Machine Learning and Pattern Recognition in Bioinformatics

Peter Sykacek[1]

Vienna Science Chair of Bioinformatics

Department of Biotechnology

BOKU University

peter.sykacek@boku.ac.at

Lecture 1 & 2

- Concepts in Machine Learning and Pattern Recognition (ML & PRN)
- Classification of Problems and Methods in ML & PRN
- Basics of Matrices in ML & PRN and Introduction to MatLab

Machine Learning

and Pattern Recognition (ML & PRN)

Discipline which emerged at the intersection of

- Computer Science and
- Applied Mathematics / Statistics

Advantage: Same objectives but less formal than Statistics; Thus accessible with less mathematics.

Idea: Data analysis with clever algorithms.

$->$ inevitably requires qualitative understanding of data analysis and implication of algorithms.

Brief Guideline to Success

ML & PRN provide powerful but dangerous tools.
Avoiding a self inflicted chain saw massacre requires:

- Understand general concepts in data analysis.
- Know the ever growing repertoire of ML & PRN methods.
- Understand implications of all methods you apply.
- Compare competing methods to avoid bias.

Do not apply published approaches in interdisciplinary fields like computational biology, etc. uncritically: Data analysis is not necessarily done by experts!

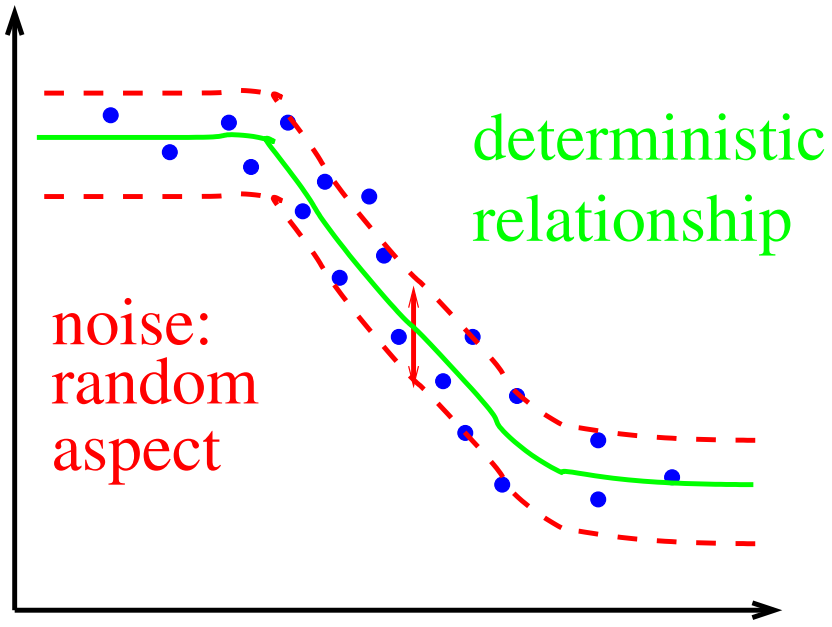
Problem 1: Noise

ML & PRN tries to “learn” reasonably good models from measured data. (in statistics they call this inference).

Model - a possibly implicit quantitative abstraction of nature implying:

- imperfection due to unknown or unmodeled factors influencing nature.
- problem of separating the data generating mechanism into:
  - explicable deterministic aspects
  - noise component which contains the rest

Noise

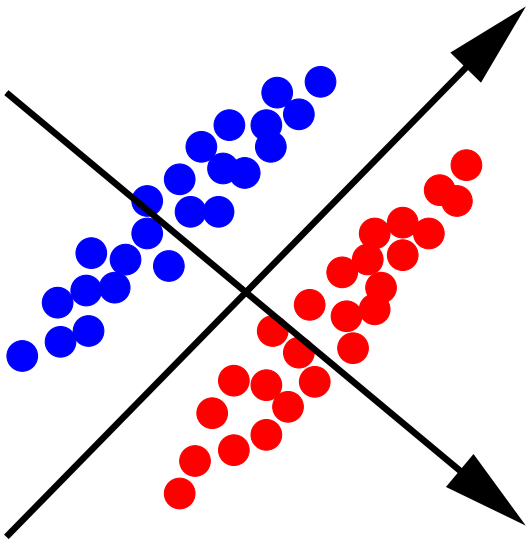


Inexplicable aspects are summarised by a noise component (red error bars).

Problem 2: Modelling Goal

Result = Data + Model!

Linear discriminant (LDA) and principle component analysis (PCA) give different projections of the same data.



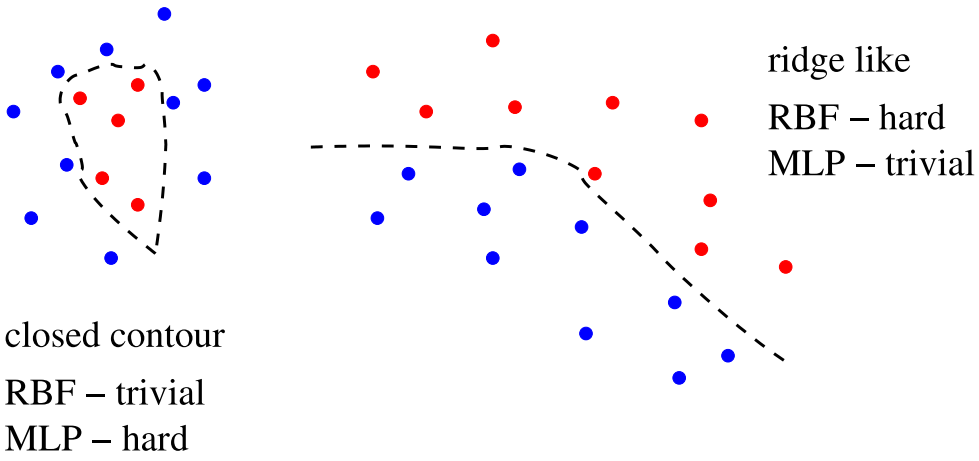
Both use linear projections!

$$t_{\mathrm{PCA}} = \theta_{\mathrm{PCA}}^T x$$

$$t_{\mathrm{LDA}} = \theta_{\mathrm{LDA}}^T x$$

Problem 3: Methods Bias

Example: dichotomous classification (2 labels)



Methods bias leads to results being influenced by the analysis method, motivating the no free lunch theorem. Lack of overall winning algorithms requires comparing competing methods to ensure acceptable performance.

The Nature of Data

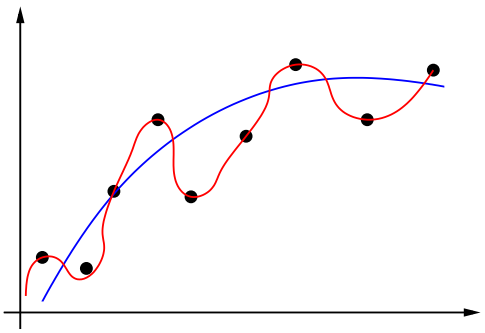
- Discrete valued observations (e.g. class labels).
- Continuous valued observations (e.g. measurements).

Measurement processes involve errors which arise from noise (fluctuations) that are or can not be captured:

- Measurement noise.
- Wrong classifications (e.g. disease state).
- Simplified Models.

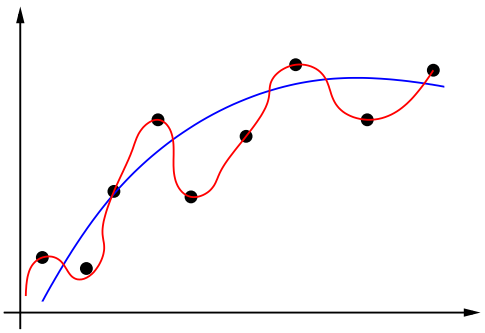
Individual data points do thus not reflect ground truth. Data analysis uses replicates to estimate noise and model the remaining aspects as good as possible.

Fundamental Principle in Data Analysis

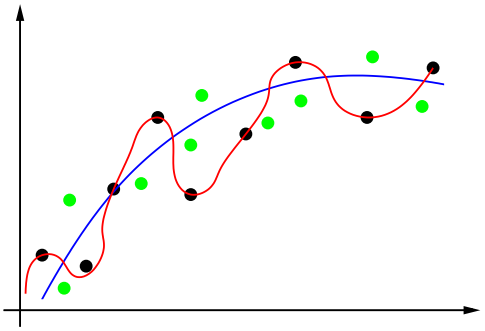


Which is the better model? Why is that the case?

Fundamental Principle in Data Analysis



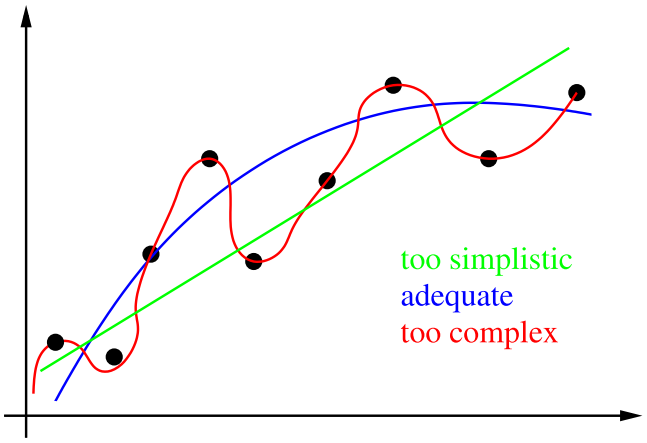
Which is the better model? Why is that the case?



Find (or abstract from) the underlying model that generated the data.

Adequate models

Capture underlying structure and avoid overfitting. “Fiddle parameters” affecting model complexity can have adverse effects.

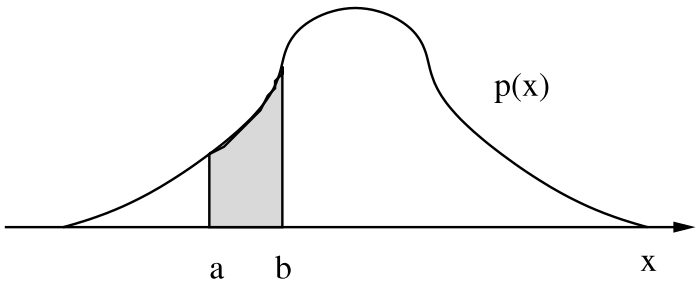


Idea: overfitting is a result of tuning the model towards the training data. Over or under-complex models that do not capture the underlying data generating mechanism will perform worse on novel data obtained from the generating model than an appropriate model.

Definition: PDF

Data analysis is inherently connected with the concept of random variables. A random variable is a non deterministic quantity where repeated observations differ and are generated according to some overall property. Properties of random variables are for example captured by an associated probability density function (pdf), $p(x)$.

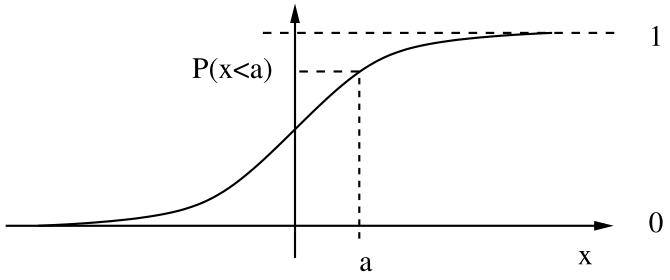
The pdf allows deducing the probability that a new realisation falls into a particular set, $P(x \in [a,b]) = \int_{x=a}^{b} p(x)dx$.



Definition: CDF

An equivalent characterisation for univariate random variables is provided by the so called cumulative distribution function (cdf), $F(x)$.

The cdf $F(x)$ denotes the probability that a realisation of the random variable is smaller than x. $F(a)$ is thus the probability $P(x < a)$.



$->$ the pdf $p(x) = \frac{dF(\xi)}{d\xi}|_{\xi=x}$ is the derivative of the cdf at x.

Classification of Methods & Problems

Data analysis can be grouped into two categories:

1. Supervised Learning methods are used for regression problems.
2. Unsupervised Learning methods are used for exploratory data analysis.

Regression Problems

Regression is concerned with data sets of size N of the type $\mathcal{Z} = \{(y_1, x_1), (y_2, x_2), ...(y_N, x_N)\}$, with the tuples (y_n, x_n) drawn from an unknown joint pdf $p(y, x)$. The learning task is modelling the dependent variable, y, as a function of the independent variable x.

Typical regression models:

Linear regression: $p(y|x) = \mathcal{N}(kx + d, \lambda)$ and $y \in \Re$.

Logistic regression: $P(y|x) = \mathrm{cdf}_{\mathrm{logistig}}(kx + d)$ and $y \in \{0, 1\}$.

Exploratory Data Analysis

Exploratory data analysis searches for unknown structure in a data set of size N, $\mathcal{X} = \{x_1, x_2, ..., x_N\}$, with $x_n \in \Re^d$ drawn from an unknown pdf $p(x)$. The learning task is modelling x as a function of an unobserved (latent) variable t, providing an easy to grasp summary of the data.

Typical models for exploratory data analysis:

Mixture density models:
$p(x_n) = \sum_k P(t_n = k)p(x_n|t_n = k)$, and
$t_n \in \{1, .., K\}$.

Continuous latent variable models:
$p(x_n) = \int p(t_n)p(x_n|t_n)dt$, $t_n \subset \Re^k$ and $d > k$.

Summary Part 1

- ML & PRN is concerned with algorithms which are used for data analysis.
- Data analysis learns models from noisy data.
- We know two types of problems:
  - Data which should be analysed for simplifying explanation. $->$ unsupervised learning.
  - Data which requires predicting responses from independent variables. $->$ supervised learning.
- Fundamental principle: Search for adequate models of appropriate complexity. Also requires comparing different models!

Matrices for Data Analysis

Important to simplify notation!
Definition n-dimensional Euclidean Space $\mathbb{R}^n$:
Definition of a matrix (n rows, m columns):

$$\boldsymbol{M} = \begin{pmatrix} m_{1,1} & \cdots & m_{1,m} \\ \vdots & \ddots & \vdots \\ m_{n,1} & \cdots & m_{n,m} \end{pmatrix} = (\boldsymbol{m}_1, \cdots, \boldsymbol{m}_m) \text{ and } \boldsymbol{m}_i \in \mathbb{R}^n$$

MatLab: $>> M = [[a, b, c]; [d, e, f]; ...];$ What are the $\boldsymbol{m}_i$?
Important: $>>$ help $<$ *commandname* $>$, where one may type any MatLab command name, provides help for the corresponding command.

Matrix Operations

Transposition: $\boldsymbol{B} = \boldsymbol{A}^T$, $\forall n, m : b_{m,n} = a_{n,m}$
MatLab: $>> B = A';$

Addition ($\boldsymbol{A}$, $\boldsymbol{B}$ equal size):
$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$, $\forall n, m : c_{n,m} = a_{n,m} + b_{n,m}$
MatLab: $>> C = A + B;$
Associative and commutative?

Matrix times constant:
$\boldsymbol{B} = \lambda \boldsymbol{A}$ $\forall n, m : b_{n,m} = \lambda a_{n,m}$
MatLab: $>> B = lambda * A;$
Associative and commutative?

Matrix Multiplication

Matrix Inner Product ($\boldsymbol{A}$'s column no. equals $\boldsymbol{B}$'s row no.): $\boldsymbol{C} = \boldsymbol{AB}\ \forall n, m:\ c_{n,m} = \sum_i a_{n,i} b_{i,m}$
MatLab: $>> C = A * B;$
Associative and commutative?
Note: $(\boldsymbol{AB})^T = \boldsymbol{B}^T \boldsymbol{A}^T$
However: $(\boldsymbol{A} + \boldsymbol{B})^2 = \boldsymbol{A}^2 + \boldsymbol{AB} + \boldsymbol{BA} + \boldsymbol{B}^2$

Hadamard Product ($\boldsymbol{A}$, $\boldsymbol{B}$ equal size):
$\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B},\ \forall n, m : c_{n,m} = a_{n,m} b_{n,m}$
MatLab: $>> C = A. * B;$
Associative and commutative?

Matrices and Data Analysis

Just to make sure that you see the connection between matrices and data analysis, here an example:

Assume N samples of k "input" measurements collected in $\boldsymbol{x}_n$ and one dependent variable y_n, which we intend modelling as a function $f(\boldsymbol{x}_n, \boldsymbol{\Theta})$ parameterised by $\boldsymbol{\Theta}$. This type of modelling is called *regression*.
We can only move on deciding on a particular $f(\boldsymbol{x}_n, \boldsymbol{\Theta})$.
For simplicity we assume that the best guess of y_n is obtained as linear combination of $\boldsymbol{x}_n$. This allows writing:

$$y_n = \sum_k \boldsymbol{x}_n[k]\boldsymbol{\Theta}[k], \text{ or } y_n = \boldsymbol{x}_n^T \boldsymbol{\Theta}$$

Model fitting I

is often done by minimising least squares differences

$$\hat{\boldsymbol{\Theta}} = \text{argmin}_{\boldsymbol{\Theta}} \left(\sum_n \left(y_n - \boldsymbol{x}_n^T \boldsymbol{\Theta} \right)^2 \right)$$

Importance of thinking in terms of matrices:

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_N^T \end{pmatrix} \text{ and } \boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_N \end{pmatrix}$$

Model fitting II

We can then immediately write

$$\text{lsd} = \sum_n \left(y_n - \boldsymbol{x}_n^T \boldsymbol{\Theta} \right)^2 = (\boldsymbol{X}\boldsymbol{\Theta} - \boldsymbol{y})^T (\boldsymbol{X}\boldsymbol{\Theta} - \boldsymbol{y})$$

which contains no sums any more and is an extremely convenient method for deriving model fitting procedures and code for numerical tools like MatLab.

MatLab:$>> y\_d = X * theta - y;$

$>> LSD = y\_d' * y\_d;$

two lines to be more efficient!

The Least Squares Optimum

is, as previously discussed, obtained as $\hat{\boldsymbol{\Theta}} = \text{argmin}_{\boldsymbol{\Theta}}(\text{lsd})$. Similar to unconditional optimisation of functions:

- We take the derivative with respect to the quantity we want to optimise for.
- and set the derivative equal to zero.

$$\text{lsd} = \boldsymbol{\Theta}^T \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\Theta} - 2\boldsymbol{\Theta}^T \boldsymbol{X}^T \boldsymbol{y} + \boldsymbol{y}^T \boldsymbol{y}$$

Special: we take derivatives w.r.t the vector $\boldsymbol{\Theta}$!

Gradient Vectors of LSD Expression

$$\nabla_{\boldsymbol{\Theta}}(\boldsymbol{\Theta}^T \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\Theta}) = 2\boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\Theta}$$

$$\nabla_{\boldsymbol{\Theta}}(-2\boldsymbol{\Theta}^T \boldsymbol{X}^T \boldsymbol{y}) = -2\boldsymbol{X}^T \boldsymbol{y}$$

and

$$\nabla_{\boldsymbol{\Theta}}(\boldsymbol{y}^T \boldsymbol{y}) = \boldsymbol{0}$$

The gradient vector w.r.t. $\boldsymbol{\Theta}$ is thus given by:

$$\nabla_{\boldsymbol{\Theta}}(\text{lsd}) = 2(\boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\Theta} - \boldsymbol{X}^T \boldsymbol{y}).$$

Solution: $2(\boldsymbol{X}^T \boldsymbol{X} \hat{\boldsymbol{\Theta}} - \boldsymbol{X}^T \boldsymbol{y}) = 0$ or $\boldsymbol{X}^T \boldsymbol{X} \hat{\boldsymbol{\Theta}} = \boldsymbol{X}^T \boldsymbol{y}$

$->$ we have to remove $\boldsymbol{X}^T \boldsymbol{X}$ from the left side of the equality.

Square Matrices

Rank of a matrix $r(\boldsymbol{A})$: number of linearly independent columns (or rows, that's the same) of $\boldsymbol{A}$.

Square matrix $\boldsymbol{A}$ is a square matrix if no. rows equals no. cols, that is: $n = m$.

Square matrix $\boldsymbol{A}$ is non-singular if rank $r(\boldsymbol{A}) = n$.

The Determinant and the inverse are defined for square matrices.

Note: $\boldsymbol{X}^T\boldsymbol{X}$ from the previous slide is a square matrix!

The Determinant

The determinant $|\boldsymbol{A}|$ (MatLab: $>> det(A)$) converts a square matrix to a real number.

$|\boldsymbol{A}| = 0$ implies that $\boldsymbol{A}$ is singular
$|\boldsymbol{A}| = \prod_n \lambda_n$, where λ_n are the eigenvalues of $\boldsymbol{A}$
For $\boldsymbol{A}$ upper/lower diag. matrix: $|\boldsymbol{A}| = \prod_{i=1}^{n} a_{i,i}$.

Recursive definition w.r.t j-th row (high school!):

$$|\boldsymbol{A}| = \sum_i (-1)^{i+j} a_{i,j} |\boldsymbol{A}_{i,j}| \text{ ,where } \boldsymbol{A}_{i,j} = \begin{pmatrix} \cdots & | & \cdots \\ \_\_ & a_{i,j} & \_\_ \\ \cdots & | & \cdots \end{pmatrix}$$

$|\boldsymbol{A}_{i,j}|$ is the determinant of the submatrix when removing the j-th row and i-th column.

Diagonal and Identity Matrices

Diagonal matrix: $\boldsymbol{A} = \mathrm{diag}(a_{1,1}, .., a_{n,n})$, defines a matrix with the only non zero elements located on the main diagonal

MatLab: $>> A = diag(a);$ % places $\boldsymbol{a}$ into main diagonal of $\boldsymbol{A}$

$>> a = diag(A);$ % places main diagonal of $\boldsymbol{A}$ into $\boldsymbol{a}$

Identity matrix: $\boldsymbol{I} = \mathrm{diag}(1, .., 1)$
neutral element of matrix multiplication:
$\boldsymbol{IA} = \boldsymbol{AI} = \boldsymbol{A}$
MatLab: $>> I = eye(n);$ % generates an $[n \times n]$ identity matrix.

Inverse Matrix

If matrix $\boldsymbol{A}$ is non-singular, we get a non-singular matrix $\boldsymbol{B} = \boldsymbol{A}^{-1}$, such that, $\boldsymbol{BA} = \boldsymbol{AB} = \boldsymbol{I}$.
Matrix $\boldsymbol{B}$ is the inverse of $\boldsymbol{A}$

MatLab:$>> B = A\hat{}(-1)$

Remarks: $(\boldsymbol{A}^{-1})^T = (\boldsymbol{A}^T)^{-1}$ and $(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$
Matrix $\boldsymbol{A}$ is orthonormal if $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$, hence $\boldsymbol{A}^T = \boldsymbol{A}^{-1}$
Examples: projection to principal axis (PCA)
Permutation matrix $\boldsymbol{P}$ in every row and column one 1 entry, otherwise 0

Using Inverse Matrices

Consider resolving:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

w.r.t $\boldsymbol{x}$ for $\boldsymbol{A}$ square $[n \times n]$, then:

$$\boldsymbol{A}^{-1}\times \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \text{ leads to } \boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$$

Such operations are often found in data analysis, e.g. in finding least squares solutions.

Most important: Unlike in one dimensional operations, you must here multiply from the left! About 80% get this wrong in the exam!

Moore Penrose Pseudo Inverse

Inverting ill conditioned (close to singular) matrices involves quantities close to machine precision. Results derived from such inverse matrices result in large numerical errors.

Practical rule - never use matrix inversion, always use the Moore Penrose pseudo inverse.

$$\boldsymbol{A}^{+} = \lim_{\delta \to 0} (\boldsymbol{A}^T\boldsymbol{A} + \delta \boldsymbol{I})^{-1}\boldsymbol{A}^T$$

MatLab: $>> A\_plus = pinv(A);$
If $\boldsymbol{A}$ square and not ill-conditioned:
$\boldsymbol{A}^{+}\boldsymbol{A} = \boldsymbol{A}^{-1}(\boldsymbol{A}^T)^{-1}\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$

Solution for Least Squares Optimum

We have to solve:

$$\boldsymbol{X}^T\boldsymbol{X}\hat{\boldsymbol{\Theta}} = \boldsymbol{X}^T\boldsymbol{y}$$

for $\hat{\boldsymbol{\Theta}}$. We thus multiply both sides of the equality from the left with $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$. Left hand slide:

$$(\boldsymbol{X}^T\boldsymbol{X})^{-1}(\boldsymbol{X}^T\boldsymbol{X})\hat{\boldsymbol{\Theta}} = \boldsymbol{I}\hat{\boldsymbol{\Theta}} = \hat{\boldsymbol{\Theta}}$$

The solution is thus:

$$\hat{\boldsymbol{\Theta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

in MatLab: $\text{theta\_hat} = \text{pinv}(X' * X) * X' * y;$

Linear Regression in MatLab I

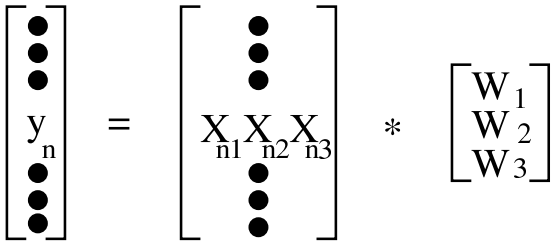
For efficiency express predictions simultaneously as $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta}$. MatLab code:

```
function [y]=linpred(Xtst, W)
% function [y]=linpred(Xtst, W) predicts outputs for linear regression
% Note: we do assume that X is properly augmented allowing for an intercept
% term (i.e. an input independent constant offset)
% Xtst:a [nsmpl x nrinputs] matrix of independent variables (test data).
% W:   a [nrinputs x 1] column vector of linear regressor.
% y:   predicted responses
% BTW : % is MatLabs comment symbol which makes it ignore the rest of the l
y=Xtst*W;
```

This function may be called in MatLab as:

```
% Assuming that xtst and W are properly defined and matching in dimension,
% y_pred contains the predictions at all samples stored in xtst:
[y_pred]=linpred(xtst, W);
```

Significance of Matrix Expressions



$y_n = \sum_c X_{n,c} * W_c$, n: row index in y and X; c: column index in X and row index in W. MatLab calculates:

```
[nrw,mcol]=size(Xtst);
for n = 1:nrw
  y(n) = 0;
  for m = 1:mcol
    y(n)=y(n)+Xtst(n,m)*W(m);
  end
end
% y=Xtst*W is MUCH faster!
```


More on MatLab Matrices

Defining Matrices:

```
xtst=[[0.1, 2.3, -5]; [1, -0.3, 4.5]]; % 2 row by 3 column matrix
xtst=rand(100, 2); % a 100 by 2 matrix of uniform random numbers
% loads tab delimited file (no headers!) and generates matrix data
load -ascii data.txt
```

Accessing matrices (indices are 1 - based):

```
xtst(3,2)=5;     % set the entry in row 3 and column 2 to 5
cv1=xtst(:,1);   % writes the first column vector in xtst into cv1
xtst(5,:)=rw5;   % stores the elements of rw3 as 5-th row in xtst
```

Calculate $\sum_n x_n^T A x_n \forall n$ assuming $x_n^T = X(n,:)$ and avoid a loop:

```
sum_xTAx=sum(sum((X*A).*X, 2)); % .*: element wise matrix multiplication
```

$(X * A).* X$ has the n-th row vector:
$[x_{n,1} * A(1,:) * x_n, x_{n,2} * A(2,:) * x_n, .., x_{n,m} * A(m,:) * x_n]$
$\mathrm{sum}((X * A).* X, 2)$ is thus a column vector containing in the n-th row the expression $x_n^T A x_n$ and the final sum completes the calculation.

Linear Regression in MatLab II

Calculation of maximum likelihood coefficients as $\hat{\Theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$ and MatLab code:

```
function [W]=lintran(X, t)
% function [W]=lintran(X, t) calculates lsd optimum of linear regression:
% Note: we do assume that X is properly augmented allowing for an intercept
% term (i.e. an input independent constant offset)
% X:   a [nsmpl x nrinputs] matrix of independent variables.
% t:   a [nsmpl x 1] column vector of corresponding target values.
% W:   a [nrinputs x 1] vector containing the optimal regression
%      coefficients.
W=pinv(X'*X)*X'*t;
```

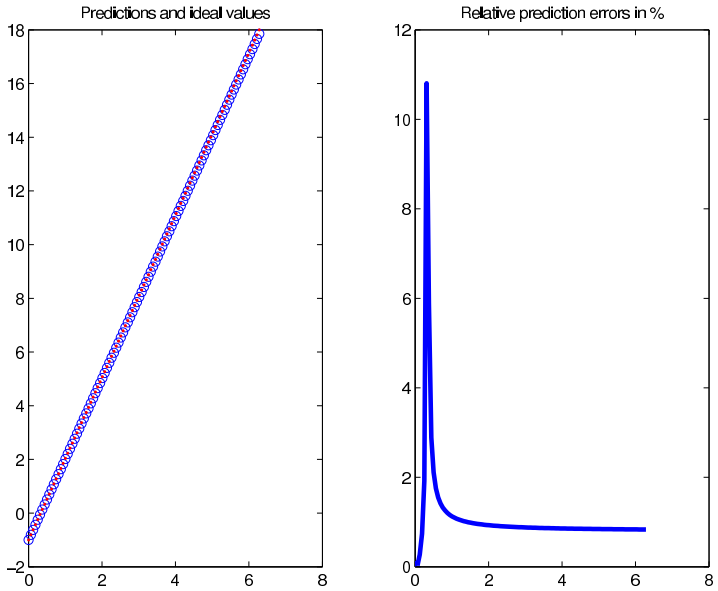
This function may be called in MatLab as:

```
% Assuming that xtran and t are properly defined and matching in dimension,
% W contains the optuimal regression coefficients:
[W]=lintran(xtran, t);
```


Test script combining both

```
xtrn=rand(100,1)*2*pi;
theta=[3;-1];
sdns=0.5;
t=xtrn*theta(1)+theta(2)+randn(size(xtrn))*sdns; % noisy targets
% we do now fit the linear regression model
xtrn=[xtrn, ones(size(xtrn))]; % add column of ones for intercept
[W]=lintran(xtrn, t);
% W can now be applied to the test data
xtst=[0:0.01:1]'*2*pi;
xtst=[xtst, ones(size(xtst))]; % add column of ones for intercept
[y_pred]=linpred(xtst, W);
% visualize quality of predictions:
y_ideal=xtst*theta; % using the known correct coefficients
subplot(1,2,1)
plot(xtst(:,1), y_ideal, 'bo', xtst(:,1), y_pred, 'r.')
title('Predictions and ideal values')
subplot(1,2,2)
plot(xtst(:,1), 100*abs(y_pred-y_ideal)./abs(y_ideal), 'b-', 'LineWidth',3)
title('Relative prediction errors in %')
```

Graphical output of script



The current figure (see figure command) can be stored as graphics file:

```
print -depsc lineval.eps   % stores the current figure as colour eps file.
```


MatLab Functions

MatLab functions like lintran and linpred are stored as m-files. MatLab's rule is to know a function under the m-file name! For calling

```
[W]=lintran(xtran, t);
```

the corresponding MatLab code must therefore be stored as file lintran.m.
MatLab functions use local variables.
All parameters are pass by value.
MatLab uses a search path for accessing functions. This path is modified using the command addpath.

MatLab Workspace and Scripts

Statements like:

```
xtrn=rand(100,1)*2*pi;
theta=[3;-1];
sdns=0.5;
t=xtrn*theta(1)+theta(2)+randn(size(xtrn))*sdns; % noisy targets
% we do now fit the linear regression model
xtrn=[xtrn, ones(size(xtrn))]; % add column of ones for intercept
[W]=lintran(xtrn, t);
```

are executed in MatLab

- by either typing each command into MatLabs command line
- or by writing all commands into a script file and executing the script.

Both cases generate all variables (above: xtrn, theta, sdns, t and W) in MatLabs workspace. After execution they are available for further use.

Workspace commands:

who, whos : display all variables at the workspace
load, save: allow loading and saving variables into/from the workspace
clear: remove variables from the workspace.

Scope ("lifetimes") of Variables I

In general, MatLab variables which are generated within a function have local scope with the following implications:

- They cease to exist as soon as the corresponding function terminates.
- Variables in MatLabs workspace which exist at the time the function gets executed are invisible.

Creating variable "foo" at the workspace: $\mathrm{foo} = [1, 2, 3]$; we can thus always create a second variable "foo" inside any function without implications for variable "foo" in MatLabs workspace.

Scope ("lifetimes") of Variables II

Exception to the previous rule are variables specifically defined to have global scope: $\text{global foo; foo} = [1, 2, 3];$ To force global scope, functions need to specify variables as such.

```
function useglobal % demonstrate global variable
global g_var
if isempty(g_var) % we initialise it
  g_var=1
end
g_var=g_var*2;
```

Calling useglobal n times results in g_var being 2^n. Using global variables at the MatLab prompt or in other functions requires defining them there as global as well.

Handling global variables

```
useglobal; useglobal; useglobal;
global g_var
disp(g_var); % prints 8
clear g_var % only removes the reference!
whos global % prints g_var (which still exists!)
useglobal;
global g_var
disp(g_var); % prints thus 16 !!
clear global g_var % removes the variable
```

clear global: removes all global variables
clear all: removes all variables including globals

Summary Part 2

Many models and algorithms in ML & PRN are best formulated and implemented via matrices.

MatLab is very efficient for dealing with matrices.

An important rule for getting fast implementations is avoiding loops and coding with matrix operations instead.

Computer science recipes like using modularised code and avoiding side effects are important.

Global variables can help speeding up code but violate the previous requirement and should only be used in a separate (final) implementation.

Machine Learning and Pattern Recognition in Bioinformatics

Peter Sykacek[1]

Vienna Science Chair of Bioinformatics

Department of Biotechnology

BOKU University

peter.sykacek@boku.ac.at

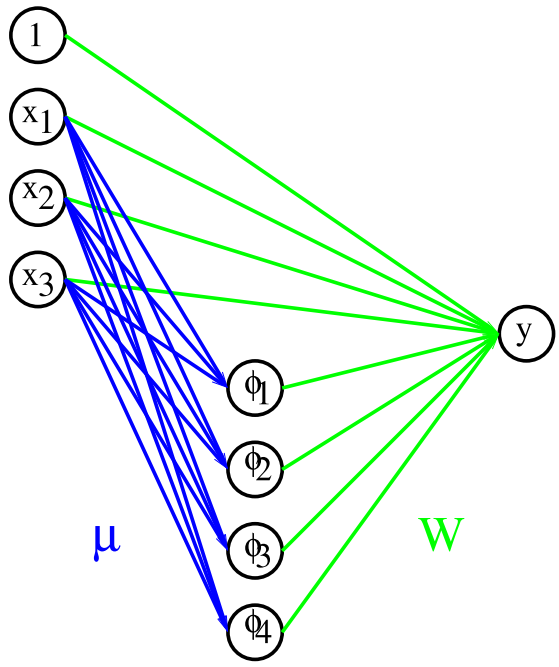
Modelling Aspects in ML & PRN

ML & PRN are concerned with algorithms which are useful in various applications of data analysis. Some properties:

- Like linear regression, ML & PRN fit models to data.
- ML & PRN typically use however more flexible models.
- Flexible models are advantageous for capturing nonlinear relationships often found in biology.
- ML & PRN approaches often suffer however from identifiability issues and overfitting (tendency of adding noise to the deterministic part of the model).

Greater flexibility inevitably renders fitting more difficult!

Towards flexible regression



Increase flexibility by introducing a nonlinear mapping $(\Phi_1, .., \Phi_4)$ parameterised by $\boldsymbol{\mu} = [\mu_1, .., \mu_4]$. Φ_k is for example a Gaussian or a *thin plate spline* centred at μ_k. Model fitting adjusts kernel parameters $\boldsymbol{\mu}$ and the regression coefficients $\boldsymbol{W}$.

Known as radial basis function network (RBF).

Thin plate splines

$\Phi_k(\boldsymbol{x}, \boldsymbol{\mu}_k) = r^2 \log(r)$ with $r^2 = (\boldsymbol{x} - \boldsymbol{\mu}_k)^T(\boldsymbol{x} - \boldsymbol{\mu}_k)$ (square Euclidean distance)

```
function [out]=thinplate(in, K)
% function [out]=thinplate(in, K) calculates size(K,1) output activations
% (columns of out) from each input (in) with each kernel (K) using thin
% plate splines. out has size(K,1) columns and size(in,1) rows.
% (C) P. Sykacek, 2009 <peter@sykacek.net>
nrin=size(in,1);
out=zeros(size(in,1), size(K,1));
for i=1:size(K,1)
  dist=[in-K(i*ones(nrin,1),:)]';
  dist=sum(dist.*dist,1)';
  out(:,i)=dist+eps; % prevent log of zeros
  lgout=log(out(:,i));
  % lgout(find(isinf(lgout)))=0; % this is some debugging output!
  out(:,i)=out(:,i).*out(:,i).*lgout;
end
out=[out, in, ones(nrin, 1)]; % concatenate with original data and ones.
```


Gaussian Kernel

$\Phi_k(\boldsymbol{x}, \boldsymbol{\mu}_k) = \exp(-0.5(\boldsymbol{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Lambda} (\boldsymbol{x} - \boldsymbol{\mu}_k))$, with $\boldsymbol{\Lambda}$ .. diagonal precision matrix.

```
function [out]=fastgauss(in, K, l)
% function [out]=fastgauss(in, K, l) calculates size(K,1) output
% activations (columns in out) from each input (in) with each
% kernel (K). out has size(K,1) columns and size(in,1) rows. The
% number of columns in in and K must be identical. l are the
% square roots of the precisions in each dimension.
% (C) P. Sykacek, 2009, <peter@sykacek.net>
nrin=size(in,1);
out=zeros(size(in,1), size(K,1));
l=l.^2;
for i=1:size(K,1)
  dist=[(in-K(i*ones(nrin,1),:))];
  dist=sum(dist.*dist.*l(ones(nrin,1),:),2);
  out(:,i)=exp(-0.5*dist);
end
out=[out, in, ones(nrin, 1)];
```

Properties of RBF networks

RBF's are universal function approximators (e.g. Hartman *et al.*(1990)).

Meaning: by increasing the number of nonlinear kernels, Φ_k, "well behaved" functions can be approximated with arbitrary accuracy.

Unless we do fancy things with the nonlinearities, increasing the number of hidden units will increase the flexibility of the model.

Fitting RBF networks

Typically, RBF networks are trained by two separate steps:

Step 1: adjust the μ parameters of the nonlinear expansion using unsupervised methods (e.g. kernels resulting from mixture density estimation).

Step 2: use the data arising from the nonlinear projection to adjust the regression coefficients $\boldsymbol{W}$. This is exactly the procedure we know from linear regression.

A proposal for step 1: don't bother with fancy approaches and use a number of randomly drawn input vectors as kernel centres. Similarity to kernel regression and the Nadaraya-Watson kernel smoother.

Fast RBF training

```
function [net]=lin2nonlin_tran(Xtrn, ttrn, nrkernels, nrnets, nettyp, krnfk
% returns net, a network structure with a committee of nrnets
% RBF networks using thin plate spline (or gaussian) activation.
% If net is 't' we use thin plate spline if net is 'g' we use Gaussians.
% Use lin2nonlin_pred for prediction.
% Xtrn:    [nsamples x nrin] matrix of independent variables (regressors).
% ttrn:    [nsamplel x 1] vector of corresponding target values (responses)
% nrkernels: number of kernels to be used in every network
% nrnets:  number of networks in the committee we average over
% nettyp:  type of RBF function, 't' .. thin plate spline, 'g' .. Gaussian,
%          default .. linear
% krnfkt:  factor for defining the precision of Gaussian kernels
% net:     [nrkernels x 1] vector of data structures describing a committee
%          of trained RBF networks.
% net.typ: type of nonlinear activation (copy of nettyp)
% net.K:   kernel modes
% net.l:   square roots of precisions, used for Gaussians only.
% net.W:   corresponding regression coefficients.
```

MatLab provides the dot symbol for denoting components of data structures.

Fast RBF training - Code

```
function [net]=lin2nonlin_tran(Xtrn, ttrn, nrkernels, nrnets, nettyp, krnfk
nrsamples=size(Xtrn,1);
l=krnfkt/(max(Xtrn) - min(Xtrn));
for i=1:nrnets
    net(i).typ=nettyp;
    if nettyp == 't' || nettyp =='g'
        net(i).K=uniquerndsub(Xtrn, nrkernels);
        net(i).l=l;
    end
    if nettyp =='t'
        out=thinplate(Xtrn, net(i).K);
    elseif nettyp == 'g'
        out=fastgauss(Xtrn, net(i).K, net(i).l);
    else  % default linear, where we only augment with '1' for the intercep
        out=[Xtrn, ones(nrsamples, 1)];
    end
    net(i).W=lintran(out, ttrn);
end
```

RBF predictions

```
function [y]=lin2nonlin_pred(net, Xtst)
% function [y]=lin2nonlin_pred(net, Xtst) predicts targets for Xtst inputs
% net:   array of trained network structures (see lin2nonlin_tran).
% Xtst:  independent test data used as inputs in the regression model.
% y:     predictions obtained by averaging over all committee members.
nrnets=length(net);
nrsmpl=size(Xtst, 1);
y=zeros(nrsmpl, size(net(1).W,2));
for i=1:nrnets
    if net(i).typ=='t'
        out=thinplate(Xtst, net(i).K);
    elseif net(i).typ =='g'
        out=fastgauss(Xtst, net(i).K, net(i).l);
    else
        out=[Xtsts, ones(nrsmpl, 1)];
    end
    y=y+out*net(i).W/nrnets;
end
```


Assessing the RBF

A) How do we assess the RBF?

One possibility is monitoring how well $\hat{\boldsymbol{y}}$ predicts the targets $\boldsymbol{y}$ by evaluating the sum of squares which we minimise during training.

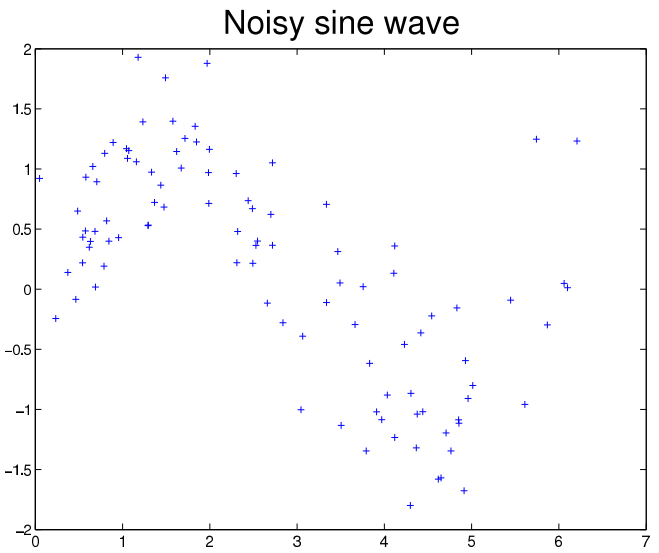
$$\mathrm{SSD}(\mathrm{RBF}_k) = (\boldsymbol{y} - \hat{\boldsymbol{y}}_k)^T(\boldsymbol{y} - \hat{\boldsymbol{y}}_k)$$

B) Which RBF parameters do influence performance?

1) number of kernels, $\mathrm{nrkernels}$
2) network type, nettyp
3) Gaussian kernel precision, krnfkt
4) number of networks in committee, nrnets

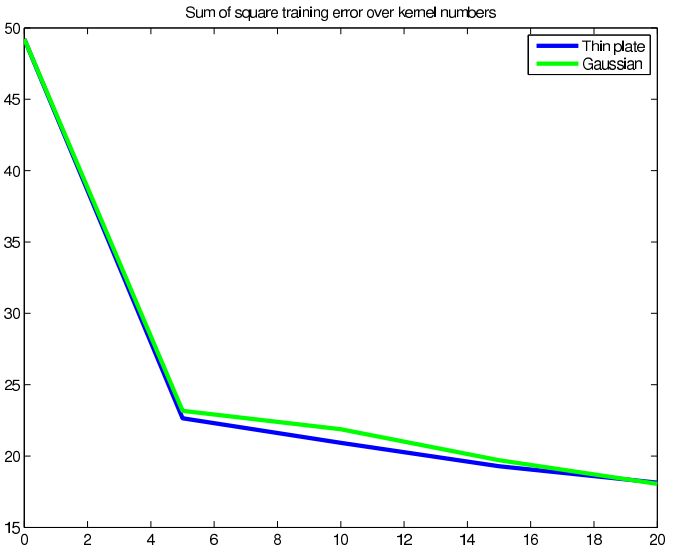
Assessment on toy data

Generate 100 samples of “noisy sine wave”.




```
x=rand(100,1)*2*pi;
sdn=0.5;
y=sin(x)+randn(100,1)*sdn;
plot(x,y,'b+','MarkerSize',5);
title('Noisy since wave',...
 'FontSize',24)
print -depsc noisysine.eps
save trandata.mat x y
```

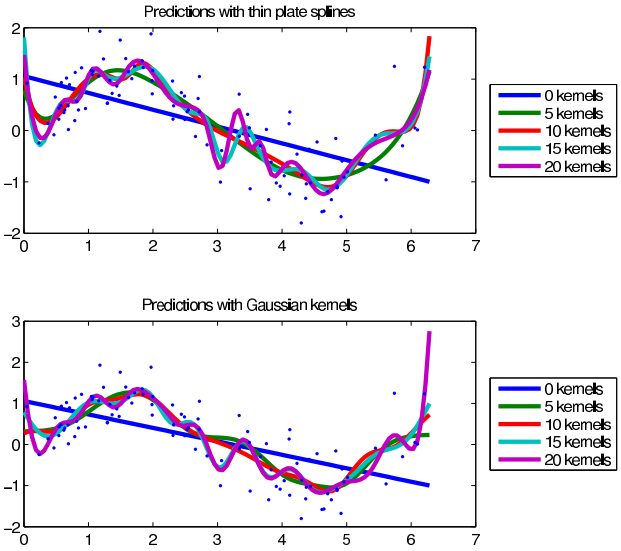

D over kernel numbers@



Based
squares
rors, bo
fit the tr
when i
number
the moc

Is this figure good news?

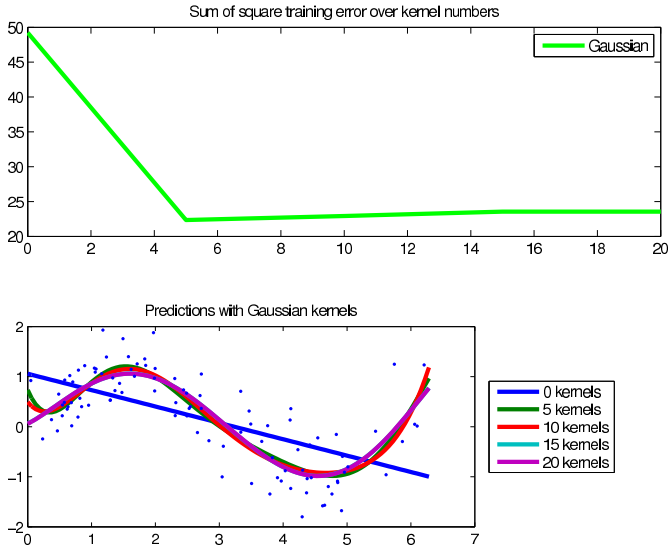
Visualising the models we fit



To get these figures we use as inputs data from the range 0 to 2π and visualise the corresponding predictions.

These visualisations show that decreasing the sum of squares of prediction errors below reasonable values is not really a good idea because it corresponds to fitting noise!

Model Complexity



A simple modification of the Gaussian kernel RBF shows that the number of kernels is not representative for model complexity. Exchange (line 2 in lin2nonlin_tran):

```
l=krnfkt/((max(Xtrn) - ...
              min(Xtrn)));
```

with:

```
l=krnfkt/((max(Xtrn) - ...
    min(Xtrn))*nkernels);
```

Model complexity depends on the effective number of parameters. Increasing the standard deviation of the kernels regularises.

Statistical Interpretation of ML & PRN

An experiment provides noisy data
$\mathcal{Z} = \{(y_1, \boldsymbol{x}_1), ..., (y_N, \boldsymbol{x}_N)\}$.
Note: $\boldsymbol{x}_n$ possibly multivariate i.e. vectors.

Based on $\mathcal{Z}$, we have an inference (or learning) problem of finding an "optimal" relation between $\boldsymbol{x}$ and y:

$$p(y|\boldsymbol{x}) = f(\boldsymbol{x}; \boldsymbol{\theta}) + \epsilon(\lambda)$$

Statistical Interpretation of ML & PRN

An experiment provides noisy data
$\mathcal{Z} = \{(y_1, \boldsymbol{x}_1), ..., (y_N, \boldsymbol{x}_N)\}$.
Note: $\boldsymbol{x}_n$ possibly multivariate i.e. vectors.

Based on $\mathcal{Z}$, we have an inference (or learning) problem of finding an "optimal" relation between $\boldsymbol{x}$ and y:

$$p(y|\boldsymbol{x}) = f(\boldsymbol{x}; \boldsymbol{\theta}) + \epsilon(\lambda)$$

Noise requires a deterministic and a random component.

$->$ Inherent uncertainty, y is a random variable.
We usually predict its expectation.

Two Stages of Inference

Parameter Inference:

Implies knowing $f(\boldsymbol{x};\boldsymbol{\theta})$ and the noise model $\epsilon(\lambda)$ up to unknown parameters ($\boldsymbol{\theta}$ and λ) which we will be inferring from data.

Two Stages of Inference

Parameter Inference:

Implies knowing $f(\boldsymbol{x};\boldsymbol{\theta})$ and the noise model $\epsilon(\lambda)$ up to unknown parameters ($\boldsymbol{\theta}$ and λ) which we will be inferring from data.

Model Inference:

A more realistic assumption is that the model class is unknown and we will be inferring model class (e.g. number of kernels) and parameters.

Assessing Model Parameters

Idea: subtract the deterministic part from y_n:

$$\epsilon_n = y_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})$$

For convenience introduce $\mathcal{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ and $\mathcal{D} = \{y_1, ..., y_N\}$. Assuming that ϵ_n are i.i.d samples, we get the likelihood function:

$$p(\mathcal{D}|\boldsymbol{\theta}, \lambda, \mathcal{X}) = \prod_n p(y_n|\boldsymbol{\theta}, \lambda, \boldsymbol{x}_n)$$

which is a suitable objective function to be maximised for $\boldsymbol{\theta}$ and λ.

Likelihood and Regression

Assuming N samples and ϵ_n being Gaussian:

$$\begin{aligned} p(y_n|\boldsymbol{x}_n; \boldsymbol{\theta}, \lambda) &= (2\pi)^{-0.5}\lambda^{0.5}\exp(-0.5\lambda(y_n - \boldsymbol{x}_n^T\boldsymbol{\theta})^2) \text{ and} \\ p(\mathcal{D}|\mathcal{X}; \boldsymbol{\theta}, \lambda) &= (2\pi)^{-\frac{N}{2}}\lambda^{\frac{N}{2}}\exp(-0.5\lambda\sum_n(y_n - \boldsymbol{x}_n^T\boldsymbol{\theta})^2) \end{aligned}$$

Taking the $\log$, we get the log likelihood:

$$\text{llh} = \frac{N}{2}(\log(\lambda) - \log(2\pi)) - 0.5\lambda(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})$$

which, if we consider maximising for $\boldsymbol{\theta}$ only, is a familiar expression.

$->$ minimising least squares assumes Gaussian noise!

Central Problem in ML & PRN

True model - linear regression, Gaussian noise:

$$p(y|\boldsymbol{x}) = f(\boldsymbol{x};\boldsymbol{\theta}) + \epsilon(\lambda)$$

$f(\boldsymbol{x};\boldsymbol{\theta}) = [1, \boldsymbol{x}^T]\boldsymbol{\theta}$ and $\epsilon(\lambda) = \mathcal{N}(\epsilon; 0, \lambda)$

Finite sample size and different model classes:

"Phone book": Perfect memorising of all y_n, modelling error 0, $\lambda -> \infty$, $p(\mathcal{D}|\boldsymbol{\theta}, \lambda, \mathcal{X}) -> \infty$.

$->$ likelihood, ssd and information theoretic measures are unsuitable for comparing models!

Central Problem in ML & PRN: controlling complexity

Summary Part 3

ML & PRN are like statistics concerned with fitting models to data.

The objective functions used in ML & PRN for model fitting are largely identical to those known in statistics.

Models used in ML & PRN are typically more flexible than those commonly used in statistics.

The core problem in ML & PRN is controlling complexity and comparing model classes.

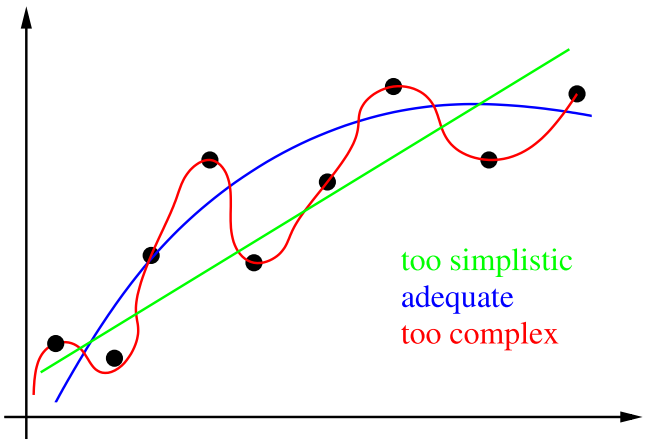
Complexity control and testing models

ML & PRN models can be arbitrary complex (cf. universal function approximator).

A very important aspect of applied ML & PRN is thus understanding and using methods for

- choosing models of appropriate dimension and
- comparing and assessing models.

Assessing model complexity



Two approaches for obtaining appropriate models:

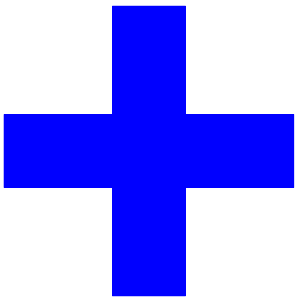
1) Testing different model classes (e.g. number of basis functions).

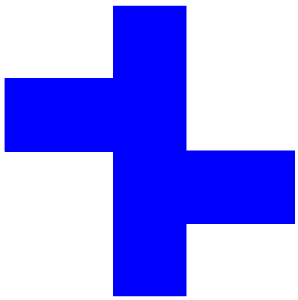
2) Allowing for highly flexible models and constraining the model parameters (e.g. regularisation).

Both approaches can either be dealt with by augmenting simple objective measures like SSD or by using validation data.

Guess the Correct "Model"

How many components?



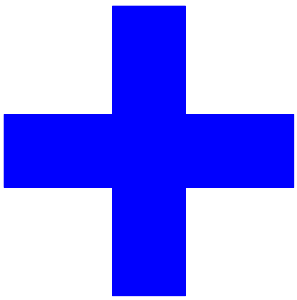


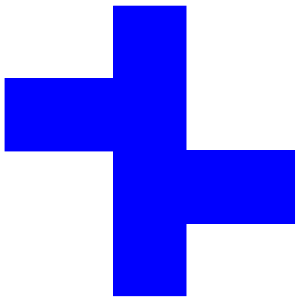
Object A

Object B

Guess the Correct "Model"

How many components?

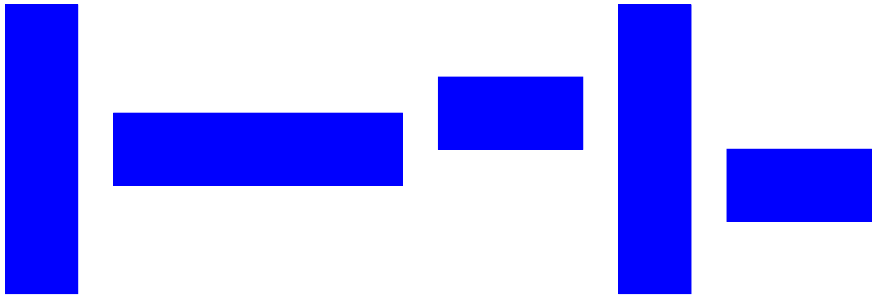




Object A

Object B

Most likely answer:



jump 2 TOC

ML & PRN in Bioinformatics (793.404), Peter Sykacek – p.24/42

Occam's Razor

We implicitly apply Occam's Razor

William of Occam (or Ockham) (1288 - 1348)

Entia non sunt multiplicanda sine necessitate: Entities are not to be multiplied without necessity.

Interpretation: One should always opt for an explanation in terms of the fewest possible number of causes, factors, or variables.

Material from `http://en.wikipedia.org/wiki/William_of_Ockham`.

Instances of model selection

Problems in context of model selection:

- Variable selection: search for input subsets which improve predictive performance or identify important variables (e.g. differentially expressed genes).
- Changepoint detection: Separating data into groups which show similar statistical properties.
- Clustering: (see above)
- Determining optimal model orders (applies to most ML& PRN methods!)
- Determining suitable noise characteristics.
- ...

Occam's Razor in ML & PRN

Model selection can be done by the following approaches:

- Adding a complexity penalty to the objective function. Many choices and active research field: AIC (Akaikes information criterion), BIC (Bayesian information criterion), MDL (minimum description length), etc.
- Using learning methods like Bayesian inference with Occam's razor built in.
- Using empirical approaches comparing model classes etc. by validation testing (computer simulation using independent data).

AIC, BIC and MDL

Akaike (1974) suggests using AIC:

$$L^k_{\mathrm{AIC}} = \log(p(\mathcal{D}|\hat{\boldsymbol{\theta}})) - K_k$$

Schwarz (1978) suggested BIC (asymptotic statistics):

$$L^k_{\mathrm{BIC}} = \log(p(\mathcal{D}|\hat{\boldsymbol{\theta}})) - \tfrac{1}{2} K_k \log(N)$$

Rissanen (1978) suggested MDL (information theory):

$$L^k_{\mathrm{MDL}} = L(\mathcal{D}|k) + L(k)$$

$p(\mathcal{D}|\hat{\boldsymbol{\theta}})$ ... likelihood of size N dataset $\mathcal{D}$, $\hat{\boldsymbol{\theta}}$ optimal model coefficients, K_k .. dimension, $L(\mathcal{D}|k)$ .. bit length modelling errors, $L(k)$ .. bit length model k. L^k_{MDL} is equivalent to the Bayesian log posterior model probability plus an additive constant which is independent of the models under consideration.

AIC, BIC should be maximised, MDL should be minimised.

Application to K-means clustering

K-means clustering is defined by a simple algorithm which iteratively minimises the cost function $J = \sum_{k=1} \sum_{x_{n,k} \in \mathcal{S}_k} (x_{n,k} - \mu_k)^2$.

Upon termination, we get cluster centres, cluster information and the optimal J as a result.

For using AIC and BIC, we need a quick and dirty hack for getting a likelihood measure from K-means. Assuming as generative model a Gaussian mixture, hard partitioning and an isotropic precision of $2 * \boldsymbol{I}$:

$\log(p(\mathcal{D}|k)) \approx \sum_k n_k \log(P_k) - J$ with $P_k = n_k/N$,

allows us calculating L^k_{AIC} and L^k_{BIC}.

MatLab script: calculation

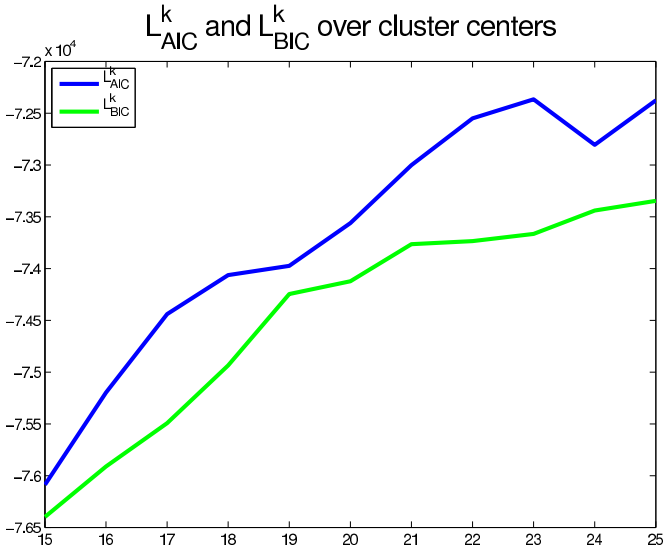
```
% use netlab library. modifiy path to install dir on your system!
addpath('~/sciwrk/mlbsrc/netlab/')
% load microarray data, etc. from a multi variable .mat file
load mca_tc
% analyse data containing the average of two replicates measured at
% 11 different timepoints stored in the [n-timepoints x n-genes] matrix
% mndat. For clustering similar genes into groups we transpose the data.
clustin=mndat';
% cluster range considered: 15 to 25 clusters.
mink=15; maxk=25;
% 10 repeated fits of k-means and maxit iterations until convergence
nrep=10; maxit=500;
method='A'; % AIC
[alloc_aic, centers_aic, nL_aic]=wrap_kmeans(clustin, mink, maxk, nrep,...
             method, maxit);
method='B'; % BIC
[alloc_bic, centers_bic, nL_bic]=wrap_kmeans(clustin, mink, maxk, nrep,...
             method, maxit);
```

MatLab script: illustrations

```
% note negation since above calculation provides penalized neg. llh
plot([mink:maxk], -nL_aic, 'b-', [mink:maxk], -nL_bic,...
 'g-', 'LineWidth', 3);
title('L_{AIC}^k and L_{BIC}^k over cluster centers', 'FontSize', 20);
legend('L_{AIC}^k','L_{BIC}^k', 'Location', 'NorthWest');
print -depsc kmeans_aicbic.eps
figure(2)
subplot(2,1,1)
plot(centers_aic', 'LineWidth', 3);
title('Cluster centers obtained using AIC penalty','FontSize', 18);
subplot(2,1,2)
plot(centers_bic', 'LineWidth', 3);
title('Cluster centers obtained using BIC penalty','FontSize', 18);
print -depsc kmeans_cluster.eps
```

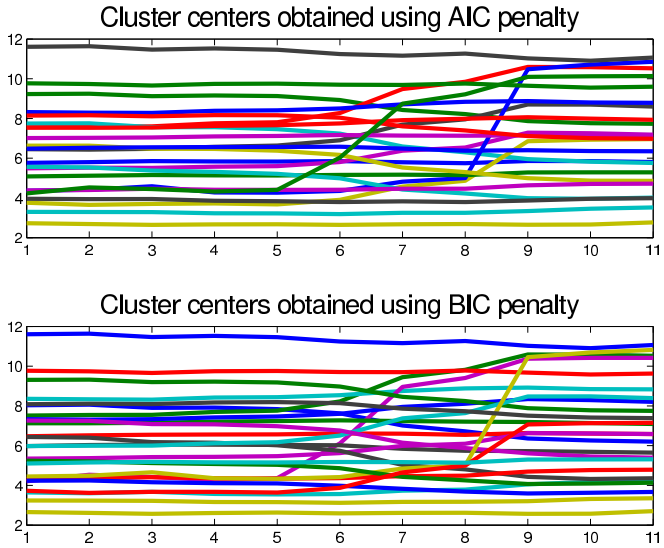
Note: see also MatLab code of functions!

K-means, AIC and BIC over kernels



Search for optimal kernel numbers peaks at values larger 25. Apparent difficulty of finding good optima at larger kernel numbers.

Cluster centres



Input: all (≈ 12500) genes measured. Several clusters represent house keeping stuff which does not change expression. As a negative side effect of having used unsupervised data analysis, we have thus to analyse the result further for interesting components.

Model selection using validation data

Applying a nonlinear RBF, we saw that increasing complexity will always improve the objective function.

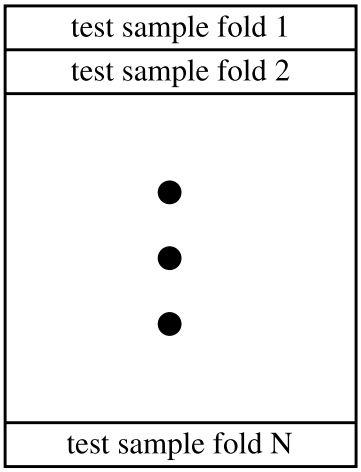
This problem can be overcome by using independent data to assess models with different complexity. Difficulties:

- Samples are usually very precious and thus rare.
- To get good results from model building, we can not afford wasting many samples for validation purposes.
- To get reliable assessments of model complexity, we also need sufficiently many samples.

Seems like a deadlock, however: $->$ iterative schemes which reuse data can help.

N-Fold Cross Testing

Sketch and MatLab like pseudo code




```
allres=[]; allpred=[];
for n=1:n_folds
    % split into training and test data
    [train, test]=foldsplit(orig_data, n_folds, n);
    % model inference
    [model]=trainfunc(train, fiddleparams);
    % store this folds true targets and predictions
    [res]=truetarg(test);
    [pred]=predtarg(test, model);
    allres=[allres; res];
    allpred=[allpred; pred];
end
```

Leave one out has as many folds as samples. An alternative by resampling with replacement is called Bootstrapping.

Means for complexity control

In addition to changing model order, a technique called regularisation allows to control the complexity during learning. Regularisation penalises non-smooth functions by adding a suitable penalty term to the objective function:

- Multi layer perceptrons (MLPs) have sigmoidal shaped nonlinearities which, for input values close to zero, are approximately linear. By forcing small coefficients, we can thus avoid excessive nonlinear behaviour: $\gamma\boldsymbol{\theta}^T\boldsymbol{\theta}$ is thus added as penalty term to the objective function. Note that considerations of scale consistency require using one γ per layer.
- If the behaviour of the resulting predictor is not directly connected to the model parameters, the penalty is formulated via the norm (size) of the second derivatives of the resulting function. (Bishop 1995), suggests for example using

$$\gamma \sum_{n,i} \left(\frac{\partial^2 f(x_n, \theta)}{\partial x_i^2} \right)^2$$

  as penalty term for RBF networks.
- A rather ad-hoc approach to regularisation can be obtained by adjusting the width of the kernel function.

Adjusting kernel widths

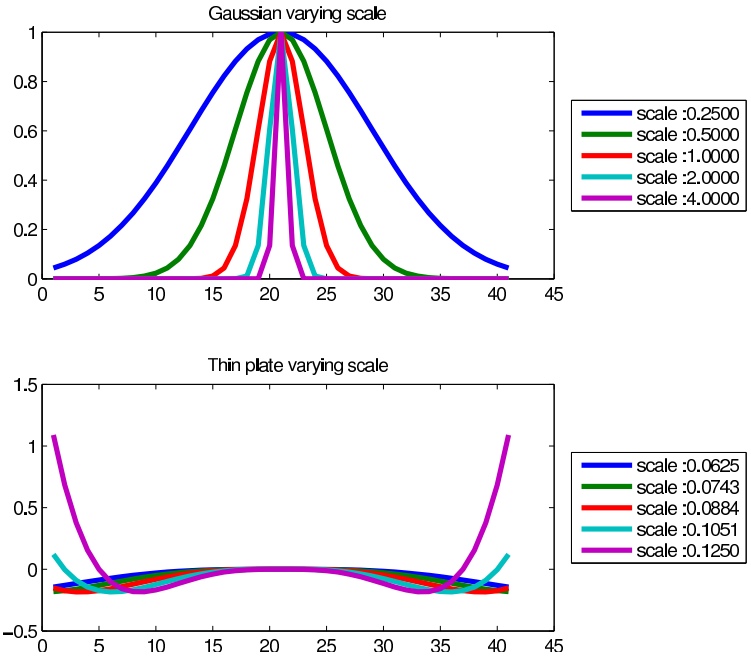
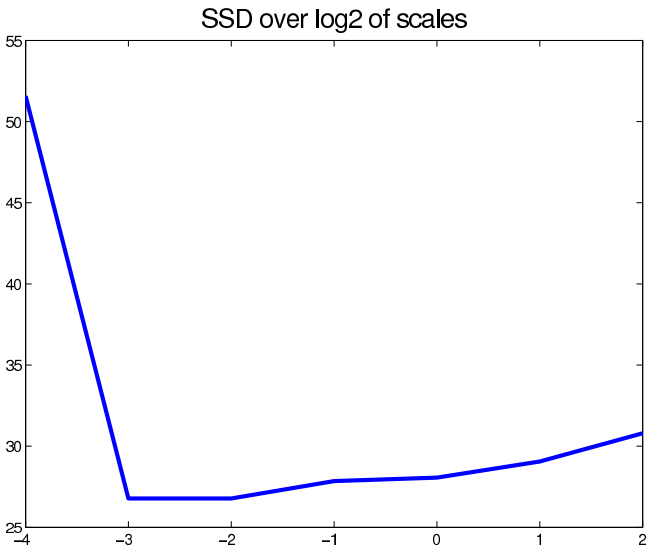


Illustration of the effect of adjusting the scales (variance) of the kernel functions.

Validating regularisation penalties

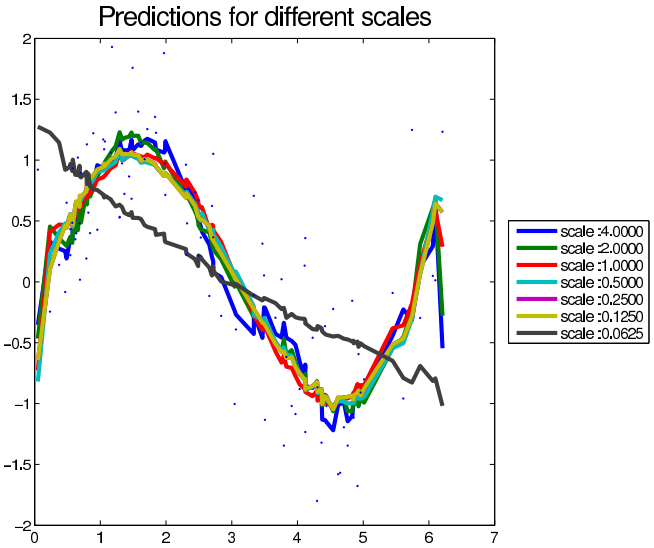
```
for n=1:length(myscaleg)
  csg=myscaleg(n); % get n-th length scale
  % init store for true targets and predictions.
  alltrtrg=[]; allpred=[]; allscales(n)=csg;
  nfolds=10;
  for fldno=1:nfolds
    % split training data into inputs and targets
    [tr, tst]=nfldsplt(data, nfolds, fldno);
    xtr=tr(:,1); ttr=tr(:,2);
    alltrtrg=[alltrtrg;tst(:,2)]; xtst=tst(:,1);
    [net, tranpred, ssd]=rbf_optim(xtr, ttr, nkrnl, nrnets, nettyp,...
            nrep, csg);
    % prediction on xtst: unbiased performance measures
    [ytst]=lin2nonlin_pred(net, xtst); allpred=[allpred; ytst];
  end
  % calculate ssd on the independent test data obtained by cv!
  regssd(n)=sum((alltrtrg-allpred).^2);
end
```

Sum of squares over length scale



The most important aspect is using independent validation data. Predictions are thus unbiased and we obtain meaningful complexity assessments.

Corresponding Predictions



Although regularisation helps, further improvements are possible. Further options: constraining the norm of model coefficients (θ) or model averaging (e.g. bagging).

Summary Part 4

ML & PRN provide different approaches (complexity penalties, “direct” regularisation and model averaging) which help

- getting model complexities right and
- obtaining solutions which generalise well on independent test data.

Technically these methods allow

- selecting most promising model classes and
- tuning of “fiddle parameters” like the amount of “force” (γ) to obtain “small” model coefficients.

Bibliography

Akaike H., A new look at the statistical identification model, in *IEEE Trans. Auto. Control* 19, 716-723, 1974.

Bishop C. M., Neural Networks for Pattern Recognition, Clarendon Press Oxford, UK, 1995.

Hartman E. J., Keeler J. D. and Kowalski J. M., Layered neural networks with Gaussian hidden units as universal approximations, in *Neural Computation* 2, 210-215, 1990.

Rissanen J., Modeling by the shortest data description, in *Automatica* 14, 465-471, 1978.

Schwarz G., Estimating the dimension of a model, in *The annals of Statistics* 6, 461-464, 1978.

Machine Learning and Pattern Recognition in Bioinformatics

Peter Sykacek[1]

Vienna Science Chair of Bioinformatics
Department of Biotechnology
BOKU University
peter.sykacek@boku.ac.at

Likelihood and Classification

"Classification" often used synonymously for regression with discrete outcomes. Likelihood of regression model:

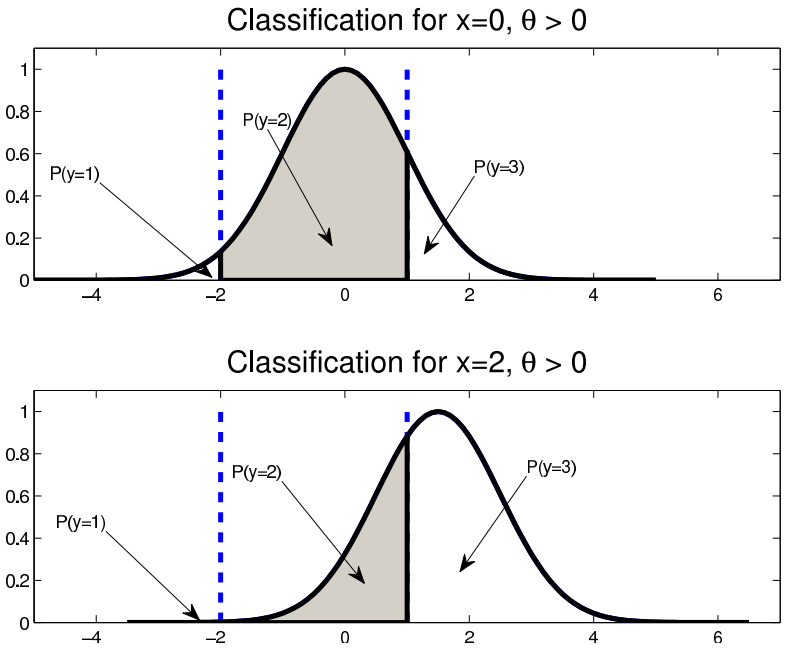
$$P(\mathcal{D}|\mathcal{X};\boldsymbol{\theta}) = \prod_n P(y_n|\boldsymbol{x}_n,\boldsymbol{\theta})$$

To enforce $\sum_{y_n} P(y_n|\boldsymbol{x}_n,\boldsymbol{\theta})$ is 1, we apply a suitable output transformation, e.g. the cdf of the logistic distribution:

$$P(y_n|\boldsymbol{x}_n^T\boldsymbol{\theta}) = \frac{1}{1+\exp((2y_n-1)\boldsymbol{x}_n^T\boldsymbol{\theta})}$$

Probabilities are certainty measures about which label to predict. The linearly parameterised logistic distribution is optimal for a two class problem where both classes are Gaussian distributed and have identical covariances.

Ordered multi class problems



Two main types of 1-of-c classification problems (c: number of classes). For ordered classes like quality or severity indices:

$\log(\gamma_k(\boldsymbol{x})/(1-\gamma_k(\boldsymbol{x}))) = \beta_k - \boldsymbol{\theta}^T\boldsymbol{x}$ with $\gamma_k(\boldsymbol{x}) = P(y <= k)$.

All classes share regression coefficients but differ in threshold β_k. To identify the model, one β_k is fixed.

The idea is best understood as a logistic distribution with mode $\boldsymbol{\theta}^T\boldsymbol{x}$ and probabilities for class corresponding to the area under the pdf between neighbouring thresholds.

Unordered multi class problems

Are usually dealt with by the output mapping:

$$P(y=k|\boldsymbol{x}) = \frac{\exp((\Phi(\boldsymbol{x})^T\boldsymbol{\theta}_k)}{\sum_k \exp((\Phi(\boldsymbol{x})^T\boldsymbol{\theta}_k)}$$

In ML & PRN this is called softmax transformation. The Φ should remind you that these output transformations can be used in combination with nonlinear mappings of the original inputs (RBF, MLP etc.). The softmax transformation can be regarded as multivariate generalisation of the logistic cdf. Expressing $P(y=k|\boldsymbol{x})$ vs. $P(y=\neg k|\boldsymbol{x})$ results consequently in a logistic cdf.

A note on evaluating softmax: for numerical reasons one should rather express it as $P(y=k|\boldsymbol{x}) = 1/(1+\sum_{i\neq k}\exp(\Phi(\boldsymbol{x})^T(\boldsymbol{\theta}_i-\boldsymbol{\theta}_k))$.

Error functions for classification

Taking the $\log$ of the above likelihood results in:

$L_{\text{class}} = \sum_n \log P(y_n|\boldsymbol{x}_n, \boldsymbol{\theta})$

The negative log likelihood, also called cross entropy error, is in classification the logical counterpart of sum of square differences in regression:

$E_{\text{ce}} = -\sum_n \log P(y_n|\boldsymbol{x}_n, \boldsymbol{\theta})$

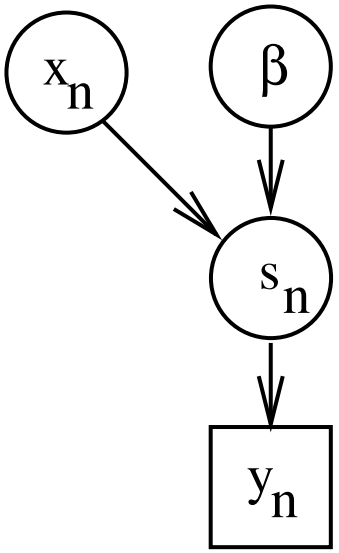
Note however that SSD and a zero one target coding is still guaranteed providing *asymptotically* posterior probabilities for class.

Fitting Classifiers

Nonlinear output mapping, makes fitting classifiers more difficult than fitting linear in the parameter models. Requires generic nonlinear optimisers (see Press *et al.* (2007)):

- gradient descent: $\delta\boldsymbol{\theta} \propto -\nabla_{\boldsymbol{\theta}} E_{\text{ce}}(\boldsymbol{\theta})$.
- quasi Newton: $\delta\boldsymbol{\theta}_n = -\alpha_n \boldsymbol{\Lambda}(\boldsymbol{\theta}_n)\nabla_{\boldsymbol{\theta}} E_{\text{ce}}(\boldsymbol{\theta}_n)$, with $\boldsymbol{\Lambda}$ approx. Hessian $\nabla\nabla^T_{\boldsymbol{\theta}} E_{\text{ce}}(\boldsymbol{\theta})$ at minimum. Close to optimum, $E_{\text{ce}}(\boldsymbol{\theta})$ approx. quadratic $-$ $>$ convergence speed close to Newton method.
- conjugate gradient similar to quasi Newton, however only memory requirement for $\dim(\boldsymbol{\theta})$ vector instead of $[\dim(\boldsymbol{\theta}) \times \dim(\boldsymbol{\theta})]$ matrix.

EM and Classification



$$p(s_n|\boldsymbol{x}_n,\boldsymbol{\theta}) = (2\pi)^{-\frac{1}{2}}\exp(-\tfrac{1}{2}(s_n - \boldsymbol{x}_n^T\boldsymbol{\theta})^2)$$

$$P(y_n = 1|s_n) = \begin{cases} 1, \text{ if } s_n > 0 \\ 0, \text{ if } s_n \leq 0 \end{cases},$$

$$P(y_n = 0|s_n) = 1 - P(y_n = 1|s_n)$$

$$\log(P(y_n|\boldsymbol{x}_n,\boldsymbol{\theta})) \geq \int_{s_n} Q(s_n)\Big(\log(P(y_n|s_n) + \log(p(s_n|\boldsymbol{x}_n,\boldsymbol{\theta})) - \log(Q(s_n))\Big)ds_n$$

$$E_{\text{ce}}(\boldsymbol{\theta}) \leq -\sum_n \Big[\int_{s_n} Q(s_n)\Big(\log(P(y_n|s_n) + \log(p(s_n|\boldsymbol{x}_n,\boldsymbol{\theta})) - \log(Q(s_n))\Big)ds_n\Big]$$

Probit EM

Expectation maximisation (EM) scheme for $E_{\text{ce}}(\boldsymbol{\theta})$.
Minimise iteratively w.r.t all $Q(s_n)$ and the parameters $\boldsymbol{\theta}$.
$Q(s_n)$: truncated Gaussian with mean $\boldsymbol{x}_n^T\boldsymbol{\theta}$, std. 1 and y_n dependant truncation:

$$Q(s_n) \propto \mathcal{N}(s_n; \boldsymbol{x}_n^T\boldsymbol{\theta}, 1)\Pi[\gamma_n, \delta_n]$$

with $\Pi[\gamma_n, \delta_n]$ denoting an improper uniform density.
$\Pi[\gamma_n, \delta_n] = [-\infty, 0]$ if $y_n = 0$ and $\Pi[\gamma_n, \delta_n] = [0, \infty]$ if $y_n = 1$.

Optimal $\hat{\boldsymbol{\theta}}$:

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{s}$$

$\boldsymbol{s}$ denotes all expectations $< s_n >_{Q(s_n)}$ as column vector.

Probit EM on toy data

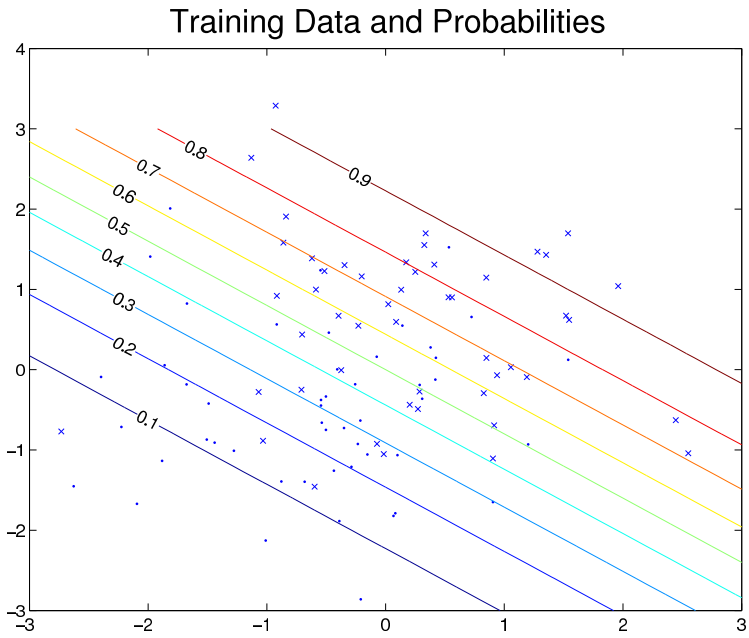
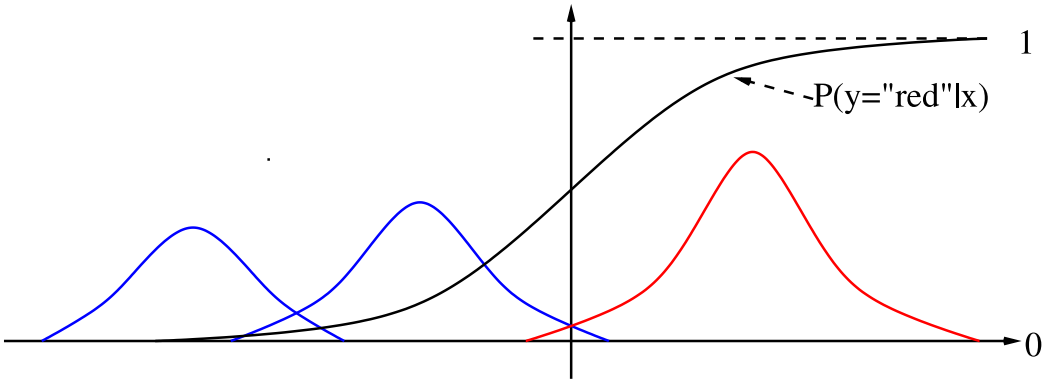


Illustration of data drawn from two Gaussians and the probabilities for class one ('x') obtained by a linear probit model. Nonlinear decision boundaries can be obtained without changing the EM algorithm. Similar to RBF regression, we only have to project inputs onto a nonlinear basis.

Classification and Sampling Paradigm

$$P(y_n|\boldsymbol{x}_n) = \frac{P(y_n)p(\boldsymbol{x}_n|y_n)}{p(\boldsymbol{x}_n)}$$

$->$ Bayes theorem suggests that we can also model class priors $P(y_n)$ and class conditional densities $p(\boldsymbol{x}_n|y_n)$.



Advantage: a useful density model, disadvantage: more complicated

Model Diagnostic Measures

The mean square generalisation error (MSE_{test}) allows assessing regression models.

$$\mathrm{MSE}_{test} = \frac{1}{N}(\boldsymbol{y}_{test} - \boldsymbol{f}(\boldsymbol{X}_{test}; \boldsymbol{\theta}_{train}))^T(\boldsymbol{y}_{test} - \boldsymbol{f}(\boldsymbol{X}_{test}; \boldsymbol{\theta}_{train}))$$

Classification aims at labelling novel samples correctly. This is tested by the generalisation accuracy acc_{test}.

$$\forall n\ \hat{\boldsymbol{y}}_{test}[n] = \mathrm{argmax}_k(P(y = k|\boldsymbol{X}_{test}[n,:]\boldsymbol{\theta}_{train}))$$

$$\mathrm{acc}_{test} = \frac{1}{N}\sum_n \delta(\boldsymbol{y}_{test}[n], \hat{\boldsymbol{y}}_{test}[n])$$

We classify such that the most probable class wins and estimate the fraction of correctly classified test cases.

Testing vs. Validation

Validation: repeated assessments of ML & PRN methods in order to calibrate fiddle parameters of algorithms.

Implication: tuning of some algorithmic aspects towards matching the validation data well.
The final results are thus no longer unbiased. We have to be careful here: too much validation has similar effects as overfitting!
Validation does not provide valid generalisation accuracies. The method will inevitably look too good.

Testing: assessment of one and only one model on independent test data.

The idea is obtaining unbiased generalisation results to

More on Assessing Classifiers

What is the implication of predicting the label which got largest posterior probability?

Consider an individual prediction and that we know the correct posterior $P(y|\boldsymbol{x})$: deciding for label $y = 1$ implies being correct with probability $P(y = 1|\boldsymbol{x})$ and being wrong with probability $1 - P(y = 1|\boldsymbol{x})$.

If we decide for the label $k = \text{argmax}_k(P(y = k|\boldsymbol{x}))$, we will thus minimise the overall number of missclassifications.

What if the missclassifications cost is class dependant?

Missclassification Cost

Classifying correctly induces zero cost. A false negative for class k induces cost α_k.
Predicting $y = t$ results in an expected cost
(deciding that the true y is probably t, this is the cost we expect to suffer):

$$C = \sum_{k \neq t} P(y = k|\boldsymbol{x})\alpha_k$$

Missclassification cost C is thus minimised by classifying $y = \text{argmax}_k(P(y = k|\boldsymbol{x})\alpha_k)$.

This structure is commonly found in life science applications. In cancer screening a false negative is obviously much worse than a false positive.

ROC Curve I

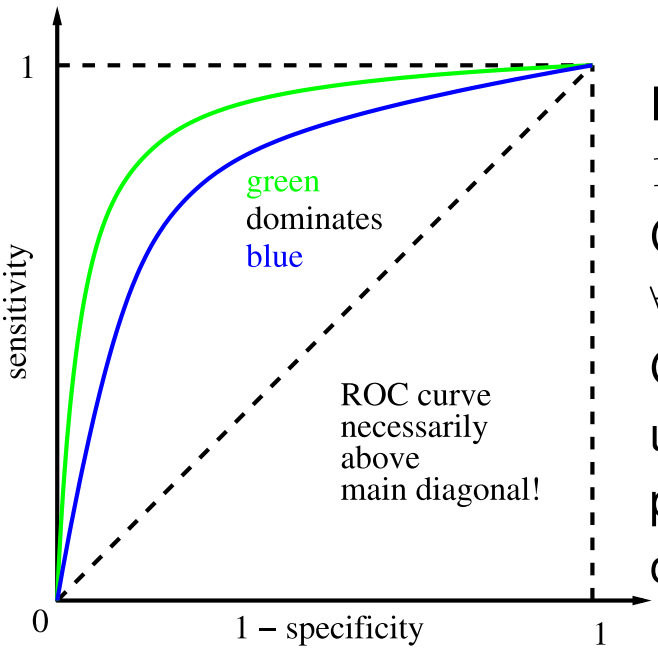
For unknown missclassification cost, the Receiver Operating Characteristic (ROC) curve provides means for assessing binary classifiers.

$$\text{Sensitivity: } \mathrm{s}(\gamma) = \frac{\sum_{n|y_n=1} \delta(P(y_n = 1|\boldsymbol{x}_n) > \gamma)}{N_+}$$

$$\text{Specificity: } \mathrm{p}(\gamma) = \frac{\sum_{n|y_n=0} \delta(P(y_n = 1|\boldsymbol{x}_n) < \gamma)}{N_-}$$

depend on a detection threshold γ; $\delta()$ maps "true" to 1 and "false" to 0; N_+ is the number of positive and N_- the number of negative samples.

ROC Curve II



ROC curve: $s(\gamma)$ over $1 - p(\gamma)$

Classifier a dominates b if $\forall s_a, s_b = s : p_b(s_b^{-1}) < p_a(s_a^{-1})$

Given domination, the area under the ROC curve (AUC) provides a quality measure of the classifier.

AUC and survival probability: $\mathrm{AUC} = P(P(y = 1|x \sim p(x|1)) > P(y = 1|x \sim p(x|0)))$.

Comparing Classifiers

Verification particular choices like the chosen model and fiddle parameters requires performance comparisons.

Default Accuracy:

The most trivial competitor predicts for every sample it's prior probability and thus always the majority class.

Competing Classifiers:

Since it is not at all clear, whether a particular model works well in the case at hand, one should use a set of different approaches. A simple, yet still powerful approach is the so called k nearest neighbour classifier.

Given several performance measures, we should investigate, whether differences are significant.

Mc Nemars Test

The idea of Mc Nemars test is that differences in classifiers manifest themselves in differently classified samples. This allows producing a 2 by 2 contingency table:

	C_{2w}	C_{2c}
C_{1c}	n_a	n_{bthc}
C_{1w}	n_{bthw}	n_b

C_{xc} and C_{xw} refer to samples correctly / wrongly labelled by classifier x. Differences manifest in the number of samples, n_a, which C_1 labels correctly and C_2 labels wrongly and the number of samples, n_b, where the situation is vice versa.

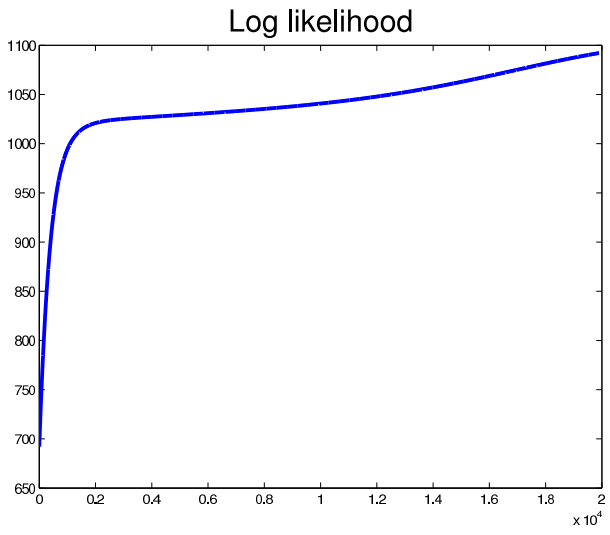
If both classifiers are equal, (n_a, n_b) is a sample of drawing $n_a + n_b$ times from a Binomial distribution with probability 0.5. The null hypothesis of Mc Nemars test is the Binomial $\mathcal{B}n(n_a + n_b, 0.5)$ and we obtain the p-value by calculating the tail probability from (n_a, n_b) onwards.

Continuous latent variable models

Common aspect: $x = [m+]Wt[+\epsilon]$, $W : [d \times k]$ dimensional coefficients matrix, m, ϵ : optional mean and noise term.
PCA (principle component analysis): $t \sim \mathcal{N}(t; 0, \Lambda)$, $\Lambda : [d \times d]$ diagonal cov. matrix, $\epsilon : 0$, m: data mean.
Factor analysis: $t \sim \mathcal{N}(t; 0, \Lambda)$, $\Lambda : [d \times d]$ general cov. matrix, $\epsilon_k \sim N(\epsilon_k; 0, \lambda_k)$ and m: data mean.
ICA (independent component analysis): $t \sim \prod_d p(t_d|\theta_d)$ and $p(t_d|\theta_d)$: univariate density functions, at maximum one Gaussian, m, ϵ : implementation dependent.

These models provide as summary a lower dimensional representation of the data $->$ dimensionality reduction.

ICA Optimisation

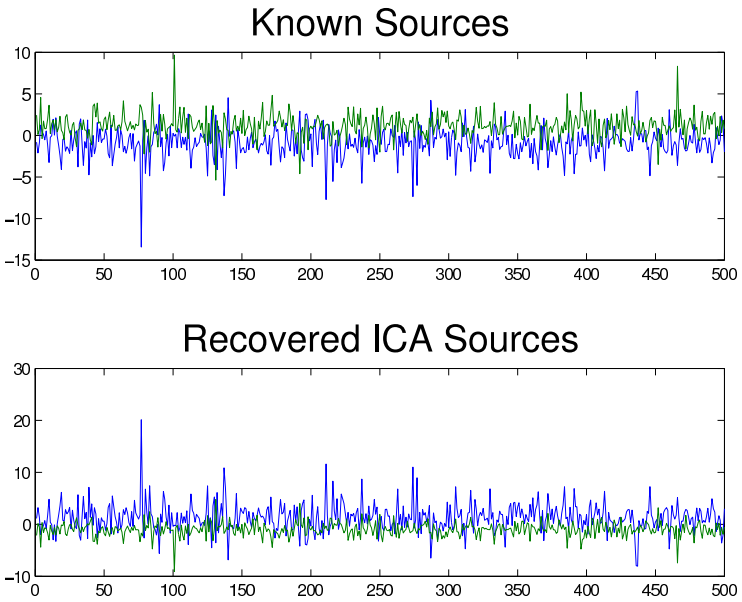


ICA and one observation:

$$p(\boldsymbol{x}_n|W) = \int_{\boldsymbol{t}_n} \left(p(\boldsymbol{x}_n|W, \boldsymbol{t}_n) \prod_k p(t_{n,k}) d\boldsymbol{t}_n \right).$$

For noiseless observations, $p(\boldsymbol{x}_n|W, t_n) = \prod_j \delta(x_{n,j} - W[j,:]t_n)$.
As can be seen, convergence of ICA is rather slow.

Original and Recovered Sources



Note the identifiability problem which looses
1) ordering (sources appear in different order)
2) scale including sign.

Mixture Density Models

Gaussian mixture model: $p(x|t=k) = \mathcal{N}(x; \mu_k, \lambda_k)$, i.e. a (possibly multivariate) Gaussian density function.
K-means clustering: can be regarded as a mixture density model with $P(t=k) = 1/K$ and $p(x|t=k)$ being K uniform densities with domains emerging from the Voronoi tessellation defined by the K cluster centres.
Hidden Markov Model: assumes a one dimensional ordering (e.g. time) among the latent variables t_n. We have thus a more complicated prior: $P(t_n|t_{n-1})$.

These models infer as summary information which mixture component generated the data point x_n $->$ Clustering.

Fitting a Gaussian Mixture - EM

$$\log(p(\mathcal{D}|\{P_k, \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k, \forall\ k\})) = \sum_n \log(\sum_k P_k p(x_n|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k))$$

Problem: $\log$ of sum:

$$\log(\sum_k \log(P_k p(x_n|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) \geq \sum_k Q(K_n = k)$$
$$(\log(P_k) + \log(p(x_n|\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)) - \log(Q(K_n = k)))$$

Introducing additional K_n latent variables into the model allows using this trick $->$ expectation maximisation (EM) algorithm. We have thus to maximise for every sample the left hand side w.r.t. $Q(K_n)$, evaluate the sums (i.e. taking expectations) $->$ E-step and then maximise the resulting expression w.r.t all parameters $\{P_k, \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k, \forall\ k\} ->$ M-step. EM-algorithm iterates E-step and M-step until convergence.

Bibliography

Akaike H., A new look at the statistical identification model, in *IEEE Trans. Auto. Control* 19, 716-723, 1974.

Bishop C. M., Neural Networks for Pattern Recognition, Clarendon Press Oxford, UK, 1995.

Hartman E. J., Keeler J. D. and Kowalski J. M., Layered neural networks with Gaussian hidden units as universal approximations, in *Neural Computation* 2, 210-215, 1990.

Press W. H., Teukolsky S. A., Vetterling W. T. and Flannery B. P., Numerical Recipes, Cambridge University Press, 2007.

Rissanen J., Modeling by the shortest data description, in *Automatica* 14, 465-471, 1978.

Schwarz G., Estimating the dimension of a model, in *The annals of Statistics* 6, 461-464, 1978.