THE DALLAS PROJECT

Report from the NUTEK-supported project AIS-8:
Application of Data Analysis with Learning Systems,
1999–2001

Edited by Anders Holst
Contents

1 Introduction 1
   1.1 The DALLAS project ........................................... 1
   1.2 Goals and objectives ........................................ 1
   1.3 Approach and experiences ................................... 2
   1.4 Participants .................................................. 2
   1.5 This report .................................................... 3

2 Method descriptions 5
   2.1 Regression ..................................................... 5
   2.2 Classification ................................................ 8
   2.3 Multilayer Perceptrons and Error Backpropagation .......... 11
   2.4 A guide to recurrent neural networks and backpropagation . 17
   2.5 Inductive Logic Programming .................................. 27
   2.6 The Bayesian modeling tools .................................. 30
   2.7 Self-organizing feature maps .................................. 36
   2.8 Genetic Algorithms ............................................ 39
   2.9 Information theory ............................................ 41
   2.10 Ensembles, Boosting and Bagging .............................. 45

3 AstraZeneca: Classification of cancer using 2D-electrophoresis 47
   3.1 Introduction .................................................... 47
   3.2 The model used at SICS ........................................ 48
   3.3 The models used at Halmstad University ....................... 51
   3.4 The model used at Skövde University ............................ 54
   3.5 Discussion ...................................................... 55
   3.6 References ...................................................... 55
CONTENTS

7.8 References ........................................................................... 159

8 Telia:
Detection of frauds in a Media-on-Demand system in an IP network 161
  8.1 Problem description ............................................................... 161
  8.2 Data description ................................................................. 162
  8.3 The model used at SICS ....................................................... 164
  8.4 The model used at Halmstad University ............................... 168
  8.5 The model used at DSV ....................................................... 170
  8.6 The model used at Mitthögskolan ....................................... 172
  8.7 Discussion ......................................................................... 174
  8.8 References ....................................................................... 175

9 Ericsson:
Quality of Service in IP-networks 177
  9.1 The task ........................................................................... 177

10 Discussion and conclusions 179
  10.1 Evaluation of the project .................................................... 179
  10.2 Comments on the project form ........................................ 180
  10.3 Conclusions .................................................................... 180
  10.4 Publications made during the project ............................... 181
Chapter 1

Introduction

Björn Levin

1.1 The DALLAS project

The DALLAS ("application of Data AnaLysis with LeArning Systems") project has been designed to bring together groups using learning systems (e.g. artificial neural networks, non-linear multi-variate statistics, inductive logic etc) at five universities and research institutes, with seven companies with data analysis tasks from various industrial sectors in Sweden. An objective of the project has been to spread knowledge and the use of learning systems methods for data analysis in industry. Further objectives have been to test the methods on real world problems in order to find strengths and weaknesses in the methods and to inspire research in the area.

1.2 Goals and objectives

Data analysis (i.e. the search for and the analysis of structures and dependencies in data) is becoming a more and more important concept in almost any industrial sector. With an ever increasing amount of automated measuring devices, sensors, computerized control equipment, networked accounting systems, internet trade etc, huge amounts of data are collected in any kind of industry or business; data that contain very valuable clues on how to improve the businesses in question. Due to the sheer size, manual analysis of these data sets is virtually impossible. However, despite obvious differences in what is measured in e.g. telephone networks, chemical plants, and advertising, the same methods for automated or semi-automated analysis can be applied, and there is therefore a need for similar data analysis tools in a large number of very different industries. A primary goal of the project has therefore been to forward the knowledge about existing new data analysis methods to the industrial partners, to test and show the usefulness of these methods and to establish them as alternatives to existing methods.

Another primary goal has been to supply the academic partners with real world problems and data, industrial feed-back, and inspiration for future research. Such important information, that cannot easily be found in laboratory environments, is of course essential for improving the methods.

The gains of using data analysis tools lie on several levels. Considerable advantages can be gained by simply re-utilizing data collected for various low-level control or administration purposes in a more global analysis. These gains are expected in the form of more even production, lower resource consumption and better competitiveness. Another important gain is better insight into the dependencies and relations in the processes in question, insights that in turn can enable improved production.
1.3 Approach and experiences

A list of tasks (e.g. data sets or processes to analyse) was set up by each participating industrial partner defining one or two items. Each industry was then assigned a main academic contact point, and visits arranged for the whole group of academic partners to each industry in order to gather background knowledge. The problems were at that point defined in more detail and e.g. formats of transferred data agreed. During the planning of the project it was anticipated that this step would require considerable time and resources, but the time actually needed still exceeded what was expected.

After each industry had collected its data, it was sent to their respective main academic contact points for initial testing and further editing. This turned out very well, since usually several iterations between the industry and the academic contact point were needed and the approach kept the required coordination down to two people.

Once edited, almost all tasks were sent to almost all academic partners and attacked with their favorite methods. A disadvantage was of course that the small resources of the project were divided into even smaller bits by this scheme. The advantage was, on the other hand, that a wider range of methods were tested on the tasks.

The preliminary results were then presented at the industries and refinements in the task definitions or in the collection of the data were decided for a second round of attack.

Finally, in some cases, a competition was arranged between the methods of the academic partners. This was very much appreciated by both the academic partners and the industries in question.

Although many of the learning system methods showed some weaknesses that need to be worked out and although some of the tasks turned out to be too difficult to make real progress on during the project, good and valuable results were obtained for a majority of the tasks and a large amount of insight was gained both among the industrial partners and the academic partners. We feel that the main objectives of the project were fulfilled.

1.4 Participants

The following persons from the five academic and seven industrial partners were involved in this project.

Academic partners

**SICS, Swedish Institute of Computer Science, Adaptive Robust Computing laboratory:**
- Björn Levin (project manager)
- Anders Holst
- Daniel Gillblad

**University of Halmstad, school of Information Science, Computer and Electrical Engineering:**
- Thorsteinn Rögnvaldsson
- Mikael Bodén
- Jim Samuelsson

**University of Skövde, dept. of Computer Science:**
- Lars Niklasson
- Henrik Jacobsson
- Fredrik Linäker
- Ulf Johansson
Stockholm University, dept. of Computer and Systems Sciences (DSV):
Lars Asker
Henrik Boström

Mitthögskolan, dept. of Physics and Mathematics:
Mikael Hall
David Martland
Johan Torbiörnson

Industrial partners

AstraZeneca:
Sven Jacobsson
Anders Hagman
Bo Franzén
Fredrik Andersson

EKA Chemicals:
Lars Renberg
Rolf Edvinsson Albers
Håkan Persson

Ericsson Switchlab:
Harald Brandt

Nordisk Media Analys:
Kristina Ericson
Johan Karlsson
Maria Celén
Helena Aava

NovaCast AB:
Rudolf Sillén
Thomas Karlsson

SCA:
Hans Pettersson
Anders Johansson
Joar Lidén
Göran Sundh

Telia:
Anders Rockström
Rolf Hulthén

1.5 This report

This report has two main parts. The first part is contained in Chapter 2, in which the different methods used in the project are described, both the actual learning system methods and various auxiliary techniques that have been useful. The second part is in Chapter 3 to Chapter 9, containing descriptions of the different industrial applications, and the results achieved when applying different methods to them. Finally, Chapter 10 contains a summary and general conclusions.
Chapter 2

Method descriptions

The problems we are considering in the DALLAS project are either classification type problems (e.g. the AstraZeneca application) or regression problems (e.g. the EKA Chemicals application). It is therefore suitable to begin with a brief introduction to these fields and some key concepts.

2.1 Regression

Thorsteinn Rögnvaldsson

2.1.1 The regression problem

We are dealing with a “fixed regressor model”. That is, we have a data set \( X = \{ (x(n), y(n)) \}_{n=1}^{N} \) of observation pairs, where \( x(n) \) is the input and \( y(n) \) is the corresponding output. We assume that the output is generated by the following process

\[
y(n) = g(x(n)) + \epsilon(n)
\]

where \( \epsilon(n) \) is a zero mean noise process with constant variance \( \sigma^2 \). We refer to \( g \) as the “underlying function”.

“Model selection” refers to the search for \( g \) by picking candidate functions \( f(x; w) \) from a model family \( F \), where \( w \) denotes the parameters of the function. We select from the model family \( F \) the function \( f(x; w) \) that has the minimum “distance” \( E(f(x; w); y) \) to our observed data \( y \) (the “distance” is often referred to as the error function).

The modeling process consists of selecting both an appropriate model family \( F \) and the best function in this family.

Examples of model families

Some examples of model families are:

\( F = \{ \text{all linear models} \} \), and \( F = \{ \text{all polynomial models of order } p \} \).

Examples of error functions (distance measures)

The summed square error (SSE):

\[
E = \text{SSE} = \sum_{n=1}^{N} (f(x(n); w) - y(n))^2 = \sum_{n=1}^{N} \epsilon^2(n)
\]
The “maximum likelihood” (ML) measure: (the negative log likelihood because it is more convenient to work with)
\[
E = -\ln L(X|w) = -\ln \left( \prod_{n=1}^{N} p(x(n), y(n)|w) \right) = -\sum_{n=1}^{N} \ln p(x(n), y(n)|w)
\]  
(2.3)

where \( p(x(n), y(n)|w) \) is the likelihood for the observation \( \{x(n), y(n)\} \) given the parameter values \( w \). The most common assumption is the Gaussian likelihood in which case the negative log likelihood is equal to the SSE.

The Bayesian measure:
Maximizing the likelihood is somewhat strange. Why maximize the likelihood for the observations given the model parameters (although we do this by changing the model parameters)? What we really would like to do is to maximize the model parameters, given the observations. Bayes’ theorem tells us how we should do. The probability for the model parameters, given the observations, is expressed as
\[
p(w|X) = \frac{p(X|w)p(w)}{p(X)} = \frac{\mathcal{L}(X|w)p(w)}{p(X)}
\]  
(2.4)

where \( p(w) \) is our “prior” for the model parameters \( w \). Just as in the case of the ML cost, it is more convenient to minimize the negative likelihood, which gives us
\[
E = -\ln p(w|X) = -\ln \mathcal{L}(X|w) - \ln p(w) + \ln p(X)
\]
\[\to -\ln \mathcal{L}(X|w) - \ln p(w)
\]  
(2.5)

since the third term does not depend on the model parameters \( w \).

The Bayesian error measure is even more general than the ML error. The ML error is equal to the special case of a uniform prior in the Bayesian picture.

The Bayesian error is important in the context of overfitting.

2.1.2 Overfitting and the bias vs. variance trade-off

The training data is, unfortunately, only a sample of the real world and it is surprisingly easy to overemphasize the importance of the training data, at the cost of worse performance on new test data (i. e. worse generalization performance).

To understand why, we can look at what is commonly referred to as “model bias” and “model variance”. Model bias is a measure of how well we can model the underlying function \( g \) with our model family \( \mathcal{F} \). If the underlying function can be modeled perfectly with a model from our family, i. e. if the underlying model is a member of our family, \( g \in \mathcal{F} \), then we say that our model family has zero model bias. If the underlying function is not a member of the model family, \( g \notin \mathcal{F} \), then we say that our model family is biased. Model variance is a measure of how much our models vary when we train them with different training sets. If the model family \( \mathcal{F} \) is very small then there will be small differences between models trained with different training sets and we say that the model variance is small. On the other hand, if the model family is large then there can be (will be, according to Murphy’s law) large differences between models trained with different training sets and we say that the model variance is large.

Examples:
Suppose that the underlying function \( g \) is linear.
\( \mathcal{F} = \{ \text{all linear models} \} \) has zero model bias and small model variance.
\( \mathcal{F} = \{ \text{all polynomial models of order 3} \} \) also has zero model bias, but a significantly larger model variance.

Suppose that the underlying function \( g \) is cubic.
\( \mathcal{F} = \{ \text{all linear models} \} \) has a significant model bias and small model variance.
\( \mathcal{F} = \{ \text{all polynomial models of order 3} \} \) has zero model bias and a large model variance.
Whenever we are constructing a model, we should remember that the ultimate goal is to minimize the expected generalization error. Generalization error is the sum of model bias (squared) and model variance. Thus, minimizing expected generalization error necessarily means weighting the model bias against the model variance. This may mean that it is a bad idea to choose a model family $\mathcal{F}$ such that it is guaranteed that $g \in \mathcal{F}$, because the accompanying model variance cancels the benefits from having zero bias.

2.1.3 Classical statistical methods for regression

The most well-known methods for regression in statistics are linear regression (LR), principal components regression (PCR), and partial least squares (PLS). All these methods are linear but they do not produce the same result in general.

Linear regression amounts to using a linear model and minimizing the summed square error (SSE). Principal components regression is also a linear model, but the variables are projected onto the principal axes of the data covariance matrix to transform them to new more informative variables (where fewer variables are needed to solve the problem). Partial least squares is also a PCR-like method where the variables are projected onto the principal axes of the data covariance matrix. However, in PLS one does also consider the variance in the output.
2.2 Classification
Thorsteinn Rögnvaldsson

To classify means that an object or event is ordered into one out of several classes, i.e. a mapping from a feature space $X^D$ to a category space $C^K$

$$f : X^D \rightarrow C^K$$

where $X^D \subset R^D$ and $C^K = \{0,1\}^K$.

2.2.1 Statistical decision theory

Classification is a decision, one decides to categorize an observation into a category. The final decision of course depends on the consequences of the decision and not just the probability that an observation $x(n)$ belongs to a given category $c_k$. Medical applications are excellent examples of this.

Statistical decision theory tells us how we should proceed to make an optimal decision, given that we know the costs associated with our decisions and the probabilities for the different categories. The optimal decision strategy, called the Bayes classifier, is the strategy that always chooses the decision that minimizes the expected conditional risk

$$R(\alpha_i|x) = \sum_{k=1}^{K} \lambda(\alpha_i|c_k)p(c_k|x).$$

where $\alpha_i$ is an action, $c_k$ is a category, and $\lambda(\alpha_i|c_k)$ is the cost for taking action $\alpha_i$ if the object belongs to category $c_k$. Thus, making the right decision means having to know the a posteriori probability $p(c_k|x)$ and the conditional costs for different actions. The a posteriori probabilities can of course be estimated from the conditional probabilities by using Bayes’ rule

$$p(c_k|x) = \frac{p(c_k)p(x|c_k)}{p(x)}.$$  \hspace{1cm} (2.8)

It is common to group classifiers into three groups, depending on the philosophy behind their construction:

1. A posteriori classifiers: Model the a posteriori probabilities $p(c_k|x)$.
2. Probability density classifiers: Model the conditional probabilities $p(x|c_k)$ and combine them with Bayes’ rule to get at the a posteriori probabilities.
3. Decision boundary classifiers: Construct only discrimination functions.

2.2.2 Parametric and non-parametric models

When modeling, it is common to make a distinction between parametric and non-parametric models. Parametric models are models where one has made an assumption about the probability density (or a posteriori probability). Non-parametric models are models where no assumption is made, so-called general approximators are used instead.

The distinction is somewhat artificial, since all models have parameters. It is more correct to speak of models with many free parameters (non-parametric), and models with few free parameters (parametric).

The advantage with parametric models is that they are simple and quick to construct. One can afford to try many different setups. The drawback of parametric models is that one may have assumed the “wrong” parametric family, in the sense that the Bayes classifier (the optimal classifier) is not a member of the hypothesis family. This leads to a model bias, meaning that we will never be able to model the Bayes optimal classifier.
The benefit of non-parametric classifiers is that they are general and that we run little risk of having a model bias. The drawback, however, is that they take a lot of effort to construct, there will be little time for experimenting, and they are likely to “overtrain” and have a large model variance. This means that the resulting classifier will depend very much on the set of observations used to construct it, and if we change one or a few observations then the resulting classifier will also change significantly. A classifier with a large model variance is unlikely to be able to generalize well to new observations.

It is therefore a good idea to work in the intermediate area between non-parametric (many free parameters) and parametric (few free parameters) classifiers. In this region it is important to do a trade-off between model bias and model variance, because both contribute to the generalization error. Artificial neural networks are an example of a classification method in the “twilight zone” between parametric and non-parametric classifiers.

### 2.2.3 Classical statistical methods for classification

The most common classifiers in statistics are: The Gaussian classifier (can be made both linear and quadratic), $k$-nearest neighbor classifiers, and linear discriminants.

Gaussian classifiers are examples of parametric probability density classifiers. The idea is to assume that the data is normally distributed (the parametric assumption) and estimate the mean and variance of this distribution.

The classic non-parametric classifier is $k$-nearest neighbors which is extremely appealing for its simplicity and speed. The $k$-nearest neighbors classifier simply classifies a new observation as belonging to the most common category among previous observations that are similar to it.

### 2.2.4 How to estimate the generalization error

The real goal when modeling is to generalize to new data, not just perform well on the training data set that is presented during training.

In general, the error on the training data will be a biased estimate of the generalization error. To be specific, it will tend to be smaller than the generalization error if we select our model such that it minimizes the training error. We can therefore not use the training error as our selection criteria.

One way to estimate the generalization error is to do cross-validation. This means using a test data set, which is a subset of the available data (typically 25-35%) that is removed before any training is done, and which is not used again until all training is done. The performance on this test data will be an unbiased estimate of the generalization error, provided that the data has not been used in any way during the modeling process. If it has been used, e.g. for model validation when selecting hyperparameter values, then it will be a biased estimate.

If there is lots of data available then it may be sufficient to use one test set for estimating the generalization error. However, if data is scarce then it is necessary to use more data-efficient methods. One such method is the $K$-fold cross-validation method.

The central idea in $K$-fold cross-validation is to repeat the cross-validation test $K$ times. That is, divide the available data into $K$ subsets, here denoted by $D_k$, where each subset contains a sample of the data that reflects the data distribution (i.e., you must make sure that one subset does not contain e.g. only one category in a classification task). The procedure then goes like this:

1. Repeat $K$ times, i.e. until all data subsets have been used for testing once.
   1.1 Set aside one of the subsets, $D_k$, for testing, and use the remaining data subsets $D_{i\neq k}$ for training.
   1.2 Train your model using the training data.
1.3 Test your model on the data subset $D_k$. This gives you a test data error $E_{test,k}$.

2. The estimate of the generalization error is the mean of the $K$ individual test errors: $E_{gen.} = \frac{1}{K} \sum_k E_{test,k}$.

One benefit with $K$-fold cross-validation is that you can estimate an error bar for the generalization error by computing the standard deviation of the $E_{test,k}$ values.

Note: The errors $E_{test,k}$ are often approximately log-normally distributed. At least, $\log E_{test,k}$ tends to be more normally distributed than $E_{test,k}$. It is therefore more appropriate to use the mean of the logs as an estimate for the log generalization error. That is

$$\log E_{gen.} = \frac{1}{K} \sum_{k=1}^{K} \log E_{test,k}$$  \hspace{1cm} (2.9)$$

$$\Delta \log E_{gen.} = 1.96 \sqrt{\frac{1}{K-1} \sum_{k=1}^{K} [\log E_{test,k} - \log E_{gen.}]^2}$$  \hspace{1cm} (2.10)$$

where the lower row is a 95% confidence band for the log generalization error.

An error bar from cross-validation includes the model variation due to both different training sets and different initial conditions.
2.3 Multilayer Perceptrons and Error Backpropagation

Thorsteinn Rögnvaldsson

2.3.1 The multilayer perceptron – general

A “multilayer perceptron” (MLP) is a hierarchical structure of several so-called “simple” perceptrons (with smooth transfer functions). For instance, a “one hidden layer” MLP with a logistic output unit looks like

\[ \hat{y}(x) = \frac{1}{1 + \exp[-a(x)]} \]  \hspace{1cm} (2.11)

\[ a(x) = \sum_{j=0}^{M} v_j h_j(x) = v^T h(x) \]  \hspace{1cm} (2.12)

\[ h_j(x) = \sum_{k=0}^{D} \phi(w_{jk}x_k) = \phi(w_j^T x) \]  \hspace{1cm} (2.13)

where the transfer function, or activation function, \( \phi(z) \) typically is a sigmoid of the form

\[ \phi(z) = \tanh(z), \] \hspace{1cm} (2.14)

\[ \phi(z) = \frac{1}{1 + e^{-z}}. \] \hspace{1cm} (2.15)

The former type, the hyperbolic tangent, is the more common one and it makes the training a little easier than if you use a logistic function.

The logistic output unit (2.11) is the correct one to use for a classification problem.

If the idea is to model a function (i.e. nonlinear regression) then it is common to use a linear output unit

\[ \hat{y}(x) = a(x). \] \hspace{1cm} (2.16)

2.3.2 Training an MLP – Backpropagation

The perhaps most straightforward way to design a training algorithm for the MLP is to use the gradient descent algorithm. What we need is for the model output \( \hat{y} \) to be differentiable with respect to all the parameters \( w_{jk} \) and \( v_j \). We have a training data set \( X = \{x(n), y(n)\}_{n=1,...,N} \) with \( N \) observations, and we denote all the weights in the network by \( W = \{w_{j}, v\} \). The batch form of gradient descent then goes as follows:

1. Initialize \( W \) with e.g. small random values.

2. Repeat until convergence (either when the error \( E \) is below some preset value or until the gradient \( \nabla_W E \) is smaller than a preset value), \( t \) is the iteration number

2.1 Compute the update

\[ \Delta W = -\eta \nabla_W E(t) = \eta \sum_{n=1}^{N} e(n, t)\nabla_W \hat{y}(n, t) \]

where \( e(n, t) = (y(n) - \hat{y}(n, t)) \)

2.2 Update the weights \( W(t+1) = W(t) + \Delta W(t) \)

2.3 Compute the error \( E(t+1) \)
As an example, we compute the weight updates for the special case of a multilayer perceptron with one hidden layer, using the transfer function $\phi(z)$ (e.g. tanh($z$)), and one output unit with the transfer function $\theta(z)$ (e.g. logistic or linear). We use half the mean square error
\[
E = \frac{1}{2N} \sum_{n=1}^{N} [y(n) - \hat{y}(n)]^2 = \frac{1}{2N} \sum_{n=1}^{N} e^2(n),
\]
(2.17)
and the following notation
\[
\hat{y}(x) = \theta [a(x)], 
\]
(2.18)
\[
a(x) = v_0 + \sum_{j=1}^{M} v_j h_j(x), 
\]
(2.19)
\[
h_j(x) = \phi [b_j(x)], 
\]
(2.20)
\[
b_j(x) = w_{j0} + \sum_{k=1}^{D} w_{jk} x_k. 
\]
(2.21)
Here, $v_j$ are the weights between the hidden layer and the output layer, and $w_{jk}$ are the weights between the input and the hidden layer.

For weight $v_i$ we get
\[
\frac{\partial E}{\partial v_i} = -\frac{1}{N} \sum_{n=1}^{N} e(n) \frac{\partial \hat{y}(n)}{\partial v_i} 
\]
\[
= -\frac{1}{N} \sum_{n=1}^{N} e(n) \theta' [a(n)] \frac{\partial a(n)}{\partial v_i} 
\]
\[
= -\frac{1}{N} \sum_{n=1}^{N} e(n) \theta' [a(n)] h_i(n) 
\]
(2.22)
\[
\Rightarrow \Delta v_i = -\eta \frac{\partial E}{\partial v_i} = \eta \frac{1}{N} \sum_{n=1}^{N} e(n) \theta' [a(n)] h_i(n) 
\]
(2.23)
with the definition $h_0(n) \equiv 1$. If the output transfer function is linear, i.e. $\theta(z) = z$, then $\theta'(z) = 1$. If the output function is logistic, i.e. $\theta(z) = \frac{1}{1 + \exp(-z)}$, then $\theta'(z) = \theta(z)[1 - \theta(z)]$.

For weight $w_{il}$ we get
\[
\frac{\partial E}{\partial w_{il}} = -\frac{1}{N} \sum_{n=1}^{N} e(n) \frac{\partial \hat{y}(n)}{\partial w_{il}} 
\]
\[
= -\frac{1}{N} \sum_{n=1}^{N} e(n) \theta' [a(n)] \frac{\partial a(n)}{\partial w_{il}} 
\]
\[
= -\frac{1}{N} \sum_{n=1}^{N} e(n) \theta' [a(n)] v_i \frac{\partial h_i(n)}{\partial w_{il}} 
\]
\[
= -\frac{1}{N} \sum_{n=1}^{N} e(n) \theta' [a(n)] v_i \phi' [b_i(n)] \frac{\partial b_i(n)}{\partial w_{il}} 
\]
\[
= -\frac{1}{N} \sum_{n=1}^{N} e(n) \theta' [a(n)] v_i \phi' [b_i(n)] x_l 
\]
(2.24)
\[
\Rightarrow \Delta w_{il} = -\eta \frac{\partial E}{\partial w_{il}} = \eta \frac{1}{N} \sum_{n=1}^{N} e(n) \theta' [a(n)] v_i \phi' [b_i(n)] x_l(n) 
\]
(2.25)
with the definition $x_0(n) \equiv 1$. If the hidden unit transfer function is the hyperbolic tangent function, *i.e.* $\phi(z) = \tanh(z)$, then $\phi'(z) = 1 - \phi^2(z)$.

This gradient descent method for updating the weights has become known as the “backpropagation” training algorithm. The motivation for the name becomes clear if we introduce the notation

$$
\delta(n) = e(n)\theta'[a(n)],
$$

(2.26)

$$
\delta_i(n) = \delta(n)\phi'[b_i(n)],
$$

(2.27)

which enables us to write

$$
\Delta v_i = \eta \frac{1}{N} \sum_{n=1}^{N} \delta(n) h_i(n),
$$

(2.28)

$$
\Delta w_{il} = \eta \frac{1}{N} \sum_{n=1}^{N} \delta_i(n) x_l(n),
$$

(2.29)

which is very similar to the good old LMS algorithm. Expression (2.27) corresponds to a propagation of $\delta(n)$ backwards through the network.

The gradient descent learning algorithm corresponds to backprop in its batch form, where the update is computed using all the available training data. There is also an “on-line” version where the updates are done after each pattern $x(n)$ without averaging over all patterns.

Backpropagation is, in general, a very slow learning algorithm – even with momentum – and there are many better algorithms which we discuss below. However, backpropagation was very important in the beginning of the 1980:ies because it was used to demonstrate that multilayer perceptrons can learn things.

### 2.3.3 RPROP

A very useful gradient based learning algorithm is the “resilient backpropagation” (RPROP) algorithm. It uses individual adaptive learning rates combined with the so-called “Manhattan” update step.

The standard backpropagation updates the weights according to

$$
\Delta w_{il} = -\eta \frac{\partial E}{\partial w_{il}}.
$$

(2.30)

The “Manhattan” update step, on the other hand, uses only the sign of the derivative (the reason for the name should be obvious to anyone who has seen a map of Manhattan), *i.e.*

$$
\Delta w_{il} = -\eta \text{sign} \left[ \frac{\partial E}{\partial w_{il}} \right].
$$

(2.31)

The RPROP algorithm combines this Manhattan step with individual learning rates for each weight, and the algorithm goes as follows

$$
\Delta w_{il}(t) = -\eta_{il}(t) \text{sign} \left[ \frac{\partial E}{\partial w_{il}} \right],
$$

(2.32)

where $w_{il}$ denotes any weight in the network (*e.g.* also hidden to output weights).

The learning rate $\eta_{il}(t)$ is adjusted according to

$$
\eta_{il}(t) = \begin{cases} 
\gamma^+ \eta_{il}(t-1) & \text{if } \partial_i E(t) \cdot \partial_i E(t-1) > 0 \\
\gamma^- \eta_{il}(t-1) & \text{if } \partial_i E(t) \cdot \partial_i E(t-1) < 0
\end{cases}
$$

(2.33)
where $\gamma^+$ and $\gamma^-$ are different growth/shrinking factors ($0 < \gamma^- < 1 < \gamma^+$). Values that have worked well for me are $\gamma^- = 0.5$ and $\gamma^+ = 1.2$, with limits such that $10^{-6} \leq \eta_{ij}(t) \leq 50$. I have used the short notation $\partial_t E(t) \equiv \frac{\partial E(t)}{\partial w_{ij}}$. The RPROP algorithm is a batch algorithm, since the learning rate update is so small that the gradient term is larger than the sum of the higher order terms. In that case we can ignore the higher order terms and write

$$E(\mathbf{W} + \Delta \mathbf{W}) \approx E(\mathbf{W}) - \eta \| \nabla_{\mathbf{W}} E(\mathbf{W}) \|^2 \leq E(\mathbf{W}).$$

Now, we want to change the weights so that the new error $E(\mathbf{W} + \Delta \mathbf{W})$ is smaller than the current error $E(\mathbf{W})$. One way to guarantee this is to set the weight update $\Delta \mathbf{W}$ proportional to the negative gradient, i.e. $\Delta \mathbf{W} = -\eta \nabla_{\mathbf{W}} E(\mathbf{W})$, in which case we have

$$E(\mathbf{W} + \Delta \mathbf{W}) \approx E(\mathbf{W}) - \eta \| \nabla_{\mathbf{W}} E(\mathbf{W}) \|^2 \leq E(\mathbf{W}).$$

However, this of course requires that $\Delta \mathbf{W}$ is so small that we can motivate (2.35).

We can extend this and also consider the second order term in the Taylor expansion. That is

$$E(\mathbf{W} + \Delta \mathbf{W}) = E(\mathbf{W}) + \nabla_{\mathbf{W}} E(\mathbf{W})^T \Delta \mathbf{W} + \frac{1}{2} \Delta \mathbf{W}^T H(\mathbf{W}) \Delta \mathbf{W} + \mathcal{O}(\|\Delta \mathbf{W}\|^3),$$

where

$$H(\mathbf{W}) = \nabla_{\mathbf{W}} \nabla_{\mathbf{W}}^T E(\mathbf{W})$$

is the Hessian matrix with elements $H_{ij}(\mathbf{W}) = \frac{\partial^2 E(\mathbf{W})}{\partial w_i \partial w_j}$. The Hessian is symmetric (all eigenvalues are consequently real and we can diagonalize $H$ with an orthogonal transformation).

If we can ignore the higher order terms in (2.37) then we have

$$E(\mathbf{W} + \Delta \mathbf{W}) \approx E(\mathbf{W}) + \nabla_{\mathbf{W}} E(\mathbf{W})^T \Delta \mathbf{W} + \frac{1}{2} \Delta \mathbf{W}^T H(\mathbf{W}) \Delta \mathbf{W}.$$

We want to change the weights so that the new error $E(\mathbf{W} + \Delta \mathbf{W})$ is smaller than the current error $E(\mathbf{W})$. Furthermore, we want it to be as small as possible. That is, we want to minimize $E(\mathbf{W} + \Delta \mathbf{W})$ by choosing $\Delta \mathbf{W}$ appropriately. The requirement that we end up at an extremum point is

$$\nabla_{\mathbf{W}} E(\mathbf{W} + \Delta \mathbf{W}) = 0$$

$$\Rightarrow \nabla_{\mathbf{W}} E(\mathbf{W}) + H(\mathbf{W}) \Delta \mathbf{W} = 0,$$

which yields the optimum weight update as

$$\Delta \mathbf{W} = H^{-1}(\mathbf{W}) \nabla_{\mathbf{W}} E(\mathbf{W}).$$
To guarantee that this is a minimum point we must also require that the Hessian matrix is positive definite. This means that all the eigenvalues of the Hessian matrix must be positive. If any of the eigenvalues are zero then we have a saddle point and \( H(W) \) is not invertible. If any of the eigenvalues of \( H(W) \) are negative then we have a maximum point for at least one of the weights \( w_j \) and (2.41) will actually move away from the minimum!

The update step (2.41) is usually referred to as a “Newton-step”, and the minimization method that uses this update step is the Newton algorithm.

Some problems with “vanilla” Newton learning (2.41) are:

- The Hessian matrix may not be invertible, i.e. some of the eigenvalues are zero.
- The Hessian matrix may have negative eigenvalues.
- The Hessian matrix is expensive to compute and also expensive to invert. The learning may therefore be slower than a first order method.

The first two problems are handled by regularizing the Hessian, i.e. by replacing \( H(W) \) by \( H(W) + \lambda I \). This effectively filters out all eigenvalues that are smaller than \( \lambda \). The third problem is handled by “Quasi-Newton” methods that iteratively try to estimate the inverse Hessian using expressions of the form \( H^{-1}(W + \Delta W) \approx H^{-1}(W) + \text{correction} \).

The Levenberg-Marquardt algorithm

The Levenberg-Marquardt is a very efficient second order learning algorithm that builds on the assumption that the error \( E \) is a quadratic error (which it usually is), like half the mean square error. In this case we have

\[
H_{ij} = \frac{1}{2} \frac{\partial^2 \text{MSE}}{\partial w_i \partial w_j} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \hat{y}(n)}{\partial w_i} \frac{\partial \hat{y}(n)}{\partial w_j} + \frac{1}{N} \sum_{n=1}^{N} e(n) \frac{\partial^2 \hat{y}(n)}{\partial w_i \partial w_j}. \tag{2.42}
\]

If the residual \( e(n) \) is symmetrically distributed around zero and small then we can assume that the second term in (2.42) is very small compared to the first term. If so, then we can approximate

\[
H_{ij} \approx \frac{1}{N} \sum_{n=1}^{N} \frac{\partial \hat{y}(n)}{\partial w_i} \frac{\partial \hat{y}(n)}{\partial w_j},
\]

\[
H(W) \approx \frac{1}{N} \sum_{n=1}^{N} J(n) J^T(n), \tag{2.43}
\]

where we have used the notation

\[
J(n) = \nabla_W \hat{y}(n) \tag{2.44}
\]

and we will refer to \( J \) as the “Jacobian”. This approximation is not as costly to compute as the exact Hessian, since no second order derivatives are needed.

The fact that the Hessian is approximated by a sum of outer products \( JJ^T \) means that the rank of \( H \) is at most \( N \). That is, there must be at least as many observations as there are weights in the network (which intuitively makes sense).

This approximation of the Hessian is used in a Newton step together with a regularization term, so that the Levenberg-Marquardt update is

\[
\Delta W = \left[ \frac{1}{N} \sum_{n=1}^{N} J(n) J^T(n) + \lambda I \right]^{-1} \nabla_W E(W). \tag{2.45}
\]
The Levenberg-Marquardt update is a very useful learning algorithm, and it represents a combination of gradient descent and a Newton step search. We have that

$$\Delta W \rightarrow \begin{cases} \frac{1}{\lambda} \nabla E(W) & \text{when } \lambda \to \infty \\ \left[ \frac{1}{N} \sum_n J(n)J^T(n) \right]^{-1} \nabla E(W) & \text{when } \lambda \to 0 \end{cases} \quad (2.46)$$

which corresponds to gradient descent, with \( \eta = 1/\lambda \), when \( \lambda \) is large, and to Newton learning when \( \lambda \) is small.

### 2.3.5 Interpretation of the MLP

#### Classification

If the multilayer perceptron will be used for classification then it should have a logistic output (in the two-class case, in multi-class cases we would use a generalization of the logistic function). If we have a single hidden layer MLP, the output is (c.f. equations (2.18) – (2.21))

$$\hat{y}(x) = \left\{ 1 + \exp \left[ v_0 + \sum_j v_j h_j(w_j, x) \right] \right\}^{-1}$$

which is of the general form

$$\hat{y}(x) = \frac{1}{1 + \exp \left[ f(x) \right]} \quad (2.47)$$

where \( f(x) \) is a nonlinear function of \( x \) (actually, it is a function of projections of \( x \) onto directions \( w_j \)).

If we compare this to the classical classification methods, we see that we are dealing with a generalization of the logistic regression, a nonlinear logistic regression model. That is, we are modeling the a posteriori probability \( p(c|x) \), but using a nonlinear decision boundary.

#### Regression

We use a linear output in the regression case. The MLP function, using a single hidden layer, is then (generally speaking)

$$\hat{y}(x) = v_0 + \sum_{j=1}^M v_j h_j(w_j^T x)$$

where \( h_j(w_j^T x) \) are nonlinear functions of the projections \( w_j^T x \). Models of this form are often referred to as projection pursuit regression (PPR) models in statistics, since projections \( w_j^T x \) are used as arguments. (To be exact, PPR refers to a specific method of minimizing the error and choosing projections but there are strong similarities between the MLP and PPR.)
2.4 A guide to recurrent neural networks and backpropagation

Mikael Bodén

This section provides guidance to some of the concepts surrounding recurrent neural networks. Contrary to feedforward networks, recurrent networks can be sensitive, and be adapted to past inputs. Backpropagation learning is described for feedforward networks, adapted to suit our (probabilistic) modeling needs, and extended to cover recurrent networks. The aim of this brief text is to set the scene for applying and understanding recurrent neural networks.

2.4.1 Introduction

It is well known that conventional feedforward neural networks can be used to approximate any spatially finite function given a (potentially very large) set of hidden nodes. That is, for functions which have a fixed input space there is always a way of encoding these functions as neural networks. For a two-layered network, the mapping consists of two steps,

$$y(t) = G(F(x(t))).$$

(2.50)

We can use automatic learning techniques such as backpropagation to find the weights of the network ($G$ and $F$) if sufficient samples from the function is available.

Recurrent neural networks are fundamentally different from feedforward architectures in the sense that they not only operate on an input space but also on an internal state space – a trace of what already has been processed by the network. This is equivalent to an Iterated Function System (IFS; see [Barnsley, 1993] for a general introduction to IFSs; [Kolen, 1994] for a neural network perspective) or a Dynamical System (DS; see e.g. [Devaney, 1989] for a general introduction to dynamical systems; [Tino et al., 1998; Casey, 1996] for neural network perspectives). The state space enables the representation (and learning) of temporally/sequentially extended dependencies over unspecified (and potentially infinite) intervals according to

$$y(t) = G(s(t))$$

(2.51)

$$s(t) = F(s(t-1), x(t)).$$

(2.52)

To limit the scope of this text and simplify mathematical matters we will assume that the network operates in discrete time steps (it is perfectly possible to use continuous time instead). It turns out that if we further assume that weights are at least rational and continuous output functions are used, networks are capable of representing any Turing Machine (again assuming that any number of hidden nodes are available). This is important since we then know that all that can be computed, can be processed\(^1\) equally well with a discrete time recurrent neural network. It has even been suggested that if real weights are used (the neural network is completely analog) we get super-Turing Machine capabilities [Siegelmann, 1999].

2.4.2 Some basic definitions

To simplify notation we will restrict equations to include two-layered networks, i.e. networks with two layers of nodes excluding the input layer (leaving us with one 'hidden' or 'state' layer, and one 'output' layer). Each layer will have its own index variable: $k$ for output nodes, $j$ (and $h$) for hidden, and $i$ for input nodes. In a feed forward network, the input vector, $x$, is propagated through a weight layer, $V$,

$$y_j(t) = f(\text{net}_j(t))$$

(2.53)

\(^1\)I am intentionally avoiding the term 'computed.'
**Figure 2.1**: A feedforward network.

\[
net_j(t) = \sum_i^n x_i(t)v_{ji} + \theta_j
\]  \hspace{1cm} (2.54)

where \( n \) is the number of inputs, \( \theta_j \) is a bias, and \( f \) is an output function (of any differentiable type). A network is shown in Figure 2.1.

In a simple recurrent network, the input vector is similarly propagated through a weight layer, but also combined with the previous state activation through an additional *recurrent* weight layer, \( U \),

\[
y_j(t) = f(net_j(t))
\]  \hspace{1cm} (2.55)

\[
net_j(t) = \sum_i^n x_i(t)v_{ji} + \sum_h^m y_h(t-1)u_{jh} + \theta_j
\]  \hspace{1cm} (2.56)

where \( m \) is the number of 'state' nodes.

The output of the network is in both cases determined by the state and a set of output weights, \( W \),

\[
y_k(t) = g(net_k(t))
\]  \hspace{1cm} (2.57)

\[
net_k(t) = \sum_j^m y_j(t)w_{kj} + \theta_k
\]  \hspace{1cm} (2.58)

where \( g \) is an output function (possibly the same as \( f \)).
2.4 A guide to recurrent neural networks and backpropagation

2.4.3 The principle of backpropagation

Any network structure can be trained with backpropagation when desired output patterns exist and each function that has been used to calculate the actual output patterns is differentiable. As with conventional gradient descent (or ascent), backpropagation works by, for each modifiable weight, calculating the gradient of a cost (or error) function with respect to the weight and then adjusting it accordingly.

The most frequently used cost function is the summed squared error (SSE). Each pattern or presentation (from the training set), \( p \), adds to the cost, over all output units, \( k \).

\[
C = \frac{1}{2} \sum_p \sum_k (d_{pk} - y_{pk})^2
\]  

(2.59)

where \( d \) is the desired output, \( n \) is the total number of available training samples and \( m \) is the total number of output nodes.

According to gradient descent, each weight change in the network should be proportional to the negative gradient of the cost with respect to the specific weight we are interested in modifying.

\[
\Delta w = -\eta \frac{\partial C}{\partial w}
\]  

(2.60)

where \( \eta \) is a learning rate.

The weight change is best understood (using the chain rule) by distinguishing between an error component, \( \delta = -\partial C / \partial \text{net} \), and \( \partial \text{net} / \partial w \). Thus, the error for output nodes is

\[
\delta_{pk} = -\frac{\partial C}{\partial y_{pk}} \frac{\partial y_{pk}}{\partial \text{net}_{pk}} = (d_{pk} - y_{pk})g'(y_{pk})
\]  

(2.61)
and for hidden nodes
\[
\delta_{pj} = -\left( \sum_{k} \frac{\partial C}{\partial y_{pk}} \frac{\partial y_{pk}}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial \text{net}_{pj}} \right) \frac{\partial \text{net}_{pj}}{\partial y_{pj}} = \sum_{k} \delta_{pk} w_{kj} f'(y_{pj}).
\] (2.62)

For a first-order polynomial, \( \partial \text{net}/\partial w \) equals the input activation. The weight change is then simply
\[
\Delta w_{kj} = \eta \sum_{p} \delta_{pk} y_{pj}
\] (2.63)
for output weights, and
\[
\Delta v_{ji} = \eta \sum_{p} \delta_{pj} x_{pi}
\] (2.64)
for input weights. Adding a time subscript, the recurrent weights can be modified according to
\[
\Delta u_{jh} = \eta \sum_{p} \delta_{pj}(t) y_{ph}(t - 1).
\] (2.65)

A common choice of output function is the logistic function
\[
g(\text{net}) = \frac{1}{1 + e^{-\text{net}}}.
\] (2.66)

The derivative of the logistic function can be written as
\[
g'(y) = y(1 - y).
\] (2.67)

For obvious reasons most cost functions are 0 when each target equals the actual output of the network. There are, however, more appropriate cost functions than SSE for guiding weight changes during training [Rumelhart et al., 1995]. The common assumptions of the ones listed below are that the relationship between the actual and desired output is probabilistic (the network is still deterministic) and has a known distribution of error. This, in turn, puts the interpretation of the output activation of the network on a sound theoretical footing.

If the output of the network is the mean of a Gaussian distribution (given by the training set) we can instead minimize
\[
C = -\sum_{p} \sum_{k} \frac{(y_{pk} - d_{pk})^2}{2\sigma^2}
\] (2.68)
where \( \sigma \) is assumed to be fixed. This cost function is indeed very similar to SSE.

With a Gaussian distribution (outputs are not explicitly bounded), a natural choice of output function of the output nodes is
\[
g(\text{net}) = \text{net}.
\] (2.69)
The weight change then simply becomes

\[ \Delta w_{kj} = \eta \sum_{p}^{n} (d_{pk} - y_{pk}) y_{pj}. \]  

(2.70)

If a binomial distribution is assumed (each output value is a probability that the desired output is 1 or 0, e.g. feature detection), an appropriate cost function is the so-called cross entropy,

\[ C = \sum_{p}^{n} \sum_{k}^{m} d_{pk} \ln y_{pk} + (1 - d_{pk}) \ln(1 - y_{pk}). \]  

(2.71)

If outputs are distributed over the range 0 to 1 (as here), the logistic output function is useful (see Equation 2.66). Again the output weight change is

\[ \Delta w_{kj} = \eta \sum_{p}^{n} (d_{pk} - y_{pk}) y_{pj}. \]  

(2.72)

If the problem is that of “1-of-n” classification, a multinomial distribution is appropriate. A suitable cost function is

\[ C = \sum_{p}^{n} \sum_{k}^{m} d_{pk} \ln \frac{e^{net_k}}{\sum_{q} e^{net_q}} \]  

(2.73)

where \( q \) is yet another index of all output nodes. If the right output function is selected, the so-called softmax function,

\[ g(net_k) = \frac{e^{net_k}}{\sum_{q} e^{net_q}}, \]  

(2.74)

the now familiar update rule follows automatically,

\[ \Delta w_{kj} = \eta \sum_{p}^{n} (d_{pk} - y_{pk}) y_{pj}. \]  

(2.75)

As shown in [Rumelhart et al., 1995] this result occurs whenever we choose a probability function from the exponential family of probability distributions.

2.4.4 Tapped delay line memory

The perhaps easiest way to incorporate temporal or sequential information into a training situation is to make the temporal domain spatial and use a feedforward architecture. Information available back in time is inserted by widening the input space according to a fixed and pre-determined “window” size, \( X = x(t), x(t - 1), x(t - 2), \ldots, x(t - \omega) \) (see Figure 2.3). This is often called a tapped delay line since inputs are put in a delayed buffer and discretely shifted as time passes.

It is also possible to manually extend this approach by selecting certain intervals “back in time” over which one uses an average or other pre-processed features as inputs which may reflect the signal decay.
The classical example of this approach is the NETtalk system [Sejnowski and Rosenberg, 1987] which learns from example to pronounce English words displayed in text at the input. The network accepts seven letters at a time of which only the middle one is pronounced.

Disadvantages include that the user has to select the maximum number of time steps which is useful to the network. Moreover, the use of independent weights for processing the same components but in different time steps, harms generalization. In addition, the large number of weights requires a larger set of examples to avoid over-specialization.

### 2.4.5 Simple recurrent network

A strict feedforward architecture does not maintain a short-term memory. Any memory effects are due to the way past inputs are re-presented to the network (as for the tapped delay line).

A simple recurrent network (SRN; [Elman, 1990]) has activation feedback which embodies short-term memory. A state layer is updated not only with the external input of the network but also with activation from the previous forward propagation. The feedback is modified by a set of weights as to enable automatic adaptation through learning (e.g. backpropagation).

**Learning in SRNs: Backpropagation through time**

In the original experiments presented by Jeff Elman [Elman, 1990] so-called truncated backpropagation was used. This basically means that $y_j(t-1)$ was simply regarded as an additional input. Any error at the state layer, $\delta_j(t)$, was used to modify weights from this additional input slot (see Figure 2.4).

Errors can be backpropagated even further. This is called backpropagation through time (BPTT; [Rumelhart et al., 1986]) and is a simple extension of what we have seen so far. The basic principle of BPTT is that of “unfolding.” All recurrent weights can be duplicated spatially for an arbitrary number...
of time steps, here referred to as $\tau$. Consequently, each node which sends activation (either directly or indirectly) along a recurrent connection has (at least) $\tau$ number of copies as well (see Figure 2.5).

In accordance with Equation 2.62, errors are thus backpropagated according to

$$\delta_{pj}(t - 1) = \sum_{h} \delta_{ph}(t)u_{hj}f'(y_{pj}(t - 1))$$ (2.76)

where $h$ is the index for the activation receiving node and $j$ for the sending node (one time step back). This allows us to calculate the error as assessed at time $t$, for node outputs (at the state or input layer) calculated on the basis of an arbitrary number of previous presentations.

It is important to note, however, that after error deltas have been calculated, weights are folded back adding up to one big change for each weight. Obviously there is a greater memory requirement (both past errors and activations need to be stored away), the larger $\tau$ we choose.

In practice, a large $\tau$ is quite useless due to a “vanishing gradient effect” (see e.g. [Bengio et al., 1994]). For each layer the error is backpropagated through the error gets smaller and smaller until it diminishes completely. Some have also pointed out that the instability caused by possibly ambiguous deltas (e.g. [Pollack, 1991]) may disrupt convergence. An opposing result has been put forward for certain learning tasks [Bodén et al., 1999].

### 2.4.6 Discussion

There are many variations of the architectures and learning rules that have been discussed (e.g. so-called Jordan networks [Jordan, 1986], and fully recurrent networks, Real-time recurrent learning [Williams and Zipser, 1989] etc). Recurrent networks share, however, the property of being able to internally use and create states reflecting temporal (or even structural) dependencies. For simpler tasks (e.g. learning grammars generated by small finite-state machines) the organization of the state space straightforwardly reflects the component parts of the training data (e.g. [Elman, 1990; Cleeremans et al., 1989]).
Figure 2.5: The effect of unfolding a network for BPTT ($\tau = 3$).
state space is, in most cases, real-valued. This means that subtleties beyond the component parts, e.g. statistical regularities may influence the organization of the state space (e.g. [Elman, 1993; Rohde and Plaut, 1999]). For more difficult tasks (e.g. where a longer trace of memory is needed, and context-dependence is apparent) the highly non-linear, continuous space offers novel kinds of dynamics (e.g. [Rodriguez et al., 1999; Bodén and Wiles, 2000]). These are intriguing research topics but beyond the scope of this introductory text. Analyses of learned internal representations and processes/dynamics are crucial for our understanding of what and how these networks process. Methods of analysis include hierarchical cluster analysis (HCA), and eigenvalue and eigenvector characterizations (of which Principal Components Analysis is one).

2.4.7 References


2.5 Inductive Logic Programming
Lars Asker and Henrik Boström

2.5.1 Introduction

Virtual Predict is a system for induction of rules from pre-classified examples. It is based on recent developments within the field of machine learning, in particular inductive logic programming. In this section, we first give a brief description of the field and then point out the main features of Virtual Predict.

2.5.2 Inductive Logic Programming

Inductive Logic Programming (ILP) is a research area in the intersection of machine learning and computational logic whose main goal is the development of theories of and practical algorithms for inductive learning in first-order logic representation formalisms. From inductive machine learning, ILP inherits its goal: to develop tools and techniques for inducing hypotheses from observations (examples) or to synthesize new knowledge from experience. By using computational logic as the representation formalism for hypotheses and observations, ILP can overcome the two main limitations of classical machine learning techniques (such as decision tree learners): the use of a limited knowledge representation formalism (essentially propositional logic), and the difficulties to use substantial background knowledge in the learning process.

The first limitation is important because many domains of expertise can only be expressed in first-order logic, or a variant of first-order logic, and not in propositional logic. The use of domain knowledge is also crucial because one of the well-established findings of artificial intelligence (and machine learning) is that the use of domain knowledge is essential for achieving intelligent behavior. From computational logic, ILP inherits not only its representational formalism but also its theoretical orientation and various well-established techniques. Indeed, in contrast to many other approaches to inductive learning, ILP is also interested in properties of inference rules, in convergence (e.g. soundness and completeness) of algorithms and the computational complexity of procedures. Because of its background, it is no surprise that ILP has a strong application potential in inductive learning. Strong applications exist in drug-design, protein engineering, medicine, mechanical engineering, etc. The importance of these applications is clear when considering that, for example, in the case of drug-design and protein-engineering the results were published in the biological and chemical literature, the results were obtained using a general purpose ILP algorithm and they were transparent to the experts in the domain.

2.5.3 Virtual Predict

Virtual Predict can be viewed as an upgrade of standard decision tree and rule induction systems in that it allows for more expressive hypotheses to be generated and more expressive background knowledge to be incorporated in the induction process. The major design goal has been to achieve this upgrade in a way so that it should still be possible to emulate the standard techniques with lower expressiveness (but also lower computational cost) within the system if desired. As a side effect, this has allowed the incorporation of several recent methods that have been developed for standard machine learning techniques into the more powerful framework of Virtual Predict.

2.5.4 Strategy

There are two main strategies for generating rules from an example file and a theory file: Divide-and-Conquer and Separate-and-Conquer. The former strategy is the same as used by decision-tree learners, allowing most techniques developed within that field to be upgraded to the ILP framework (see following sections). The second strategy is the one adopted by most previous ILP systems. The first strategy works in time linear in the number of examples, while the second works in quadratic time (in the worst case).
The latter may however be more effective than the first in cases where the target is highly disjunctive (see [Boström and Idestam-Almquist, 1999; Boström and Asker, 1999] for further details and a comparison of the two strategies).

2.5.5 Measure

The strategies for generating rules use a measure for choosing among several candidate rules. Methods that use the Divide-and-Conquer strategy can use either the information gain measure [Quinlan, 1986] or adaptive coding measure [Quinlan and Rivest, 1989], while methods that use Separate-and-Conquer can choose between weighted information gain [Quinlan, 1990] or a measure based on the hypergeometric distribution [Boström and Asker, 1999].

2.5.6 Probability estimate

When estimating the probability that an example that is covered by a particular rule belongs to a particular class, two different probability measures may be used by the methods: the La Place estimate and the m estimate (see [Cestnik and Bratko, 1991] for details).

2.5.7 Structure cost

The minimum description length principle according to [Quinlan and Rivest, 1989] may optionally be used both in divide-and-conquer and separate-and-conquer, penalizing extensive search for hypotheses at the cost of information gain according to the chosen measure.

2.5.8 Pruning methods

Some kind of pruning is often necessary in order to avoid the problem of over-fitting the training data. The pruning methods that have been incorporated in Virtual Predict are pre-pruning, post-pruning and incremental reduced error pruning.

Pre-pruning may be used optionally for Divide-and-Conquer and is sometimes desired in order to speed up the induction process and avoid over-fitting. However, it should be used with some care since it may cause the search to stop prematurely.

Post-pruning may be used optionally for Divide-and-Conquer, and it will after having terminated the initial tree structured search select the nodes in the tree that correspond to the highest information gain (i.e. possibly considering structure cost). Optionally a fraction of the training examples will not be used when growing the initial set of rules, but will only be used as a validation set for estimating the information gain.

Incremental reduced error pruning can be used optionally for Separate-and-Conquer. This strategy prunes a rule immediately after a search path has been terminated, resulting in a very efficient induction process (c.f., [Cohen, 1995]). The pruning criterion can be set to one of the following: accuracy on the training set (using the probability estimate), accuracy on a separate validation set (fraction of the training examples to be used for this is set by the user) and information gain (using structure cost).

2.5.9 References


2.6 The Bayesian modeling tools
Anders Holst

2.6.1 Introduction

The Bayesian modeling tools used at SICS consists of a number of statistical models that can be combined with each other, and used to model a variety of different domains in a very generally applicable way. The methods are mainly the same as are used in a Bayesian neural network [Lansner and Ekeberg, 1989; Kononenko, 1989; Holst, 1997], although they are here used separate from the neural network structure. This makes it possible to build more general models.

The original purpose of the models built is to calculate the probability of some attribute given the other attributes. However, the same models can also be used for prediction, clustering, and likelihood calculations.

The techniques that are used and combined are probabilistic graphical models, mixture models, and Markov models. Bayesian statistics is used throughout to estimate the parameters. The resulting model family includes as special cases such standard methods as the naive Bayesian classifier, the quadratic (or Gaussian) classifier, and a kind of linear regression.

2.6.2 Theoretical background

The original purpose of the models is to calculate probabilities. The probability of some attribute or class \( y \) given a vector of attributes \( x \) can be written as:

\[
P(y \mid x) = \frac{P(y)P(x \mid y)}{P(x)} \propto P(y)P(x \mid y)
\]  

Since the denominator \( P(x) \) is the same for all classes, and the probabilities over all classes has to sum to 1, the rightmost expression can be used by normalizing over the classes. The main objective here is therefore to estimate the distribution \( P(x \mid y) \) for each class \( y \) as accurately as possible.

If \( y \) represents a continuous variable instead of a class, and the model is to be used for prediction of that variable, it is more convenient to estimate the joint distribution \( P(x, y) \) instead. The known vector \( x \) can then be inserted, and the marginal distribution \( y \) calculated from this. Depending on what the result should be used for, one can either calculate the mean and variance of this distribution, or make some other more advanced operation on it.

Now, if the distribution of \( x \) is high dimensional or complicated, \( P(x \mid y) \) (or \( P(x, y) \)) can not be estimated directly. The number of degrees of freedom increases exponentially with the number of attributes, and the available data used for training will soon be insufficient. Also if the attributes are continuous valued, some model distribution must be assumed before the estimation, and it should be noted that all distributions are not Gaussian. The idea here is to use the available structure of the domain to break down the distribution in several subdistributions, each of which are easier to estimate.

2.6.3 The naive Bayesian classifier

The first step is to assume independence between the individual attributes in \( x \) (given each class \( y \)). Then the complete distribution can be expressed as a product of the probabilities of the individual attributes:

\[
P(y \mid x) \propto P(y)P(x \mid y) = P(y) \prod_{i=1}^{n} P(x_i \mid y)
\]  

The distribution for each attribute given a specific class, \( P(x_i \mid y) \), is significantly easier to estimate. For example, for \( n \) binary attributes and two classes, there are only \( 4n \) probabilities to be estimated, as opposed to \( 2^{n+1} \) for the complete distribution. This independence assumption is what is used in the
2.6 The Bayesian modeling tools

A naive Bayesian classifier. It actually often gives surprisingly good results, in spite of the simplifying assumption that is usually only approximately fulfilled.

The way to think of this classifier is that each input attribute contributes with its evidence for or against each class, and then all the individual evidence is weighted together to the final result. It can not properly handle cases where the combination of two attributes are more important than the sum of considering them separately. In general, it does not account for the dependence between different attributes. Since in most domains there are some dependences between the attributes, this may be a too big simplification.

2.6.4 Probabilistic graphical models

In the situations where the naive Bayesian classifier is to simple, and the correlations between attributes has to be accounted for, one can instead use a probabilistic graphical model [Chow and Liu, 1968].

The graph describes how the attributes depends on each other: each node in the graph represents one attribute and each edge represents a dependency between two attributes. (I general a “hyper-graph” would be required, which can contain edges that can each connect three or more nodes, thus representing higher order dependencies between the corresponding attributes.) A dependency graph and can be built by searching for strong correlations in the data. Using such a graph it is again possible to write the complete distribution as a product of simpler distributions, i.e. the joint distributions of attributes that are directly dependent on each other according to the graph.

This is the same technique that is used in Bayesian belief networks [Pearl, 1988; Lauritzen and Spiegelhalter, 1988; Heckerman, 1995], but the way it is used here is slightly different. For example, here the output attribute $y$ is kept outside of the graph (or rather, all probabilities are conditional on $y$), whereas in Bayesian belief networks the output attribute is part of the graph. We claim that it is computationally advantageous to keep the class outside the graph, since there is no need to iterate probabilities through the graph in our case. It should also be more robust, since probabilities are calculated in “parallel” rather than in “series”, and thus noise will cancel out rather than accumulate.

The product expressions here are somewhat more complicated than for the naive Bayesian classifier. They are best exemplified with an example.

If there are six attributes with dependencies between them as in figure 2.6, it is possible to write the joint probability as:

$$P(\mathbf{x}) = P(a)P(b)P(c \mid ab)P(d)P(e \mid c)P(f \mid cd)$$

By rewriting the conditional probabilities as fractions, and using that different parts of the tree are independent, this can be rewritten as:

$$P(\mathbf{x}) = P(a)P(b)P(c)P(d)P(e)P(f) \cdot \frac{P(abc)}{P(a)P(b)P(c)} \frac{P(ce)}{P(c)P(e)} \frac{P(cdf)}{P(c)P(d)P(f)}$$
This expression corresponds to the undirected hyper graph in figure 2.7. Every individual attribute, plus every hyper edge in the graph corresponds to one factor in the product.

### 2.6.5 Markov models

A special case of the above dependency graphs occurs for sequential data. If each attribute is sampled at a number of different times, it is reasonable to expect a strong correlation between successive sample points of each attribute. Then it is natural to assume a Markov chain of some order for each attribute. Again this allows that the distribution over a whole sequence can be written as a product (and fraction) of simpler distributions. In the case of a first order Markov model, this will include joint distributions for consecutive samples from the sequence:

\[
P(x) = P(x \mid x_{n-1})P(x_{n-1} \mid x_{n-2}) \cdots P(x_{2} \mid x_{1})P(x_{1}) = \frac{P(x_{n}, x_{n-1})P(x_{n-1}, x_{n-2}) \cdots P(x_{2}, x_{1})}{P(x_{n-1})P(x_{n-2}) \cdots P(x_{2})}
\]  

(2.79)

This can also be generalized to several dimensions, by using Markov grids. It can also be combined with graphs between different attributes.

### 2.6.6 Continuous valued attributes

If all attributes are discrete the above models can be used directly. However, if one or more attributes are continuous valued this must be handled first. This is done in these tools by starting with Gaussian distributions, and combining these in various ways to build up the appropriate distributions.

The simplest case is when the continuous attributes can be considered as normally distributed directly. If all attributes are Gaussians (including the output), the entire space can be modeled as one multivariate Gaussian distribution. When this model is used for prediction, it is equivalent to the normal linear regression. However, if there are many attributes and too few data, there might be severe over-fitting in this case. Therefore an alternative is to separately model the relation between each input attribute and the output, and then combine these models with the naive Bayesian classifier. It is still a linear model, but less prone to over-fitting, due to the reduced number of degrees of freedom inherent in the independence assumption.

If all attributes except the output are continuous, one simple model is the Gaussian classifier, which models each class (i.e., each possible output value) with a Gaussian distribution.

### 2.6.7 Mixture models

If the continuous attributes have more complicated distributions than Gaussians, they can be modeled with a Mixture model [McLachlan and Basford, 1988]. This means that the distribution is approximated by a sum of a number of simpler distributions, e.g., Gaussians (see figure 2.8):

\[
P(x) = \sum_{i=1}^{n} P(v_{i})P(x \mid v_{i}) = \sum_{i=1}^{n} \pi_{i} \cdot f_{i}(x; \theta_{i})
\]  

(2.80)

Here \(f_{i}(x; \theta_{i})\) is the \(i\)th component of the mixture, with parameters \(\theta_{i}\), and the weight of the component \(\pi_{i}\).

Formally any distribution can be approximated sufficiently well by a sum of Gaussians, provided they are many enough. In practice the number of Gaussians used should be small, and the typical situation when this is useful, is when the data consist of a number of clusters, each of which is approximately Gaussian.
To estimate the parameters of a mixture model, an iterative method called *Expectation Maximization* [Dempster et al., 1977] is used. It works by alternately partitioning the data between the components in the mixture and then estimating the parameters of the component from the assigned data.

Mixture models are not limited to Gaussian distribution, but can also combine distributions over discrete attributes, or more complex distributions like graphs and products.

### 2.6.8 Gamma functions

A technique using aspects of both mixture models and Markov models can be used to handle sequential data with information in various time scales. By introducing a number of running average filters, coupled in series, the effect will be that of a number of *Gamma functions*, each sensitive for the information at a certain time scale (see figure 2.9. These can then be combined using a Markov model, and included in the rest of the model.
2.6.9 Bayesian statistics

Throughout these tools, Bayesian statistics [Cox, 1946; Jaynes, 1986] are used to estimate the probability distributions. This means that the influence of the data is moderated by priors, which acts as regularizing factors. The resulting models are less prone to over-training, and especially when there are few data the estimates will be less noise sensitive and more reliable than with classical statistics.

The main difference between Bayesian and classical statistics in this context, is how the parameters of the distributions are estimated. If the parameter to estimate is \( p \), and the data is denoted by \( D \), the objective in classical estimation is to find the \( p \) that maximizes \( P(D \mid p) \), the probability of obtaining the data if that were the true parameter value. This is also called the likelihood of the parameter. In Bayesian statistics the goal is instead to find \( P(p \mid D) \), the probability distribution for \( p \) given the data \( D \). The expectancy of \( p \) can be used as the estimation, and the variance of the distribution is a measure of the precision of this estimation.

The relation between the (classical) likelihood and the (Bayesian) probability distribution of the parameter is again given by Bayes theorem:

\[
P(p \mid D) \propto P(p)P(D \mid p)
\]  

\( P(p) \) is the apriori distribution over \( p \), the distribution of it before any data is observed. Typically this can be a uniform distribution, saying that all probabilities are equally likely before any indication from data. When the apriori distribution is multiplied with the likelihood and normalized, the posterior distribution is obtained, i.e. the distribution over \( p \) given the data.

For example, in the case of a binary variable \( x \), the classical estimate for \( p = P(x) \) is:

\[
\hat{p} = \frac{n_x}{N}
\]

where \( n_x \) is the number of cases with \( x \) out of \( N \) cases in total. The Bayesian estimate is:

\[
\hat{p} = \frac{n_x + \alpha/2}{N + \alpha}
\]

where \( \alpha \) is a factor determining how much effect the prior should have. When the number of data is large, the effect of \( \alpha \) is negligible and the two estimates are almost equal. However, for few data, the Bayesian estimate will avoid the extremes zero and one as probability estimates, because they probably just signal that something is too unusual to have happened yet in the limited data, rather than impossible as the classical estimate would suggest.

Similarly, when a multivariate Gaussian distribution is estimated, the classical estimate tends to over-fit the data and produce very “thin” Gaussians along some axis. The Bayesian prior tends to “span out” the distribution, by using a prior assumption of independence between the attributes.

When a mixture is used, the prior for each component is based on all data, making each component tend to the whole distribution. This prevents individual components from completely disappearing or locking on to single data points, as is otherwise a risk when classical estimates are used in these mixtures.

Bayesian statistics is also useful when two models are to be compared. The likelihood says which model fits the data the best, but a more complex model can usually fit the data better at expense of generalization. The Bayesian prior adds a penalty for the complexity of the model, thus giving an advantage to simple models that explain the data.

2.6.10 Summary

Together these techniques comprise a very powerful toolbox for modeling. Starting with the basic distributions like Bernoulli and Gauss, more complex distributions can be built up by combining these in products, graphs, mixtures, and Markov models. The combination can be done hierarchically: you can for example create mixtures of graphs, or graphs of mixtures. This gives a very high flexibility and the possibility to construct models that can handle complex distributions of very high dimension and with mixed types of variables.
2.6.11 References


2.7 Self-organizing feature maps
Mikael Hall and David Martland

2.7.1 Introduction

Clustering data can be done for a number of reasons, but the act of clustering can be seen as a way to make a smaller description of the data, than that given by the actual collection of samples. Thus it is possible to view clustering as simply being data compression. Perhaps we want to compress data images, or we may want to make some kind of analysis of the sample set. Whatever our motifs are, we need a framework in which the description is given. The crucial difference between say, describing the data by its statistics alone and clustering is that the statistics is only fruitful if we already have a context which give them meaning. We have to have some valid way of knowing when the statistics are good or bad and so on. Merely saying that the variance is one, is not enough.

Suppose we have a data set which in some sense is viewed to be unique so that relations to other sets is complicated or not relevant, like a picture or measurements of a system. Then we must find a framework within the data itself. A histogram is a clustering method, which replace talk about individual samples with properties of bins which are evenly distributed in some portion of the space of vector values representing the samples. Each bin contains some nonnegative number of samples. This way we can talk about groups of samples in terms of bins and compare different groups (bins). The number of bins, location and the width of the bins will consequently determine what is displayed about the data. Clearly it may be beneficial if we in some way can let the things which can be displayed affect the framework by which we view the data, instead of the other way around. In particular we would like the data to determine the width and the location of individual bins, replacing one general design choice regarding the bins, made by us, with parameters unique to individual bins, tuned by the data. Histograms just cut the data into parts and displays whatever can be seen between parts, thus hiding almost everything only describable by differences smaller than the width of the bins and/or if it is out of phase relative the location of the bins.

2.7.2 K-means clustering

K-means clustering differs from histograms in that only the number of bins is determined by the user, assuming the notion of distance is set. The volume of individual bins and their location is determined by the data itself, within the bounds given by the number of bins. Many small bins occur in regions of high sample density. This makes sense in data compression. The algorithm minimize the following cost function:

$$\sum_{j=1}^{K} \sum_{i \in C_j} \| x_i - \mu_j \|^2,$$

where the $N_j$ samples $x_i$ belonging to one of the $K$ bins (clusters) $C_j$ is used to calculate the corresponding center (mean, centroid) $\mu_j$ of the bins by

$$\mu_j = \frac{1}{N_j} \sum_{i \in C_j} x_i.$$

We are thus trying to find $K$ means so that the summed (Euclidean) distance, or dissimilarity, between each sample and its nearest mean is minimized. The search procedure is one of two types. In the batch type, shown above, $K$ clusters of samples are formed randomly, new means are calculated and the samples are appropriately rearranged. This continues recursively until the means have settled. In sequential search, $K$ points are randomly chosen and samples are picked one by one attracting the most similar mean.
The driving force behind the scheme is to avoid counting components of the distances to a degree depending on the number $K$, the data given and the already found sequence of means. We can see this by fixing one sample and its nearest centroid. From the centroid’s point of view this sample could be replaced by any points lying on the surface of a hypersphere centered on the mean and with the same radius as the distance to the given sample, since there would be no change in the associated cost. Given $K = 1$, imagine that the samples are symmetrically distributed around a circle and then let one point on the circle represent two samples (add a duplicate). Then the mean would move towards this point, thereby reducing a double cost. To illustrate this further, but also how the history of the search may affect the quality of the solution found, imagine a data set consisting of the corners of a ordinary dice (a unit hypercube of dim 3). If $K = 1$, the center of the cube would be ideal and found (with a total cost of 6), being the center of a hypersphere (of dim 3) and with the corners lying on the surface of that hypersphere, while if $K = 2$, two disks lying on opposite sides of the hypercube, would be ideal bins (with the four corners on the corresponding side on their surfaces), resulting in a total cost of 4. A non-ideal solution would be to choose one corner as the second mean in the ideal solution of case $K = 1$, obtaining a total cost more than 4. If we during the search procedure get to close to the suboptimal solution (when the bipartition of the corners coincide with that of the solution), we will be drawn into it, as into a black hole, since no corner affects any but the nearest centroid.

One way to view the $K$-mean clustering algorithm would be to regard it as a simple model of a economy with social goals. Each mean would then be an individual, who is trying to exist at a minimal cost, while at the same time forced to share wealth (lack off cost). In an economy without taxes etc, each mean would pick out one sample only and the descriptive task is abandoned. Although economic experiments almost suggest them self, we are not taking this view in order to make political statements. We are trying to make a point regarding the difference between the $K$-mean and the self-organizing feature map. In a real economy people are trying to maximize their wealth (minimize the cost of living) by doing what they are or feel they are most suited to do. But they are also cooperating with other individuals, not only through the tax they pay, but also directly. They form companies and so on. This is modelled by the $K$-mean.

2.7.3 Self-organizing feature maps

As in the previous section, we are viewing the means to be individuals with different skills represented by their prototype vector. During the search which we conduct, we can view the $K$ means as growing up, developing skills, learning to cooperate with others and thus finding a place in life. The data samples are thus rewarding the most fit individual during the search, by lowering its cost. Cooperation, according to the means, could consist in sharing wealth to other means. Now this is not the case. Cooperation means sharing knowledge. The rewarded mean is saying, “follow me, you might be rewarded”. And the receivers of this information, being individuals believing in the Bayesian view of statistics, follows.

The above view is very picturesque, but the real motivation behind self-organizing feature maps is not social behavior of people. Instead, it is meant as a model of how neurons in certain regions of the brain cooperate, where stimuli seem to excite groups of neurons which are close to each other in that region of the brain. So instead of allowing the information pathways to change during the course of life (according to the data), they are most often fixed or changed only in a fixed way. The self-organizing feature map can in addition be viewed as introducing the concept of identity, compared to the $K$-mean. Passing information to others, but not all, assumes an address system. This address system consists of coordinates, which remain unchanged. The information is passed along according to a distance function in the most often two dimensional space of coordinates. The means are placed in this space so that they form a regular grid (hexagonal or other). Before mathematics of the generic kind is presented, we want to point out that there are variants, which do updates the information pathways according to the data.

The mathematics, where we start with the definition of the best-matching unit of the map grid, $m_b$, is:

$$
\| x - m_b \| = \min_i \{ \| x - m_i \| \},
$$
where \( x \) is one randomly chosen sample. The learning rule used at each sample presentation \( t \) updates every map unit by

\[
m_i(t + 1) = m_i(t) + \alpha(t) h_{bi}(t)[x - m_i(t)].
\]

Note that these equations exactly describe the sequential \( K \)-mean if the neighborhood kernel in the map, \( h_{bi} \), satisfy \( h_{bi} = 1 \) if \( m_b = m_i \) and if it is zero otherwise, for all \( t \).

### 2.7.4 Common usages

The Self-organizing feature map is used for different reasons. The main areas of use are:

1. Visualization and clustering of data sets, most often in the early stages of correlation hunting.
2. Modelling, either directly or to direct samples to different submodels.
3. Data preparation. This include replacement of missing values, noise reduction and data compression.

These task are achieved in a robust manner, due to the fact that every map unit is a weighted mean of all samples, through the neighborhood kernel. This also produce some disadvantages like, data range contraction and the creation of interpolating units disturbing correlation cues. But all in all, self-organizing feature maps constitute an efficient multipurpose datamining tool.

### 2.7.5 Further reading


2.8 Genetic Algorithms
Mikael Hall and David Martland

2.8.1 Introduction
As the name suggest, genetic algorithms mimic and tries to harness the search power of evolution. The success of evolution as a search for solutions may or may not be taken as evident. However, biological differentiation has taken place which often do constitute highly optimal solutions relative to the surrounding environments. This specialization often takes place in relatively few generations, compared to the vast number of conceivable solutions. The relation between evolution and genetic algorithms is a two-way relation, where computer experiments has been made to investigate things like fitness versus sexual appeal. In machine learning, genetic algorithms is suitable when the solutions sought are fairly complex. It has been used to find sets of rules, to design artificial neural networks and even to find and construct computer programs.

The ideas behind genetic algorithms is very simple analogies drawn from evolution, such as fitness, sex and disease. These operations are combined with a simple trial and error methodology. Intuitively genetic algorithms is just a way to test and rank solutions. But they often seem to work much more efficiently than what can be expected from this view, and this can be explained by a simple analysis of the way genetic algorithms work.

2.8.2 Fitness
At the heart of genetic algorithms lies the concept of fitness. The first thing to do when designing a genetic algorithm is therefore to decide in what sense individuals or solutions shall be considered to be fit or good. This choice implicitly induce some ordering, total or partial, of the possible solutions. While trivial at times, this is generally a hard, but crucial step. We must direct the search toward nothing but the intended things, without excluding acceptables. This amounts to specifying our needs in terms of some sort of distance measure-we must be able to measure how far different alternatives are from being an acceptable solution, at least we must recognize improvements.

According to rank or some other function of the chosen fitness, candidate solutions are then allowed to participate in making other solutions in the next generation and to survive or die. Also some random mutations and/or some immigration of solutions are allowed to occur. In this way genetic algorithms may manage to both search for new types of solutions and to improve upon the “good” ones. There will be a crucial trade of between these two principal tasks. One must carefully decide how fast the algorithm is allowed to develop a bias versus maintaining some degree of variance.

2.8.3 Variance and bias
Often solutions consists of smaller parts and in genetic algorithms these parts are combined in various ways. The principal way to combine partial solutions is called crossover. This means that the representation of two candidate solutions are each divided into smaller parts and mixed solutions are formed. The degree of bias (or variance) can be seen to be the closely related to the number of partial solutions the algorithm holds on to, the rate at which it sees new ones and the novelty of these solutions. Individual partial solutions have a good chance to contribute to biased search if they are contained in solutions which are ranked high. This effect will depend on the amount of randomness in the algorithm. Randomness is built into genetic algorithms in different ways. The crossover, mutation and survival is made to depend upon random variables and the quality of the fitness function may also introduce some randomness. Through fitness very large partial solutions can survive so that optimal solutions can be found. Partial solutions can also be copied more times if they are small, because the probability of being cut into pieces by crossover or being modified by mutation increase with size; that is, with encoding length. This suggest that the degree to which good solutions are investigated depend upon the design of the crossover operation, thus increasing variance “locally”, while increasing bias “globally”. The crossover operation
works as a Occam’s razor, trying do extract the working parts in fit solutions. The fittest try to sell their solution, while crossover wants to ensure that unnecessary goods aren’t bought.

2.8.4 Important dimensions

Although the different aspects of genetic algorithms are clearly highly interconnected, we can separate them at least in the first generation. Then we can view the sequence of generations as being a Bayesian search, where prior knowledge direct the search and in which knowledge is revised in the light of new evidence. The dimensions in which room for development is made, are given by the number of “atomic” parts individuals are made of, as well as the number of individuals competing at each generation and the number of generations. These three dimensions determines the volume investigated. The number of generations determine the pure random nature of the search, while the size of the population determine the degree to which comparison is exhaustive. The genetic operators add additional power with the number of atomic parts, by exploring the revised knowledge in the light of evidence gained by that generation.

2.8.5 Usability

The main reason genetic algorithms are so interesting is that they are capable of exploring different regions simultaneously and they can move abruptly to new areas. Hence local minima won’t impair the search, in the same way as gradient descent. However a similar problem can arise if a strong bias is developed too soon. This is called crowding out and happens when the “road to success” of the contenders is blocked by the most fittest. One simple way to avoid this is by ranking the solutions and letting this ranking determine the probability of parenthood. This way the degree of superiority do not matter and the less fit are given more room, provided the fitness values has high variability. If not, ranking will give more room to the fittest. The way in which fitness is used in revising the prior knowledge is therefore crucial, as already noted.

The importance and usability of genetic algorithms can be expected to increase with cheap computing power. High dimensional data sets increase the need for automatic datamining tools, as well as for a more diverse set of robust optimization techniques. Genetic algorithms can also easily be parallelized, boosting the applicability further.

2.8.6 Further reading


2.9 Information theory

Anders Holst

2.9.1 Introduction

Here we will briefly present the information theory concepts that have been used in the DALLAS project. The focus is on how to detect strong correlations between attributes in the domain.

2.9.2 Correlation measures

Consider the task of measuring the correlation between two attributes in the data. That is, we have two series of measurements, and want to know how much they are dependent on each other. If there is a complicated process generating several series of measurements, it may be very useful to find out how they depend on each other, to get a better understanding of the process. Another use is to select the most relevant series when some other attribute of the process is to be predicted.

Unfortunately, there is no single best way to measure the correlation that works in all cases. There are however a number of common measures used in different situations:

Correlation between binary variables:

\[ P(x_1, y_1) - P(x_1)P(y_1) \] (2.84)

\[ \frac{P(x_1, y_1)}{P(x_1)P(y_1)} \] (2.85)

\[ \frac{P(x_1, y_1)P(x_2, y_2)}{P(x_1, y_2)P(x_2, y_1)} \] (2.86)

\[ \frac{P(x_1, y_1) - P(x_1)P(y_1)}{\sqrt{P(x_1)P(x_2)P(y_1)P(y_2)}} \] (2.87)

Correlation between continuous variables:

\[ \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \] (2.88)

\[ \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} \] (2.89)

The simplest case is for two binary variables \( x \) and \( y \) (where \( x_1 \) and \( x_2 \) in the equations above are the two possible outcomes of \( x \)). If \( x \) and \( y \) are independent, then \( P(x_1, y_1) = P(x_1)P(y_1) \). This means that a natural test for dependency is to compare \( P(x_1, y_1) \) with \( P(x_1)P(y_1) \), which is what is done in both (2.84) and (2.85). Equation (2.86) is called the “interplay” between \( x \) and \( y \), and is also useful in some contexts. Equation (2.87) is the usual correlation coefficient for binary variables.

When it comes to continuous variables, the most commonly used measure is the linear correlation coefficient, equation (2.88). (Equation (2.88) is of course a special case of this when the variables can only take the values 0 and 1.) If the interesting entity is instead the “slope” of the linear correlation (with \( y \) considered as a function of \( x \)), the regression coefficient (2.89) is used instead.
CHAPTER 2. METHOD DESCRIPTIONS

One problem with all the above measures is that they only consider the linear part of the correlation. This means that important dependencies can be missed, just because their linear components are too small. The classical example is a number of points lying on a circle. The two coordinates are dependent on each other, but nevertheless the correlation coefficient are zero.

Another problem is what to do with ternary variables, or in general variables without any natural ordering between the outcomes. In such cases there are no natural generalization of the correlation coefficient.

This is where information theory offers a solution.

2.9.3 Information

*Information* in this context should not be confused with “knowledge” or “meaning”. It is merely a measure of the smallest number of bits required to transmit messages. One way of putting it as the smallest average number of bits required to inform someone whether an event has happened or not. An event that always happens (with probability 1) contains no information at all: we already knew that it must have happened before we got the message. A very unusual event on the other hand, which causes much surprise, contains a lot of information.

The formal definition of the information of a stochastic variable \( A \) (representing an event that can happen with probability \( P(A) \)) is:

\[
\text{Info}(A) \equiv -\log P(A)
\] (2.90)

This definition is indeed very natural: Information should be additive. If there are two independent events \( A \) and \( B \), and we first get to know \( A \) (giving us \( \text{Info}(A) \) bits of information) and then \( B \) (giving us another \( \text{Info}(B) \) bits), the total information is \( \text{Info}(A) + \text{Info}(B) \) bits. If the events are independent, the probability for both \( A \) and \( B \) to happen is \( P(A, B) = P(A)P(B) \). The only function (up to a constant factor) that in this way can make a product into a sum, is the logarithm:

\[
\text{Info}(A, B) = -\log P(A, B) = -\log P(A) - \log P(B) = \text{Info}(A) + \text{Info}(B)
\]

The minus sign makes the information positive (since the logarithm is negative for numbers less than 1), or zero for probabilities equal to 1. It does not matter which base we use for the logarithm, since this only affects which units of information are used. Base 2 gives the information in bits, but another common choice is to use the natural logarithm, giving the unit “nats”.

Another important concept here is the *entropy* of a variable. It is the average information of that variable, *i.e.* the expectation of the information:

\[
H(X) \equiv -\sum_k P(x_k) \log P(x_k)
\] (2.91)

where \( x_k \) is the \( k \):th possible outcome of the variable. For two independent variables it again holds that \( H(X, Y) = H(X) + H(Y) \). However, what happens if \( X \) and \( Y \) are not independent?

If \( X \) contains some information about \( Y \), this means that \( H(X) + H(Y) \) is greater than \( H(X, Y) \), since a part of the information in \( H(X, Y) \) is contained in both \( H(X) \) and \( H(Y) \), and therefore is counted twice when they are added. The part of the information that \( X \) and \( Y \) has in common, is called the *mutual information* between \( X \) and \( Y \):

\[
I(X, Y) \equiv H(X) + H(Y) - H(X, Y)
\] (2.92)

\[
= -\sum_{i,j} P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)}
\]
2.9.4 Mutual information used as a correlation measure

As opposed to the correlation coefficient, the mutual information is zero if and only if the variables are independent. It is not fooled by a linear component that is zero. The mutual information can also be generalized to any kind of probability distributions, and is not limited to any special cases. This makes it suitable as a general way to measure the correlation between variables.

There are a few practical considerations though: To calculate the entropy of a variable, its distribution must be known. In the binary case, or more generally a discrete variable with a finite number of outcomes, estimating this distribution is straightforward, since it amounts to estimating the probability of each outcome. The only worry is that when the number of outcomes is large, the number of training data may not be sufficient to make a reliable estimate of the probability for each outcome. However, for continuous valued attributes more care has to be taken when estimating the distribution. Some clue as to what form the distribution has is required, i.e. we need a model. If a too simple model is selected important aspects of the real distribution can be missed. For example, if a Gaussian distribution is assumed, estimation is again straightforward, but making this assumption also means that we can not detect any nonlinear parts of the correlation. On the other hand, if a too complex model is selected, two variables will always seem correlated due to overfitting, even if they are independent.

A method which does not require any further knowledge of the exact form of the distribution, is to make a histogram. A grid is placed in the joint space for the two variables that should be checked, and the probability of a data point being in each slot is estimated and fed directly into the equation for mutual information. It is still necessary to select a suitably fine grid. Too many slots will again cause overfitting, making the mutual information tend to the logarithm of the number of data points regardless of where they are. Making a too coarse grid again misses important information. (In specific, using just two slots per variable will again only detect a linear part of the correlation). The appropriate number of slots per variable depends on the amount of available data – enough data points must go into each slot to give a reliable estimate.

Another trick can be used here, which makes the method less sensitive to outliers: The slots need not be of the same size. One can use histogram equalization to adjust the slot sizes for each variable until they each get equally many data points. The grid resulting from both variables will for dependent variables not divide the data points evenly, and it is this deviation from an even distribution that is measured by the mutual information in this case.

Just as a final detail, a relation between mutual information and the normal correlation coefficient can be noted for the case where a linear dependency (or more specifically, Gaussian distributed data) can be assumed. The mutual information in this case can be expressed as:

\[ I(X, Y) = \frac{-\log(1 - r^2)}{2} \]

where \( r \) is the correlation coefficient. While the correlation coefficient lies between \(-1\) and \(1\), the mutual information goes from \(0\) to \(+\infty\), and makes no difference between positive and negative “slopes” of the correlation.

2.9.5 Time series

Finally we will consider how information theory can be used to handle time series. Suppose that we have a sequence of measurements of some variables, e.g. from some process, and again we want to know how dependent they are.

The first impulse is to pair values from the same time of the two variables, and calculate the correlation between these pairs as usual, either with mutual information or some other correlation measure. But there are some problems with this approach. First, successive measurements are not independent of each other. If a variable has a high value at some time, it may be more likely to be high also a few time steps before and after this. This magnifies the impact of coincidences in data: If the first variable just happens to be low at some time when the other variable is high, they are likely to remain in that relation for
several time steps. So even if this was just a coincidence between two random fluctuations, there will be a large numbers of pairs of values with this relation contributing to the correlation measure, making it seem more significant than it is.

The second complication is of course that the variables can depend on each other with some time delay. This is usually handled by align the series with different time delays, and select the delay that gives the highest correlation. The problem is that the dependency can be spread over several steps, in which case the total correlation is not found regardless of which delay is selected. In conjunction with the time dependency within the series, this causes the “peaks” in the correlogram to be very wide, and hard to locate with precision.

One solution here is to use the entropy rate of a time series. Instead of considering the information in each time step separately, it looks at how much new information that is conveyed at any moment in average, i. e. the information given the entire series at earlier times:

\[
H(X_t \mid X_{t-1}, X_{t-2}, \ldots) = -\sum_k P(x_{k,t} \mid x_{k,t-1}, x_{k,t-2}, \ldots) \log P(x_{k,t} \mid x_{k,t-1}, x_{k,t-2}, \ldots)
\] (2.93)

This can be generalized to the mutual information rate:

\[
I_{rate}(X, Y) = H(X_t \mid X_{t-1}, X_{t-2}, \ldots) + H(Y_t \mid Y_{t-1}, Y_{t-2}, \ldots) - H(X_t, Y_t \mid X_{t-1}, Y_{t-1}, X_{t-2}, Y_{t-2}, \ldots)
\] (2.94)

Unfortunately, the entropy rate of a time series can not be calculated directly. The probability distributions that is to be used contains an unlimited number of variables (i. e. the same variable but at an unlimited number of time steps). This is of course impossible to estimate from a limited amount of data. The solution here is to make a Markov assumption, i. e. that the value of a variable depends directly only on the preceding value:

\[
P(X_t \mid X_{t-1}, X_{t-2}, \ldots) = P(X_t \mid X_{t-1})
\] (2.95)

This is a reasonable assumption for many processes, at least as an approximation. This again leaves us with a distribution over only two variables when calculating the entropy rate.

However, the last term in (2.94) is still a problem. As long as it is conditioned on the entire series, time delays doesn’t matter. Even if it takes ten time steps for a change in X to affect Y, ten steps old values of X will be included in the conditioning, and thus taken care of. However, when making a Markov assumption for the joint XY series, we must find the right time delay, or the assumption will not hold. In practice one has to try different time delays \(d\) and select the one with lowest \(H(X_t, Y_{t-d} \mid X_{t-1}, Y_{t-1-d})\), i. e. highest mutual information rate. Dependencies between the two series that are spread out over several time steps will be handled correctly, if the spread is due to the Markov properties (i. e. that the value at one point in time depends on the previous value). However, if the series instead interact at two or more different delays (for example changes that have several effects but at different speeds, or feedback loops such that both variables affect each other) this will not be handled, but only the strongest dependency will be considered. In practice the hope is that this will be sufficient though.

Both the mutual information and the mutual information rate has been used to select variables and find the strongest dependencies between variables in this project.
2.10 Ensembles, Boosting and Bagging
Lars Asker and Henrik Boström

2.10.1 Ensemble learning

The generation of ensembles (i.e. sets of hypotheses) whose predictions are combined has been demonstrated to improve accuracy in many domains. The ensemble learning methods boosting, bagging and randomisation may be used in Virtual Predict in conjunction with the Divide-and-Conquer strategy. All of them require the user to specify the number of iterations (i.e. the number of hypotheses to be generated).

2.10.2 Boosting

Boosting is an ensemble learning method that uses a probability distribution over the training examples, that on each iteration is re-adjusted so that the learning algorithm focuses on those examples that have been incorrectly classified on previous iterations. The method used in Virtual Predict is called AdaBoost (see [Freund and Schapire, 1996] for details). There is an option to tell the system to use so-called stumps only, which allow for faster induction and more compact hypotheses.

2.10.3 Bagging

Bagging is a method that works by creating a number of bootstrap replicates of the training set which are used for generating the hypotheses, and where the entire set of training examples is used as a validation set (see [Breiman, 1996] for details). This strategy has shown to be particularly effective in noisy domains.

2.10.4 Randomization

Randomization works by choosing alternatives according to a probability distribution that is based on the information gain. In this way the hypotheses in the ensemble are slightly varied, and the combined prediction is often more accurate then the prediction made by the single best hypothesis. This strategy may also be used in conjunction with bagging.

Another way of producing ensembles by randomization in Virtual Predict is to generate hypotheses by repeatedly splitting the training examples into a grow set and a validation set (see section on pruning methods).

2.10.5 References


3.1 Introduction

Anders Holst and Thorsteinn Rögnvaldsson

The AstraZeneca task consisted in discriminating between three ovarian tumor types: Benign, borderline, and malignant. The discrimination is based on the spots found on a 2D electrophoresis gel using the PDQuest software. Each spot on a 2D gel corresponds to a polypeptide (proteinlike structure) and the hypothesis was that different types of ovarian cancers were indicated by different proteins (or modified proteins).

The PDQuest software was used, by AstraZeneca, to find a set of potentially interesting spots in the set of gels and the intensity for each spot was estimated for each gel (some spots had zero intensity meaning that they were absent from that particular gel).

Our task was to take this set of intensities for the gels and apply classification methods to see how well the cancer types can be categorized and separated.

The challenge here is the very few available data samples (40) combined with the very high number of inputs (1553). Even if all inputs are random, it would suffice with 39 dimensions to find separating planes for the three classes in the 40 samples. This calls for a method inherently robust to overtraining. One approach is to reduce the dimensionality of the data space with PCA or PLS. This has also been done in another study [Alaiya et al., 2000], and fairly good results were achieved when the three first principal components were used.
3.2 The model used at SICS
Anders Holst

3.2.1 Choice of method

Another approach is to use the naive Bayesian classifier. This is what was done at SICS. In calculating what each of the inputs means by itself for the data samples, there are only four probabilities (for each) to estimate from the 40 samples, which is a dramatic improvement to the situation. Still, among 1553 inputs, some of them will by pure chance seem to be good indicators for the classes on the training set. The hope is that by then combining the evidence from all inputs, there will be as many accidentally bad as accidentally good indicators, canceling each other out, which will thus make the possibly real indicators for the classes stand out from the noise.

The 1553 inputs, representing the proteins at different locations in the gel, is extracted at AstraZeneca by an image manipulating program, which must decide which spots represent real protein traces and which are only noise in the gel image. Since this preprocessing step is in itself quite complicated, there was an idea that instead of feeding these extracted 1553 values into the naive Bayesian classifier, the pixels of the image could be used directly. There would be a lot of noise, and many proteins would cover several pixels, but provided that the same protein location in the different gels are located at approximately the same pixels in the image, the noise would again cancel out and only the pixels corresponding to real and relevant protein locations would contribute to the classification. However, the gels are soft and quite heavily deformed, and therefore the image manipulation must involve a warping of the images to fit the proteins into place. Unfortunately, we had access only to the raw images, and not the warped ones. We tried hard for some time to warp the pictures ourselves, but the gels are too deformed for a simple linear or almost linear warping to suffice. Since AstraZeneca already knows how to warp the images, we figured that this was not the problem we should spend our time to solve, so we gave up this approach and used the pre-extracted values for the 1553 proteins instead.

3.2.2 Preprocessing and experimental setup

For each protein location and gel there is a number in the data that tells how strong the spot is at that location. The maximum value of the strength varies a lot between different proteins. A value of -1 means that no spot was detected. We assumed that the important difference is between the cases of no spot or anything at all, rather than the exact strength of the existing spots. (The alternative assumption, not used at SICS, was that -1 should be interpreted as missing data, and that the relevant information was in the strengths of the detected spots.) Thus we changed -1 in the data to zero, put a threshold at some strength above which the value was changed to one, and below to zero. Since different proteins had very different maximum strengths of their spots, using thresholds at some ratio of the maximum strength of each protein was also tried. Six proteins that had value -1 in all samples were removed.

To be able to compare the results with those from AstraZeneca, the same division in training and testing data was made. This meant that 22 patterns were used for training and 18 for test. Runs were also done with cross-validation, where one of the 40 patterns were removed for testing each time and the other 39 used for training.

The naive Bayesian classifier was trained using Bayesian estimation of the probabilities, and the alpha value of Equation (2.83) set to the reciprocal of the number of training patterns in each case (39 when crossvalidation is used, 22 otherwise). The class with the highest probability is selected as the answer. Ties are counted as failures.

3.2.3 Results

The naive Bayesian classifier used here doesn’t have any further free parameters. The only parameter that is not fixed in this setting, is when to consider a protein to be present or not. Therefore several different thresholds of the spot strengths were tested. In table 3.1 the results of using the same fixed
3.2 The model used at SICS

Table 3.1: Results with the naive Bayesian classifier for different thresholds of strengths.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Train All</th>
<th>Crossvalidation</th>
<th>Trainingset</th>
<th>Testset</th>
</tr>
</thead>
<tbody>
<tr>
<td>100% (22)</td>
<td>50% (9)</td>
<td>100% (22)</td>
<td>50% (9)</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>100% (22)</td>
<td>55.6% (10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>55.6% (10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>55.6% (10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1300</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1400</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1700</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>55.6% (10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>44.4% (8)</td>
<td>100% (22)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2: Results with the naive Bayesian classifier for different relative thresholds of the maximum strength for each protein.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Train All</th>
<th>Crossvalidation</th>
<th>Trainingset</th>
<th>Testset</th>
</tr>
</thead>
<tbody>
<tr>
<td>40%</td>
<td>55.6% (10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55%</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td>72.2% (13)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65%</td>
<td>66.7% (12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70%</td>
<td>66.7% (12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80%</td>
<td>61.1% (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since the tests took some time to perform, all slots in the table were not run. Specifically, since testing on the same set as was used for training (first and third columns) always seemed to give 100% correct, this didn’t seem meaningful to run for the rest of the cases. Although interesting, the second column was the most heavy of them all to run, which is why it is only sparsely filled. The most important results here though are in the fourth column, which shows the generalization of the classifier on the test set. The result varies somewhat with the threshold, but a typical result is around 60% correct. The maximum is achieved with with relative thresholds of 60% of the protein strength, which gave 72%, or 13 out of 18 samples correctly classified.

From the naive Bayesian classifier it is possible to read out how much evidence each input contributed to the conclusion. Table 3.3 lists the indices of the proteins whose existence contributes the most information about each of the three classes of cancer.

3.2.4 Conclusions

The results show that the Naive Bayesian Classifier is very suitable for a problem like this. In spite of the very large number of inputs and few training data, it succeeds to classify the samples at least as good as the other methods tried. This is done using all the available input dimensions, i.e. no dimension reduction via variable selection or principal component analysis is needed.
Table 3.3: Indices for the proteins with the highest expected information gains to each of the three types of cancer.

One advantage with the naive Bayesian classifier is that it can easily be read out how much each input contributes to the classification, and thus to find out which inputs are most important for a certain class. This can be of great value when analyzing the domain.
3.3 The models used at Halmstad University
Thorsteinn Rögnvaldsson

We decided to compare two different types of classifiers: Two simple Gaussian classifiers, which are Bayesian classifiers, and a multilayer perceptron, which is an “a posteriori” classifier.

It is described by Alaiya et al. [2000] how principal component regression and partial least squares regression are used on this data set. We have used the results in [Alaiya et al., 2000] as benchmark for our methods.

3.3.1 Data, preprocessing, and variable selection

The data consists of 40 vectors with spot intensities (i.e., 40 cases). Each vector has 1553 spot intensities. Of the 40 cases, 22 are used for training, and 18 are saved for out-of-sample testing after the model has been built. The true classification of the test cases was unknown throughout the modeling process, but the test case inputs were known (to check e.g. if there was a systematic difference between training and test cases).

We removed all spots that were not present in all the training cases. This was potentially dangerous, since it may well be that the most interesting difference would be the absence or presence of a unique polypeptide. However, we wanted to save time and started with this simple approach. This leaves 88 spots (of which 66 are in fact absent in the test cases), i.e. a 95% dimensionality reduction. These spot intensities were then projected onto the 20 first principal components of the training data, yielding a data matrix of 40 observations and 20 variables for each observation.

The exact preprocessing steps were:

1. Select only those spots which are present in all the training samples. This reduces the number of spots from 1553 down to 88.
2. Compute principal components by taking the eigenvalues and eigenvectors of the covariance matrix for the observations. Keep 20 of these, since about 99.9% of the signal power is retained in the first 20 eigenvalues.

We used a model based variable selection method. The principal component variables were included in the model one by one, in order of decreasing eigenvalues, and the model generalization performance was estimated using cross validation. In each step was the variable selected which resulted in the largest decrease in classification error, as estimated using cross validation. An example of this process is shown for the Gaussian classifiers in Figure 3.1, and the same general approach was used also in the MLP case.

3.3.2 Constructing the classifiers

The Gaussian classifiers were very simple to construct (the whole construction process takes a few hours including the experimenting). In the case of the linear Gaussian classifier, the means and the covariances of the three classes were computed, and then a pooled covariance matrix was computed for the entire data set according to

\[ \Sigma = \frac{N_{\text{benign}} \Sigma_{\text{benign}} + N_{\text{borderline}} \Sigma_{\text{borderline}} + N_{\text{malignant}} \Sigma_{\text{malignant}}}{N_{\text{benign}} + N_{\text{borderline}} + N_{\text{malignant}}}. \]  

(3.1)

Here, \( \Sigma_{nnn} \) denotes the covariance matrices for the individual categories.

In the case of the quadratic Gaussian classifier the individual covariance matrices were retained.

The MLP, however, required considerable more effort. Mostly because it takes longer to train for each model setup. The input variables were selected in a forward manner and the generalization error estimated using \( n \)-fold cross validation (not leave-one-out because this took too much time). The final MLP was
Figure 3.1: Examples of the model based forward variable selection process. The top panel shows the result of including more and more principal components in a linear Gaussian classifier, and the bottom panel shows the same for a quadratic Gaussian classifier. The abscissa is the leave-one-out estimate of the generalization classification error. The very large errors (i.e., making more than 15 errors out of 22 possible) occur because the matrices in the Gaussian classifiers become ill-conditioned. The top panel shows e.g. that the estimated generalization error for a linear Gaussian classifier is zero if we include principal components \{1, 14, 12, 3, 6, 13, 7, 16, 19\} in the model. The bottom panel shows that the lowest generalization error for a quadratic Gaussian classifier is achieved using principal components \{1, 20, 10\}. 
a committee combination of 420 different MLPs (trained using Levenberg-Marquardt minimization and different initial conditions).

### 3.3.3 Results

Several different variable setups yield zero generalization error for the linear Gaussian classifier. We can not choose reliably whether to use 9, 10, 11 or 12 (or more) principal component variables for the linear Gaussian model. We therefore used four different setups (9, 10, 11, and 12 input variables) that all gave zero generalization error and combined them into a committee. We did similarly with the quadratic Gaussian classifier, but used only two setups (3 and 4 input variables). The Gaussian classifier committees were voting committees.

A large set of MLPs (420 of them to be exact) were combined into an averaging committee before the final cancer categories were registered.

The final results are listed in Table 3.4.

<table>
<thead>
<tr>
<th>True category</th>
<th>Benchmark</th>
<th>Lin. Gauss</th>
<th>Quad. Gauss</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>B/C</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td>A</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td>A</td>
<td>B/C</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>A/B</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 3.4: Summary of results on the ovarian cancer hold-out test data. The classes are: A = Benign cancer, B = Borderline, and C = Malignant cancer. The column “Benchmark” shows the results reported in [Alaiya et al., 2000]. Just random guessing with equal a priori probabilities of the cancer categories would give about 12 mistakes on average. The 95% significance limit for a non-random result is about 8 mistakes (i.e. if the model makes more than 8 mistakes then we cannot with 95% confidence discard the hypothesis that it is no better than a random guess). Apart from the Benchmark, it is only the linear Gaussian classifier, the simplest model of all, that is significantly better than a random guess. Both the Gaussian classifier results were done using voting member committees.

### 3.3.4 Conclusion

The simplest model of them all, a linear Gaussian classifier, did the best on this problem. This is not an unusual finding in cases with many input variables and few observations (we had only 22 observations). None of the other models tested here were significantly (95%) better than a random guess of the cancer types.
3.4 The model used at Skövde University  
Henrik Jacobsson

As mentioned above, the data consisted of 22 training examples, 18 test examples and 1553 variables. The variables was either instantiated with a value typically in the range $10^3$ to $10^5$ or with a -1. -1 was interpreted as a missing value. All, except 88 variables had missing values. The data was sorted according to how many missing values it had. 88 variables had no missing values, these were considered as more informative than the other variables.

To reduce the data we chose to use a principal component analysis (PCA). We extracted 21 principal components from the data. This is quite straightforward for the 88 variables with no missing data, but missing data cannot be handled by the PCA. The ANN was then trained on this preprocessed data.

The model we used for training on this preprocessed data was a feedforward neural network. We tried with several different topologies, output representations and training parameters. The target values of the test set was at this time unknown to us and the models were all evaluated on the training set. When the full test set became available, it turned out that none of our models generalized to the test set at all. After several attempts to train the networks on this model we tried to incorporate the variables with missing data in the principal component analysis. There are no given method to do this, so we tried a few ad hoc techniques. No significant effect on the results could be seen however. As we did not achieve any good results, the parameters of the training algorithm are not presented in detail.

It is important to bear in mind that the known classification of the test examples were not given to us. This made it quite difficult to determine whether a specific network performed well in terms of generalization etc. It might also be worth noting that by having only 22 training examples for 1553 variables, any significant results or reliable networks are highly unlikely to be found using this type of model.
3.5 Discussion

Anders Holst

This task is, as mentioned above, a quite hard problem due to the extreme under-determination of the model. The methods that gave reasonable results here were all very simple models with few parameters or very strong regularization: PLS with four latent variables in Alaiya et al. [2000] (11/18 correct), a linear Gaussian classifier at Halmstad University (12/18 correct), and a naive Bayesian classifier at SICS (13/18 correct).

These approaches all gave approximately the same results, and the differences between them can not be considered very significant. It seems feasible to believe that it is hard to get any further on exactly this data set, with any method. Specifically since the classes are not really homogeneous: there seems to be several types of cancer in each of the three classes, some of which occur only once in the 40 samples.

3.6 References

Chapter 4

EKA Chemicals: The hydrogen peroxide production process

4.1 Introduction to the task
Thorsteinn Rögnvaldsson

4.1.1 Background

EKA Chemicals produces chemicals for pulp bleaching processes and paper making. The problem studied in the DALLAS project is the industrial production of hydrogen peroxide at EKA Chemicals’ plant in Bohus, Sweden.

Figure 4.1 shows a simple sketch of the hydrogen peroxide production process. A “working solution”, a mixture of a complex organic chemical called an anthraquinone and solvents, is circulated in the plant. Hydrogen is added to the anthraquinone molecule in a first hydrogenation step. In the following step it is oxidized with air to produce hydrogen peroxide ($H_2O_2$). The anthraquinone compound acts as a carrier for the hydrogen and is needed since mixing oxygen and hydrogen directly will be dangerous (explosive mixture). Moreover, the direct reaction tends to give water rather than hydrogen peroxide. The anthraquinone is recycled after the hydrogen peroxide has been extracted from the solution, and recirculated in the process chain. However, some of the expensive anthraquinone is consumed with time and needs to be refilled.

The hydrogenation is done in two towers, where the working solution (initially just the dissolved anthraquinone) is mixed with hydrogen. The oxidation is done by adding normal air to the mixture after the hydrogenation, and after this the hydrogen peroxide is extracted. After the extraction the working fluid is dried (water is removed) and reentered into the process. Large buffer tanks are used to store the working fluid between the different steps (hydrogenation, oxidation, extraction, and drying).

Several variables are measured at different places, such as valve openings, flows, levels, temperatures, etc. Settings for different controllers are also recorded.

Some interesting questions for this process are:

- Can we extract the dependencies between variables in the data? How do different methods for this variable selection compare?

- Are the nonlinear techniques used in the DALLAS project better than the standard linear method used today, e.g. partial least squares? (This should be quantified.)
• Is it possible to optimize the process using the models? For instance, can we achieve a higher yield without increasing the consumption of expensive raw material (i.e. anthraquinone)?

• Can the models be used for diagnosis/analysis of the industrial process?

• Can the models be used for supporting the operator, i.e. can they give a hint as to how to change some controls to achieve a desired operating state?

Two slightly different data sets were produced in the course of the project. These two data sets were used to ask two different questions but with identical methods for solution.

The first data set spanned a short time (3 months) with a high resolution (data collected every minute). The task here was to model a product quality parameter as a function of the process variables. The second data set spanned a much longer time (19 months) with a lower resolution (the time interval between observations was 10 minutes). In the latter case the task was to try to detect cases when the state of the working fluid was a mix of gas and liquid, instead of a pure liquid, since this indicates sub-optimal operation.

4.1.2 The first problem: Identification

The first data set spanned 3 months (from midnight June 20, 1997, until 20:16 September 26, 1997) with data sampled every minute. The data set contained one output (dependent) variable and 79 input variables (independent)\(^1\) that could potentially provide information about the output variable. The output variable was the density of the fluid extracted from the process. This density reflects, in an indirect way, the amount of hydrogen peroxide that was being produced.

This output variable (the density) was a controlled variable, which was a potential problem with it. The operators try to keep this variable from fluctuating and at an approximately constant value since the process should run steadily with a high yield. It is controlled by the process operators in the sense that they select settings to give a steady and desirable value. This is an indirect, soft, control compared to the more direct control by e.g. a feedback loop.

The first data set had a total of 142,337 observations. In some cases the variables had missing values, or unreasonable values (values far outside their normal range). The time series were spiky (i.e. sudden changes in the variable values, guessed to be caused by e.g. errors in the data collection system).

These data were distributed in the files dallas1fix.dat, dallas2fix.dat, dallas3fix.dat, dallas4fix.dat, and dallas5fix.dat. SICS did later do a filtering of these data which was made available to other groups as well.

4.1.3 The second problem: Identification and diagnosis

The second data set spanned the time from midnight 1997-06-01 to midnight 1998-12-31, i.e. 19 months. The “on-line” sampled variables were sampled every ten minutes, but there were also some additional variables which had been sampled much less frequently (laboratory tests done twice a week). The desired outputs are the openings of valves in two heat exchangers (one for each of the two hydrogenization towers). This opening is feedback controlled, based on the temperature of the working fluid passing through the heat exchanger. The temperature of the working fluid should be maintained at a constant level and the working fluid should be in a purely liquid state. However, a lot of cooling (i.e. the valve opening is large) indicates that the working fluid may be in a mixed state of gas and liquid, and it is interesting to analyze the conditions when this happens.

The output here was not controlled by the process operators, and the signal excitation was significantly larger than in the first data set case.

\(^1\)We use the notation “independent” within parenthesis since the input variables are not independent of each other. On the contrary, many input variables displayed strong dependencies. However, independent and dependent is a common notation used for input and output.
There were also some derived variables\textsuperscript{2} with a physical meaning for the process. These should preferably also be included in the variable selection.

The second data set contained 83,376 observations, 84 variables, including the two outputs, sampled on line (of which eight were empty and thus never used in the modeling, and one was binary), nine variables sampled with a low resolution (manual laboratory experiments, of which one was always zero), and six derived variables (of which one required information about an empty variable in the “online” set, why it could not be included). Thus, 89 potential variables were available for the model building. Most of the variables had some missing values, or unreasonable values (values far outside the variables normal span). There were also several time periods when the process had been stopped, which could be observed through e.g. the hydrogen flow (which was essentially zero during a stop). The variable time series were also very spiky, meaning that the variable’s value could change drastically (e.g. drop from a high value down to zero) and then change back in a few time steps (e.g. back to the high value).

The second data set was distributed as a MATLAB .mat file, called ewing.mat, and as an ascii file called newData.txt.

A third data set was provided after all the models had been built, to serve as a blind test data set for all project groups. This data set was similar to the second data set, except that it contained data from a period after the second data set had been collected. The results on this out-of-sample test data are reported at the end, after the groups’ descriptions of their model(s).

The third data set, the blind test data, was distributed as a MATLAB file, EwingTestBlind.mat, and as an ascii file. The true output values were all set to zero in these files.

\subsection*{4.1.4 A note on notation confusion}

There were only three data sets: The first data set, the second data set, and the hold out test data set for the problem studied using the second data set. However, groups did their own filtering (removing of outliers and removing of uninteresting variables) which lead to different notations by different groups. Table 4.1 A and B contains a key to these notations to help the reader understand and compare the different descriptions.

The outputs for the second data sets are sometimes referred to as C30 and C31 (original HiH notation, first data set), sometimes as TC1105 and TC1125 (these are process codes), sometimes as \#64 and \#66 (HiH original notation), and sometimes as \#57 and \#59 (SICS notation).

\textsuperscript{2}“Derived” means here that the variables have been computed from other variables in the data set. The derived variables thus provide no new information, but the functional dependencies between the derived variables and the output may be simpler (i.e. more linear) than the functional dependencies between the original variables and the output.
Table 4.1: A: Key to translating between the different enumerations. The two data sets used for constructing models were sent out by Halmstad with one notation. SICS did then clean the data further, by removing zero valued variables and variables that were constant, and supply their data also for the other project groups. Some groups use one notation, others use the other notation, and some use both notations. In the first data set, X14 contained no useful information (either missing or constant), and X1 and X17 were identical, why X17 was removed. In the second data set, variables 3-8, 62, 68, and 71 were empty and therefore removed by SICS. The left side of the table has the variables sorted according to the original notation in the first data set. The right hand side has the variables sorted according to the original notation in the second data set.
### Table 4.1: B: Key for translating between the different notations, continued. See Table 4.1A for details.

<table>
<thead>
<tr>
<th>First data set</th>
<th>Second data set</th>
<th>First data set</th>
<th>Second data set</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiH Original</td>
<td>SICS</td>
<td>HiH Original</td>
<td>SICS</td>
</tr>
<tr>
<td>C38</td>
<td>48</td>
<td>X27</td>
<td>31</td>
</tr>
<tr>
<td>X39</td>
<td>49</td>
<td>C28</td>
<td>32</td>
</tr>
<tr>
<td>X40</td>
<td>50</td>
<td>X28</td>
<td>33</td>
</tr>
<tr>
<td>X41</td>
<td>51</td>
<td>C8</td>
<td>11</td>
</tr>
<tr>
<td>X42</td>
<td>52</td>
<td>X8</td>
<td>12</td>
</tr>
<tr>
<td>X43</td>
<td>53</td>
<td>C29</td>
<td>34</td>
</tr>
<tr>
<td>X44</td>
<td>54</td>
<td>X29</td>
<td>35</td>
</tr>
<tr>
<td>X45</td>
<td>55</td>
<td>C37</td>
<td>46</td>
</tr>
<tr>
<td>X46</td>
<td>56</td>
<td>X37</td>
<td>47</td>
</tr>
<tr>
<td>C47</td>
<td>57</td>
<td>C38</td>
<td>48</td>
</tr>
<tr>
<td>X47</td>
<td>58</td>
<td>X39</td>
<td>49</td>
</tr>
<tr>
<td>X48</td>
<td>59</td>
<td>X30</td>
<td>41</td>
</tr>
<tr>
<td>X49</td>
<td>60</td>
<td>X50</td>
<td>44</td>
</tr>
<tr>
<td>X50</td>
<td>61</td>
<td>X51</td>
<td>45</td>
</tr>
<tr>
<td>X51</td>
<td>62</td>
<td>X52</td>
<td>46</td>
</tr>
<tr>
<td>X52</td>
<td>63</td>
<td>X53</td>
<td>47</td>
</tr>
<tr>
<td>X53</td>
<td>64</td>
<td>X54</td>
<td>48</td>
</tr>
<tr>
<td>X54</td>
<td>65</td>
<td>X55</td>
<td>49</td>
</tr>
<tr>
<td>X55</td>
<td>66</td>
<td>X56</td>
<td>50</td>
</tr>
<tr>
<td>X56</td>
<td>67</td>
<td>X57</td>
<td>51</td>
</tr>
<tr>
<td>X57</td>
<td>68</td>
<td>X58</td>
<td>52</td>
</tr>
<tr>
<td>X58</td>
<td>69</td>
<td>X59</td>
<td>53</td>
</tr>
<tr>
<td>X59</td>
<td>70</td>
<td>X60</td>
<td>54</td>
</tr>
<tr>
<td>X60</td>
<td>71</td>
<td>X61</td>
<td>55</td>
</tr>
<tr>
<td>X61</td>
<td>72</td>
<td>X62</td>
<td>56</td>
</tr>
<tr>
<td>X62</td>
<td>73</td>
<td>Y</td>
<td>57</td>
</tr>
<tr>
<td>Y</td>
<td>74</td>
<td>C63</td>
<td>58</td>
</tr>
<tr>
<td>C63</td>
<td>75</td>
<td>X63</td>
<td>59</td>
</tr>
<tr>
<td>X63</td>
<td>76</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The table continues with the same pattern as the beginning of Table 4.1A.
Figure 4.1: Simple sketch of the Hydrogen Peroxide production process. See the text for details.
4.2 The model used at Halmstad University
Thorsteinn Rögnvaldsson

We applied feed forward multi layer perceptrons (MLP) algorithms to the EKA data, and also some more standard linear methods like linear regression, principal components regression (PCR), and partial least squares (PLS). The latter methods were included as benchmarks for the more “esoteric” methods like neural networks, Bayesian statistical models, and regression trees.

4.2.1 Data, preprocessing, and variable selection

The first data set came in a set of Microsoft access files, which were exported to ascii text (this decreased the file size considerably). All missing values and erroneous values were then replaced by the value -1.0, since all measured variables were supposed to be positive. Erroneous values were e.g. negative values and outliers (occasional spikes with obviously unreasonable values). Some variables had only missing values and these variables were removed from the data set. This final data set had 78 remaining variables, of which two turned out to be a duplicate copy, and this data set was distributed to the other participants in the project. It is this data set that is described above.

The second data set (and the final hold out test data set) came in a much more convenient format, as a MATLAB .mat file. These files had already been cleaned to some extent by EKA, e.g. missing values had been “removed” (actually set to zero), but outliers remained.

The data was filtered before any models were constructed. This means that outliers, stops, and suchlike were removed from the data. Also, a separate data set was set aside for testing the models after the model selection process was finished.

Preprocessing the first data set: The filter steps for the first data set were

1. Remove data between 12:29 August 10 and 05:29 August 28. The process was stopped, or semi-stopped, during this period and the data may not have been representative for the process.

2. All entries with missing values were removed (no attempt was made to “fill in” missing values). Here great care was taken to ensure that the correct lags, as determined by the variable selection algorithm, were considered.

This data filtering was considerably more liberal than the data filtering used by SICS.

The last 10% of the data set was set aside for testing, i.e. the period from 21:20 September 16 until 20:16 September 26 (10 days).

The data was normalized to zero mean and unit standard deviation before models were built. That is, each variable (and the output) $x_k$ was linearly transformed into a new variable $z_k$ according to

$$z_k = \frac{x_k - \mu_k}{\sigma_k} \quad (4.1)$$

where $\mu_k$ is the average of $x_k$ over the training data, and $\sigma_k$ is the standard deviation for $x_k$ over the training data. The new variable, $z_k$, has zero mean and a standard deviation equal to one. After the model was built and test data was presented to it, the test data was transformed in the same way, using the mean and standard deviation from the training data.

Preprocessing the second data set: The filter steps for the second data set were

1. Remove all observations which correspond to “process off”, which was defined as
   - Hydrogen flow $< 250$ for all hydrogen flows into the process.
- Hydrogen valve openings < 10 for all valves for hydrogen into the process.
- If the flow of the working solution < 50, for any of the main valves in the process.

- The variables \{1, 2, 3, 4, 5, 6, 7, 14, 19, 28, 52, 56, 62, 65, 68, 71\} were not included in any runs. One of these was the date, some were empty, some were considered weird, and some were constant over long periods.

This filtering was even more liberal than the filter used for the first data set. Several spikes were allowed to pass, even some huge ones. The intention, however, was to filter in a simple and automatic way. The SICS group did a manual and much more conservative (in the meaning of removing everything that looked suspicious) filtering which was also distributed to the other groups.

Half the data was set aside for testing after the models had been constructed. This data was selected by dividing the entire data set into blocks, 10,000 entries in size (equal to 28 hours of data), and every second block was used for training and the other for testing. (Note that this is quite different from the procedure used for the first data set.)

The data was normalized to zero mean and unit standard deviation before models were built.

**Variable selection:** The variable selection was done in the same way for both data sets, using the mutual information measure. The procedure followed Battiti [1994], i.e. the variables were selected in a forward manner (one by one) to maximize the heuristic cross mutual information

\[
I(y, x_k) - \beta \sum_{j \in S} I(x_j, x_k), \tag{4.2}
\]

where \(\beta\) was a parameter between zero and one, and \(S\) was the set of variables selected so far. For example, assume that \(x_1\) had the maximum \(I(y, x_k)\) and was consequently selected as the first variable to include. The second variable was then selected as the variable that maximized

\[
I(y, x_k) - \beta I(x_1, x_k). \tag{4.3}
\]

Suppose that \(x_2\) was selected as the second variable, the third included variable was then the variable that maximized

\[
I(y, x_k) - \beta [I(x_1, x_k) + I(x_2, x_k)], \tag{4.4}
\]

and so on and so forth.

The variable selection procedure is summarized below:

1. For each input variable \(x_k\) and a set of delays \(\tau\), we computed the mutual information \(I[x_k(t-\tau), y(t)]\) between the lagged input variable and the output. Here we used \(\tau \in \{0, 15, 30, \ldots, 350\}\) minutes. This resulted in a total of \(78 \times 25 = 1950\) runs for the first data set, and a minimum of \(85 \times 25 \times 2 = 4250\) runs for the second data set (the factor of two came from the fact that we were dealing with two outputs in the second data set). The Pearson correlation coefficient was also computed so that the mutual information result could be checked against it.

2. For each pair of input variables \(x_j\) and \(x_k\), we computed the mutual information \(I[x_j(t-\tau_j), x_k(t-\tau_k)]\). However, here we used fixed values of \(\tau\), those values that corresponded to maximum mutual information with \(y\) (in step 1 above). This resulted in a total of \(78 \times 78 = 6084\) runs for the first data set, and a minimum of \(85 \times 85 = 7396\) runs for the second data set.

3. We computed, for each \(\beta \in 0, 0.1, 0.2, \ldots, 0.9, 1.0\), the set of variables that maximized expression (4.2). The final set of variables that were included in the model were those that corresponded to either \(\beta = 0\) (no interaction) or \(\beta = 0.5\) (the variable selection result did not change much above \(\beta = 0.5\)).
Figure 4.2: Example output for the variable selection for the first data set. The figures shows results for the output $y$, which is the density of the process output, against the input $x_1$, which is the hydrogen flow into one of the hydrogenators. Top left: The correlation function, the linear correlation between $x_1(t - \tau)$ and $y(t)$ plotted versus $\tau$. Top right: The correlation function for $\log[x_1(t - \tau)]$ and $\log[y(t)]$. Bottom left: The mutual information for $x_1(t - \tau)$ and $y(t)$ plotted versus $\tau$. Bottom right: The mutual information for the logarithms. Each curve is for one third of the data set.

This variable selection was done independently for three non-overlapping subsets of the data, so that the results could be compared after the selection. The set of selected variables was set to be maximally 10, to allow for more experimentation with the models. (This excepts the PLS and PCR models which consider all variables.)

Figure 4.2 shows a sample output for the variable selection for the first data set. Here the correlation function and the mutual information gave the same information, and it was decided in this case that the best lag between $x_1$ and $y$ was 150 minutes. Figure 4.3 shows a summary plot for 79 plots similar to the one shown in Figure 4.2. A similar plot could be shown for the cross correlations between variables, but it would be difficult to grasp.

### 4.2.2 Model construction and model selection

The data analysis software MATLAB was used throughout for all analysis. The PCR and PLS models were constructed using the PLSToolBox set of functions. The MLP models were constructed using the MATLAB NNET toolbox. The linear regression models were constructed using the public STIXBOX toolbox.

The linear regression models and MLP models were constructed based on the selected variable sets. A maximum of 10 input variables was allowed here. The PCR and PLS models, on the other hand, used all the variables, but projected onto a sub manifold in the data space. The dimension of this sub manifold (i.e. the number of principal components and the number of latent variables, respectively) was set to be maximally 20.

The MLP models were trained using two different techniques: Early stopping and weight decay using an automatic Bayesian technique for selecting $\lambda$. 
CHAPTER 4. EKA CHEMICALS

Linear regression, PCR, and PLS model selection: The linear models were all selected using \( N \)-fold cross validation. The training data set was split into \( N \) parts, \( N - 1 \) parts were used to set the parameters of the model, and the \( N \):th part was used to validate the model. This was repeated \( N \) times, yielding \( N \) different models, and the min, max, and median squared validation errors were plotted (see Figures 4.4 and 4.5 for examples). The best model was then selected “by hand” by looking at the plots and judging what would be a reasonable model, weighting model complexity (number of free parameters) against validation error performance. For the first data set, \( N \) was set to 10. For the second data set \( N \) was set to 3.

Early stopping MLP training: The early stopping training was done in the following way: The training data (the \( N - 1 \) parts mentioned above) was split into two parts, one for doing training validation (i.e. checking the network performance on an independent data set) and one for computing gradients etc. in the training. During training, the training validation performance was monitored with the idea of stopping the training when the training validation performance started to deteriorate. Surprisingly, this rarely happened for the first data set and the algorithms would stop after the maximum number of iterations. For the second data set, however, did the early stopping actually stop the training before the maximum number of iterations.

Bayesian weight decay MLP training: The Bayesian framework allows an automatic determination of the weight decay parameter [MacKay, 1992] [Thodberg, 1996], by iteratively determining the network weights and the hyperparameters (e.g. the weight decay parameter). We used a version of this method that is implemented in the MATLAB NNET toolbox.

MLP model selection: The model selection procedure was the same for all MLP training procedures (i.e. both early stopping and the Bayesian regularization scheme). The data was split into \( N \) data sets and \( N \)-fold cross validation was used to estimate the model’s performance on new data. For the first data
Figure 4.4: Model selection for the principal components regression on the second data set. The plot summarizes the results from $20 \times 3 = 60$ principal component regressions. For each number of principal components, the data set is split into three parts and three different models built, using two thirds of the data to construct the model and one third to validate the model every time. Here 18 principal components were selected as optimal for TC1105 and 3 for TC1125. The very high maximum cross validation errors indicate that one of the data sets (one of the three splits) contains observations that are very hard to model.

Figure 4.5: Model selection for the partial least squares regression. Here 6 latent variables was selected as the optimal for both TC1105 and TC1125.
set $N$ was set to 10, and for the second data set $N$ was set to 3. A number of different MLP setups were tried, $N_{\text{hidden}} \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and $N_{\text{inputs}} \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$. The $N$-fold cross validation error was estimated for each MLP setup, and the final optimal MLP architecture was selected manually by viewing the summary plots for the setups. It was always verified that the best MLP models did significantly better than corresponding linear models (i.e. linear models that used the same inputs as the MLP models). Figure 4.6 shows an example for the first data set.

The set of $N$ MLP models were then combined into an averaging committee when the predictions for the hold-out test data set were made. This means that the predicted output was

$$\hat{y}(t) = \frac{1}{K} \sum_{k=1}^{N} \hat{y}_k(t)$$

(4.5)

where $\hat{y}_k(t)$ was the output from model $k$ for the time step $t$.

### 4.2.3 Results

**The first data set:** The top ten variables that were selected for the first data set are tabulated in Table 4.2. The lags for the variables varied and several of the longer lags agreed with delays in the plant, which we took as an indication of the power of mutual information.

The variables corresponding to $\beta = 0$ and $\beta = 0.5$ were then used as inputs to a linear and a MLP model. The best, as judged by the cross validation errors, were the models with the maximum number of hidden units and input variables. The predicted outputs for the training data set (i.e. the data that was not kept for hold-out test) were considerably better than what a linear model could produce. The training set prediction were true predictions since they were produced using the cross validation procedure so that each model predicted an observation that it had not observed before.

However, when the committee of these networks was applied to the hold-out test data set, then the MLP was in no way superior over the linear model, as illustrated in Figure 4.7, which is representative...
4.2 The model used at Halmstad University

Table 4.2: The “top ten” selected variables for the first data set using mutual information, with the heuristic cross mutual information measure. The variable sets are ordered into columns, with the corresponding value at the top. The first three variables selected with \( \beta = 0 \) (i.e. no cross mutual information considered) are X62, X11, and X2. The variable X62 is always selected, since it has the highest mutual information with the output and is therefore always picked first. To each variable is connected an optimum lag (not shown).

<table>
<thead>
<tr>
<th>Rank</th>
<th>( \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>2</td>
<td>X11</td>
</tr>
<tr>
<td>3</td>
<td>X2</td>
</tr>
<tr>
<td>4</td>
<td>C31</td>
</tr>
<tr>
<td>5</td>
<td>C35</td>
</tr>
<tr>
<td>6</td>
<td>X47</td>
</tr>
<tr>
<td>7</td>
<td>C4</td>
</tr>
<tr>
<td>8</td>
<td>X3</td>
</tr>
<tr>
<td>9</td>
<td>C63</td>
</tr>
<tr>
<td>10</td>
<td>X20</td>
</tr>
</tbody>
</table>

Table 4.3: The “top ten” selected variables for the second data set, output TC1105. The variable sets are ordered into columns, with the corresponding \( \beta \) value at the top. All lags are zero.

<table>
<thead>
<tr>
<th>Rank</th>
<th>( \beta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>57</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>79</td>
</tr>
</tbody>
</table>

for all the other test results as well. Quite the contrary, the simple linear model behaved much better than the MLP models.

It was speculated that this could have been due to the low excitation of the target value, since even a constant output model would have done well on the problem. This belief was one of the reasons why it was decided to try a different data set with more excited outputs.

The second data set: The variable selection on the second data set did not imply any significant lags, why we used zero lags for all variables. The selected variables for the two outputs are listed in Table 4.3 and Table 4.4.

The variables that corresponded to \( \beta = 0.5 \) were used in the model construction. The best early stopping MLP models for TC1105 used 6 or 7 input variables, and a varying number of hidden units. The best early stopping MLP models for TC1125 used 4 inputs. However, the results were noisy and it was hard to draw clear conclusions (there was no strong difference in cross validation error between the different models). The best behavior was observed for TC1105, which seemed easier to predict. The best Bayesian regularization (weight decay) models used 8 or 9 inputs and 10 hidden units for TC1105. For TC1125 the best Bayesian regularization networks used 9 inputs, but the evidence was weak that these models were better than the others. Similarly did none of the linear regression models do much better than another linear model. We decided to keep the models with 10 inputs for testing. The cross validation errors for the PCR and PLS models are shown in Figures 4.5 and 4.4. Only the PLS models for the output TC1105 show a strong minimum for the cross validation error.
The models that were finally selected for the blind data set test are listed in Table 4.5 together with their performance on the test data that were kept aside from the start of the modeling (not to be confused with the blind test data set later supplied by EKA). The results in the table indicate that the MLP models could have an edge over the other models, since the test errors are smaller for them than for other models.

4.2.4 Summary and discussion

We applied variable selection using mutual information and feed forward neural networks to the EKA data, and benchmarked the performance against standard linear models. Based on the models’ performance on validation data (the validation data used in the cross validation) we would conclude that the MLP models should come out better than the linear models. A general caveat, however, is the time spent with variable selection and model construction. The linear PLS and PCR models were very quick to construct and thus allowed a lot of experimenting, and the performance differences in Table 4.5 are not that large.
### Table 4.4: The “top ten” selected variables for the second data set, output TC1125. The variable sets are ordered into columns, with the corresponding \( \beta \) value at the top. All lags are zero.

<table>
<thead>
<tr>
<th>Rank</th>
<th>( \beta ) 0</th>
<th>( \beta ) 0.1</th>
<th>( \beta ) 0.2</th>
<th>( \beta ) 0.3</th>
<th>( \beta ) 0.4</th>
<th>( \beta ) 0.5</th>
<th>( \beta ) 0.6</th>
<th>( \beta ) 0.7</th>
<th>( \beta ) 0.8</th>
<th>( \beta ) 0.9</th>
<th>( \beta ) 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>69</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
<td>20</td>
<td>33</td>
<td>44</td>
<td>8</td>
<td>45</td>
<td>72</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>57</td>
<td>20</td>
<td>33</td>
<td>44</td>
<td>8</td>
<td>44</td>
<td>27</td>
<td>8</td>
<td>44</td>
<td>74</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>79</td>
<td>76</td>
<td>72</td>
<td>72</td>
<td>74</td>
<td>72</td>
<td>27</td>
<td>74</td>
<td>27</td>
<td>54</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>11</td>
<td>44</td>
<td>8</td>
<td>74</td>
<td>75</td>
<td>8</td>
<td>44</td>
<td>27</td>
<td>45</td>
<td>44</td>
</tr>
<tr>
<td>7</td>
<td>23</td>
<td>69</td>
<td>72</td>
<td>45</td>
<td>27</td>
<td>27</td>
<td>54</td>
<td>54</td>
<td>45</td>
<td>44</td>
<td>74</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>33</td>
<td>8</td>
<td>27</td>
<td>45</td>
<td>72</td>
<td>74</td>
<td>45</td>
<td>54</td>
<td>54</td>
<td>45</td>
</tr>
<tr>
<td>9</td>
<td>79</td>
<td>36</td>
<td>45</td>
<td>74</td>
<td>54</td>
<td>54</td>
<td>48</td>
<td>74</td>
<td>72</td>
<td>72</td>
<td>48</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>76</td>
<td>74</td>
<td>54</td>
<td>48</td>
<td>48</td>
<td>44</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>18</td>
</tr>
</tbody>
</table>

### Table 4.5: Hold-out test set errors and correlation with the true output value for the models that were selected for each method (note that this is not the blind test data set later supplied by EKA). The MSE and correlation (\( \rho \)) values can not be compared directly with the other groups since the data set we used was considerably more noisy than theirs.

<table>
<thead>
<tr>
<th>Output</th>
<th>Model</th>
<th>Test data performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MSE</td>
</tr>
<tr>
<td>TC1105</td>
<td>MLP 8-10-1, Bayesian regularization</td>
<td>11.8</td>
</tr>
<tr>
<td>TC1105</td>
<td>MLP 6-8-1, Early stopping</td>
<td>13.2</td>
</tr>
<tr>
<td>TC1105</td>
<td>LR 10 inputs</td>
<td>19.5</td>
</tr>
<tr>
<td>TC1105</td>
<td>PCR 18 principal components</td>
<td>19.8</td>
</tr>
<tr>
<td>TC1105</td>
<td>PLS 6 latent variables</td>
<td>15.7</td>
</tr>
<tr>
<td>TC1125</td>
<td>MLP 9-6-1, Bayesian regularization</td>
<td>204.5</td>
</tr>
<tr>
<td>TC1125</td>
<td>MLP 4-6-1, Early stopping</td>
<td>198.0</td>
</tr>
<tr>
<td>TC1125</td>
<td>LR 10 inputs</td>
<td>220.6</td>
</tr>
<tr>
<td>TC1125</td>
<td>PCR 3 principal components</td>
<td>233.3</td>
</tr>
<tr>
<td>TC1125</td>
<td>PLS 6 latent variables</td>
<td>259.4</td>
</tr>
</tbody>
</table>
4.3 The model used at Skövde University
Fredrik Linäker

We applied recurrent neural networks to the EKA datasets, trying to determine the output values at each
time step. The strength of the neural network learning methods is that they can automatically determine
the most relevant input variables for the task. That is, all available input data can be fed into a neural
network without any advanced filtering or preprocessing except perhaps for some normalization.

A downside using the neural network approach however comes clear when trying to analyze a trained
neural network, especially if hidden layers are being used. It is then very difficult to determine which
the most relevant inputs are according to the network. That is, attribute selection and finding attribute
correlations is not a task which is well suited for neural networks, their very strength being that they
can be considered as simple, opaque, black-box systems which do not have to be understood in order to
work. The analysis problem becomes even more difficult when recurrent connections are being used, as
the resulting system may in fact show chaotic, i. e. very hard to characterize, behaviour.

4.3.1 The first set of experiments

On the first data set focus was on determining whether recurrent connections would in fact improve
performance on the task at hand, comparing a standard Feed-Forward (FF) architecture with a Simple
Recurrent Network (SRN) architecture. If the recurrent connections do not improve performance, they
should of course not be included in the network since they only make it harder to analyze, and then
standard FF analysis techniques could be used.

All 79 input variables were used in our simulations, 20 hidden nodes, and 1 output node were used.
All connection layers were fully connected, with the SRN having an extra 20 context nodes which fed a
copy of the previous hidden node activation as input to the hidden layer, thus providing a memory trace
of the previous inputs.

Several experiments, where the weights were randomly initialized, were conducted. The first 20000
data points were used for training, and the other 54190 data points (i. e. data points before the long
reported down time) were used as a test set. A mean square error of 0.07, with a standard deviation of
0.03 was obtained for both the FF and the SRN after 25 epochs on the training set. On the test set, a
mean square error of approximately 0.83, with a standard deviation of 0.21, was obtained. Outputs on
the training set are shown in Figure 4.8(a) and on the test set in Figure 4.8(b).

There were no significant difference in performance between the SRN and the simpler FF network,
_i. e._ the use of recurrent connections did not improve performance on the task. Several runs had to be
made since some of the networks get stuck early in a local optimum, whereas others can find gradually
better solutions due to better randomized starting positions in weight space. (Simulations with different
numbers of hidden units were also carried out, but again showing no significant improvement when using
recurrent connections.)

The recurrent connections do not seem to improve the performance of the neural network. Standard
feed-forward networks solve the task equally well. It should be noted that SRNs in general can handle
time dependencies up to something like five time steps in-between, and have in some rare cases been
shown to handle up to thirteen time steps. That is, if there are any long-term dependencies which can
improve performance, they probably occur more than five time-steps earlier. If it however is known that
there is a dependency between data points with a certain time gap, the presentation to the network can
be modified in a manner which enables the recurrent network to make use of this. That is, performance
could most likely be improved for the SRN if presentation order is based on data points which have been
shown to be correlated temporally, something which SICS looked at.
Figure 4.8: Typical output from the trained neural networks. The output matches the trained data set quite well, as shown in (a). When the same network is then subjected to new inputs, the resulting output guesses are not as good (b). To improve performance further, the training set could be extended; the whole data set could be used for training, but generalization performance may then decrease due to over-fitting.
4.3.2 The second set of experiments

Another EKA dataset was introduced. The problem involved trying to determine the C30 and C31 values at each time step. A baseline was calculated (see below) and compared to a feed-forward network as well as a recurrent network. The data in ewing.mat was used in all experiments. Each variable was normalized to the range -1.0 to 1.0 as tansig activation functions were used throughout in the neural networks. A total of 85 variables were available, and were used as follows:

- Variable 1 was not used as it only contained the date and time information. (This could potentially carry some information, and could be included in future experiments.)
- Variables 2-63 and 68-85 were used as input to the neural networks.
- Variables 64 and 66 were used as target outputs for the neural networks.
- Variables 65 and 67 were not used, as using them could be considered as “cheating” (too directly coupled with the output).

Due to time restrictions, training and testing of the networks was limited to the first 20,000 datapoints. A feed-forward network with 80 input units and 2 output units (see above) was constructed. The network had 20 hidden units and was trained using the BFGS quasi-Newton method in Matlab. A similar network but with recurrent connections to the hidden units was also evaluated, to see whether the net could use previous inputs to improve the performance.

Further, a simple baseline was calculated, namely a system which just produces the average of the outputs all the time. (After normalization, these values turned out to be -0.7003 for C30 and -0.0338 for C31, or -0.6921 and -0.2475 on the first 20,000 datapoints, respectively.) We however (correctly) expected that the neural networks would be better than such a simple scheme.

Using the average of the first 20,000 outputs as a prediction would yield a mean squared error, again on the first 20,000 datapoints, of 0.2126. Since the training is largely dependent on the initial conditions—sometimes causing the network to get stuck in a local minimum—five runs of 50 epochs each were conducted for each type of network. The results are presented in Figure 4.9.
The model used at Skäovde University

Figure 4.10: Results from 5 nets on each target variable. The variation between runs is quite low on training.

The feed-forward networks had after 50 epochs a mean squared error of 0.053 with a stddev of 0.02, and the recurrent networks had an error of 0.048 with a stddev of 0.01. While both the feed-forward and the recurrent neural networks greatly outperform the baseline approach, there is no clear difference in performance between the different networks. While the recurrent networks were better on average, the best network was a feed-forward one, with an MSE of 0.028. The differences between the nets does not seem to be statistically significant. Again, the recurrent connections do not seem to improve the performance on this task.

4.3.3 The third set of experiments

These experiments used the filtered data set from SICS, where variables #64 and #66 are now called #57 and #59. Again, simple recurrent neural networks (SRNs) were used for determining these variables, but this time separately.

Each of the 76 variables was normalized (scaled) to the range -1.0 to 1.0 as tanSig activation functions were used throughout in the neural networks. (The premmax function in Matlab was used.) Please note that all results in this paper are on these normalized values (i.e. the outputs were not de-normalized before MSE calculations). Datasamples 1-10000, 20001-30000, 40001-50000 were used for training and samples 10001-20000, 30001-40000, 50001-53303 for testing (described below).

Ten separate simple recurrent networks each with 74 input units, 10 hidden (recurrent) units and 1 output unit were constructed. Five of the networks were trained using variable #57 as target output, and the other five using variable #59. Thus, the problem involved the setting of \((74+10+1)\cdot 10+ (10+1)\cdot 1 = 861\) parameters for each such net (bias units were used). Several nets had to be used as the networks are trained using a local search method whose success depends greatly on the initial state (see also below). The nets were trained using the BFGS quasi-Newton method in Matlab. A single data sample at a time was presented sequentially to the neural networks, the recurrent connections inside the network providing a temporal context for the inputs.

Training curves are displayed in Figure 4.10. The mean squared error (to clarify: using the standard MSE-function in Matlab, i.e. not divided by 2) gave the following results after 50 training epochs (the best networks are in bold text):
The correlation coefficient for the best networks on the training set was 0.9722 for variable #57 and 0.9416 for variable #59. Variable #57 has a lower overall error than #59, which might be an effect of the normalization process, a smaller overall variation, or just because it really is simpler to predict.

Results from testing are summarized below. (Correlation coefficients are displayed after the MSE, in brackets.)

<table>
<thead>
<tr>
<th>MSE train #57</th>
<th>MSE test #57</th>
<th>MSE train #59</th>
<th>MSE test #59</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0044</td>
<td>0.0221 [0.8382]</td>
<td>0.0239</td>
<td>0.1095 [0.6248]</td>
</tr>
<tr>
<td>0.0045</td>
<td>0.0195 [0.8648]</td>
<td>0.0274</td>
<td>0.1337 [0.6908]</td>
</tr>
<tr>
<td>0.0042</td>
<td>0.0172 [0.8756]</td>
<td>0.0252</td>
<td>0.1206 [0.6968]</td>
</tr>
<tr>
<td>0.0049</td>
<td>0.0258 [0.8267]</td>
<td><strong>0.0205</strong></td>
<td>0.1313 [0.5943]</td>
</tr>
<tr>
<td><strong>0.0037</strong></td>
<td><strong>0.0146 [0.9027]</strong></td>
<td>0.0232</td>
<td>0.1202 [0.6815]</td>
</tr>
</tbody>
</table>

As expected, the errors on the test set are considerably higher than on the training set. The best net on the training set for variable #57 is also best during testing. The same does not hold for variable #59 where the best net on the training set was not best on testing. This sort of over-specialization is quite common in neural network learning; the net has over-trained on a specific set of data samples and can no longer generalize to other examples. (As can be noted, the best net for variable #59 in terms of MSE did however not have the best correlation coefficient.)

There is a greater variation in the results on the test set (±22%) compared to results the training set (±10%) for variable #57. The same did however not hold for variable #59 where the training set showed an 11% variation in results whereas on training the results differed by only 7%. (Percentages calculated: stddev / mean.) That is, for both variables there are variations in the results. This is because the network weights need to be initialized randomly each run, specifying the start location from which a local search is conducted during training. The effect of this starting position does matter on this data set; this need not always be the case (there are rare cases in which there are no local minima in which to get stuck). The nets do seem to capture relevant correlations between inputs and outputs. The outputs from the best nets for each variable are shown in Figure 4.12.

As mentioned earlier, the test set consisted of a total of 23303 data samples. Looking at the errors (Figure 4.13) for each of these, they are distributed over the entire data set. For variable #57 there are however some time points where the network has extra difficulties (around data sample 8000). The nets seem to have a tendency to dampen rapid fluctuations, as can be seen in Figure 4.12, for variable #59, time-steps 10000 to 14000.

How does the network perform the task? A downside using the neural network approach comes clear when trying to analyze a trained neural network, especially if hidden layers are being used. It is then very difficult to determine which the most relevant inputs are according to the network. That is, attribute selection and finding attribute correlations is not a task which is well suited for neural networks, their very strength being that they can be considered as simple, opaque, black-box systems which do not have to be understood in order to work. How well this black-box system actually works, can only be judged when compared with results of the other applied techniques.
Figure 4.11: The outputs from the best nets on the training set.
Figure 4.12: Outputs from the best nets on the test set. There are definite deviations from the target, but the main characteristics have been captured.
Figure 4.13: The distribution of errors (deviations from target) for both nets.
4.4 The model used at SICS
Anders Holst

4.4.1 Preprocessing of the first data set

When looking at the hydrogen peroxide process data, most variables look similar at a first glance: long quite stable periods interrupted by either spikes or dips, occurring at approximately the same times for all variables. This corresponds to the periods when the process was running or stopped respectively. One important question here is whether the interesting aspect to predict when the process will stop, how it behaves during upstart, or how it behaves when it is stable. At SICS we made the assumptions that the interesting part was probably during the running of the process, after the transients during upstart. The transients at startup looked all rather similar and probably has a well known form, and when to shut down the process is probably decided from external factors.

Therefore we started by removing parts from the data: from the time when the hydrogen to the process was stopped, and until it was running and stable again (which could take up to approximately 24 hours from the onset of hydrogen). The service period that occurred in the middle of the data, during which the process ran at half the normal speed, was also removed. To reduce the number of samples to a more reasonable size, the data was downsampled to one pattern every ten minutes, by averaging the original patterns. This resulted in a final set of 8516 patterns. These data were divided in four equal parts, of which the first and third were used for training and the second and fourth were used as test set.

4.4.2 Choice of method

SICS has done two different things here. Besides trying to predict the hydrogen peroxide, we have also made an analysis of the dependencies in the domain. For each input variable, diagrams were generated to show its relation to the amount of hydrogen peroxide. One diagram shows the points of the two series plotted against each other, to show if there is any systematic relation between them. Another diagram shows the change over time in the two series plotted against each other, to show if there is any systematic relation between the direction of movement between the two series. Then there are two correlograms to investigate if they depend on each other with some specific delay in time. For this analysis we used the mutual information and the mutual information rate between the time series. The information measures were based on a linear model of the data, a Gaussian distribution. These two diagrams show how strong the information shared between the series is for different time delays, up to 400 minutes in both directions.

The relation between the mutual information and the mutual information rate is that the mutual information tends to give a too high value on the correlation, due to the slow movements in the time series. The mutual information rate compensates for this, but instead requires a more complicated model to estimate, which makes it more noisy. Due to this, the diagrams with information rates show much lower levels of correlations and more emphasized peaks (not so smeared out over several different delays), but also more noise in the form of random spikes in the correlograms.

The linear information rate measure was also used between all pairs of variables in the domain, and those with strongest correlation was marked in a dependency graph of the domain, showing how the variables depend on each other. The complete set of diagrams were generated for those pairs of variables that correspond to arcs in the dependency graph. All these diagrams have been delivered to EKA separately. The dependency graph is shown in figure 4.14.

If the periods when the process is stopped and restarted is included, there are quite strong dependencies from many variables to the output variable (the density indicating the proportion of hydrogen peroxide). There are also some characteristic time delays, for example many variables affect the output after about 110 minutes. However, most of these correlations disappear when only the stable process parts are included in the data. Most of the correlation comes from the fact that there is a big difference in most parameters when the process is running and when it is stopped. The detected time delay almost entirely consists of the time it takes from when the hydrogen to the process is stopped until the output density starts to decrease because of this.
4.4 The model used at SICE

When the process is in its steady state on the other hand, there are no visible time delays, and there are only quite weak correlations between the input variables and the output. This of course means that the process is quite optimally regulated, and that small fluctuations are compensated for quick enough not to show up in the following steps of the process (which would otherwise be visible as time delay indications in the correlograms).

Most process parameters are moving quite slowly. However, there are also some faster components in them. The changes on these different time scales may each contain relevant information when trying to predict the output variable. Therefore, rather than taking a window in time as input to base the prediction on, each time series was transformed to three series capturing the changes on different time scales, and only the “current time” value from each of these new series were used as input. The new series were constructed by using Gamma filters (exponential decay filters) on the original time series. So, starting from the 10 minutes averaged series mentioned above, a medium time scale series was constructed by feeding it through a Gamma filter with gamma 0.1 (approximately smearing it 10 times). To produce the slow time scale series, this smeared series were fed through another filter with gamma 0.01 (smearing it a 100 times approximately). To make these three series less redundant, the medium scale series was subtracted from the original series, leaving only the fast fluctuations. Then the slow series was subtracted from the medium scale series leaving only the medium scale changes in that one.

The prediction was then made both on the original (ten minutes averaged) output variable and on similar time scale transformations of the output. This was to see if changes on some time scale is easier to predict than others. However, this time the Gamma filters used to smear the series was used “backwards” in time. This was to make sure that the prediction used “entirely past” values to predict “entirely future” values. Predicting past values from the current and past values may be easier (one could imagine that some input parameters are adjusted in response to the output), and using the normal “forward” Gamma filter would then appear to give better predictions than could be achieved in reality.
4.4.3 Results on the first data set

All predictions were made with a Naive Bayesian classifier, i.e., the evidence from each variable (in each time scale) was considered separately, and then combined into the total prediction. A Gaussian distribution was used to model the joint distribution over each of the input variables and the output. For comparison, a Gaussian distribution over the entire input and output space was also tried. The results are presented in table 4.6. (The output variable is not normalized, so the root mean square values are in the original unit of the output variable.) The predicted series are also shown in figures 4.15 to 4.18.

As expected, the joint distribution is severely over-fitted to the series. The naive Bayes manages much better, 0.7 is not so bad for a correlation coefficient on the test set. When looking at the different time scales, it is evident that the slow scale can be predicted very good (correlation coefficient 0.85 on the test set), whereas both the medium and the fast scale series are not predicted correctly at all. This could indicate that the fast changes are primarily noise in the process.

4.4.4 Preprocessing of the second data set

The second data set consists of mainly the same variables as the first, but sampled every ten minutes and during a much longer period. Thus, just as for the first data set, the largest dynamic in most of the variables is due to whether the process is running or not. Since again the interesting aspect may be what happens when the process is running, the parts when the process is stopped or just started was first removed. This left most variables in a much smaller range than before this removal.

The remaining samples were then split in chunks of 10000 each (the last one somewhat smaller), and the odd parts put together to a training set of 30000 patterns (plus some empty “padding” patterns in between the chunks, not to confuse the time series analysis in the joints between them) and a test set of 23303 patterns (padded in the same way).

<table>
<thead>
<tr>
<th>Description</th>
<th>Corr Train</th>
<th>Corr Test</th>
<th>RMS Train</th>
<th>RMS Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint space, total series</td>
<td>0.70</td>
<td>0.05</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Naive Bayes, total series</td>
<td>0.88</td>
<td>0.70</td>
<td>1.053</td>
<td>1.70</td>
</tr>
<tr>
<td>Naive Bayes, fast component</td>
<td>0.47</td>
<td>0.22</td>
<td>0.36</td>
<td>0.40</td>
</tr>
<tr>
<td>Naive Bayes, medium component</td>
<td>0.62</td>
<td>-0.04</td>
<td>0.93</td>
<td>1.34</td>
</tr>
<tr>
<td>Naive Bayes, slow component</td>
<td>0.92</td>
<td>0.85</td>
<td>0.75</td>
<td>1.09</td>
</tr>
</tbody>
</table>

Table 4.6: Prediction results.
When checking the dependencies between attributes, there were some stronger dependencies to the new output variables than what was found to the output variables used for the first data set (see table 4.7). This together with the much longer period of data, made it meaningful to try some more advanced models than for the first task: In addition to the naive Bayesian classifier, variations using a graphical model and a Markov model was tried. To let the Markov model handle the time dependencies by itself, there were no Gamma filters applied to the series this time.
Table 4.7: The variables with the strongest dependency to the output variables 57 and 59. The mutual information rate and the detected delay between the variables are also given. A positive delay means that the output variable is affected that many time steps later.

The naive Bayesian classifier combines the evidence from the individual input variables as if they were independent. The purpose of the graphical model is to take care of this by compensating for strong dependencies between different input variables. The strongest dependencies were searched for with mutual information rate, and of those strongest forming a “tree”, i.e. without cycles, were selected for the graphical model. The indices for those variables are shown in table 4.8.

The other limitation of the naive Bayesian classifier used here concerns time: only the values at one instant in time is considered for the prediction. To combine the evidence from several time steps, a Markov model can be used. It is similar to the graphical model, but has a less complicated form, since it always compensates for a dependence between two consecutive time steps of the same variable. This simple form also makes it possible to combine a Markov model and a graphical model, thus taking care of dependencies both between variables and in time. In the Markov models used below, only two time steps were considered.

The results of the four used models are shown in tables 4.9 and 4.10. The best model is in both cases to use both graphs and Markov models. However, the difference of using Markov models is very small. The main improvement compared to the naive Bayesian classifier is because of the graph model.

The prediction of the two variables made by the combined graph and Markov model is shown in figures 4.19 and 4.20. As indicated already in the tables, the second variable seems to be much harder to predict. In both cases there seem to be a systematic shift between the answer and the prediction. this explains why the RMS is quite high even though the correlation coefficient is quite good.
Table 4.8: The pairs of variables with strong dependencies that were used as "edges" in the graph model.

<table>
<thead>
<tr>
<th>Pair</th>
<th>Mutual information rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 29</td>
<td>0.775</td>
</tr>
<tr>
<td>47 51</td>
<td>0.614</td>
</tr>
<tr>
<td>9 33</td>
<td>0.536</td>
</tr>
<tr>
<td>29 30</td>
<td>0.487</td>
</tr>
<tr>
<td>37 38</td>
<td>0.433</td>
</tr>
<tr>
<td>4 9</td>
<td>0.391</td>
</tr>
<tr>
<td>31 32</td>
<td>0.364</td>
</tr>
<tr>
<td>15 29</td>
<td>0.323</td>
</tr>
<tr>
<td>33 34</td>
<td>0.230</td>
</tr>
<tr>
<td>47 53</td>
<td>0.205</td>
</tr>
<tr>
<td>36 44</td>
<td>0.205</td>
</tr>
<tr>
<td>4 19</td>
<td>0.195</td>
</tr>
<tr>
<td>3 29</td>
<td>0.187</td>
</tr>
<tr>
<td>58 67</td>
<td>0.169</td>
</tr>
<tr>
<td>22 26</td>
<td>0.154</td>
</tr>
<tr>
<td>20 39</td>
<td>0.147</td>
</tr>
<tr>
<td>4 37</td>
<td>0.142</td>
</tr>
<tr>
<td>22 25</td>
<td>0.141</td>
</tr>
<tr>
<td>18 56</td>
<td>0.126</td>
</tr>
<tr>
<td>21 26</td>
<td>0.126</td>
</tr>
</tbody>
</table>

Table 4.9: Prediction results on both the training and the test set, for the four models used, on the first output variable (#57).

<table>
<thead>
<tr>
<th>Description</th>
<th>Corr Train</th>
<th>Corr Test</th>
<th>RMS Train</th>
<th>RMS Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.781</td>
<td>0.510</td>
<td>4.15</td>
<td>5.53</td>
</tr>
<tr>
<td>Markov compensations</td>
<td>0.780</td>
<td>0.509</td>
<td>4.16</td>
<td>5.55</td>
</tr>
<tr>
<td>Graph compensations</td>
<td>0.882</td>
<td>0.781</td>
<td>2.87</td>
<td>4.00</td>
</tr>
<tr>
<td>Graph&amp;Markov compensations</td>
<td>0.883</td>
<td>0.785</td>
<td>2.86</td>
<td>3.98</td>
</tr>
</tbody>
</table>

Table 4.10: Prediction results on both the training and the test set, for the four models used, on the second output variable (#59).

<table>
<thead>
<tr>
<th>Description</th>
<th>Corr Train</th>
<th>Corr Test</th>
<th>RMS Train</th>
<th>RMS Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.689</td>
<td>0.527</td>
<td>18.1</td>
<td>18.7</td>
</tr>
<tr>
<td>Markov compensations</td>
<td>0.688</td>
<td>0.527</td>
<td>18.0</td>
<td>18.5</td>
</tr>
<tr>
<td>Graph compensations</td>
<td>0.720</td>
<td>0.570</td>
<td>16.8</td>
<td>17.3</td>
</tr>
<tr>
<td>Graph&amp;Markov compensations</td>
<td>0.720</td>
<td>0.570</td>
<td>16.7</td>
<td>17.1</td>
</tr>
</tbody>
</table>
Figure 4.19: The prediction result on the test set with the combined graph and Markov model on the first output variable (#57). The gray plot is the real value and the black plot the prediction.

Figure 4.20: The prediction result on the test set with the combined graph and Markov model on the second output variable (#59). The gray plot is the real value and the black plot the prediction.
4.5 The model used at Mitthögskolan
Mikael Hall and David Martland

The problem supplied by EKA was treated using K-means clustering, mutual information and self-organizing feature maps to do a training-example-subset selection and genetic algorithm was used for the final variable selection. The clustering was applied to the target instead of the input space.

4.5.1 The usage of K-means clustering

The data and preprocessing used was the same as that used at SICS. And, since the data didn’t contain any significant time lags, we decided not to use any time delays and we also decided to use ordinary feedforward networks. The control variables, which was to be predicted, were designed to keep the temperature of the working fluid low so that gas didn’t emerge. These targets are clearly not important to predict per se, but if they could be modelled by other process variables then maybe knowledge could be gained.

As laymen, the chemical nature of the problem led us to believe there maybe distinct states of behavior and that maybe some clustering of the data was needed. Normally this clustering of the data would be performed in the input space and different models would be trained on each cluster. However, assuming that process variables can be used to model the control variables, the values of the control variables should work as labels on input patterns, or groups of values as labels of groups of input patterns. We therefore decided to cluster the targets (although the clusters was found using only one of the targets). This can be done in different ways. One may simply divide the range of the target into equal sized subranges, but it is better to do this so that the distribution of values is taken into account. Hence the K-means clustering method was chosen. K-means search for K centroids (or points, means), such that, over the samples, the summed distances between each sample and it’s nearest centroid is minimized. The samples belonging to a centroid forms one of the clusters. The range of sample values in a cluster will be less than the naively chosen subranges above if that range contain more samples, since each of the K clusters compete to represent as small portion of the sample space as possible in a very solidary way.

An advantage of clustering the target instead of the input space is that if the input space is incomplete, so that there are unmeasured factors which affect the target, we might still use the inputs to build a submodel, assuming the unknown factors is roughly constant within a certain cluster (or state represented by that cluster). A trained feedforward network would in fact be forced to output a weighted mean of different output possibilities. By clustering the target we are able to chose between them and thus get a lower error. The disadvantage of clustering the target is obviously that we no longer can choose between submodels by the input, when the model is validated or put to work, since some input patterns may be equal in more than one cluster. Thus we have to choose a subset of the clusters and train one model on them all. To do this we wanted to use mutual information within each cluster for cluster selection as well as variable selection. Thus K must be somewhat low, since the mutual information is affected by sample size. It was decided that four clusters was reasonable. Also some initial trials suggested that one of the two targets were harder to predict. Thus we we used that one to form the clusters, since the harder to predict target would possibly reveal more information.

The K-means algorithm resulted in four clusters from the given train portion of the data, where two was considerably larger. The two larger ones both represented values of the target above the value 40, see figure 4.21.

4.5.2 Mutual information

To chose between clusters we wanted to calculate the mutual information values between each input and the target within each cluster. Thus we would also be able to perform a variable selection at the same time. However, since mutual information is not only sensitive to the variability of the signals, but also to the number of samples etc, the mutual information values was normalized with the mutual information value between the target against itself. Obviously, there is no more information to be sought for if we know the answer to our query. Of course this will not guarantee that the comparison will be unambiguous.
Figure 4.21: The mutual information is most trustworthy in cluster 4, as can be seen in the rightmost picture. The order of the clusters in the leftmost picture is 4,2,3,1 from top to bottom.

As can be seen in figure 4.21 one cluster seemed to get the highest mutual information. This was also confirmed by running networks modelling each cluster with the highest ranked variables according to their own ranking. However, the cluster in second place did not get second place according to the mutual information values. In order to validate this in some way we ran self-organizing feature maps.

4.5.3 Self-organizing feature maps

Self-organizing feature maps can be seen as a K-means algorithm where the centroids are linked to each other outside the sample space on a most often two dimensional grid, called the map. When the closest centroid is found for a sample, that centroid is moved towards the sample in the input space, along with its neighbours on the map. This way the map will be unfolded in the input space, performing vector quantization and clustering while the topology of the map is developing a two dimensional description of the sampled data. The so obtained map can be used to get a quick picture of high dimensional data, in numerous ways. Here we color code the input space coordinates of the centroids so that they can be compared. This way possible correlations can be seen even if the behavior is significantly different in
4.5 The model used at Mitthögskolan

The map is constructed by competition and cooperation but instead of minimizing one global cost as in the K-means, several cost functions are developed as groups of map units is formed by units which has learned to work together in the search for features in the data to describe. The features found is clearly dependent of the number of samples having this feature as well as the correlation between input space dimensions described by the samples and the power distribution of the feature over the input dimensions. Each centroid is a weighted average of the samples, where the weighting function depends not only by the samples, but also on the surrounding centroids, and the feature described by the group of units it belongs. So if a feature is constituted by only some input dimensions, the other dimensions will be more averaged according to a feature they do not describe and ultimately even give a totally false description of the behavior of these input dimensions, with regards to the fuzzy set of samples which may be denoted by that feature. This means that the interpretation of the map is highly complicated. In our case, we want to see how some variables correlate to a target variable in the different clusters. To some degree assure that the map will find features in which the target plays a role the target can be preprocessed to have a larger amplitude than the others. Thus it will cost more to neglect these features. This will in some sense tend to force the quality of the map as a summary of the data to go down, but not necessarily - hopefully the features found will be more relevant.

We have thus selected a set of variables according to mutual information within one K-means cluster. The other clusters displayed some ambiguity, so we want to use a self-organizing feature map, to select additional clusters. We ran different maps for each cluster using the ten highest ranked variables in the winning cluster, successively increasing the magnitude of the target dimension. As we can see in figure 4.22, clusters reveal more relevant features when the target has a much larger amplitude. The target need this boost in order to form the map appropriately. We also see that the variables in the two biggest clusters seems to be more correlated. And since this fits with what we already suspected, we selected the two largest clusters.

We are not content with the mutual information though, since there seems to be some amount of noise disturbing the ranking. We ran some models using the easy to predict target and it seemed that the winner was a linear PCA regressor, using the twenty three highest ranked variables and using the nineteen largest principal components. We trained these models on the training data given, divided into two portions, one for validation. The use of a validation set was used for the winning linear model to be able to compare it with non-linear models were a validation set to determine when to stop training is more important. We tested the models on the given test data, without any clustering. The result was good and likewise when substituting for the harder to predict target used for clustering. However, the list of inputs used may be unnecessarily large. To investigate this we finally used a genetic algorithm for variable selection.

4.5.4 The genetic algorithm

The ideas behind genetic algorithms is very simple analogies drawn from evolution, such as fitness, sex and disease. These operations are combined with a simple trial and error methodology. Here, we let subsets of input variables form the individuals. We then use the subsets in a linear model. The fitness will be the correlation between the target and model output on a test set. The data used is given by the best selected cluster only and we choose from the thirty highest ranked variables, according to mutual information. The algorithm started with a population of fifty random lists of ten variables. These lists, or individuals, were used to predict the target and ranked as described above. Then a sort of elitist procedure was used to create the next generation. The most fit individuals (top quarter of the list) was allowed to mate with a randomly selected individual regardless by ranking. This mating could be with itself, so there were a possibility of making no children. The offspring was created by forming new lists of variables by dividing the parents lists in two parts (at a random point) and forming two mixed lists. The new generation consisted of the survivors (the five highest ranked), the children and complemented by new random individuals. We ran five hundred generations. The mutation, or disease, operator occurred with 0.1 chance and altered one randomly chosen gene (variable).
Figure 4.22: There are eight maps in this figure, two for each cluster, arranged from top to bottom. For each cluster there are two versions, one for each scaling of the target (1,1000). Each map is represented as U-matrix, where the distance between neighboring unit is shown and by the i'th color coded input variable. The target is the eleventh variable. The magnified target maps are on the right side. As can be seen in the maps to the right, cluster two and four seems to be easier to model, while this is not apparent in the maps to the left. By giving the target much more power, the maps found more relevant features, enabling us to do a cluster selection.
4.5.5 Results

The genetic algorithm resulted in the selection of ten variables, for the easier target. They resulted in slightly better test results than before. The run selecting variables for modelling the other target lead to surprising findings. Ten variables were selected by the algorithm. But since the ranking using mutual information was more reliable in this case, we found that the three of the highest ranked variables that also were selected by the genetic algorithm was sufficient, when using a non-linear 3-2-1 PCA network.

The variables chosen and the correlation coefficients is found in the texts under the figures 4.24 and 4.23. The maps were computed using the SOM Toolbox, which for use with Matlab can be obtained free at http://www.cis.hut.fi/projects/somtoolbox/.
Figure 4.23: The network was a feedforward neural network PCA regressor with three inputs, two hidden nodes in the hidden layer and a linear output node. The correlation coefficient between target and model output was 0.942 and 0.824 for train and test respectively. The inputs were 11 60 67 and target 59 (the harder to predict target), using the same enumeration as is used at SICS. The target has the larger variance.
Figure 4.24: The network was a linear feedforward neural network PCA with ten inputs and a linear output node. The correlation coefficient between target and model output was 0.948 and 0.899 for train and test respectively. The inputs were 58 75 60 6 63 18 47 61 62 3 and the target 57 (easy to predict variable), using the same enumeration as is used at SICS. The target has the larger variance.
4.6 The model used at DSV

Lars Asker

4.6.1 Introduction

The regression tasks, when working with the new EKA dataset were to predict the values of one of two variables (C30 and C31) given the other variables as input. For each of these tasks DSV used the bagging option in the inductive logic programming tool Virtual Predict to train a combined model consisting of 20 regression trees.

4.6.2 Pre-processing

Examples 1-10000, 20001-30000, 40001-50000 were used for training and samples 10001-20000, 30001-40000, 50001-53303 for testing. The training data were subsampled by selecting every 10th example. This resulted in a training data set consisting of 3000 examples. The dimensionality of the experimental data was further reduced in the following way. First, 10-fold cross validation on the training data, was used to construct 10 models (regression trees). Any variable that had been used in at least one of the models was kept, while those variables that had not been used in any model were removed. This process was then iterated by again performing a 10-fold cross validation and removing variables etc. until no more variables could be removed. This resulted in a dataset represented using 27 variables + one regression variable (out of the original 76).

4.6.3 Experimental results

When the task was to predict the value of variable C30, the correlation coefficient was 0.78, the mean squared error was 15.01 and the mean error was 2.77. For variable C31, the correlation coefficient was 0.64, the MSE was 196.47, and the mean error was 10.95.

<table>
<thead>
<tr>
<th>Prediction Task</th>
<th>Corr Coeff</th>
<th>MSE</th>
<th>Mean Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>C30</td>
<td>0.78</td>
<td>15.01</td>
<td>2.77</td>
</tr>
<tr>
<td>C31</td>
<td>0.64</td>
<td>196.47</td>
<td>10.95</td>
</tr>
</tbody>
</table>

*Table 4.11: Prediction results.*
4.7 Results from the blind test

EKA Chemicals provided, after all the groups had specified and presented their models, a third data set that in all respects was like the second data set except that it spanned a different time period. The intention being that the groups should produce predictions for this data set and then the predictions would be compared. Halmstad University took responsibility for distributing the data to the other participants, after removing information about the true output values, so that the test became a true blind test. Halmstad University also ran the predictions against the true outputs.

The blind test data contains 26,064 entries and spans the period from 00:00 January 1 until midnight June 30, 1999. These are the six months following directly after the second data set ends.

The results for each group on the blind test data set are summarized below in Tables 4.12 and 4.13.

<table>
<thead>
<tr>
<th>Model</th>
<th>MSE</th>
<th>RMS</th>
<th>ρ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH (Clustering and linear PCR)</td>
<td>6.3</td>
<td>2.5</td>
<td>0.88</td>
</tr>
<tr>
<td>HiS (Recurrent ANN)</td>
<td>5.4</td>
<td>2.3</td>
<td>0.87</td>
</tr>
<tr>
<td>DSV (Linear PLS)</td>
<td>9.8</td>
<td>3.2</td>
<td>0.87</td>
</tr>
<tr>
<td>HH (Linear PLS, 6 latent variables)</td>
<td>9.6</td>
<td>3.1</td>
<td>0.86</td>
</tr>
<tr>
<td>SICS (Bayesian model)</td>
<td>47.1</td>
<td>6.9</td>
<td>0.86</td>
</tr>
<tr>
<td>DSV (Regression tree, 27 variables)</td>
<td>29.8</td>
<td>5.5</td>
<td>0.76</td>
</tr>
<tr>
<td>HH (Linear regression, 10 MI-selected inputs)</td>
<td>19.3</td>
<td>4.4</td>
<td>0.66</td>
</tr>
<tr>
<td>HH (Bayes weight decay MLP)</td>
<td>41.1</td>
<td>6.4</td>
<td>0.52</td>
</tr>
<tr>
<td>HH (Early stopping MLP 6-8-1)</td>
<td>66.2</td>
<td>8.1</td>
<td>0.43</td>
</tr>
<tr>
<td>HH (Linear PCR, 18 principal components)</td>
<td>33.6</td>
<td>5.8</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Table 4.12: Summary of results on the hold-out test set for the output TC1105. The results are ordered in decreasing order for the correlation coefficient.

<table>
<thead>
<tr>
<th>Model</th>
<th>MSE</th>
<th>RMS</th>
<th>ρ</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH (Clustering and nonlinear PCR)</td>
<td>67.1</td>
<td>8.2</td>
<td>0.94</td>
</tr>
<tr>
<td>DSV (Linear PLS)</td>
<td>238.6</td>
<td>15.4</td>
<td>0.74</td>
</tr>
<tr>
<td>HiS (Recurrent ANN)</td>
<td>66.1</td>
<td>8.1</td>
<td>0.69</td>
</tr>
<tr>
<td>HH (Early stopping MLP, 4-6-1)</td>
<td>417.3</td>
<td>20.4</td>
<td>0.63</td>
</tr>
<tr>
<td>HH (Linear Regression, 10 MI-selected inputs)</td>
<td>381.5</td>
<td>19.5</td>
<td>0.61</td>
</tr>
<tr>
<td>DSV (Regression tree, 27 variables)</td>
<td>196.4</td>
<td>14.0</td>
<td>0.54</td>
</tr>
<tr>
<td>HH (Linear PLS, 6 latent variables)</td>
<td>95.6</td>
<td>9.8</td>
<td>0.52</td>
</tr>
<tr>
<td>HH (Linear PCR, 3 principal components)</td>
<td>357.1</td>
<td>18.9</td>
<td>0.46</td>
</tr>
<tr>
<td>SICS (Bayesian model)</td>
<td>160.7</td>
<td>12.7</td>
<td>0.45</td>
</tr>
<tr>
<td>HH (Bayes weight decay MLP 9-6-1)</td>
<td>432.6</td>
<td>20.8</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Table 4.13: Summary of results on the hold-out test set for the output TC1125. The results are ordered in decreasing order for the correlation coefficient.

4.8 Summary and discussion

The results from the blind test are both discouraging and encouraging. There are only two models that definitely outperform globally linear models, Mitthögskolan and Högskolan i Skövde. Both these models are also consistently good on the training data (the second data set) and the test data (the blind test data, the third data set). The other models, e.g. Halmstad’s MLP models, perform very poorly on the blind test data even though the cross validation errors on the training data set have indicated otherwise.

There are several open questions related to the result. How much of the differences in results come from using different data (e.g. SICS data or Halmstad data, data that had more or less spikes removed)? Does mutual information work (uncertain) and, if so, when does it fail? Why do the Bayesian methods work so poorly?
In the beginning of the project, the following questions were asked:

1. Can we extract the dependencies between variables in the data? How do different methods for this variable selection compare?
2. Are the nonlinear techniques used in the DALLAS project better than the standard linear method used today, e.g., partial least squares? (This should be quantified.)
3. Is it possible to optimize the process using the models? For instance, can we achieve a higher yield without increasing the consumption of expensive raw material (i.e., anthraquinone)?
4. Can the models be used for diagnosis/analysis of the industrial process?
5. Can the models be used for supporting the operator, i.e., can they give a hint as to how to change some controls to achieve a desired operating state?

Regarding the first question: The best model in the blind test, the one from Mitthögskolan, did succeed with extracting a small set of relevant variables (as few as three were needed for the “hard to predict” output) and they do a very good job of predicting with these variable sets. One of the crucial steps here was the use of clustering (a “divide and conquer” idea) and genetic algorithms to really come to the bottom of which variables that mattered. None of the other two groups that used mutual information type methods (Halmstad and SICS) came up with the same variables. Why this is so is unclear (Mitthögskolan used the very same script as Halmstad for computing the mutual information, but Halmstad did not use the same data set as Mitthögskolan). The issue is not settled on why some methods work better than others but it is clear that Mitthögskolan have succeeded in finding the crucial variable set.

The contender for the winning seat, Högskolan i Skövde, also did a good modeling job but are unable to provide information on what variables are used in the model.

Regarding the second question: In general, the nonlinear techniques used here have not proved themselves better than e.g., PLS. Two groups produce a much better result than PLS or linear regression. However, these better results come at the expense of much more time spent with modeling. The PLS results are produced in “no time” whereas there are weeks and months of working with the nonlinear techniques.

Regarding the third and fifth questions: The answer to this question is still open. It would be an interesting exercise to use the best models, preferably the feedforward type for simplicity of understanding them, and see if they can give a hint on why things look like they do.

Regarding the fourth question: Perhaps, the possibility is definitely not ruled out.

4.9 References


Chapter 5

Nordisk Media Analys (NMA): The brand awareness task.

5.1 Introduction

Daniel Gillblad

5.1.1 The data

NMA analyses the effects of media investments using a method called tracking. This consists of recording all the investments made in most of the media types available, for example different newspapers, TV channels and outdoors advertising, on a daily basis. The impact of these investments are measured using telephone interviews with a statistically representative set of people, who answer questions about what brands they have noticed advertising for lately and which trademarks they prefer. The questions can be categorized into four different classes:

- Top of Mind advertising awareness (TOM)
- In Mind advertising awareness (IM)
- Preferred Brand
- Considered Brand

Top of Mind refers to the first brand mentioned by an interview subject when asked which brands he or she have noticed advertising for lately. In Mind are all the brands that the subject have noticed advertising for. The brand mentioned in the Top of Mind question is included in In Mind. Preferred and Considered refers to which brand that is preferred by the interview subject and which are possible. For example, if the questions concern cars, then Preferred would be which brand of car that most likely would be bought by the interview subject and Considered other possible car brands. The relationship between Preferred and Considered is similar to the relationship between Top of Mind and In Mind.

5.1.2 Preprocessing of the database

Two datasets have been delivered from NMA, one in late 1999 and one early 2001. From both datasets, time series were produced containing investments for a certain brand in eight different media types and the total investment with one sample a week. These media types were:

- Total media investments
• TV
• Radio
• Cinema
• Outdoors Advertising
• Morning Newspapers
• Evening Newspapers
• Popular Press
• Trade/Professional Journals

Time series showing the advertising impact was also constructed, with Top of Mind, In Mind, Preferred and Considered represented as a percentage of the total number of interview subjects on a weekly basis.

The first data delivered from NMA was in the form of a relational database, containing all investments and answers from telephone interviews between approximately August 1996 and August 1999. This database was flattened into two files containing the relevant information for this project, one for the investments and one for the interviews. From these, the relevant time series were produced. These series were used in all the early tests with the NMA data.

The second dataset delivered from NMA covered the period August 1996 to December 2000. The data was delivered as text files containing all media investments with the corresponding date and investor and the value of Top of Mind, In Mind, Preferred and Considered for each brand. Again, time series on the same format as before was produced. This data was used in the final evaluation of the methods on NMA data, using 1996 to 1999 as a training set and the year 2000 as the evaluation set.

5.1.3 The task

Predicting the advertising impact is an important application for NMA since that is essentially their core business. The task is to predict, from the media investments for different brands in previous weeks, the value of Top of Mind, In Mind, Preferred and Considered the next four weeks. Since the exact value of these predictions is not that interesting and since the data contains a fair amount of noise, an average of these four predictions is calculated. To evaluate the results, the smoothed predictions are compared to the same kind of average over the correct values, and a mean square error (MSE) and correlation coefficient are calculated.

It is generally much easier to predict Top of Mind and In Mind from the media investments than Preferred and Considered because of the much more direct influence of advertising. In the case of Preferred and Considered, accurate predictions are probably very hard to make. Therefore it was decided to concentrate on Top of Mind and In Mind in the final evaluation.
5.2 The models used at SICS

Anders Holst

SICS tried two prediction methods, both based on the Naive Bayesian classifier, and estimation of Gaussian distributions. Only the total media investments were used to predict In Mind and Top Of Mind in both methods. This is because a such a simple model as possible was preferred for this task, and the total investment seem to sum up the relevant information in the input pretty well.

The first method uses the four last weeks investments as input. Since these weeks are correlated with each other, a first order Markov model is used to compensate for these dependencies. To be more specific, the complete equation used for calculating the prediction is:

\[
P(y_t | x_t, x_{t-1}, x_{t-2}, x_{t-3}) \propto \frac{P(y_t)P(x_t | x_{t-1}, y_t)P(x_{t-1} | x_{t-2}, y_t)P(x_{t-2} | x_{t-3}, y_t)P(x_{t-3} | y_t)}{P(x_{t-1}, y_t)P(x_{t-2}, y_t)}
\]

All the distributions in the equation are estimated as Gaussian distributions, and once specific values are inserted on the $x$:es, the result is a Gaussian distribution for $y_t$. The mean of this distribution is used as the prediction. The results from this model is labeled SICS 1 below.

The other model is even simpler. The average over the four last weeks is used as a single value from which to predict the output, using a singe Gaussian distribution. There is one twist though: The investment the current week is also used to estimate the difference in the output value. This difference is then added to the current week prediction, to get an adjustment due to this possible change. The motivation for this is that when trying different filters on the inputs and output, the highest correlation to the output was found from the average of the four last week (rather than any specific one of those weeks), and there were also a rather high correlation from the last week to the difference in output value between weeks. One would imagine then that to estimate the output for the current week, the estimation of the difference should be added to the estimation of the output for the last week. However, the estimates from week to week vary so much, so that a more consistent and stable approach is to say that the last weeks output is probably the same as this weeks output, except for the estimated difference. Therefore the estimation of the difference is added together with the above prediction of the same week, although the sum is weighted with the certainty (the variance) of the two estimated Gaussian distributions. To use the difference was a kind of experimental idea, but it probably had a very limited effect, since the estimation of the difference usually had a quite high variance, and therefore only affected the other prediction in a very limited way. Anyway, the results from this model is labeled SICS 2 below.
5.3 The model used at Mitthögskolan
Mikael Hall and David Martland

The problem supplied by NMA was treated by using feedforward neural networks. The data set was augmented by integrating and by taking first differences of the inputs.

5.3.1 Treatment

The model space used was given by feedforward artificial neural networks with time delays and converged to networks which had 5 hidden units plus linear output, and delays 0-3. The inputs consisted of the first difference and a running four point mean of the given inputs. The data consisted of telephone interview records regarding the visibility in media for certain car brands and for companies selling clothes and patterns of investments in various media types made by these companies. The following inputs were available: Total investment, TV, Morning newspapers, Evening newspapers, Popular magazines and Specialized magazines.

The task was to predict the running four point mean of targets 'Top of Mind' and 'In Mind', which reflect the frequency interviewees report having seen the brand in media (the first brand they recall and at all, respectively), from patterns of investment. The models were tested at NMA on previously unavailable data, consisting of targets from year 2000.

The idea behind using the first difference and the running mean was to give the network a more explicit description of the current state. However this contributes to a certain redundancy of information in the inputs besides increasing the ratio between the number of model parameters and the number of training examples. Therefore steps were taken to reduce the networks ability to overtrain. First some of the available input variables were removed - Radio, Cinema and Outdoors. These variables had usually no or few data entries. Secondly a validation set was used to determine when to stop training. Lastly we used a cost function that besides the mean square error also takes into account the sum of the absolute values of the network weights, thus impairing the ability to fit the training data too closely. This type of addition to the cost function, or regularization, is called weight decay. To see what this does, one have to know how the transfer function used look like. In this case we use a shifted and scaled version of the sigmoid. The sigmoid is given by:

\[ f(\omega) = \frac{1}{1 + e^{-\omega}} \]

where \( \omega \) is a linear combination of the inputs, \( \omega = \sum w_i x_i \). The sigmoid squash the range of this sum into the range 0 and 1 in a symmetrical way. If \( \omega \) approach -\( \infty \) \( f \) will tend to 0, if \( \omega \) approach \( \infty \) \( f \) will tend to 1, and if \( \omega \approx 0 \), \( f \) will keep the sum down almost to the same degree as up, so the behavior is almost linear. When weight decay introduce the bias \( \sum w_i^2 \approx 0 \), the non-linear network must find stronger evidence to develop non-linearities.

The model was run one hundred times for each brand from which the best was chosen according to a test set from the available data.

5.3.2 Results and conclusions

The performance showed a large variance over the brand names. On certain brands good predictions could be made, while other brands were harder to predict. This can be explained by the method of selecting the best network. Instead perhaps we should have formed the prediction of the mean of all (or almost all the runs neglecting the most extreme). This can also depend on changes in media investment behaviour by the harder to predict brands.
5.4 The model used at DSV

Lars Asker

Given the task of predicting the values of the two variables Top of Mind, and In Mind, DSV used the method Bagging-50. After some initial experiments it was decided to use only the total investment for each week for the previous four weeks to represent the data. An additional feature that was also experimented with was the number of different media types for each week, but this additional feature made no difference for the prediction results. The results are presented below in Table 5.1 in Section 5.7.
5.5 The model used at Skövde University
Lars Niklasson

5.5.1 Introduction

Data from three different domains was supplied; the car industry, the travel industry and the clothing industry. For each industry, the investments per week (divided into different media types, e.g. TV, outdoor, popular press, etc.) are available. Connected to this, information of the effects of the advertising is also given in the data sets. Each week NMA randomly interviews a number of individuals and rate the percentage of individuals who first mentioned that they had seen an advert for a particular make/brand/company, as the ‘Top of Mind’ answer for that make/brand/company. In addition, ‘In Mind’ (i.e. if the make/brand/company is mentioned at all), ‘Preferred’ (i.e. if the particular make/brand/company is the preferred choice) and ‘Considered’ (i.e. that the make/brand/company could be a possible choice of purchase) are also given.

The task is to use data from investments and predict different effects of the advertising, e.g. rating an alternative as ‘Top of Mind’, ‘In Mind’, ‘Preferred’ or ‘Considered’. A typical example can be found in Figure 5.1, where the percentages have been scaled to hide the actual values.

5.5.2 Recurrent networks for time-series prediction

The technique that HS apply is based on simple recurrent artificial neural networks (Elman, 1990), often termed SRNs. An SRN is a network that utilizes implicit representation of time. In this particular case, the investments in advertising (divided into different media types) each week are used as input to a recurrent net. The output is the predicted value of the particular variable describing the effect of the advertising, in terms of ‘Top of Mind’, ‘In Mind’, ‘Preferred’ or ‘Considered’.

One network is trained on the data for each make/brand/company. The hypothesis is that different makes/brands/companies have different characteristics, i.e. that there is no ‘general’ model for all (this was also tested in a series of simulations). So far, only data from the car industry has been used.
5.5.3 The data


One network is trained for each of the four variables, to give a prediction for a particular make. The first 100 weeks are used as training data, and the complete set (about 150 weeks) is used for testing. This means that at least 50 weeks have not been included in the training of the network.

As is often the case with ANN experimentation, a series of simulations was needed to decide on the network topology, i.e. training algorithm, representation of data, number of hidden units, etc. Some initial simulations showed that 15, rather than 10 hidden units gave the best result. It should be noted that this is a rather ad hoc number and that no thorough sensitivity analysis has been conducted. The initial simulations also showed improved performance when 'share of voice' was included in the input data. The inclusion of this variable is dependent on its availability during testing. It might be hard to include in long term predictions, since it requires that the investments for a company/make (which is assumed to be known) is related to the investment of the whole industry (which appears to be harder to predict).

5.5.4 Some assumptions

The particular training environment (Matlab) used demands that a min and max value is given for each variable. The approach adopted here is to scale each value such that it is divided with 3 times the max value (in the training set) for the particular variable in question. The hypothesis is that extrapolation will start linearly rather than squashed (since sigmoidal units are used). This hypothesis can simply be tested by re-running the simulations using another scaling factor.

Since multi-layered ANNs have a tendency to generate different results depending on the initial random weight sets, the approach here is to run each simulation 5 times and take the average output as result. The current model does not have a sensitivity check for each output (e.g. standard deviation), but that is trivial to include. If so, it would become obvious if different networks come to very different results. This could be used to identify makes/brands/companies for which the prediction is less reliable. The cause could be related to various situations, e.g. errors or noise in the data, availability of training data, etc.

5.5.5 The simulations

Each network (i.e. one output variable for one make/company) is trained in 5 independent simulations using 100 weeks for training and the complete set for testing (plotting both network output and target values). The 100 week sequence is trained 2000 times (epochs). Total simulation time for each network is about 1.5 hours. This time can probably be substantially shortened if implemented in a more suitable programming language (e.g. C) or the training regime is improved.

5.5.6 Results

The approach taken here is to be objective in the separation of training and test data. The first 100 weeks are always (independent on data quality) used as training set. The results are the average output from the 5 independent networks.

5.5.7 Top of Mind

This variable appears to be predictable for some makes. The problem is that some makes do not have enough data for the training set. Another problem is that the values for some makes is in the region of
1/10 of percents. Since the number of people interviewed each week is in the hundreds, the results for these makes are very sensitive for noise. Examples of makes which have any of the problems include, Alfa Romeo, BMW, Chevrolet, Chrysler, Fiat, Honda, Jaguar, Jeep, Mazda, Lada, Peugeot, Renault, Rover, Seat, Skoda, Subaru, Suzuki.

The makes which appear to be possible to predict include, Audi, Ford (the best candidate), Hyundai, Mitsubishi, Nissan, Opel, SAAB, Toyota, Volkswagen and Volvo. Included here is the best (Ford) and worst case (Volkswagen) for these.

Mercedes has a special problem. During the introduction of a new car, it turned over in a test. This had a large effect on the attention it received in the media. The problem occurred around weeks 95-100, and caused the model to extrapolate in such a way that investments for Mercedes were predicted to have an extremely high effect, in the test set. This could possibly also explain some effects for other makes, e.g. Volvo, since ‘Top of Mind’ is a relative variable.

5.5.8 In Mind

The result for this variable is very similar to the results for ‘Top of Mind’.

5.5.9 Preferred and Considered

These variables are much harder to predict. The results are not nearly as clear as the ones presented above.

5.5.10 The results

The results of these tests are presented in Table 5.1 in Section 5.7 summing up the results of all partners.
Figure 5.3: The results for Volkswagen.

Figure 5.4: The large generalization error for Mercedes, caused by an extreme interest in the car, due to some technical errors.
Figure 5.5: ‘In Mind’ for Ford.

Figure 5.6: ‘In Mind’ for Volkswagen.
5.5 The model used at Skovde University

Figure 5.7: ‘In Mind’ for Mercedes.

Figure 5.8: ‘Considered’ for Ford.
Figur 5.9: ’Preferred’ for Ford.
5.6 The model used at Halmstad University
Thorsteinn Rögnvaldsson and Jim Samuelsson

5.6.1 Definitions

Before presenting our results, we introduce some definitions.

We define the following sets of brand names, media categories, and advertisement impact categories:

\[
C = \{\text{Alfa Romeo, Audi, BMW, Chevrolet, Chrysler, Citroën, Daewoo, Daihatsu, Dodge, Fiat, Ford, Honda, Hyundai, Jaguar, Jeep, Kia, Lada, Land-Rover, Lexus, Mazda, Mercedes, Mitsubishi, Nissan, Opel, Peugeot, Renault, Rover, Saab, Seat, Skoda, Subaru, Suzuki, Toyota, Volkswagen, Volvo}\},
\]

\[
G = \{\text{tv, radio, cinema, outdoor, morning papers, evening papers, popular press, specialist press}\}
\]

\[
A = \{\text{Top of Mind, In Mind, Preferred, Considered}\}\]

\[
C_{\text{german}} = \{\text{Audi, BMW, Mercedes, Opel, Volkswagen}\},
\]

\[
C_{\text{asian}} = \{\text{Daewoo, Daihatsu, Honda, Hyundai, Kia, Lexus, Mazda, Mitsubishi, Nissan, Subaru, Suzuki, Toyota}\},
\]

\[
C_{\text{swedish}} = \{\text{Saab, Volvo}\}
\]

\[
C_{\text{others}} = \{\text{Alfa Romeo, Chevrolet, Chrysler, Citroën, Dodge, Fiat, Ford, Jaguar, Jeep, Lada, Land-Rover, Peugeot, Renault, Rover, Seat, Skoda}\},
\]

\[
C_{<50} = \{\text{Alfa Romeo, Chevrolet, Daewoo, Daihatsu, Dodge, Jaguar, Jeep, Kia, Lada, Land-Rover, Lexus, Subaru, Suzuki}\},
\]

\[
C_{50-150} = \{\text{BMW, Chrysler, Citroën, Fiat, Honda, Mazda, Mercedes, Peugeot, Rover, Seat, Skoda}\},
\]

\[
C_{>150} = \{\text{Audi, Ford, Hyundai, Mitsubishi, Nissan, Opel, Renault, Saab, Toyota, Volkswagen, Volvo}\}.
\]

For later work, we have also set the following brand names aside to be used as an out-of-sample test set:

\[
C_{\text{test}} = \{\text{Kia, Lada, Land-Rover, Lexus, Peugeot, Rover, Saab, Seat, Skoda, Subaru, Suzuki, Toyota, Volvo}\}.
\]

The Swedish brands Volvo and Saab were set aside for the reason that Swedish brand names have a “national” aspect that none of the others show. This fact is illustrated in figure 5.10.

We denote an individual car brand name by \(c\), a media category by \(g\), and \(w\) denotes the week number. Then \(i_{cg}(w)\) is defined as the advertisement investment, measured in money, made by car maker \(c\) in media category \(g\), during week \(w\).

We define \(t_{c}(w)\) as the level of Top of Mind that car maker \(c\) has reached during week \(w\).

We have set two tasks for our work so far:
1. For an individual car brand, explore the possibility of constructing a model for how investment affects the Top of Mind. That is, construct prediction models of the form: 
\[ \hat{t}_c(w) = f_c(i_{cg}(w), i_{cg}(w-1), \ldots) \].

2. For all car brands, explore the possibility of constructing a general model for how investment affects Top of Mind. That is, construct prediction models of the form: 
\[ \hat{t}(w) = f(i_{cg}(w), i_{cg}(w-1), \ldots) \].

Here the hat \( \hat{t} \) over \( t \) indicates that it is an estimate and not the true value. In the first case, we use historic data about car brand name \( c \) and try to predict how the mapping from investment to Top of Mind will be in the future. In the second case, we use historic data from a pool of car brand names, and try to find a model that generalizes to another brand name.

All models are evaluated using the normalized prediction error (NPE)
\[
NPE = \frac{\sum_w \left[ \hat{t}(w) - t(w) \right]^2}{\sum_w \left[ \hat{t} - t(w) \right]^2},
\]

where \( \hat{t} \) is the mean Top of Mind over the training weeks, and the sum runs over all test weeks (or training weeks if the NPE is computed over the training set). The NPE is a measure of how good our model is compared to the best constant model, which outputs the mean training value. An NPE close to unity indicates a less well performing model.

We throughout use the first 100 weeks for training and the last 50+ weeks for testing.

### 5.6.2 Data preprocessing

**Removing data**

It is unusual to find any investment in movie commercials (some car manufacturers invest in movie commercials but most do not). We have therefore excluded the movie media from our investigation.

**Transforming data**

The investment data is (obviously) not normally distributed. Instead, it is rather skewed since small investments are significantly more frequent than large investments. We have tried to transform the investment data using Box-Cox like transformations, i.e.
\[
x \rightarrow (x + \lambda)\gamma,
\]
so that the data is less skewed.

**Normalization**

It intuitively seems reasonable to normalize the numbers, both investment quantities and impact quantities, before building a model. We have tried two different normalization strategies for the investment data. These are:
\[
\hat{i}_{cg}(w) = \frac{i_{cg}(w)}{\sum_C i_{cg}(w)}
\]
\[
\tilde{i}_{cg}(w) = \frac{i_{cg}(w)}{\sum_C i_{cg}(w)}
\]

where the sum in equation (5.15) runs over over all car brands, and the sum in (5.14) runs over the different car brands as well as media categories. Expression (5.15) is a measure of the "share of voice" in
media channel \( g \), whereas expression (5.14) is a measure of the “share of voice” in the total signal from all media channels and car brands.

Note: The data contains a variable that is expected to hold the total investment over all media groups in week \( w \). We found that this differed quite a bit from what we get when we sum over car brand names and/or media groups. We therefore did not use the “total” variable in the data set.

For Top of Mind, we normalize according to

\[
\tilde{t}_c(w) = \frac{t_c(w)}{\sum_c t_c(w)}
\]

where the sum runs over the different car makers. This normalized Top of Mind is a measure of what fraction of Top of Mind the car brand name has among people who actually have a Top of Mind opinion at all. The unnormalized Top of Mind typically sums to only about 0.7, when summed over all car brand names, indicating that about 30% of the interviewed subjects have no car name on the top of their mind.

Filtering

The data is quite noisy and it is difficult to discern any clear structure. We therefore filter the data to remove the high frequency components.

We have tried two different methods for filtering the investment data, based on two different assumptions about the process. The first is a moving average, the second is a first order Gamma filter.

The moving average filter works as follows:

\[
I_{r1}^{cg}(N; w) = \frac{\sum_{n=0}^{N-1} t_{cg}(w-n)}{N}; r = 1, 2.
\]

That is, the investment in week \( w \) is replaced by the average investment over the current week and previous \( N - 1 \) weeks (\( N = 5 \) typically). The “assumption” here is that an investment made in week \( w \) will have an impact \( N \) weeks into the future, and the fractional impact is the same for all weeks, but no impact after that.

The first order Gamma filter works as follows:

\[
I_{r2}^{cg}(\beta, w) = \frac{\sum_{n=0}^{\infty} \tilde{t}_{cg}(w-n)e^{-\beta n}}{1 - e^{-\beta}}; r = 1, 2.
\]

That is, the investment in week \( w \) is replaced by the exponentially averaged investment over the current week and previous weeks. The “assumption” here is that an investment made in week \( w \) will have an exponentially decaying impact on “all” weeks into the future, with a decay factor \( \beta \) (\( \beta = 0.3 \) typically).

The Top of Mind is only filtered with a moving average filter:

\[
T_{r1}^{1}(N; w) = \frac{\sum_{n=0}^{N-1} \tilde{t}_c(w-n)}{N}
\]

We have tried both building models for the filtered \( T_{r1}^{1}(N; w) \) and for the unfiltered \( t_c(w) \). In the latter case we filter the predictions and the true Top of Mind afterwards, to be able to compare the results.

5.6.3 Results

The time lag

There are several different methods for estimating the time lag between the “input”, i.e. the investment, and the “output”, i.e. the Top of Mind score. We have tried four different methods:
• Computing the Pearson correlation coefficient $\rho$.
• Computing a correlation between the sign of change for the different series, i.e. sign runs.
• Look at the series and subjectively judge what seems to be the lag.
• Construct models with different lags and see which model that gives the best performance.

The “optimal” time lag also depends on the filtering method.

The moving average filter

When using the moving average filter (5.17) we conclude that the “optimal” time lag seems to be of the order of two weeks. This will be illustrated below, for the car maker Ford.

The Gamma filter

We explored the time lag when using the Gamma filter in several ways. First we experimented a little to find a suitable value for the decay constant $\beta$. This was set to $\beta = 0.3$. We then did single variable linear regressions for each of the variables and for different groups of car brand names, using different lags.

In the case of large investors, $C_{250}$ (excluding the car manufacturers that are in the test set $C_{test}$), we concluded that the lag for TV commercials is zero, whereas other media seem to have a lag of 2-4 weeks. However, the TV media has the overall strongest effect on the Top of Mind.

In the case of medium size investors, $C_{50-150}$ (excluding the car manufacturers that are in the test set $C_{test}$), we concluded that the lag for TV commercials is 2, whereas other media have a lag of zero weeks. In this group the TV is not dominant. Instead movies, popular and evening press seem to have the strongest effect (although the effect is not very prominent).

Because of this observed difference between groups of car brands we therefore chose to work only with large investors. We try both a lag of 2 weeks for all inputs, and to set the lag for TV commercials to zero and other advertisement media to 2 weeks. The exponential decay in the Gamma filter is set to $\beta = 0.3$ (no significant difference is seen for $\beta \in [0.25, 0.45]$).

Linear models for individual car brands

We do most of the model exploration using linear models, to allow more freedom for choosing input variables and time lag. We have tried using both partial least squares (PLS) and ordinary least squares (LS) for the model construction. PLS was used in the case of moving average filter on the inputs, LS was used in the case of the Gamma filter representation.

In the PLS case, we also normalize the input variables to have zero mean and unit variance. In the LS case we do not do this.

Ford benchmark: PLS using moving average filter

To show the impact of changes in the models when varying different model parameters we have presented and discussed above, we use the following model parameters as a benchmark reference:

(i) car maker: $c = Ford$, denoted $F$ below

(ii) normalization: $r = 1$ (i.e. sum over both media group and brand names)

(iii) filtering: $s = 1$ and $N = 5$ weeks (i.e. moving average filter)
(iv) time lag: \( l = 2 \) weeks (for all inputs)

(v) media categories: \( G - \text{cinema} = \{ \text{tv, radio, outdoor, morning papers, evening papers, popular press and specialist press} \} \) (cinema is excluded because of lack of data)

When training brand name specific models, we use as training set (\( \Omega_{\text{train}} \)) the first 100 weeks out of the total number of weeks that are available (typically 157 weeks). The remaining weeks constitute the test set (\( \Omega_{\text{test}} \)).

The NPE values for the Ford benchmark model are: \( \text{NPE}_{\text{train}} = 0.36 \), \( \text{NPE}_{\text{test}} = 0.48 \).

**PLS using moving average & varying model parameters**

Figure 5.11 shows (to the left) the two curves that represent our benchmark reference model defined above: the real data for Top of Mind, \( T_{5}^{F}(5; w) \) and the corresponding modeled values, \( \hat{T}_{5}^{F}(5; w) \). To the right in the same figure is shown what happens when we choose a different normalization of the input data; \( i.e. r = 1 \rightarrow 2 \). In the latter case, the investments are normalized over media categories only. In the benchmark case, the normalization is done over both media categories and car brand names.

There seems to be no major difference between the two normalization methods.

In figure 5.12 (left) we have switched back to \( r = 1 \), but instead turned off the moving average filtering, which mathematically means \( N = 1 \) week. The predictive power goes down (obviously since our output now has a larger noise content), which is indicated by a higher NPE value. In Figure 5.12 (right) we plot NPE as a function of \( N \).

Going back to \( N = 5 \) weeks, we now remove the time lag, \( i.e. l = 0 \). Figure 5.13 (left) shows the result, and in the right part of the figure we plot NPE as a function of \( l \). Here we use the same lag for all inputs.

**Results for some other car makers**

To investigate how well our model does for some other car makers, we pick other cars from the subsets \( C_{<50}, C_{50-150}, C_{>150} \) defined in (5.8), (5.9) and (5.10). From \( C_{<50} \) (to which Ford also belongs) we choose Audi and Volvo, from the middle group \( C_{50-150} \) we take BMW, and from the low-level investors \( C_{<50} \) we choose Alfa Romeo. We otherwise use the same parameters as in the benchmark reference set.

In figure 5.14 we see that for Audi our model performs rather well, for the training set as well as the test set. For Volvo however, the model has problems, especially for the data in the test set. Figure 5.15 illustrates the situation for BMW and Alfa Romeo. The NPE values for both these car makers are \( \approx 1 \), confirming the impression of poor performance from the plots. The poor performance is probably due to lack of data for these two car makers.

**PLS with moving average & impact of media groups**

Having convinced ourselves that our approach makes sense, we can now tackle the probably most interesting problem: Through which media group does an investor reach the highest Top of Mind level? We have in the first phase of the project addressed this question by studying how the NPE value varies with the choice of input; \( i.e. \) media groups. Can we, for example, make a model with good predictive power by only using a few media groups; \( i.e. \) a few members of the set \( G \)? Once again we choose to study Ford, but instead of the full set \( G \), we now try subsets \( G' \), various combinations of the members of \( G \).

In figure 5.16 we show the result when only using the media group \( TV \) as input. The NPE value is basically the same as when using the full set \( G \), indicating that Ford and TV commercials go along well.
In Figure 5.17 (left) we have only used popular press as input. One sees that the predictive power is gone, and this is confirmed by a higher NPE value. This fact indicates that the impact from popular press is very low for Ford. From Figure 5.17 (right) we make the same conclusion for morning press.

As a matter of fact, when looping over all the different input combinations; i.e. all possible different subsets $G'$, for Ford, Audi and Volvo, we find that the subset with the lowest value for $NPE_{pred}$ contains $tv$ as an element, for all three car makers.

**LS using Gamma filter**

All the LS models are constructed by regressing on the unfiltered output (Top of Mind or In Mind). Both the prediction and the model output are then filtered, with a moving average filter, afterwards.

**Ford:** We first constructed a LS model for Ford using all available variables (a total of 7). This gave $NPE = 0.52$ for normalized Top of Mind, and $NPE = 0.67$ for In Mind, over the test set $\Omega_{test}$. The same numbers for the training set $\Omega_{train}$ were 0.38 and 0.38, respectively.

We then tried backward elimination, i.e. removing the variables one by one, always removing the one that resulted in the maximum improvement. This resulted in the following set of “optimal” four variables for Ford: $TV$, evening press, popular press, specialist press. They yield $NPE = 0.48$ for normalized Top of Mind, and $NPE = 0.51$ for In Mind, over the test set $\Omega_{test}$. The numbers for the training set $\Omega_{train}$ were 0.45 and 0.45, respectively. The model’s output over training and test sets are shown in Figure 5.18.

Transforming the variables using a transformation of the general form (5.13) gives models with the following performance: $NPE = 0.53$ for normalized Top of Mind, and $NPE = 0.50$ for In Mind (test data) when all variables are used. Backward elimination gives $TV$, morning press, popular press as the “optimal” variable set, and the NPE scores are 0.45 on normalized Top of Mind and 0.37 on In Mind (test set). The result using these variables (transformed) is shown in Figure 5.19.

*Note:* The performance using only investment in TV commercials as input is essentially as good as the “optimal” variables.

**Audi:** We also constructed a model using all inputs for Audi. This gave $NPE = 3.86$ for normalized Top of Mind, and $NPE = 2.07$ for In Mind, over the test set $\Omega_{test}$. The same numbers for the training set $\Omega_{train}$ were 0.82 and 0.75, respectively.

Backward elimination resulted in the “optimal” variable $TV$ for Audi. Using only investment in TV commercials as input gave $NPE = 0.84$ for normalized Top of Mind, and $NPE = 0.88$ for In Mind (test data). The result is shown in Figure 5.20

Forward selection gave the same result.

Transforming the variables a la (5.13) gives a model with $NPE = 1.37$ for normalized Top of Mind, and $NPE = 0.79$ for In Mind, when all input variables are used. Backwards elimination leads to a model with only one variable, investment in TV commercial, and the NPE values 0.71 and 0.78 for normalized Top of Mind and In Mind, respectively. The results using only TV investments is shown in Figure 5.21.

**All car brands:** We tried optimizing the variables with respect to the median test NPE for Top of Mind over all car brands in $C_{>150}$. This experiment would reveal investment variables that are generally informative, i.e. not only important for a specific car brand name but also for other car brand names. Our conclusion from this experiment was that the median $NPE_{test}$ does not change much when variables are removed. However, the spread (especially upwards) decreases significantly. The best overall model is achieved when only TV commercial investment is used. The result is shown in Figure 5.22.

The same is true if we transform the variables, which is shown in Figure 5.23.
Typically, the test NPE for normalized Top of Mind for a large investor when using transformed variables will be about 0.7. However, the spread is large with values as low as 0.4 and as high as above 1.0.

**Linear models across car brands**

One interesting question that we see is if it is possible to construct a model based on a few car brands and generalize this model to a new brand name. That is, is the effect of commercials on Top of Mind brand specific or is it a generic effect?

We have built models on a subset of \( C_{>150} \) and then tested on another car brand in \( C_{>150} \) (so-called cross validation). Our conclusion is that only TV commercials have a general effect on Top of Mind (or In Mind for that matter).

The typical test NPE value is 0.7. Again, the spread is considerable.

It seems possible to use only TV commercial investment as input, and train on any set of car brands, and the resulting model will on average be as good as a model constructed for that specific brand name. The typical NPE for normalized Top of Mind will be 0.7.

For some individual brands, like Ford, it seems possible to make a better model than the generic TV-model, if one trains the model on brand specific data.
Figure 5.10: Car maker patriotism: Time series for normalized levels of investment (solid lines) and Top of Mind (dashed lines). Top left: Car brands from Asia (Toyota, Honda, Hyundai, ...). Top right: Car brands from Germany (Mercedes, BMW, Volkswagen, Audi). Bottom right: Swedish car brands (Volvo and Saab). Bottom left: Other brand names (Chrysler, Jaguar, ...). Swedish car makers get the best Top of Mind returns for invested money, followed by German car makers.
Figure 5.11: Left: Results for the parameters in the benchmark reference (section 5.6.3): NPE_{train}=0.36, NPE_{test}=0.48. Right: Results when normalizing the investment over media group only: NPE_{train}=0.33, NPE_{test}=0.62. (Solid lines are true values, and dashed lines are predicted values.)

Figure 5.12: Left: Filtering switched off; NPE_{train}=0.77 and NPE_{test}=0.86. (Solid line: real data; dashed line: modelled data.) Right: NPE_{test} as a function of the filter length N. Note that the decrease of NPE_{test} is mostly due to the decrease in noise content in the output.
Figure 5.13: Left: No time lag; $\text{NPE}_{\text{train}}=0.67$, $\text{NPE}_{\text{test}}=0.69$. (Solid line: real data; dashed line: modelled data.) Right: $\text{NPE}_{\text{test}}$ as a function of the time lag $t$.

Figure 5.14: Results for Audi (left) and Volvo (right) with parameters otherwise as in the benchmark reference; Audi: $\text{NPE}_{\text{train}}=0.60$, $\text{NPE}_{\text{test}}=0.64$; Volvo: $\text{NPE}_{\text{train}}=0.37$, $\text{NPE}_{\text{test}}=0.92$. (Solid lines: real data; dashed lines: modelled data.)
Figure 5.15: Results for BMW (left) and Alfa Romeo (right) with parameters otherwise as in the benchmark reference: BMW: NPE\textsubscript{train}=0.90, NPE\textsubscript{test}=1.00; Alfa Romeo: NPE\textsubscript{train}=0.94, NPE\textsubscript{test}=1.10. (Solid lines: real data; dashed lines: modelled data.)

Figure 5.16: The result for Ford when using only TV as input: NPE\textsubscript{train}=0.39, NPE\textsubscript{test}=0.45
Figure 5.17: Left: The result for Ford when using only *popular press* as input: NPE$_{\text{train}}$=1.00, NPE$_{\text{test}}$=1.03. Right: The result for Ford when using only *morning press* as input; NPE$_{\text{train}}$=1.00, NPE$_{\text{test}}$=1.00 (Solid lines: real data; dashed lines: modelled data.)

Figure 5.18: Result when using the “optimal” set of variables for Ford (optimal with respect to test set NPE). The optimal set is *TV, evening press, popular press, specialist press*. The Top of Mind NPE values are: NPE$_{\text{train}}$ = 0.45 and NPE$_{\text{test}}$ = 0.48. (Solid lines are true values, and dashed lines are predicted values.)
Figure 5.19: Result when using the “optimal” set of transformed variables for Ford (optimal with respect to test set NPE). The optimal set is TV, morning press, popular press. The Top of Mind NPE values are: $NPE_{\text{train}} = 0.45$ and $NPE_{\text{test}} = 0.37$. (Solid lines are true values, and dashed lines are predicted values.)

Figure 5.20: Result when using the “optimal” set of variables for Audi (optimal with respect to test set NPE). The optimal set is TV. The Top of Mind NPE values are: $NPE_{\text{train}} = 0.84$ and $NPE_{\text{test}} = 0.88$. (Solid lines are true values, and dashed lines are predicted values.)
Figure 5.21: Result when using the “optimal” set of transformed variables for Ford (optimal with respect to test set NPE). The optimal set is TV. The Top of Mind NPE values are: $\text{NPE}_{\text{train}} = 0.71$ and $\text{NPE}_{\text{test}} = 0.78$. (Solid lines are true values, and dashed lines are predicted values.)

Figure 5.22: Backwards elimination for all models. The median $\text{NPE}_{\text{test}}$ does not change much when variables are removed, but the spread in values (over brands) decreases significantly.
Figure 5.23: Backwards elimination for all models, when the variables have been “Box-Cox” transformed. The median $\text{NPE}_{\text{test}}$ does decrease when variables are removed, and the spread in values (over brands) decreases significantly. However, the spread is larger in the end than what we find for untransformed variables.
5.7 Results
Anders Holst

This is a summary of the results of the different methods on a “blind test”, where the correct answer had not been given to the participants in advance.

<table>
<thead>
<tr>
<th>Method</th>
<th>Audi</th>
<th>Ford</th>
<th>Mazda</th>
<th>Nissan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ToM</td>
<td>IM</td>
<td>ToM</td>
<td>IM</td>
</tr>
<tr>
<td>SICS 1</td>
<td>-0.27</td>
<td>-0.17</td>
<td>0.29</td>
<td>0.58</td>
</tr>
<tr>
<td>SICS 2</td>
<td>-0.34</td>
<td>-0.31</td>
<td>0.23</td>
<td>0.49</td>
</tr>
<tr>
<td>DSV</td>
<td>0.11</td>
<td>-0.21</td>
<td>0.18</td>
<td>0.36</td>
</tr>
<tr>
<td>MittH</td>
<td>0.10</td>
<td>-0.17</td>
<td>0.36</td>
<td>0.49</td>
</tr>
<tr>
<td>Skövde</td>
<td>-0.13</td>
<td>-0.10</td>
<td>-0.57</td>
<td>-0.09</td>
</tr>
<tr>
<td>NMA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Volvo</th>
<th>Dressman</th>
<th>H&amp;M</th>
<th>KappAhl</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ToM</td>
<td>IM</td>
<td>ToM</td>
<td>IM</td>
</tr>
<tr>
<td>SICS 1</td>
<td>0.32</td>
<td>0.52</td>
<td>0.03</td>
<td>0.31</td>
</tr>
<tr>
<td>SICS 2</td>
<td>0.22</td>
<td>0.40</td>
<td>0.00</td>
<td>0.30</td>
</tr>
<tr>
<td>DSV</td>
<td>0.00</td>
<td>0.24</td>
<td>-0.11</td>
<td>0.34</td>
</tr>
<tr>
<td>MittH</td>
<td>0.64</td>
<td>0.40</td>
<td>0.04</td>
<td>0.30</td>
</tr>
<tr>
<td>Skövde</td>
<td>0.63</td>
<td>0.56</td>
<td>0.17</td>
<td>0.31</td>
</tr>
<tr>
<td>NMA</td>
<td></td>
<td>0.18</td>
<td></td>
<td>0.64</td>
</tr>
</tbody>
</table>

Table 5.1: The correlation coefficients between the predictions and the true series when the different methods are applied to the different brands.

Most correlations in the table are really low, or even negative indicating that there is a high level of noise, or being correct or wrong by “coincidence” rather than skill. When counting the number of “highest” rates each method gets, they all get a couple each, except SICS 1 which gets six best results (one of which is shared with SICS 2). Also, the method used by NMA seems to produce lower correlations than most of the other methods on many brands (except Dressman where everyone else made worse). Everyone gets a negative correlation on Audi, which may signal some change in advertising behavior. Everyone also get quite high correlation on Mazda, In Mind. Whether this is pure chance, or would be maintained in future tests is hard to tell.

In summary, the results suggest that NMA could gain by introducing some of the techniques above for their brand awareness analysis.
Chapter 6

NovaCast: Prediction of alloy parameters

6.1 Introduction

Daniel Gillblad

NovaCast produces software for the foundry industry, including tools for solidification simulation and thermal analysis. These tools are used for metallurgical process control to achieve less casting defects, a high yield and reduced costs.

Thermal analysis studies the solidification from liquid metal into solid iron or an alloy. It is based on recording temperatures at certain time intervals during the solidification progress, and from them constructing a cooling curve. The cooling curve is essentially a plot of the temperature of the metal as a function of time.

By thermal analysis of a sample from the furnace, it is possible to extract information that can be used to predict properties of the alloy produced by the contents of the furnace. In the thermal analysis, a sample from the furnace is cooled and the cooling curve is recorded. Several parameters can then be extracted from the curve, describing important properties of the cooling process. The parameters include, among other things, plateau temperatures and cooling rates in the different states. Using this, information about when different state transitions occur in the furnace can be extracted, which in turn gives an opportunity to predict properties such as chemical composition and final quality of the alloy.

Predicting properties of the final alloy from a sample taken from the furnace is an important task. The ability to make such predictions reliably could potentially help in the reduction of scrap material and defects in the foundry. There are two completely separate NovaCast data sets. For the first data set, the task was to estimate the number of graphite nodules per \( \text{mm}^2 \). For the second data set, which contains more variables and several alloy properties, the main focus of the work within the DALLAS project has been on predicting one of these properties, the oxygen content.

6.1.1 The first NovaCast data set

The first NovaCast dataset consists of measurements from 96 different furnace samples. For each sample, the following attributes are available:

1. TL, Liquidus temperature in the cooling curve.
2. TES, Start eutectic cooling.
3. TEU, Lower eutectic temperature.
### Table 6.1: The attributes included in all measurements except the tellurium cup.

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TL</td>
<td>Liquidus temperature in the cooling curve</td>
<td></td>
</tr>
<tr>
<td>TES</td>
<td>Start eutectic cooling</td>
<td></td>
</tr>
<tr>
<td>dT/dTES</td>
<td>Cooling rate at the start of eutectic solidification</td>
<td></td>
</tr>
<tr>
<td>TE Low</td>
<td>Lower eutectic temperature</td>
<td></td>
</tr>
<tr>
<td>TE High</td>
<td>Upper eutectic temperature</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Recalescence</td>
<td></td>
</tr>
<tr>
<td>Max Rate</td>
<td>Max R rate</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>Temperature 1</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>Temperature 2</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>Temperature 3</td>
<td></td>
</tr>
<tr>
<td>TL Plata</td>
<td>Liquidus temperature plateau</td>
<td>Only in ladle data</td>
</tr>
<tr>
<td>dT/dtTS</td>
<td>Cooling rate at the solidus temperature</td>
<td></td>
</tr>
<tr>
<td>TS</td>
<td>Solidus temperature</td>
<td></td>
</tr>
<tr>
<td>GRF2</td>
<td>Grafit factor 2</td>
<td></td>
</tr>
</tbody>
</table>

4. TEH, Upper eutectic temperature.
5. GRF1, Grafite factor 1.
6. dT/dt, Cooling rate at the solidus temperature.
7. GRF2, Grafite factor 1.

There are two attributes that are interesting to predict:

8. MICROSHR, Micro suction tendency.
9. NOD_COUNT, The nodule count, the number of graphite nodules per \textit{mm}^2.

All input variables are continuous. Of the output variables, the nodule count is a continuous variable and the micro suction tendency categorical.

### 6.1.2 The second NovaCast data set

The second dataset consists of measurements from two different places in the process, denoted furnace data and ladle data. The furnace data set contain 45 samples, and the ladle data 46 samples. The datasets have been treated as completely separate. Although both data sets contain roughly the same attributes, they must be regarded as behaving significantly different from each other. There are six different kinds of measurements in both data sets:

1. Grey unioculated
2. 12mm cup
3. Grey inoculated
4. Tellurium cup
5. Second grey unioculated
6. Second 12mm cup
Name | Explanation
--- | ---
TL | Liquidus temperature in the cooling curve
TES | Eutectic temperature

**Table 6.2:** The attributes included in the tellurium cup.

The first and second 12mm cups and the first and second grey unioculated measurements are duplicate samples, and should be highly correlated. All of the different kinds of measurements except the tellurium cup include the attributes listed in table 6.1 along with short explanations of some of them. The tellurium cup measurements contains just 2 attributes. These attributes are listed in table 6.1. All the attributes are continuous.

The data also contains thirty attributes that might be interesting to predict. All the these possible output attributes are continuous, and describe for example chemical composition and hardness. The most important output attribute to predict is the oxygen content.
6.2 The model used at Halmstad University
Thorsteinn Rögnvaldsson

6.2.1 Data, preprocessing, and variable selection

The first NovaCast data set had two parts, of which we only considered one, namely the regression problem of estimating the number of graphite nodules per $mm^2$. The data set is very limited (few variables and few observations).

For the first data set, NovaCast were interested in what variables that are important for the prediction, how good predictions can be made, and if there were dependencies among the variables.

The data set consists of 96 observations, with 7 possible input variables for each observation, and two outputs, of which we only focus on one. Of these, 11 were set aside for a blind out-of-sample test after the model construction, and the remaining 85 were used for training.

6.2.2 Preprocessing

We first studied how the input variables varied with the output, which is shown in Figure 6.1. From this we decided to invert variable 7, which improved the correlation with the output. A cross correlation study, see Figure 6.2, showed that variables 1 and 2 were strongly linearly correlated, as well as variable 6 with the inverse of variable 7. this could mean that one of the variables should be removed or they should be replaced by composite variables. However, the exhaustive search variable selection method we used (see below) meant that we did not have to care too much about this.

We also looked for outliers in the data, but did not see any observations that were out of the ordinary.

The variables were preprocessed in the following way

1. The seventh variable was inverted (i.e. $x_7 \rightarrow 1/x_7$).
2. All variables were normalized to zero mean and unit standard deviation.
Figure 6.2: Scatter plots of the input variables versus each other. Variables 1 and 2 are positively linearly correlated. Variable 6 and the inverse of variable 7 are negatively linearly correlated. The plots along the diagonal are normal probability plots for the variable in question.

6.2.3 Variable selection

The number of variables was very small and the number of observations was also very small. We therefore tried all possible combinations of input variables to the simple models (i.e. those that were quick to construct). The total number of possibilities is given by

\[ N_{\text{poss}} = \sum_{k=1}^{K} \binom{K}{k} \]

which equals 127 for \( K = 7 \). Each variable set is evaluated by computing the 5-fold cross validation error for the model.

In the multilayer perceptron case, were the training time is considerable, we instead selected variables by backwards elimination. That is, we started with using all 7 variables and then removed the variable that lead to the largest decrease (or smallest increase) of the cross validation error.

6.2.4 Constructing the regression models

We tried three different regressors: Linear regression, \( k \)-nearest neighbors (\( k \)NN) regression, and the multilayer perceptron (MLP).

The linear regressor was constructed by using the pseudoinverse method. Each linear model, defined by the input variable set, was evaluated using the 5-fold cross validation error (i.e. \( 127 \times 5 = 635 \) linear models were constructed and tested).

The \( k \)NN regressor is constructed by trying \( k = \{1, 2, 3, 4, 5, 6, 7, 8\} \) and all possible combinations of input variables. Each combination of \( k \) and input variable set was evaluated using the 5-fold cross validation error (i.e. a total of \( 127 \times 8 \times 5 = 3810 \) models were constructed and tested). Euclidean metric was used throughout for all \( k \)NN models.
The MLP regressor was trained using the Levenberg-Marquardt optimization and early stopping. We used 3/5 of the data for computing the gradient, 1/5 of the data for determining the stopping point, and 1/5 of the data for validating after training. The number of hidden units was varied between 3 and 12. The five best models trained for the best variable set were combined into an averaging committee (the committee members have the same number of hidden units, and the same input variables). In summary, we trained $7 \times 10 \times 5 = 350$ MLP models and tested them.

6.2.5 Results

The best linear model turned out to be a model with all seven input variables. The cross validation mean square error (CV-MSE) for this model was $285 \pm 95$ (the latter number is the cross validation estimated standard deviation). The best $k$NN model used only variables 7 and 2, with $k = 1$ (i.e. only one nearest neighbor). The CV-MSE for this model was $4.2 \pm 0.9$. A $k$NN model with $k = 1$ is equivalent to a look up table with no interpolation. The best MLP model used seven hidden units and variables 5, 6, and 7. The CV-MSE for this model was $4.7 \pm 2.5$. An MLP using all input variables and seven hidden units gives CV-MSE = $8.3 \pm 13$, for comparison.

These models were then tested on the hold-out test data set. The result is summarized in Tables 6.3 and 6.4. The outputs from each model is plotted versus the true output in Figure 6.3.

<table>
<thead>
<tr>
<th>Model</th>
<th>MSE</th>
<th>RMS</th>
<th>RMS%</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear (all variables)</td>
<td>140.5</td>
<td>11.8</td>
<td>0.4%</td>
<td>0.9900</td>
</tr>
<tr>
<td>$k$NN ($k = 1$, 2 variables)</td>
<td>2.7</td>
<td>1.7</td>
<td>0.008%</td>
<td>0.9998</td>
</tr>
<tr>
<td>MLP (7 hidden, 3 variables)</td>
<td>2.0</td>
<td>1.4</td>
<td>0.007%</td>
<td>0.9999</td>
</tr>
</tbody>
</table>

Table 6.3: Summary of results on the hold-out test set (11 samples). The RMS% column shows how many percent wrong the model is on average. The $\rho$ column shows the correlation coefficient (Pearson's correlation coefficient) for the prediction versus the true value. The precision of the numbers sets a theoretical lower limit for MSE at 0.25.

<table>
<thead>
<tr>
<th>Model</th>
<th>Linear</th>
<th>$k$NN</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$k$NN</td>
<td>+</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>MLP</td>
<td>+</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.4: The significance of the results, tested using the Mann-Whitney test. The test tests whether the residuals from two models come from the same distribution. The linear model is significantly poorer than both the $k$NN and MLP models, at the 95% significance level. The $k$NN and MLP results are not significantly different, at a 95% significance level.

6.2.6 Conclusion

The results illustrates that a $k$NN model is often a strong competitor to methods like MLP. The $k$NN model is both simple to construct, easily modified, nonparametric, and nonlinear. It is often a good idea to use a $k$NN model as benchmark for another nonlinear method, $k$NN being a well established statistical method with a very appealing simplicity.

In this case the MLP gave slightly better results (not significant though) on the hold-out test set. However, the results on the cross validation data during the training phase indicated that the $k$NN method should have been the better one of the two.

The input data had strong correlations, which indicated that partial least squares (PLS) would have been a better linear benchmark than simple least squares.

A bigger data set is needed to really compare models and test whether the important variables have really been extracted.
6.2 The model used at Halmstad University

Figure 6.3: Scatter plots for the predicted output versus the true output for the hold-out test data set.
6.3 The model used at SICS
Daniel Gillblad

6.3.1 Oxygen content prediction using a mixture model

A natural approach to predict a continuous parameter, in this case the oxygen content, is to create a mixture of Gaussians over the whole input space and the output space. The mixture model parameters, such as the means and the covariance matrices of the Gaussians are estimated from the training data set, using for example expectation maximization (EM). The marginal of the output space given an input pattern can then be calculated. When we know the marginal distribution of the output variables, the expectation or the maximum of this distribution can be used as a prediction, depending on whether we want to minimize the mean square error or the absolute error.

The performance of a mixture model over all input attributes and the output attribute were tested. All the tests used the furnace data set, and all possible input attributes were used, a total of 68. The number of available training samples were 44. The expectation of the resulting marginal distribution for a test pattern was used as the prediction. Table 6.5 a and b show the test results, table a showing the results when the model was tested on training data and table b with leave one out cross-validation. With leave one out cross-validation, a model is estimated from all entries in the data except one, which the model is tested on. This is done for all patterns in the data set. The first column in the tables shows the number of Gaussians used in the mixture. The second column the resulting root mean square error (RMS), and the third and fourth column show the fraction of patterns that are within one and three standard deviations of the predicted pattern. The third and fourth column can be viewed as a measure of how many of the predictions that are, in one sense, reasonable. The standard deviation of the oxygen content is 0.26.

The results are reasonably good, both on training and test data. The mean square error is rather low, at least for some of the tested models. It is obvious from the test results that while using only one Gaussian, the mean square error is rather high. On the other hand, for the results with cross-validation, the number of patterns within one standard deviation is high, suggesting that a simple linear predictor might be sufficient to produce good results if this is considered to be the most important property. Increasing the number of Gaussians though leads to lower mean square errors, and the number of predictions that fall within three standard deviations rise up to 100 lower. This probably makes for a more practically useful prediction.

Note that the results both on the training set and with cross-validation are not completely consistent, in the respect that the quality of the results do not follow the number of Gaussians in a very organized way. This is due to random effects. When generating the prediction model, the initial model that is trained is generated at random with some considerations to the data.

6.3.2 Dependency structure analysis

To gain insight and knowledge of a data set, a dependency structure analysis can be very useful. The dependency graph will show the dependencies between attributes, and what attributes that affects the prediction the most. Even if not used for predictions or some other application, the creation of a dependency graph can still be very valuable for getting a feel for the relationships in the data.

When constructing the dependency graph, we need to keep in mind what we want to use it for. By calculating all pairwise correlations between attributes and then showing the strongest ones in a dependency graph, we might get useful information about the general dependency structure and what attributes that might be redundant. On the other hand, dependencies that might be interesting for a certain task might not be visible using this approach. Often the strongest dependencies in data are between similar input attributes, not between input attributes and the output attribute we are interested in. If we want to visualize the dependencies to a specific output attribute, we must use another approach.

The dependency graph in figure 6.4 was generated keeping the relevant output attribute, the oxygen content, in mind. All pairwise linear correlations, i.e. the correlation coefficients, between attributes
4 a. Training and testing on all data

<table>
<thead>
<tr>
<th>Number of Gaussians</th>
<th>Root mean square error (RMS)</th>
<th>Within one standard deviation</th>
<th>Within three standard deviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.35</td>
<td>70.5%</td>
<td>88.6%</td>
</tr>
<tr>
<td>2</td>
<td>0.20</td>
<td>77.2%</td>
<td>100.0%</td>
</tr>
<tr>
<td>3</td>
<td>0.23</td>
<td>86.4%</td>
<td>100.0%</td>
</tr>
<tr>
<td>4</td>
<td>0.20</td>
<td>77.2%</td>
<td>100.0%</td>
</tr>
<tr>
<td>5</td>
<td>0.17</td>
<td>86.4%</td>
<td>100.0%</td>
</tr>
<tr>
<td>6</td>
<td>0.14</td>
<td>88.6%</td>
<td>100.0%</td>
</tr>
<tr>
<td>7</td>
<td>0.18</td>
<td>77.2%</td>
<td>100.0%</td>
</tr>
<tr>
<td>8</td>
<td>0.51</td>
<td>81.8%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

Table 6.5: Oxygen content prediction results on furnace data

4 b. Leave one out cross-validation

<table>
<thead>
<tr>
<th>Number of Gaussians</th>
<th>Root mean square error (RMS)</th>
<th>Within one standard deviation</th>
<th>Within three standard deviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.57</td>
<td>77.2%</td>
<td>84.1%</td>
</tr>
<tr>
<td>2</td>
<td>0.35</td>
<td>52.2%</td>
<td>95.5%</td>
</tr>
<tr>
<td>3</td>
<td>0.32</td>
<td>59.1%</td>
<td>95.5%</td>
</tr>
<tr>
<td>4</td>
<td>0.40</td>
<td>86.4%</td>
<td>95.5%</td>
</tr>
<tr>
<td>5</td>
<td>0.32</td>
<td>65.9%</td>
<td>95.5%</td>
</tr>
<tr>
<td>6</td>
<td>0.30</td>
<td>50.0%</td>
<td>97.7%</td>
</tr>
<tr>
<td>7</td>
<td>0.26</td>
<td>61.4%</td>
<td>100.0%</td>
</tr>
<tr>
<td>8</td>
<td>0.27</td>
<td>61.4%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

in the ladle data were calculated. Then the strongest correlations between input attributes and the oxygen content were selected, as well as all the stronger correlations between these input attributes. All other correlations were discarded. The graph in the figure was then generated by running a greedy tree construction algorithm on the selected dependencies. The attributes in the graph are denoted with the attribute name and a number in parenthesis describing from what kind of measurement the attribute belongs to (see section 6.1.2). When studying figure 6.4, keep in mind that the oxygen content has a strong correlation to all the input attributes in the graph. The tree structure shown is most of all an aid to understand the relationship between the input variables.

Two general observations can be made by examining figure 6.4. First, most input attributes shown belong to either Grey unioculated or Second grey unioculated. This means that these measurements might contain more useful information about the oxygen content than the other measurements. To be fair though, the grey inoculated is also rather common while the tellurium cup and 12mm cup is barely represented. Second, most attributes are temperatures or the maximum or minimum value of temperatures. This is perhaps not surprising since most of the inputs in fact represent different temperatures, but for example TEHigh and TELow are clearly over represented, indicating that these attributes might be important for the oxygen content.

Also note that after the tree generation, the oxygen content ended up as a leaf, with only one connection. This is a result of the fact that the dependencies between the input attributes are generally stronger than to the oxygen content.

On the whole, though, the dependency graph must be looked upon with some scepticism. The number of examples are low, only 44, while at the same time the data must be considered to be fairly noisy. This means that reliable correlation estimation might be hard, and is also the reason why the simple correlation coefficient was used instead of some more advanced measure more prone to suffer from the low number of examples available. Also the tree generation algorithm used is very sensitive to random effects and noise in the correlation estimations, but still it can provide some useful information as a suggestion of what the dependency structure might look like.
6.3.3 Comments and conclusions

The predictions produced by the very straightforward method of using a Gaussian mixture model over the whole input and output space are promising. The results are reasonably good, and there is probably room for significant improvement using similar but more specialized models. The data set is very small, though, and it is hard to tell whether this dataset correctly reflects all the properties of the data.
Chapter 7

SCA: The dewatering task

7.1 Introduction
Anders Holst

A main concern for SCA is to decrease the number of paper breaks in a paper machine. Since this was considered a too complex problem for the Dallas project, a partial task of this was selected: predicting the “dewatering” in the process, i.e. how much of the water has left the pulp. Another issue — how various parameters of the paper machine affects the number of paper breaks — were saved as a secondary alternative in case time would allow it.

From the paper machine, PM5 in Ortviken in Sundsvall, we received data from 137 different variables, sampled once a minute during one month. The machine underwent a major service during five days in the middle of that period, from which time there are no meaningful data. The data before and after this service is also slightly different from each other. Because of this, most analysis has been done separately on the period before the service and the period after it. In detail, the first period lasted from 08:00 the 2/5-2000 until 02:08 the 15/5-2000 (in total 18369 minutes), and the second period from 19:04 the 19/5-2000 until 14:00 the 31/5-2000 (in total 16976 minutes).

Among the data received, there is one variable for the speed of the tambour reel, one variable for the steam flow to the dryer section meant to indicate whether the paper is broken or not, and a variable for the dewatering which was measured by a special equipment during the period. Unfortunately that equipment only delivered data approximately half of the time. Also there are many other variables in the data that are missing at certain times, or indicating sensor conditions like “overflow”, “underflow”, or “shutdown”. These values had to be handled in some way, either substituting some other value for them, or considering them as unknown.

The complete set of variables in the data set is shown in Table 7.1.
<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Index</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Produktkod Lab PM5</td>
<td>71</td>
<td>LinjetryckPress3</td>
</tr>
<tr>
<td>2</td>
<td>K46 PQM Freeness</td>
<td>72</td>
<td>Tryck zon1 3:e Pr</td>
</tr>
<tr>
<td>3</td>
<td>K46 PQM Fiberlängd</td>
<td>73</td>
<td>Tryck zon2 3:e Pr</td>
</tr>
<tr>
<td>4</td>
<td>K46 PQM Fiber 0,0-0,</td>
<td>74</td>
<td>Tryck zon3 3:e Pr</td>
</tr>
<tr>
<td>5</td>
<td>K46 PQM Fiber 0,0-0,</td>
<td>75</td>
<td>Tryck zon4 3:e Pr</td>
</tr>
<tr>
<td>6</td>
<td>K46 PQM Fiber 0,5-1,</td>
<td>76</td>
<td>Tryck zon5 3:e Pr</td>
</tr>
<tr>
<td>7</td>
<td>K46 PQM Fiber 1,0-3,</td>
<td>77</td>
<td>Ecoflow Viratotal</td>
</tr>
<tr>
<td>8</td>
<td>K46 PQM Fiber 3,0-7,</td>
<td>78</td>
<td>Ecoflow s:a 1:a total</td>
</tr>
<tr>
<td>9</td>
<td>Temp efter K-904</td>
<td>79</td>
<td>Ecoflow s:a 3:e total</td>
</tr>
<tr>
<td>10</td>
<td>Tmp-fiberflöde %</td>
<td>80</td>
<td>Ecoflow s:a P/U total</td>
</tr>
<tr>
<td>11</td>
<td>Ledn.form. TMP-kar PM5</td>
<td>81</td>
<td>Ecoflow s:a total</td>
</tr>
<tr>
<td>12</td>
<td>MaldTMP PQM Freeness</td>
<td>82</td>
<td>Avvattning 1:a press nyp</td>
</tr>
<tr>
<td>13</td>
<td>MaldTMP Fiber 0,0 - 0,2mm</td>
<td>83</td>
<td>2:a press nyp</td>
</tr>
<tr>
<td>14</td>
<td>MaldTMP Fiber 0,2 - 0,5mm</td>
<td>84</td>
<td>3:e press nyp</td>
</tr>
<tr>
<td>15</td>
<td>MaldTMP Fiber 0,5 - 1,0m</td>
<td>85</td>
<td>Fl-trägv.fr.press,</td>
</tr>
<tr>
<td>16</td>
<td>MaldTMP Fiber 1,0 - 3,0mm</td>
<td>86</td>
<td>Vatten fr.3 presse</td>
</tr>
<tr>
<td>17</td>
<td>MaldTMP Fiber 3,0 - 7,0mm</td>
<td>87</td>
<td>Tr.sprits pu-filt</td>
</tr>
<tr>
<td>18</td>
<td>Sulfatfiberflöde %</td>
<td>88</td>
<td>Tr.sprits 1:pilt</td>
</tr>
<tr>
<td>19</td>
<td>Freen Sulfat News f. uppl</td>
<td>89</td>
<td>Tr.sprits 3:pfilt</td>
</tr>
<tr>
<td>20</td>
<td>MFL Sulfat News f. uppl</td>
<td>90</td>
<td>Tr.sprits 3:pfilt</td>
</tr>
<tr>
<td>21</td>
<td>Utskottsfib. flöde %</td>
<td>91</td>
<td>Falskl.p-u sugl.övre</td>
</tr>
<tr>
<td>22</td>
<td>Freeness utsk.kar PM5</td>
<td>92</td>
<td>Falskl.p-u sugl.undre</td>
</tr>
<tr>
<td>23</td>
<td>Maskinkar Fiber 0,0 - 0,2m</td>
<td>93</td>
<td>Falskl.3:e sugl.övre</td>
</tr>
<tr>
<td>24</td>
<td>Maskinkar Fiber 0,2 - 0,5m</td>
<td>94</td>
<td>Falskl.1:a sugl.övre</td>
</tr>
<tr>
<td>25</td>
<td>Maskinkar Fiber 0,5 - 1,0m</td>
<td>95</td>
<td>Falskl.1:a sugl.undre</td>
</tr>
<tr>
<td>26</td>
<td>Maskinkar Fiber 1,0 - 3,0m</td>
<td>96</td>
<td>Maskinkar Fiber 1,0 - 3,0m</td>
</tr>
<tr>
<td>27</td>
<td>Maskinkar Fiber 3,0 - 7,0m</td>
<td>97</td>
<td>Maskinkar Fiber 3,0 - 7,0m</td>
</tr>
<tr>
<td>28</td>
<td>Maskinkar PQM Freeness</td>
<td>98</td>
<td>Temp Anga t Anglåda</td>
</tr>
<tr>
<td>29</td>
<td>Lufthalt eft. Maskinkar</td>
<td>99</td>
<td>1:a pr.riktv.sch.bel</td>
</tr>
<tr>
<td>30</td>
<td>Bentionit</td>
<td>100</td>
<td>2:a Pr.sch.bel</td>
</tr>
<tr>
<td>31</td>
<td>Polymer</td>
<td>101</td>
<td>Centrv. sch.bel</td>
</tr>
<tr>
<td>32</td>
<td>Vak.avlufningstank</td>
<td>102</td>
<td>Luft fr 1:a press filt</td>
</tr>
<tr>
<td>33</td>
<td>Total tryck Inloppsl.</td>
<td>103</td>
<td>Hastighet Pope</td>
</tr>
<tr>
<td>34</td>
<td>Utloppskvot</td>
<td>104</td>
<td>Porositet,P PM5</td>
</tr>
<tr>
<td>35</td>
<td>Diff.tr. Rercirkulat.</td>
<td>105</td>
<td>Porositet,Labprofil PM 5, 01</td>
</tr>
<tr>
<td>36</td>
<td>Konc.inloppslåda</td>
<td>106</td>
<td>Porositet,Labprofil PM 5, 02</td>
</tr>
<tr>
<td>37</td>
<td>Konc Inloppslåda PM5</td>
<td>107</td>
<td>Porositet,Labprofil PM 5, 03</td>
</tr>
<tr>
<td>38</td>
<td>Fines Inloppslåda PM5</td>
<td>108</td>
<td>Porositet,Labprofil PM 5, 04</td>
</tr>
<tr>
<td>39</td>
<td>MFL Inloppslåda PM5</td>
<td>109</td>
<td>Porositet,Labprofil PM 5, 05</td>
</tr>
<tr>
<td>40</td>
<td>pH Inloppslåda PM5</td>
<td>110</td>
<td>Porositet,Labprofil PM 5, 06</td>
</tr>
<tr>
<td>41</td>
<td>PH före Inloppslåda</td>
<td>111</td>
<td>Porositet,Labprofil PM 5, 07</td>
</tr>
<tr>
<td>42</td>
<td>Ledn.form. inl.låda PM5</td>
<td>112</td>
<td>Anisotropi D Flab PM5</td>
</tr>
<tr>
<td>43</td>
<td>COD-halt Inloppslåda PM5</td>
<td>113</td>
<td>Formation PM5</td>
</tr>
<tr>
<td>44</td>
<td>Laddningsmätare PM5</td>
<td>114</td>
<td>Tjocklek,P PM5</td>
</tr>
<tr>
<td>45</td>
<td>Retention fiber PM5</td>
<td>115</td>
<td>Densitet PM5</td>
</tr>
<tr>
<td>46</td>
<td>Konc.virabakvatten</td>
<td>116</td>
<td>Ytvikt pope PM5</td>
</tr>
<tr>
<td>47</td>
<td>Läpplutning</td>
<td>117</td>
<td>Fukt pope PM5</td>
</tr>
<tr>
<td>48</td>
<td>Läppöppning</td>
<td>118</td>
<td>Tjocklek PM5</td>
</tr>
<tr>
<td>49</td>
<td>Läppuppvärming</td>
<td>119</td>
<td>Torrvikt PM5</td>
</tr>
<tr>
<td>50</td>
<td>Vakuum virusuglåda</td>
<td>120</td>
<td>Alfatex f Bl.pump</td>
</tr>
<tr>
<td>51</td>
<td>Vakuum prim.gusk lv</td>
<td>121</td>
<td>Temp.varmvattenstank</td>
</tr>
<tr>
<td>52</td>
<td>Låg vacuum planuglåda</td>
<td>122</td>
<td>Ånga till VVX kc</td>
</tr>
<tr>
<td>53</td>
<td>Hög vacuum planuglåda</td>
<td>123</td>
<td>Uttemp värmesystem</td>
</tr>
<tr>
<td>54</td>
<td>Tr.sprits vira 1</td>
<td>124</td>
<td>Temp korta cirkulation</td>
</tr>
<tr>
<td>55</td>
<td>Tr.sprits vira 2</td>
<td>125</td>
<td>Sum Svavelysa PM5</td>
</tr>
<tr>
<td>56</td>
<td>Vakuum pick-up vals</td>
<td>126</td>
<td>Konc Klarfiltr.v.pump PM5</td>
</tr>
<tr>
<td>57</td>
<td>Vakuum sugvals lv</td>
<td>127</td>
<td>Pålåggsm. skift 1</td>
</tr>
<tr>
<td>58</td>
<td>Vakuum sugvals lv</td>
<td>128</td>
<td>Varv skifilter 1</td>
</tr>
<tr>
<td>59</td>
<td>Nyp 1:a press</td>
<td>129</td>
<td>Konc Utskottslar PM5</td>
</tr>
<tr>
<td>60</td>
<td>Tryck zon1 1:a Pr</td>
<td>130</td>
<td>Härhdhet Färskv. ÅC</td>
</tr>
<tr>
<td>61</td>
<td>Tryck zon2 1:a Pr</td>
<td>131</td>
<td>KMnO4 Färskv. ÅC</td>
</tr>
<tr>
<td>62</td>
<td>Tryck zon3 1:a Pr</td>
<td>132</td>
<td>SiO2 Färskv. ÅC</td>
</tr>
<tr>
<td>63</td>
<td>Tryck zon4 1:a Pr</td>
<td>133</td>
<td>pH Färskv. ÅC</td>
</tr>
<tr>
<td>64</td>
<td>Tryck zon5 1:a Pr</td>
<td>134</td>
<td>Konduktivitet Färskv. ÅC</td>
</tr>
<tr>
<td>65</td>
<td>LinjetryckPress2</td>
<td>135</td>
<td>Klorerorsk. Farskatten in</td>
</tr>
<tr>
<td>66</td>
<td>Tryck zon1 2:a Pr</td>
<td>136</td>
<td>Ånga till PM5</td>
</tr>
<tr>
<td>67</td>
<td>Tryck zon2 2:a Pr</td>
<td>137</td>
<td>Gusktohrralt PM5</td>
</tr>
<tr>
<td>68</td>
<td>Tryck zon3 2:a Pr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>Tryck zon4 2:a Pr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>Tryck zon5 2:a Pr</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 7.1:** All the variables (names in Swedish).
7.2 The model used at SICS

Anders Holst

7.2.1 Preprocessing

At SICS we chose the following way to preprocess the data: All underflow and overflow values were changed to values respectively lower or higher than all other values of the same variable. All other non-numerical values were treated as unknown.

Test and training sets had also to be selected from data. Since the two periods differed slightly in their process characteristics, we chose to limit the primary experiments to one of the periods (since it may be hard to use data from one period to try to predict the behavior during the other period from), the first one. Also, since it is a time series which moves relatively slowly through the state space, training and test samples must be taken from sufficiently separate portions in the data, otherwise the test cases will in effect be so similar to the training cases that the results can not be generalized to new data. In the first period, values for dewatering exist in one sequence in the beginning with qualitatively very different appearance than the rest, and thereafter in three “bursts” at the second half of the period. We ignored the first sequence, since the appearance suggested that the sensor was not yet properly calibrated. Of the three following bursts the two first were used as training data (samples 8764 through 9842 and 10231 through 12284) and the third one as test data (samples 12554 through 14435).

7.2.2 Choice of method

SICS has done two different things here. Besides trying to predict the dewatering, we have also made an analysis of the dependencies in the domain. Between each variable and dewatering we have generated diagrams to show the correlation between them. One diagram shows the points of the two series plotted against each other, to show if there is any systematic relation between them. Another diagram shows the change over time in the two series plotted against each other, to show if there is any systematic relation between the direction of movement between the two series. Then there are a set of correlograms to investigate if they depend on each other with some specific delay in time. For this analysis we used the mutual information and the mutual information rate between the time series. The information measures were based on either a purely linear model of the data distribution, or on a histogram based model. This gave in total four correlograms, showing how the strong the information shared between the series is for different time delays, up to 300 minutes in both directions. This was done separately for both periods.

The difference between the histogram model and the linear model (a Gaussian distribution) is that the histogram model can potentially catch non-linear relations which the linear model can not find. On the other hand, the histogram model is more complicated and therefore more sensitive to noise and random fluctuations. The linear model on its side is sensitive to extreme values in a way that the histogram is not. If one of the series has a highest peak somewhere, and the other series also an extreme value somewhere, selecting a delay that aligns those extreme values will give a strong peak in the correlogram, almost regardless of how the series behaves relative to each other at other times. This gives a risk that the wrong delay between the series is selected due to purely random extreme values. Because of this both kind of diagrams are given, and a feature should hopefully give some evidence in both of them if it is significant.

A similar tradeoff exist between the “normal” mutual information diagrams and those showing the mutual information rate. As discussed in the method chapter, using mutual information (or normal correlation coefficient also) between time series tends to give a too high value of the correlation. This is because if the time series moves slowly enough, the relation between the series at one point in time is likely to be maintained for several time steps. This means that pure random coincidences between the series gets multiplied with some factor depending on how slow the series are, making the correlation seem more significant than it is. The mutual information rate on the other hand, which only considers the new information in every step, correctly compensates for that effect, but instead requires a more complicated model to estimate, which makes it more noisy.

In practice it turns out that the diagrams using histograms have quite little interesting variation,
whereas the linear diagrams show more features, some of which are clearly anomalous. As expected, the diagrams with information rates show much lower levels of correlations and more emphasized peaks (not so smeared out over several different delays), but also more noise in the form of random spikes in the correlograms. All in all it seems that the linear information rate is the one that gives the most reliable indications of correlations between the time series in this case. However, due to the different tradeoffs, a rule of thumb is that a feature (peak) should appear on at least two of the four diagrams to be considered significant.

The linear information rate measure was also used between all pairs of variables in the domain, and those with strongest correlation was marked in a dependency graph of the domain, showing how the variables depend on each other. The complete set of diagrams were generated for those pairs of variables that correspond to arcs in the dependency graph. All these diagrams have been delivered to SCA separately.

The conclusions that can be drawn from these diagrams is partially disappointing. There are only very weak correlations from the given variables to dewatering. Most of those correlations found are quite diffuse, with no specific time delay of the interaction, indicating that they may be mere coincidences in trends: for example when one of the series tends to go up over a long time, the dewatering may tend to go down during the same period. None of the variables seem to follow the dewatering on a faster scale. On the other hand, this indicates that SCA runs their process quite optimally (with respect to dewatering) given the available information.

This also means that there are no specific delays in the interactions with dewatering that are more significant than others. In the dependency graph between pairs of attributes other than dewatering there are some significant points in time for the interactions, but the majority of these have a delay close to zero. When trying to predict the dewatering, all time delays of the series where therefore set to zero.

### 7.2.3 Results

The models used for prediction are all based on the Naive Bayesian classifier, augmented with graphical models to compensate for time dependencies in each time series (but not for dependencies between different variables in this setting). No mixtures were used here, but all relevant distributions were modeled as single Gaussian distributions. All input variables were used (except those that were constant in the selected training and test sets). For comparison, there is also test results for the case of using on single Gaussian distribution over the entire input and output space. The results are shown in table 7.2.

The result with a joint distribution over the entire space shows, as expected, severe signs of overtraining; the results are quite good at the training data, but extremely bad on the test set. The other models, all based on the naive Bayesian classifier, but considering different number of time steps of the series, are all quite similar in their results. The achieved correlation coefficients on the test sets are not very high, but at least reasonable. However, considering more than one time step seems not to help anything in this case. The prediction curves for the first two models in the table are shown in figures 7.1 and 7.2.

A general comment is that even if the prediction is able to follow the trends in dewatering reasonably, it fails to catch the fast movements of the series.

<table>
<thead>
<tr>
<th>Description</th>
<th>Corr Train</th>
<th>Corr Test</th>
<th>RMS Train</th>
<th>RMS Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current time only, joint space</td>
<td>0.89</td>
<td>-0.56</td>
<td>0.37</td>
<td>4.59</td>
</tr>
<tr>
<td>Current time only, naive Bayes</td>
<td>0.82</td>
<td>0.54</td>
<td>0.54</td>
<td>0.72</td>
</tr>
<tr>
<td>Two time steps, 1 min apart</td>
<td>0.82</td>
<td>0.51</td>
<td>0.54</td>
<td>0.70</td>
</tr>
<tr>
<td>Two time steps, 20 min apart</td>
<td>0.83</td>
<td>0.40</td>
<td>0.53</td>
<td>0.69</td>
</tr>
<tr>
<td>Two time steps, 60 min apart</td>
<td>0.83</td>
<td>0.45</td>
<td>0.53</td>
<td>0.64</td>
</tr>
<tr>
<td>Three time steps, 1 min apart</td>
<td>0.83</td>
<td>0.39</td>
<td>0.52</td>
<td>0.63</td>
</tr>
<tr>
<td>Three time steps, 60 min apart</td>
<td>0.83</td>
<td>0.46</td>
<td>0.53</td>
<td>0.69</td>
</tr>
</tbody>
</table>

Table 7.2: The results of different models.
7.2 The model used at SICS

Figure 7.1: The prediction result of a single Gaussian over the joint space. Of the three "gray bursts" of
dewatering data in the middle and to the right in the plot, the two first were used as training set and the third
as test set. The sequence of data to the far left in the plot looks anomalous and was not used. The prediction
(dark curve) follows the average dewatering very well in the training set, but in the test set it is completely erratic
(and it fluctuates quite much in the gap between the two training sequences). This is due to the high degree of
overtraining of this type of model.

Figure 7.2: The prediction result of a naive Bayesian classifier. The training and test sets are as above. Here the
prediction follows both the test and training sequences reasonably well, i.e. in spite of using all input variables
there is no problem with overtraining for this model.
7.3 The model used at Halmstad University
Thorsteinn Rögnvaldsson

7.3.1 Introduction

The SCA problem consisted in modeling the humidity of the paper as a function of different process variables. The goal was to see if it was possible to estimate the humidity from process variables only, and which variables that were important. The problem was potentially very high dimensional, since there are many sensors, which means that a dimensionality reduction step should be included in the model (e.g. principal components or a variable selection step). Furthermore, several sensors were correlated with other sensors and this should also be handled by the model.

The quality of the model was measured by how well the model was able to estimate the humidity in the paper on data that was not used in the modeling process.

7.3.2 Data, preprocessing, and variable selection

The SCA data set contained 35,345 observations, observed every minute, i.e. about 24 days of data. Each observation consisted of a time stamp, the output “gusktorhålt”, and 136 other measurements which could be used as input variables. The data set covered about two weeks of operation for the actual paper machine.

Several observations were made in the data:

**Missing output value:** The output value (“gusktorhålt”) was missing in 19,079 observations, i.e. 54% of the cases. There was thus only about two weeks of data available for constructing the model. This is quite a short period, and some of the variables showed signs of slower underlying dynamics which had not been captured over this short period.

**Typos in the output value:** Several erroneous values in the output value looked like typical typos. Some examples: 118.08 which should (judging from values before and after) have been 18.08, 1797 which should have been 17.97, and 29.36 which should have been 19.36. This was surprising and indicated to us that parts of the data may have been manually edited. The typos resulted in large spikes in the output value, see 7.3.

**Covariation in the data:** There was a lot of correlation between the variables. We therefore decided to use modeling methods that would be able to handle the covariation problem.

**Stop in the middle of the period:** The paper machine had been stopped for four days in the middle of the measurement period. Several variables exhibited different behaviors before and after this stop. An example is shown in 7.4 where a variable is shifted downwards after the stop. This indicated that it could be difficult to model the period after the stop using the same model as before the stop.

**Interpolated sensor values:** Several variables showed “interpolated” values, i.e. values that seemed to be interpolated estimates of their true values. This is visible e.g. in 7.4. These values could be an artifact of how the sampling system worked (it will only detect changes that are above a certain threshold) and did not reflect the true behavior of the measured physical quantity.

**Constant values:** Some variables were constant throughout the entire sampling period, or changed once or twice and remained constant the rest of the time.
Figure 7.3: Large spikes in the data caused by “typing errors”.

Figure 7.4: Example of the change in behavior before and after the machine was stopped. In this case the variable is shifted downwards after the stop. The top panel shows the variable before the stop. The second panel from the top shows the variable after the stop. The two bottom panels show nonparametric density estimates for the variable before and after. Left bottom panel: The density before the stop is drawn with a solid line, and the density after the stop is drawn with a dashed line. Right bottom panel: The density before the stop is drawn with a dashed line, and the density after the stop is drawn with a solid line.
The observations listed above lead us to clean the data significantly before any modeling was done. Unfortunately, this cleaning had to be done manually because of the nature of the errors (or odd behaviors).

- All typos in the output were corrected by manually looking at the values where large spikes occurred, and correcting to a value that was more in accordance with values before and after in time (the correction was obvious in all cases). A total of 17 values were corrected in this way.

- Long periods of spikes and interpolated values were removed from the data. A total of 5,775 observations (about four days) were removed manually in this way.

- All missing data were replaced by the last known value for that variable.

- All variables that were constant or changed only a few times were excluded from the data. This eliminated 11 variables before the stop, and 4 variables after the stop.

- The data was not filtered in any other way (i.e. no low pass filtering).

- All variables, including the output, are normalized before the model is built. This means that all variables are rescaled to have zero mean and unit standard deviation.

The large number of variables, the short measurement period, and the collinearity of the variables, suggested to lower the dimensionality of the problem. Either by using projection based methods, like principal components or partial least squares (or the neural network equivalent to partial least squares, principal component pruning), or to select the set of variables with the most information. We used the latter of the two, using the mutual information measure for selecting useful variables, as described in [Battiti, 1994].

In Battiti’s scheme the variables are selected in a forward manner, including in each step the variable \( x_k \) that maximizes the information gain

\[
I(y, x_k) - \beta \sum_{j \in S} I(x_j, x_k).
\]

(7.1)

Here \( S \) is the set of variables that have already been included in the model, \( I(x, y) \) is the mutual information between \( x \) and \( y \), and \( \beta \) is a (heuristic) factor between zero and one. The larger \( \beta \) the more are the cross mutual informations between variables in the set considered. If \( \beta = 0 \) then the variable with the maximum mutual information with the output value is always selected, regardless of which variables that have already been selected for the model. After some initial experimentation, we opted for \( \beta = 0 \) and to consider also the linear correlation. The time lags where also considered, i.e. the mutual information was computed for different time lags

\[
I[x_k(t - \tau), y(t)].
\]

(7.2)

The variable selection was done independently for the time periods before and after the stop. However, both periods were considered in the final selection of variables and the same variables were used in the models before and after the stop. This was based on our notion that a true physical model for the paper humidity should depend on the same variables both before and after the stop.

The final manual selection of variables was therefore based on the following criteria:

- The mutual information \( I(x_k, y) \) should be large in both the period before and after the stop.

- The linear correlation should not have opposite sign before and after the stop (we did in some cases discard this requirement because the correlations were very strong).

A positive example is shown in Figure 7.5. Here both the mutual information and the linear correlation measures agree on the conclusion. The optimal lag for this variable was set to 120 minutes. The reason
for this was a principle of “simplicity”, meaning that the longer the time lags are, the less simple is the model. Therefore the lag was set to 120 minutes rather than the 200-250 minutes that correspond to the maximum of the linear correlation and mutual information. The linear correlation for this variable at a lag of 90 minutes is not significantly different from the linear correlation at 225 minutes. The mutual information does not display any strong dependency with time. Unfortunately, this particular variable displays a strong connection with the output also before the stop, but now with an opposite sign on the linear correlation, as shown in Figure 7.6.

Such plots were made for each one of the 136 variables and the final variable set was decided manually.

### 7.3.3 Neural network model construction

A common technique for handling correlated variables in the context of linear models is to use ridge regression [Hoerl and Kennard, 1970b, 1970a]. The equivalent technique for neural networks is called weight decay. Here the idea is to use biased regression, or a so-called regularization term, so that the parameters $W$ of the model are chosen to minimize

$$E = \frac{1}{N} \sum_{n=1}^{N} [y(n) - \hat{y}(W, n)]^2 + \lambda \frac{1}{M} \sum_{m=1}^{M} w_m^2,$$

instead of just the mean square error (which is the first term in the expression above). This can be given a Bayesian interpretation where the second term in (7.3) corresponds to a prior distribution for
Figure 7.6: The same plot as in the previous figure, but for the period before the stop. This an example of a case where the linear correlation changes sign before and after the stop. However, the correlation is so strong that the variable should be included in the model (but we should not be surprised if the model does not work well across the stop boundary).

The parameters $W$. This prior is then a multivariate Gaussian with zero mean and a covariance matrix proportional to the identity matrix.

The combination of a prior with the standard likelihood (e.g., the mean square error) allows us to tune the power of the model, and effectively tell the model to try to keep the parameter values close to zero unless there is strong empirical evidence for giving them a nonzero value. The prior expresses the belief that all parameters are zero. This belief is weighted against the empirical evidence, in the form of the mean square error that tells how well the training data are estimated. This weighting is done with the $\lambda$ parameter, which is usually called a “hyperparameter” since it is not a part of the model itself but a parameter that controls the optimization of the model. The hyperparameter tuning is done by estimating the generalization error, using e.g., $n$-fold cross validation, for different values of $\lambda$ and the value that corresponds to the lowest generalization error is chosen.

In the extreme case of $\lambda \rightarrow \infty$ then all parameters are set to zero and the model outputs zero, independently of what input is presented to it. In the other extreme, $\lambda = 0$, we recover the standard least squares error.

An example is shown in Figure 7.7.

Of course, the $\lambda$ parameter selection is not independent of other hyperparameters for the model, like the number of hidden units or the number of inputs. The modeling must therefore be done in an iterative fashion, or via a “global” search over several values for these hyperparameters. This is exemplified in Figure 7.8, where a search has been done over $\lambda$ and the number of hidden units.

Expression (7.3) is minimized using a second order Levenberg-Marquardt algorithm. This is quite fast and efficient.

The whole model parameter fitting process can be summarized in the following steps:

1. The data is separated into two sets, a training set used for fitting the model parameters and a test set which is saved for final “out-of-sample” test. The latter set, the test set, is not used before all the model parameters have been set.
7.3 The model used at Halmstad University

Figure 7.7: Example of a scan over different values of $\lambda$ for a model that uses 10 input variables. The correlation of the variables does not cause a big problem for the model since the best results are achieved using very small (zero) values for $\lambda$. However, the case is very different if we test the model on data from the other side of the stop. In the latter case the model gets better and better the larger $\lambda$ we use (in the extreme case of $\lambda \to \infty$ the model outputs the mean of the training data, due to the normalization of the variables).

Figure 7.8: Training (top left), validation (top right), test (bottom left), and other side of stop errors (bottom right) for different values of the number of hidden units and $\lambda$. If the model should be used on the same side of the stop as it was trained, then it works fine to use many hidden units and $\lambda = 0$. However, if the model should be used on both sides of the stop, then few hidden units and large $\lambda$ are better. The errors have been normalized by the error made by a linear model (i.e. a value of 0 on the z-axis means that the model is equally good/bad as a linear model).
2. The training data is separated into five subsets which are used for cross validation: Four sets are used to compute error gradients while the fifth is used to validate, to estimate the generalization performance after the fit. This is done in a round-robin fashion until all five subsets have been used for validation.

3. The best hyperparameter value is selected using curves of the form in Figure 7.7. The five networks for this $\lambda$ value are then combined into a committee and the committee is used to produce results for the hold out test set from step 1.

7.3.4 Results

The variable selection process resulted in the following set of variables when ten variables are picked (in order of importance): \{34, 112, 106, 109, 105, 108, 111, 115, 107, 113\}. The corresponding lags are: \{0, 120, 0, 195, 0, 195, 195, 0, 0, 120\} minutes. The names of the variables are \{“Utloppskvot”, “Anisotropi D P.lab PM5”, “Porositet, Labprofil PM 5”, “Porositet, Labprofil PM 5”, “Porositet, Labprofil PM 5”, “Porositet, Labprofil PM 5”, “Porositet, Labprofil PM 5”, “Porositet, Labprofil PM 5”, “Densitet PM5”, “Porositet, Labprofil PM 5”, “Formation PM5” \} (it is intentional that six variables have the same name).

Both ANN models and linear models where then constructed using these variables (picking them in the order they are listed above). The results are summarized in Figures 7.9 and 7.10.

Figures 7.11 and 7.12 show how a committee model with three input variables compares to the true values before and after the stop. The model captures the trend when used on the same side of the stop as it was trained on, but it does not do better than a mean value model when tested on the other side of the stop.
7.3 The model used at Halmstad University

Figure 7.10: Summary of the best models that could be achieved given the number of input variables and training on data after the stop. Nothing is gained by using more than two input variables, when the model is trained and tested after the stop. Nothing is gained from using more than three-four input variables when the model is trained after the stop and tested before the stop, but the nonlinear model is better than the linear model.

Figure 7.11: The results of a committee model, trained on data from before the stop.
CHAPTER 7: SCA

7.3.5 Conclusion

Modeling the SCA data was to a large extent a frustrating experience. It is clear that there are slow effects (time constants of weeks) that are missed by the model, since no model is able to extrapolate over the stop. It is possible to construct a model for applying it on the same side of the stop as it was trained, but the physical interpretation and the predictive capability of such a model is questionable. Furthermore, the model is only applicable to about two weeks of data.

One could speculate on the origins of this long term behaviour. Is information about it present in the SCA data set at all, or are there other external pieces of information that are crucial? It could be due to buildup of residue somewhere.

The failure to construct a model can be due to any of three reasons: The key information is not present in the data set. The key information is there but data from a longer time period (the present data set covers only four weeks) is needed to spot it. The key information is there but the models are unable to extract it. We do, however, not believe in the last of these three.

Figure 7.12: The results of a committee model, trained on data from after the stop.
7.4 The model used at Skövde University

Mikael Bodén

7.4.1 Introduction

A set of feed forward and recurrent neural networks were trained to predict the degree of water in the produced pulp. Two input sets were used: raw values as measured from 136 sensors (D1) and the eight strongest principal components of the complete set of sensor data (D2). The inputs were also normalized so that the mean was 0 and the variance 1. The targets were in both cases the dewatering value. Two learning problems were studied: Predicting the current dewatering value (P1) and predicting the dewatering value 20 time steps ahead (P2; a setting which was motivated by the time needed to avoid paper breaks).

7.4.2 Method

The task was to evaluate different neural network architectures, specifically recurrent, on the previously described problem.

Simple recurrent networks (SRN; equipped with 3, 6, 11, 18, or 27 hidden self-recurrent units).

Feed forward networks (FFN; equipped with one hidden layer, FFN2, with 3, 6, 11, 18, or 27 hidden units, or no hidden layer, FFN1).

A suitable learning strategy is to let the model approximate the mean of the data given the input and assume noise with a Gaussian distribution. A linear output function was employed for the output unit. This allows the output unit take on any possible value. Non-linearities can still be encoded by the hidden layer which employs a non-linear squashing function. To vary the degrees of freedom of the model we tried various numbers of hidden units.

Learning parameters such as learning rate, momentum term and number of epochs until convergence were adapted after preliminary trials.

To evaluate the degree of generalization of the model it was subjected to a test data set which was prepared according to the same strategy as the training data but never shown during training. We chose the interval 13000-14000 (the third “burst” in the available data set).

The performance was measured according to the root-mean-squared (RMS) error and the correlation coefficient (CC) between the model’s output and the target data. Missing values were not used when assessing performance of the architectures.

The architectures are compared head-to-head according to their RMS error and CC. In P2 we also use the copy of the current (t) dewater level as a benchmark for the level t+20.

7.4.3 Results P1

The RMS errors and CC for the training data are generally lower than for the test data. Some overspecialization occurs in all tested architectures.

Short-term fluctuations observed in the real dewater values are not present in the output of any of the tested architectures. Longterm changes are, by visual inspection, matched by the models (see Figures 7.13–7.16).

The RMS error and the correlation coefficients indicate that all architectures perform equally bad (see Tables 7.3–7.4).
Figure 7.13: The output of some of the tested architectures over the whole data set (D1).

<table>
<thead>
<tr>
<th>HUs</th>
<th>3</th>
<th>6</th>
<th>11</th>
<th>18</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFN2/D1</td>
<td>0.3510</td>
<td>0.3525</td>
<td>0.3549</td>
<td>0.3499</td>
<td>0.3557</td>
</tr>
<tr>
<td>SRN/D1</td>
<td>0.3504</td>
<td>0.3530</td>
<td>0.3580</td>
<td>0.3483</td>
<td>0.3539</td>
</tr>
<tr>
<td>FFN2/D2</td>
<td>0.3558</td>
<td>0.3525</td>
<td>0.3493</td>
<td>0.3522</td>
<td>0.3600</td>
</tr>
<tr>
<td>SRN/D2</td>
<td>0.3543</td>
<td>0.3534</td>
<td>0.3504</td>
<td>0.3518</td>
<td>0.3589</td>
</tr>
</tbody>
</table>

Table 7.3: RMS error of the tested architectures (lower value is better).

7.4.4 Results P2

The prediction of dewater level 20 time steps ahead is in line with results on P1. Apart from the delay the only difference between P1 and P2 is that the current dewater value is used as input.

Training performance is similar to that of P1.

The ability to generalize is also similar to that on P1. As a benchmark to the errors presented in Tables 7.5 and 7.6, the copy of dewater level 20 time steps previous to the target ends up at RMS error 0.4790. The output values were again plotted for the test interval and compared with the mean target +/-10 time steps from the current time step (see Figure 7.17).

7.4.5 Discussion

Preliminary trials not reported above showed that locally recurrent architectures are slightly worse than the reported simple recurrent networks. Moreover, we have detected that refined learning by “back-propagation through time” in recurrent architectures presents no advantages in the studied domains. It was also observed that feed forward architectures with no hidden layer did much worse than multi-layered architectures. For these reasons the study focused on simple recurrent networks and feed forward architectures with a hidden layer.
Figure 7.14: The output of some of the tested architectures over the test data set 13000-14000 (D1).

<table>
<thead>
<tr>
<th>HUs</th>
<th>3</th>
<th>6</th>
<th>11</th>
<th>18</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFN2/D1</td>
<td>0.1918</td>
<td>0.1997</td>
<td>0.2039</td>
<td>0.1774</td>
<td>0.1399</td>
</tr>
<tr>
<td>SRN/D1</td>
<td>0.1950</td>
<td>0.1972</td>
<td>0.1821</td>
<td>0.1818</td>
<td>0.1538</td>
</tr>
<tr>
<td>FFN2/D2</td>
<td>0.1758</td>
<td>0.2116</td>
<td>0.2401</td>
<td>0.2356</td>
<td>0.2212</td>
</tr>
<tr>
<td>SRN/D2</td>
<td>0.1964</td>
<td>0.2099</td>
<td>0.2380</td>
<td>0.2412</td>
<td>0.2279</td>
</tr>
</tbody>
</table>

Table 7.4: Correlation coefficient of the tested architectures (higher value is better).

In general the results are poor. The data consists of many short-term fluctuations and it is believed that such fluctuations are too frequent and noisy for a model with only a limited number of parameters to adapt to. Short-term fluctuations are believed to be caused by the radioactive isotopic sensor used for measuring the dewater level. Speculatively, some preprocessing smoothing filter could be used for improved performance. General trends (longterm) were tracked but such were reportedly not very interesting for SCA.

The simple recurrent network is slightly better than the feed forward architectures but the difference is not significant. The recurrent networks have the advantage to be able to use inputs even when a target is missing to develop a state which is used later.

There is no result which give any advantage to any of the two data sets on P1. For P2 the principal component data is slightly better.
Figure 7.15: The output of some of the tested recurrent architectures over the test data set (D1) and the principal components (D2).

<table>
<thead>
<tr>
<th>HUs</th>
<th>3</th>
<th>6</th>
<th>11</th>
<th>18</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFN2/D1</td>
<td>0.3604</td>
<td>0.3502</td>
<td>0.3564</td>
<td>0.3529</td>
<td>0.3555</td>
</tr>
<tr>
<td>SRN/D1</td>
<td>0.3570</td>
<td>0.3523</td>
<td>0.3686</td>
<td>0.3549</td>
<td>0.3551</td>
</tr>
<tr>
<td>FFN2/D2</td>
<td>0.3624</td>
<td>0.3489</td>
<td>0.3535</td>
<td>0.3539</td>
<td>0.3536</td>
</tr>
<tr>
<td>SRN/D2</td>
<td>0.3622</td>
<td>0.3479</td>
<td>0.3528</td>
<td>0.3525</td>
<td>0.3514</td>
</tr>
</tbody>
</table>

Table 7.5: RMS error of the tested architectures on P2 (lower value is better).

<table>
<thead>
<tr>
<th>HUs</th>
<th>3</th>
<th>6</th>
<th>11</th>
<th>18</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFN2/D1</td>
<td>0.1990</td>
<td>0.1942</td>
<td>0.1970</td>
<td>0.1908</td>
<td>0.1832</td>
</tr>
<tr>
<td>SRN/D1</td>
<td>0.1896</td>
<td>0.1849</td>
<td>0.1788</td>
<td>0.1700</td>
<td>0.1877</td>
</tr>
<tr>
<td>FFN2/D2</td>
<td>0.2136</td>
<td>0.2061</td>
<td>0.2000</td>
<td>0.1918</td>
<td>0.1723</td>
</tr>
<tr>
<td>SRN/D2</td>
<td>0.2145</td>
<td>0.2160</td>
<td>0.1960</td>
<td>0.2077</td>
<td>0.2033</td>
</tr>
</tbody>
</table>

Table 7.6: Correlation coefficient of the tested architectures on P2 (higher value is better).
Figure 7.16: The output of some of the tested recurrent architectures over the test data set (D1) and the principal components (D2). The mean target value (+/- 10 time steps from the current time step) is plotted as a comparison.

Figure 7.17: The output of some of the tested architectures over the test principal components data (D2). The mean target value (+/- 10 time steps from the current time step) is plotted as a comparison.
7.5 The model used at DSV

Lars Asker

The task was to predict the values of the “dewatering” variable. Given the large amount of data and the quality of it, we decided to select for training data a subsampling from two time intervals where the process had been relatively stable, and to use a third stable time interval for testing. The method used was to build a bagged regression tree constructed from 20 component classifiers. When performing cross-validation of the training data, the model was able to get a correlation coefficient in the range between 0.7 and 0.8, while the results were much worse when applied to the independent test data. The conclusion was that, although the models were quite good at showing trends in the training data, they suffered badly from overtraining and had no capability to generalize to unseen data. This is something that all methods suffered from; see section 7.7 below for a more detailed discussion about this.
7.6 The model used at Mitthögskolan

Mikael Hall and David Martland

The search for the most useful variables and time delays was accomplished by running each variable for over a large number of delays against the target and by selecting the ones with a mutual information value over 1, which was arbitrarily chosen by trial and error. This resulted in twenty input variables with a quite big span of delays. The time delays found was taken as is, though the noise content is too large to pinpoint any optimal delays by mutual information more than roughly. We then conducted an experiment to investigate if the SCA data has some bound on the state memory by running a number of networks without state information, where the “distance” between training and test data is successively increased.

The experiment was set up the following way:

1. First the data was adjusted according to the results of task 1 above, resulting in 6900 data points consisting of the twenty inputs, the current value of the target and the target twenty points into the future. The points are one minute apart.

2. Then we ran ordinary feedforward networks (Levenberg-Marquardt backpropagation) with 0, 1, 2, 5, 10, 15 hidden units plus linear output without state information. Each network was given training, validation and test sets according to the following:

   • The timeseries was divided into four sets by taking every fourth data point starting from the first, to form the first set. Then we did the same, starting from the second data point, to form the second set and so on. The first and third sets formed the training set, second the validation and the fourth the test set. Thus the “distance” between the training- and test set is one data point.
   
   • By increasing the so defined distance from 1 to 400 (step-size 10) the goal is to see if networks stops “cheating” at some distance.

From twenty runs on each network and distance, the median and variance of the performance (measured as the correlation coefficient between the network output and the target) was calculated (boxplots). The smaller networks showed a critical point at distance 160-180, see 7.18. Training performance started lower and increased up to this point showing no variance before or after, but test performance steadily decreased starting to show variance at this point. There seems to some loss of memory at distance 160-180. However, the investigation showed no critical point of loss of memory for the larger networks, see 7.19. The variance of network performance decreased smoothly for the training performance and test performance variance increased with distance, showing a somewhat periodic behavior.

The results may indicate that the data is more scarce than suspected at first glance with respect to the number of states it contains, since when using models with high storage capacity they manage to cheat even when the the above distance is large. The degradation of the models is most likely due to the decreasing number of samples.
Figure 7.18: The linear model, at different "distances" (train and test).
Figure 7.19: The model with 15 nodes in one hidden layer, at different “distances” (train and test).
7.7 Discussion
Anders Holst

The dewatering dataset has been carefully analyzed by the partners, and the general conclusion is that there are no or very little information in the given input variables about the dewatering. Only very weak correlations were found when checking with mutual information and mutual information rate. The trained models were able to predict the large features of the dewatering, but the fast changes was not caught by any model. Due to the relative short range of useful data, it is unclear whether the prediction of the large features of the dewatering was a coincidence only working in this piece of data, or whether it will also hold more generally. Those tests that tried to predict the behavior during the second period (after the service stop) after having been trained only on the first period (before the stop) indicate a poor generalization, which casts serious doubt on the reliability of also the large scale predictions of the different methods.

This lack of information in the data may however have several reasons. For example, dewatering is very hard to measure, which is the reason a prediction using the other sensor readings would be preferred. If we have got it right, the actual measurements of the dewatering was obtained by a relatively new method by some external consultants. How reliable is this method? Can we be sure that it reflects the water content, or could the output perhaps be mainly noise? The latter would explain why the fast fluctuations could not be predicted by any method.

On the other hand, supposing that it reflects the real dewatering, some other interesting details could be seen. An interesting tendency, especially clear in the first period, but also to some extent present in the second, is that the dewatering value slowly decreases, then there are some missing values, and then it is suddenly high again. This could be explained by some parts in the paper machine slowly clogging up, and then suddenly getting cleaned again. Again, this clogging up could of course be happening in the measuring device itself, in which case this behavior says nothing about how the real dewatering changes. An indication that would support this is that no other variable in the data behaved in the same way. However, such accumulation of dirt is known to occur in the process. Specifically, the “felt” on which the pulp is applied is gradually clogged up, causing the dewatering to decrease. Still, the degree of clogging is not shown in any of the variables in the data we had received. When discussing this with SCA, it was remarked that the power consumption of the engines for that “felt” indeed goes up gradually as it clogs. Perhaps the power consumption would then be a useful indicator on the dewatering. In the discussions with SCA we identified several similar sensor readings not contained in the original data but which could be of interest here. Unfortunately that data was not made available to the project.

Also unfortunate was that time did not allow us to look seriously into the second issue of interest: predicting the rate of paper breaks. There are however some interesting details in the data that can be noted also with regard to this. We were told in advance that two factors that are known to affect the rate of paper breaks are the dewatering and the speed of the tambour reel. Looking at the data no significant correlation could however be found between the occurrence of paper breaks and the dewatering value. This may not be surprising, considering the limited time the dewatering measurements were given, and given the uncertain quality of those measurements. More surprising however is that there is also no significant correlation between the speed and the occurrence of breaks! This can also be confirmed visually in the data, where almost exactly as many breaks occur in periods when the tambour reel is running fast, as during approximately equally long periods when it is running relatively slow.

Although it is a tempting conclusion that the paper breaks does not depend on the speed, this would be a little premature statement. For example, it could be that the operators generally manage to keep the probability of a break to a minimum by not running faster than is allowed by the current circumstances. It would then look like the number of breaks are independent, although it is instead a regulatory effect of selecting the speed that keeps the break rate constant. To always run fast would then certainly increase the number of breaks. Still, the speed did in turn not seem to have any strong dependencies on the other variables in the data. Such a dependency would have been expected if the speed were regulated as response to some other circumstances in the process. But then again, we only received a small subset of the available sensor readings, so there may very well be such a dependency somewhere. In any case, this may be worth for SCA to look further into.

Although it unfortunately did not fit into the limited scope and time of the DALLAS project, there
are as shown above several further questions that would need to be answered, and several important issues that would need further attendance, in the problem complex surrounding a paper machine.

7.8 References


Chapter 8

Telia:
Detection of frauds in a Media-on-Demand system in an IP network

8.1 Problem description
Henrik Jacobsson

Telia has performed field pilot tests on a Media-on-Demand (MoD) system for downloading and paying for movies and television over an ADSL connection. The problem is that the service may invite morally non-scrupulous people to take advantage of the system for their own fraudulent purposes. Telia is interested in developing an automated system for detecting these frauds. There is a large number of possible frauds that can be conducted. However, for practical purposes, Telia has selected a small number of known frauds, feasible in terms of detectability, to be studied in this project. These frauds are:

Illegal redistribution. This is when a user downloads movies or other broadcasted material (in a legal or illegal way) and later redistributes them to other users.

Excess download. This is when the user has tampered with the billing system, making it possible to download movies or TV without paying for it.

Subscription fraud. A user registers himself under false identity and manages to use the services of the media-on-demand system without paying for it.

In addition to detecting these frauds, Telia wants to detect new fraud types. This is hypothetically possible by detecting behaviours deviating from normal behaviour and then further analyze these behaviours to establish whether they are fraudulent or not.

8.1.1 Fraud indicators

Telia lists a number of possible indicators for the three kinds of frauds being studied.

Illegal redistribution. If the ratio between downloaded and uploaded data volumes is suspiciously high in general, this may be an indicator that the user redistributes movies. If the user is uploading a lot of data at the same time as he/she is downloading a live event, this is an even stronger indicator. If the user is spending a lot of money on an unreasonable number of movies, it is reasonable to
believe that the user is (financing this by) selling them. Since the redistribution is more likely to be focused on a specific service, an abnormal increase in usage of a service together with an increased traffic is considered as an indicator. If the received and transmitted data is of the same kind (e.g. by showing similar statistics or burst patterns), this may indicate redistribution fraud.

**Excess download.** If more data is transmitted than the user is billed for, excess download is probably in progress; alternatively, a billing system error has appeared.

**Subscription fraud.** If a new user deviates substantially from the average new user and uses services for an excessive amount it may be an indicator that the user ID is used by someone who will not pay for the services.

However, due to lack of time, not all of the indicators above are supported by the data. The indicators used must all be derivable from the data described in the following text. The extraction and interpretation of such indicators belongs to the task that should be solved by the learning systems.

### 8.2 Data description

**Henrik Jacobsson**

#### 8.2.1 The data collection

The original idea was to use real data, i.e. data from the pilot tests Telia is conducting. The collection of real data was problematic since there were unfortunately very few users (only 12 users, many of which did never order any movies) and it was difficult to explicitly define when exactly a fraud is being committed. This data was used only at SICS (see section 8.3.2). To provide more data, Telia has created a simulator in which thousands of users can be simulated in a controlled way.

We choose to use 518 simulated users for training of which 18 are fraudulent (6 of each fraud type). The same number of users is used for the test set. The simulator was run for 6 months of simulated time. The fraudulent users where programmed to behave as normal users until they start their fraudulent behaviour some time during the 3 first months. Unfortunately, the simulator did not log when a user turns fraudulent. Therefore we filtered out the first three months, where there is an uncertainty if the user is fraudulent or not. This means that, in practice, the problem was reduced to a classification of users.

#### 8.2.2 The raw data

The original simulated data distributed by Telia will only be described briefly since no classification will be based on this data directly. In summary, the data can be said to be “difficult” in many ways. It is sampled at different parts of the system and at different rates. Some of the data is sampled regularly (e.g. router statistics) and some is generated at the triggering of some events (e.g. ordering of movies). This type of irregular representation of time is not suitable for the majority of methods used in this project; yet, this situation reflects the actual situation of an operating telecom system. Moreover, it is difficult to identify users since the identity of a user is not represented in the same way in all files. To separate users from each other is very important since we want to separate fraudulent from normal users.

#### 8.2.3 Refined data

The difficulties of identification and resampling of data with a synchronous time representation were overcome and it resulted in a refined dataset. This dataset was divided into one sequence for each separate user. Each row in the sequence represents one half hour of simulated time where everything that was started within that interval is included.
<table>
<thead>
<tr>
<th>Column #</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Line number</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>Time</td>
<td>Time after 2001-05-05 00:00:00 in seconds. Denotes the starting point of the interval.</td>
</tr>
<tr>
<td>3</td>
<td>User ID</td>
<td>A unique user ID.</td>
</tr>
<tr>
<td>4</td>
<td>Order requests</td>
<td>Ordered movies during that interval.</td>
</tr>
<tr>
<td>5</td>
<td>Delivery Notification</td>
<td>Delivered movies during that interval.</td>
</tr>
<tr>
<td>6</td>
<td>Billing Notification</td>
<td>The number of movies the user has been billed for during the interval.</td>
</tr>
<tr>
<td>7</td>
<td>Downloaded (DL)</td>
<td>Downloaded bytes during the interval.</td>
</tr>
<tr>
<td>8</td>
<td>Uploaded (UL)</td>
<td>Uploaded bytes during the interval.</td>
</tr>
<tr>
<td>9</td>
<td>UD-ratio (UD)</td>
<td>$UD = \begin{cases} \frac{UL}{DL} &amp; \text{if } DL \neq 0 \ \frac{UL}{DL} &amp; \text{otherwise} \end{cases}$ where $DL$ is the average downloaded bytes for all intervals.</td>
</tr>
<tr>
<td>10</td>
<td>Downloaded unicast</td>
<td>Downloaded Unicast-packets.</td>
</tr>
<tr>
<td>11</td>
<td>Uploaded unicast</td>
<td>Downloaded Unicast-packets.</td>
</tr>
<tr>
<td>12</td>
<td>UD-Ucast-ratio</td>
<td>The same procedure as for attribute 9.</td>
</tr>
<tr>
<td>13</td>
<td>Hour</td>
<td>Hour of the day the interval started, i.e. an integer in between 0 and 23</td>
</tr>
<tr>
<td>14</td>
<td>Active</td>
<td>1 if user is active in the interval, 0 otherwise.</td>
</tr>
<tr>
<td>15</td>
<td>No fraud</td>
<td>1 if the user never commits any fraud, 0 otherwise.</td>
</tr>
<tr>
<td>16</td>
<td>Illegal redistribution</td>
<td>1 if the user commits illegal redistribution, 0 otherwise</td>
</tr>
<tr>
<td>17</td>
<td>Billing fraud</td>
<td>1 if the user commits billing fraud, 0 otherwise (same as Excess download described above).</td>
</tr>
<tr>
<td>18</td>
<td>Break in fraud</td>
<td>1 if the user commits break in fraud, 0 otherwise (same as Subscription fraud described above).</td>
</tr>
</tbody>
</table>

Table 8.1: The attributes used extracted from Telia’s data. The last four attributes represent the target of the domain and the first three are not used for the classification.

A description of the attributes is found in table 8.1. Since some information may occur in the original data that are not transferred into the suggested representation there is a risk that some helpful information may have been deleted. This is especially critical for detecting novel fraud types. There are, for example, assumptions regarding the identification of users that work fine for the normal case but may not be applicable if someone starts hacking other user accounts or if the IP-addresses are changed, which may happen since dynamic IP-addresses are used.
8.3 The model used at SICS
Daniel Gillblad

8.3.1 Supervised and unsupervised fraud detection

Fraud detection can be performed in two fundamentally different ways. Either the fraud types are known, and data classified into fraudulent, perhaps with a fraud type tag, or non fraudulent is available. It is then at least hypothetically possible to construct a classifier that will classify new data patterns into fraudulent or non fraudulent, perhaps even with a very simple set of rules specified by someone who knows how these frauds are usually committed. The other possibility is that there is no knowledge about what fraudulent behaviour might look like, or perhaps that we want to be able to detect new types of frauds. This makes the task much harder, since we cannot extract any information on what the frauds might look like from data. Instead, we have to try to build a model of normal, non fraudulent behaviour. When new data patterns deviate significantly from this model, we can classify it as a possible fraud.

In the first scenario, where we have labelled data available for training of a learning system, we can use a supervised learning algorithm to construct our classifier. Supervised learning is only possible when labelled data is available. When this is not the case, we have to rely on unsupervised learning, i.e. there are no labelled examples to be learned by the system. This is the situation in the second scenario, where we just want to build a general model of the data that represents normal behaviour. The second scenario can be regarded as generally more difficult and sensitive to the choice of model. On the other hand it is, if it is performing with sufficient precision, perhaps even more useful to a company than a normal classifier trained on labelled data.

The supervised training method uses a mixture of Gaussians, with one Gaussian for each fraud type. The model is trained by estimating each Gaussians parameters on the corresponding data, i.e. the Gaussian that represents fraud type one is estimated from all data categorized as fraud type one, the Gaussian that represents fraud type two is estimated from all data belonging to fraud two and so on. When a classification is made, the input pattern is presented to each Gaussian. The corresponding fraud type of the Gaussian with the highest likelihood for the pattern is the classification result.

In the unsupervised case, a mixture of Gaussians is trained on non fraudulent data by the Expectation Maximization algorithm (see section 2.6.7). The resulting distribution is an estimate of the real distribution of non fraudulent data. If a new pattern has a very low likelihood of being drawn from this distribution, it can be considered to be a possible fraud or at least deviant behaviour.

8.3.2 The data used to train the models

The pilot test data set distributed by Telia contained fraud, but without any good indication of when and by whom the frauds were committed. To be able to use the dataset for testing fraud detection, the data was labelled for all users and time steps using a simple scheme based on the hints given about when the different frauds took place. The data was labelled as

1. Inactive
2. Active
3. Redistribution
4. Excess download
5. Subscription fraud

Patterns were only labelled as frauds at the time they could be considered to be committed, resulting in a rather low number of fraudulent patterns. This and the fact that each type of fraud is committed only by one user makes it impossible to divide the dataset into one training set and one testing set for
cross validation. One separate dataset was constructed though for training of the unsupervised model. This dataset contains every second day from all users, with all fraudulent behaviour filtered out.

Not all available data was used for training the models. Only order requests, delivery notification, billing notification, downloaded bytes and uploaded bytes were used, both on pilot test data and on simulated data. In both cases, the uploaded and downloaded bytes were also transformed by the logarithm to the power of three to compensate for dynamics in the data.

In the simulated data case, the amount of data was simply too large to train the models in a reasonable amount of time. Therefore, a subset of the users in the training set were selected for the training data. This subset includes all six fraudulent users of each fraud type and 18 randomly selected normal users, i.e. the training data set consists of 18 fraudulent users and the same number of normal users. For the unsupervised training, only the non-fraudulent users were used.

### 8.3.3 Results on pilot test data with supervised training

A mixture model was trained supervised over all pilot test data as described above. The model was then tested on the same dataset, with the results shown in table 8.2. The rows of the table show the classification made by the model, the columns the true value, and the each value show the number of patterns that fall into that category. Thus, if the prediction was perfect, all values except the diagonal would be zero. From the table we can see that the classifier sometimes confuses active and inactive behaviour. This is not much of a problem in this experiment though, since both types are non-fraudulent. The classifications of subscription frauds and excess download are without errors. Some slight mistakes are made for the redistribution fraud, classifying redistribution as normal active behaviour.

### 8.3.4 Results on pilot test data with unsupervised training

The unsupervised model contained three Gaussians and was trained on every second day of data from all users, with all fraudulent examples removed from the training dataset. The reason for training on every second day of data was to test if the model could generalize the information about the normal distribution to the whole dataset. The model was then tested on the complete dataset, with the results shown in figure 8.1. The upper plot shows the log likelihood of each pattern, while the lower shows the fraud type. The plots show that fraud types 3 and 4 (excess download and subscription fraud) causes significantly lower log likelihood than the non-fraudulent patterns. Redistribution fraud on the other hand does not show any lower log likelihood and could not be detected by the system. This is probably not a problem with the model, but rather a problem with the labelling of the data. The patterns marked as redistribution fraud are based on very basic assumptions that might very well prove to be wrong. In fact, later information suggested that there were probably no real redistribution fraud in the used data set at all.

### 8.3.5 Results on simulated data with supervised training

A mixture model was trained as described above on the reduced data set. The classification is made for each time step, not one general classification for one user. The results of the classifications are shown in

<table>
<thead>
<tr>
<th>Inactive</th>
<th>Active</th>
<th>Redistribution</th>
<th>Excess download</th>
<th>Subscription fraud</th>
</tr>
</thead>
<tbody>
<tr>
<td>23292</td>
<td>1995</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Active</td>
<td>81</td>
<td>28730</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Redistribution</td>
<td>0</td>
<td>0</td>
<td>153</td>
<td>0</td>
</tr>
<tr>
<td>Excess download</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Subscription fraud</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 8.2: Supervised prediction results on pilot test data
CHAPTER 8. TELIA

Figure 8.1: Log likelihood and fraud class type for pilot test data

Table 8.3: Supervised prediction results on simulated test data

<table>
<thead>
<tr>
<th></th>
<th>No fraud</th>
<th>Redistribution</th>
<th>Excess download</th>
<th>Subscription fraud</th>
</tr>
</thead>
<tbody>
<tr>
<td>No fraud</td>
<td>2310670</td>
<td>0</td>
<td>105</td>
<td>56</td>
</tr>
<tr>
<td>Redistribution</td>
<td>694</td>
<td>584</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Excess download</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Subscription fraud</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 8.3. The layout of the table is the same as for table 8.2, except that the active and inactive classes in table 8.2 have been joined to one no fraud class.

If we assume that a user is fraudulent if he or she has been classified as having fraudulent behaviour at any point in time, we can translate the classification per time step to a classification per user. We can then calculate sensitivity and specificity, as described earlier, for all types of fraud. The result is shown in table 8.4.

The table does not show any results for the excess download fraud. It turned out after the experiments that in the test set, all data marked as excess download frauds were incorrectly labelled and should really belong to the illegal redistribution fraud. The results in table 8.4 compensates for this.

8.3.6 Results on simulated data with unsupervised training

The unsupervised model again contained three Gaussians and was trained on the selected normal users. The model was tested on the whole dataset and the results are shown in figure 8.2. As before, the upper plot shows the log likelihood and the lower plot the fraud type, here denoted 1 for no fraud and 2 to 4 for the different fraud types. All fraud types show significantly lower log likelihood in the model. The model also produces some dips in the log likelihood even when there is no fraud, but they are smaller than the ones produced by fraudulent behaviour. This has one exception though. In the beginning of
The model used at SICS

<table>
<thead>
<tr>
<th></th>
<th>Sensitivity</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>No fraud</td>
<td>0.96</td>
<td>1.0</td>
</tr>
<tr>
<td>Redistribution</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Subscription fraud</td>
<td>1.0</td>
<td>0.24</td>
</tr>
</tbody>
</table>

**Table 8.4:** Supervised prediction results on simulated test data, as sensitivity and specificity.

Figure 8.2: Log likelihood and fraud class type for simulated test data

In the last quarter of the data there is one large dip in the log likelihood, but the pattern is not marked as fraudulent. The reason is that at that time, the user is billed for a movie that has not been ordered or delivered. This makes the pattern stand out from normal behaviour, although it is not the customer that behaves differently but the movie provider.
8.4 The model used at Halmstad University

Mikael Bodén

8.4.1 Summary

A set of feed forward and recurrent neural networks was trained to continuously classify user actions into four classes: “no fraud” (1), “illegal redistribution” (2), “billing fraud” (3) and “break-in fraud” (4). Recurrent architectures are more suited for the task and outperforms feed forward architectures consistently. The best models are able to correctly classify fraudulent behavior but are unable to distinguish (3) from (2).

8.4.2 Background

From the original 11 input variables, 5 were selected on basis of suspected importance (numbers of ordered, delivered and billed movies, ratio of download/upload multicast data, and ratio of download/upload unicast data.

Fraudulent behavior was suspected to be best identified by taking into account temporally extended patterns of activity. Two strategies for enabling a model to find such patterns were implemented in this study. First, averages of input values were collected for various intervals (1 [no averaging], 10, 50, 100, or 200). That is, for any instance in time the input values were a result of a set of earlier input values. Secondly, feed back activation were available to recurrent networks.

User input data was presented sequentially for 40 steps incremented each instant with the selected interval. Since user activity was limited to about 4000 tuples, only 20 sweeps were used for the 200-interval setup. The starting point was chosen randomly for each user (both for training and testing). The user specific target class was made available to the network at all times.

All networks were equipped with 5 inputs (one for each input variable) and 4 outputs (one for each class, designated to be “true” if activation was 1 and “false” if activation was 0). Two feed forward networks were tested. One single layered network (fnh0; no hidden units) and one network with 5 hidden units (fnh5). Three recurrent networks were tested: one with 5, one with 10 and one with 15 fully recurrent hidden units (srnh5, srnh10 and srnh15).

8.4.3 Model adaptation

Training and testing data each consisted of about 500 user activity scenarios. Of the 500, 18 were fraudulent (6 of each fraud class). Only the 18 fraudulent users plus another 30 “legal” users were used for directly adapting weights of the networks (the training set). One user from each fraud class plus another 5 normal users (not the same as in the training set) were selected as a validation set and used for choosing a suitable stopping criterion for training. The networks were trained for a maximum of 100000 users chosen in random order. Tests were performed on the full data set intended for testing.

Weight adaptation was based on the gradient of a maximum likelihood cost function defined over training data. In the case of recurrent networks, the gradient was propagated “backwards” for 5 time steps. The output function on output units in all networks was the so-called softmax function, which normalizes outputs so that they can be interpreted as probabilities. Forced classification was based on the output unit with the highest activation.

Various learning rates were tested. Good performance for all network types was found – with the exception of the single layered network (which never performed well) – when the learning rate was around 0.001 and our study focuses on those results. Other learning rates gave similar performance.
8.4 The model used at Halmstad University

<table>
<thead>
<tr>
<th>No fraud</th>
<th>Illegal redistribution</th>
<th>Billing fraud</th>
<th>Break-in fraud</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.98</td>
<td>0.37</td>
<td>0.00</td>
<td>0.51</td>
</tr>
<tr>
<td>0.98</td>
<td>0.27</td>
<td>0.00</td>
<td>0.35</td>
</tr>
</tbody>
</table>

Table 8.5: Average performance of all tested recurrent networks.

<table>
<thead>
<tr>
<th>No fraud</th>
<th>Illegal redistribution</th>
<th>Billing fraud</th>
<th>Break-in fraud</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.95</td>
<td>0.14</td>
<td>0.00</td>
<td>0.36</td>
</tr>
<tr>
<td>0.97</td>
<td>0.29</td>
<td>0.00</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 8.6: Average performance of all tested feed forward networks.

<table>
<thead>
<tr>
<th>No fraud</th>
<th>Illegal redistribution</th>
<th>Billing fraud</th>
<th>Break-in fraud</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>0.91</td>
<td>0.00</td>
<td>0.98</td>
</tr>
<tr>
<td>1.00</td>
<td>0.48</td>
<td>0.00</td>
<td>0.83</td>
</tr>
</tbody>
</table>

Table 8.7: Average performance of all recurrent networks trained using averages over 100 time steps.

<table>
<thead>
<tr>
<th>No fraud</th>
<th>Illegal redistribution</th>
<th>Billing fraud</th>
<th>Break-in fraud</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.97</td>
<td>0.47</td>
<td>0.29</td>
<td>0.29</td>
</tr>
<tr>
<td>0.98</td>
<td>0.50</td>
<td>0.16</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 8.8: Average performance of all feed forward networks trained using averages over 100 time steps.

8.4.4 Results

Two performance measures were identified: sensitivity (the ratio between true positives and true positives plus false negatives; informally, the degree of within-class members classified correctly; 1 is best, 0 is worst) and specificity (the ratio between true positives and true positives plus false positives; informally, the degree of misclassifications of outside-class samples; 1 is best, 0 is worst). To test generalization, all measurements were done of the test data.

With no consideration to average interval, recurrent networks outperformed the feed forward networks (see Table 8.5 and 8.6).

Best classification performance was achieved when averages over 100 time steps were used as inputs (see Table 8.7 and 8.8).

The networks confuse class 3 with class 2. None of class 3 members were correctly classified (zero sensitivity of class 3) and about half of those classified as class 2 were misclassifications of class 3 members (about half specificity of class 2). It later turned out that this misclassification was actually due to mislabeled data, see section 8.7.1.

The classification performance on class 1 is excellent. Class 4 is classified correctly most of the time. When misclassification occurs class 4 members still end up as being fraudulent (2).

As only a limited number of simulations were performed, no significant differences were identified between networks with different number of hidden units.
8.5 **The model used at DSV**  
* Lars Asker

8.5.1 **Summary**

Two classifiers were trained to classify the data into four classes. The four classes were labeled “no fraud” (1), “illegal redistribution” (2), “billing fraud” (3) and “break-in fraud” (4). The first classifier used ordered Separate and Conquer (Ordered-SAC), and the second used Divide and Conquer with the Minimum description length principle to select split points (DAC-MDL). The data was divided into one training and one test set. Each set consisted of 518 examples of which 500 were normal users, and 6 from each of the 3 fraudulent classes.

8.5.2 **Data preparation**

We decided to represent the data by using the summary information that was available for each user. The summary files contained the following information.

- (UID) User id
- (LOC) Physical location
- (SID) Subscriber id
- (FT) Fraud type
- (AC) Activity
- (OR) Total number of ordered films during the interval
- (DN) Total number of delivered films during the interval
- (BN) Total number of films that were paid for during the interval
- (MOR) Maximum number of ordered films
- (MDN) Maximum number of delivered films
- (MBN) Maximum number of films that were paid for
- (TADL) The total average of downloaded bytes
- (TAUL) The total average of uploaded bytes
- (TADLU) The total average of downloaded unicast packages
- (TAULU) The total average of uploaded unicast packages
- (TAUDR) The total average ratio between uploaded and downloaded bytes
- (TAUDUR) The total average ratio between uploaded and downloaded unicast packages
- (AADL) The average of downloaded bytes during active periods
- (AAUL) The average of uploaded bytes during active periods
- (AADLU) The average of downloaded unicast packages during active periods
- (AAULU) The average of uploaded unicast packages during active periods
- (AAUDR) The average ratio between uploaded and downloaded bytes during active periods
- (AAUDUR) The average ratio between uploaded and downloaded unicast packages during active periods
• (NOI) The number of intervals
• (IV) The interval length

The variables containing information about User id, Physical location, and Subscriber id, were removed, and the remaining variables were used to represent the data.

8.5.3 Results

Both methods were able to correctly distinguish between normal and fraudulent users while Ordered SAC incorrectly classified 5 examples of class 3 (billing fraud) as class 2 (illegal redistribution), and DAC-MDL incorrectly classified 6 examples of class 3 as class 2. This fact (together with the results from the other groups) led us to suspect that an error had occurred while labeling the training data (see section 8.7.1). If the labels of the 6 training examples that where labeled as belonging to class 3 where changed to class 2, DAC-MDL classified all the training example correctly while Ordered SAC misclassified one example.

The model created by Ordered SAC could be represented as the following decision list:

IF (BN < 4) THEN <class 3>
ELSE IF (AC < 29.6) THEN <class 4>
ELSE IF (TAUL > 8075434.91) THEN <class 2>
ELSE <class 1>

where BN = ‘The total number of films that were paid for during the interval’, AC = ‘Activity’, and TAUL = ‘The total average of uploaded bytes’.

The model created by DAC-MDL could be represented as the following set of rules:

IF (AAUL <= 2480.58) AND (BN <= 5.5) THEN <class 3>
IF (AAUL <= 2480.58) AND (BN > 5.5) THEN <class 1>
IF (AAUL > 2480.58) AND (AC <= 36.557) THEN <class 4>
IF (AAUL > 2480.58) AND (AC > 36.557 THEN <class 2>

where AAUL = ‘The average of uploaded bytes during active periods’, BN = ‘The total number of films that were paid for during the interval’, and AC = ‘Activity’.
Figure 8.3: There are two maps represented by a U-matrix showing the distance between map units, measured in the input space, and by maps color coded by the values of each input dimension as represented by the map unit prototype vectors. The map to the right has the best-matching units, bmu’s, marked. Each sample has a bmu on the map and the size of each marker is proportional to the number of samples it resembles most relative the other units.

8.6 The model used at Mitthögskolan
Mikael Hall and David Martland

8.6.1 Initial try

The distributed data was quite big, some 5 million samples in total, we only took a randomly chosen subset. Each of the users in the train and the test portion (1036 files in total, with about 4000 samples each originally), contributed 250 samples. The target in this data had one target with range 1-4, representing no fraud, illegal redistribution, billing fraud and break in fraud respectively. This target was preprocessed into an orthogonal form, so that it was represented by four variables, each of range 0-1, where '1' indicates fraud. The training part contained eighteen cheaters, six of each type.

The self-organizing feature maps did not give other useful clues other than that the labelling was done constant for each cheating user, regardless of him/her being active or not. There seemed to be some correlating behavior only in a small subset of the samples labelled fraud. Also the test portion was useless in the sense that it contained the same cheaters as in the train part.

8.6.2 Second try

Because of the problems mentioned above we did some filtering of the data and we changed the goal from modelling to that to illustrate the use of self-organizing feature maps. First, we argued that the billing fraud type is only interesting when the user has received a movie. Thus we only work with samples in which a delivery has occurred. Perhaps we can see if the maps reveal the obvious clue that no payment was done. We also have tried to capture other clues, but due to time shortage, no other filtering has been investigated properly and no other fraud type. The self-organizing feature map without any magnification with regards to the target (billing fraud) is given in figure 8.3. The variables are in ascending order number of movies ordered, number of movies delivered, number of movies payed for, number of bytes downloaded, number of bytes uploaded, upload/download-byte ratio, number of downloaded unicast-packages, number of unicast packages uploaded, upload/download-unicast ratio, hour (0-23), active and a Billing Fraud indicator. The variables are preprocessed to have mean zero and variance one.

As can be seen in figure 8.3, the third variable shows no payment when fraud occurs. We also see that the user of course is active (it is included for a purpose explained below). The two corners at the left in both the fraud and payment variable display fraud. Now look at variable six. It seems that fraud
occurs both when the number of bytes downloaded is mid-high and low. However, by multiplying the target, variable twelve, we may force the map to develop a description of the data which is more tied to fraud behavior. Hopefully this will reveal weaker correlation behavior. We know that a larger part of the map will place them self according to the fraud variable. The other dimensions of the prototype vectors will form a weighted mean, and so describe the mean value of the values in the corresponding input dimension, given the values of target, as well as the neighboring prototype values and so on. Variable six now says that the number of bytes when fraud is committed is medium, so a mean of the previous values is described in the prototype, see figure 8.4. This behavior is important, since we cannot from one map only tell what they describe. By changing the variances we get more information. In this case we did not gain more details about the target, but at other times such gain may result. The target can be neglected to a high degree. By giving it more power (variance), we gain in that the features found must be given by the behavior of the target to a higher degree. Tendencies involving variables with low magnitudes, may appear in a map if they get help to be heard- the map units neglect other samples than before variance adjustment. Of course one would like to further examine the clues obtained, but self-organizing feature maps are still effective in giving initial, although rough, clues to work with. When dimensionality is high, this can lighten the workload and bring the user closer to the data quicker.

We conclude by remarking that the features shown is given within the boundaries given by the samples. A constant dimension stays constant, as can be seen in variable eleven. This means that correlating behavior clues are only given if that variable allows it.

The maps were computed using the SOM Toolbox, which for use with Matlab can be obtained free at http://www.cis.hut.fi/projects/somtoolbox/.
8.7 Discussion
Anders Holst and Henrik Jacobsson

8.7.1 Mislabeled test data

As can be seen above, all methods fail to identify class (3), Excess download, in the test set. Most methods seemed to confuse them with class (2), Illegal redistribution. This was considered strange, since there is no similarity between those two fraud types.

After all the experiments had been completed it was discovered that all cases labeled as class (3) actually were generated as class (2) and then mislabeled as class (3). This explains the confusion of the learning systems. When compensating for this mislabeling in the result tables above, it turns out that most of the methods indeed manage to classify the frauds in the test set almost 100% correct.

8.7.2 Real versus simulated data

This task has many aspects in common with for example detecting and diagnosing faults or disturbances in an industrial process. Fraud is, just as a malfunction, hopefully relatively uncommon in the real world situation, at least compared to the vast amount of non-fraudulent cases. This means that even if the total amounts of data seem very large, the number of fraudulent cases may be quite small. Also, since there are so many different ways to commit a fraud, many of which may not even have been conceived of yet, each type of fraud may have occurred only one or a few times. This again is similar to process diagnosis, where the number of ways something can go wrong are almost unlimited, but most of the things that can go wrong has never occurred in the real data.

This is of course very fortunate for the industrial plant or system, but unfortunate for the learning systems. To be able to learn a fraud or a fault, there should be enough examples to allow for a statistical significant analysis of what the relevant features of it are. To provide the learning systems with enough data, it may then be necessary to use a simulator to produce this data. In the simulator any number of serious faults or any conceivable frauds can be produced, without disturbing the real system.

However, simulated data has its own problems. The whole point with the simulator is of course that it should look indistinguishable from real data, from the learning system’s point of view. Our experience is that in practice this is never the case. Building a simulator is in the first place a very complex and time consuming task, and simplifying assumptions have to be made all the time. For example, when simulating a process plant, physical entities like pressure and temperature may behave deterministically, like they would in an ideal world, and chemical reactions go with the speeds they ought to go. This means that the simulator is often deterministic – if started the same way the results will look exactly the same. The real world is not deterministic in the same way. Even if all input parameters are set exactly the same, there is a lot of noise. In a simulator the output of a variable should ideally be completely stable during steady state of the process, but in reality all measurements constantly vary, in the best case around a stable value. Also, sensors may not be perfect, but fluctuate somewhat or even drift, so some of the variation may come from the this. Further, dust and dirt may come into the process and disturb the ideal chemical circumstances. Clearly, a learning system trained on the ideal data from such a simulator would not recognize the situations encountered in the real world.

In the situation with frauds, it is even more complicated, since the behavior of human beings must be simulated. This can clearly not be done deterministically, but should be done with some statistical model. Constructing this statistical model to make it realistic is of course very difficult. Real data can be used to estimate the parameters of the model, but the hard task is to figure out which parameters to estimate. And then again, how should the parameters of fraudulent users be estimated, when the number of real cases to estimate them from is so small?

This fraud detection data illustrates these problems in a direct way. Note that, as mentioned above, writing a good realistic simulator is extremely hard, and the simulator here must in light of this be considered a good first step. Nevertheless, we found several discrepancies between the real and simulated data:
• Too homogeneous users.
   All users in the simulated data are quite similar, ordering about the same number of movies every week. In the real data there are of course several different user types, some almost never ordering any movies, to those ordering a lot, and those having periods of high activity and are passive in between. If we got it right, this property of the simulator is mainly due to the short time in which it had to be constructed, and it was originally planned that it should use several user profiles. Still, in the general case it is important to span up all possible behavior types if the learning systems should perform well.

• Too clean download and upload activities.
   In the simulated data all larger amounts of downloaded of data were due to movies, and all uploaded data due to illegal redistribution of movies. In the real data many users were constantly both uploading and downloading large amounts of data, even without ever ordering any movies. This simplification of course made it easier for the learning systems to spot illegal redistribution, but those same trained systems would not work well if given the real data.

• Differing characteristics of subscription fraud.
   In the real data there were a few examples of subscription frauds, seen as a large amount of ordered movies of some normal users, those being subjected to the fraud. In the simulated data on the other hand, this fraud was characterized by a large number of failed order requests, presumably trying to model the impostors failing attempts to log in as something else.

• Unphysical cases.
   At least one unrealistic sequence of events was found in the simulated data: According to the data, a user once tried to order a film, but this failed and no delivery notification was sent. Nevertheless, a billing notification was sent and the movie downloaded. This should not happen in reality if the order failed. This was however only seen once, and therefore it could be some error that has occurred somewhere else in the preprocessing, and not in the simulator. But something to watch out for, is that too imprecise statistical models in the simulator may with a small probability produce unphysical or unrealistic data.

More details about the simulated data and the hard problems involved can be found in [Lundin et al., 2001].

8.7.3 Summary

In summary, all results of the Telia project are promising. The systematic misclassifications are due to mistakes in the generation and preprocessing of data rather than due to the models themselves. The fact that the data was simulated does cast a shadow of doubt as to the value of the results, but it is important to remember that real world data would be on a similar format and contain similar characteristics. This is very promising for the future, as the training of the models was not the main problem.

8.8 References

Chapter 9

Ericsson:
Quality of Service in IP-networks

9.1 The task
Harald Brandt and Anders Holst

This task concerns Quality of Service in IP-networks, or in more detail to be able to describe and predict the delay and losses of data packets in the network. This is required for the network operators to be able to sell a given capacity with a given quality. The purpose of learning systems in this context is to replace difficult, expensive, and interfering measurements with approximations based on other less complicated measurements.

Since the kind of communication network of interest here, Differentiated Services, is currently under development and not used in commercial systems today, training data for the learning systems can not be collected from any real network. Instead, test equipment has to be built up in the lab, traffic generated, and measurements performed on this.

In the beginning of the project, routers were acquired and a testbed was built. Synthetic test traffic was generated and used for measurements on the test network. Unfortunately, the envisaged way to collect the measured performance statistics, required for the DALLAS project, failed. In spite of several attempts, and a large amount of man months, it was not possible to generate any useful set of data.

Since data for the learning systems could not be produced, this task of the DALLAS project unfortunately had to be aborted.
Chapter 10

Discussion and conclusions

Anders Holst, Daniel Gillblad and Björn Levin

The main objectives of the DALLAS project has been to spread knowledge about the potential and state-of-the-art of learning systems to the industrial partners; to provide the academic partners with real world data to evaluate the methods on; and to build a contact network for learning systems between industry and academia. The means to achieve this was to try to apply the learning systems methods used at the participating institute and universities to seven tasks provided by the industrial partners. Four tasks were treated by all academic partners, two tasks were treated by a subset of the academic partners, and for one of the tasks it was impossible to provide any data.

Although the tasks were from quite different domains, they had many things in common. For example, most of the tasks involved time series, and the need to find relations between these. Some new methodology were also developed in this context, like the mutual information rate measure [Gillblad and Holst, 2001]. There were in total three conference contributions produced within the DALLAS project [Gillblad and Holst, 2001; Johansson and Niklasson, 2001, 2002].

10.1 Evaluation of the project

The industrial partners have had slightly different objectives when joining the project, but the main common reason for participation was knowledge transfer. Although some partners have looked upon the project also as a potential supplier of an algorithm that works on a single problem, getting an overview of current learning systems techniques and their potential was the most important goal. The partners feel that this has generally worked very well. Not only have the industries been introduced to new methods that were only vaguely known to them before, but they have also had the opportunity to see the results when these methods are applied to familiar problems within their own domains.

The knowledge transfer from the industry to the academics has also been substantial. The availability of a large number of real data sets has made it possible to find and deal with practical limitations and problems of the used methods. It has also given a thorough understanding for the practical problems involved in data analysis tasks. One example is the quality of real data, which is never as clean and straightforward as filtered or simulated data. The degree to which it contains noise, missing values, anomalies, and outright errors, is always much higher than expected.

The project has also compared the main methods represented by each academic partner to industrial standard methods such as PLS (partial least squares, widely used in the process industry). This has been very useful and interesting not only for the industrial partners, but also for the academics, since these methods are sometimes overlooked when working mainly with the development of ones own methods.

To summarize, the DALLAS project has given the participants, both industrial and academic, a good overview of methods and their strengths and limitations. The opportunity to see the methods being
tested on problems outside ones own specific domain has also increased everyones understanding of the methods and their possible applications.

Another very important effect of the DALLAS project is that it has created a lot of contacts between the industrial and academic partners. When one of the industrial partners face a problem that might benefit from the use of one of the methods presented in the DALLAS project or just a question about it, they can contact the academic partner that represented that method within the project.

The general agreement among the partners is that the DALLAS project has worked very well for this type of cooperation project, and it has even exceeded the expectations of several partners who have participated in similar projects before.

10.2 Comments on the project form

In a cooperation project like this, all partners must be prepared for the time needed to understand the different traditions in thinking and in how to express things. Early in the project the academic partners were not good at providing method descriptions on the appropriate level. This made it difficult for the industrial partners to get into the project in the beginning, and also to motivate participation in the project internally. This was considerably improved during the course of the project. It is also important that the companies reserve the time needed to absorb the results. Most companies in the DALLAS project did this very well.

Important is also how to transfer algorithms and software from the academic partners to the industry. It is clear that industry wants to and could benefit from being able to run the methods tested in the project themselves, on other kinds of data or new data sets. This is very difficult, though, since most software used by the academic partners are in the form of research prototypes, with very limited data import abilities and low flexibility. This kind of tools is necessary for method development, but not of very much use to the industry. The issue of who should adapt the research prototypes and create tools that are ready for industrial use must be considered in the future.

The ambition in the DALLAS project was to investigate the performance of a large number of algorithms on a large number of tasks. This of course resulted in very limited time for each academic partner to spend on each of the tasks. Because of this it was sometimes not possible to go as deep into each task as would have been preferred. Also, the focus had to be on applying existing algorithms rather than developing new ones.

The scheme with one academic partner responsible for each dataset worked out really well. It established close contacts between the academic partners and the industrial partners and made it easier to handle the sometimes large and complicated data sets.

Generally, the industrial partners were pleased with the ambition of the academic partners and the academic partners appreciated the commitment and interest from the industrial partners.

10.3 Conclusions

The main conclusions are perhaps two things that has been said many times before, but which should not be underestimated.

The first is that in all data analysis projects, the data collection, preprocessing, and the getting to know the domain and what the problem really is about, always takes a huge and by far the largest effort. It is also an iterative process that must be cycled a few turns before it gets right. Things that may happen is that something goes wrong with the data files, that there are too few effective data samples, or that data is selected in an unfortunate way. After a few iterations with new data sets it may also suddenly turn out that the initial formulation of the problem is not a good one, and that it has to be reformulated. In summary, a lot can go wrong with the data, and it is important not to give up at this
stage. Although all participants knew at the start that this stage would take a lot of time, everyone was still surprised to see that it took even longer than expected.

The second conclusion is that, once these initial obstacles are overcome, often the exact choice of which learning system to use is not so important since many methods perform roughly the same. The hardest part is the preprocessing, and once this has turned the data into something reasonable, it may suffice with rather simple methods to solve the real task. This also means that it often turned out that once the preprocessing was done, the results of a linear model was close to those of the non-linear models. This is true partly because, with the limited amounts of independent data, the number of free parameters had to be kept low. This in turn prevented too complicated models from being used.

Despite the above, some differences in the performance of the methods can still be seen. In for example the EKA case, two non-linear neural network models outperformed the other models, specifically the linear ones. Note also that there are many different kinds of linear models, of which for example partial least squares, Perceptron neural networks without hidden layers, and the naive Bayesian classifier, has been used in this project. Even when a linear model is sufficient to solve a task, all linear models do not perform the same. For example the naive Bayesian classifier performed somewhat better than PLS on the AstraZeneca task.

Thus, there is no single method that can be said to perform significantly better than all the rest for all problems. Some tendencies could be discerned however: the Bayesian methods worked better when there were extremely few data samples as in the AstraZeneca and NMA cases; the artificial neural networks worked better when there were large amounts of data and the nonlinearities could not be ignored as for EKA and SCA; and inductive logic methods worked best when there were a relatively small number of variables and a small number of classes as for Telia.

In conclusion, we feel that this has been a successful project, from which we have all gained valuable experience. It has given the academic partners a chance to test their methods on real data, and given important experience of working with industry. The industrial partners have learned about the potential of learning systems, and have got tasks relevant to them thoroughly analyzed. Finally, many useful contacts for possible future cooperation have been established.

### 10.4 Publications made during the project


