Classification of Repeated Measurement Data Using Growth Curves and Neural Networks

Department of Mathematics, Linköping University

Kasper Andersson

LiTH-MAT-EX–2022/23–SE

Credits:	**30hp**
Level:	**A**
Supervisor:	**Martin Singull** Department of Mathematics, Linköping University
Examiner:	**Dietrich von Rosen** Department of Mathematics, Linköping University
Linköping:	**2022**

Abstract

This thesis focuses on statistical and machine learning methods designed for sequential and repeated measurement data. We start off by considering the classic general linear model (MANOVA) followed by its generalization, the growth curve model (GMANOVA), designed for analysis of repeated measurement data. By considering a binary classification problem of normal data together with the corresponding maximum likelihood estimators for the growth curve model, we demonstrate how a classification rule based on linear discriminant analysis can be derived which can be used for repeated measurement data in a meaningful way.

We proceed to the topics of neural networks which serve as our second method of classification. The reader is introduced to classic neural networks and relevant subtopics are discussed. We present a generalization of the classic neural network model to the recurrent neural network model and the LSTM model which are designed for sequential data.

Lastly, we present three types of data sets with an total of eight cases where the discussed classification methods are tested.

Keywords:

Repeated Measurement Data, Sequential Data, Growth Curve Model, Linear Discriminant Analysis, Neural Network, Recurrent Neural Network, LSTM

URL for electronic version:

-

Sammanfattning

Den här uppsatsen introducerar klassificeringsmetoder skapade för data av typen upprepade mätningar och sekventiell data. Den klassiska MANOVA modellen introduceras först som en grund för den mer allmäna tillväxtkurvemodellen (GMANOVA), som i sin tur används för att modellera upprepade mätningar på ett meningsfullt sätt. Under antagandet av normalfördelad data så härleds en binär klassificeringsmetod baserad på linjär diskriminantanalys, som tillsammans med maximum likelihood-skattningar från tillväxtkurvemodellen ger en binär klassificeringsregel för data av typen upprepade mätningarn.

Vi fortsätter med att introducera läsaren för klassiska neurala nätverk och relevanta ämnen diskuteras. Vi generaliserar teorin kring neurala nätverk till typen "recurrent" neurala nätverk och LSTM som är designade för sekventiell data.

Avslutningsvis så testas klassificeringsmetoderna på tre typer av data i totalt åtta olika fall.

Nyckelord:
Upprepade Mätningar, Sekventiell Data, Tillväxtkurvor, Linjär Diskriminantanalys, Neurala Nätverk, "Recurrent" Neurala Nätverk, LSTM

URL för elektronisk version:
-

Acknowledgements

First of, I want to thank my supervisor Martin Singull for giving me the opportunity to write about this topic.

I also want thank David for all the gym sessions, Filip for all the group projects and Axel for all the "kaffepauser".

Then, finally, I want to thank my mother for all the support I got throughout my studies.

Contents

Chapter 1

Introduction

This chapter intend to give the reader a brief introduction of this thesis. We give a short summary of each chapter which hopefully further encourage the reader to take interest in the relevant topics.

1.1 Outline of the thesis

Chapter 1 introduces the reader to each topic of this thesis and give a brief introduction to the topic of sequential and repeated measurements data.

Chapter 2 serves as a collection of relevant definitions and theorems which will be applied in Chapter 3, where the statistical algorithms are presented.

Chapter 3 is the main chapter of this thesis and serves as a collection of all applied statistical algorithms. First of, the reader is introduced to the general linear model which is a multivariate generalization of the classic multiple linear regression model. We give a short introduction to some of the statistical models which are incorporated in the general linear model and how they are used to perform statistical analysis. Then we further generalize the general linear model to the growth curve model which is used to study sequential data characterized as repeated measurements. We then proceed to derive our first classification method based on linear discriminant analysis, where we obtain a binary classification rule based on estimated growth curves. Then, we introduce the reader to neural networks, some of their properties and how to further develop neural networks to contain a recurrent structure designed for sequential data.

Chapter 4 collects all of the results from applying the statistical algorithms. The reader is presented to five cases where the statistical algorithms has been used and compared on different data sets. The first three cases uses simulated repeated measurement data while the last two cases uses temperature data where we deviate from the design of repeated measurements.

Chapter 5 gives a short review of what we have done and discusses how one can further develop the presented statistical algorithms. We focus on how to develop the growth curve model to work on more general problems than those presented in chapter 3 and 4.

1.2 Purpose of this Thesis

Statistical analysis of sequential data differs from the one on non-sequential data by the fact that order of observation matters [6]. In practice, this means that one cannot assume independence between observations in the data, that is

$$p(x_{i+1}|x_i) \neq p(x_{i+1})$$

for some probability distribution p. Two examples of this is weather temperature observed once per day and stock prices updated each time a trade agreement has been fulfilled. Another example is medical data of child growth where multiple children has been observed and studied at the same age during their youth so that potential outliers can be observed early on and helped. When we have multiple subjects observed at the same time points, eventually split into different groups, we refer to this as repeated measurements design, or simply repeated measurements.

This thesis aims to introduce some classic classification algorithms and further generalize these algorithms so that repeated measurements can be studied in a meaningful way.

Chapter 2

Theory

Although most theory will be presented when it's needed to give the reader a complete view of how to apply the statistical algorithms which will be discussed later, some mathematics would remove focus from relevant theory if it were to be presented together with the statistical algorithms. This chapter will serve as a collection for mathematical tools falling under this category and the chapter is split into two parts. One part where pure mathematics is presented and another part where relevant statistical definitions are collected.

2.1 Mathematical Theory

Definition 2.1. *[9] The **Kronecker product** between two matrices $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{m \times n}, \mathbf{B} = (b_{ij}) \in \mathbb{R}^{p \times q}$ is denoted $\mathbf{A} \otimes \mathbf{B}$ and is defined by the block matrix*

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \dots & a_{mn}\mathbf{B} \end{pmatrix} \in \mathbb{R}^{mp \times nq}.$$

The Kronecker product between two matrices can be seen as a generalization of the outer product between two vectors and will be utilized when we introduce the matrix normal distribution for the multivariate models.

Definition 2.2. *[9] The **Hadamard product** between two matrices $\mathbf{A} = (a_{ij})$ and $\mathbf{B} = (b_{ij})$, both in $\mathbb{R}^{m \times n}$, is denoted $\mathbf{A} \odot \mathbf{B}$ and is defined by the matrix*

$$\mathbf{A} \odot \mathbf{B} = \begin{pmatrix} a_{11}b_{11} & \dots & a_{1n}b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & \dots & a_{mn}b_{mn} \end{pmatrix} \in \mathbb{R}^{m \times n}.$$

The Hadamard product, often referred to as the element-wise matrix product, is of practical usage when data needs to be processed with separate routines but merged for a final result. It will be used when we discuss optimization methods relevant for the neural networks.

Definition 2.3. *The* ***vectorization*** *of a matrix* $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_k) \in \mathbb{R}^{r \times k}$ *is denoted* $\text{vec}(\mathbf{A})$ *and defined by the column vector*

$$\text{vec}(\mathbf{A}) = \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_k \end{pmatrix} \in \mathbb{R}^{rk}.$$

The vectorization of a matrix, together with the Kronecker product, will be convenient to use when we consider the matrix normal distribution and how it relates to the multivariate normal distribution.

Theorem 2.1 (**Extended Cauchy-Schwarz Inequality**). *[13] Let* $\mathbf{x} : p \times 1$ *and* $\mathbf{d} : p \times 1$ *be vectors and let* $\mathbf{B}$ *be a positive definite matrix. Then*

$$(\mathbf{x}'\mathbf{d})^2 \leqslant (\mathbf{x}'\mathbf{B}\mathbf{x})(\mathbf{d}'\mathbf{B}^{-1}\mathbf{d}) \tag{2.1}$$

with equality if and only if $\mathbf{x} = c\mathbf{B}^{-1}\mathbf{d}$ *or* $\mathbf{d} = c\mathbf{B}\mathbf{x}$ *for some constant* c.

Corollary 2.1.1. *Under the same conditions as in Theorem 2.1 we have*

$$\max_{\mathbf{x} \neq 0} \frac{(\mathbf{x}'\mathbf{d})^2}{\mathbf{x}'\mathbf{B}\mathbf{x}} = \mathbf{d}'\mathbf{B}^{-1}\mathbf{d} \tag{2.2}$$

Proof: The maximum is attained by inserting $\mathbf{x} = c\mathbf{B}^{-1}\mathbf{d}$ and expanding.

Remark: Due to the fact that c can be any constant, we can, without loss of generalization, simply ignore it by setting $c = 1$ when applying the result.

2.2 Statistical Theory

Definition 2.4. *A random variable* X *is* ***normal*** *(Gaussian)* ***distributed****, with mean* μ *and variance* σ^2 *if its pdf is given by*

$$f(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in (-\infty, \infty) \tag{2.3}$$

and we denote this by $X \sim N(\mu, \sigma^2)$. *For* $X \sim N(0, 1)$ *we denote* $P(X < c)$ *as* $\Phi(c)$.

A natural generalization of Definition 2.4 is the *multivariate normal distribution*, which occurs when we treat multiple normal variables at the same time.

Definition 2.5. *A random vector* $\mathbf{x} : p \times 1$ *is* ***multivariate normal distributed****, with mean* $\boldsymbol{\mu} : p \times 1$ *and positive definite covariance matrix* $\boldsymbol{\Sigma} : p \times p$ *if its pdf is given by*

$$f(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-p/2}|\boldsymbol{\Sigma}|^{-p/2} exp\left(-\frac{(\mathbf{x}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}{2}\right) \tag{2.4}$$

and we denote this by $\mathbf{x} \sim N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$*.*

One can further generalize Definition 2.5 to the so called *matrix normal distribution.*

Definition 2.6. *A random matrix* $\mathbf{X} : p \times n$ *is* ***matrix normal distributed****, with mean* $\mathbf{M} : p \times n$*, positive definite row-covariance matrix* $\boldsymbol{\Sigma} : p \times p$ *and positive definite column-covariance matrix* $\boldsymbol{\Psi} : n \times n$ *if its pdf is given by*

$$f(\mathbf{X}|\mathbf{M}, \boldsymbol{\Sigma}, \boldsymbol{\Psi}) = c_{p,n} exp\left(-\frac{1}{2}\operatorname{tr}\left(\boldsymbol{\Sigma}^{-1}(\mathbf{X}-\mathbf{M})\boldsymbol{\Psi}^{-1}(\mathbf{X}-\mathbf{M})'\right)\right) \tag{2.5}$$

where $c_{p,n} = (2\pi)^{-pn/2}|\boldsymbol{\Sigma}|^{-n/2}|\boldsymbol{\Psi}|^{-p/2}$*,* $\operatorname{tr}(\cdot)$ *is the trace and we denote this distribution by* $\mathbf{X} \sim N_{p,n}(\mathbf{M}, \boldsymbol{\Sigma}, \boldsymbol{\Psi})$*.*

Remark 1: We can define the matrix normal distribution for a matrix $\mathbf{X} \sim N_{p,n}(\mathbf{M}, \boldsymbol{\Sigma}, \boldsymbol{\Psi})$ with Definition 2.5 by considering its vectorization $\operatorname{vec}(\mathbf{X}) \sim N_{pn}(\operatorname{vec}(\mathbf{M}), \boldsymbol{\Psi} \otimes \boldsymbol{\Sigma})$.

Remark 2: As the names suggest, the row-covariance matrix $\boldsymbol{\Sigma}$ and the column-covariance matrix $\boldsymbol{\Psi}$ in Definition 2.6 characterizes the covariance for the p rows and n columns respectively for a corresponding matrix $\mathbf{X} \sim N_{p,n}(\mathbf{M}, \boldsymbol{\Sigma}, \boldsymbol{\Psi})$. Depending on the literature we are referring to, we might choose to model $\mathbf{X}$ as $n \times p$ and the corresponding distribution would be $\mathbf{X} \sim N_{n,p}(\mathbf{M}', \boldsymbol{\Psi}, \boldsymbol{\Sigma})$ where the matrix dimensions would change accordingly.

Example 2.1. A common statistical model usually contains n independent p-dimensional observations $\mathbf{x}_i, i = 1, \dots, n$ under the assumptions of equal mean, equal covariance (homoscedasticity) and normal distribution. If we choose to put our observations as columns in a data matrix $\mathbf{X} : p \times n$, this could be modeled as $\mathbf{X} \sim N_{p,n}(\boldsymbol{\mu}\mathbf{1}_n', \boldsymbol{\Sigma}, \mathbf{I}_n)$ or $\operatorname{vec}(\mathbf{X}) \sim N_{pn}(\mathbf{1}_n \otimes \boldsymbol{\mu}, \mathbf{I}_n \otimes \boldsymbol{\Sigma})$ where $\mathbf{1}_n : n \times 1$ is a vector filled with ones and $\mathbf{I}_n : n \times n$ is the identity matrix. If we instead choose to put our observations as rows, the corresponding model would be given by $\mathbf{X} : n \times p$, $\mathbf{X} \sim N_{n,p}(\mathbf{1}_n\boldsymbol{\mu}', \mathbf{I}_n, \boldsymbol{\Sigma})$ or $\operatorname{vec}(\mathbf{X}) \sim N_{np}(\boldsymbol{\mu} \otimes \mathbf{1}_n, \boldsymbol{\Sigma} \otimes \mathbf{I}_n)$.

Chapter 3

Statistical Learning

3.1 The General Linear Model

This section is devoted to statistical models that can be written on the form

$$\mathbf{Y} = \mathbf{X}\mathbf{B} + \mathbf{E}, \tag{3.1}$$

where $\mathbf{Y} : n \times p$, $\mathbf{X} : n \times (q+1)$, $\mathbf{B} : (q+1) \times p$ and $\mathbf{E} \sim N_{n,p}(\mathbf{0}, \mathbf{I}_n, \mathbf{\Sigma})$. The matrices $\mathbf{B}$ and $\mathbf{\Sigma}$ are unknown parameter matrices, $\mathbf{X}$ is a known design matrix and we assume that $n > (q+1) + p$. The model (3.1) is known as the general linear model [26] and we devote the following two subsections to represent some of the statistical models included in it. We finish the section with a test for making inference about (3.1) before we move forward to the more general GMANOVA model.

3.1.1 A Univariate Linear Model

Consider (3.1) but with only one response variable ($p = 1$). We refer to this as the univariate linear model, which we denote as

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{e} \tag{3.2}$$

where $\mathbf{y} = (Y_1, \ldots, Y_n)'$ is the response vector, $\mathbf{X} = (\mathbf{x}_1', \ldots, \mathbf{x}_n')'$ is the design matrix, $\mathbf{b}$ is the parameter vector and $\mathbf{e} \sim N_n(0, \sigma^2\mathbf{I}_n)$ is the random error. Alternatively, we might write this as $Y_i \sim N(\mathbf{x}_i\mathbf{b}, \sigma^2)$, $i = 1, \ldots, n$. We give a short review of two types of statistical models represented by (3.2), namely, multiple linear regression and the one factor ANOVA.

Multiple Linear Regression

Now consider (3.2) where all input variables are continuous. The first column of the design matrix $\mathbf{X}$ is filled with ones corresponding to an intercept term and all other elements corresponds to observations of our input variables. This gives us the multiple linear regression model

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1q} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{nq} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_q \end{pmatrix} + \begin{pmatrix} E_1 \\ \vdots \\ E_n \end{pmatrix},$$

where $E_i \sim N(0, \sigma^2), i = 1, \dots, n$. Our goal with multiple linear regression is to fit a hyperplane in $\mathbb{R}^q$ used for predicting new values in $\mathbb{R}$, i.e., our estimated model will be on the form of

$$\hat{y} = \hat{b}_0 + \hat{b}_1 x_1 + \dots + \hat{b}_q x_q$$

or simply $\mathbf{x}'\hat{\mathbf{b}}$, where $\hat{\mathbf{b}}$ is our estimator of $\mathbf{b}$. By minimizing the residual $\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}$ one can show that the best estimator $\hat{\mathbf{b}}$ in terms of least squares is given by $(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ and an unbiased estimator of σ^2 is given by

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \mathbf{X}\hat{\mathbf{b}})'(\mathbf{y} - \mathbf{X}\hat{\mathbf{b}})}{n - q - 1} = \frac{\mathbf{y}'(\mathbf{I} - \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}')\mathbf{y}}{n - q - 1}$$

where $(\mathbf{X}'\mathbf{X})^{-1}$ can be assumed to exist as we are working under the condition of $n > q + 1$, which ensures that $\mathbf{X}'\mathbf{X}$ is of full rank [13].

To validate the use of our estimated linear equation $\mathbf{x}'\hat{\mathbf{b}}$ in order to predict our response variable y we consider the hypothesis

$$\begin{gathered} H_0 : b_1 = \dots = b_q = 0 \\ \text{vs.} \\ H_A : \text{At least one } b_i \neq 0. \end{gathered} \tag{3.3}$$

ANOVA

Analysis of variance (ANOVA) is the study of statistical methods for comparing means of groups of continuous observations where the groups are defined by levels of factors [2]. Again consider (3.2) but this time with all of our predictors as categorical and that all our variables in $\mathbf{X}$ are dummy variables.

In the one factor ANOVA model we have responses $Y_{ij} = \mu_i + e_{ij} = \mu + \tau_i + e_{ij}$ together with a sum-to-zero constraint $\sum_{i=1}^{g} \tau_i = 0$ to handle the overparameterization introduced by including τ_i, where τ_i is our non-random but unknown factor for group $i = 1, \ldots, g$, $e_{ij} \sim N(0, \sigma^2)$ and we want to test

$$\begin{gathered} H_0 : \mu_1 = \ldots = \mu_g \\ \text{vs.} \\ H_A : \text{At least one of the group means differ from the other} \end{gathered}$$

or equivalently,

$$\begin{gathered} H_0 : \tau_1 = \ldots = \tau_g = 0 \\ \text{vs.} \\ H_A : \text{At least one } \tau_i \neq 0. \end{gathered} \tag{3.4}$$

The one-factor model can be interpreted as that we have g treatments that we want to test, n_i subjects per group and one treatment per group. Before the treatments take place we assume that every subject has the same mean value μ and a random error e_{ij}. The question we are interested in is if the treatments have effect on the subjects, or equivalently, if the group means differ between the groups after the treatment, i.e., is one of the $\tau_i \neq 0$.

3.1.2 The General Linear Model

We now generalize the theory about ANOVA and multiple linear regression by considering the full general linear model by relaxing the condition $p = 1$. In practice this means that we allow more than one response variable per sample in our regression model, and that we have multivariate factors in our ANOVA model.

The density for the general linear model is given by

$$f(\mathbf{Y}|\mathbf{B}, \mathbf{\Sigma}) = (2\pi)^{-\frac{np}{2}} |\mathbf{\Sigma}|^{-\frac{n}{2}} e^{-\frac{1}{2}\operatorname{tr}\left((\mathbf{Y}-\mathbf{XB})\mathbf{\Sigma}^{-1}(\mathbf{Y}-\mathbf{XB})'\right)} \tag{3.5}$$

and under the assumption $n > p + (q + 1)$, which ensures that $(\mathbf{X}'\mathbf{X})^{-1}$ is well defined, it can be shown that

$$\widehat{\mathbf{B}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y},$$
$$\widehat{\mathbf{\Sigma}} = \frac{1}{n}(\mathbf{Y} - \mathbf{X}\widehat{\mathbf{B}})'(\mathbf{Y} - \mathbf{X}\widehat{\mathbf{B}})$$

are the corresponding maximum likelihood estimators (MLE's) [13].

Multivariate Linear Regression

We consider the multiple linear regression model but this time with p response variables per sample. The multivariate linear regression [13] model is given by

$$\begin{pmatrix} Y_{11} & Y_{12} & \dots & Y_{1p} \\ Y_{21} & Y_{12} & \dots & Y_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{n1} & Y_{n2} & \dots & Y_{np} \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1q} \\ 1 & x_{21} & \dots & x_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{nq} \end{pmatrix} \begin{pmatrix} b_{01} & b_{02} & \dots & b_{0p} \\ b_{11} & b_{12} & \dots & b_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{q1} & b_{q2} & \dots & b_{qp} \end{pmatrix} + \begin{pmatrix} E_{11} & E_{12} & \dots & E_{1p} \\ E_{21} & E_{12} & \dots & E_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ E_{n1} & E_{n2} & \dots & E_{np} \end{pmatrix}$$

or more compactly written

$$\mathbf{Y} = \mathbf{XB} + \mathbf{E}.$$

For every column in $\mathbf{Y}$ we have the linear relationship

$$\mathbf{y}_i = \mathbf{X}\mathbf{b}_i + \mathbf{e}_i \in \mathbb{R}^{n\times 1}$$

where $\mathbf{b}_i$ and $\mathbf{e}_i$ is the i:th column of $\mathbf{B}$ and $\mathbf{E}$ respectively. The covariance for the error terms $\mathbf{e}_i$ are given by $\text{cov}(\mathbf{e}_i, \mathbf{e}_j) = \sigma_{ij}\mathbf{I}_n$ and for the rows $\mathbf{e}^{(i)}$ of $\mathbf{E}$ it holds that $\text{cov}(\mathbf{e}^{(i)}) = \mathbf{\Sigma}$ and $\text{cov}(\mathbf{e}^{(i)}, \mathbf{e}^{(j)}) = \mathbf{0}$ when $i \neq j$.

Shortly, these covariances means that only the variables from the same subject are correlated, the variation and correlation for the predictor variables $x_1, \dots x_q$ is given by $\mathbf{\Sigma}$ and that different samples are independent.

Just as for the multiple linear regression model we often want to test if the coefficients are significant. The difference now is that we are testing the columns $\mathbf{b}_i$, $i = 1, \dots, p$ of $\mathbf{B}$ instead of scalar coefficients as in (3.3). The standard hypothesis to check if any of the predictor variables are of use is given by

$$\begin{gathered} H_0 : \mathbf{b}_1 = \dots = \mathbf{b}_p = 0 \\ \text{vs.} \\ H_A : \text{At least one } \mathbf{b}_i \neq 0. \end{gathered} \tag{3.6}$$

MANOVA

We consider the same setup as we had in the univariate ANOVA but now we generalize it to the multivariate setting. In the one factor MANOVA model we have responses $\mathbf{Y}_{ij} = \boldsymbol{\mu}_i + \mathbf{e}_{ij} = \boldsymbol{\mu} + \boldsymbol{\tau}_i + \mathbf{e}_{ij}$ together with a sum-to-zero constraint $\sum_{i=1}^{g} \boldsymbol{\tau}_i = \mathbf{0}$, where $\boldsymbol{\tau}_i$ is our non-random but unknown factor and $\mathbf{e}_{ij} \sim N_p(\mathbf{0}, \boldsymbol{\Sigma})$ and we want to test

$$\begin{gathered} H_0 : \boldsymbol{\tau}_1 = \ldots = \boldsymbol{\tau}_g = 0 \\ \text{vs.} \\ H_A : \text{At least one } \boldsymbol{\tau}_i \neq 0. \end{gathered} \tag{3.7}$$

3.1.3 The General Hypothesis

We now present a general hypothesis for the four tests (3.3), (3.4), (3.6) and (3.7). Since all of the models can be written in the form of the general linear model, we can formulate a general hypothesis for all of them instead of treating them one by one. The hypothesis of interest is given by

$$\begin{gathered} H_0 : \mathbf{CB} = \mathbf{0} \\ \text{vs.} \\ H_A : \mathbf{CB} \neq \mathbf{0}, \end{gathered} \tag{3.8}$$

where $\mathbf{C} : m \times (q+1)$ is known and $m \leqslant q$. For instance, if we want to test (3.6), then $\mathbf{C}$ will take the form of an q-dimensional identity matrix but with an additional column $\mathbf{0} : m \times 1$ at start, that is

$$\mathbf{C} = [\mathbf{0}, \mathbf{I}_q] = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix}. \tag{3.9}$$

We can test (3.8) by considering

$$\begin{aligned} \mathbf{V} &= n\widehat{\boldsymbol{\Sigma}} = \mathbf{Y}'(\mathbf{I} - \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}')\mathbf{Y}, \\ \mathbf{W} &= \widehat{\mathbf{B}}'\mathbf{C}'(\mathbf{C}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{C}')^{-1}\mathbf{C}\widehat{\mathbf{B}}, \end{aligned}$$

where $\mathbf{V}$ is the sum of squares under the full model and $\mathbf{W}$ is the sum of squares under H_0. The likelihood ratio test for (3.8) is based on the statistic

$$\Lambda = \frac{|\mathbf{V}|}{|\mathbf{V} + \mathbf{W}|}$$

and the asymptotic distribution for Λ is approximately given by

$$\begin{aligned} &P\left(-(n - q - \frac{1}{2}(p - m + 1)) \ln \Lambda \geqslant z\right) \\ &\approx P\left(\chi^2(f) \geqslant z\right) + \frac{\gamma}{\nu^2}\left(P\left(\chi^2(f+4) \geqslant z\right) - P\left(\chi^2(f) \geqslant z\right)\right) \end{aligned} \tag{3.10}$$

where z equals to the observed valued of $-(n - q - \frac{1}{2}(p - m + 1)) \ln \Lambda$, $f = pm, \gamma = \frac{pm(p^2+m^2-5)}{48}$ and $\nu = n - \frac{p-m+1}{2}$ [21] [24]. This means that we reject (3.8) at a significance level α if (3.10) is less than α.

3.2 The Growth Curve Model

We are now going to further generalize the general linear model (3.1) by including time regressors in the model, which will be used to model repeated measurement data.

The growth curve model, also known as the generalized multivariate analysis of variance model [23] (GMANOVA) or the bilinear regression model [26], is given by

$$\mathbf{Y} = \mathbf{ABC} + \mathbf{E}, \tag{3.11}$$

where $\mathbf{Y} : p \times n$ is the response matrix, $\mathbf{A} : p \times q$ is the within-group design matrix, $\mathbf{B} : q \times k$ is the parameter matrix, $\mathbf{C} : k \times n$ is the between-group design matrix and $\mathbf{E} \sim N_{p,n}(\mathbf{0}, \mathbf{\Sigma}, \mathbf{I}_n)$ is the random error. We also assume that $q \leqslant p$, $\text{rank}(\mathbf{C}) + p \leqslant n$ and $\mathbf{\Sigma} : p \times p$ to be positive definite. Note that we now have transposed our model compared to (3.1), i.e., the columns in $\mathbf{Y}$ are now the individual p-dimensional responses and the columns of $\mathbf{E}$ are now independent. We have also replaced the notation of our design matrix $\mathbf{X}$ to $\mathbf{C}$. This is done to be consistent with the literature we are working with.

The standard set up is that p is the number of time points we have observed our n samples and we assume that the mean value μ_i for each of the k groups is a polynomial in time t of degree $q - 1$, that is

$$\mu_i = b_{1i} + b_{2i}t + \ldots + b_{qi}t^{q-1}, \quad i = 1, \ldots, k. \tag{3.12}$$

If we compare (3.11) to the general linear model in the previous section we can see that the main difference, besides that we have transposed our model and changed index notation, is that we have post-multiplied (3.1) with the within-group design matrix $\mathbf{A}$, which takes the form

$$\mathbf{A} = \begin{pmatrix} 1 & t_1 & \dots & t_1^{q-1} \\ 1 & t_2 & \dots & t_2^{q-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_p & \dots & t_p^{q-1} \end{pmatrix} \tag{3.13}$$

to model the 3.12. Furthermore, we have that $\mathbf{Y} \sim N_{p,n}(\mathbf{ABC}, \mathbf{\Sigma}, \mathbf{I}_n)$, so the density function for the growth curve model is given by

$$f(\mathbf{Y}|\mathbf{B}, \mathbf{\Sigma}) = (2\pi)^{-\frac{pn}{2}} |\mathbf{\Sigma}|^{-\frac{n}{2}} e^{-\frac{1}{2}\mathrm{tr}\left(\mathbf{\Sigma}^{-1}(\mathbf{Y}-\mathbf{ABC})(\mathbf{Y}-\mathbf{ABC})'\right)}.$$

It can be shown that if $\mathbf{A}$ and $\mathbf{C}$ are of full rank (q and k), then the MLE's are given by

$$\widehat{\mathbf{B}} = (\mathbf{A}'\mathbf{S}^{-1}\mathbf{A})^{-1}\mathbf{A}'\mathbf{S}^{-1}\mathbf{Y}\mathbf{C}'(\mathbf{C}\mathbf{C}')^{-1}, \tag{3.14}$$

$$\widehat{\mathbf{\Sigma}} = \frac{1}{n}(\mathbf{Y} - \mathbf{A}\widehat{\mathbf{B}}\mathbf{C})(\mathbf{Y} - \mathbf{A}\widehat{\mathbf{B}}\mathbf{C})', \tag{3.15}$$

where $\mathbf{S} = \mathbf{Y}(\mathbf{I} - \mathbf{C}'(\mathbf{C}\mathbf{C}')^{-1}\mathbf{C})\mathbf{Y}'$ [16].

The general hypothesis for the GMANOVA model is given by

$$\begin{gathered} H_0 : \mathbf{GBH} = \mathbf{0} \\ \text{vs.} \\ H_A : \mathbf{GBH} \neq \mathbf{0}, \end{gathered} \tag{3.16}$$

where $\mathbf{G} : r \times q, r \leqslant q, \mathbf{B} : q \times k$ and $\mathbf{H} : k \times t, t \leqslant q$. Under the assumption of $\mathbf{G}$ and $\mathbf{H}$ having full rank, we can use the statistic

$$\Lambda = \frac{|\mathbf{G}(\mathbf{A}'\mathbf{S}^{-1}\mathbf{A})^{-1}\mathbf{G}'|}{|\mathbf{G}(\mathbf{A}'\mathbf{S}^{-1}\mathbf{A})^{-1}\mathbf{G}' + \mathbf{G}\widehat{\mathbf{B}}\mathbf{H}(\mathbf{H}'\mathbf{R}\mathbf{H})^{-1}\mathbf{H}'\widehat{\mathbf{B}}'\mathbf{G}'|},$$

where

$$\begin{aligned} \mathbf{R} &= (\mathbf{C}\mathbf{C}')^{-1} + (\mathbf{C}\mathbf{C}')^{-1}\mathbf{Y}'\left(\mathbf{S}^{-1} - \mathbf{S}^{-1}\mathbf{A}(\mathbf{A}'\mathbf{S}^{-1}\mathbf{A})^{-1}\mathbf{A}'\mathbf{V}^{-1}\right)\mathbf{Y}\mathbf{C}'(\mathbf{C}\mathbf{C}')^{-1}, \\ \mathbf{S} &= \mathbf{Y}(\mathbf{I} - \mathbf{C}'(\mathbf{C}\mathbf{C}')\mathbf{C})\mathbf{Y}', \end{aligned}$$

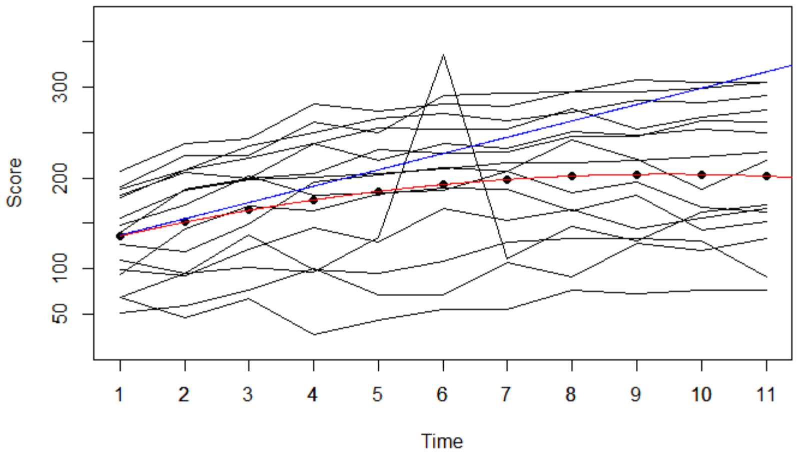


Figure 3.1: Samples for the patients in treatment group 75-100r (black curves), mean value at each time (black circles) together with two growth curves fitted, one with a quadratic time regressor included (red) and one without (blue). The quadratic growth curve matches the mean value almost perfectly throughout the whole treatment while the linear growth curve deviate noticeable already at day 2 ($t = 3$).

to test (3.16) [14]. By defining T as

$$T = -(n - k + q - p - \frac{1}{2}(r - t + 1)) \ln \Lambda, \tag{3.17}$$

one can show that the asymptotic distribution for T is approximately $\chi^2(rt)$ [14] which means that for large n, we reject H_0 in (3.16) at a significance level α if

$$T > \chi^2_\alpha(rt). \tag{3.18}$$

Example 3.1. Danford et al. [1] analyzed the results of patients being exposed to different levels of radiation, see Table 3.1. By studying the mean value of the third group (75-100r) we can observe a small quadratic growth over time which is illustrated in Figure 3.1. It may thus be interesting to test if a quadratic term is needed based on all of the available data. Using (3.11) gives us

$$\mathbf{Y} = \left(\boldsymbol{y}_1 \dots \boldsymbol{y}_{45}\right), \quad \mathbf{A} = \begin{pmatrix} 1 & t_0 & t_0^2 \\ & \vdots & \\ 1 & t_{10} & t_{10}^2 \end{pmatrix},$$

$$\mathbf{B} = \begin{pmatrix} b_{01} & b_{02} & b_{03} & b_{04} \\ b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} \mathbf{1}_6' & & & \\ & \mathbf{1}_{14}' & & \\ & & \mathbf{1}_{16}' & \\ & & & \mathbf{1}_{10}' \end{pmatrix}.$$

The hypothesis of interest is to test if the last row of $\mathbf{B}$ is equal to zero. This can be done using (3.16) with $\mathbf{G} = (0, 0, 1)$ and $\mathbf{H}$ equal to the identity matrix, resulting in the hypothesis

$$\begin{aligned} H_0 &: (b_{21}, b_{22}, b_{23}, b_{24}) = 0 \\ & \quad \text{vs.} \\ H_A &: (b_{21}, b_{22}, b_{23}, b_{24}) \neq 0, \end{aligned} \tag{3.19}$$

and by estimating $\mathbf{B}$ using (3.14) as

$$\widehat{\mathbf{B}} = \begin{pmatrix} 111.93 & 93.01 & 119.16 & 146.60 \\ 15.52 & 13.06 & 17.94 & 9.56 \\ -0.60 & -0.56 & -0.95 & -0.311 \end{pmatrix}$$

The relevant test variable can now be calculated using (3.17) to be $T = 17.85636$, which should be compared to the asymptotic distribution (3.18). At a significance level 1%, we have that $T > \chi^2_{0.99}(1 \cdot 4) = 13.28$ and thus, we reject H_0 in (3.19).

3.3 Linear Discriminant Analysis

We are now going to study our first classification procedure, namely linear discriminant analysis (LDA), which allocates observation into predetermined groups. The allocation rule is a function of measurements that maximizes the separation between the groups relative to the within-group variability [13]. We are going to work under the assumption of normal data, that the groups have same covariance $\boldsymbol{\Sigma}$ and that there only exists two predetermined groups, i.e., we have a binary classification procedure. This is illustrated in Figure 3.2.

There are essentially two ways to derive the classification rule of two-group LDA. One based on decision theory and the likelihood ratio rule [7] [13] and

Table 3.1: Data of 45 patients split into 4 groups. One control group without any radiation exposure and three groups at increasing level of radiation. Each patient was observed before the treatment took place and then repeatedly over 10 days.

Patient	**Before radiation**	**Day 1**	**Day 2**	**Day 3**	**Day 4**	**Day 5**	**Day 6**	**Day 7**	**Day 8**	**Day 9**	**Day 10**
						Controls					
1	191	223	242	248	266	274	272	279	286	287	286
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
6	15	22	24	24	38	41	46	62	62	79	74
						25-50r					
7	53	53	102	104	105	125	122	150	93	127	132
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
20	205	234	260	269	274	282	282	290	298	304	308
						75-100r					
21	181	206	199	237	219	237	232	251	247	254	250
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
35	156	186	198	201	205	210	217	217	219	223	229
						125-150r					
36	201	202	229	232	224	237	217	268	244	275	246
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
45	246	257	269	280	289	291	306	301	295	312	311

another based on Fisher's work [3] [13] which we will present here. Consider a sample $\mathbf{x}_0$ and two groups π_1 and π_2 representing the populations $N_p(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$ and $N_p(\boldsymbol{\mu}_2, \boldsymbol{\Sigma})$ respectively. The task is to classify $\mathbf{x}_0$ to either π_1 or π_2. The classification rule is based on the idea that we are trying to find a vector $\mathbf{a}$ so that the the ratio

$$\frac{(\mathbf{a}'\boldsymbol{\mu}_1 - \mathbf{a}'\boldsymbol{\mu}_2)^2}{\mathbf{a}'\boldsymbol{\Sigma}\mathbf{a}} \tag{3.20}$$

is maximized. We are essentially trying to maximize the between-group variability relative to the within-group variability for the linear combinations $\mathbf{a}'\boldsymbol{\mu}_1$ and $\mathbf{a}'\boldsymbol{\mu}_2$. As we will show, by considering the condition of equal misclassification rate for the two groups, all what's left to do is to choose a decision constant K such that we classify $\mathbf{x}_0$ to π_1 if $\mathbf{a}'\mathbf{x}_0 > K$ and to π_2 otherwise.

3.3.1 Deriving a Classification Rule

Consider a sample consisting of two groups, π_1 of size n_1 and π_2 of size n_2, with corresponding population distributions $N_p(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ and parameter estimators $\bar{\mathbf{x}}_i$, $\mathbf{S}_i, i = 1, 2$. Let $\mathbf{x}_0$ be a new observation that belongs to one of the two groups and let

$$\mathbf{S}_p = ((n_1 - 1)\mathbf{S}_1 + (n_2 - 1)\mathbf{S}_2)/(n_1 + n_2 - 2)$$

be the pooled sample covariance for the two groups. Now consider the maximization of (3.20) using the sample data, we get

$$\max_{\hat{\mathbf{a}}\neq 0} \frac{(\hat{\mathbf{a}}'\bar{\mathbf{x}}_1 - \hat{\mathbf{a}}'\bar{\mathbf{x}}_2)^2}{\hat{\mathbf{a}}'\mathbf{S}_p\hat{\mathbf{a}}} = \max_{\hat{\mathbf{a}}\neq 0} \frac{(\hat{\mathbf{a}}'(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2))^2}{\hat{\mathbf{a}}'\mathbf{S}_p\hat{\mathbf{a}}} \overset{(2.2)}{=} (\bar{\mathbf{x}}_\mathbf{1} - \bar{\mathbf{x}}_\mathbf{2})'\mathbf{S}_p^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$$

where $\hat{\mathbf{a}} = \mathbf{S}_p^{-1}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)$.

To decide K we'll work under the condition that the probabilities of misclassification shall be equal. We have

$$\begin{aligned}
&P(\text{classify } \mathbf{x}_0 \text{ to } \pi_1|\mathbf{x}_0 \in \pi_2) = P(\mathbf{a}'\mathbf{x}_0 > K|\mathbf{x}_0 \sim N_p(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}))\\
&= 1 - P(\mathbf{a}'\mathbf{x}_0 \leqslant K|\mathbf{a}'\mathbf{x}_0 \sim N_p(\mathbf{a}'\boldsymbol{\mu}_2, \mathbf{a}'\boldsymbol{\Sigma}\mathbf{a}))\\
&= 1 - \Phi\left(\frac{K - \mathbf{a}'\boldsymbol{\mu}_2}{\sqrt{\mathbf{a}'\boldsymbol{\Sigma}\mathbf{a}}}\right)\\
&= \Phi\left(\frac{-K + \mathbf{a}'\boldsymbol{\mu}_2}{\sqrt{\mathbf{a}'\boldsymbol{\Sigma}\mathbf{a}}}\right) \qquad (3.21)
\end{aligned}$$

and in the same way we can calculate

$$P(\text{classify } \mathbf{x}_0 \text{ to } \pi_2|\mathbf{x}_0 \in \pi_1) = \Phi\left(\frac{K - \mathbf{a}'\boldsymbol{\mu}_1}{\sqrt{\mathbf{a}'\boldsymbol{\Sigma}\mathbf{a}}}\right). \qquad (3.22)$$

By setting (3.21) equal (3.22), K can be calculated as $-K + \mathbf{a}'\boldsymbol{\mu}_2 = K - \mathbf{a}'\boldsymbol{\mu}_1 \Leftrightarrow K = \frac{1}{2}\mathbf{a}'(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)$. Now, by replacing $\mathbf{a}, \boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ with samples estimates $\hat{\mathbf{a}}, \bar{\mathbf{x}}_1$ and $\bar{\mathbf{x}}_2$ we get

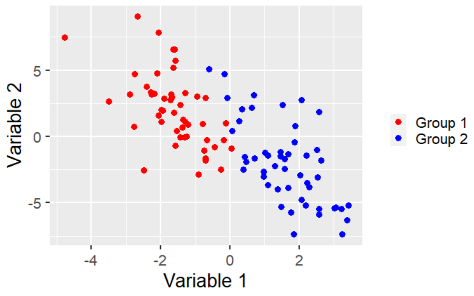
$$\hat{K} = \frac{1}{2}\hat{\mathbf{a}}'(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2) = \frac{1}{2}(\bar{\mathbf{x}}_\mathbf{1} - \bar{\mathbf{x}}_\mathbf{2})'\mathbf{S}_p^{-1}(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2)$$

and thus, we have the classification rule

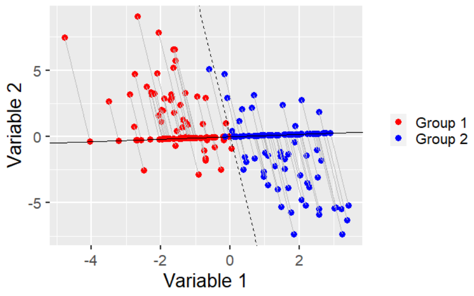
$$\text{Classify observation } \mathbf{x}_0 \text{ to } \begin{cases}\pi_1, & \text{if } (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{S}_p^{-1}\mathbf{x}_0 > \frac{1}{2}(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_2)'\mathbf{S}_p^{-1}(\bar{\mathbf{x}}_1 + \bar{\mathbf{x}}_2),\\ \pi_2, & \text{otherwise.}\end{cases}$$

We can simplify the notation for the classification rule by defining $L(\mathbf{x}_0|\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \mathbf{S}_p) = \hat{\mathbf{a}}'\mathbf{x}_0 - \hat{K}$. This results in

$$\text{Classify observation } \mathbf{x}_0 \text{ to } \begin{cases}\pi_1, & \text{if } L(\mathbf{x}_0|\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \mathbf{S}_p) > 0,\\ \pi_2, & \text{otherwise.}\end{cases} \qquad (3.23)$$



(a) Raw data.



(b) Data projected onto discriminant (solid line) and seperated (dashed line).

Figure 3.2: Illustration of the LDA procedure of a two-dimensional data set that consists of two groups.

3.3.2 Linear Discriminant Analysis for Growth Curves

We are now going to apply the classification rule derived for LDA to the growth curves discussed in (3.2). The growth curve model, is given by

$$\mathbf{Y} = \mathbf{ABC} + \mathbf{E},$$

where $\mathbf{Y} : p \times n$ is the response matrix, $\mathbf{A} : p \times q$ is the within-group design matrix, $\mathbf{B} : q \times k$ is the parameter matrix, $\mathbf{C} : k \times n$ is the between-group design matrix and $\mathbf{E} \sim N_{p,n}(\mathbf{0}, \mathbf{\Sigma}, \mathbf{I}_n)$ is the random error. In the two-group model, i.e., $k = 2$, this reduces to

$$(\mathbf{y}_1, \ldots, \mathbf{y}_{n_1}, \ldots, \mathbf{y}_{n_2}) = \mathbf{A}(\mathbf{b}_1, \mathbf{b}_2) \begin{pmatrix} \mathbf{1}'_{n_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1}'_{n_2} \end{pmatrix} + \mathbf{E}$$

or simply, a sample $\mathbf{x}_0 : p \times 1$ belongs to one of the two populations

$$\pi_1 : \mathbf{x}_1 = \mathbf{A}\mathbf{b}_1 + \mathbf{e},$$
$$\pi_2 : \mathbf{x}_2 = \mathbf{A}\mathbf{b}_2 + \mathbf{e},$$

where $\mathbf{e} \sim N_p(\mathbf{0}, \mathbf{\Sigma})$.

Now, let $\widehat{\mathbf{B}} = \left(\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2\right)$ and $\widehat{\mathbf{\Sigma}}$ be the MLE's (3.14) for the growth curve model and let's replace the parameters in the standard LDA classifier (3.23) with the MLE's. We get

$$L(\mathbf{x}_0|\mathbf{A}\hat{\mathbf{b}}_1, \mathbf{A}\hat{\mathbf{b}}_2, \widehat{\mathbf{\Sigma}}) = (\mathbf{A}\hat{\mathbf{b}}_1 - \mathbf{A}\hat{\mathbf{b}}_2)'\widehat{\mathbf{\Sigma}}^{-1}\mathbf{x}_0 - \frac{1}{2}(\mathbf{A}\hat{\mathbf{b}}_1 - \mathbf{A}\hat{\mathbf{b}}_2)'\widehat{\mathbf{\Sigma}}^{-1}(\mathbf{A}\hat{\mathbf{b}}_1 + \mathbf{A}\hat{\mathbf{b}}_2)$$

which gives us a classification rule for the GMANOVA model

$$\text{Classify observation } \mathbf{x}_0 \text{ to growth curve } \begin{cases} \pi_1, & \text{if } L(\mathbf{x}_0|\mathbf{A}\hat{\mathbf{b}}_1, \mathbf{A}\hat{\mathbf{b}}_2, \widehat{\mathbf{\Sigma}}) > 0, \\ \pi_2, & \text{otherwise.} \end{cases}$$

3.4 Neural Networks

We shall now leave LDA and look at our next type of classification method, namely neural networks. Studying neural networks, often referred to as deep learning, gives us a large collection of prediction methods which can be used both for classification and regression problems.

Neural networks are essentially data transformers that make affine transformations of data and then apply non-linear functions so that the data eventually gets linearly separable. We *train* the network by adjusting the parameters in the network, often with gradient-based learning methods, so that the predictions it assembles ($\hat{y}_i$) matches the true values (y_i) *as close as possible.* What close means in this context is not well defined, but it comes down to *empirical risk minimization* of a risk function defined by a function L, which we refer to as the loss function. When we are working with regression problems we often use the squared error as our loss function, while discrete cross-entropy is commonly used for classification [7].

As we shall see, we are not going to limit ourselves to assumptions about the parameters, distribution of data or that there are only two groups as we did for LDA in Section 3.3, and this will give us a more general prediction framework to work with. However, this results in a less statistical elegant method without closed form solutions and we will rely on finding local minima using iterative optimization methods.

3.4.1 Introduction - Feedforward Neural Networks

A feedforward neural network, which we mostly will refer to as neural network or FNN, takes an labeled input vector $\mathbf{x} \in \mathbb{R}^p$ and defines an output $f(\mathbf{x}; \boldsymbol{\theta}) := \hat{\mathbf{y}}$ where $\boldsymbol{\theta}$ are the parameters. The goal is to adjust $\boldsymbol{\theta}$ such that the neural network can predict the input $\mathbf{x}$ so that the output $\hat{\mathbf{y}}$ matches the true value $\mathbf{y}$ as good as possible.

In a one hidden layer network, the i':th output variable is given by

$$\hat{y}_{i'} = v_{i'0} + \sum_{k=1}^{p'} v_{i'k} z_k = v_{i'0} + \sum_{k=1}^{p'} w_{i'k} \underbrace{f\Big(w_{k0} + \sum_{j=1}^{p} w_{kj} x_j\Big)}_{z_k}, \tag{3.24}$$

where f is the activation function, z_k is the k:th output from the hidden layer, w_{kj} and $v_{i'k}$ are the weights for respectively layer and w_{k0} and $v_{i'0}$ are the biases. This is illustrated in Figure 3.3 for a neural network with $p = 10$ predictors and $p' = 20$ hidden variables and a single output. Note that bias in this scenario refers to the fact that without any input, the prediction will be biased towards the bias term. It does not represent statistical bias.

The output from a one hidden layer neural network can be more compactly written by considering a matrix-vector notation. Let $\mathbf{a} := \mathbf{W}\mathbf{x} + \mathbf{w_0}$ and apply f entry wise to each element of $\mathbf{a}$. Then, (3.24) can be written as

$$\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{w}_0, \tag{3.25}$$

$$\mathbf{z} = f(\mathbf{a}), \tag{3.26}$$

$$\hat{\mathbf{y}} = \mathbf{V}\mathbf{z} + \mathbf{v}_0. \tag{3.27}$$

This procedure can be generalized to arbitrary many layers by repeating (3.25) and (3.26) for additional weights before considering a final output (3.27). This is illustrated in Figure 3.4 for a neural network with three hidden layers.

The activation functions in the hidden layers are needed to ensure that our FNN has the capability to classify non-linear separable data, since a FNN which only relies on affine transformation is only capable of classifying already linear separable data [5]. One can think about this as an arbitrary amount of linear and affine transformations of a vector $\mathbf{x}$, which never will result in a mapping $f(\mathbf{x})$ where f is any non-linear function. Two historically commonly used activation functions are

$$\sigma_s(x) = \frac{1}{1 + e^{-x}} \quad \text{(sigmoid)}$$

and

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad \text{(hyperbolic tangent).}$$

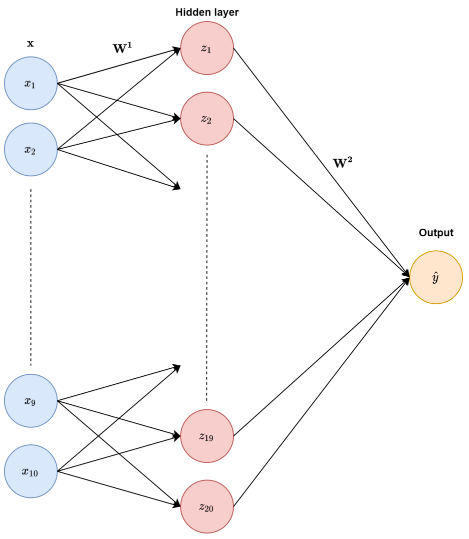


Figure 3.3: Illustration of a one hidden layer feedforward neural network with 10 input variables, 20 nodes in the hidden layer and one output variable.

Although the two functions solves the problem of separability, it is recommended in modern FNN to use the ReLU function

$$\text{ReLU}(x) = \max\{0, x\} \quad \text{(rectified linear unit)}$$

to avoid vanishing gradient effects while updating (training) the network [12] [5], which we will discuss later. Note that in classification tasks, such as image classification, the sigmoid function or its corresponding multivariate version

$$f(x_i) = \frac{e^{x_i}}{\sum_{i=1} e^{x_i}} \quad \text{(softmax)}$$

is still standard to use in the output layer to get a probabilistic interpretation of our outputs.

One of the things that make neural networks powerful is that networks with hidden layers provide a *universal approximation framework*, which is based on the universal approximation theorem [10]. In context of neural networks, this mean that a network with a linear output layer and at least one hidden layer with any "squashing" (sigmoid, tanh etc.) or ReLU activation function can approximate any continuous function on $\mathbb{R}^n$ arbitrary well, from one finite-dimensional space

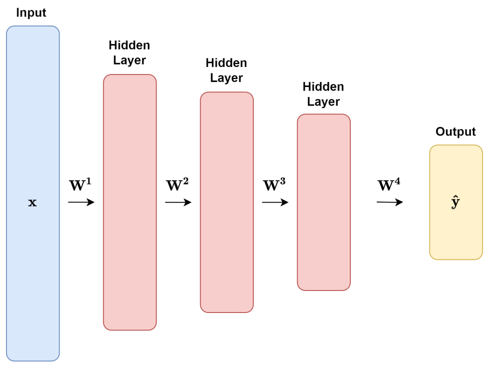


Figure 3.4: A three hidden layer feedforward neural network with a more general presentation.

to another [5] [20]. This result also holds for function mappings between finite discrete spaces [5].

In practice, the universal approximation theorem states that a large enough one hidden layer neural network is able to represent any function we are trying to learn. However, there is no guarantee that our training algorithm will manage to adjust the parameters of the neural network properly, neither does the theorem specify on how large the network has to be nor can we be sure that the training samples contain all information for complete generalization. Nevertheless, the theorem provides us an important result of the capacity of neural networks in general.

3.4.2 Optimization

As we are applying non-linear activation functions to our affine transformations, the convexity of our functions quickly disappears. This, together with the large amount of parameters neural networks often are built upon, makes it infeasible to find minima of the loss function with analytical methods. A classic example of this is *LeNet-5* (1998) [18] with 60000 parameters, which was used for classification of the MNIST data set. Another neural network design which has been influential in the deep learning community, also used for image classification, is *AlexNet* (2012) [17] with 60 million parameters.

The substantial non-linearity in neural networks makes it easy to understand that we have to rely on iterative methods to find local minima when we tune the parameters $\boldsymbol{\theta}$. The parameters should be chosen such that the empirical risk R becomes as small as possible. The empirical risk is computed as the average *loss* or *cost* of predicting $\mathbf{y}_i$ with $\hat{\mathbf{y}}_i$, that is

$$R(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\hat{\mathbf{y}}_i, \mathbf{y}_i), \tag{3.28}$$

where L is the *loss function* for each sample $i = 1, \dots, n$ and we refer to this training process as empirical risk minimization [5].

Gradient Descent

The general approach to minimize R is to use the *gradient-descent* method, which most optimization algorithms for neural networks are based on. The gradient-descent method updates the parameters as

$$\boldsymbol{\theta}^{(new)} = \boldsymbol{\theta}^{(old)} - \gamma \boldsymbol{\nabla}_{\boldsymbol{\theta}} R,$$

where γ is the step size, also known as the learning rate. The gradient-descent method is based on the idea that if γ is sufficiently small, then it holds that $R(\boldsymbol{\theta}^{(new)}) < R(\boldsymbol{\theta}^{(old)})$ as long as $\boldsymbol{\nabla}_{\boldsymbol{\theta}} R \neq \mathbf{0}$.

The process of calculating the gradient in a neural network is referred to as *back propagation* since the derivatives are calculated by applying the chain rule, starting backwards with the predictor variables $y_{i'}$. By considering equation (3.24) with a squared error loss function, the loss for a single output variable $\hat{y}$ in the i:th sample is given by

$$L_i = L(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2 = \Big(y_i - \big(w_0 + \sum_{k=1}^{p'} w_k f(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij})\big)\Big)^2$$

and the corresponding back propagation equations are given by

$$\begin{aligned} \frac{\partial L_i}{\partial w_{kj}} &= -2\big(y_i - \hat{y}_i\big) w_k f'\big(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\big) x_{ij}, \\ \frac{\partial L_i}{\partial w_k} &= -2\big(y_i - \hat{y}_i\big) f\big(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\big), \end{aligned} \tag{3.29}$$

which results in the final updating rule

$$
\begin{aligned}
w_{kj}^{(t+1)} &= w_{kj}^{(t)} - \frac{\gamma}{n}\sum_{i=1}^{n}\frac{\partial L_i}{\partial w_{kj}^{(t)}},\\
w_{k}^{(t+1)} &= w_{k}^{(t)} - \frac{\gamma}{n}\sum_{i=1}^{n}\frac{\partial L_i}{\partial w_{k}^{(t)}},
\end{aligned}
$$

at the t:th iteration. Alternatively, we can consider a general loss function L (cross entropy etc.) and formulate a more compact notation of the back propagation equations. Let $a_{ik} = w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}$ and $z_{ik} = f(a_{ik})$ as in (3.25) and (3.26). Then (3.29) can be written as

$$
\begin{aligned}
\frac{\partial L_i}{\partial w_{kj}} &= \frac{\partial L_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_{ik}}\frac{\partial z_{ik}}{\partial a_{ik}}\frac{\partial a_{ik}}{\partial w_{kj}},\\
\frac{\partial L_i}{\partial w_{k}} &= \frac{\partial L_i}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial w_{k}}.
\end{aligned}
$$

Variations of Gradient Descent

A common variation of the gradient-descent method is the mini-batch *stochastic gradient descent* method (SGD). SGD makes a stochastic approximation of the true gradient while training the neural network. This means that randomly chosen subsets, so called mini-batches, of the whole training set are used to approximate the true gradient, which is also where the name of the method comes from [7]. In practice, SGD results in less accurate but much faster iterations than standard gradient descent.

A common problem with the gradient based methods is that the gradient oscillates through the level curves of the function being minimized and the only way to prevent this is to adjust the learning rate. This is one of the more troublesome problems to tackle when training neural networks since there is no exact rule of how to set the learning rate and it is often guided by previous experience. One way to overcome this is to adapt the learning rate to the current iteration using *adaptive learning methods* [5], such as *RMSprop* and *Adam.*

The RMSProp [8] (Root Mean Square Propagation) algorithm adjusts regular gradient descent by scaling the learning rate γ proportionally to the gradient. Let $\delta \approx 0$ for numerical stability, $\rho \in [0, 1)$ be the decay rate and initialize a vector $\mathbf{r} = \mathbf{0}$. Then, for each iteration,

$$\begin{aligned} \mathbf{r}^{(new)} &= \rho \mathbf{r}^{(old)} + (1-\rho)\nabla_{\boldsymbol{\theta}} R \odot \nabla_{\boldsymbol{\theta}} R, \\ \boldsymbol{\theta}^{(new)} &= \boldsymbol{\theta}^{(old)} - \frac{\gamma}{\delta + \sqrt{\mathbf{r}}} \odot \nabla_{\boldsymbol{\theta}} R, \end{aligned} \tag{3.30}$$

where the root is taken component-wise. Looking at the individual components of $\boldsymbol{\theta}^{(new)}$ in (3.30), we have

$$\theta_i^{(new)} = \theta_i^{(old)} - \gamma \left(\frac{g_i}{\delta + \sqrt{\rho r_i + (1-\rho) g_i^2}} \right),$$

where g_i denotes the i:th component of the gradient $\nabla_{\boldsymbol{\theta}} R$. The RMSProp algorithm scales the updating values so that the components of $\nabla_{\boldsymbol{\theta}} R$ with larger values get scaled more than those with smaller values. In practice, this normalizes each direction of how the empirical risk R is updated, and thus, we have less oscillation. RMSProp together with standard gradient descent is illustrated in Figure 3.5.

The Adam (Adaptive moment estimation) algorithm [15] aims to improve RMSProp by including momentum [22], which is designed to accelerate learning where the curvature of the loss surface is high [5]. Let γ and δ be as before, let $\rho_1, \rho_2 \in [0, 1)$ be the decay rates and initialize $t = 0$ and vectors $\mathbf{r} = \mathbf{s} = \mathbf{0}$. Then, for each iteration, update $t = t + 1$ and

$$\begin{aligned} \mathbf{s}^{(new)} &= \frac{\rho_1 \mathbf{s}^{(old)} + (1-\rho_1)\nabla_{\boldsymbol{\theta}} R}{1-\rho_1^t}, \\ \mathbf{r}^{(new)} &= \frac{\rho_2 \mathbf{r}^{(old)} + (1-\rho_2)\nabla_{\boldsymbol{\theta}} R \odot \nabla_{\boldsymbol{\theta}} R}{1-\rho_2^t} \end{aligned} \tag{3.31}$$

and then update $\boldsymbol{\theta}$ as in (3.30). Furthermore, $\mathbf{r}$ and $\mathbf{s}$ is being normalized to prevent divergence in early iterations [15].

3.4.3 Some Issues with Neural Networks and how to Tackle Them

A common problem with optimization of neural networks is overfitting, which means that the parameters that results in a low empirical risk on the training data does not generalize well to unseen test data. This is simply illustrated with polynomial regression, see Figure 3.6. Deciding the overall structure of a neural network, which we refer to as the *architecture* of the network, is not a

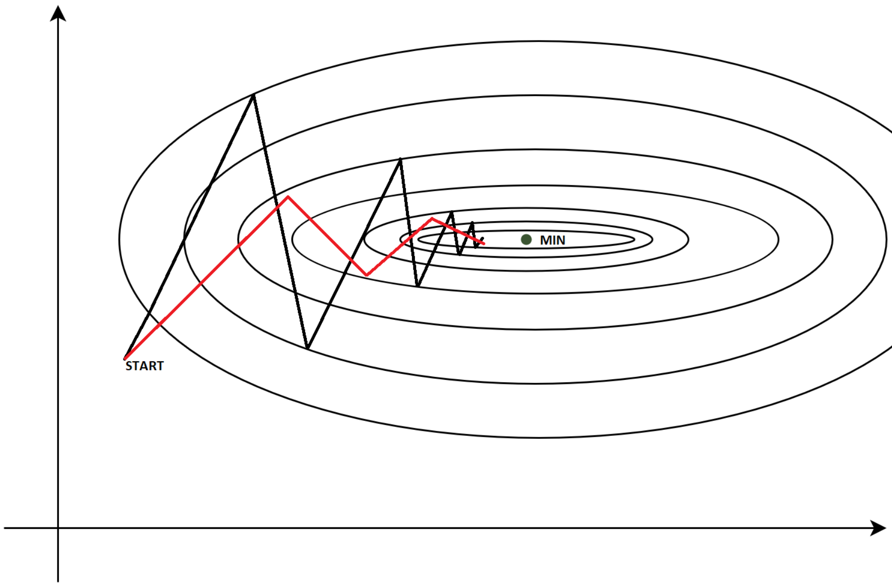


Figure 3.5: Illustration of how the standard gradient descent algorithm (black lines) moves on the level curves of a three-dimensional function towards the minimum, compared to the RMSProp algorithm (red lines). RMSProp tries to normalize the gradient to avoid the "zig-zag" pattern which often occurs with the gradient descent algorithm.

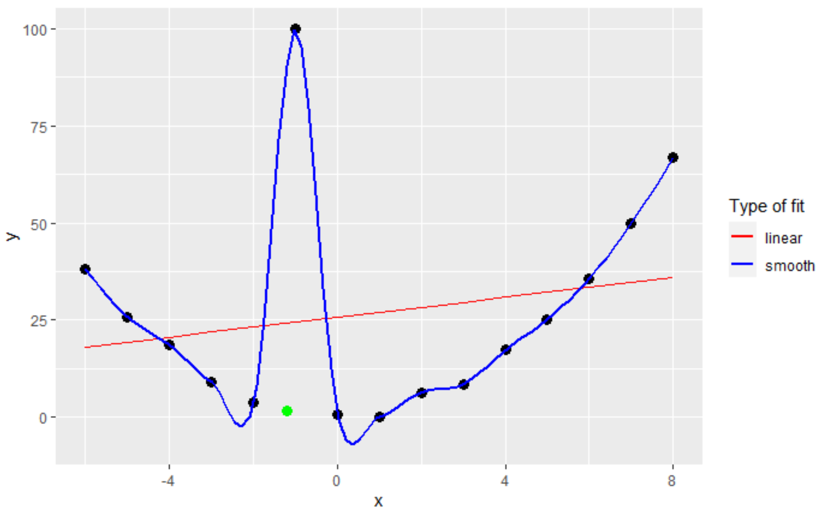


Figure 3.6: Fitting two different models to the same training data (black dots). A linear regression model (red curve) of degree one and a local polynomial model (blue curve) which tries to smoothly fit all points. When we try to predict the value of a new observation (green dot) we can observe that both of the models will be way off. The linear model will off due to underfitting and the smooth fitted model will be off due to overfitting.

straightforward task. In general, the choice of architecture in a neural network for a specific task is guided by background knowledge and experimentation [7] [5].

Regularization

One way to tackle overfitting is with *regularization*. In general, regularization is any modification to a statistical algorithm designed to reduce the test error, possibly at the cost of increased training error [5]. A standard regularization technique is to add a norm penalty Ω to the empirical risk (3.28) so that the new risk function is given by

$$\tilde{R}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L(\hat{\mathbf{y}}_i, \mathbf{y}_i) + \lambda\Omega(\boldsymbol{\theta}), \tag{3.32}$$

where $\lambda \in [0, \infty)$ is the regularization parameter. The usage of (3.32) enforces the parameters to be small while minimizing $\tilde{R}$, which often reduces overfitting

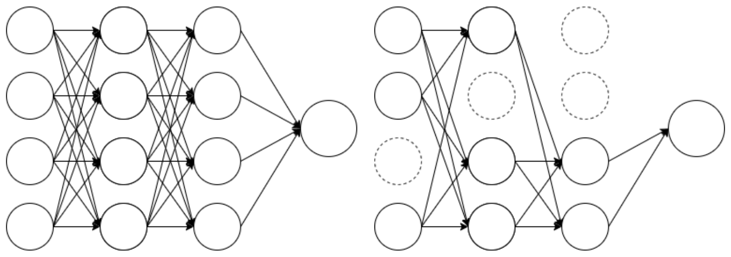

(a) Before drouput. (b) After dropout.

Figure 3.7: Illustration of dropout regularization. which randomly zeros out (drops) nodes together with their weights in the neural network

to some extent. In general, the effect of regularization is that we make a trade of increased statistical bias and decreased variance, hopefully to make the test error lower.

Another type of regularization is *dropout* [25], which randomly zeros (drops) nodes in a neural network. Dropout is performed by multiplying each node $z_i^{(l)}$ in layer l with $b_i^{(l)} \sim \text{Bernoulli}(p)$, where $p \in (0,1)$. The obtained weights $\mathbf{W}_{\text{train}}^{(l)}$ is multiplied with p so that the final weights are given by $\mathbf{W}_{\text{test}}^{(l)} = p\mathbf{W}_{\text{train}}^{(l)}$, which is done to ensure that the expected output during testing is the same as the one during training. Dropout regularization is illustrated in Figure 3.7.

One can think about dropout as training each sample on a different neural network, and then we average the final result. Note that if we have in total N nodes in the neural network, then there is 2^N possible network architectures to train with, so for a network with 100 nodes in total, we already have $2^{100} > 10^{30}$ different networks to use. Alternatively, we can see dropout as a type of data augmentation where we create new data by adding noise to the current samples [5].

Parameter Initialization and Scaling Inputs

The parameters of a neural networks has to be initialized before training can take place. Starting with all weights equal to zero results in $\boldsymbol{\nabla}_{\boldsymbol{\theta}} R = \mathbf{0}$ and the optimization algorithm will be stuck. On the contrary, starting with too large values on the parameters often results in poor solutions [7]. In practice, we initialize biases to zero and the weights with either a uniform distribution

$U[-\alpha, \alpha]$ or a normal distribution $N(0, \sigma^2)$, where α and σ^2 is close to 0 [5]. A popular method is to use *normalized initialization* [4] which initializes the weights as

$$w_{ij}^{(l)} \sim U\left(-\sqrt{\frac{6}{n^{(l)} + n^{(l+1)}}}, \sqrt{\frac{6}{n^{(l)} + n^{(l+1)}}}\right),$$

where $n^{(l)}$ and $n^{(l+1)}$ is the number of nodes connected by the l:th layer, respectively. For instance, the deep learning library Keras uses normalized initialization as default weight initialization for most neural networks.

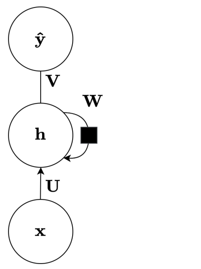
It is also important to scale the input variables to avoid learning issues [7]. To see this, consider a neural network with n input variables such that $x_1 \in [-L, L]$ and $x_i \in [-l, l], i = 2, \ldots, n$, where $L \gg l$. A single output from the layer would equal to $a := w_0 + w_1 x_1 + \ldots + w_n x_n$ for corresponding weights $w_i, i = 0, \ldots, n$. Due to the differing size between L and l, it would hold with a high probability that $a \approx w_1 x_1$ before training and thus, we would have a vanishing effect of the variables $x_2, \ldots, x_n$, which in practice would result in slow learning and eventual numerical issues. To avoid this, it is a common procedure to standardize the input variables so they have mean zero and standard deviation one [7] [19].

3.4.4 Recurrent Neural Networks and LSTM

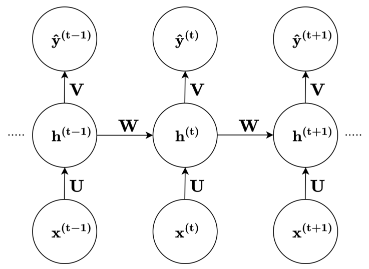
In the previous section has it always been assumed that the neural networks has been of type feedforward which means that the networks contain no cycles. In practice, this means that we always send our data $\mathbf{x}$ forward in the network, or alternatively, deeper into a long chain of functions. We are now going to drop the assumption of no cycles and focus on *recurrent neural networks*.

Recurrent neural networks is a large class of neural networks that are designed to process sequential data. What differs sequential data from non-sequential data is that we cannot assume independence between individual data observations. Some examples of sequential data are weather data, text documents, speech recordings and financial data.

A feedforward neural network can only map a single input to output, while a recurrent neural network can map a history of inputs to each output over time. Each hidden layer is updated in a similar way for the recurrent networks as in the forward case but with the addition of information passed from the previous time step, which is illustrated in Figure 3.8. The i':th output variable from a



(a) Folded recurrent neural network.



(b) Unfolded recurrent neural network.

Figure 3.8: Illustration of standard recurrent neural network design.

recurrent neural network at time t can be formulated as

$$\hat{y}_{i'}^{(t)} = v_{i'0} + \sum_{i=1}^{p'} v_{i'i} \underbrace{f\left(u_{j0} + \sum_{j=1}^{p} u_{ij} x_j^{(t)} + \sum_{k=1}^{p'} w_{ik} h_k^{(t-1)}\right)}_{h_i^{(t)}}, \tag{3.33}$$

where f is the activation function, $x_j^{(t)}$ is the j:th predictor variable and $h_k^{(t)}$ is the the k:th output from the hidden layer, each at time step t. Furthermore, $v_{i'i}$, u_{ij} and w_{ik} are the corresponding weights between the layers and $v_{i'0}$ and u_{j0} are the bias weights. Alternatively, we can consider a matrix-vector notation as we did for (3.24) and write (3.33) as

$$\mathbf{a}^{(t)} = \mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{u}_0, \tag{3.34}$$

$$\mathbf{h}^{(t)} = f(\mathbf{a}^{(t)}), \tag{3.35}$$

$$\hat{\mathbf{y}}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{v}_0. \tag{3.36}$$

If we are working with a classification task and are in need of a probabilistic interpretation of the output, we can simply apply a sigmoid or softmax function to (3.33) and (3.36).

One problem with recurrent neural networks is that the long term dependencies that we are trying to learn easily vanishes. To see this, we can consider a single $y^{(t)}$ output variable an look at the corresponding backpropagation equation for the w_{ik} variables. Computing $\frac{\partial L}{\partial w_{ik}}$ for the m:th sample $\hat{y}_m^{(t)}$ at time t gives us

$$\frac{\partial L_m^{(t)}}{\partial w_{ik}} = \frac{\partial L_m^{(t)}}{\partial \hat{y}_m^{(t)}} \frac{\partial \hat{y}_m^{(t)}}{\partial h_{mi}^{(t)}} \frac{\partial h_{mi}^{(t)}}{\partial w_{ik}}. \tag{3.37}$$

The problem arises when we evaluate $\frac{\partial h_{mi}^{(t)}}{\partial w_{ik}}$ since $h_{mi}^{(t)}$ is a function depending on w_{ik}, but also of $h_{mi}^{(t-1)}$ which itself is a function depending on the same w_{ik}. Evaluating $\frac{\partial h_{mi}^{(t)}}{\partial w_{ik}}$ provides

$$\begin{aligned}
\frac{\partial h_{mi}^{(t)}}{\partial w_{ik}} &= f'\left(a_{mi}^{(t)}\right)\left(h^{(t-1)} + w_{ik}\frac{\partial h_{mi}^{(t-1)}}{\partial w_{ik}}\right) \\
&= \frac{\partial h_{mi}^{(t)}}{\partial a_{mi}^{(t)}}\left(\frac{\partial^+ a_{mi}^{(t)}}{\partial w_{ik}} + \frac{\partial a_{mi}^{(t)}}{\partial h_{mi}^{(t-1)}}\frac{\partial h_{mi}^{(t-1)}}{\partial w_{ik}}\right) \\
&= \frac{\partial^+ h_{mi}^{(t)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t)}}{\partial h_{mi}^{(t-1)}}\frac{\partial h_{mi}^{(t-1)}}{\partial w_{ik}},
\end{aligned}$$

where $\frac{\partial^+}{\partial w_{ik}}$ denotes that we are taking the explicit derivative with respect to w_{ik} once possible and $a_{mi}^{(t)} = u_{j0} + \sum_{j=1}^{p} u_{ij}x_{mj}^{(t)} + \sum_{k=1}^{p'} w_{ik}h_{mk}^{(t-1)}$. As we can see, $\frac{\partial h_{mi}^{(t)}}{\partial w_{ik}}$ depends on itself but from an earlier phase. Continuing with the evaluation of $\frac{\partial h_{mi}^{(t)}}{\partial w_{ik}}$ yields

$$\begin{aligned}
\frac{\partial h_{mi}^{(t)}}{\partial w_{ik}} &= \frac{\partial^+ h_{mi}^{(t)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t)}}{\partial h_{mi}^{(t-1)}}\frac{\partial h_{mi}^{(t-1)}}{\partial w_{ik}} \\
&= \frac{\partial^+ h_{mi}^{(t)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t)}}{\partial h_{mi}^{(t-1)}}\left(\frac{\partial^+ h_{mi}^{(t-1)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t-1)}}{\partial h_{mi}^{(t-2)}}\frac{\partial h_{mi}^{(t-2)}}{\partial w_{ik}}\right) \\
&= \frac{\partial^+ h_{mi}^{(t)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t)}}{\partial h_{mi}^{(t-1)}}\left(\frac{\partial^+ h_{mi}^{(t-1)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t-1)}}{\partial h_{mi}^{(t-2)}}\left(\frac{\partial^+ h_{mi}^{(t-2)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t-2)}}{\partial h_{mi}^{(t-3)}}(\dots)\right)\right) \\
&= \frac{\partial^+ h_{mi}^{(t)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t)}}{\partial h_{mi}^{(t-1)}}\frac{\partial^+ h_{mi}^{(t-1)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t)}}{\partial h_{mi}^{(t-1)}}\frac{\partial h_{mi}^{(t-1)}}{\partial h_{mi}^{(t-2)}}\frac{\partial^+ h_{mi}^{(t-2)}}{\partial w_{ik}} + \frac{\partial h_{mi}^{(t)}}{\partial h_{mi}^{(t-1)}}\frac{\partial h_{mi}^{(t-1)}}{\partial h_{mi}^{(t-2)}}\frac{\partial h_{mi}^{(t-2)}}{\partial h_{mi}^{(t-3)}}(\dots) \\
&= \sum_{\alpha=1}^{t}\left(\prod_{\beta=\alpha+1}^{t}\frac{\partial h_{mi}^{(\beta)}}{\partial h_{mi}^{(\beta-1)}}\right)\frac{\partial^+ h_{mi}^{(\alpha)}}{\partial w_{ik}}.
\end{aligned}$$

Now, by inserting this into (3.37), we have the final derivative

$$\frac{\partial L_m^{(t)}}{\partial w_{ik}} = \frac{\partial L_m^{(t)}}{\partial \hat{y}_m^{(t)}} \frac{\partial \hat{y}_m^{(t)}}{\partial h_{mi}^{(t)}} \sum_{\alpha=1}^{t} \left(\prod_{\beta=\alpha+1}^{t} \frac{\partial h_{mi}^{(\beta)}}{\partial h_{mi}^{(\beta-1)}} \right) \frac{\partial^+ h_{mi}^{(\alpha)}}{\partial w_{ik}}. \tag{3.38}$$

As we can see, we are multiplying a long chain of derivatives as t gets large which in general will lead to either vanishing or sometimes even exploding gradients. One way to overcome this is to use *long short-term memory* recurrent neural networks.

Long short-term memory (LSTM)[6] networks is a type of *gated* recurrent neural network which are based on the idea of creating paths through time that have gradients with elements that neither vanish or explode [5]. In a LSTM network, we replace the standard hidden layers from a recurrent neural network with LSTM layers that contain internal time dependent self loops in addition to the regular recurrence. LSTM networks has the same inputs and outputs as the hidden units in regular recurrent neural networks, but with the addition of an internal *cell state*, an *input gate*, a *forget gate* and an *output gate* that control the information flow in the layer and therefore even in the network. The architecture of a node inside a LSTM layer is illustrated in Figure 3.9.

The gates in a LSTM node are all equipped with a sigmoid activation function (σ_s) so that they let information flow if the gates are open ($\sigma_s \approx 1$) or they hinder the flow if the gates are closed ($\sigma_s \approx 0$). At time step t and for the k:th LSTM node, these gates are defined as

$$i_k^{(t)} = \sigma_s \left(b_k^i + \sum_j U_{kj}^i x_j^{(t)} + \sum_j W_{kj}^i h_j^{(t-1)} \right) \quad \text{(input gate)},$$

$$f_k^{(t)} = \sigma_s \left(b_k^f + \sum_j U_{kj}^f x_j^{(t)} + \sum_j W_{kj}^f h_j^{(t-1)} \right) \quad \text{(forget gate)},$$

$$o_k^{(t)} = \sigma_s \left(b_k^o + \sum_j U_{kj}^o x_j^{(t)} + \sum_j W_{kj}^o h_j^{(t-1)} \right) \quad \text{(output gate)},$$

where $b_{kj}^\bullet, U_{kj}^\bullet$ and $W_{kj}^\bullet$ are the biases, input weights and recurrence weights for each gate. The inner cell is defined as

$$c_k^{(t)} = f_k^{(t)} c_k^{(t-1)} + i_k^{(t)} \sigma_i \left(b_k + \sum_j U_{kj} x_j^{(t)} + \sum_j W_{kj} h_j^{(t-1)} \right) \quad \text{(cell state)},$$

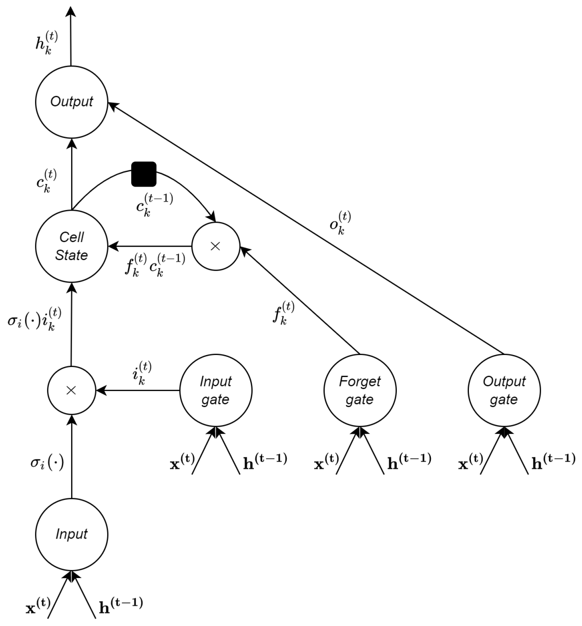


Figure 3.9: Illustration of inside the k:th LSTM node inside a LSTM layer at time t. Arrows and gates indicates how information is allowed to flow in the node. The black square indicates a self loop with a time delay of one time step.

where b_{kj}, U_{kj} and W_{kj} are the biases, input weights and recurrence weights for the input data and σ_i can be any squashing activation function [5]. The cell state at time t is defined as a summation of the previous node state multiplied with the forget gate and the transformed input multiplied with the input gate. Previous information stays in the node only if the forget gate allows it, and the cell gets updated with new information only if the input gate allows it. The k:th output $h_k^{(t)}$ from the LSTM node at time t can now be calculated as

$$h_k^{(t)} = o_k^{(t)} \sigma_o(c_k^{(t)}) \quad \text{(output)},$$

where σ_o can in practice be any transforming function, even the identity [6], but tanh is most commonly used [5]. The output works in a similar way as the previous states. Information is read from the node only if the output gate allows it.

Chapter 4

Results

We shall now present results from usage of our classification methods. The task has been to perform binary classification for different data sets using the methods discussed. We start of by using simulated data and then we proceed to look at two examples of real world data. The simulated data and the first set of real world data follows a classic repeated measurement design, while the weather samples from the second set of real world data are snippets from a long sequence of data. This means that the first and second example is designed to fit the usage of growth curves, while the third example deviates from this.

All train and test data has been normalized using the train data. This means that the train data has mean zero and standard deviation one and the same holds approximately for the test data. All trainable parameters in the neural networks has been adjusted with either the RMSprop optimizer or the Adam optimizer, both found in the Keras package for R. Hyperparameters has been adjusted by hand to find satisfiable results.

The results will be listed with abbreviations for all the methods. LDA4GCM$_{q-1}$ stands for the Linear Discriminant Analysis classifier for Growth Curves estimated using a polynomial of degree $q-1$. FNNk, RNNk and LSTMk stands for a one layer feedforward, recurrent or LSTM neural network with k hidden nodes, while an underscore followed by a number l means that we have another hidden layer with l nodes. All of the results for feedforward and recurrent neural networks has been acquired using ReLU activation functions and all of the neural networks used a logistic sigmoid function for output. Each neural network was trained 3-5 times with new weight initialization and the result with highest accuracy was saved.

4.1 Simulated Data

The first task is to perform binary classification of simulated data. Each sample has been generated independently as

$$\mathbf{y}_i \sim N_p(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}), \tag{4.1}$$

where $\boldsymbol{\mu}_i$ $i = 1, 2$ are observations of the corresponding true deterministic class $f_i(x)$ at $p = 12$ points evenly spaced and $\boldsymbol{\Sigma}$ is a covariance matrix given by

$$\boldsymbol{\Sigma} = \sigma^2 \begin{pmatrix} 1 & 0.5 & \dots & 0.5^{p-1} \\ 0.5 & 1 & \dots & 0.5^{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ 0.5^{p-1} & 0.5^{p-2} & \dots & 1 \end{pmatrix},$$

where σ^2 is some known constant. Each model was fit using a total of $n_{train} = 24$ samples evenly split between the two classes, and the evaluation was done using $n_{test} = 500$ samples evenly split. The size of n_{train} was chosen small to mimic a problematic real world situation where access to data is limited, while a large n_{test} was chosen to get a true evaluation of the fitted models. All of the methods has been tested on three data models (Case 1-3) where new data has been generated five times. This was done to ensure a fair comparison between the methods used.

In the first case we look at polynomial data where the true models are different but there is a high variance so that the observations by themselves are hard to classify correctly. In the second case we keep the variance relatively high but the true polynomial models are more similar. In the third and final case we deviate from using a polynomial model but we lower the variance.

4.1.1 Case 1

The true models are given by

$$\begin{aligned} f_1(x) &= 0.075x^3 - 1.44x^2 + 7.8x - 5.43, \\ f_2(x) &= 0.1x^2 - 0.2x + 1.1 \end{aligned}$$

over the interval $[1, 12]$ and they are displayed in Figure 4.1 together with a sample from each model, generated as (4.1) with $\sigma^2 = 15$. The growth curves was estimated using a polynomial of degree $q - 1 = 3$ and results from prediction are given in Table 4.1 and Figure 4.2.

Table 4.1: Results for simulated data - case 1. Each row represent the used algorithm together with the number of parameters in the model and the accuracy for each of the five simulations. We can observe that the recurrent neural network scores best in all of the five cases.

Method	**Number of trainable parameters**	**1**	**2**	**3**	**4**	**5**
LDA4GCM$_3$	b: 8 Σ: 78	0,82	0,75	0,786	0,804	0,78
FNN12	169	0,826	0,808	0,816	0,75	0,766
FNN12_6	241	0,792	0,852	0,754	0,824	0,714
RNN12	181	0,86	0,864	0,854	0,866	0,82
LSTM1	14	0,5	0,614	0,746	0,5	0,676
LSTM3	64	0,724	0,682	0,718	0,63	0,704
LSTM12	685	0,806	0,798	0,79	0,854	0,738

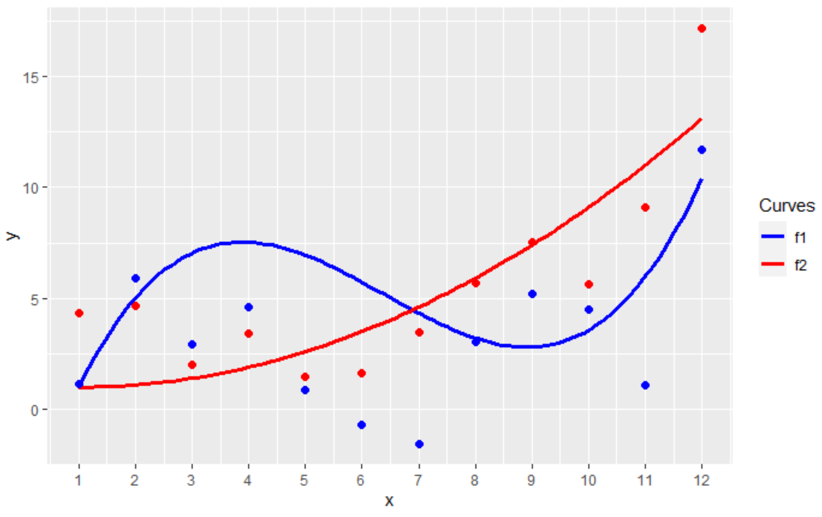


Figure 4.1: Simulated data - Case 1. The plot represent the true models of case 1 together with a simulated sample for each group.

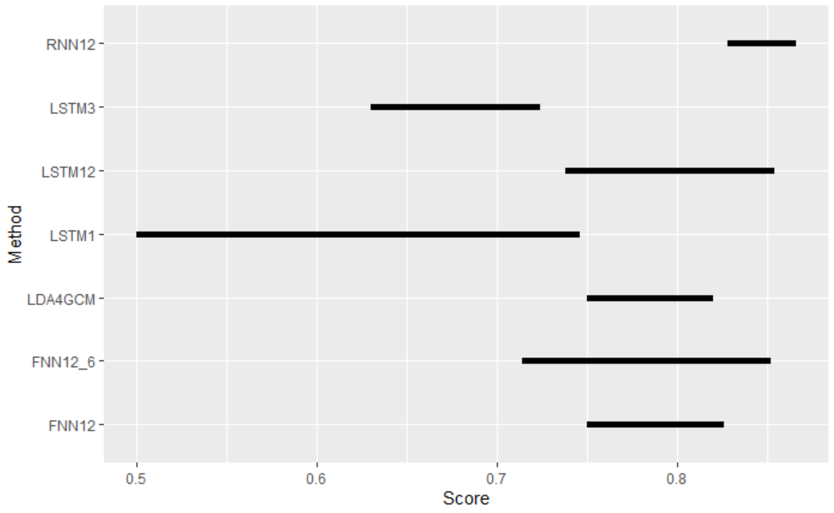


Figure 4.2: Range of prediction accuracy for case 1. Each bar represent the range of accuracy for each of the tested algorithm in the five simulations. For instance, we can observe that the RNN12 classifier seem to perform really stable.

4.1.2 Case 2

The true models are given by

$$f_1(x) = 2(1.3x - 1)^4 - 0.08x^6 + 4.04,$$
$$f_2(x) = 2.4(1.3x - 1)^4 - 0.14x^6 + 4.06$$

over the interval $[0.6, 1.3]$ with $\sigma^2 = 0.002$ and they are displayed in Figure 4.3. The growth curves was estimated using a polynomial of degree $q - 1 = 6$ and results from prediction are given in Table 4.2 and Figure 4.4.

4.1.3 Case 3

The true models are given by

$$f_1(x) = 0.1x^4 - 0.4x^3 + 0.7x^2 - 0.5x + 2.5\cos(10x),$$
$$f_2(x) = 0.1x^4 + 0.1x^3 + 0.7x^2 - 0.5x + 2.5\cos(9x)$$

over the interval $[0, 2.7]$ with $\sigma^2 = 1.$ and they are displayed in Figure 4.5. The growth curves was estimated using a polynomial of degree $q-1 = 8$ and results from prediction are given in Table 4.3 and Figure 4.6.

Table 4.2: Results for simulated data - case 2. Each row represent the used algorithm together with the number of parameters in the model and the accuracy for each of the five simulations. We can observe that the most accurate classifier differs between the five simulations.

Run	**Number of trainable parameters**	**1**	**2**	**3**	**4**	**5**
LDA4GCM$_6$	b: 14 Σ: 78	0,656	0,772	0,644	0,774	0,786
FNN12	169	0,66	0,628	0,766	0,534	0,806
FNN12_6	241	0,566	0,63	0,55	0,622	0,696
RNN12	181	0,808	0,764	0,782	0,75	0,75
LSTM1	14	0,76	0,782	0,592	0,504	0,714
LSTM3	64	0,846	0,758	0,686	0,75	0,696
LSTM12	685	0,758	0,788	0,758	0,752	0,784

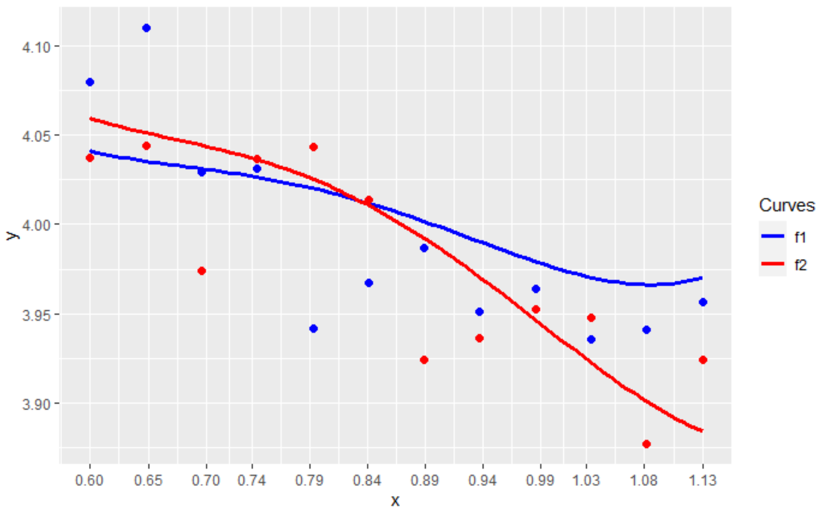


Figure 4.3: Simulated data - Case 2. The plot represent the true models of case 2 together with a simulated sample for each group.

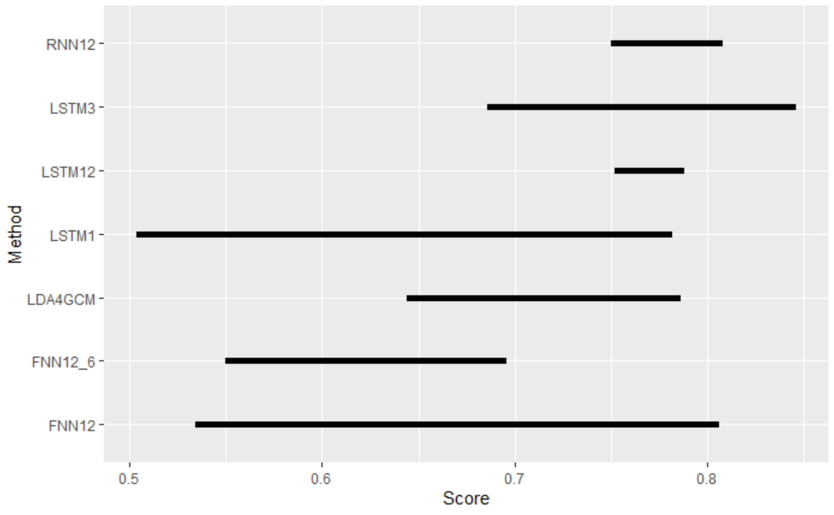


Figure 4.4: Range of prediction accuracy for case 2. Each bar represent the range of accuracy for each of the tested algorithm in the five simulations. We observe that the RNN12 and the FNN12 classifiers performs with approximately same accuracy when they perform as best, but that the accuracy of FNN12 seem to be much more sensitive to the simulated data.

Table 4.3: Results for simulated data - case 3. Each row represent the used algorithm together with the number of parameters in the model and the accuracy for each of the five simulations. The LSTM classifiers in the bottom three rows performed poorly compared to the other classifiers which got 1 or almost 1 accuracy in each simulation.

Run	**Number of trainable parameters**	**1**	**2**	**3**	**4**	**5**
LDA4GCM$_8$	b: 18 Σ: 78	1	0,996	1	1	0,991
FNN12	169	1	0,998	0,994	0,998	0,998
FNN12_6	241	1	0,992	0,996	0,994	0,992
RNN12	181	1	1	1	1	1
LSTM1	14	0,584	0,618	0,536	0,624	0,71
LSTM3	64	0,674	0,672	0,59	0,572	0,574
LSTM12	685	0,704	0,658	0,608	0,678	0,624

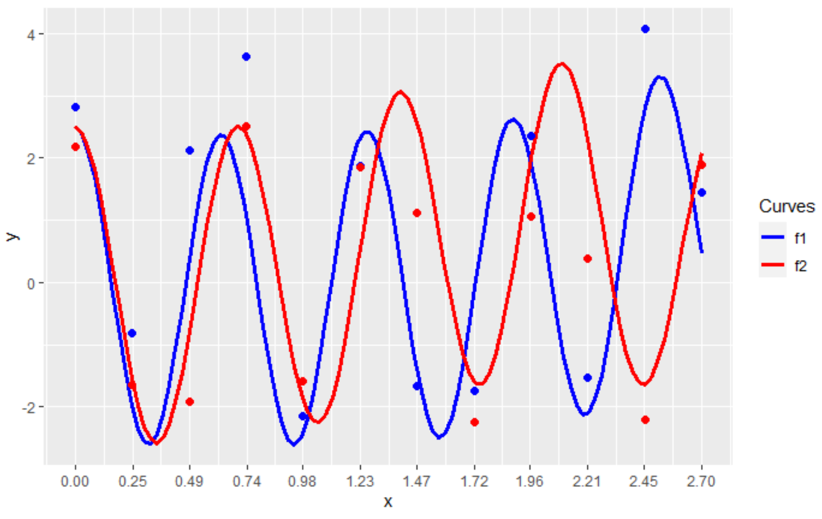


Figure 4.5: Simulated data - Case 3. The plot represent the true models of case 3 together with a simulated sample for each group.

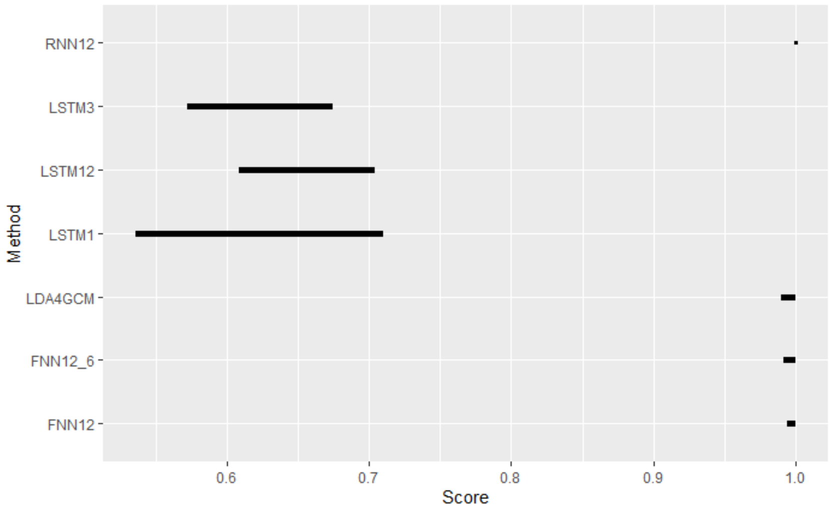
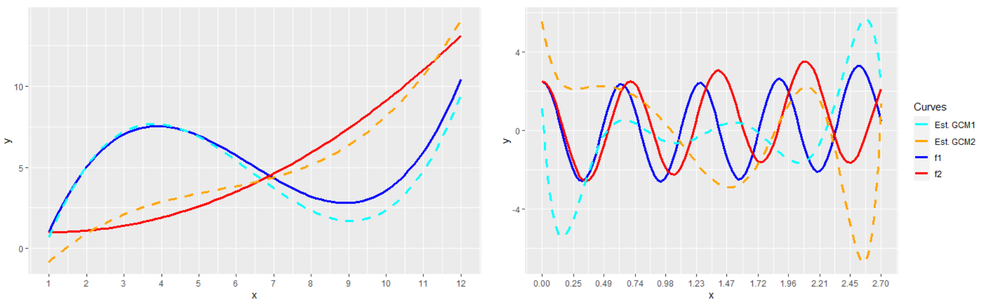


Figure 4.6: Range of prediction accuracy for case 3. Each bar represent the range of accuracy for each of the tested algorithm in the five simulations. All classifiers but the LSTM models seem to separate the data without complications.



(a) Case 1. (b) Case 3.

Figure 4.7: Growth curves fitted to case 1 and case 3. Although the growth curve classifier almost got 100% accuracy each run for case 3, the fitted curves hardly resembles the true curves.

4.1.4 Discussion - Simulated Data

The results for the simulated data is relatively noisy in the sense that each simulation of data yields different accuracy for most of the classification methods, this can be observed in Figure 4.2, Figure 4.4 and to some extent in Figure 4.6. This noise can be explained by the relatively small sample size that was used for fitting the models, but also by the high variance used for simulating data in the first two cases and the low variance in case 3 where some of the methods performed almost equally for different simulations. The LDA4GCM classifier in case 3 performs somewhat odd since the prediction accuracy is almost perfect even though we deviate from using a polynomial base model. Note that we won't be able to interpret the corresponding estimated growth curves for the non-polynomial model, which is illustrated in Figure 4.7, so validation of using LDA4GCM in case 3 is questionable.

4.2 Radiation Data

Next we are going to look at the data from Danford et al. [1] which we discussed in Example 3.1 and the task of interest is to classify which group the patients belongs to. We split the Dandford data ($n = 45$) into four groups, the control group ($n_c = 6$) the 25-50r group ($n_1 = 14$), the 75-100r group ($n_2 = 15$) and the 125-150r group ($n_3 = 10$). We then fit our classifiers with the control group data and the data from one of the groups exposed for radiation, using all but

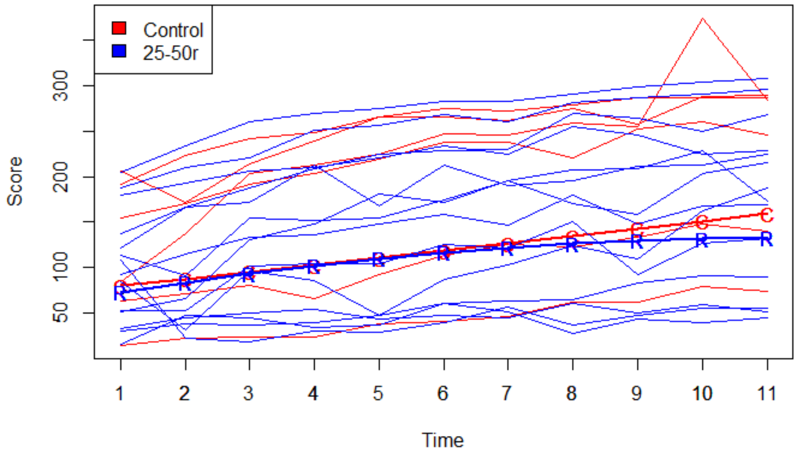


Figure 4.8: Data from the control group and the 25-50r group, together with estimated growth curves in thick lines marked with "C" and "R" respectively.

one sample which will be used for testing, also known as *leave-one-out cross-validation (LOOCV)*. Full tables of classifications will be presented together with accuracy values.

4.2.1 Case 4 - Control Group vs 25-50r

Classifiers are fit using the data from the control group and the 25-50r group. Sample size is $n_{tot} = 20$, data distribution is $(0.3, 0.7)$ for control and 25-50r respectively. Data is shown in Figure 4.8 together with fitted growth curves and results are presented in Table 4.4.

4.2.2 Case 5 - Control Group vs 75-100r

Classifiers are fit using the data from the control group and the 75-100r group. Sample size is $n_{tot} = 21$, data distribution is $(0, 286, 0, 714)$ for control and 75-100r respectively. Data is shown in Figure 4.9 together with fitted growth curves and results are presented in Table 4.5.

Table 4.4: Results from classification of case 4 - control group vs 25-50r. Each row represents one of the twenty runs, the true label of the test class and the classification for each of the used algorithms.

Run	**Correct Class**	**LDA4GCM**	**NN11**	**NN22_11**	**RNN11**	**LSTM11**
1	0	1	1	1	1	1
2	0	0	1	1	1	1
3	0	1	1	1	0	1
4	0	0	1	1	1	1
5	0	0	1	1	0	1
6	0	1	1	1	1	1
7	1	1	1	1	1	1
8	1	1	1	1	1	1
9	1	1	1	1	1	1
10	1	0	1	1	1	1
11	1	1	1	1	0	1
12	1	1	1	1	1	1
13	1	1	1	1	1	1
14	1	0	1	1	1	1
15	1	0	1	1	1	1
16	1	1	1	1	1	1
17	1	0	0	0	0	0
18	1	1	1	1	0	1
19	1	0	1	1	1	1
20	1	0	0	1	0	0
Accuracy		0.55	0.6	0.65	0.6	0.6

Table 4.5: Results from classification of case 5 - control group vs 75-100r. Each row represents one of the twenty-one runs, the true label of the test class and the classification for each of the used algorithms.

Run	**Correct Class**	**LDA4GCM**	**NN11**	**NN22_11**	**RNN11**	**LSTM11**
1	0	1	1	1	1	1
2	0	0	1	1	1	1
3	0	0	1	1	1	1
4	0	0	1	1	1	1
5	0	0	1	1	1	1
6	0	1	1	1	1	1
7	1	0	1	1	1	1
8	1	1	1	1	1	1
9	1	0	1	1	1	1
10	1	0	1	0	1	1
11	1	0	1	1	1	1
12	1	1	1	1	1	1
13	1	1	1	1	1	1
14	1	1	1	1	1	1
15	1	1	1	1	1	1
16	1	1	1	1	1	1
17	1	1	0	0	0	1
18	1	1	1	1	1	1
19	1	0	1	1	1	1
20	1	1	1	1	1	1
21	1	1	1	1	1	1
Accuracy		0.667	0.667	0.619	0.667	0.714

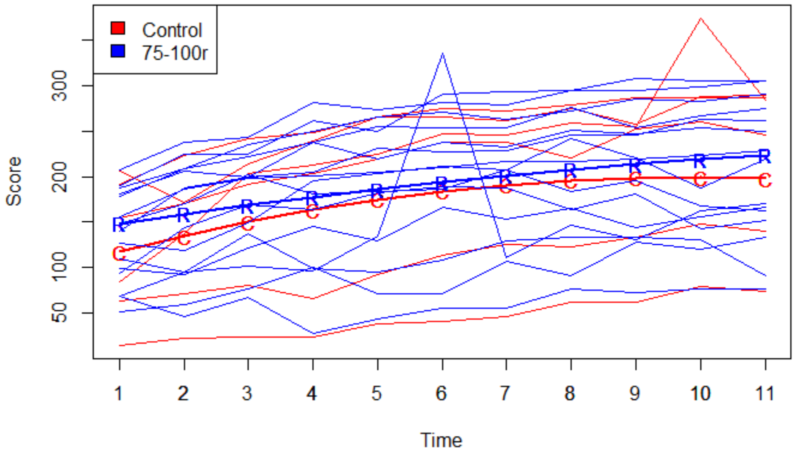


Figure 4.9: Data from the control group and the 75-100r group, together with estimated growth curves in thick lines marked with "C" and "R" respectively.

4.2.3 Case 6 - Control Group vs 125-150r

Classifiers are fit using the data from the control group and 125-150r group. Sample size is $n_{tot} = 16$, data distribution is $(0.375, 0.625)$ for control and 125-150r respectively. Data is shown in Figure 4.10 together with fitted growth curves and results is presented in Table 4.6

4.2.4 Discussion - Radiation Data

Looking at the result in Table 4.4, Table 4.5 and Table 4.6 we can observe that most of the classifiers performs really poorly. The best results was from the LSTM11 classifier with an accuracy of 0.714 for the 75-100r data while the worst result was as low as 0.25 using an RNN11 classifier for the 125-150r data. It should be noted that the LSTM11 classifier with 0.714 accuracy simply classified all test data as same class, so it is questionable how good the classifier actually is for the data used.

Looking at the plots Figure 4.8, Figure 4.9 and Figure 4.10, one can see that the fitted growth curves are extremely close to each other and the data itself is very spread and hard to distinguish between the separate groups. There are no

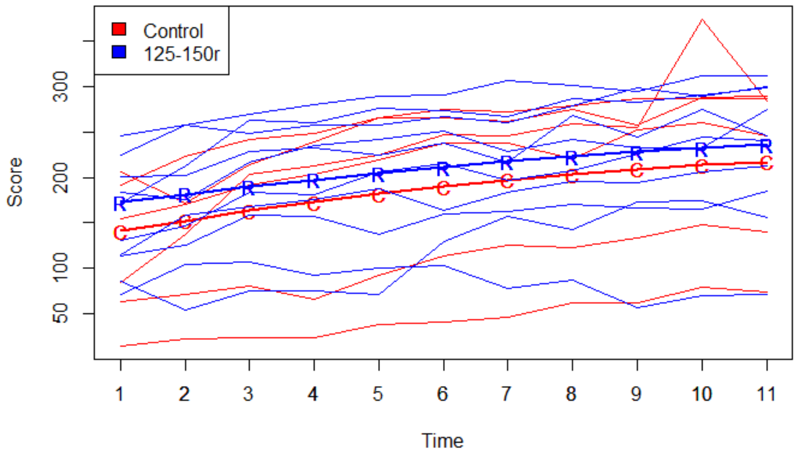


Figure 4.10: Data from the control group and the 125-150r, together with estimated growth curves in thick lines marked with "C" and "R" respectively.

Table 4.6: Results from classification of case 6 - control group vs 125-150r. Each row represents one of the sixteen runs, the true label of the test class and the classification for each of the used algorithms.

Run	**Correct Class**	**LDA4GCM**	**NN11**	**NN22_11**	**RNN11**	**LSTM11**
1	0	0	1	1	1	1
2	0	1	1	1	1	1
3	0	0	1	1	1	1
4	0	1	1	1	1	1
5	0	0	1	1	1	1
6	0	0	1	1	1	1
7	1	1	1	1	0	1
8	1	1	1	1	1	1
9	1	0	1	0	0	1
10	1	0	1	1	1	1
11	1	0	1	1	0	1
12	1	1	0	0	0	1
13	1	0	0	1	0	0
14	1	0	1	0	0	1
15	1	0	1	1	1	1
16	1	1	1	1	1	1
Accuracy		0.5	0.5	0.4375	0.25	0.5625

visible patterns in the data and the groups are not separated from each other, one may thus not expect the classifiers to perform well.

4.3 Weather Data

The last data we are going to look at is weather data provided by SMHI[1] containing temperature observations from different weather stations in Sweden over a two year time period. By studying the temperature change over time we can observe that the samples are highly correlated and that there exist a short-term periodicity, but even that the data repeats itself on a more far-reaching timescale. This is illustrated in Figure 4.11(a). It is also important to note that the weather data deviates from a repeated measurement design by only consisting of a long snippet of data taken from a sequence of observations. With this in mind, the LDA4GCM classifier won't be used for classification.

Firstly we are going to look at data from a single weather station where we try predict the temperature change given sequential time observations. Then we proceed to use data from two different weather stations where we try to predict where the sample originates from.

4.3.1 Case 7 - One station

The classification task of interest is to determine if the temperature observed in Östergötland will increase or decrease the next hour, given temperature observations of the preceding hours. More precisely, given t hours of sequential temperature observations, we will use our discussed methods to predict if the temperature at time $t+1$ either has increased or decreased compared to at time t.

The sample sizes are $n_{\text{train}} = n_{\text{test}} = 8760$ and the distribution between the class labels are approximately 47% for lower and 53% for higher. The data and results are illustrated in Table 4.7 and Figure 4.11(b), respectively.

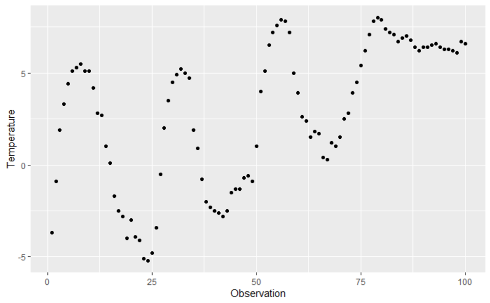
4.3.2 Case 8 - Two stations

The next task is to train our models with sequential data from two different weather stations and then test if our models can classify to which station new samples originates from. The stations of interest is placed in Malmö and at Kiruna airport with an approximate 1400 kilometers distance.

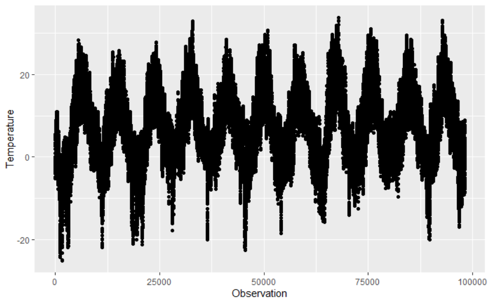
[1]Sveriges meteorologiska och hydrologiska institut.

Table 4.7: Results for weather data - case 7. *, ** and *** corresponds to $t = 10, 24$ and $t = 100$ respectively. Each row represents the classifier used, the number of parameters in the model and the accuracy for each of the three values of t.

Method	**Number of trainable parameters**	**t = 10**	**t = 24**	**t = 100**
FNN16	$193^\star, 417^{\star\star}, 1633^{\star\star\star}$	0.718	0.731	0.744
FNN32	$385^\star, 833^{\star\star}, 3265^{\star\star\star}$	0.72	0.735	0.743
FNN64	$769^\star, 1665^{\star\star}, 6529^{\star\star\star}$	0.722	0.741	0.748
FNN64_32_16	$3329^\star, 4225^{\star\star}, 9089^{\star\star\star}$	0.73	0.735	0.733
RNN16	305	0.72	0.735	0.745
RNN32	1121	0.723	0.74	0.737
RNN64	4289	0.716	0.73	0.735
LSTM16	1169	0.711	0.717	0.738
LSTM32	4385	0.712	0.723	0.739
LSTM64	16961	0.717	0.723	0.73



(a) Short-term periodicity.



(b) Long-term periodicity.

Figure 4.11: Temperature data observed over different time spans. By looking at a shorter time span (100 hours) we can observe the daily periodicity of the temperature, while the longer time span of approximately 12 years captures the yearly periodicity.

Table 4.8: Results for weather data - case 8. *, ** and *** corresponds to $t = 10, 24$ and $t = 100$ respectively. Each row represents the classifier used, the number of parameters in the model and the accuracy for each of the three values of t.

Method	**Number of trainable parameters**	**t = 10**	**t = 24**	**t = 100**
FNN16	$193^\star, 417^{\star\star}, 1633^{\star\star\star}$	0,76	0,773	0,829
FNN32	$385^\star, 833^{\star\star}, 3265^{\star\star\star}$	0,757	0,79	0,848
FNN64	$769^\star, 1665^{\star\star}, 6529^{\star\star\star}$	0,758	0,809	0,861
FNN64_32_16	$3329^\star, 4225^{\star\star}, 9089^{\star\star\star}$	0.78	0,825	0,924
RNN16	305	0,76	0,789	0,775
RNN32	1121	0,769	0,814	0,761
RNN64	4289	0,766	0,815	0,768
LSTM16	1169	0,762	0,839	0,844
LSTM32	4385	0,765	0,86	0,95
LSTM64	16961	0,768	0,89	0,997

The sample sizes are $n_{\text{train}} = 17411$ and $n_{\text{test}} = 17412$ and the distribution between the class labels are approximately 50%. The data and results are illustrated in Table 4.8 and Figure 4.12, respectively.

4.3.3 Discussion - Weather Data

All neural network models performed decently for the weather data. In case 7, all of the neural networks got similar accuracy while in case 8, we can observe significantly better results for the large neural network models as the sequence length increases, especially with the LSTM models. Note again that these samples are over different time periods, and thus, we deviate from the repeated measurement design, so the usage of growth curve modeling for case 7 and case 8 is not well suited. On the contrary, it shows the power of classification methods based on neural networks, although these methods encounter other problems such as long training times and troublesome interpretation. A perhaps more appropriate task using LDA4GCM would be if the training samples were over same time periods each day and then test samples would be observations over the few remaining hours from the corresponding days. However, due to the yearly periodicity of the temperature, it would probably still be hard to fit a proper growth curve.

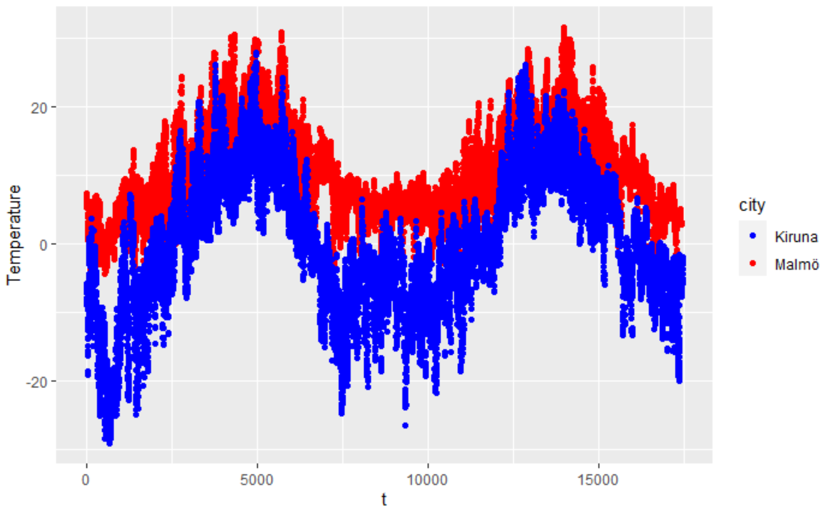


Figure 4.12: Temperature data observed for weather stations placed in Kiruna and Malmö. Each point corresponds to the temperature for one of the two stations at a given time t, and we can observe that the data consists of two long sequences rather than multiple sequential samples.

Chapter 5

Ending Discussion

5.1 General Discussion

We have introduced, discussed different types of neural networks and applied linear discriminant analysis to the growth curve model for binary classification problems. The goal was to compare these methods and by looking at the results, we can conclude that LDA4GCM indeed do get comparable accuracy with different neural networks for appropriate problems at hand, although it never outperformed all of the other classifiers.

The results in case 4-6 demonstrates the importance of looking at the data, as one can simply conclude from Figure 4.8, Figure 4.9 and Figure 4.10 that separation of the different groups will be extremely hard. One should simply not conclude that the classifiers themselves are bad, which is easily done if one only look at result tables.

Furthermore, it's interesting to consider the interpretability of the methods. Although LDA4GCM got in average approximately 7.6% lower accuracy than RNN12 in case 1, LDA4GCM only used 8 parameters to estimate the corresponding curves which indeed can be used for satisfying interpretation and illustration, see Figure 4.7(a). Comparing this to the RNN12 classifier with 181 parameters, which of none can be used for interpretation.

5.2 Further Work

As mentioned earlier, the growth curve model which we have introduced is limited by the fact that we are working with a relatively low dimensional polynomial of a prior set degree, see for instance Figure 4.7(b). Although the accuracy of the classifier almost topped 100% each run, the interpretability is limited. When the data follows a periodic pattern, such as a day-and-night cycle, one can modify the within-group design matrix (3.13) for growth curves to take the form

$$\mathbf{A} = \begin{pmatrix} 1 & \sin(c_{11}\omega) & \cos(c_{22}\omega) & \sin(c_{13}\omega) & \dots & \cos(c_{1q}\omega) \\ 1 & \sin(c_{21}\omega) & \cos(c_{22}\omega) & \sin(c_{23}\omega) & \dots & \cos(c_{2q}\omega) \\ 1 & \sin(c_{31}\omega) & \cos(c_{32}\omega) & \sin(c_{33}\omega) & \dots & \cos(c_{3q}\omega) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ 1 & \sin(c_{p1}\omega) & \cos(c_{p2}\omega) & \sin(c_{p3}\omega) & \dots & \cos(c_{pq}\omega) \end{pmatrix},$$

where ω is a multiple of π and c_{ij} are proper chosen constants for all $(i, j) = (1, 1), \dots, (p, q)$, as demonstrated in [26]. Moreover, the growth curve model presented in Section 3.2 is limited by using one dimensional observations. The growth curve model given by (3.11) can be extended to higher dimensions by considering a tensor structure for the input data by including more than one within-group design matrix. [11] discusses and demonstrates how one can extend (3.11) and estimate the corresponding parameters using two within-group design matrices, where the data has not only been measured repeatedly over time, but also at different levels.

Bibliography

[1] M. B. Danford, H. M. Hughes, and R. C. McNee. On the analysis of repeated-measurements experiments. *Biometrics*, 16(4):547–565, 1960.

[2] A. J. Dobson and A. G. Barnett. *An Introduction to Generalized Linear Models (4th ed.).* Chapman and Hall/CRC, Boca Raton, London, New York, 2018.

[3] R. A. Fisher. The statistical utilization of multiple measurements. *Annals of Eugenics*, 8(4):376–386, 1938.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks.* Springer Berlin, Heidelberg, 2012.

[7] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., New York, 2017.

[8] G. Hinton. *Neural networks for machine learning.* Coursera, video lectures., 2017. `https://www.cs.toronto.edu/~hinton/coursera_lectures.html`.

[9] R. A. Horn and C. R. Johnson. *Topics in Matrix Analysis.* Cambridge University Press, Cambridge, 1991.

[10] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[11] M. Singull J. Nzabanita, D. von Rosen. Maximum likelihood estimation in the tensor normal model with a structured mean. *Linköping University Electronic Press*, page 16, 2015.

[12] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning.* Springer New York, NY, 2021.

[13] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis (6th ed.).* Pearson Education Limited, Harlow, 2013.

[14] C. G. Khatri. A note on a manova model applied to problems in growth curve. *Annals of the Institute of Statistical Mathematics*, 18:75–86, 1966.

[15] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. 2014.

[16] T. Kollo and D. von Rosen. *Advanced Multivariate Statistics with Matrices.* Springer Netherlands, Dordrecht, 2005.

[17] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(86):2278–2324, 1998.

[19] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Müller. Efficient backprop. *Springer Berlin Heidelberg*, 1524:9–48, 1998.

[20] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. *CoRR*, abs/1709.02540, 2017.

[21] R. J. Muirhead. *Aspects of Multivariate Statisical Theory.* John Wiley Sons Inc, Hoboken, 1982.

[22] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[23] R. F. Potthoff and S. N. Roy. A generalized multivariate analysis of variance model useful especially for growth curve problems. *Biometrika*, 51:313–326, 1964.

[24] M. S. Srivastava and C. G. Khatri. *An Introduction to Multivariate Statistics.* North-Holland, New York, 1979.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[26] D. von Rosen. *Bilinear Regression Analysis.* Springer International Publishing, Cham, 2018.