambda}{2}||\overrightarrow{w}||^2$$

with

$$||\overrightarrow{w}||^2 = \overrightarrow{w}^T\overrightarrow{w} = w_0^2 + w_1^2 + \cdots + w_M^2$$

Polynomial curve fitting

Depending on the value of λ, overfitting is controlled



$$\overline{E}(\overrightarrow{w}) = \frac{1}{2}\sum_{i=1}^{N}\left[y(x_i, \overrightarrow{w}) - t_i\right]^2 + \frac{\lambda}{2}||\overrightarrow{w}||^2$$

Outline

Bayesian decision theory

With probability theory, the probability of events can be estimated by repeatedly generating events and counting their occurrences

When, however, an event only very seldom occurs or is hard to generate, other methods are required

Example:

Probability that the Arctic ice cap will have disappeared by the end of this century

In such cases, we would like to model uncertainty

In fact, it is possible to represent uncertainty by probability

Conditional probability

Conditional probability

The conditional probability of two events χ_1 and χ_2 with $P(\chi_2) > 0$ is denoted by $P(\chi_1|\chi_2)$ and is calculated by

$$P(\chi_1|\chi_2) = \frac{P(\chi_1 \cap \chi_2)}{P(\chi_2)}$$

$P(\chi_1|\chi_2)$ describes the probability that event χ_2 occurs in the presence of event χ_2.

Bayesian decision theory

With the notion of conditional probability we can express the effect of observed data $\overrightarrow{t} = t_1, \ldots, t_N$ on a probability distribution of $\overrightarrow{w}$: $P(\overrightarrow{w})$.

Thomas Bayes described a way to evaluate the uncertainty of $\overrightarrow{w}$ after observing $\overrightarrow{t}$

$$P(\overrightarrow{w}|\overrightarrow{t}) = \frac{P(\overrightarrow{t}|\overrightarrow{w})P(\overrightarrow{w})}{P(\overrightarrow{t})}$$

$P(\overrightarrow{t}|\overrightarrow{w})$ expresses how probable a value for $\overrightarrow{t}$ is given a fixed choice of $\overrightarrow{w}$

Bayesian decision theory

A principle difference between Bayesian viewpoint and frequentist viewpoint is that prior assumptions are provided

Example:

Consider a fair coin that scores heads in three consecutive tosses

Classical maximum likelihood estimate will predict head for future tosses with probability 1

Bayesian approach includes prior assumptions on the probability of events and would result in a less extreme conclusion

Bayesian curve fitting

In the curve fitting problem, we are given $\overrightarrow{x}$ and $\overrightarrow{t}$ together with a new sample x_{M+1}

The task is to find a good estimation of the value t_{M+1}

This means that we want to evaluate the predictive distribution

$$p(t_{M+1}|x_{M+1}, \overrightarrow{x}, \overrightarrow{t})$$

To account for measurement inaccuracies, typically a probability distribution (e.g. Gauss) is underlying the sample vector $\overrightarrow{x}$

Bayesian curve fitting

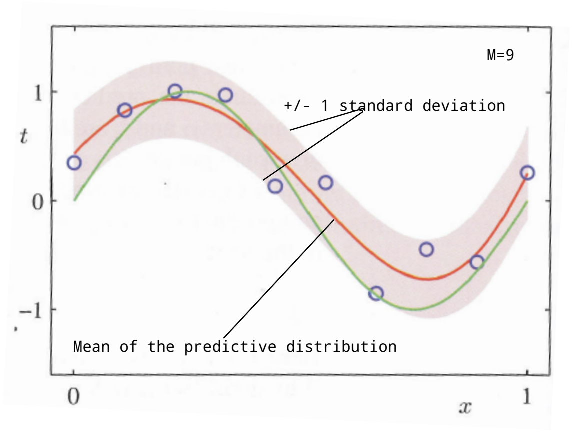
This means that we want to evaluate the predictive distribution

$$p(t_{M+1}|x_{M+1}, \overrightarrow{x}, \overrightarrow{t})$$

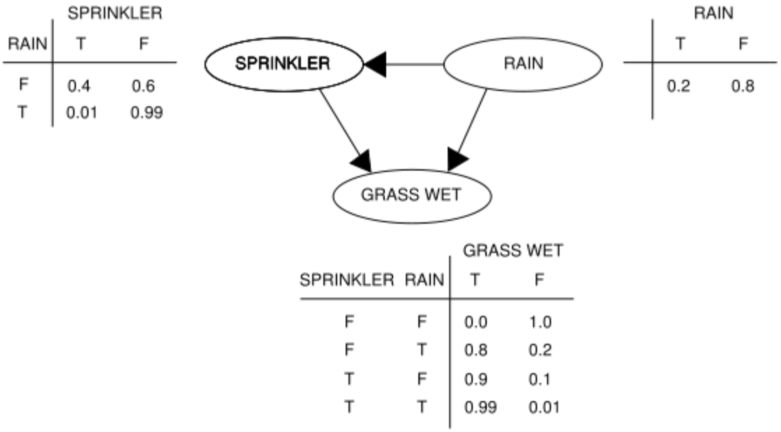
After consistent application of the sum and product rules of probability we can rewrite this as

$$p(t_{M+1}|x_{M+1}, \overrightarrow{x}, \overrightarrow{t}) = \int p(t_{M+1}|x_{M+1}, \overrightarrow{w})p(\overrightarrow{w}|\overrightarrow{x}, \overrightarrow{t})d\overrightarrow{w}$$

Bayesian curve fitting



Example



	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99

RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Outline

Histogram methods

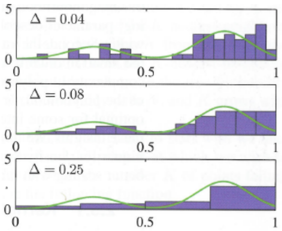
Alternative approach to function estimation: histogram methods

In general, the probability density of an event is estimated by dividing the range of N values into bins of size Δ_i

Then, count the number of observations that fall inside bin Δ_i

This is expressed as a normalised probability density

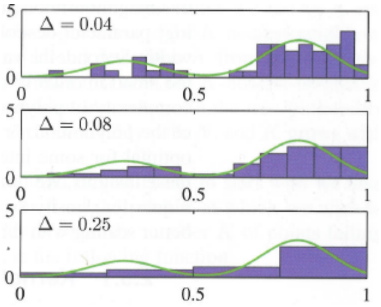
$$p_i = \frac{n_i}{N\Delta_i}$$



Histogram methods

Accuracy of the estimation is dependent on the width of the bins

Approach well suited for big data since the data items can be discarded once the histogram is created

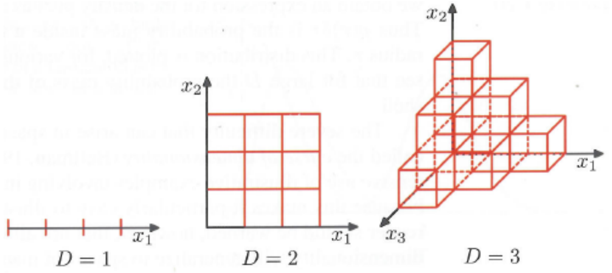


Histogram methods

Issues:

Due to the edges of the bins, the modelled distribution is characterised by discontinuities not present in the underlying distribution observed

The method does not scale well with increasing dimension (Curse of dimensionality)



Parzen estimator methods

Assume an unknown probability density $p(\cdot)$

We want to estimate the probability density $p(\overrightarrow{x})$ of $\overrightarrow{x}$ in a $\mathcal{D}$-dimensional Euclidean space

We consider a small region $\mathcal{R}$ around $\overrightarrow{x}$:

$$P = \int_{\mathcal{R}} p(\overrightarrow{x}) d\overrightarrow{x}$$

Parzen estimator methods

We utilise a data set of N observations

Each observation has a probability of P to fall inside $\mathcal{R}$

With the binomial distribution we can calculate the count K of points falling into $\mathcal{R}$:

$$\mathrm{Bin}(K|N,P) = \frac{N!}{K!(N-K)!} P^K (1-P)^{N-K}$$

Parzen estimator methods

We utilise a data set of N observations

Each observation has a probability of P to fall inside $\mathcal{R}$

With the binomial distribution we can calculate the count K of points falling into $\mathcal{R}$:

$$\mathrm{Bin}(K|N,P) = \frac{N!}{K!(N-K)!} P^K (1-P)^{N-K}$$

For large N we can show

$$K \approx NP$$

With sufficiently small $\mathcal{R}$ we can also show for the volume V of $\mathcal{R}$

$$P \approx p(\overrightarrow{x})V$$

Therefore, we can estimate the density as

$$p(\overrightarrow{x}) = \frac{K}{NV}$$

Parzen estimator methods

We assume that $\mathcal{R}$ is a small hypercube

In order to count the number K of points that fall inside $\mathcal{R}$ we define

$$k(\overrightarrow{u}) = \begin{cases} 1, & |u_i| \leq \frac{1}{2}, \quad i = 1, \ldots, D, \\ 0, & \text{otherwise} \end{cases}$$

This represents a unit cube centred around the origin

This function is an example of a kernel-function or Parzen window

Parzen estimator methods

$$k(\overrightarrow{u}) = \begin{cases} 1, & |u_i| \leq \frac{1}{2}, \quad i = 1, \ldots, D, \\ 0, & \text{otherwise} \end{cases}$$

When the measured data point $\overrightarrow{x_n}$ lies inside a cube of side h centred around $\overrightarrow{x}$, we have

$$k\left(\frac{\overrightarrow{x} - \overrightarrow{x_n}}{h}\right) = 1$$

The total count K of points that fall inside this cube is

$$K = \sum_{n=1}^{N} k\left(\frac{\overrightarrow{x} - \overrightarrow{x_n}}{h}\right)$$

Parzen estimator methods

The total count K of points that fall inside this cube is

$$K = \sum_{n=1}^{N} k\left(\frac{\overrightarrow{x} - \overrightarrow{x_n}}{h}\right)$$

When we substitute this in the density estimate derived above

$$p(\overrightarrow{x}) = \frac{K}{NV}$$

with volume $V = h^D$ we obtain the overall density estimate as

$$p(\overrightarrow{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}\left(\frac{\overrightarrow{x} - \overrightarrow{x_n}}{h}\right)$$

Parzen estimator methods

$$p(\overrightarrow{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}\left(\frac{\overrightarrow{x}-\overrightarrow{x_n}}{h}\right)$$

Again, this density estimator suffers from artificial discontinuities

(Due to the fixed boundaries of the cubes)

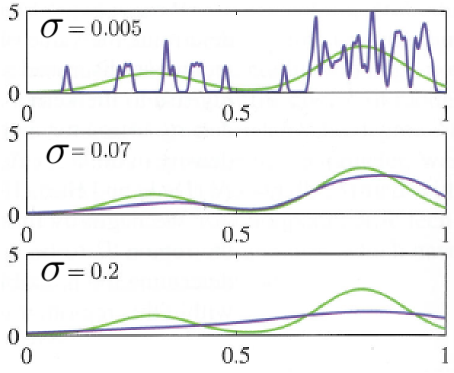
Problem can be overcome by choosing a smoother kernel function

(A common choice is a Gaussian kernel with a standard deviation σ)

$$p(\overrightarrow{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi\sigma^2)^{\frac{D}{2}}}e^{-\frac{||\overrightarrow{x}-\overrightarrow{x_n}||^2}{2\sigma^2}}$$

Parzen estimator methods

Density estimation for various values of σ

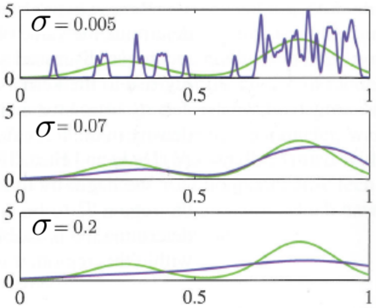


Nearest neighbour methods

A problem with Parzen estimator methods is that the parameter governing the kernel width (h or σ) is fixed for all values $\vec{x}$

In regions with

...high density, a wide kernel might lead to over-smoothing
...low density, the same width may lead to noisy estimates



Nearest neighbour methods

NN-methods address this by adapting width to data density

Parzen estimator methods fix V and determine K from the data
Nearest neighbour methods fix K and choose V accordingly

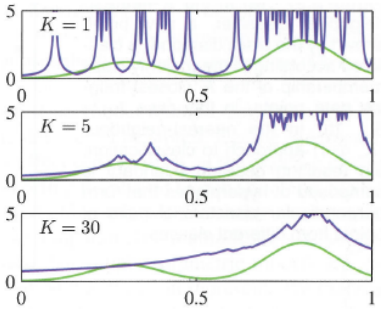
Again, we consider a point $\vec{x}$ and estimate the density $p(\vec{x})$

The radius of the sphere is increased until K data points (the nearest neighbours) are covered

Nearest neighbour methods

The value K then controls the amount of smoothing

Again, an optimum value for K exists



Nearest neighbour methods

Classification: Apply KNN-density estimation for each class

Assume data set of N points with N_k points in class C_k

To classify sample $\overrightarrow{x}$, draw a sphere containing K points around $\overrightarrow{x}$

Sphere can contain other points regardless of their class

Assume sphere has volume V and contains K_k points from C_k

Nearest neighbour methods

Assume: Sphere of volume V contains K_k points from class C_k

We estimate the density of class C_k as

$$p(\overrightarrow{x}|C_k) = \frac{K_k}{N_k V}$$

The unconditional density is given as

$$p(\overrightarrow{x}) = \frac{K}{NV}$$

The probability to experience a class C_k is given as

$$p(C_k) = \frac{N_k}{N}$$

With Bayes theorem we can combine this to achieve

$$p(C_k|\overrightarrow{x}) = \frac{p(\overrightarrow{x}|C_k)p(C_k)}{p(\overrightarrow{x})} = \frac{K_k}{K}$$

Nearest neighbour methods

$$p(C_k|\overrightarrow{x}) = \frac{p(\overrightarrow{x}|C_k)p(C_k)}{p(\overrightarrow{x})} = \frac{K_k}{K}$$

To minimise the probability of misclassification, assign $\overrightarrow{x}$ to class with the largest probability

This corresponds to the largest value of

$$\frac{K_k}{K}$$

Nearest neighbour methods

To classify a point, we identify the K nearest points

And assign the point to the class having most representatives in this set

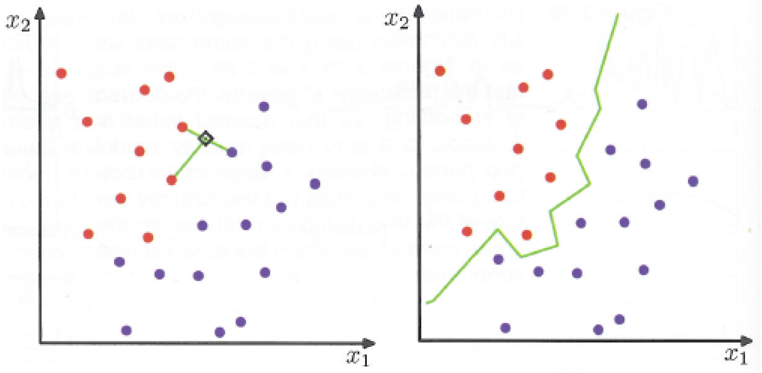
Choice $K = 1$ is called nearest neighbour rule

For this choice, the error rate is never more than twice the minimum achievable error rate of an optimum classifier[2]

[2] T. Cover and P. Hart: Nearest neighbour pattern classification. IEEE Transactions on Information Theory, IT-11, 21-27, 1967

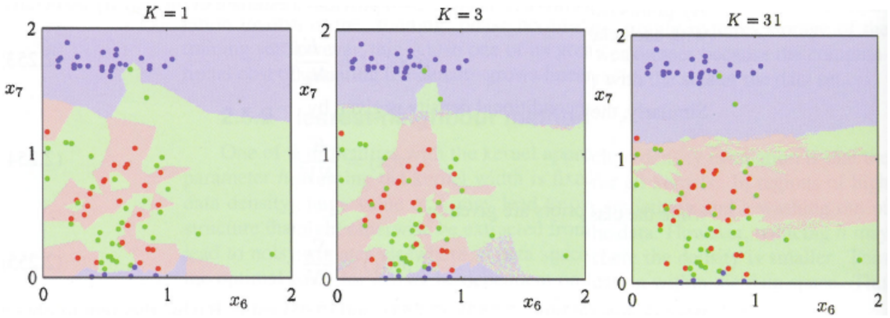
Nearest neighbour methods

Classification of points by the K-nearest neighbour classifier



Nearest neighbour methods

Classification of points by the K-nearest neighbour classifier



Nearest neighbour methods

The KNN-method and the Parzen-method are not well suited for large data sets since they require the entire data set to be stored

Outline

Support vector machines (SVM)

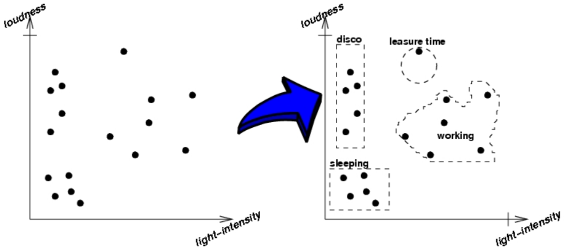
In classification we assign $\vec{x}$ to one of K discrete classes C_k

The input is divided by decision boundaries

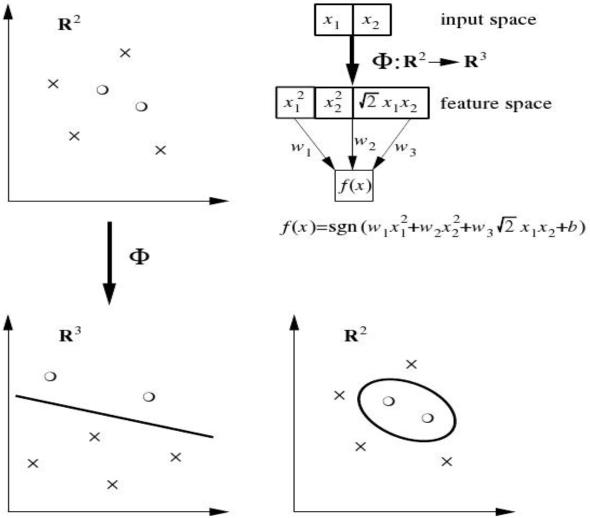
Here we assume that decision boundaries are linear functions of $\vec{x}$

Data separable by linear decision surfaces are linear separable

With high dimension, a set of two classes is always linear separable



Support vector machines (SVM)

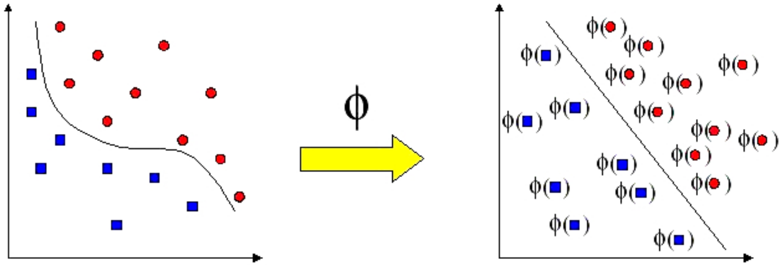

$\mathbf{R}^2$
input space
x_1 x_2
$\Phi: \mathbf{R}^2 \rightarrow \mathbf{R}^3$
x_1^2 x_2^2 $\sqrt{2}\, x_1 x_2$
feature space
w_1 w_2 w_3
$f(x)$
$f(x) = \mathrm{sgn}\,(w_1 x_1^2 + w_2 x_2^2 + w_3 \sqrt{2}\, x_1 x_2 + b)$
Φ
$\mathbf{R}^3$
$\mathbf{R}^2$

Support vector machines (SVM)

SVM pre-processes data to represent patterns in a high dimension

Dimension often much higher than original feature space

Then, insert hyperplane in order to separate the data



Support vector machines (SVM)

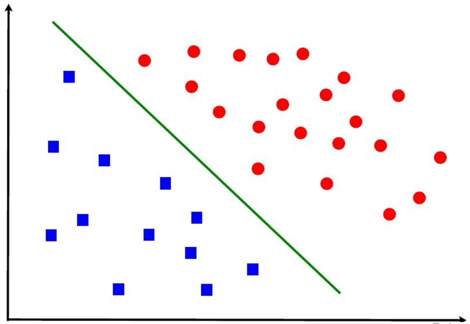
A pattern $\overrightarrow{x_k}$ is transformed to $\overrightarrow{y_k} = \varphi(\overrightarrow{x_k})$

Also, each $\overrightarrow{x_k}$ is associated with $z_k \in \{-1, 1\}$

A linear discriminant in an augmented $\overrightarrow{y}$ space is $g(\overrightarrow{y}) = \overrightarrow{a}^t \overrightarrow{y}$

A separating hyperplane ensures for $y_0 = 1, a_0 \geq 1$

$$z_k g(y_k) \geq 1$$



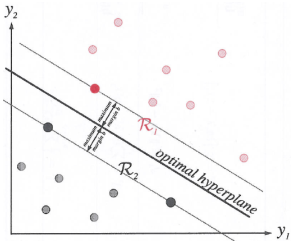
Support vector machines (SVM)

The goal for support vector machines is to find a separating hyperplane with the largest margin b to the outer points in all sets

$$\frac{z_k g(y_k)}{||\overrightarrow{a}||} \geq b, \;\; k = 1, \ldots, n$$

If no such hyperplane exists, map all points into a higher dimensional space until such a plane exists

Support vectors satisfy '$\cdot = b$'



Support vector machines (SVM)

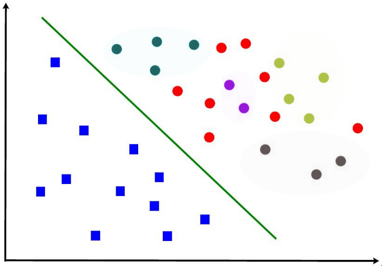
Simple application to several classes by iterative approach:

belongs to class 1 or not?

belongs to class 2 or not?

...

Search for optimum mapping between input space and feature space complicated (no optimum approach known)



Support vector machines (SVM)

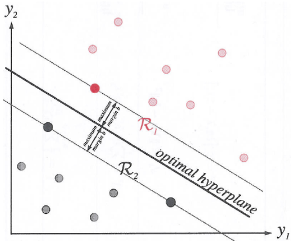
Simple learning approach to find the correct hyperplane:

Starting from an initial separating hyperplane

Find worst classified pattern (on the wrong side of the hyperplane)

Design a new hyperplane with this pattern as one of the support vectors

Iterate until all patterns are correctly classified



Outline

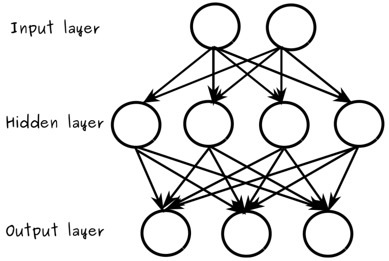
Neural networks

Learn mapping from input to output vector

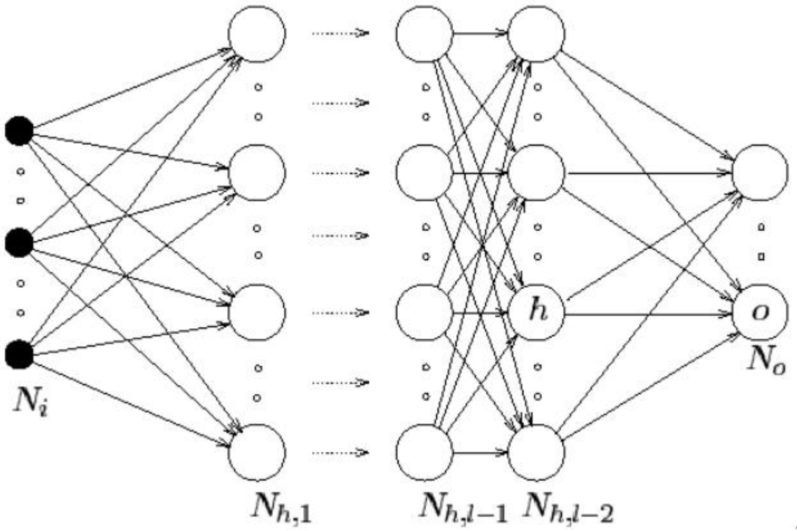
Representation by edge-weighted graph

Distinction between

- Input neurons
- Output neurons
- Hidden nodes



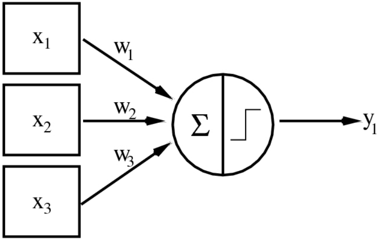
Neural networks



Neural networks

Input neurons are only equipped with outgoing edges

Hidden nodes ‘fire‘ (output value 1) when weighted inputs exceed threshold function Θ

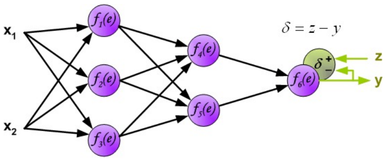


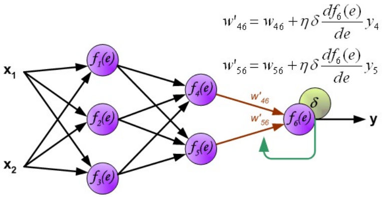
$$y_i = \begin{cases} 1, & if \sum_{i=1}^{n} x_i w_i \geq \Theta \\ 0, & else. \end{cases}$$

Neural networks

Learning with back-propagation (schematic):
(Iterate until the error is sufficiently small)

1. Choose a training-pair and copy it to the input layer
2. Propagate it through the network
3. Calculate error between computed and expected output
4. Propagate the sum product of the weights back into the network in order to calculate the error in internal layers
5. Adapt weights to the error




$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$
$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
w'_{46}
w'_{56}
δ
y

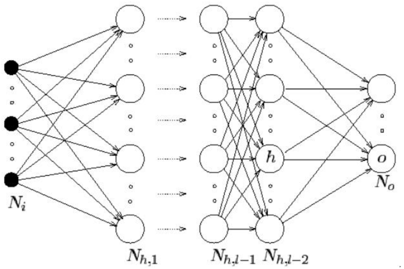
Neural networks

Single hidden layer sufficient to represent arbitrary multi-dimensional functions

Well suited for noisy input data

Implicit clustering of input data possible

Complex to extend network (e.g. add new features)



Neural networks

For the input layer, we construct M linear combinations of the input variables $x_1, \ldots, x_D$ and weights $w_1, \ldots, w_D$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

Each a_j is transformed using a differentiable, non-linear activation function

$$z_j = h(a_j)$$

Neural networks

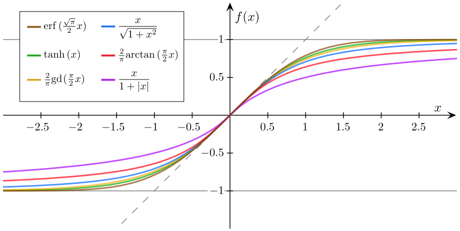
Input layer M linear combinations of $x_1, \ldots, x_D$ and $w_1, \ldots, w_D$

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

Activation function: Differentiable, non-linear

$$z_j = h(a_j)$$

$h(\cdot)$ function is usually a sigmoidal function or tanh



Neural networks

Values z_j are again linearly combined in hidden layers:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

with $k = 1, \ldots, K$ describing the total number of outputs

Again, these values are transformed using a sufficient transformation function σ to obtain the network outputs y_k

$$y_k = \sigma(a_k)$$

For multi-class problems, we use a function such as

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Neural networks

Combine these stages to achieve overall network function:

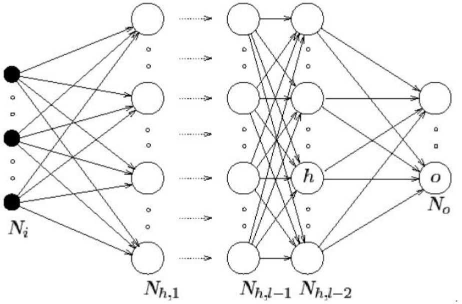
$$y_k(\overrightarrow{x}, \overrightarrow{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

(Multiple hidden layers are added analogously)

Neural networks

Activation functions of hidden units are linear ⇒ Always find equivalent network without hidden units

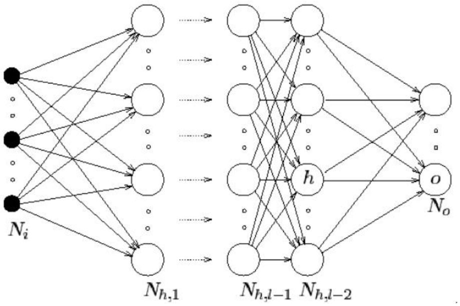
(Composition of successive linear transformations itself linear transformation)



Neural networks

Number of hidden units $<$ number of input or output units $\Rightarrow$ not all linear functions possible

(Information lost in dimensionality reduction at hidden units)



Neural networks

Neural networks are Universal approximators[3] [4] [5] [6] [7] [8] [9] [10]
$\Rightarrow$ 2-layer linear NN can approximate any continuous function

[3] K. Funahashi: On the approximate realisation of continuous mappings by neural networks, Neural Networks, 2(3), 183-192, 1989

[4] G. Cybenko: Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2, 304-314, 1989

[5] K. Hornik, M. Sinchcombe, H. White: Multilayer feed-forward networks are universal approximators. Neural Networks, 2(5), 359-366, 1989

[6] N.E. Cotter: The stone-Weierstrass theorem and its application to neural networks. IEEE Transactions on Neural Networks 1(4), 290-295, 1990

[7] Y. Ito: Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. Neural Networks 4(3), 385-394, 1991

[8] K. Hornik: Approximation capabilities of multilayer feed forward networks: Neural Networks, 4(2), 251-257, 1991

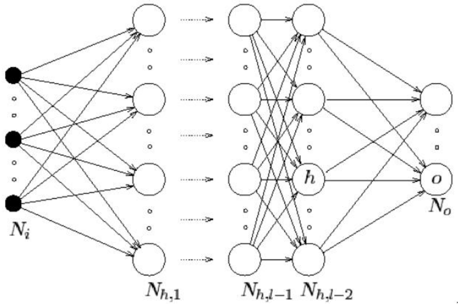
[9] Y.V. Kreinovich: Arbitrary non-linearity is sufficient to represent all functions by neural networks: a theorem. Neural Networks 4(3), 381-383, 1991

[10] B.D. Ripley: Pattern Recognition and Neural Networks. Cambridge University Press, 1996

Neural networks

Remaining issue in neural networks

- Find suitable parameters given a set of training data
- Several learning approaches have been proposed



Neural networks

Simple approach to determine network parameters: Minimise sum-of-squared error function

- Given a training set $\overrightarrow{x_n}$ with $n \in \{1, \ldots, N\}$
- And corresponding target vectors $\overrightarrow{t_n}$
- Minimise the error function

$$E(\overrightarrow{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(\overrightarrow{x_n}, \overrightarrow{w}) - \overrightarrow{t_n})^2$$

Neural networks – Classification

2 classes $\mathcal{C}_1$ and $\mathcal{C}_2$

- We consider a network with a single output

$$y = \sigma(a) \equiv \frac{1}{1 + e^{-a}}$$

- Output interpreted as conditional probability $p(\mathcal{C}_1|\overrightarrow{x})$
- Analogously, we have $p(\mathcal{C}_2|\overrightarrow{x}) = 1 - p(\mathcal{C}_1|\overrightarrow{x})$

K classes $\mathcal{C}_1, \cdots, \mathcal{C}_K$

- Binary target variables $t_k \in \{0, 1\}$
- Network outputs are interpreted as $y_k(\overrightarrow{x}, \overrightarrow{w}) = p(t_k = 1|\overrightarrow{x})$

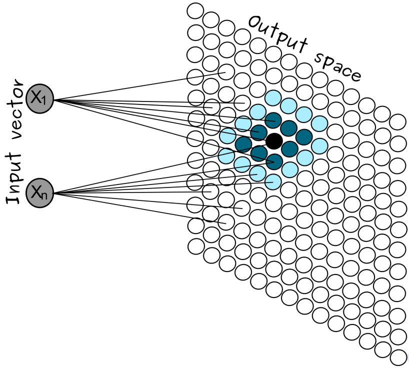
Introduction to self organising maps (SOM)

Proposed by Teuvo Kohonen[11]

As a model of the self-organisation of neural connections

Maps high dimensional input to low dimensional output

Based on neural network learning of the underlying mapping

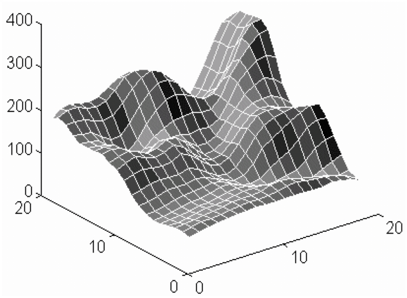


Introduction to self organising maps

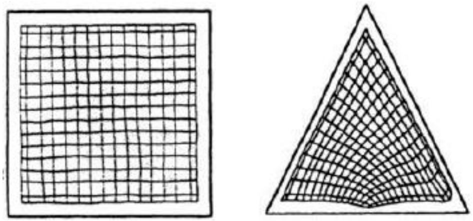
Present all points in a source space by points in a target space

Given a sequence of points in a sample space,

Create a mapping of these points into a target space that respects the neighbourhood relation in the sample space



Introduction to self organising maps



SOM is a topology preserving lattice of predefined number of nodes

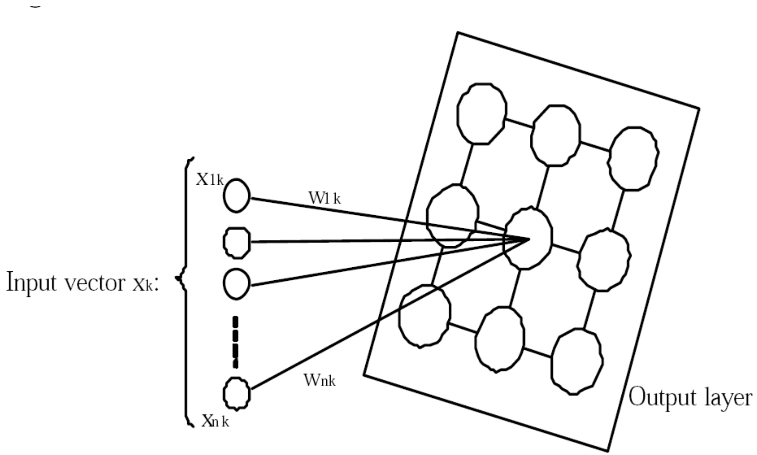
Represents topology of elements in input space.

Algorithm inherits self-organisation property

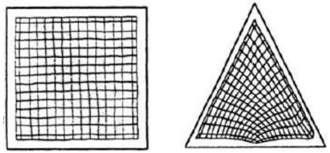
- Able to produce organisation starting from total disorder.
- Defines and preserves neighbourhood structure between nodes

Learning by two layer neural network

Introduction to self organising maps



Introduction to self organising maps



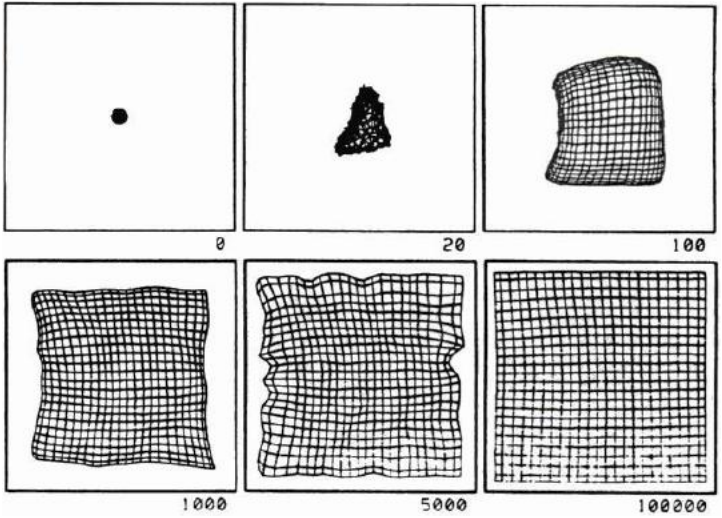
When a pattern $\overrightarrow{\phi_i}$ is presented, each node (represented by outer neurons) in the target space computes its activation $\overrightarrow{\phi}_i^t \overrightarrow{w}$.

Most activated node y^* and weights to its neighbours are updated according to a learning rate $\rho(t)$

$$w_{ki}(t+1) = w_{ki}(t) + \rho(t)\Lambda(|y - y^*|)(\overrightarrow{\phi_i} - w_{ki}(t))$$

$\Lambda(\cdot)$ defines a non-increasing neighbourhood function and $|y - y^*|$ describes the distance of nodes in the neighbourhood

SOM – Self organisation



SOM – Definition

Condensed definition of SOM from Cottrell et al.[12]

Self organising maps

- Let $I = \{\overrightarrow{\eta_1}, \dots, \overrightarrow{\eta_{|S|}}\}$ be a set of *km*-dimensional vectors that are associated with nodes in a lattice.
- Neighbourhood structure provided by symmetrical neighbourhood function $d : I \times I \to \mathbb{R}$ which depends on the distance between two nodes $\overrightarrow{\eta_i}$ and $\overrightarrow{\eta_j} \in I$.
- State of the map at time t given by

$$\eta(t) = \left(\overrightarrow{\eta_1(t)}, \overrightarrow{\eta_2(t)}, \dots, \overrightarrow{\eta_{|S|}(t)}\right),$$

[12] M. Cottrell, J.C. Fort and G. Pages, *Theoretical aspects of the SOM algorithm*, Neurocomputing, pp. 119-138, vol 21, 1998.

SOM – Definition

Self organising map algorithm

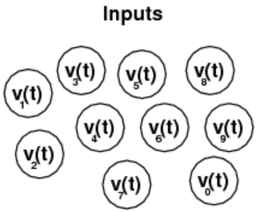
The SOM algorithm is recursively defined by

$$\begin{aligned} i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right) &= argmin\left\{\left\|\overrightarrow{v(t+1)} - \overrightarrow{\eta_i(t)}\right\|, \overrightarrow{\eta_i(t)} \in \eta(t)\right\}, \\ \overrightarrow{\eta_i(t+1)} &= \overrightarrow{\eta_i(t)} - \varepsilon_t d\left[i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right), \overrightarrow{\eta_i}\right] \\ &\cdot \left(\overrightarrow{\eta_i(t)} - \overrightarrow{v(t+1)}\right), \forall \overrightarrow{\eta_i} \in I. \end{aligned}$$

In this formula, $i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right)$ corresponds to the node in the network that is closest to the input vector.

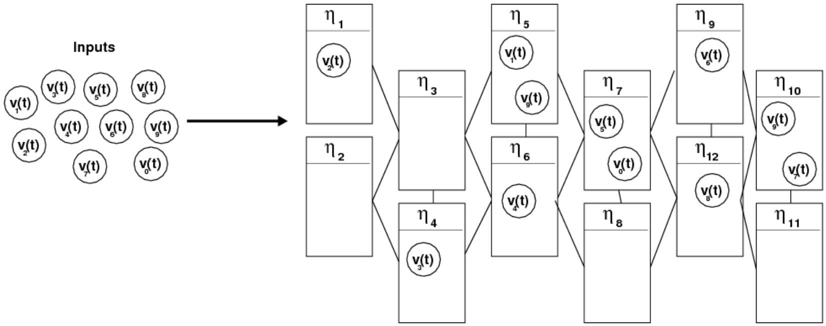
Parameter ε_t controls the adaptability.

SOM – Operational principle



Input values $v_i(t)$ are to be mapped onto the target space

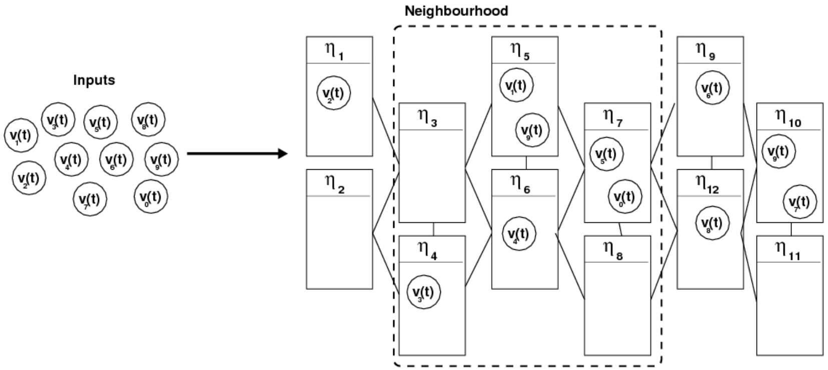
SOM – Operational principle



Node with the lowest distance is associated with the input value:

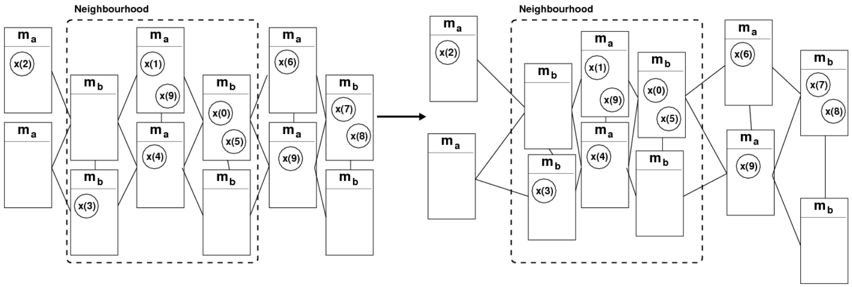
$$i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right) = argmin\left\{\left\|\overrightarrow{v(t+1)} - \overrightarrow{\eta_i(t)}\right\|, \overrightarrow{\eta_i(t)} \in \eta(t)\right\}$$

SOM – Operational principle


Inputs
Neighbourhood
$v_1(t)$ $v_2(t)$ $v_3(t)$ $v_4(t)$ $v_5(t)$ $v_6(t)$ $v_7(t)$ $v_8(t)$ $v_9(t)$ $v_0(t)$
η_1 η_2 η_3 η_4 η_5 η_6 η_7 η_8 η_9 η_{10} η_{11} η_{12}

Nodes in the neighbourhood of the associated node are moved closer to the input value

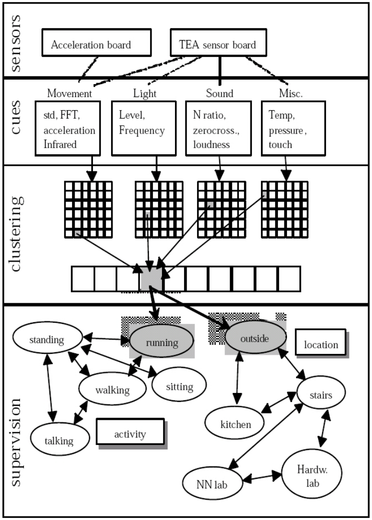
SOM – Operational principle



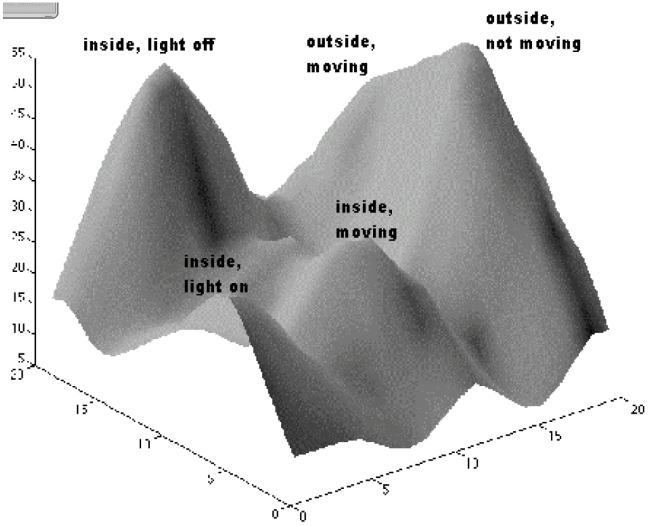
Nodes in the neighbourhood of the associated node are moved to the input value

$$\overrightarrow{\eta_i(t+1)} = \overrightarrow{\eta_i(t)} - \varepsilon_t d\left[i_c\left(\overrightarrow{v(t+1)}, \overrightarrow{\eta(t)}\right), \overrightarrow{\eta_i}\right] \cdot \left(\overrightarrow{\eta_i(t)} - \overrightarrow{v(t+1)}\right), \forall \overrightarrow{\eta_i} \in I.$$

SOM – Example application: TEA



SOM – Example application: TEA



SOM – Remarks

SOM algorithm always converges[13]

Normalisation of input vectors might improve numerical accuracy

Not guaranteed that self-optimisation will always occur
(Dependent on choice of parameters)

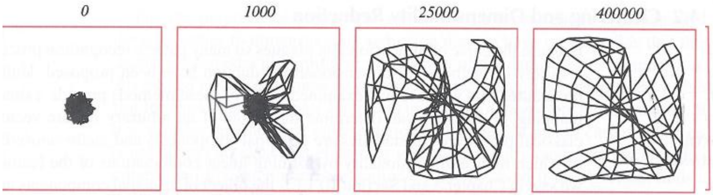
Difficult to set parameters of the model since SOM is not optimising any well-defined function[14]

If neighbourhood is chosen to be too small, the map will not be ordered globally

[13] Y. Cheng, *Neural Computation*, 9(8), 1997.

[14] E. Erwin, K. Obermayer, K. Schulten: Self-organising maps: Ordering, convergence properties and energy functions. Biological Cybernetics, 67, 47-55, 1992

Problems of SOMs



Map created as target space might have several orientations

One part of the map might follow one orientation, while other parts are following other orientations

Outline

Markov chains

Markov processes

- Intensively studied
- Major branch in the theory of stochastic processes

A. A. Markov (1856 – 1922)

Extended by A. Kolmogorov to chains of infinitely many states

- 'Anfangsgründe der Theorie der Markoffschen Ketten mit unendlich vielen möglichen Zuständen' (1936) [15]

[15] A. Kolmogorov, *Anfangsgründe der Theorie der Markoffschen Ketten mit unendlich vielen möglichen Zuständen*, 1936.

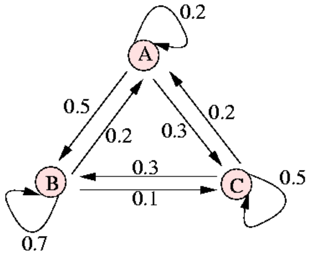
Markov chains

- Theory applied to a variety of algorithmic problems
- Standard tool in many probabilistic applications

Intuitive graphical representation

- Suitable for graphical illustration of stochastic processes

Popular for their simplicity and easy applicability to huge set of problems[16]



[16] William Feller, *An introduction to probability theory and its applications*, Wiley, 1968.

Markov chains

Independent trials of events

Dependent trials of events

Markov chains

Independent trials of events

- Set of possible outcomes of a measurement E_i associated with occurrence probability p_i
- Probability to observe sample sequence:
    - $P\{(E_1, E_2, \dots, E_i)\} = p_1 p_2 \cdots p_i$

Dependent trials of events

Markov chains

Independent trials of events

- Set of possible outcomes of a measurement E_i associated with occurrence probability p_i
- Probability to observe sample sequence:
  - $P\{(E_1, E_2, \ldots, E_i)\} = p_1 p_2 \cdots p_i$

Dependent trials of events

- Probability to observe specific sequence $E_1, E_2, \ldots, E_i$ obtained by conditional probability:

$$P(E_i | E_1, E_2, \ldots, E_{i-1})$$

- In general:

$$P(E_i | E_1, E_2, \ldots, E_{i-1}) \neq P(E_i | E_2, E_1, E_3, E_4, \ldots, E_{i-1})$$

Markov chains

Independent random variables

Dependent random variables

Markov chains

Independent random variables

- Number of coin tosses until 'head' is observed
- Radioactive atoms always have same probability of decaying at next trial

Dependent random variables

Markov chains

Independent random variables

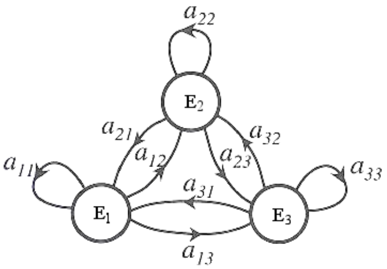
- Number of coin tosses until 'head' is observed
- Radioactive atoms always have same probability of decaying at next trial

Dependent random variables

- Knowledge that no car has passed for five minutes increases expectation that it will come soon.
- Coin tossing:
  - Probability that the cumulative numbers of heads and tails will equalize at the second trial is $\frac{1}{2}$
  - Given that they did not, the probability that they equalize after two additional trials is only $\frac{1}{4}$

Markov property

In the theory of stochastic processes the described lack of memory is connected with the Markov property.



Outcome depends exclusively on outcome of directly preceding trial

- Every sequence (E_i, E_j) has a conditional probability p_{ij}
- Additionally: Probability a_i of the event E_i

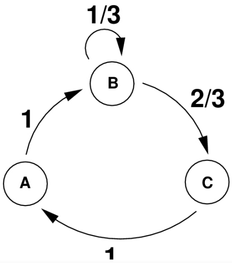
Markov chains

Markov chain

A sequence of observations $E_1, E_2, \ldots$ is called a Markov chain if the probabilities of sample sequences are defined by

$$P(E_1, E_2, \ldots, E_i) = a_1 \cdot p_{12} \cdot p_{23} \cdot \cdots \cdot p_{(i-1)i}.$$

and fixed conditional probabilities p_{ij} that the event E_i is observed directly in advance of E_j.



Markov chains

Described by probability a for initial distribution and matrix P of transition probabilities.

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots \\ p_{21} & p_{22} & p_{23} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

P is called a stochastic matrix

(Square matrix with non-negative entries that sum to 1 in each row)

Markov chains

p_{ij}^k denotes probability that E_j is observed exactly k observations after E_i was observed.

Calculated as the sum of the probabilities for all possible paths $E_i E_{i_1} \cdots E_{i_{k-1}} E_j$ of length k

We already know

$$p_{ij}^1 = p_{ij}$$

Consequently:

$$\begin{aligned} p_{ij}^2 &= \sum_\nu p_{i\nu} \cdot p_{\nu j} \\ p_{ij}^3 &= \sum_\nu p_{i\nu} \cdot p_{\nu j}^2 \end{aligned}$$

Markov chains

By mathematical induction:

$$p_{ij}^{n+1} = \sum_{\nu} p_{i\nu} \cdot p_{\nu j}^{n}$$

and

$$p_{ij}^{n+m} = \sum_{\nu} p_{i\nu}^{m} \cdot p_{\nu j}^{n} = \sum_{\nu} p_{i\nu}^{n} \cdot p_{\nu j}^{m}$$

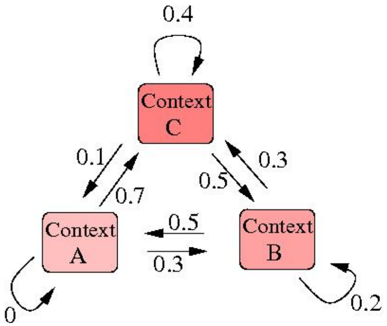
Similar to matrix P we can create a matrix P^n that contains all p_{ij}^n

p_{ij}^{n+1} obtained from P^{n+1}: Multiply row i of P with column j of P^n

Symbolically: $P^{n+m} = P^n P^m$.

$$P^n = \begin{bmatrix} p_{11}^n & p_{12}^n & p_{13}^n & \cdots \\ p_{21}^n & p_{22}^n & p_{23}^n & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Markov chains



	Context A	Context B	Context C
Context A	0	0.3	0.7
Context B	0.5	0.2	0.3
Context C	0.1	0.5	0.4

	Context A	Context B	Context C
Context A	0.22	0.41	0.37
Context B	0.13	0.34	0.53
Context C	0.29	0.33	0.38

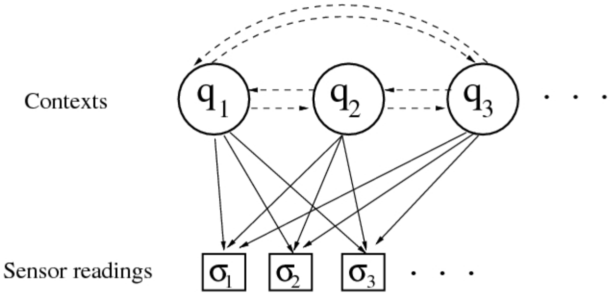
	Context A	Context B	Context C
Context A	0.242	0.333	0.425
Context B	0.223	0.372	0.405
Context C	0.203	0.343	0.454

Hidden Markov Models

Make a sequence of decisions for a process that is not directly observable[17]

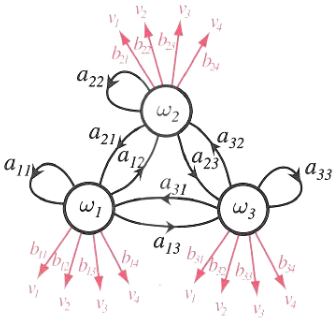
Current states of the process might be impacted by prior states

HMM often utilised in speech recognition or gesture recognition



[17] Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern classification*, Wiley-interscience, 2001.

Hidden Markov Models

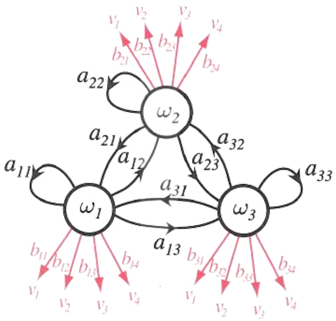


At every time step t the system is in an internal state $\omega(t)$

Additionally, we assume that it emits a (visible) symbol $v(t)$

Only access to visible symbols and not to internal states

Hidden Markov Models



Probability to be in state $\omega_j(t)$ and emit symbol $v_k(t)$:

$$P(v_k(t)|\omega_j(t)) = b_{jk}$$

Transition probabilities: $p_{ij} = P(\omega_j(t+1)|\omega_i(t))$

Emission probability: $b_{jk} = P(v_k(t)|\omega_j(t))$

Hidden Markov Models

Central issues in hidden Markov models:

Evaluation problem Determine the probability that a particular sequence of visible symbols V^T was generated by a given hidden Markov model

Decoding problem Determine the most likely sequence of hidden states ω^T that led to a specific sequence of observations V^T

Learning problem Given a set of training observations of visible symbols, determine the parameters p_{ij} and b_{jk} for a given HMM

Hidden Markov Models – Evaluation problem

Probability that model produces a sequence V^T:

$$P(V^T) = \sum_{\overline{\omega}^T} P(V^T|\overline{\omega}^T)P(\overline{\omega}^T)$$

Also:

$$\begin{aligned} P(\overline{\omega}^T) &= \prod_{t=1}^{T} P(\omega(t)|\omega(t-1)) \\ P(V^T|\overline{\omega}^T) &= \prod_{t=1}^{T} P(v(t)|\omega(t)) \end{aligned}$$

Together:

$$P(V^T) = \sum_{\overline{\omega}^T} \prod_{t=1}^{T} P(v(t)|\omega(t))P(\omega(t)|\omega(t-1))$$

Hidden Markov Models – Evaluation problem

Probability that model produces a sequence V^T:

$$P(V^T) = \sum_{\overline{\omega}^T} \prod_{t=1}^{T} P(v(t)|\omega(t)) P(\omega(t)|\omega(t-1))$$

Formally complex but straightforward

Naive computational complexity

- $\mathcal{O}(c^T T)$

Hidden Markov Models – Evaluation problem

Probability that model produces a sequence V^T:

$$P(V^T) = \sum_{\overline{\omega}^T} \prod_{t=1}^{T} P(v(t)|\omega(t))P(\omega(t)|\omega(t-1))$$

Computationally less complex algorithm:

- Calculate $P(V^T)$ recursively
- $P(v(t)|\omega(t))P(\omega(t)|\omega(t-1))$ involves only $v(t), \omega(t)$ and $\omega(t-1)$

$$\alpha_j(t) = \begin{cases} 0 & t = 0 \text{ and } j \neq \text{ initial state} \\ 1 & t = 0 \text{ and } j = \text{ initial state} \\ \left[\sum_i \alpha_i(t-1)p_{ij}\right] b_{jk} & \text{otherwise } {\scriptstyle (b_{jk} \text{ leads to observed } v(t))} \end{cases}$$

Hidden Markov Models – Evaluation problem

Forward Algorithm

Computational complexity: $O(c^2T)$

Forward algorithm

```
1 initialise $t \leftarrow 0, p_{ij}, b_{jk}, V^T, \alpha_j(0)$
2     for $t \leftarrow t+1$
3         $j \leftarrow 0$
4         for $j \leftarrow j+1$
5             $\alpha_j(t) \leftarrow b_{jk} \sum_{i=1}^{c} \alpha_i(t-1)p_{ij}$
6         until $j = c$
7     until $t = T$
8 return $P(V^T) \leftarrow \alpha_j(T)$ for the final state
9 end
```

Hidden Markov Models – Decoding problem

Given a sequence V^T, find most probable sequence of hidden states

Enumeration of every possible path will cost $O(c^T)$

- Not feasible

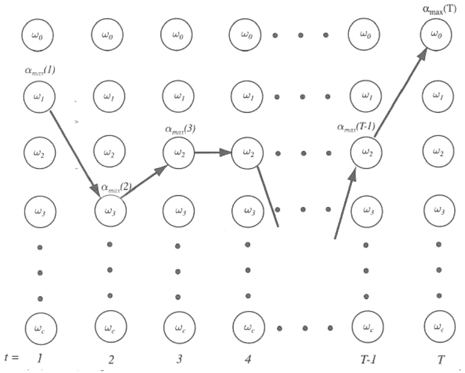
Hidden Markov Models – Decoding problem

Given a sequence V^T, find most probable sequence of hidden states

Decoding algorithm

```
1 initialise:  path $\leftarrow \{\}, t \leftarrow 0$
2     for $t \leftarrow t + 1$
3         $j \leftarrow 0$;
4         for $j \leftarrow j + 1$
5             $\alpha_j(t) \leftarrow b_{jk} \sum_{i=1}^{c} \alpha_i(t-1) p_{ij}$
6         until $j = c$
7         $j' \leftarrow arg \max_j \alpha_j(t)$
8         append $\omega_{j'}$ to path
9     until $t = T$
10 return path
11 end
```

Hidden Markov Models – Decoding problem



Computational time of the decoding algorithm

- $O(c^2 T)$

Hidden Markov Models – Learning problem

Determine the model parameters p_{ij} and b_{jk}

- Given: Training sample of observed values V^T

No method known to obtain the optimal or most likely set of parameters from the data

- However, we can nearly always determine a good solution by the forward-backward algorithm
- General expectation maximisation algorithm
- Iteratively update weights in order to better explain the observed training sequences

Hidden Markov Models – Learning problem

Probability that the model is in state $\omega_i(t)$ and will generate the remainder of the given target sequence:

$$\beta_i(t) = \begin{cases} 0 & t = T \text{ and } \omega_i(t) \text{ not final hidden state} \\ 1 & t = T \text{ and } \omega_i(t) \text{ final hidden state} \\ \sum_j \beta_j(t+1) p_{ij} b_{jk} & \text{otherwise } (b_{jk} \text{ leads to } v(t+1)) \end{cases}$$

Hidden Markov Models – Learning problem

$\alpha_i(t)$ and $\beta_i(t)$ only estimates of their true values since transition probabilities p_{ij}, b_{jk} unknown

Probability of transition between $\omega_i(t-1)$ and $\omega_j(t)$ can be estimated

- Provided that the model generated the entire training sequence V^T by **any** path

$$\gamma_{ij}(t) = \frac{\alpha(t-1)p_{ij}b_{jk}\beta_j(t)}{P(V^T|\Omega)}$$

Probability that model generated sequence V^T:

$$P(V^T|\Omega)$$

Hidden Markov Models – Learning problem

Calculate improved estimate for p_{ij} and b_{jk}

$$\overline{p_{ij}} = \frac{\sum_{t=1}^{T} \gamma_{ij}(t)}{\sum_{t=1}^{T} \sum_k \gamma_{ik}(t)}$$

$$\overline{b_{jk}} = \frac{\sum_{t=1, v(t)=v_k}^{T} \sum_l \gamma_{jl}(t)}{\sum_{t=1}^{T} \sum_l \gamma_{jl}(t)}$$

Start with rough estimates of p_{ij} and b_{jk}

Calculate improved estimates

Repeat until some convergence is reached

Hidden Markov Models – Learning problem

Forward-Backward algorithm

```
1 initialise $p_{ij}, b_{jk}, V^T$, convergence criterion $\Delta, t \leftarrow 0$
2     do $t \leftarrow t+1$
3         compute $\overline{p_{ij}(t)}$
4         compute $\overline{b_{jk}(t)}$
5         $p_{ij}(t) \leftarrow \overline{p_{ij}(t)}$
6         $b_{jk}(t) \leftarrow \overline{b_{jk}(t)}$
7     until $\max_{i,j,k}[p_{ij}(z) - p_{ij}(z-1), b_{jk}(t) - b_{jk}(t-1)] < \Delta$
                  (convergence achieved)
8 return $p_{ij} \leftarrow p_{ij}(t)$, $b_{jk} \leftarrow b_{jk}(t)$
9 end
```

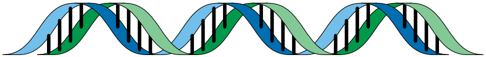
Outline

Stochastic methods

When problem structure is not well known, it might be hard to configure classification methods correctly

Solution: randomised search approaches

Search space spanned by possible configurations for all parameters



Solutions found are not necessarily optimal

Randomised search approaches

Local random search

Metropolis algorithm

Simulated annealing

Evolutionary algorithms

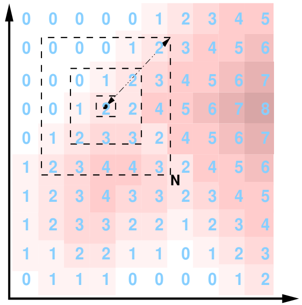
Local random search heuristics

- Local random search
  - Intuitive way to climb a mountain (by a sightless climber)

Local random search

$\forall x$ in search space S, define non-empty neighbourhood $N(x) \subseteq S$
Iteratively draw one sample $x' \in N(x)$.
<u>Fitness improved</u> $(F(x) > F(x')) \Rightarrow$ new best search point.
<u>Otherwise</u> $\Rightarrow$ discarded.

Local random search heuristics



$N(x) = x$ or $N(x) = S$ valid, but original idea is that $N(x)$ is small set of search points.

Points $x' \in N(x)$ expected nearer to x than points $x'' \notin N(x)$

Typically, $x \in N(x)$

Local random search heuristics

Complexity reduction by restriction of the search space size

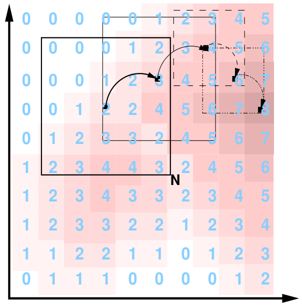
Example: $S = \{0,1\}^n$ and $N_d(x)$ are all points y with Hamming distance smaller than d ($H(x,y) \leq d$)

- For constant d we obtain: $|N_d(x)| = \Theta(n^d) \ll |S| = 2^n$

$d \leq 1$	$d \leq 2$	$d \leq 3$
1 0 1 0	1 0 1 0	1 0 1 0
0 0 1 0	0 0 1 0	1 1 1 0
1 1 1 0	1 1 1 0	1 1 1 0
1 0 0 0	1 0 0 0	1 0 0 0
1 0 1 1	1 0 1 1	1 0 1 1
	0 1 1 0	0 1 1 0
	0 0 0 0	0 0 0 0
	0 0 1 1	0 0 1 1
	1 1 0 0	1 1 0 0
	1 1 1 1	1 1 1 1
	1 0 0 1	1 0 0 1
		0 1 0 0
		0 1 1 1
		1 1 0 1
		1 0 0 1

$$|N_d(x)| = \binom{n}{d} + \binom{n}{d-1} + \cdots + \binom{n}{1} + \binom{n}{0}$$

Local random search heuristics



Small neighbourhood: Fast conversion to local optima

Huge neighbourhood: Similar to random search

Variable neighbourhood :

- Initially, big neighbourhood, then decrease
- Challenging: Not to decrease too fast

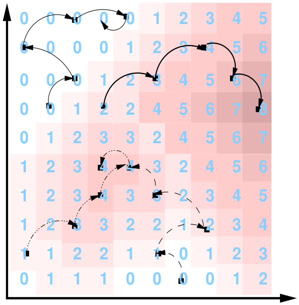
Local random search heuristics

Local optima avoidance: Multistart

- Search applied t times on problem
- Probability amplification : respectable result also with low success probability

  Assume: success probability $\delta > 0$ for one iteration

  After t iterations overall success probability: $1-(1-\delta)^t$



Metropolis algorithms

Local random search: only multistart can avoid local optima.

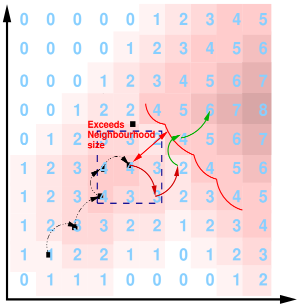
Metropolis approach accepts also search points that decrease fitness value

$F(x') > F(x) \Rightarrow x'$ discarded with prob.

$$1 - \frac{1}{e^{(F(x') - F(x))/T}}$$

$T \to 0$ random search

$T \to \infty$ uncontrolled local search



Simulated annealing

Choice of optimal T not easy $\Rightarrow$ Change during optimisation

Initially: T should allow to 'jump' to other regions of the search space with increased fitness value

Finally: Process should gradually 'freeze' until local search approach propagates the local optimum in the neighbourhood

Analogy to natural cooling processes in the creation of crystals:

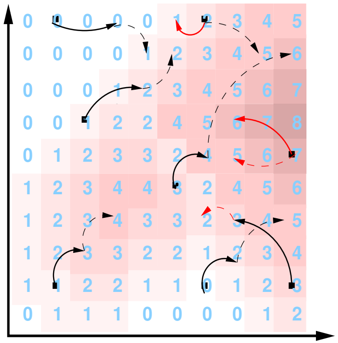
- Temperature gradually decreased so that Molecules that could move freely at the beginning are slowly put into their place

Simulated annealing

No natural problem known for which it has been proved that Simulated Annealing is sufficiently more effective than the Metropolis algorithm with optimum stationary temperature

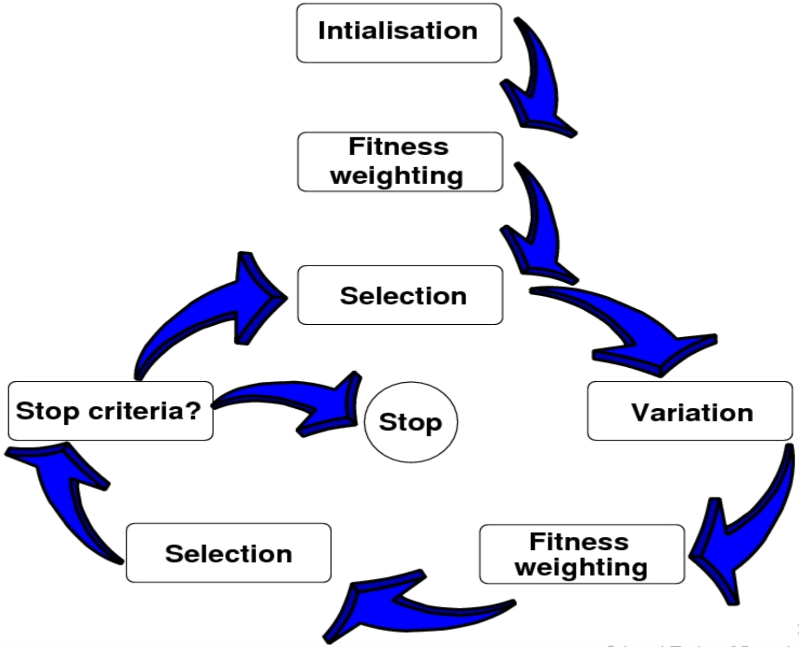
Artificially constructed problems exist, for which this could be shown

Evolutionary algorithms



Utilise evolution principles for optimisation purposes

Evolutionary algorithms combine Genetic algorithms, Evolution strategies, Evolutionary programming and Genetic programming


Intialisation
Fitness weighting
Selection
Variation
Fitness weighting
Selection
Stop criteria?
Stop

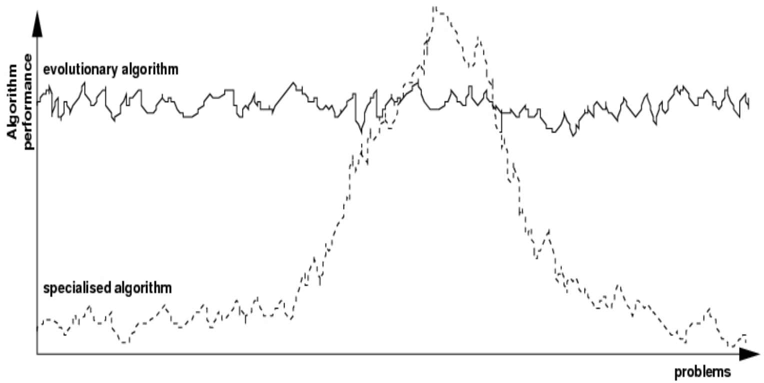
Restrictions of evolutionary approaches

It has been argued that

- Problem specific algorithms better than evolutionary on small subset of problems
- Evolutionary algorithms better on average over all problems

Evolutionary algorithms proposed as general purpose optimisation scheme

Restrictions of evolutionary approaches



Restrictions of evolutionary approaches

Can an algorithm be suited for 'all' problems?

What does 'all problems' mean?

Can one algorithm be better on average than another algorithm on 'all' problems?

Restrictions of evolutionary approaches

Can an algorithm be suited for 'all' problems?

- Distinct coding of the search space
- Various fitness functions

What does 'all problems' mean?

Can one algorithm be better on average than another algorithm on 'all' problems?

Restrictions of evolutionary approaches

Can an algorithm be suited for 'all' problems?

- Distinct coding of the search space
- Various fitness functions

What does 'all problems' mean?

- All possible representations and sizes of search space
- All possible fitness functions
- Every single point is the optimum point in several of these problems

Can one algorithm be better on average than another algorithm on 'all' problems?

Restrictions of evolutionary approaches

Wolpert and Macready formalised this assertion:[18]

- Set of all functions $f : S \to W$ given by F
- S and W finite (every computation on physical computers only has finite resources)
- Fitness function evaluated only once per search point
- $A(f)$ is number of points evaluated until optimum is found

[18] D.H. Wolpert and W.G. Macready, *No Free Lunch Theorems for Optimisation*, IEEE Transactions on Evolutionary Computation 1, 67, 1997.

Restrictions of evolutionary approaches

No free lunch theorem

Assume that the average performance of an algorithm in the No Free Lunch Scenario for S and W is given by $A_{S,W}$, the average over all $A(f), f \in F$. Given two algorithms A and A', we obtain $A_{S,W} = A'_{S,W}$

- This means that two arbitrary algorithms perform equally well on average on all problems

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

W.l.o.g.: $W = \{1, \ldots, N\}$

We consider sets $F_{s,i,N}$ of all functions f on a search space of non-visited search points of size s with at least one x with $f(x) > i$

Observe that for every function f and every permutation π also f_π belongs to $F_{s,i,N}$

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Proof by induction over $s := |S|$.
Induction start: $s = 1$

Every algorithm has to choose the single optimum search point with its first request.

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s-1 \rightarrow s$
Define $a : S \rightarrow \mathbb{N}$ so that $\forall x \in S$ the share of functions with $f(x) = j$ is exactly $a(j)$.

This is independent of x, since all permutations f_π of a function f also belong to $F_{s,i,N}$

$a(j)$ is therefore the probability to choose a search point with fitness value j (Independent of the concrete algorithm A)

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s - 1 \rightarrow s$

With probability $a(j)$ an algorithm A finds a search point with fitness value j.

Count of functions $f(x) = j$ is equal to the number of functions $f_\pi(y) = j$, since all permutations of f are also in $F_{s,i,N}$.

The probability to achieve a fitness value $j > i$ is therefore independent of the algorithm.

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Induction: $s-1 \rightarrow s$

With probability $a(j)$ an algorithm A finds a search point with fitness value j.

If $j \leq i$, x is not optimal in scenario $F_{s,i,N}$ and the new scenario is $F_{s-1,i,N}$

Restrictions of evolutionary approaches

Proof of the No Free Lunch Theorem

Summary – in other words:

For any two algorithms we can state a suitable permutation of the Problem-function for one problem (i.e. state another problem), so that both algorithms in each iteration request identical search points.

- Especially, since every search point could be optimal, there are always algorithms that request the optimal search point right from the start.

Restrictions of evolutionary approaches

NFL is possible, since ALL algorithms and ALL problems are considered

Is there an NFL valid in smaller, more realistic scenarios?

In [19] a similar theorem was proved for more realistic problem scenarios.

[19] S. Droste, T. Jansen and I. Wegener, *Perhaps not a free lunch but at least a free appetizer*, Proceedings of the 1st Genetic and Evolutionary Computation Conference, 1999.

Design aspects of evolutionary algorithms

Selection principles

Uniform selection

Individuals chosen uniformly at random

Deterministic selection

Deterministically choose the highest rated individuals for the selection

Threshold selection

Candidates for offspring population drawn uniformly at random from the t highest rated individuals

Design aspects of evolutionary algorithms

Selection principles

Fitnessproportional selection

- For population $x_i, \dots, x_n$ individual x_i chosen with

$$p(x_i) = \frac{f(x_i)}{f(x_1) + \cdots + f(x_n)}$$

- Draw random variable u from $[0, 1]$ and consider x_i if

$$p(x_1) + \cdots + p(x_{i-1}) < u \leq p(x_1) + \cdots + p(x_i)$$

- Frequently applied for evolutionary approaches

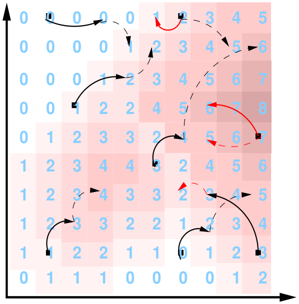
Design aspects of evolutionary algorithms

Selection principles

- Problems with Fitnessproportional selection
  - Linear modification of the fitness function ($f \rightarrow f + c$) results in different behaviour
  - When fitness values sufficiently separated, selection is nearly deterministic
  - When deviation in fitness values is small relative to absolute values, similar to uniform selection

Design aspects of evolutionary algorithms

Variation – Mutation



- Mutation creates one offspring individual from one individual
- Operators are designed for specific search spaces
- Shall apply only few modifications of individuals on average
- Distant individuals have smaller probability

Evolutionary algorithms

Mutation operators for individuals from $\mathbb{B}^n$ (similar operators for other search spaces):

Standard bit mutation

- Offspring individual created bit-wise from parent individual
- Every bit 'flipped' with probability p_m
- Common choice: $p_m = \frac{1}{n}$

1 bit mutation

- Offspring individual identical in all but one bit.
- This bit chosen uniformly at random from all n bits

Design aspects of evolutionary algorithms

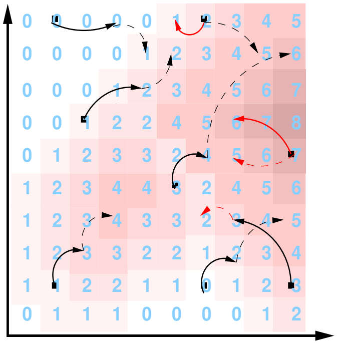
Variation – Crossover

- Crossover typically takes two individuals and results in one or two offspring individuals
    - Also crossover of more than two individuals possible
    - Often generalisations of the two-individual case
- Distinct crossover methods for various search spaces
- Crossover parameter p_c specifies the probability with which crossover (and not mutation) is applied for one selected individual
- In some cases (e.g. binary coded numbers) not all positions in the individual string are allowed to apply crossover on

Design aspects of evolutionary algorithms

Typical crossover variants

- One-point crossover
- k-point crossover
- Uniform crossover



Evolutionary algorithms

Crossover operators for $\mathbb{B}^n$
(Operators for other search spaced similar)

One-point crossover: Individual x'' from two individuals x and x' according to uniformly determined crossover position:

$$x''_j = \begin{cases} x_j & \text{if } j \leq i \\ x'_j & \text{if } j > i \end{cases}$$

Evolutionary algorithms

Crossover operators for $\mathbb{B}^n$

k-point crossover: Choose $k \leq n$ positions uniformly at random:

$$\begin{array}{rl}
x_1 = & x_{11}, x_{1,2}, \ldots, x_{1,k_1} | x_{1k_1+1}, \ldots, x_{1k_2} | x_{1k_2+1}, \ldots, x_{1n} \\
x_2 = & x_{21}, x_{2,2}, \ldots, x_{2,k_1} | x_{2k_1+1}, \ldots, x_{2k_2} | x_{2k_2+1}, \ldots, x_{2n} \\
\hline
y_1 = & x_{11}, x_{1,2}, \ldots, x_{1,k_1} | x_{2k_1+1}, \ldots, x_{2k_2} | x_{1k_2+1}, \ldots, x_{1n} \\
y_2 = & x_{21}, x_{2,2}, \ldots, x_{2,k_1} | x_{1k_1+1}, \ldots, x_{1k_2} | x_{2k_2+1}, \ldots, x_{2n}
\end{array}$$

Evolutionary algorithms

Crossover operators for $\mathbb{B}^n$

Uniform crossover: Each bit chosen with uniform probability from one of the parent individuals

Design aspects of evolutionary algorithms

Discussion

- Evolutionary algorithms are easy to implement when compared to some complex specialised approaches
- However, Evolutionary algorithms are computationally complex
- It is therefore beneficial to implement efficient variants to the distinct methods

Questions?

Stephan Sigg
stephan.sigg@cs.uni-goettingen.de

Literature

- C.M. Bishop: Pattern recognition and machine learning, Springer, 2007.
- P. Tulys, B. Skoric, T. Kevenaar: Security with Noisy Data – On private biometrics, secure key storage and anti-counterfeiting, Springer, 2007.
- R.O. Duda, P.E. Hart, D.G. Stork: Pattern Classification, Wiley, 2001.