Monotone Models for Prediction in Data Mining

Monotone Models for Prediction in Data Mining

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit van Tilburg, op gezag van de rector magnificus, prof. dr. F.A. van der Duyn Schouten, in het openbaar te verdedigen ten overstaan van een door het college voor promoties aangewezen commissie in de aula van de Universiteit op maandag 13 november 2006 om 14:15 uur

door

Marina Velikova Velikova

geboren op 2 april 1977 te General Toshevo, Bulgarije.

Promotores:	prof. dr. ir. H. A. M. Daniels
	prof. dr. J. P. C. Kleijnen
Copromotor:	dr. A. J. Feelders

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems (Series No. 2006-20), and CentER, the Graduate School of the Faculty of Economics and Business Administration of Tilburg University.

To my family

and

To those seeking
beauty and value
in the monotony

Preface

The monotony and solitude of a quiet life
stimulates the creative mind (Albert Einstein).

What is your first association with the term *monotone*? Given the answers of most of the people whom I have asked this question, I presume that the reply would be *boring*, *tedious*, *flat*, or another of the dozen synonyms related to everything that lacks interest. Although in some cases this might be the right association, there is a wide range of real-world situations where *monotone* implies *interesting* and *useful*. Here are a few examples.

First, *monotone* basically means *order-preserving*. A monotone property is simple and easily applied in practice. Let us consider a task where students must be seated in a classroom according to their heights: shorter get front seats and taller get back seats. Given this ordering, a new student can easily find her seat based on her height.

Second, monotone properties are often related to consistent behavior. People usually prefer consistencies in their lives, and they do not like contradictions. Typical cases occur in evaluation and selection procedures. It would not be acceptable, for example, that a higher-qualified employee receives lower wage than a lower-qualified employee with otherwise equal characteristics; or a student with higher entrance grade is rejected, whereas another student with lower entrance grade is accepted (ceteris paribus).

Finally, in our highly computerized world, the design of many devices is made to preserve the monotone properties of the input-output system. Good examples are digital-to-analog converters (DACs), which are widely used in various audio and video applications like computers, TV, Radio, CD, and MP3 Players. For instance, by converting digital (usually binary) signals into analog signals, DACs allow us to hear music stored in MP3 format through speakers. To make the conversion possible and appropriate, it is required that DACs' analog output increases with the increase in the digital input.

In this thesis I demonstrate the useful and interesting implications of monotone properties in the field of information technology. In particular, I consider monotone properties as a type of domain or expert knowledge, which can be incorporated into a data mining process to improve knowledge discovery and to facilitate decision making for end-users.

The completion of a PhD dissertation is a long journey, and I would not have been able to realize it without the continuous efforts and support of many people. It is a pleasure to convey my gratitude to them.

First, I have been very fortunate to have Professor Hennie Daniels, Professor Jack Kleijnen and Dr. Ad Feelders as my supervisors. I am much indebted to them for their invaluable guidance and encouragement during my PhD journey. Professor Daniels has undoubtedly been my strongest motivator for the beginning and the successful completion of this PhD project. His truly scientific intuition, his vast knowledge in many areas, and his assistance in academic writing inspired and enriched my growth as a researcher. I greatly enjoyed our scientific discussions, and the moments of Professor Daniels' "Let me puzzle you", which often challenged my mind to reach deep scientific analysis. In his supervision, Professor Daniels appeared to be not only a great scientist but also a kind person. I deeply appreciate his concern and support during difficult periods of my PhD journey, or when I dealt with bureaucratic problems. A very special thanks goes out to Professor Kleijnen for his timely and instructive comments at every stage of the thesis writing, allowing me to complete this work on schedule. I have benefited tremendously from working with Dr. Feelders, who served as a true mentor. I am grateful for his invaluable guidance and for being always present for discussions from the very early stage of this research. His expertise, insights, and critical thinking greatly enriched my PhD experience. I am much indebted to him for providing the Dutch translation of the summary at the end of this thesis.

Next I wish to thank the other members of my thesis committee, Professors Dick den Hertog, Jan Magnus, and Philip Franses, and Dr. Rob Potharst for their valuable feedback and suggestions to improve this thesis.

I would also like to express my appreciation to the colleagues from the Department of Information Management at Tilburg University. I want to particularly thank Bartel Van de Walle, Bert Bettonvil, Hans Weigand, Leo Remijn, and Manfred Jeusfeld for their help with ideas and support at var-

ious stages of my research and teaching activities. Furthermore, I much appreciate the kind assistance of Alice Kloosterhuis, Mieke Smulders, Ettie Barajanan, Sandra de Bruin and Eva Jonkman without whom no administrative work would have gone smoothly and in time.

I am very grateful to Emiel Caron for providing the article on monotone neural networks by Sill (1998), which is one of the main works used in this thesis.

And then there are those people whose influence cannot be directly related to this thesis, but whose support and care have provided me with the strength to successfully complete my PhD project.

During my stay in The Netherlands I met many wonderful people, who made me feel at home. I am especially thankful to: Silvia Fernandez and Gloria da Silva for making the beginning of my life in Tilburg easy and pleasant; Amar Sahoo and Mohammed Ibrahim for their invaluable help and understanding in my tough times; Lai Xu for being a great officemate and my "PhD buddy"; Andrey Vasnev for his kindness and for being my fantastic dancing partner; Akos Nagy, Aminah Santowikromo, Attila Korpos, Corrado di Maria, Kanat Camlibel, Olena Lyesnikova, Marta and Piotr Stryszowski, Reuben Jacob, Rejie George and many others for their support and the cheerful time we spent together. Thank you all for your friendship, which helped make my PhD journey more enjoyable and interesting. In The Netherlands, I was happy and proud to meet many great Bulgarian people. Among them, I am very grateful to my special friends Emilia Lazarova, Svetlana Bialkova, Boryana Inkova and her Dutch husband with Bulgarian spirit Koen Giesen for always being kind, helpful and understanding whenever I needed. Our Bulgarian gatherings are unforgettable!

I convey special acknowledgment to my Dutch friend Aldo de Moor and his family for their kindness, hospitality and support in various ways. They showed me that the cultural and language differences are not barriers to understanding one other–on the contrary, they can establish great friendly relationships of help and tolerance. I am very grateful for the wonderful *gezellige* time I spent with them on various occasions. They provided me with the unique opportunity to experience the most typical Dutch traditions. I am especially thankful to Aldo for sharing his enthusiasm and natural curiosity for life and learning. Hartelijk bedankt!

Needless to say, nothing would have been possible without my loving family. To my parents, Maria and Veliko Velikovi, I am much indebted for

their encouragement, love and persistent confidence in me provided through my entire life. They have set an example for devotion, responsibility and hard work to follow my chosen path. To my caring sister, Valentina, I am very grateful for her strong emotional support and great sense of humor that helped me to overcome the PhD hardships. Мили мамо, татко и Вале, безкрайно благодаря за вашата непрестанна любов, подкрепа и вяра в мен! Посвещавам тази дисертация на вас.

I am greatly indebted to Penka Bocheva, Svetla Paneva, and Dimitar Alexandrov for contributing to my intellectual and personal development.

Finally, I take this opportunity to thank all my teachers who showed me that education is the road to a better future.

Contents

Chapter 1

Introduction

1.1 Introduction to the field of data mining

Data Mining: One of the ten emerging technologies that will "change the world" (MIT Technology Review, 2001).

1.1.1 Definition of data mining

Thanks to the fast development of computer technology and data storage capacity, the amounts of data collected in all domains of life have increased dramatically–from supermarket transactions and credit card records to molecular bodies and images of astronomical objects. Analyzing and understanding these data provide the decision makers with a vital tool to improve the accuracy and usefulness of information for strategic decision-making. The question, however, is how to gain insight into tremendous amounts of data, and how to extract valuable information from those data.

The need for automatic approaches to effective and efficient manipulation of massive amounts of data–to turn these data into useful knowledge–led to the development of a new area in the information technology industry–*data mining*.

The data mining literature gives several formal definitions of the field (Berry and Linoff, 1997; Giudici, 2003; Hand et al., 2001; "Insightful Miner 3.0 User's Guide", 2003). Here are two of them:

> *Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner* (Hand et al., 2001).

> *Data mining is the application of statistics in the form of exploratory data analysis and predictive models to reveal patterns and trends in very large data sets* ("Insightful Miner 3.0 User's Guide", 2003).

This broad range of definitions is due to the interdisciplinary nature of data mining: data mining is a synthesis of statistics, artificial intelligence, machine learning, database technology, data visualization, etc., which explains the subjective user's perspective on the goal of the field. Nevertheless, in the variety of definitions, one can still notice some common issues related to the essence of data mining. They are discussed below by comparing data mining with mathematical statistics, which is another scientific field for analyzing data.

First, similarly to mathematical statistics, data mining is not just a tool or algorithm, but a complex process for "learning" from data that requires profound understanding and mastering. The data mining process consists of several phases, which are presented in Figure 1.1, following the CRISP-DM (CRoss-Industry Standard Process for Data Mining) reference model (Chapman et al., 2000). In the literature, often the overall process of deriving knowledge from data is called a *knowledge discovery process* and data mining is considered to be a step in it related to the application of specific methods and algorithms. In this thesis, however, the term *data mining* is used to refer to the multistage process of knowledge discovery.

As the arrows in the figure show, the sequence of phases in a data mining process is not strict: the outcome of a particular phase determines the next phase, which needs to be performed, but moving back and forth is typical in the process. The usual relationships between phases are indicated by the arrows in Figure 1.1.

Another important issue is the purpose of the data used in the analysis process. This concerns one of the fundamental differences between statistics and data mining. Statistics is concerned with analyzing data that are primarily collected for checking a hypothesis formulated beforehand, whereas data

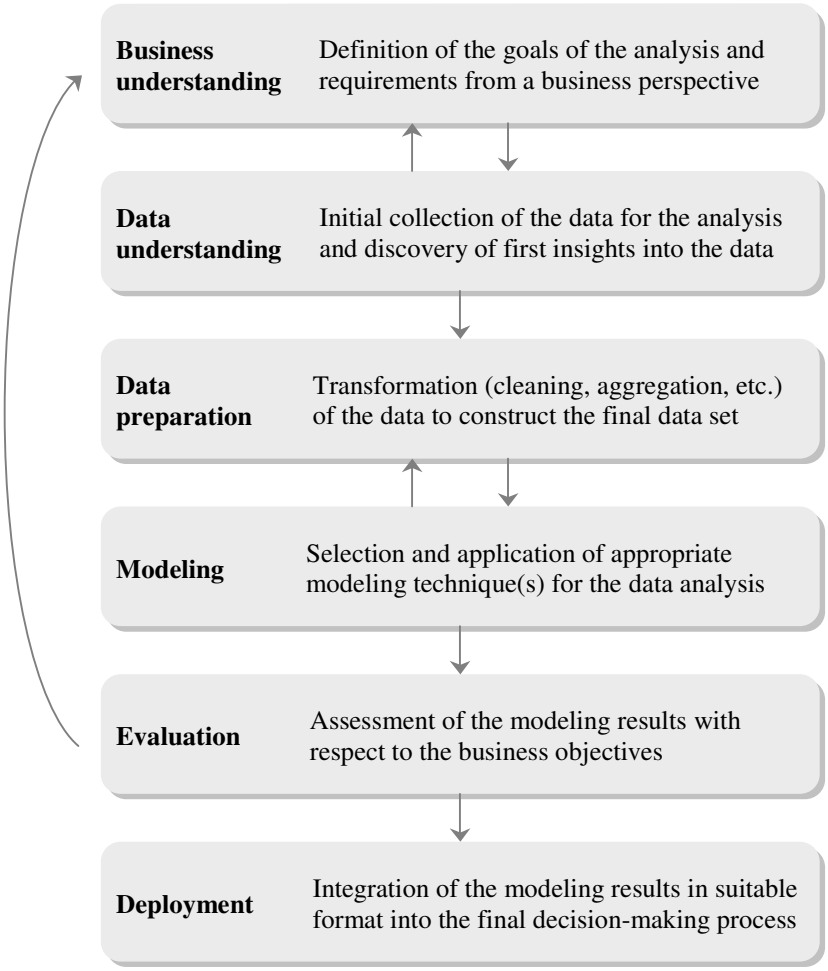


Figure 1.1: Phases of the data mining process as defined by the CRISP-DM reference model (source: Chapman et al. (2000))

mining deals with "secondary" data, i.e., data gathered for other purposes (e.g., operational), which are different from the data mining purposes.

Furthermore, data mining usually deals with vast amounts of data, from hundreds to billions of observations, and with hundreds of characteristics describing an observation. This characteristics requires appropriate and sophisticated methods for data access and analysis, which are often beyond the scope of classical statistics. In addition, the conventional requirement in statistical analysis that the data should be presented in a matrix form is not necessarily imposed in data mining (see Section 1.1.2).

Last but not least, it is essential that the results obtained from a data mining process are easy to understand by and explain to the human decision makers; moreover, these results should comply with the business objectives. For this purpose, domain experts are often involved in the development and implementation of data mining methods.

1.1.2 Data sets

As its name suggests, data mining is a process based on data. Nowadays, data are considered to be everything–from textual and numerical facts to graphics, images, sound, and video objects. Here we discuss two main issues concerning the data used in a data mining process, namely their form and type.

Form of data

There exist various data forms, for example, multi-relational data, time-series, spatial data, string sequences (e.g., DNA/RNA), and hierarchical structures, as discussed in (Hand et al., 2001). The simplest data form, however, often considered in the data mining literature is the *matrix form*. So, data are represented as an $N \times k$ matrix (N rows and k columns). The rows represent objects–such as customers, patients, transactions–and they are called *instances*, *records*, *individuals*, or *observations*. The columns contain set of measurements on each object, and they are called *variables*, *attributes*, or *features*. We call such a collection of objects, on which a set of measurements is taken, a *data set*.

Type of data: scaling

The data type is determined by the nature of the measurements on the object. Here we make a major distinction, namely between *continuous* and *discrete* data types. Continuous data are measured on a real-valued scale, whereas discrete data take values from a range that has integer values or nominal values.

Furthermore, within discrete data we distinguish between *ordinal data*, which preserve a predefined ordering, and *nominal data*, where numbers or names are used only to discriminate between different values without preserving additional properties.

Finally, measurements that can take only two values are called *binary* data.

Table 1.1 presents an example of a data set in a matrix form consisting of information on ten houses with their characteristics. The attributes VOLUME, NUMBER OF ROOMS, and PRICE are continuous; GARAGE and LOCATION are discrete (the former is ordinal and the latter is nominal); finally BRICK? is a binary attribute.

Table 1.1: Example of housing data

№	Location	Volume	Number (#) rooms	Garage	Brick?	Price in euro
1.	Rotterdam	385.2	8	Large	yes	788 500
2.	Amsterdam	156.0	5	Small	no	449 000
3.	Utrecht	90.4	3	Small	yes	169 300
4.	Rotterdam	86.3	3	Medium	yes	269 000
5.	Amsterdam	73.7	2	Small	no	225 200
6.	Amsterdam	113.0	4	Medium	yes	487 500
7.	Utrecht	201.4	5	Large	yes	560 400
8.	Amsterdam	69.5	1	Small	no	87 000
9.	Rotterdam	94.2	3	Medium	no	365 800
10.	Utrecht	100.3	4	Medium	yes	299 600

1.1.3 Data mining tasks

Although the main objective of any data mining system is the extraction of valuable knowledge from large data sets, there are different data mining tasks, which depend on the user's goals:

- *Classification and Regression*: predicting the value of one of the variables of interest, called *dependent*, *response*, or *target variable*, given the known values of the other variables, called *independent*, *explanatory*, or *predictor variables*. In classification, the variable being predicted is discrete, and it is called *class*, whereas in regression, the variable is continuous. A classification example is predicting the bond rating of a company based on its characteristics. A regression example is the estimation of the price of a house, given its attributes (again see Table 1.1).

- *Association*: finding interesting associations (relationships) between attributes in a data set. A well-known example is market-basket analysis, where the task is to find combinations of items (products) that are often purchased together.

- *Clustering*: putting objects into a number of groups–called clusters–in such a way that the objects within the same group are similar, whereas the groups are dissimilar. A typical example is market segmentation based on past purchasing behavior, demographic characteristics, or other customers' features.

- *Visualization*: exploring the data by using visual and interactive graphical techniques, such as histograms, pie charts, scatter, and contour plots. Of course, low-dimensional data are more easily displayed than high-dimensional data. In the latter case, additional methods such as principal component analysis (Jolliffe, 1986) and projection pursuit (Friedman and Tukey, 1974; Huber, 1985), are used to reduce data dimensionality or to allow projection of higher-dimensional into lower-dimensional data.

The focus in this thesis is on classification and regression problems, hereafter called *prediction problems*, in general. In addition, clustering and visualization techniques are used in the development of some of the methods presented here.

1.1.4 Data mining models, patterns, and methods

Models and patterns

The final outcome of a data mining process is knowledge, which can be presented in different ways. The usual representations are *models* and *patterns*.

Pidd (1996) broadly defines a model as follows:

> *A model is an external and explicit representation of part of reality as seen by the people who wish to use that model to understand, to change, to manage and to control that part of reality.*

In a more strict sense used in data mining, a model is a global representation of a data set that can be used for descriptive or predictive purposes

(Hand et al., 2001). In the descriptive case, the model is a simplified description of the data; examples are models for association and clustering. In the predictive case, the model represents a process that generates the data, and that is used to make inferences for future data values; examples are models used for classification and regression.

A simple example of a predictive model is the linear regression function

$$y = \beta_0 + \beta x + \epsilon, \tag{1.1}$$

which specifies the relation between variables x and y; the β's are called *parameters* of the model; ϵ is a random variable that captures the noise recorded in the data. The latter is discussed in more details in Chapter 2.

The model in (1.1) belongs to the class of *parametric models* for which a particular functional form is assumed beforehand, and which are completely specified by a set of parameters. The objective of modeling in this case is to find appropriate values for the parameters by optimizing a criterion function for fitting the data, for example, least squares. Linear parametric models have the advantage of simplicity, as they are easy to estimate and interpret. Their main disadvantage, however, is that they may produce much bias, i.e, they have systematic error (see Section 1.3.3), if the assumed functional form is inappropriate. This leads to the development of parametric non-linear models such as neural networks (discussed below), which are very flexible and accurate tools used for prediction.

In contrast to the parametric models, *non-parametric models* are data-driven and do not require a specification of the functional form a priori. On the one hand, non-parametric models prevent the construction of erroneous models caused by incorrect assumptions about the underlying function. Furthermore, non-parametric models are very flexible as they can fit (almost) any data by using a few or no nuisance parameters (parameters that are used in the modeling procedure but that are not of interest to the data analyst). On the other hand, the learning process of these models might be expensive in terms of training time and memory requirements, as all data need to be stored. In addition, the incorporation of prior knowledge might be difficult, especially for high-dimensional data, due to the lack of explicit parameters used to express such knowledge. A type of non-parametric models are decision trees, which are very well-known in practice and used in this dissertation too.

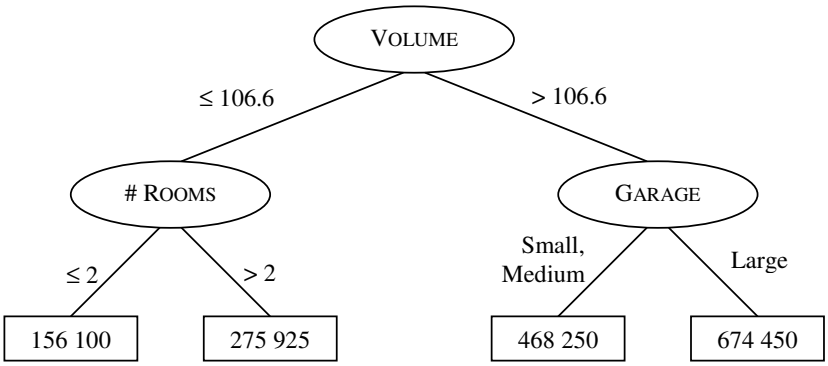


Figure 1.2: Example of a decision tree built on the housing data of Table 1.1

Between these two extremes lies the class of *semi-parametric models*, which combines features from both parametric and non-parametric models. Examples are so-called *mixture models*, which are weighted linear combinations of local parametric models built on subsets of the input space. The models presented in Chapter 5 represent this class.

Whereas models in data mining are global summaries of the data measurement space, *patterns* are local structures describing parts of this space. One of the most typical applications of patterns is the detection of unusual observations (outliers), which have values very different from the majority of the data, for example, fraud detection in banking and fault detection in industrial processes. Again, patterns can be used for descriptive or inferential purposes.

In this study, we are interested in the global nature of data for prediction problems, so we discuss models only in the remainder of the thesis. In particular, we restrict ourselves to models that are derived from two of the most popular methods used for classification and regression tasks in data mining, namely decision trees and neural networks.

Decision trees

The basic idea of tree-based models is to partition the input space by a sequence of recursive splits into a set of rectangles. For each rectangle the response variable is usually set to a constant. An example of a decision tree based on the housing data in Table 1.1 is represented in Figure 1.2.

Typical tree-based algorithms employ a top-down, greedy search strategy

for growing a decision tree. The top (starting) node of a tree is called *root*; it contains the full data set. The terminal nodes (leaves) represent the rectangles as a result of partitioning the input space; they determine the predicted value of the response variable for the data belonging to a particular rectangle (in the housing example, the average house price). The splitting of the input variables is performed in the non-terminal nodes; it can take various forms depending on

- Number of variables: the splitting is *univariate* if only one variable is tested or *multivariate* if more than one variable is tested at once.
- Number of outcome splits: two (binary) or more.
- Type of splitting variable(s): continuous or discrete.

The splitting tests are mutually exclusive and exhaustive. The selection of the variable(s) for splitting is based on a measure for the quality of the partition–the best split results in nodes among which the values of target variable vary at most; for example, if the predicted variable has a standard normal distribution, then the nodes after the best split contain means, which are very far apart from one other.

The target variable of a new observation is predicted by performing tests on the independent variables–starting from the top node until a leaf node is reached. As a result, a *decision rule* in **if-then** form is generated. The **if**-part consists of a single or a conjunction of attribute-value pairs, whereas the **then**-part contains only the predicted value of the target variable. For example, suppose we have to predict the house price of a house with the following characteristics

Location	Volume	# Rooms	Garage	Brick?
Rotterdam	98.4	3	Small	no

Based on the tree in Figure 1.2, the decision rule generated for prediction of the house price is then

if Volume ≤ 106.6 **and** # Rooms > 2 **then** Price $= 275\ 925$

One of the main strengths of decision trees is their ability to represent rules that are easy to understand by human-decision makers. In many applications it is crucial not only to make accurate prediction but also to explain

the reason for the final decision. The discovered knowledge needs to be recognized by the domain experts; this recognition requires good descriptions, such as the ones provided by decision trees. Furthermore, the selection of the variables used to construct a decision tree gives a clear indication for the set of attributes that play the most important role in the prediction of the target variable; the most important variable is at the top of the tree.

Another advantage of decision trees is that they do not require the specification of a functional form a priori. The models are derived from data, which provides flexibility of the tree construction.

Like any other data mining method, decision trees also have their application limitations. One of the main problems is to determine the right size of the final tree. The construction of large trees leads to two problems: (i) the model complexity increases, i.e., the resulting rules are very complex and hard to interpret by the end user; (ii) "overfitting" leads to bad model performance on new data, typical problem in the data mining field (see Section 1.3.3 for further discussion). Low prediction accuracy of decision trees may also be due to the lack of a sufficient amount of data.

Numerous tree-based approaches have been developed to tackle these limitations (see Section 3.2); this makes decision trees attractive methods for application in many fields; for example market and customer analysis, medicine and physics, and manufacturing data exploration.

Neural networks

The development of artificial neural networks (in short neural networks, NNs) has been inspired by the way biological nervous systems (brains) are structured and work. Consisting of a number of interconnected elements called *units* or *neurons*, neural networks process information in a parallel and distributed manner, which makes them powerful computational tools widely applied in many areas; for example, finance and business, manufacture industry, chemical and electrical engineering, and telecommunications. As in the human brain, learning in neural networks is a constant process, which is based on the adjustment of the connections among the neurons.

Whereas neural networks were originally developed with simple architectures (topologies) consisting of input and output elements only, their current successors have more complex multilayer structures. Figure 1.3 is an example of architecture of the most widely used type of neural networks, namely

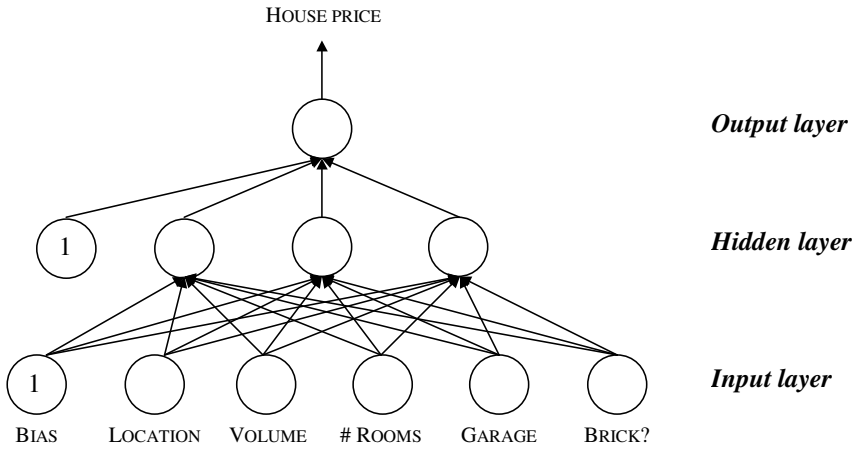


Figure 1.3: Example of a feed-forward neural network based on the housing data of Table 1.1

a *feed-forward neural network*. The example is again based on the housing data from Table 1.1.

The topology of a feed-forward neural network is based on multilayer structure with three main components

- *Input layer*: provides external input to the neural network, where every unit corresponds to an input variable, and one additional unit called *bias* set to a constant value of 1.

- *Hidden layer(s)*: transforms the input it gets from either the input layer or another hidden layer to the next layer.

- *Output layer*: produces the output of the network.

All the layers consist of a set of one or more units, which are (fully) connected with the units from the neighboring layers. All the connections between the layers are weighted. The weights are the parameters in the network model that are to be optimized.

The name "feed-forward" implies that the flow of information is one-way, i.e., from the input layer to the hidden layer(s) to the output layer; there are no feedback connections between the layers. The output from one layer serves as an input to the next layer.

This one-way processing of information determines the basic functionality of a feed-forward neural network. For every input vector, each input node passes the value of an independent variable to all the nodes of the hidden layer. Each hidden node computes a weighted sum of the input values. Furthermore, an activation or transfer function (typically sigmoid) is applied to the value thus computed to provide a bounded output of the hidden node. This computational procedure is repeated for all hidden layers. Finally, the output layer calculates a weighted sum of the inputs received from the hidden nodes connected to this output layer. In regression problems, this is often the final network's output, whereas in classification problems, sigmoid transformation is used again to determine the probability for the predicted class. The final network's output is compared with the target output and the error (difference) is propagated back to adjust the connecting weights. This procedure is called *error backpropagation* and it iterates until the error is less than a pre-determined threshold.

The architecture and functionality of feed-forward neural networks generates arguments pro and con their application. On the one hand, neural network's learning ability and flexible nature–determined by an arbitrary large number of degrees of freedom (parameters)–allow them to model complex (non-linear) functional relationships with high accuracy. On the other hand, over-parametrization usually result in modeling the noise present in the data, which leads to overfitting. Furthermore, the non-linear functional form of the network's output makes the model hard to interpret by human decision makers. Therefore, neural networks are often called black boxes.

Despite these weaknesses, in many applications where the accuracy of prediction is the main objective, neural networks are one of the most popular techniques used.

1.2 Monotonicity constraints as domain knowledge in data mining

1.2.1 Domain knowledge

The successful implementation of any data mining system depends on the outcome of each stage of the mining process. Though, the data mining literature emphasizes on the analysis and interpretation phase, other impor-

tant aspects in building a data mining system are data selection and data pre-processing. The right description of the domain, data cleaning, data integration, and data transformation can significantly improve the efficiency of the data mining process.

Besides limitations resulting from poor data quality, there can also be problems in the application of the model if the mining process is conducted by blind search. Frequently, the models derived are incompatible with business regulations. Another problem may be the lack of interpretability of the model; in general, human decision makers require models that are easy to understand so that may not accept incomprehensible models, for example very complex decision trees. Furthermore, the knowledge derived is inherently user subjective and domain dependent. In other words, the outcome from a data mining system cannot be treated only quantitatively–without understanding and interpretation.

Therefore, there is a need for integration of (i) the knowledge discovered by data mining algorithms, and (ii) the knowledge based on intuition and experience of the domain experts in order to construct comprehensible and plausible decision models. In the literature, *expert knowledge* is also referred to as *domain knowledge* or *prior knowledge*.

Several types of domain knowledge can be distinguished:

- Common sense: knowledge collected through life and working experience over time. Reasoning based on common sense is very typical for humans and it is often done unconsciously. Unfortunately computer systems cannot "draw an inference" from common sense as they do not possess such knowledge.

- Normative knowledge: knowledge related to the desired input/output and goals (e.g., simplicity, monotonicity of the outcome) of the data mining process. Usually, it is hand-coded as requirements by a domain expert. Normative knowledge can be used to constrain or prune the search space, and thereby enhances the performance of the models derived.

- Semantic knowledge: highly organized knowledge of concepts, facts and their relationships within a particular domain. It is well structured and formally represented as a hierarchy; for example, in organizations like universities the hierarchy starts with the university as whole at the

highest level, followed by faculties, departments, groups, etc. This facilitates easy inference of causal relationships among facts and concepts at a lower and higher level of abstraction.

In this study we focus on normative knowledge, which can be incorporated in several ways and at different stages in a data mining process.

First, the role of normative knowledge may be crucial for the design of a data mining process where the aim is to determine the most effective way(s) for knowledge discovery. Various requirements can provide mechanisms (instruments) to guide the process, which may lead to restricting the search space of plausible solutions, reducing human and computational costs, saving time, better managing and understanding of the whole knowledge discovery process, etc.

At the data pre-processing stage, the use of normative knowledge might be also necessary. As mentioned in Section 1.1.2 data in data mining are usually represented by a single table in a matrix form. However, there are domains where data are organized in a (multi-)relational structure corresponding to several databases, which are connected in 1:M or M:M relations. Then, it is necessary to combine (aggregate) all these databases to obtain one single "flattened" source (Feelders et al., 2000).

Finally, one of the simplest approaches to apply normative knowledge in data mining is by imposing various constraints on the data used or the model built. We give three examples.

- Constraints for the values of the attributes; for instance, define a range or a set of permissible values. In the housing example, the number of rooms must take positive integer values.

- Constraints for the attributes and the relationships among them; for example, attribute(s) can be excluded or combined in the mining process. In the housing example, kitchen space cannot be larger than the total house space.

- Constraints for the model built. In practice, the objective of data mining is to obtain models that are novel, valid and useful. Furthermore, if for a particular problem there are two or more models that give plausible solutions, then the simplest one is chosen as a final model (Occam's razor principle for simplicity). However, given the particular task at

hand, there may be additional constraints such as interpretability, efficiency, and misclassification costs of the model. Last but not least, it is often required that the models built preserve certain relationships between the predictor and target variables known a priori. In the housing example, the predicted house price is expected to increase with the increase of the house volume.

Enforcing constraints on the decision models can significantly improve the data mining process by making it more accurate, robust, and transparent. Therefore, in this thesis, we consider a special type of constraint that is typical in decision problems, namely *monotonicity constraints* described in more details in the next section.

1.2.2 Monotonicity constraints

The motivation for considering monotonicity constraints in this research is based on the following observations:

1. **Monotonicity is common in scientific disciplines (domains)**. Monotonicity is a simple and intuitive property stating that the greater an input is, the greater the output must be, all other inputs being equal (ceteris paribus). For example, given the data in Table 1.1, the increase of the volume of a house would lead to increase of the house price. This is so-called *increasing monotonicity*. Similarly, *decreasing monotonicity* is defined whenever an input increases, the output decreases (ceteris paribus). Without loss of generality, we consider only increasing monotonicity.

   Monotonicity properties are known frequently in various scientific domains:

   - Business and Economics: Economic theory would state that people tend to buy less of a product if its price increases (ceteris paribus), so there would be a negative relationship between price and demand. Another well-known example is the positive dependence of labour wages on age and education (Mukarjee and Stern, 1994). In loan acceptance, the decision rule should be monotone with respect to income, i.e., it would not be acceptable policy that a high-income applicant is rejected, whereas a low-income

applicant with otherwise equal characteristics is accepted. Monotonicity is also common in so-called hedonic price models where the price of a consumer good depends on a bundle of characteristics for which a valuation exists (Harrison and Rubinfeld, 1978). In house pricing, for instance, the price of a house increases with the house area, and decreases with the distance to the city center. Another example is option pricing, where the price of an American call option is a monotone increasing function of the duration and the price of the underlying asset, and a decreasing function of the strike price (Gamarnik, 1998).

- Operations research: It is well known that more traffic on the road or more customers at a supermarket leads to more waiting time.
- Computer Science: Monotone relationships are present in diagnosing performance-problems of computer systems, e.g., paging delays increase with the number of logged-on users (Hellerstein, 1989).
- Law systems: An example of a law application where the factors (attributes) have monotone influence on the result of a judgment process is a wage-earner system (Karpf, 1991). The objective of the system is to classify an employee as either a wage-earner or not (part-time operators, independent sales-consultants, etc.), based on a number of factors. This is done for the purposes of the employment law where wage-earner employees are entitled to a substantial holiday allowance. In this system, for example, factors such as "Working at the liability of the employer" and "Employer has authority to instruct the employee" have monotone effects on the judgment whether an employee is a wage earner. In other words, the change of the value assigned to these factors from *no* to *yes* implies a tendency of the result of the judgment to become *yes*.
- Natural sciences: Numerous examples exist here. For instance, the body size of an animal is in a monotone relationship with its maintenance requirement, i.e., the larger the animal, the higher the amount of energy required to keep the animal alive for movement, production of body warmth, etc., without increasing or de-

creasing the body weight. Furthermore, for animals of the same size young animals need proportionally more feed for maintenance and of better quality than older animals. Another example is the effect of the increase in the human body weight that leads to substantial increase of the risk of heart disease, cancer, or other chronic diseases (NIH Report, 1998).

2. **Monotonicity improves the decision-making process**. The application of the monotonicity principle considerably reduces the amount of data needed by human-decision makers or inductive systems to make accurate judgments (Ben-David et al., 1989; Karpf, 1991). This speeds up the decision-making process without worsening its correctness.

   Furthermore, taking into account monotone relationships between the dependent and independent attributes, allows us to fill in missing attribute values in the data set as well as to make plausible predictions about objects that are not present at the data at hand (Ben-David et al., 1989; Moshkovich et al., 2002). This improves the quality of the data and their analysis.

3. **Monotone decision models perform better than non-monotone models**. For problems with monotonicity properties, monotone models outperform their non-monotone counterparts:

   - Monotone models are easier to understand as they agree with the decision makers' expertise; in other words, non-monotone models are much harder to interpret as they present inconsistent and less intuitive dependencies (Feelders, 2000; Potharst and Feelders, 2002).
   - Enforcing monotonicity of the models removes noise, resolves inconsistencies, and suppresses overfitting. As a result monotone models give better predictions, i.e., have smaller error rates, on new data (Sill, 1998).
   - Monotone models have less variability upon repeated sampling (known as stable in data mining). The monotonicity leads to reduction in the variance and hence the models derived are more stable (Sill, 1998).

1.3 Monotonicity in prediction problems and models

Now, we formally introduce the key concepts discussed in this thesis.

Let $\mathcal{X} = \prod_{i=1}^{k} \mathcal{X}_i$ be an *input space* represented by k attributes (features). A particular point $\mathbf{x} \in \mathcal{X}$ is defined by the vector $\mathbf{x} = (x_1, x_2, \ldots, x_k)$, where $x_i \in \mathcal{X}_i$, $i = 1, 2, \ldots, k$.

Furthermore, a totally ordered set of labels $\mathcal{L}$ is defined. In the discrete case, we have $\mathcal{L} = \{1, 2, \ldots, \ell_{max}\}$ where ℓ_{max} is the maximal label. Note that ordinal labels can be easily quantified by assigning numbers from 1 for the lowest category to ℓ_{max} for the highest category. In the continuous case, we have $\mathcal{L} \subset \Re$ or $\mathcal{L} \subset \Re^+$. Unless the distinction is made explicitly, the term *label* is used to refer generally to the dependent variable irrespective of its type (continuous or discrete).

Next a function f is defined as a mapping

$$f : \mathcal{X} \to \mathcal{L}$$

that assigns a label $\ell \in \mathcal{L}$ to every input vector $\mathbf{x} \in \mathcal{X}$. Hence, f is the underlying model.

In prediction problems, the objective is to find an approximation $\hat{f}$ of f as close as possible; for example in L_1,L_2, or L_∞ norm. In particular, in regression we try to estimate the average dependence of ℓ given $\mathbf{x}$, $\mathcal{E}[\ell|\mathbf{x}]$, whereas in classification, we look for a discrete mapping function represented by a classification rule $r(\ell_{\mathbf{x}})$ assigning a class ℓ to each point $\mathbf{x}$ in the input space.

In reality, the information we have about f is mostly provided by a data set $D = (\mathbf{x}^n, \ell_{\mathbf{x}^n})_{n=1}^N$, where N is the number of points, $\mathbf{x} \in \mathcal{X}$ and $\ell_{\mathbf{x}} \in \mathcal{L}$. In other words, $X = \{\mathbf{x}^n\}_{n=1}^N$ is a set of k independent variables represented by an $N \times k$ matrix, and $L = \{\ell_{\mathbf{x}^n}\}_{n=1}^N$ is a vector with the values of the dependent variable. In this context, D corresponds to a mapping $f_D : X \to L$ and we assume that f_D is a close proximity of f. Ideally, f_D is equal to f over X, which is seldomly the case in practice due to the noise present in the data (see Chapter 2).

Hence, our ultimate goal in prediction problems is restricted to obtaining a close approximation $\hat{f}_{M_D}$ of f by building a prediction model M_D from the given data D.

Furthermore, the main assumption we make here is that f exhibits monotonicity properties with respect to the input variables; therefore, $\hat{f}_{M_D}$ should also obey these properties in a strict fashion.

In this study we distinguish between two types of problems, and their respective models, concerning the monotonicity properties. The distinction is based on the set of input variables, which are in monotone relationships with the response:

1. *Totally monotone prediction problems (models)*: f ($\hat{f}_{M_D}$) depends monotonically on *all* variables from the input space.
2. *Partially monotone prediction problems (models)*: f ($\hat{f}_{M_D}$) depends monotonically on *some* variables from the input space but *not* on all.

Though this distinction is also made in the literature (e.g., by Tuy (2000)), we want to emphasize that the terms "totally" and "partially" refer to the set of inputs for which monotone relationships hold with respect to the target– not to the monotonicity property as such. Furthermore, we omit "totally" from the name of the first type of problems (models) in the remainder of this thesis.

1.3.1 Monotone prediction problems and models

Suppose $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ and there exists a total ordering $\geq_i$ on $\mathcal{X}_i$, for $i = 1, 2, \ldots, k$. We say that $\mathbf{x}$ *dominates* $\mathbf{x}'$ if $\forall_{1 \geq i \geq k}$, $x_i \geq x'_i$, in short expressed as $\mathbf{x} \geq \mathbf{x}'$. The dominating relationships define *a partial ordering* on $\mathcal{X}$. Unless $k = 1$, the ordering is partial rather than total because there exist points $\mathbf{x}$ and $\mathbf{x}'$ such that neither $\mathbf{x} \leq \mathbf{x}'$ nor $\mathbf{x} \geq \mathbf{x}'$.

A monotone problem is defined by the partial ordering of the input space $\mathcal{X}$ and a function f that is monotone in all input variables. This is represented by the constraint

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}: \quad \mathbf{x} \geq \mathbf{x}' \Rightarrow f(\mathbf{x}) \geq f(\mathbf{x}'). \tag{1.2}$$

In particular, f is $\mathcal{E}[\ell|\mathbf{x}]$ in regression tasks and $r(\ell_{\mathbf{x}})$ in classification tasks, respectively.

Given a data set $D = (\mathbf{x}^n, \ell_{\mathbf{x}^n})_{n=1}^N$, we call M_D a *monotone model* if the approximation $\hat{f}_{M_D}$ of f satisfies the following condition:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}: \quad \mathbf{x} \geq \mathbf{x}' \Rightarrow \hat{f}_{M_D}(\mathbf{x}) \geq \hat{f}_{M_D}(\mathbf{x}'). \tag{1.3}$$

There have been developed several methods that incorporate monotonicity constraints such as decision trees (Ben-David, 1995; Bioch and Popova, 2002; Cao-Van and De Baets, 2003; Feelders, 2000), neural networks (Kay and Ungar, 2000; Sill, 1998; Wang, 1994; Daniels and Kamp, 1999), isotonic regression (Ayer et al., 1955; Robertson et al., 1988), regression with polynomials (Siem et al., 2005), rational cubic interpolation of one-dimensional functions (Sarfraz et al., 1997), rough sets (Popova, 2004). In this thesis we consider two types of monotone models, namely monotone decision trees (Chapter 3) and monotone neural networks (Chapter 4).

1.3.2 Partially monotone prediction problems and models

Suppose $\mathcal{X} = \mathcal{X}^m \cup \mathcal{X}^{nm}$ with $\mathcal{X}^m = \prod_{i=1}^{m} \mathcal{X}_i$ and $\mathcal{X}^{nm} = \prod_{i=m+1}^{k} \mathcal{X}_i$ for $1 \leq m < k$. Furthermore, let $\mathbf{x}^m \in \mathcal{X}^m$, and $\mathbf{x}^{nm} \in \mathcal{X}^{nm}$. Then a data point $\mathbf{x} \in \mathcal{X}$ is represented by $\mathbf{x} = (\mathbf{x}^m, \mathbf{x}^{nm})$.

A partially monotone problem is defined by the partial ordering of $\mathcal{X}^m$ and a function f that is monotone in all input variables in $\mathcal{X}^m$. This is represented by the constraint

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}: \quad \mathbf{x}^{nm} = \mathbf{x}'^{nm} \text{ and } \mathbf{x}^m \geq \mathbf{x}'^m \Rightarrow f(\mathbf{x}) \geq f(\mathbf{x}'). \tag{1.4}$$

Similarly, given a data set $D = (\mathbf{x}^m, \mathbf{x}^{nm}, \ell_{\mathbf{x}})^N$, where $\ell_{\mathbf{x}}$ is the label of $\mathbf{x}$, we call M_D a *partially monotone model* if the approximation $\hat{f}_{M_D}$ of f satisfies the following condition:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}: \quad \mathbf{x}^{nm} = \mathbf{x}'^{nm} \text{ and } \mathbf{x}^m \geq \mathbf{x}'^m \Rightarrow \hat{f}_{M_D}(\mathbf{x}) \geq \hat{f}_{M_D}(\mathbf{x}'). \tag{1.5}$$

In Chapter 5 we propose a method for building a class of partially monotone models based on neural networks.

1.3.3 Monotonicity and model evaluation

Once a model is built, the next major step in the data mining process is the evaluation of the model performance, which determines whether or not the model will be employed in practice. Therefore, it is crucial to define appropriate techniques for assessing the quality of the results obtained from

the modeling step. These techniques should provide the end user with direct, truthful and detailed insight into the model performance.

Given the objective of this thesis, we consider evaluation techniques restricted to prediction tasks. From this perspective, the *predictive accuracy* of the models built is one of most important characteristics that need to be assessed. In other words, we seek models that can make as correct future predictions as possible, i.e., models with good *generalization capabilities*.

The question is how to measure the generalization performance of a model? Recall that the information we have is the (historical) data on which the model is built. Hence, these data are also the source for our model evaluation.

Suppose we have a prediction model M_D built on a data set D for estimating the dependent variable $\ell_{\mathbf{x}}$ given the set of explanatory variables $\mathbf{x}$. Then the quality of the estimator $\hat{f}_{M_D}(\mathbf{x})$ based on M_D is measured by the so-called *prediction error* computed as the deviation of $\hat{f}_{M_D}(\mathbf{x})$ from the target $\ell_{\mathbf{x}|D}$ given in D.

In regression problems, the prediction error is usually taken to be the *mean-squared error* (MSE):

$$\mathrm{MSE}(\mathbf{x}) = (\ell_{\mathbf{x}|D} - \hat{f}_{M_D}(\mathbf{x}))^2. \tag{1.6}$$

In classification problems, the simplest and most commonly used prediction error is the *misclassification* (0–1) *loss function* (*Miscl*):

$$Miscl(\mathbf{x}) = \begin{cases} 0 & \text{if } \ell_{\mathbf{x}|D} = \hat{f}_{M_D}(\mathbf{x}), \\ 1 & \text{otherwise.} \end{cases} \tag{1.7}$$

We use the expressions in (1.6) and (1.7) to measure the prediction error of models for regression and classification problems, respectively.

Furthermore, given D and M_D, the prediction error can be represented as a sum of three components. As shown by Geman et al. (1992), MSE in (1.6) can be decomposed:

$$\begin{aligned} \mathrm{MSE}(\mathbf{x}) &= \mathcal{E}_D[(\ell_{\mathbf{x}|D} - f(\mathbf{x}))^2] \\ &\quad + (f(\mathbf{x}) - \mathcal{E}_D[\hat{f}_{M_D}(\mathbf{x})])^2 \\ &\quad + \mathcal{E}_D[(\hat{f}_{M_D}(\mathbf{x}) - \mathcal{E}_D[\hat{f}_{M_D}(\mathbf{x})])^2] \\ &= \sigma_\epsilon^2 + \mathrm{Bias}^2 + \mathrm{Variance} \end{aligned} \tag{1.8}$$

The term σ_ϵ^2 is the variance of the target around its true mean, i.e., this is the variance of the noise term ϵ. This is so-called *irreducible error*, which cannot be avoided (unless $\sigma_\epsilon^2 = 0$). The second term is the *squared bias*, which gives the difference between the true function value and the average estimate over all data samples of a fixed size. The last term is the *variance* of an estimate obtained for a particular data set around its mean.

The decomposition of the misclassification error in (1.7) is derived in (Kohavi and Wolpert, 1996); using our notation, this decomposition is

$$\begin{aligned} Miscl(\mathbf{x}) = \sum_{\mathbf{x}} \Pr(\mathbf{x}) \Big[& \frac{1}{2}\big(1 - \sum_{\ell \in \mathcal{L}} \Pr(\ell_{\mathbf{x}|D} = \ell)^2\big) \\ & + \frac{1}{2} \sum_{\ell \in \mathcal{L}} \big[\Pr(\ell_{\mathbf{x}|D} = \ell) - \Pr(\hat{f}_{M_D}(\mathbf{x}) = \ell)\big]^2 \\ & + \frac{1}{2}\big(1 - \sum_{\ell \in \mathcal{L}} \Pr(\hat{f}_{M_D}(\mathbf{x}) = \ell)^2\big) \Big] \\ = \sum_{\mathbf{x}} \Pr(\mathbf{x}) & \big[\sigma_\epsilon^2 + \text{Bias}^2 + \text{Variance}\big], \end{aligned} \tag{1.9}$$

where $\Pr(\cdot)$ denotes a probability.

The expressions in (1.8) and (1.9) represent the so-called *bias-variance decomposition* of the prediction error, which have been extensively discussed in the literature (Geman et al., 1992; Kohavi and Wolpert, 1996; Hastie et al., 2001; Feelders, 2002). Here we briefly present the main idea of this decomposition, which will facilitate the later discussion in this thesis.

Although the bias-variance decomposition of the prediction error cannot be applied in practice–because the noise σ_ϵ^2 and the true function are unknown–it has an important implication for the understanding of the performance of the models obtained.

Since noise is intrinsic to real data and we cannot do much about it, we consider the other two terms in the prediction error in more details. First it is necessary to note that the squared bias and the variance have opposite influence on each other: the decrease in the one leads to the increase of the other. "Bias" of a model is related to its accuracy, i.e., an incorrect model leads to high bias. In order to reduce the bias, one needs to increase the flexibility of the model by, for example, increasing the size of the decision tree or introducing more parameters in the neural network, so that the model better fits the data. However, highly flexible models tend to be unstable due

to their high variance, i.e., the results obtained from them will show much variation if they are presented with other data samples of the same size. Hence, the so-called *bias-variance dilemma* rises, which is one of the crucial issues in the model construction stage. The optimal choice of the level of the model's flexibility or complexity determines the model performance. Of course, this choice is not trivial in practice.

In this context, monotonicity plays an important role. On the one hand, imposing monotonicity constraints on the data mining models leads to much lower model variance because the results obtained from different data sets preserve a main property in the true function. On the other hand, the variance reduction is not expected to lead to significant increase of the bias because there is no high deviation from the target. Hence, the overall prediction accuracy of models with monotonicity constraints is supposed to be, in general, better than that of unconstrained models.

Until now, we considered prediction errors from a computational point of view. As mentioned earlier, we are interested in the generalization capabilities (accuracy of future predictions) of the models built. Usually we have only one data source on which a model is built and we do not have information about the new data that may occur in reality. Hence, the problem is how to measure the generalization prediction error of a model. An intuitive solution is to use the same data again but this time to estimate the error. Our objective is then to minimize this error by constructing a model that fits perfectly the data at hand. In this way, however, we have also been modeling the noise inherently present in the data. This phenomenon is known as *overfitting*. To solve this problem, several approaches have been developed depending on the type of data mining models; for example, pruning of decision trees (Breiman et al., 1984), regularization methods for neural networks (Bishop, 1997).

In this study, we employ the most popular method in practice, which is based on the random partitioning of the original data into three sets, namely training, validation and test sets (Hastie et al., 2001). The *training set* is used to build various models. The best one is selected on the basis of the minimum prediction error computed on the *validation set*. Then the generalization error of the final model is measured on the *test set*. In general, this random splitting of the data is performed a number of times; the overall average is computed as a final error estimate. This procedure implies that the obtained error is an honest measure for the generalization capabilities of

the model.

Another important issue in model evaluation is the extent to which the results obtained from data mining models comply with the human expertise and business objectives. Incorporation of monotonicity as a type of domain knowledge in data mining plays an important role because it prevents results that are contrary to the knowledge of human experts. Hence, models that preserve monotone relationships are preferred for future predictions.

1.4 Research objectives

Our *general research objective* is to study the incorporation of monotonicity constraints as a way to express domain knowledge in a data mining process.

Given the description of the data mining process in Figure 1.1, there are two stages where monotonicity can be incorporated, namely data preparation and modeling. Hence, our general objective can be decomposed into the following two more specific goals.

Research objective - 1 Preprocessing (transforming) data such that they obey monotonicity constraints before using the data to build monotone decision models.

Research objective - 2 Enforcing monotonicity in data mining models based on decision trees and neural networks for prediction tasks.

Based on the formulation of these two research objectives, we define a number of research questions. Related to *Research objective - 1* are the following questions:

Given a data set at hand,

- How can we measure its degree of monotonicity?
- How can we transform this data set from non-monotone into monotone?

With respect to *Research objective - 2*, the research questions are:

- How can we build monotone models?
- How can we build partially monotone models?

In addition to these questions, we want to test the following three hypotheses:

Hypothesis - 1 For monotone problems, monotone models have superior predictive performance to non-monotone models.

Hypothesis - 2 For monotone problems, monotone models derived from monotone data (i.e., data obtained after the transformation) outperform monotone models derived from the original data, i.e., the former are more accurate and their variance on new data is lower.

Hypothesis - 3 For partially monotone problems, partially monotone models have superior predictive performance to non-monotone models.

1.5 Research methodology

To answer our research questions and accomplish the objectives of this study, we apply a research methodology that is based on the development of theoretical concepts and practical computational methods. With respect to the latter, some of the methods we shall propose in later chapters are novel for the field (e.g., the procedure for testing monotonicity of data and the greedy algorithm for relabeling in Chapter 2, and the approach for partial monotonicity in Chapter 5), whereas other methods are extensions of existing approaches (e.g., monotone trees in Chapter 3 and monotone networks in Chapter 4).

Typically, in the field of data mining and any other quantitative research study, new methods need to be validated in order to demonstrate their performance: how accurate, how efficient and how fast are they (Galliers, 1992; Vogel and Wetherbe, 1984). For the purpose of this study this validation is provided by the following two research approaches.

1. *Simulation experiments.* Our simulation studies are designed to demonstrate (i) the performance of our methods and (ii) the sensitivity of the performance to change(s) in the input or internal parameters of the approaches developed. The main advantage of the simulation is that the conditions and the design of the experiments are well controlled. This allows us to study the relationship between different factors and provides insight for the anticipated performance of the methods in situations that have not yet occurred in practice.

2. *Real-world applications.* We use four case studies in this research, namely two for monotone problems and two for partially monotone problems. Furthermore, we illustrate the application of our methods through real data for both regression and classification tasks.

1.6 Thesis outline

Although some of the work presented here has been already published (Velikova and Daniels, 2004; Daniels and Velikova, 2003, 2006; Velikova et al., 2006a, 2006b), this thesis is not organized as a collection of separate papers. There are four main chapters, which are devoted on separate topics but they are all related by commonly defined notations and concepts. Each chapter begins with an introduction, which establishes the main concepts discussed in that chapter. It then proceeds with presenting earlier work related to the topic of that chapter, followed by a description of the methods we propose. Each chapter ends with a summary of the work presented in it.

Chapter 2 introduces main definitions and theoretical concepts related to monotone and noisy (non-monotone) data. Benchmark measures for the degree of monotonicity of a data set are derived, which are used for comparison with indicators obtained from real data. Furthermore, a greedy algorithm to transform non-monotone into monotone data is presented. Simulation and real case studies are used to demonstrate the application of the methods proposed.

Chapters 3 and 4 present methods for deriving monotone models based on decision trees and neural networks, respectively. These methods are based on existing approaches, which we extend to deal with both classification and regression problems.

Chapter 5 deals with the concept of partial monotonicity. The main theoretical contribution is an algorithm for building partially monotone models. Simulation and real case studies are used to demonstrate the application of the method.

Finally, Chapter 6 presents general conclusions of our research and discusses possible future developments of the current work.

1.7 Thesis contributions

We discuss the contributions of this thesis from two perspectives, namely research perspective, and business and user's perspective.

Research perspective

Although there have already been several studies in the literature dealing with the incorporation of monotonicity in data mining, this thesis contributes to the development of the research field in several ways.

The first contribution we propose is *a novel straightforward procedure to test the degree of monotonicity of a real data set*. The procedure is based on a comparison between the observed and benchmark measures for monotonicity we derive. Two measures for monotonicity are considered, namely fraction (percentage) of monotone pairs and number of monotone points. The benchmark measures are computed from data, which are defined simply by taking the same structure (values) of the independent variables as in the original data set and a random permutation of the set of the original labels. If the observed measures obtained are significantly larger (this is checked by a statistical test) than the benchmark measures, then we can conclude that the original data exhibit monotonicity properties; otherwise, monotonicity assumptions are questionable. Compared to previous approaches, our testing procedure has two main advantages: (i) the comparison analysis between the observed and benchmark measures is independent of any assumptions about the functional form for the data generating process, and (ii) it does not require modeling the data beforehand.

Our second major contribution is *a greedy algorithm for making data monotone*. Given a data set with a number of monotonicity violations, we can simply change (relabel) the values of the dependent variable of some of the points in order to resolve the inconsistencies. We argue that such transformation leads to monotone data that are source for building better (more accurate and stable) monotone prediction models than the ones derived from the original (non-monotone) data.

In order to provide such comparison analysis, we construct monotone models for classification and regression based on decision trees and neural networks. The algorithms we use are *enhanced versions of two existing ap-*

proaches, namely Feelders (2000) for monotone decision trees and Sill (1998) for monotone neural networks. The extension of these methods we consider as our third contribution to the research field.

The fourth contribution of this thesis is *the approach for building partially monotone models*. It is based on the convolution of monotone neural networks built on the variables that are in a monotone relationship with the response variable and weight functions built on the other variables. To the best of our knowledge, this is the first method that deals with mixture-of-networks modeling with partial monotonicity constraints. We prove that our partially monotone models have universal approximation capabilities. Simulation and real case studies show that our approach has significantly better performance than partially monotone linear models. Furthermore, the incorporation of partial monotonicity constraints not only leads to models that are in accordance with the decision maker's expertise, but also reduces considerably the model variance in comparison to standard neural networks.

Our final contribution is *the formal proof for the universal function approximation capabilities of three-layer neural networks with a combination of minimum and maximum operators over linear functions*. We show this for two types of network: (i) without any constraints on the weights, and (ii) with monotonicity constraints on some of the weights. The latter is an alternative method to our approach for building partially monotone models.

Business and user's perspective

The success of a data mining process is measured with respect to the business objectives that are achieved, and the acceptance of the knowledge discovery results by the end users. Therefore it is crucial to guarantee that the data mining models derived meet the business requirements, comply with business regulations and they agree with the human decision-maker's expertise.

Hence, our research contributes to improve business practice and decision-making processes by providing methods for incorporating domain knowledge into a data mining process. In particular, monotonicity constraints are enforced by building models based on decision trees and neural networks. This leads to better accuracy, robustness, or interpretability of the decision models.

Furthermore, our procedure for testing the degree of monotonicity of a

data set facilitates the data mining process at the data preprocessing step where the suitability of a data set for deriving monotone decision models is determined. In other words, if the tests indicate that the data at hand do not exhibit monotonicity properties, then these data are not used for further analysis in monotone problems. Thus, by using only appropriate data, we can considerably improve the knowledge discovery process and obtain more accurate and plausible results.

Finally, the approach we propose for transforming non-monotone into monotone data resolves inconsistencies in the data and thus provides the user with an unambiguous source of information for decision-making.

Chapter 2

Monotone and noisy data

The successful implementation of any data mining system depends to a large extent on the quality of its main source–data. Given the objectives of our study, in this chapter we discuss monotone and noisy (non-monotone) data as a source for building monotone models. We focus on two main issues: (i) how to measure the degree of monotonicity of a data set; (ii) how to make data monotone. We begin with definitions of the main terms related to monotone/non-monotone data. Next, we proceed with a review of previous studies dealing with the notion of monotonicity of the data; in particular, we discuss studies that are related to measuring the degree of monotonicity of data, and making data monotone. We provide a summary of the advantages and disadvantages of these works, and we justify the need for the development of our methods. The first major contribution we propose is a new procedure to measure to what extent a data set is monotone. The second contribution is a greedy algorithm to make data monotone by changing the labels (relabeling) of some of the data points. We conduct simulation studies with artificial data in order to demonstrate the algorithm's ability to restore to a large extent the original monotone data by removing the noise. Finally, we present two case studies on bond rating (classification problem) and house pricing (regression problem) in order to illustrate the application of the approaches we propose in this chapter. The greedy algorithm for relabeling, the simulation and real case studies have been published in Velikova and Daniels (2004) and Daniels and Velikova (2006).

2.1 Introduction

Consider the following table with information on five houses:

Table 2.1: A sample of house pricing data

No	Area	Number of rooms	Volume	Price (Euro)
1	90	2	210	121 000
2	86	2	255	130 500
3	125	3	320	119 750
4	210	4	405	165 200
5	174	3	373	190 000

Suppose the analysts would like to use these data to make future predictions for the price of a house based on its characteristics. Common sense suggests a monotone dependency between the three house attributes given in Table 2.1, and the house price: more area, rooms, and volume result in a higher price, in general. However, the data in Table 2.1 show a number of inconsistencies, i.e., pairs of observations contradicting the property of monotonicity: the characteristics of the third house have larger values than those of the first two houses, but the prices are in opposite order; the fourth and fifth houses show a non-monotone relationship too.

Inconsistencies may have several reasons:

- Noise; for example, errors made during data entry, data collection, or on a measurement process; discrepancies due to the change of data over time and inconsistencies after merging data sets.

- Incompleteness, i.e., there might be a latent relationship between the dependent variable and missing information. For instance, factors not listed in Table 2.1 such as availability of a shopping center and sport facilities, may have an important effect on the house price, which is not explicitly shown in the recorded data.

To verify the monotonicity assumptions, we would like to check the extent to which monotone relationships are present in the data under study. One obvious question, therefore, is how to measure the degree of monotonicity of a data set. A straightforward method is to compute the fraction of monotone

pairs with respect to the total number of comparable pairs in the data. Another measure is the number of monotone points (see Definition 2.1.2). Besides measures computed from the data, we also need benchmarks for comparison. In Section 2.3, we derive such benchmark measures to express the degree of monotonicity of the so-called *benchmark data*–data that are defined by taking the same structure (values) of the independent variables as in the original data set and labels that are a random permutation of the set of the original labels.

By comparing the values of the actual indicators with benchmark measures, one can verify the monotonicity of the data under study. We do so by performing a statistical test for the significance of the difference. If the benchmark measures are not significantly different from the actual indicators, then the data set is highly non-monotone and the assumptions for monotonicity are questionable. In this case, it is not appropriate to use the data for building monotone models. However, if the comparison shows that there is only a small fraction of instances in the real data contradicting the monotonicity assumption, this confirms what is expected by theory. Then, the presence of conflicting (inconsistent) pairs in the actual data indicates noise in the data.

Using these noisy data for building monotone models for prediction can mislead the decision-making process and produce a unreliable output with a high variance on new data. Therefore, an appropriate modification of the data–such as transformation from non-monotone into monotone data–can remove the noise, better capture implicit dependences between missing information and the decision variable, and reduce the model variance.

In Section 2.4, we present a greedy algorithm for transforming non-monotone into monotone data by changing the value of the dependent variable. We call this process *relabeling*. The idea is to reduce the number of inconsistent pairs by relabeling one data point in each step–until the data set is monotone (see Definition 2.1.3). In the example given in Table 2.1, we may firstly modify (relabel) the price of the third house so that its price is not smaller than the prices of the first two houses. Analogously, we proceed with the fourth house with respect to the fifth house. The resulting data are monotone.

In the remainder of this section we introduce some notation and definitions, which are used in the discussion throughout the thesis.

Notation and definitions

Let $\mathcal{D}_\pi^N$ denote the ensemble of data sets each consisting of N points drawn from a probability distribution $\pi(\mathbf{x}, \ell)$. Here π is defined on $\mathcal{X} \times \mathcal{L}$, where $\mathcal{X}$ is a k-dimensional input space and $\mathcal{L}$ is the set of univariate labels.

A data point is denoted by $\mathbf{x} = (x_1, x_2, \ldots, x_k)$, where $x_1, x_2, \ldots, x_k$ are the values of the independent variables; the label of $\mathbf{x}$ is $\ell_\mathbf{x}$. Hence, a data set D is denoted by $D = (\mathbf{x}^n, \ell_{\mathbf{x}^n})_{n=1}^N$.

Next for each $\mathbf{x}$ we define:

$$\begin{aligned} Down(\mathbf{x}) &= \{\mathbf{x}' \in \mathcal{X} \mid \mathbf{x}' \leq \mathbf{x}\}, \\ Up(\mathbf{x}) &= \{\mathbf{x}' \in \mathcal{X} \mid \mathbf{x}' \geq \mathbf{x}\}, \\ Incomp(\mathbf{x}) &= \mathcal{X} - Down(\mathbf{x}) \cup Up(\mathbf{x}). \end{aligned}$$

Hence, all the points belonging to $Down(\mathbf{x})$ and $Up(\mathbf{x})$ are *comparable* with $\mathbf{x}$, whereas all the points belonging to $Incomp(\mathbf{x})$ are *incomparable* with $\mathbf{x}$.

Furthermore, let X, $X \subset \mathcal{X}$ be the set of values of independent variables belonging to D. Then for each $\mathbf{x} \in D$ we define:

$$\begin{aligned} Down_D(\mathbf{x}) &= \{\mathbf{x}' \in X \mid \mathbf{x}' \leq \mathbf{x}\}, \\ Up_D(\mathbf{x}) &= \{\mathbf{x}' \in X \mid \mathbf{x}' \geq \mathbf{x}\}, \\ Incomp_D(\mathbf{x}) &= X - Down_D(\mathbf{x}) \cup Up_D(\mathbf{x}). \end{aligned}$$

Definition 2.1.1. The pair $(\mathbf{x}, \mathbf{x}')$ is called non-monotone (inconsistent) if

$$\begin{aligned} &\mathbf{x} > \mathbf{x}' \text{ and } \ell_\mathbf{x} < \ell_{\mathbf{x}'} \quad \text{or} \\ &\mathbf{x} < \mathbf{x}' \text{ and } \ell_\mathbf{x} > \ell_{\mathbf{x}'} \quad \text{or} \\ &\mathbf{x} = \mathbf{x}' \text{ and } \ell_\mathbf{x} \neq \ell_{\mathbf{x}'} \end{aligned} \tag{2.1}$$

Otherwise, $(\mathbf{x}, \mathbf{x}')$ is called a monotone (consistent) pair.

Hence, it is clear that the relationship between incomparable points is always called monotone.

Definition 2.1.2. We call $\mathbf{x} \in D$ a monotone point if $\forall \mathbf{x}' \in D:$ $(\mathbf{x}, \mathbf{x}')$ is a monotone pair. This is equivalent to

$$\mathbf{x}' \in Down_D(\mathbf{x}) \Rightarrow \ell_\mathbf{x} \geq \ell_{\mathbf{x}'} \quad \text{and} \quad \mathbf{x}' \in Up_D(\mathbf{x}) \Rightarrow \ell_\mathbf{x} \leq \ell_{\mathbf{x}'}. \tag{2.2}$$

Definition 2.1.3. A data set is monotone if all pairs (points) in the data set are monotone.

Total monotonicity of a data set is defined as $\ell_{\mathbf{x}}$ being monotonically dependent on all input attributes $x_i, i = 1, 2, \ldots, k$, in the data set.

Furthermore, for the monotone problems considered in this thesis, we assume that the data set D is generated by the following process

$$\ell_{\mathbf{x}} = f(\mathbf{x}) + \epsilon, \tag{2.3}$$

where f is a monotone function and ϵ is a random error. In regression problems, ϵ has zero mean, whereas in classification problems ϵ is a small probability that the assigned class is incorrect.

The total monotonicity of f on $\mathbf{x}$ is defined on all independent variables by

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} : \quad \mathbf{x} \geq \mathbf{x}' \Rightarrow f(\mathbf{x}) \geq f(\mathbf{x}')$$

Note that even though f is monotone, the data D generated by (2.3) may not be monotone due to the random error ϵ. In other words, there are non-monotone pairs of points in the data D. In such cases we refer to D as *non-monotone* or *noisy data*.

To illustrate the notion and definitions introduced in this section, we consider the simple example depicted in Figure 2.1. The figure shows a data set $D = (\mathbf{x}^n, \ell_{\mathbf{x}^n})_{n=1}^5$ of five points that take values in a two-dimensional input space. The dependent variable is discrete and ranges from 1 to 3. We assume that there exist monotone relationships between the attributes and the dependent variable.

There are ten pairs in total and seven out of them are comparable such as $(\mathbf{x}^1, \mathbf{x}^3)$. The remaining three pairs are incomparable; for example, the pair $(\mathbf{x}^1, \mathbf{x}^2)$ because $x_1^1 > x_1^2$ but $x_2^1 < x_2^2$.

Now we focus on point $\mathbf{x}^3$. The set $Down_D(\mathbf{x}^3)$ consists of all points smaller than or equal to $\mathbf{x}^3$, i.e., $Down_D(\mathbf{x}^3) = \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3\}$. Analogously, $Up_D(\mathbf{x}^3) = \{\mathbf{x}^3, \mathbf{x}^5\}$ and $Incomp_D(\mathbf{x}^3) = \{\mathbf{x}^4\}$. Furthermore, given the labeling of the points, it is obvious that the pairs $(\mathbf{x}^2, \mathbf{x}^3)$ and $(\mathbf{x}^3, \mathbf{x}^5)$ are monotone as they meet the conditions in (2.2), whereas the pair $(\mathbf{x}^1, \mathbf{x}^3)$ is non-monotone, which is also the only inconsistency in D (as shown by the dotted line). Hence, the monotone points are only $\mathbf{x}^2, \mathbf{x}^4$ and $\mathbf{x}^5$, so the data set D is non-monotone.

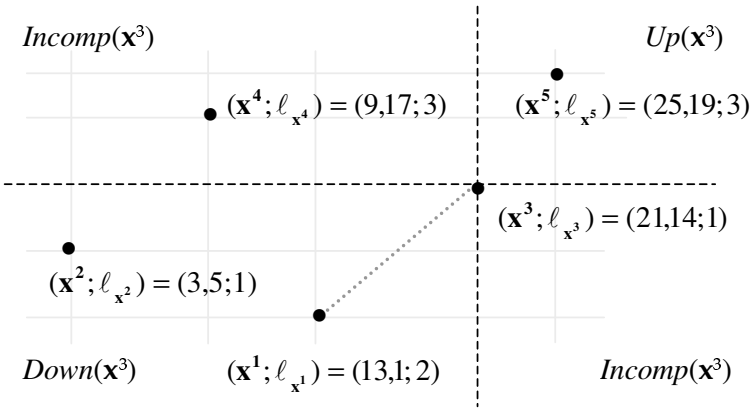


Figure 2.1: An example of a data set D with five points.

2.2 Related work

As we discussed in Section 1.3, a monotone data set D can be considered as a mapping function $\hat{f} : X \to L$, which is assumed to be close proximity of a monotone function $f : \mathcal{X} \to \mathcal{L}$.

The problem of monotonicity testing of functions has attracted a great deal of interest in the literature. Goldreich et al. (1998) consider this problem in the context of gap property testing where the goal is to determine whether a given object (function) does have a particular property or is "far" from satisfying it. The authors propose an algorithm (tester) for monotonicity testing of Boolean functions. The tester has a probabilistic nature, i.e., it determines the probability whether a function belongs to a class of monotone functions or is "far" from it.

Raskhodnikova (1999) suggests further improvement of the algorithms developed by Goldreich et al. (1998). On the one hand, she reduced the query complexity of the tester compared to previous tester bounds. On the other hand, she proposed an alternative test for Boolean functions whose query complexity is independent of the size of the domain.

Monotonicity testing of Boolean functions over general partially ordered sets is studied by Fischer et al. (2002). First, the authors show that the problem of monotonicity testing of Boolean functions (Boolean monotonicity) is equivalent to three testing problems in logic and graph theory. The first problem is testing 2-CNF (conjunctive normal form) assignments of

Boolean variables. The second problem is testing whether a set of vertices in a fixed graph is close to a vertex cover (a set of vertices in an undirected graph where every edge connects at least one vertex). Here the closeness is measured by the number of the vertices that need to be added to make the set vertex cover. The third problem is testing if a set of vertices is close to a clique (a complete subgraph of an undirected graph), where closeness is defined by the number of vertices that need to be removed to make the set a clique. Furthermore, Fischer et al. (2002) show that the problem of monotonicity on general graphs can be reduced to monotonicity on bipartite graphs.

The studies mentioned so far discuss the notion of "closeness" between objects. Considering two functions f and $\hat{f}$, their distance is measured by the number of domain elements on which they differ. Thus, the distance of function $\hat{f}$ to a property M, for example monotonicity, is given by the minimum over all monotone functions f that satisfy M. Hence, the relative distance is measured by the distance between $\hat{f}$ and M divided by the size (number of elements) of the domain. Finally, a function $\hat{f}$ is ϵ-close to monotone if $\hat{f}$ can be made monotone by changing its value on at most an ϵ-fraction of the domain.

In this context, the greedy algorithm for relabeling that will be described in Section 2.4 constructs a function $\hat{f}$ that is monotone and close to f in the sense that the fraction of label changes is small. Here ϵ is the fraction of label changes made to get monotone data of the total number of points in the data.

Potharst (1999) also presents a mapping between monotone data sets and a certain type of labeled directed graphs. The set of graph vertices corresponds to the set of objects from the input space belonging to the data set. Furthermore, each vertex is labeled with the respective class label of the object in the data. The vertices are joined by arcs (links/edges) with direction. Any path of verticies that forms a non-increasing sequence of labels is called non-increasing. He proved that a data set is monotone if and only if all the paths in the corresponding graph are non-increasing. He also shows that by using such a type of labeled directed graphs, random monotone data sets can be generated with predefined parameters: number of data points, vectors of attribute values, labels.

Hellerstein (1989) also employs a graphical representation of the monotone relationships between the input variables for diagnosing performance

problems of computer systems. Such a representation is called a diagnosis search-graph (DSG), which is a directed graph consisting of source nodes (dependent or target variables), nodes (independent or measurement variables) and arcs (variable relationships). The monotone relationships between the dependent and influencing variables is indicated by labeling the arcs of the DSG with the sign (direction) of influence–positive or negative.

By using a top-down strategy (arc traversal), a subset of the measurement variables is identified that best explains the variations in the target variable. As Hellerstein argues, this is equivalent to finding the subset of independent variables that constitutes the best model for the dependent variable–which is a typical problem in statistics. He proposes a statistical non-parametric test that evaluates the significance of the monotone relationships in a given data set. His approach is based on computing the fraction of observation-pairs that agree with the monotonicity constraint; it does not require specifying a function form a priori. His only assumption is that the dependent variable has a monotone relationship with each independent variable. In other words, the claim under the null hypothesis H_0 in the test is that monotonicity explains nothing about the variations in the target, i.e, the function is constant. In order to accept that monotonicity does play a role, it is necessary to reject the null hypothesis. For this purpose, Hellerstein determines the probability of getting at least a given number of monotone observation-pairs computed from the data at hand. Under H_0, if the probability obtained is less than the critical values of 0.01 or 0.05, then the null hypothesis is rejected in favor of the alternative hypothesis H_1.

The results from the application of the test to diagnosing computer performance problems show that the approach is superior to least-squares regression. However, as Hellerstein points out, whenever the algebraic functional form is known, regression is preferred. Furthermore, given the definitions of the hypotheses (H_0 and H_1), the test allows to check only whether the functional form is constant or monotone. Thus, the possibility of having non-monotone relationships between the variables is excluded, which limits the application of the approach as non-monotone cases also occur in practice. Finally, the test results are very sensitive to the number of comparable pairs used, i.e., too few of them might produce statistically insignificant monotone relationships, whereas too many lead to cumbersome data collection and reduction.

Daniels and Kamp (1999) propose another approach for testing mono-

tone relationships in data. It is based on computing a monotonicity index to measure the degree of monotonicity of the output of a neural network with respect to each input variable. By taking the partial derivative $\partial f/\partial x_i$ at each data point $\mathbf{x}^n$, the monotonicity index in variable x_i is formally expressed as

$$mon(x_i) = \frac{1}{N}\sum_{n=1}^{N} sign\left(\frac{\partial f}{\partial x_i}(\mathbf{x}^n)\right)$$

where N is the number of points in the data, $sign(u) = 1$ if $u > 0$ and $sign(u) = -1$ if $u \leq 0$. The value of the monotonicity index thus computed lies in the range $[-1, 1]$. A value close to 0 indicates a non-monotone relationship, whereas a value close to 1 (-1) indicates an increasing (decreasing) monotone relationship.

The proposed monotonicity index has been applied to two economic classification problems, namely bond rating and house pricing. Although the results comply with what is expected from theory, a major disadvantage of this method is that it first requires building a model whose results are highly dependent on the architecture of the neural network. In other words, constructing an inaccurate model (neural network with inappropriate structure) would lead to a wrong monotonicity index and hence an incorrect conclusion about the direction of influence of a particular variable on the output.

Apart from measures for the degree of monotonicity, there has been much research on transformations of a non-monotone data set making the resulting set monotone.

A popular technique for improving the quality of data is so-called *isotonic regression*, discussed in Robertson et al. (1988). Isotonic regression stands for a class of non-decreasing regression functions; non-increasing functions are called *antitonic*. Both types of functions (isotonic and antitonic) are generally referred to as *monotonic* functions.

The basic principle of isotonic regression partitions the input space into consecutive subsets–called *level sets* or *blocks*–where the estimated regression function is constant. In other words, the outcome is a piecewise constant function.

A common method in isotonic regression is the *Pool-Adjacent-Violators-Algorithm* (PAVA), which has been first developed by Ayer et al. (1955). The algorithm works for input data $\mathcal{X}$ with a total order given by

$$\mathbf{x}^1 \leq \mathbf{x}^2 \leq \ldots \leq \mathbf{x}^N.$$

Now given a function f on $\mathcal{X}$ and positive weight function w defined on $\mathcal{X}$, the objective of PAVA is to find an isotonic function m^* such that m^* minimizes

$$\sum_{n=1}^{N} \Big(f(\mathbf{x}^n) - m^*(\mathbf{x}^n) \Big)^2 w(\mathbf{x}^n)$$

subject to the constraint

$$m^*(\mathbf{x}^1) \leq m^*(\mathbf{x}^2) \leq \ldots \leq m^*(\mathbf{x}^N).$$

If $\mathbf{x}$ is discrete, then the weights w are simply the number of points belonging to each category of $\mathbf{x}$; if $\mathbf{x}$ is continuous, then w is usually equal to one.

The PAVA algorithm starts with checking whether there are points for which the order restriction between their function values is violated. If no violation is found, then the algorithm terminates and the isotonic solution is simply the set of original function values. Otherwise, there is a pair of points $\mathbf{x}^{n-1}$ and $\mathbf{x}^n$ such that $f(\mathbf{x}^{n-1}) > f(\mathbf{x}^n)$. Then to resolve the inconsistency, these two points are pooled by replacing them with their weighted average. The two weights $w(\mathbf{x}^{n-1})$ and $w(\mathbf{x}^n)$ are replaced by $w(\mathbf{x}^{n-1}) + w(\mathbf{x}^n)$. In the next step, the algorithm checks whether the new set of function values is isotonic. If not, the violating points are again replaced by their weighted average. This process continues until an isotonic set of values is obtained.

Robertson et al. (1988) discuss various other algorithms for isotonic regression, such as max-min, minimum lower sets (maximum upper sets), and algorithms for the matrix partial order. All these algorithms are *structure algorithms* that produce ordered-level sets until an isotonic solution is obtained. In Section 2.4.5 we demonstrate the application of a matrix-ordered approach for isotonic regression, and compare it with the algorithm for relabeling we describe in the next section.

In general, isotonic regression is a mean-squared error (MSE) technique. In this sense, Robertson et al. (1988) prove that the solution generated by the isotonic regression is unique.

The main advantage of isotonic regression is that it is a non-parametric approach, which does not require any specific assumptions about the func-

tional form–apart from monotonicity. However, a disadvantage of standard isotonic regression is that it sometimes produces a large number of level sets, which may lead to overfitting the data. This implies that some level sets may contain only one point or neighboring levels sets do not differ considerably in their estimated response function. Schell and Singh (1997) propose a solution to this problem by applying a backward elimination procedure to isotonic regression, which leads to great reduction of the number of level sets. To do so they first pool all the sets (blocks) with their predecessor (if no one exist then with their successors) that contain less than a certain percentage of the total data. Then, they pool all the blocks whose estimated response values do not differ significantly according to a Fischer test. This procedure leads to a smaller number of level sets and hence, more parsimonious models.

This section shows that there has been much research in the last 50 years, dealing with testing monotonicity of data or applying methods to make data monotone. The following sections in this chapter describe our contributions to this field. Our motivation for considering new methods, which are alternatives to the approaches proposed so far in the literature, is based on several observations.

Given the problem of testing monotonicity of a real data set, we find that earlier approaches have certain limitations: (i) some methods are developed only for particular type of functions; for example, Boolean functions; (ii) the method proposed by Hellerstein (1989) is based on the assumption that the data generating function is monotone, which may not be the case for any data; this may lead to false conclusions about the existence of monotone relationships; (iii) the approaches, which use a graph to map the relationships in a monotone data set, require an additional step for data representation; this is not only time consuming but also may not be feasible for large data sets with many attributes; (iv) other methods need first to model the data, and then on the basis of the model, to check the presence of monotone dependencies in the data; the drawbacks of such procedures are the additional time and efforts devoted to pre-modeling the data, and the strong dependence of the results on the model outcome.

To overcome the limitations of these approaches, in Section 2.3 we propose a novel procedure to test monotonicity of a real data set. The advantages of our procedure are:

- It is intuitive, straightforward and can be applied to any data set, irrespective of the number or type of variables.

- It works for both continuous and discrete labels.
- It uses to large extent the information given in the real data.
- It is not constrained by the form or type (monotone/non-monotone) of the function assumed to generate the real data.
- It does not require any pre-modeling data procedure.

Related to the problem of making data monotone, the previous approaches we discussed earlier have the following limitations: (i) many methods are developed to deal with regression problems only; (ii) the application and the efficiency of some methods depends on the dimensionality of the data; for example, some approaches work for one-dimensional or low-dimensional input data only; (iii) the data transformation in most of the methods is based on the construction of a model, which requires setting various parameters, or optimizing a certain function; this often may be a cumbersome and a time-consuming procedure.

These drawbacks are avoided in the greedy algorithm for relabeling we propose in Section 2.4. The main advantages of our algorithm are:

- It is straightforward and can be applied to low- and high-dimensional data.
- It works for regression and classification problems.
- It uses only the information given in the real data, without the need to model the data beforehand.

2.3 Testing monotonicity of a data set

2.3.1 Benchmark measures for monotonicity of a data set

Let $D \in \mathcal{D}_{\pi}^{N}$ be a data set of N points drawn from a probability distribution $\pi(\mathbf{x}, \ell)$. We assume that the set of independent variables is drawn from a probability distribution $\pi_1(\mathbf{x})$ (e.g., normal, uniform). Furthermore, $\pi_2(\ell)$ denotes the probability distribution from which the labels are drawn; π_2 has

a density ρ if $\mathcal{L}$ is continuous, or is a discrete probability measure represented by $\sigma_1, \sigma_2, \ldots, \sigma_{\ell_{max}}$, with $\sum_{i=1}^{\ell_{max}} \sigma_i = 1$ if $\mathcal{L}$ is discrete.

Then, we define $\mathcal{D}_{\mathcal{B}}$ as the collection of all data sets generated with the same structure of independent variables as D and a set of labels that is a randomly generated permutation of the labels in D. The definition of $\mathcal{D}_{\mathcal{B}}$ implies independence between the explanatory variables and the labels. We call $\mathcal{D}_{\mathcal{B}}$ a *benchmark class of data*.

We now compute two measures for the degree of monotonicity of a benchmark data set $D_{\mathcal{B}} \in \mathcal{D}_{\mathcal{B}}$:

- The expected value of the fraction of monotone pairs among the number of comparable pairs.
- The expected value of the number of monotone points.

We call these *benchmark measures*.

Lemma 2.3.1. *For a data set $D_{\mathcal{B}} \in \mathcal{D}_{\mathcal{B}}$, the expected value of the fraction of monotone pairs is:*

$$\mathcal{E}[fr_M] = 1 - \frac{1}{2}\big(1 + fr_{\mathbf{x}=\mathbf{x}'}\big) fr_{\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'}}$$

where $fr_{\mathbf{x}=\mathbf{x}'}$ is the fraction of pairs with identical points among the comparable pairs in $D_{\mathcal{B}}$, and $fr_{\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'}}$ is the fraction of pairs with different labels among the total number of pairs in $D_{\mathcal{B}}$.

Proof. Let $(\mathbf{x}, \mathbf{x}')$ be a comparable pair of points $\mathbf{x}, \mathbf{x}' \in D_{\mathcal{B}}$ to which labels from the set of the original labels in D are assigned randomly. This can result in either a monotone pair corresponding to success or a non-monotone pair corresponding to failure. In other words, this can be considered as a Bernoulli trial (random variable) with two outcomes with probability $\Pr(M)$ of obtaining a monotone pair and $1 - \Pr(M)$, otherwise. Furthermore, let A denote a Bernoulli random variable indicating whether or not a pair is monotone, that is, $A = 1$ if and only if a pair is monotone, otherwise $A = 0$. Then,

$$\mathcal{E}[A] = 1 \cdot \Pr(M) + 0 \cdot \big(1 - \Pr(M)\big) = \Pr(M).$$

Now if we consider a random permutation of the class labels, we have a total of N_{CP} (dependent) Bernoulli trials, one for each comparable pair. Let

N_M denote the number of successes (monotone pairs) among the number of comparable pairs. Then,

$$\mathcal{E}[N_M] = \mathcal{E}\left[\sum_{i=1}^{N_{CP}} A_i\right] = \sum_{i=1}^{N_{CP}} \mathcal{E}[A_i] = N_{CP} \Pr(M).$$

Hence, for the fraction of monotone pairs fr_M computed as N_M/N_{CP} we get

$$\mathcal{E}[fr_M] = \frac{N_{CP} \Pr(M)}{N_{CP}} = \Pr(M).$$

Now we compute the probability $\Pr(M)$ of a comparable pair being monotone. First note that $\Pr(M) = 1 - \Pr(NM)$ where $\Pr(NM)$ denotes the probability of a comparable pair being non-monotone. Thus computing the latter will suffice for our objective.

For every comparable pair the outcome of the random assignment of labels to the points can be a pair with the same labels or a pair with different labels. According to the conditions in Definition 2.1.1 a non-monotone pair can be obtained only for points with different labels. Furthermore, the random assignment of different labels to a comparable pair of non-identical points has simply a 50% chance of obtaining a non-monotone pair, whereas for a comparable pair of identical points this chance is 100%. Hence,

$$\begin{aligned}\Pr(NM) &= \frac{1}{2}\left(1 - fr_{\mathbf{x}=\mathbf{x}'}\right) \Pr(\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'}) + fr_{\mathbf{x}=\mathbf{x}'} \Pr(\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'}) \\ &= \frac{1}{2}\left(1 + fr_{\mathbf{x}=\mathbf{x}'}\right) \Pr(\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'}),\end{aligned}$$

where $\Pr(\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'})$ is the probability that a pair $(\mathbf{x}, \mathbf{x}')$ has different labels in $D_{\mathcal{B}}$. To compute $\Pr(\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'})$, we proceed as follows.

By the definition of benchmark data, the labels in $D_{\mathcal{B}}$ are a random permutation of the labels in the original data. Since the latter may contain identical points, we expect that a label may occur more than once in $D_{\mathcal{B}}$, irrespective of the label type (discrete or continuous). Therefore, to compute the probability that a pair of points has different labels, we simply need to compute the fraction of pairs with different labels among the total number of pairs in the data. Thus we obtain

$$\Pr(\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'}) = fr_{\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'}},$$

and

$$\Pr(M) = 1 - \frac{1}{2}\big(1 + fr_{\mathbf{x}=\mathbf{x}'}\big) fr_{\ell_{\mathbf{x}} \neq \ell_{\mathbf{x}'}}.$$

□

We now compute the expected value of the number of points that are monotone in an arbitrary benchmark data set $D_{\mathcal{B}} \in \mathcal{D}_{\mathcal{B}}$. We do so by applying the Monte Carlo technique, which is a popular approach of statistical sampling employed to approximate solutions to quantitative problems. The classic reference on Monte Carlo methods is Hammersley and Handscomb (1964) and a recent reference is Kleijnen (2004).

We generate a finite collection $\{D_{\mathcal{B}}\}^S$ of S benchmark data sets $D_{\mathcal{B}}$; for each of them we compute the number of monotone points, N_{Mpt}. Then, over the whole collection, the mean $\mathcal{E}[N_{Mpt}]$ is estimated by

$$\mathcal{E}[N_{Mpt}] = \frac{1}{S} \sum_{s \in \{D_{\mathcal{B}}\}^S} N^s_{Mpt}$$

as a final benchmark indicator for the number of monotone points in a data set.

2.3.2 Statistical test of the difference between the observed and benchmark monotonicity measures

Given a real data set D, we can simply compute two natural measures for the degree of monotonicity of the data:

- The fraction of monotone pairs among comparable pairs, fr_M.
- The number of monotone points, N_{Mpt}.

Having computed these measures, we would like to compare them with the respective measures obtained from the class of benchmark data sets $\mathcal{D}_{\mathcal{B}}$ defined in the previous section. For this purpose, we design a statistical test to check whether or not the difference between the actual indicators and benchmarks is significant.

We consider the following null hypothesis:

H_0: the data are not generated by a monotone process

against the alternative

H_1: the data are generated by a monotone process.

So, the null hypothesis states that in the real data the labels are not ordered with respect to the points, whereas the alternative hypothesis implies such an ordering.

To perform the test, we need to determine the distribution of a statistic presuming that H_0 is correct. In our case we use two statistics corresponding to the benchmark measures for monotonicity (fraction of monotone pairs and number of monotone points). Since we do not know the theoretical distribution of the measures, we cannot perform standard statistical tests. Alternatively, we simply generate the empirical distributions of both statistics. To do so we first generate a finite collection of benchmark data sets, as defined in the previous section. Then for each of the data sets we compute the values of the corresponding benchmark measures. From the distributions obtained we find the critical values of the statistics at 95% and 99% significance levels. Here we consider right-hand tail tests since the null hypothesis can be rejected only if the observed measures are significantly higher than the benchmarks. Finally, we compare the critical values with the observed values of the indicators computed from the real data. If the former are smaller than the latter then we reject the null hypothesis as the observed values fall in the right tail area of the distribution; otherwise, we do not reject the null hypothesis.

Remark. There are two pathological cases where the degree of monotonicity of the original data is known a priori: (i) if all the points and labels are identical, then the data are totally monotone; (ii) if all the points are identical and all the labels are unique, then the data are totally non-monotone. In both cases, the original data do not provide useful information to build models for prediction tasks. In practice, however, we expect that data sets mostly contain unique points, like in the case studies presented in Section 2.5. Then, the proposed benchmark measures and the statistical test are appropriate tools to check the degree of monotonicity of the data.

2.4 Greedy algorithm for relabeling

The objective of the greedy algorithm is to transform non-monotone into monotone data. We make two assumptions about the original (non-monotone)

data:

- The data are presumably monotone meaning that there are only a small number of non-monotone (inconsistent) pairs of points; this can be checked by using, for example, the testing procedure described in the previous section.

- The violation of the monotonicity assumption in the data is caused by noise in the labels, i.e., the labels of the points participating in non-monotone relationships are incorrect with a small fixed probability.

Given these assumptions, the monotone transformation in the greedy algorithm is obtained by changing the values of the dependent variable (label) of the points participating in non-monotone relationships; so-called *relabeling*. Hence, our ultimate goal is to make the data monotone while we try to preserve the original data by making as few label changes as possible. The idea is to reduce the number of non-monotone pairs by relabeling only one data point in each step. To do this, we choose a data point such that the increase in correctly labeled points is maximal (this is not necessarily the point which is involved in the maximal number of non-monotone pairs). The process is continued, until the data set is monotone (see Definition 2.1.3).

2.4.1 Notation and description

Let $D = \left(\mathbf{x}^n, \ell_{\mathbf{x}^n}\right)_{n=1}^N$ denote the original data set, and $Q(D)$ denote the set of all non-monotone points in D. For each data point $\mathbf{x} \in Q(D)$ and $\ell \in D$, we define $A_\ell(\mathbf{x}) \subset Down_D(\mathbf{x})$ and $B_\ell(\mathbf{x}) \subset Up_D(\mathbf{x})$ by

$$A_\ell(\mathbf{x}) = \{\mathbf{x}' \in Down_D(\mathbf{x}) \mid \mathbf{x}' \neq \mathbf{x} \text{ and } \ell_{\mathbf{x}'} = \ell\}$$
$$B_\ell(\mathbf{x}) = \{\mathbf{x}' \in Up_D(\mathbf{x}) \mid \mathbf{x}' \neq \mathbf{x} \text{ and } \ell_{\mathbf{x}'} = \ell\}\,.$$

Note that point $\mathbf{x}$ belongs neither to $\bigcup_{\ell \in D} A_\ell(\mathbf{x})$ nor to $\bigcup_{\ell \in D} B_\ell(\mathbf{x})$. For example in Figure 2.1, we have $A_1(\mathbf{x}^3) = \{\mathbf{x}^2\}$, $A_2(\mathbf{x}^3) = \{\mathbf{x}^1\}$ and $B_3(\mathbf{x}^3) = \{\mathbf{x}^5\}$.

Let $a_\ell^{\mathbf{x}}$ and $b_\ell^{\mathbf{x}}$ denote the number of points in $A_\ell(\mathbf{x})$ and $B_\ell(\mathbf{x})$, respectively and $c_{\mathbf{x}}$ denotes the number of points in $Down_D(\mathbf{x}) \cup Up_D(\mathbf{x})$, i.e., this is the number of all points comparable to $\mathbf{x}$.

Furthermore, we define

ℓ_{min}, ℓ_{max} – the minimum and maximum of the labels in D,
$\ell_{maxDn(\mathbf{x})}$ – the maximum of the labels in $Down_D(\mathbf{x})$,
$\ell_{minUp(\mathbf{x})}$ – the minimum of the labels in $Up_D(\mathbf{x})$,
$N^{\mathbf{x}}_{\ell_{\mathbf{x}}}$ – total number of points correctly labeled with respect to $\mathbf{x}$ for the current label of $\mathbf{x}$, $\ell_{\mathbf{x}}$, i.e.,

$$N^{\mathbf{x}}_{\ell_{\mathbf{x}}} = a^{\mathbf{x}}_{\ell_{min}} + \ldots + a^{\mathbf{x}}_{\ell_{\mathbf{x}}} + b^{\mathbf{x}}_{\ell_{\mathbf{x}}} + \ldots + b^{\mathbf{x}}_{\ell_{max}}.$$

Note that if there exist points $\mathbf{x}' \in D$ such that $\mathbf{x} = \mathbf{x}'$ we can easily modify $N^{\mathbf{x}}_{\ell_{\mathbf{x}}}$ by adding the number of points $\mathbf{x}'$ for which $\ell_{\mathbf{x}} = \ell_{\mathbf{x}'}$. To simplify the notations, however, in the remainder of Section 2.4 we assume that all data points in the data set D are unique, i.e., no points are identical.

For each data point $\mathbf{x} \in Q(D)$ we compute the maximal increase, $I^{\mathbf{x}}_{max}$, in the number of correctly labeled points with respect to $\mathbf{x}$ if the label of $\mathbf{x}$ is changed into ℓ' where $\ell' \in D$. If there is more than one label with the same maximal increase in correctly labeled points, then we choose the label closest to the current label of $\mathbf{x}$. Finally, we select a point $\mathbf{x} \in Q(D)$ for which $I^{\mathbf{x}}_{max}$ is the largest, and change its label. This process is repeated until the data set is monotone. The algorithm outline is given in Algorithm 2.1.

Algorithm 2.1 Transformation of non-monotone into monotone data

Initialization: Compute $Q(D)$ on the basis of D
while $Q(D) \neq \emptyset$ **do**
  for all $\mathbf{x} \in Q(D)$ **do**
    $I^{\mathbf{x}}_{max} = \max\{N_\ell - N_{\ell_{\mathbf{x}}} | \ell \in D\}$
    Λ = set of labels ℓ for which $N_\ell - N_{\ell_{\mathbf{x}}}$ is maximal
    Form a triple $(\mathbf{x}, I^{\mathbf{x}}_{max}, \ell')$ where $\ell' \in \Lambda$ is the closest label to $\ell_{\mathbf{x}}$
    (in Lemma 2.4.2 it is shown that ℓ' is unique)
  end for
  From all triples, choose the one where $I^{\mathbf{x}}_{max}$ is maximal and change the label of the point into ℓ'
  Update $Q(D)$ on the basis of the modified data set D
end while

In general, the points correctly labeled with respect to $\mathbf{x}$ for its current label, $\ell_{\mathbf{x}}$, are both all points incomparable to $\mathbf{x}$ and all the points in $A_{\ell_{min}} \cup \ldots \cup A_{\ell_{\mathbf{x}}}$ and $B_{\ell_{\mathbf{x}}} \cup \ldots \cup B_{\ell_{max}}$. Since the number of the points incomparable to

$\mathbf{x}$ is constant and these points do not contribute to $I^{\mathbf{x}}_{max}$, we may completely ignore them.

The correctness of the algorithm follows from Lemmas 2.4.1 and 2.4.2. Lemma 2.4.1 states that it is always possible to reduce the number of non-monotone pairs by changing the label of only one point, as long as the data set is non-monotone. In Lemma 2.4.2, it is shown that there is a canonical choice for the new label for which a maximal reduction can be obtained. There may be more than one label for which this can be achieved, but these are all smaller or all larger than the current label of the point–so the closest one is chosen, which is unique.

Lemma 2.4.1. *Let D^i denote the data set D after i iterations. If $Q(D) \neq \emptyset$, then there is at least one point $\mathbf{x} \in Q(D^i)$ that can be relabeled such that the number of non-monotone pairs is reduced by at least one.*

Proof. Since $Q(D^i)$ is a non-empty partially ordered set, there is a maximal point $\mathbf{x}$ with label $\ell_{\mathbf{x}}$. Because $\mathbf{x}$ participates in at least one non-monotone pair, there is another point $\mathbf{x}' \in Q(D^i)$ with label $\ell_{\mathbf{x}'}$ such that $\mathbf{x} > \mathbf{x}'$ and $\ell_{\mathbf{x}} < \ell_{\mathbf{x}'}$. If we relabel $\mathbf{x}$ with $\ell_{\mathbf{x}'}$, then the increase in the number of correctly labeled points with respect to $\mathbf{x}$, $I^{\mathbf{x}}_{max}$ will be

$$\begin{aligned} I^{\mathbf{x}}_{max} = N^{\mathbf{x}}_{\ell_{\mathbf{x}'}} - N^{\mathbf{x}}_{\ell_{\mathbf{x}}} = &\sum_{\ell \in [\ell_{min}, \ell_{\mathbf{x}}]} a^{\mathbf{x}}_{\ell} + \sum_{\ell \in (\ell_{\mathbf{x}}, \ell_{\mathbf{x}'}]} a^{\mathbf{x}}_{\ell} + \sum_{\ell \in [\ell_{\mathbf{x}'}, \ell_{max}]} b^{\mathbf{x}}_{\ell} \\ &- \sum_{\ell \in [\ell_{min}, \ell_{\mathbf{x}}]} a^{\mathbf{x}}_{\ell} - \sum_{\ell \in [\ell_{\mathbf{x}}, \ell_{\mathbf{x}'})} b^{\mathbf{x}}_{\ell} - \sum_{\ell \in [\ell_{\mathbf{x}'}, \ell_{max}]} b^{\mathbf{x}}_{\ell} \\ = &\sum_{\ell \in (\ell_{\mathbf{x}}, \ell_{\mathbf{x}'}]} a^{\mathbf{x}}_{\ell}, \end{aligned}$$

because $\sum_{\ell \in [\ell_{\mathbf{x}}, \ell_{\mathbf{x}'})} b^{\mathbf{x}}_{\ell} = 0$ ($\mathbf{x}$ is maximal in $Q(D^i)$ and the points larger than $\mathbf{x}$ have label $\ell \geq \ell_{\mathbf{x}'}$).

However,

$$I^{\mathbf{x}}_{max} = \sum_{\ell \in (\ell_{\mathbf{x}}, \ell_{\mathbf{x}'}]} a^{\mathbf{x}}_{\ell} \geq 1,$$

as there is at least one point that is smaller than $\mathbf{x}$ and has label $\ell_{\mathbf{x}'}$, that is $\mathbf{x}'$. Therefore, by relabeling $\mathbf{x}$ with $\ell_{\mathbf{x}'}$, the number of non-monotone pairs is reduced by at least one. □

Lemma 2.4.2. *Suppose that the maximal increase in the number of correctly labeled points with respect to* $\mathbf{x}$, $I^{\mathbf{x}}_{max}$, *can be obtained by at least two labels* r *and* s, $r < s$. *Then*

$$r < s < \ell_{\mathbf{x}} \quad \text{or} \quad \ell_{\mathbf{x}} < r < s,$$

where $\ell_{\mathbf{x}}$ *is the label of* $\mathbf{x}$.

Proof. In order to prove this lemma, we assume that $r < \ell_{\mathbf{x}} < s$. Next we show that this leads to a contradiction.

First we choose labels p and q closest to $\ell_{\mathbf{x}}$ such that $r \leq p < \ell_{\mathbf{x}} < q \leq s$ and the maximal increase for p and q is $I^{\mathbf{x}}_{max}$. Then

$$N^{\mathbf{x}}_p = N^{\mathbf{x}}_q > N^{\mathbf{x}}_\ell \quad \forall \ell,\, \ell \in (p, q). \tag{2.4}$$

For $\ell_{\mathbf{x}}$ (the current label of $\mathbf{x}$) the number of correctly labeled points is

$$N^{\mathbf{x}}_{\ell_{\mathbf{x}}} = \sum_{\ell \in [\ell_{min}, \ell_{\mathbf{x}}]} a^{\mathbf{x}}_\ell + \sum_{\ell \in [\ell_{\mathbf{x}}, \ell_{max}]} b^{\mathbf{x}}_\ell$$

and if $\mathbf{x}$ is labeled with q, it is

$$N^{\mathbf{x}}_q = \sum_{\ell \in [\ell_{min}, q]} a^{\mathbf{x}}_\ell + \sum_{\ell \in [q, \ell_{max}]} b^{\mathbf{x}}_\ell.$$

Therefore, the maximal increase for $\mathbf{x}$ is

$$I^{\mathbf{x}}_{max} = N^{\mathbf{x}}_q - N^{\mathbf{x}}_{\ell_{\mathbf{x}}} = \sum_{\ell \in (\ell_{\mathbf{x}}, q]} a^{\mathbf{x}}_\ell - \sum_{\ell \in [\ell_{\mathbf{x}}, q)} b^{\mathbf{x}}_\ell.$$

We now define the set $\mathcal{B}(\mathbf{x}) = \bigcup_{\ell \in D} B_\ell(\mathbf{x})$, and show that $\mathcal{B}(\mathbf{x})$ is non-empty.

According to (2.4), $N^{\mathbf{x}}_p > N^{\mathbf{x}}_{p'}$, where p' is the label following immediately p in the list sorted in ascending order of all labels in D, i.e.,

$$\sum_{\ell \in [\ell_{min}, p]} a^{\mathbf{x}}_\ell + b^{\mathbf{x}}_p + \sum_{\ell \in [p', \ell_{max}]} b^{\mathbf{x}}_\ell > \sum_{\ell \in [\ell_{min}, p]} a^{\mathbf{x}}_\ell + a^{\mathbf{x}}_{p'} + \sum_{\ell \in [p', \ell_{max}]} b^{\mathbf{x}}_\ell \qquad \Rightarrow$$

$$b^{\mathbf{x}}_p > a^{\mathbf{x}}_{p'}. \tag{2.5}$$

From (2.5) it follows that $b^{\mathbf{x}}_p \geq 1$ and therefore, $B_p(\mathbf{x})$ and $\mathcal{B}(\mathbf{x})$ are non-empty sets. Moreover, because we consider the case where $p < \ell_{\mathbf{x}} < q$, it is

impossible to have $p' = q$. Otherwise, $\ell_{\mathbf{x}}$ should be equal either to p or to q; then $I^{\mathbf{x}}_{max} = 0$, contradicting the fact that $I^{\mathbf{x}}_{max}$ is maximal.

We now choose a maximal point $\mathbf{x}' \in \mathcal{B}(\mathbf{x})$ with current label $\ell_{\mathbf{x}'}$ such that $p \leq \ell_{\mathbf{x}'} < \ell_{\mathbf{x}}$. The number of correctly labeled points with respect to $\mathbf{x}'$ is

$$N^{\mathbf{x}'}_{\ell_{\mathbf{x}'}} = \sum_{\ell \in [\ell_{min}, \ell_{\mathbf{x}'}]} a^{\mathbf{x}'}_{\ell} + \sum_{\ell \in [\ell_{\mathbf{x}'}, \ell_{\mathbf{x}})} b^{\mathbf{x}'}_{\ell} + \sum_{\ell \in [\ell_{\mathbf{x}}, \ell_{max}]} b^{\mathbf{x}'}_{\ell},$$

and if $\mathbf{x}'$ is labeled with q, it is

$$N^{\mathbf{x}'}_{q} = \sum_{\ell \in [\ell_{min}, q]} a^{\mathbf{x}'}_{\ell} + \sum_{\ell \in [q, \ell_{max}]} b^{\mathbf{x}'}_{\ell}.$$

Therefore, the increase in correctly labeled points with respect to $\mathbf{x}'$ is

$$I^{\mathbf{x}'} = N^{\mathbf{x}'}_{q} - N^{\mathbf{x}'}_{\ell_{\mathbf{x}'}} = \sum_{\ell \in (\ell_{\mathbf{x}'}, q]} a^{\mathbf{x}'}_{\ell} - \sum_{\ell \in [\ell_{\mathbf{x}'}, \ell_{\mathbf{x}})} b^{\mathbf{x}'}_{\ell} - \sum_{\ell \in [\ell_{\mathbf{x}}, q)} b^{\mathbf{x}'}_{\ell}. \tag{2.6}$$

We now show that $I^{\mathbf{x}'} > I^{\mathbf{x}}_{max}$. Since $\ell_{\mathbf{x}'} < \ell_{\mathbf{x}}$, the first summation in (2.6) can be rewritten as

$$\sum_{\ell \in (\ell_{\mathbf{x}'}, q]} a^{\mathbf{x}'}_{\ell} = \sum_{\ell \in (\ell_{\mathbf{x}'}, \ell_{\mathbf{x}}]} a^{\mathbf{x}'}_{\ell} + \sum_{\ell \in (\ell_{\mathbf{x}}, q]} a^{\mathbf{x}'}_{\ell}. \tag{2.7}$$

Considering both terms in (2.7), we have

$$\sum_{\ell \in (\ell_{\mathbf{x}'}, \ell_{\mathbf{x}}]} a^{\mathbf{x}'}_{\ell} \geq a^{\mathbf{x}'}_{\ell_{\mathbf{x}}} \geq 1, \quad \text{because } \mathbf{x} \in A_{\ell_{\mathbf{x}}}(\mathbf{x}'),$$

and

$$\sum_{\ell \in (\ell_{\mathbf{x}}, q]} a^{\mathbf{x}'}_{\ell} \geq \sum_{\ell \in (\ell_{\mathbf{x}}, q]} a^{\mathbf{x}}_{\ell}, \quad \text{because } \bigcup_{\ell \in (\ell_{\mathbf{x}}, q]} A_{\ell}(\mathbf{x}) \subset \bigcup_{\ell \in (\ell_{\mathbf{x}}, q]} A_{\ell}(\mathbf{x}').$$

Moreover, the choice of $\mathbf{x}'$ implies that there is no data point that is larger than $\mathbf{x}'$ and has label $\ell \in [\ell_{\mathbf{x}'}, \ell_{\mathbf{x}})$, i.e.,

$$\sum_{\ell \in [\ell_{\mathbf{x}'}, \ell_{\mathbf{x}})} b^{\mathbf{x}'}_{\ell} = 0. \tag{2.8}$$

Finally, since

$$\bigcup_{\ell\in[\ell_{\mathbf{x}},q)} B_\ell(\mathbf{x}') \subset \bigcup_{\ell\in[\ell_{\mathbf{x}},q)} B_\ell(\mathbf{x})$$

for the third term in (2.6), we have

$$\sum_{\ell\in[\ell_{\mathbf{x}},q)} b_\ell^{\mathbf{x}'} \leq \sum_{\ell\in[\ell_{\mathbf{x}},q)} b_\ell^{\mathbf{x}}. \tag{2.9}$$

According to (2.7),(2.8), and (2.9)

$$I^{\mathbf{x}'} = N_q^{\mathbf{x}'} - N_{\ell_{\mathbf{x}'}}^{\mathbf{x}'} \geq \sum_{\ell\in(\ell_{\mathbf{x}},q]} a_\ell^{\mathbf{x}} - \sum_{\ell\in[\ell_{\mathbf{x}},q)} b_\ell^{\mathbf{x}} + 1 = I_{max}^{\mathbf{x}} + 1.$$

Hence, $I^{\mathbf{x}'} > I_{max}^{\mathbf{x}}$ contradicting the fact that $I_{max}^{\mathbf{x}}$ is maximal. □

2.4.2 Efficiency

In this section we discuss several issues concerning the efficiency of the greedy algorithm for relabeling.

Number of label checks

It is possible to reduce the number of label checks in the relabeling process for each point $\mathbf{x} \in Q(D)$. First note that $\mathbf{x}$ is a non-monotone point if and only if $\ell_{minUp(\mathbf{x})} < \ell_{maxDn(\mathbf{x})}$. Therefore, the new labels ℓ_{new} that need to be considered as candidates for relabeling $\mathbf{x}$ fall within the range $[\ell_{minUp(\mathbf{x})}, \ell_{maxDn(\mathbf{x})}]$.

To show this, we first consider all labels $\ell_{new} \in D$ with $\ell_{new} < \ell_{minUp(\mathbf{x})} \leq \ell_{\mathbf{x}}$. Then the change in the number of correctly labeled points with respect to $\mathbf{x}$ if we relabel $\mathbf{x}$ with ℓ_{new} is

$$I_{\ell_{new}}^{\mathbf{x}} = N_{\ell_{new}}^{\mathbf{x}} - N_{\ell_{\mathbf{x}}}^{\mathbf{x}},$$

where

$$\begin{aligned} N^{\mathbf{x}}_{\ell_{new}} &= \sum_{\ell\in[\ell_{min},\ell_{new}]} a^{\mathbf{x}}_{\ell} + \sum_{\ell\in[\ell_{new},\ell_{\mathbf{x}})} b^{\mathbf{x}}_{\ell} + \sum_{\ell\in[\ell_{\mathbf{x}},\ell_{max}]} b^{\mathbf{x}}_{\ell} \\ N^{\mathbf{x}}_{\ell_{\mathbf{x}}} &= \sum_{\ell\in[\ell_{min},\ell_{new}]} a^{\mathbf{x}}_{\ell} + \sum_{\ell\in(\ell_{new},\ell_{\mathbf{x}}]} a^{\mathbf{x}}_{\ell} + \sum_{\ell\in[\ell_{\mathbf{x}},\ell_{max}]} b^{\mathbf{x}}_{\ell}. \end{aligned}$$

Hence,

$$I^{\mathbf{x}}_{\ell_{new}} = \sum_{\ell\in[\ell_{new},\ell_{\mathbf{x}})} b^{\mathbf{x}}_{\ell} - \sum_{\ell\in(\ell_{new},\ell_{\mathbf{x}}]} a^{\mathbf{x}}_{\ell}.$$

Next, we compute the change in the number of correctly labeled points with respect to $\mathbf{x}$ if we relabel $\mathbf{x}$ with $\ell_{minUp(\mathbf{x})}$,

$$I^{\mathbf{x}}_{\ell_{minUp(\mathbf{x})}} = N^{\mathbf{x}}_{\ell_{minUp(\mathbf{x})}} - N^{\mathbf{x}}_{\ell_{\mathbf{x}}},$$

where

$$N^{\mathbf{x}}_{\ell_{minUp(\mathbf{x})}} = \sum_{\ell\in[\ell_{min},\ell_{minUp(\mathbf{x})}]} a^{\mathbf{x}}_{\ell} + \sum_{\ell\in[\ell_{minUp(\mathbf{x})},\ell_{\mathbf{x}})} b^{\mathbf{x}}_{\ell} + \sum_{\ell\in[\ell_{\mathbf{x}},\ell_{max}]} b^{\mathbf{x}}_{\ell}.$$

Hence,

$$I^{\mathbf{x}}_{\ell_{minUp(\mathbf{x})}} = \sum_{\ell\in[\ell_{minUp(\mathbf{x})},\ell_{\mathbf{x}})} b^{\mathbf{x}}_{\ell} - \sum_{\ell\in(\ell_{minUp(\mathbf{x})},\ell_{\mathbf{x}}]} a^{\mathbf{x}}_{\ell}.$$

However, we have $\ell_{new} < \ell_{minUp(\mathbf{x})} \leq \ell_{\mathbf{x}}$ meaning that there is no point larger than $\mathbf{x}$ with a label smaller than $\ell_{minUp(\mathbf{x})}$, i.e.,

$$\sum_{\ell\in[\ell_{new},\ell_{minUp(\mathbf{x})})} b^{\mathbf{x}}_{\ell} = 0.$$

Hence,

$$\sum_{\ell\in[\ell_{new},\ell_{\mathbf{x}})} b^{\mathbf{x}}_{\ell} = \sum_{\ell\in[\ell_{new},\ell_{minUp(\mathbf{x})})} b^{\mathbf{x}}_{\ell} + \sum_{\ell\in[\ell_{minUp(\mathbf{x})},\ell_{\mathbf{x}})} b^{\mathbf{x}}_{\ell} = \sum_{\ell\in[\ell_{minUp(\mathbf{x})},\ell_{\mathbf{x}})} b^{\mathbf{x}}_{\ell}. \quad (2.10)$$

Furthermore, we have

$$\sum_{\ell\in(\ell_{new},\ell_{\mathbf{x}}]} a^{\mathbf{x}}_{\ell} = \sum_{\ell\in(\ell_{new},\ell_{minUp(\mathbf{x})}]} a^{\mathbf{x}}_{\ell} + \sum_{\ell\in(\ell_{minUp(\mathbf{x})},\ell_{\mathbf{x}}]} a^{\mathbf{x}}_{\ell}. \quad (2.11)$$

According to (2.10) and (2.11)

$$\begin{aligned} I^{\mathbf{x}}_{\ell_{new}} &= \sum_{\ell \in [\ell_{minUp(\mathbf{x})}, \ell_{\mathbf{x}})} b^{\mathbf{x}}_{\ell} - \sum_{\ell \in (\ell_{new}, \ell_{minUp(\mathbf{x})}]} a^{\mathbf{x}}_{\ell} - \sum_{\ell \in (\ell_{minUp(\mathbf{x})}, \ell_{\mathbf{x}}]} a^{\mathbf{x}}_{\ell} \\ &= I^{\mathbf{x}}_{\ell_{minUp(\mathbf{x})}} - \sum_{\ell \in (\ell_{new}, \ell_{minUp(\mathbf{x})}]} a^{\mathbf{x}}_{\ell}. \end{aligned}$$

Hence, $I^{\mathbf{x}}_{\ell_{new}} \leq I^{\mathbf{x}}_{\ell_{minUp(\mathbf{x})}}$.

Analogously, it can be shown that $I^{\mathbf{x}}_{\ell_{new}} \leq I^{\mathbf{x}}_{\ell_{maxDn(\mathbf{x})}}$ if we relabel $\mathbf{x}$ with $\ell_{new} > \ell_{maxDn(\mathbf{x})} \geq \ell_{\mathbf{x}}$. Therefore, we can consider only the labels $\ell_{new} \in [\ell_{minUp(\mathbf{x})}, \ell_{maxDn(\mathbf{x})}]$ as candidates for relabeling $\mathbf{x}$.

Number of candidate points for relabeling

Our algorithm can be further improved by reducing the number of candidate points considered for relabeling. For this purpose, we first compute the number of all points comparable to each point $\mathbf{x} \in Q(D)$, $c_{\mathbf{x}}$, and sort $Q(D)$ in descending order by $c_{\mathbf{x}}$. Then, starting with the first point $\mathbf{x} \in Q(D)$, we compute the maximal increase in correctly labeled points with respect to $\mathbf{x}$, $I^{\mathbf{x}}_{max}$. Now, all points $\mathbf{x}' \in Q(D)$ with $c_{\mathbf{x}'} < I^{\mathbf{x}}_{max}$ can be skipped in the next step, because $I^{\mathbf{x}'}_{max} < I^{\mathbf{x}}_{max}$.

Choice of a point with a maximal increase in the number of correctly labeled points

In general, there is no straightforward way to directly find the point with a maximal increase in the number of correctly labeled points. All labels in the range defined in Section 2.4.2 must be considered for relabeling, because the dependence of the change in correctly labeled points on the label can be arbitrary. This is illustrated in the following example.

Let $D = \left(\mathbf{x}^n, \ell_{\mathbf{x}^n}\right)_{n=1}^{9}$ denote a data set of nine points and labels $\ell_i < \ell_{i+1}$, $i = 1, \ldots, 4$; see Figure 2.2 (for clarity we present only the labels of the points). We focus on the point $\mathbf{x} \in D$ with label ℓ_1.

We now compute the change in the number of correctly labeled points with respect to $\mathbf{x}$ if we relabel $\mathbf{x}$ with $\ell \neq \ell_1$. Figure 2.3 shows the results.

It is obvious that the maximal increase is obtained for ℓ_4, and that $I^{\mathbf{x}}_{max} = 4$. Furthermore, it is easily seen that for all other points $\mathbf{x}' \in D$, $I^{\mathbf{x}'}_{max} \leq 3$.

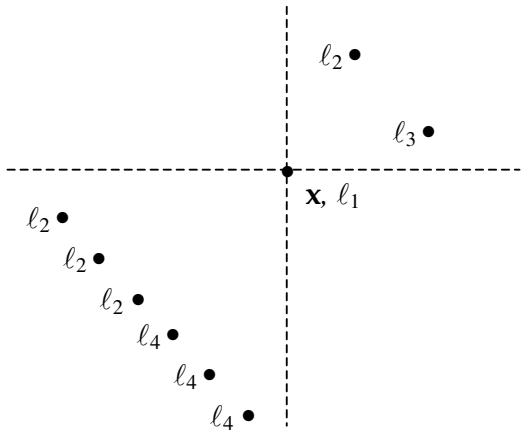


Figure 2.2: Distribution of all points in D with respect to $\mathbf{x}$.

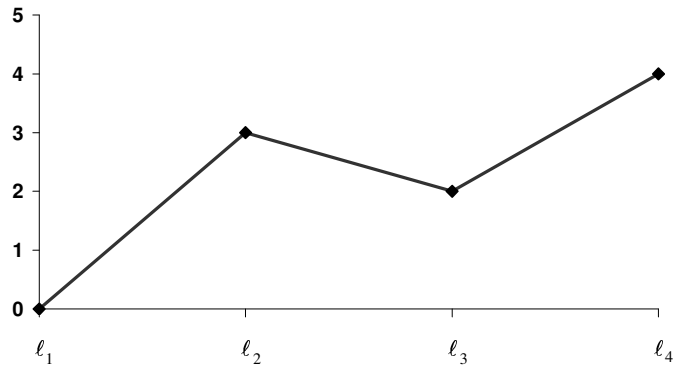


Figure 2.3: The change in the number of correctly labeled points with respect to $\mathbf{x}$ for the range of labels from ℓ_1 to ℓ_4

Consequently $\mathbf{x}$ is the point with the maximal increase.

Minimum number of label changes

Given the description of the algorithm and the efficiency issues discussed so far, there is no guarantee that the greedy algorithm will lead to a minimum number of label changes. To illustrate this property, we consider the following example.

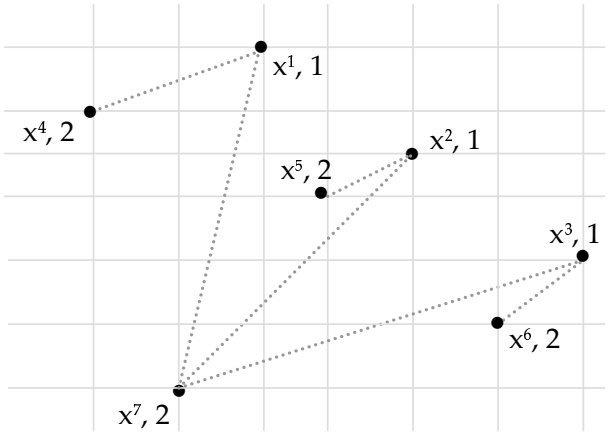


Figure 2.4: Data set of seven points in a two-dimensional input space

Figure 2.4 represents the structure of a data set of seven points with their labels, in a two-dimensional input space; the dotted lines show the non-monotone relationships between the points. Obviously, the data set is non-monotone. In order to make these data monotone, we apply the greedy algorithm. First, we compute the maximal increase in the number of correctly labeled points with respect to each point, as follows.

$$\begin{array}{cccc} \mathbf{x}^1,\mathbf{x}^2,\mathbf{x}^3 & \mathbf{x}^4 & \mathbf{x}^5,\mathbf{x}^6 & \mathbf{x}^7 \\ \boldsymbol{N_1} = 0 & N_1 = 1 & N_1 = 1 & N_1 = 5 \\ N_2 = 2 & \boldsymbol{N_2} = 0 & \boldsymbol{N_2} = 1 & \boldsymbol{N_2} = 2 \\ \Downarrow & \Downarrow & \Downarrow & \Downarrow \\ I^{\mathbf{x}^1}_{max} = I^{\mathbf{x}^2}_{max} = I^{\mathbf{x}^3}_{max} = 2 & I^{\mathbf{x}^4}_{max} = 1 & I^{\mathbf{x}^5}_{max} = I^{\mathbf{x}^6}_{max} = 0 & I^{\mathbf{x}^7}_{max} = 3 \end{array}$$

The results show that the maximal increase in the number of correctly labeled points, $I_{max} = 3$, is obtained for $\mathbf{x}^7$; so, this will be the point chosen to be relabeled at the first step. In the next steps, the algorithm needs to relabel three other points to make the data monotone. In other words, the greedy algorithm will make four label changes in total. However, if we relabel only the three points $\mathbf{x}^1$, $\mathbf{x}^2$, and $\mathbf{x}^3$, we also obtain a monotone data set.

This indicates that the greedy algorithm could make a sub-optimal choice for the set of points to be relabeled. However, we expect that this will happen only in pathological examples (such as the one shown here), which are rare in practice.

Comparison between the greedy algorithm for relabeling and the minimum flow algorithm

As we showed in the previous section, the greedy algorithm for relabeling does not guarantee a minimum number of label changes to make the data monotone. There exists a polynomial-time optimal relabeling algorithm that is based on the flow network (graph) concept. In Appendix A we discuss the theoretical background of network flow problems, in general, and *minimum network flow problems*, in particular; the solution to the latter gives the solution to the problem of making data monotone with the minimum number of label changes. We apply both the greedy algorithm and the minimum flow algorithm to the real data sets described in Section 2.5. It turns out that both algorithms give the same number of points that need to be relabeled to make the data monotone. Below we compare the algorithms from different perspectives.

For practical applications, the most attractive characteristic of both algorithms is their efficiency, i.e., they solve the problem of making data monotone in polynomial time (see Sections 2.4.3 and A.2). Furthermore, the minimum flow algorithm guarantees finding the minimum number of points that need to be relabeled. In our experiments with real data, in all cases the same minimum number was found by the greedy algorithm for relabeling. This suggests that the number of label changes made by the greedy algorithm is very close to the minimum, in general.

One advantage of the greedy algorithm over the minimum flow algorithm is that the former has explicit stepwise nature. This allows the user to get more insight into the data by directly finding points that violate the monotonicity constraint most. This result can be used for additional data analysis. The outcome from the minimum flow algorithm is a set of points that can be relabeled in an arbitrary order. To find the points with most violations, the user needs to make additional computations.

2.4.3 Complexity

In the greedy algorithm for relabeling the basic operations at each step are finding the set of non-monotone points, $Q(D)$, and computing the maximal increase in the number of correctly labeled points with respect to each point $\mathbf{x} \in Q(D)$. Hence the time required by the algorithm depends on both the structure of the data under study (number of monotone points, number of comparable pairs) and the order in which the points are chosen for relabeling (see the example in Section 2.4.2). Since in practice the data structures vary, it is impossible to estimate the exact complexity of the algorithm. Therefore, we discuss only the worst case.

Let D denote a data set of N points and $\tilde{L}$ labels. At each iteration of the algorithm, we compute $Q(D')$ where D' denotes the modified data set after a number of label changes. Suppose there are p points in $Q(D')$. Then, for each step described in Algorithm 2.1 (see Section 2.4.1), the effort is computed as follows:

$$\begin{aligned} p\,\frac{N(N-1)}{2} &\quad \text{to compute } Q(D') \\ p\,\tilde{L}(N-1) &\quad \text{to compute } I_{max} \\ \tilde{L} &\quad \text{to compute } \Lambda \\ p\,\tilde{L} &\quad \text{to form the triples} \\ p &\quad \text{to find the triple with maximal } I_{max} \end{aligned}$$

Hence, the total effort for one iteration, $C(p)$, is

$$C(p) = p\,\frac{N(N-1)}{2} + p\,\tilde{L}(N-1) + \tilde{L} + p\,\tilde{L} + p.$$

In the worst case when there are N non-monotone points in the data set and $Q(D)$ decreases by only one point at each step, the complexity is

$$\sum_{p=2}^{N} C(p) = O(N^3\tilde{L}).$$

This result shows that the greedy algorithm is polynomial in the number of points and labels in the data.

2.4.4 Simulation studies

In order to check to what extent the algorithm for relabeling can remove noise added to a monotone data set, simulation studies were conducted using artificial data with continuous and discrete labels, respectively.

Continuous labels

First we generate a data set D_1 of N points with k independent variables that are drawn randomly from the uniform distribution on [0,1]. The label of each point is computed by applying a monotone function to the independent variables. Depending on the number of the independent variables, several monotone functions are used to construct the initial label; for example, $x_1 sin\frac{\pi}{2} x_2$ based on two independent variables x_1 and x_2, or $x_1 x_2 sin\frac{\pi}{2} x_3$ based on three independent variables x_1, x_2, and x_3. Then, the monotone data set is converted into a non-monotone one D_2 by adding random noise to the labels. Next, the algorithm for relabeling is applied to the modified data to obtain a monotone data set D_3. In the next step, the mean-squared error (MSE) is used as a performance measure to check to what extent the algorithm restores the original data:

$$\text{MSE}_{mon} = \frac{1}{N}\sum_{i=1}^{N}\left(\ell_i^{D_3} - \ell_i^{D_1}\right)^2 \quad \text{and} \quad \text{MSE}_{nonmon} = \frac{1}{N}\sum_{i=1}^{N}\left(\ell_i^{D_2} - \ell_i^{D_1}\right)^2$$

where ℓ^{D_j} is the label set in the data set D_j, $j = 1, \ldots, 3$.. This experiment was repeated ten times with different numbers of points, independent variables, and percentages of noise ranging from 7% to 16%. The results, summarized in Table 2.2, show that the cleaned data are much closer to the original one than the noisy data.

Discrete labels

Following the same experimental set-up we also carried out simulation studies with discrete label data. The only difference is that we discretize the continuous dependent variable (label) into a finite number of classes. To do so, we split the range of estimated (continuous) function values into a number of intervals corresponding to the number N_{cl} of classes in the final

Table 2.2: Results after implementation of the algorithm for relabeling on artificially generated data sets with continuous labels

# points in a data set	# independent variables	Noise	MSE	
			Monotone data	Non-monotone data
100	2	10 %	0.0008	0.0211
100	2	15 %	0.0015	0.0240
100	3	12 %	0.0014	0.0055
100	3	15 %	0.0029	0.0133
100	5	14 %	0.0079	0.0292
200	2	7 %	0.0009	0.0224
200	2	15%	0.0006	0.0518
200	3	10%	0.0044	0.0250
200	5	12%	0.0152	0.0222
200	5	15%	0.0244	0.0885

data. Each interval i, $i = 1, 2, \ldots N_{cl}$, is defined by $\big[lb(i), lb(i+1)\big)$; $lb(i)$ is the lower bound of the interval computed by

$$lb(i) = \frac{i-1}{N_{cl}} \cdot z_{max},$$

where z_{max} is the maximum in the range of continuous values.

In the next step, we turn again the monotone data set into a non-monotone set by adding random noise to the discrete labels. This is done by changing randomly the labels with certain probabilities; for example, label ℓ_2 does not change with a probability of 90% and change to either ℓ_1 or ℓ_3 with a probability of 5%. The algorithm for relabeling is applied to the modified data and the percentage of correctly restored labels was computed. Table 2.3 shows that the algorithm restores the original data set to a large extent (7 out of 10 times the restoration is above 90%).

Software. The implementation of the greedy algorithm for relabeling is done in MATLAB (MATrix LABoratory), a powerful language providing an interactive environment for algorithm development, data analysis, simulation and technical computing (see the MATLAB web-site in the bibliography).

Table 2.3: Results after implementation of the algorithm for relabeling on artificially generated data sets with discrete labels

# points in a data set	# independent variables	# label categories	Noise	Restoration (%)
100	2	3	15 %	99 %
100	2	3	15 %	98 %
100	2	4	11 %	96 %
100	3	4	15 %	94 %
100	5	3	15 %	88 %
200	2	3	15 %	97 %
200	3	4	16 %	92 %
200	3	5	16 %	92 %
200	5	4	15 %	89 %
200	7	5	15 %	88 %

2.4.5 Other issues

Comparison of the algorithm for relabeling with isotonic regression

As we discussed in Section 2.2, isotonic regression is a popular technique for transformation of a non-monotone into a monotone data set. Given the same objective of the greedy algorithm for relabeling, in this section we compare both approaches by providing two simple examples and draw some conclusions about their application.

Example–1. Let us consider a simple data set D of two data points $\mathbf{x}^1$ and $\mathbf{x}^2$ with labels $\ell_{\mathbf{x}^1}$=3 and $\ell_{\mathbf{x}^2}$=1, respectively. Furthermore, we assume that $\mathbf{x}^1 \leq \mathbf{x}^2$. Hence, the pair $(\mathbf{x}^1, \mathbf{x}^2)$ is non-monotone, so D also is a non-monotone data set. In order to resolve the inconsistency, we can apply either the greedy algorithm for relabeling or isotonic regression. The results from the application of both approaches are presented in Figure 2.5.

As the dashed lines in Figure 2.5 show, the greedy algorithm for relabeling will relabel only one of the points: either $\ell_{\mathbf{x}^1}$ from 3 to 1, or $\ell_{\mathbf{x}^2}$ from 1 to 3 in order to obtain monotone data. The double dotted lines in the figure represent the changes made by the isotonic regression: both data points will obtain new labels, namely $\ell_{\mathbf{x}^1} = \ell_{\mathbf{x}^2} = 2$, because this method tries to minimize the mean-square error. So, this simple example illustrates three main differences between both approaches:

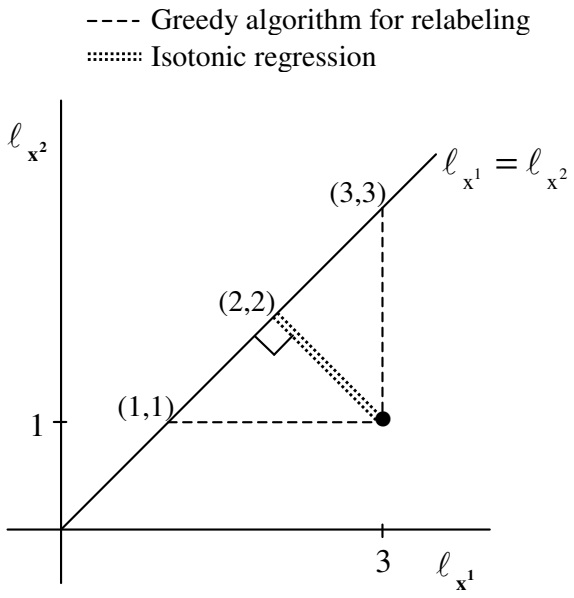


Figure 2.5: Application of the algorithm for relabeling and isotonic regression on a non-monotone data set of two points.

1. Isotonic regression yields a unique solution for the monotone data set in least-square sense, whereas the greedy algorithm for relabeling may lead to more than one monotone data sets.

2. At each step of the modification procedure, isotonic regression may relabel more than one point at once, whereas the greedy algorithm for relabeling changes the label of one point only.

3. The new labels assigned by the greedy algorithm for relabeling always belong to the label set in the original data, which may not be the case in isotonic regression.

Example–2. Now we consider a data set D of four points that take values in a two-dimensional input space. The partial ordering between the points is represented by a matrix order, i.e., the set of points form a two-dimensional grid and the ordering is the natural two-dimensional ordering ($\mathbf{x}_{ij} \leq \mathbf{x}_{pq}$ if $i \leq p$ and $j \leq q$). The structure and the corresponding labels of the points in D are given below:

$$D = \begin{pmatrix} 10 & 4 \\ 4 & 6 \end{pmatrix}.$$

In these data there are three non-monotone pairs, namely $\mathbf{x}_{11}$ with the remaining three points; for example, $\mathbf{x}_{11} \leq \mathbf{x}_{12}$ but $\ell_{\mathbf{x}_{11}} = 10 > 4 = \ell_{\mathbf{x}_{12}}$. Hence, D is a non-monotone data set. Again, we apply the greedy algorithm for relabeling and isotonic regression to make the data monotone and we compare the outcomes.

In the greedy algorithm for relabeling, first we find the point with the maximal increase in the number of correctly labeled points; it is $\mathbf{x}_{11}$ with $I_{max} = 3$, if we relabel $\mathbf{x}_{11}$ from 10 to 4; note that $I_{max} = 0$ for $\mathbf{x}_{12}$ and $\mathbf{x}_{21}$, and $I_{max} = 1$ for $\mathbf{x}_{22}$. Then we perform the relabeling of $\mathbf{x}_{11}$. The resulting data are monotone:

$$D' = \begin{pmatrix} 4 & 4 \\ 4 & 6 \end{pmatrix}.$$

In isotonic regression, first we combine in one block all the points that violate monotonicity, i.e., the four points in D. Then we relabel these points with their weighted average, which is $(10 + 4 + 4 + 6)/4 = 6$. The resulting data are monotone:

$$D'' = \begin{pmatrix} 6 & 6 \\ 6 & 6 \end{pmatrix}.$$

Though in one step in both algorithms we obtain a monotone data set, the resulting data sets, D' and D'', are different, which is due to the different objectives of the methods. The greedy algorithm tries to obtain monotone data by relabeling only one point at each step, and as few points as possible in the whole data. Isotonic regression aims to make the data monotone by minimizing the least squares deviations between the labels of the non-monotone points. Hence, other major differences between both algorithms are:

1. Our algorithm for relabeling preserves the major set of original labels, whereas isotonic regression may lead to completely different set of labels; this means that with the greedy algorithm we try to obtain monotone data that are close to the original data by making as less changes as possible.

2. In the presence of outliers violating monotonicity, the isotonic regression yields a constant value solution for all the points participating in non-monotone relationships with the outliers (as in the above example), whereas the greedy algorithm modifies only the labels of the outliers, preserving the labels of the other points.

Relabeling versus deletion of data points

In the following proposition we show that the minimum number of points that need to be either relabeled or removed from a data set in order to make the data monotone is the same.

Proposition 2.4.1. *Suppose D is a non-monotone data set. To make D monotone, either a minimum number of data points m^r can be relabeled appropriately or a minimum number of data points m^d can be deleted from D. Then*

$$m^r = m^d.$$

Proof. First we consider the minimum number of points m^r that should be relabeled to turn D into a monotone data set. If we remove all these points from D, we obtain monotone data. Hence,

$$m^r \geq m^d. \tag{2.12}$$

Now, we remove the minimum number of points m^d that should be deleted from D in order to get monotone data. By applying an appropriate relabeling procedure to all these points, the data set obtained is monotone. To show this, take one of the deleted points, say $\mathbf{x}$. Define $\underline{\ell} = \max\{\text{labels in } Down_D(\mathbf{x}) \setminus \{\mathbf{x}\}\}$, and $\overline{\ell} = \min\{\text{labels in } Up_D(\mathbf{x}) \setminus \{\mathbf{x}\}\}$. Note that $\underline{\ell} \leq \overline{\ell}$ as the data are monotone after the removal of m^d points, including $\mathbf{x}$. Then, we can add $\mathbf{x}$ to the data set with a new label from the range $[\underline{\ell}, \overline{\ell}]$. Thus, the relabeling of $\mathbf{x}$ does not violate monotonicity with the other points in the data, and the new data set is again monotone. Similarly, this procedure can be repeated for the other deleted points such that the monotonicity of the final data set is preserved. Hence,

$$m^r \leq m^d. \tag{2.13}$$

From (2.12) and (2.13) it follows that $m^r = m^d$. □

Note that although the minimum number of points to be relabeled is equivalent to the minimum number of points to be deleted from a data set, we obtain different monotone data sets as a result. As the latter case leads to loss of information that might have significant influence on the decision-making process, ignoring data points is hardly considered as a good approach to get monotone data. Therefore in the case studies presented in Section 2.5 we use the greedy algorithm for relabeling to make the data sets monotone.

2.5 Real case studies

In this section, we introduce two case studies where monotonicity should hold in the data. The first one (bond rating) is a classification problem whereas the second one (house pricing) is a regression problem. For each case study we briefly describe the nature of the data set. Monotonicity of the data sets is verified by using the testing procedure introduced in Section 2.3. Since these data sets contain non-monotone pairs, we apply the greedy algorithm for relabeling to resolve the discrepancies. In Chapters 3 and 4 these data sets are used for building monotone models for prediction tasks.

A. Bond rating

As explained in Daniels and Kamp (1999), bond ratings are subjective opinions on the ability to pay interest and debt by economic entities such as industrial and financial companies, municipals, and public utilities. Bond ratings are published by two major bond rating agencies, namely Moody's and Standard & Poor's, in the form of a letter code, ranging from AAA–for excellent financial strength–to D for entities in default. Bond ratings are based on extensive financial analysis by the bond rating agencies. The exact determinants of a bond rating, however, are unknown, since the interpretation of financial information relies heavily on professional judgment.

Publications of bond rating agencies offer some insight into the relevant factors that determine bond ratings. Bond rating analysis recognizes the following areas of attention: profitability, liquidity, asset protection, indenture provisions, and management quality.

Bond rating models use independent variables, often calculated as ratios, which are predominantly derived from public financial statements. However, not all of the above-mentioned areas are covered by financial statement; for

Table 2.4: Definition of the variables for the bond rating data

Symbol	Definition
D/C	Debt to capital ratio
CF/D	5 years average cash flow to debt ratio
CF	5 years average cash flows (in 100 millions)
COV	3 years average interest coverage ratio
VOL/COV	3 years volatility of interest coverage

Table 2.5: Correlation coefficients between the input variables (Table 2.4) and bond rating

Variable	D/C	CF/D	CF	COV	VOL/COV
Corr. coef.	0.50	-0.64	-0.46	-0.52	0.38

example, aspects like quality of management, market positions, and asset protection are captured to a limited extent only.

From the Standard & Poor's Bond Guide (April 1994), we select 256 companies. The bond ratings of these companies range from AAA to D. The ratings are not homogeneously distributed; i.e., the largest classes are A, BBB, and B. Only a few companies have ratings lower than CCC. Therefore, we decided to remove all ratings below CCC. As in other studies, the + and − signs were omitted; for example, AA+, AA, and AA− are all considered as AA. Finally, the bond rating (class variable) contains seven distinctive categories.

From the S&P Bond Guide, several financial figures are obtained. From Datastream, we download additional financial figures and ratios relating to leverage, coverage, liquidity, profitability, and size. These figures have been transformed into 5 years averages and trend indicators, resulting in 45 explanatory variables. For each variable, the linear correlation with the quantified bond rating is calculated. For the purposes of the current case study, five variables with high correlations with respect to the bond rating are chosen; see Tables 2.4 and 2.5.

To synchronize the direction of influence of all the variables with respect to the bond rating, we perform a linear transformation on CF/D, CF, and COV such that they have a positive effect on the target variable. This is done

simply by reversing the range of values for each of the three variables.

The data set thus constructed contain unique points only. To verify the monotonicity of the data, we compare the indicators (fraction of monotone pairs and number of monotone points) computed from the real data with the benchmark measures defined in Section 2.3.1. The results are presented in Table 2.6.

Table 2.6: Degree of monotonicity of the bond rating data compared with benchmark data

Indicators	Bond rating	Benchmark data
Comparable pairs	9 685	9 685
Fraction of monotone pairs	0.99	0.60
Number of points	256	256
Monotone points	168	2.7

Next we perform a statistical test as described in Section 2.3.2 to check the significance of the difference between the indicators obtained from the real and benchmark data. For this purpose we generate a collection of 1000 benchmark data sets and for each of them we generate the empirical distribution of both indicators (fraction of monotone pairs and number of monotone points); see Figures 2.6 and 2.7. Note that the empirical mean of the fraction of monotone pairs computed from the collection of benchmark data sets is 0.60, which is the same as the theoretical benchmark measure reported in Table 2.6.

Next we compute the critical values of the statistics corresponding to both indicators at 95% and 99% significance levels; see Table 2.7.

Table 2.7: Critical values of both benchmark measures for monotonicity of the bond rating data based on the empirical distribution of a collection of 1000 data sets

Significance level	Fraction of monotone pairs	Number of monotone points
95 %	0.66	6
99 %	0.68	8

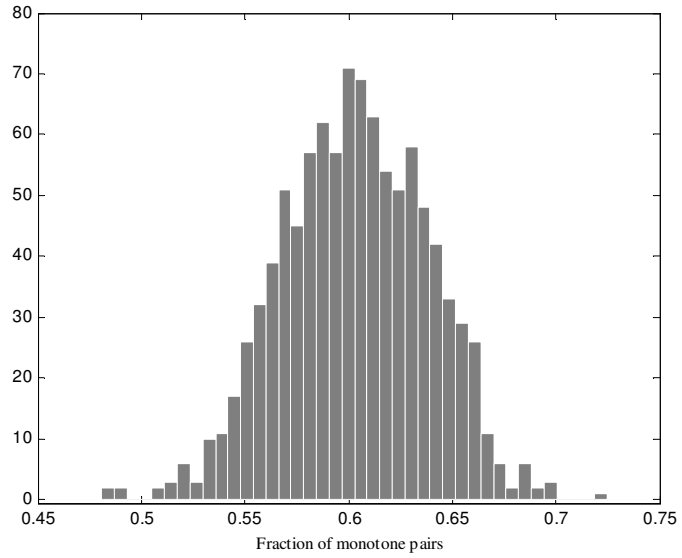


Figure 2.6: Empirical distribution of the fraction of monotone pairs generated from a collection of 1000 benchmark data sets based on the bond rating data

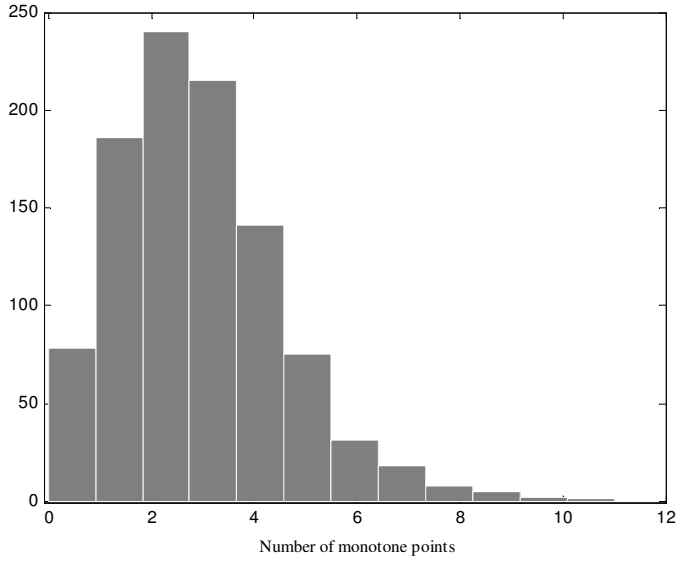


Figure 2.7: Empirical distribution of the number of monotone points generated from a collection of 1000 benchmark data sets based on the bond rating data

The results show that the critical values of the benchmark measures are smaller than the observed indicators reported in Table 2.6. Hence, at the given significance levels the observed indicators fall in the right tail of the distribution and we reject the null hypothesis in favor of the alternative. This implies that the labels in the bond rating data preserve ordering with respect to the points, i.e., there exist monotone relationships between the target and all predictor variables in the data. This clearly indicates that the bond rating data represent a class of monotone problems, and therefore these data can be used to build monotone decision models for prediction tasks.

Finally, as a data pre-processing step before using the data in the modeling process, we suggest to remove the noise and resolve the discrepancies in the bond rating data by applying the greedy algorithm for relabeling. As a result, monotone data have been obtained after 28 label changes.

B. Moscow house pricing

The basic principle of a hedonic price model is that the consumption good is regarded as a bundle of characteristics for which a valuation exists (Harrison and Rubinfeld, 1978). The price P of the good is determined by a combination of these valuations x

$$P = P(x_1, x_2, \ldots, x_k).$$

In the case study presented here we want to predict the house price, given a number of house characteristics. The data set consists of 150 observations of flats in the city of Moscow. In the original data set, there are ten explanatory variables. For each of them, we calculated the correlation coefficient with the flat price. For the purposes of the current case study, we chose six variables with the highest correlation; see Table 2.8.

The correlations in Table 2.9 suggest that the total flat area, living room area, and the number of rooms are the most important determinants of the housing value. The direction of influence corresponds to common sense: more area and rooms will, in general, result in a higher flat value. In addition, for the sake of computational and analytical convenience, we reversed the direction of influence of DISTKM on the flat price. This is done by the following linear transformation:

$$\forall_{1 \le i \le 150} \text{DISTKM}_i = \text{DISTKM}_{max} - \text{DISTKM}_i + \text{DISTKM}_{min},$$

Table 2.8: Definition of the input variables for the Moscow data

Symbol	Definition
TOTSPACE	Total flat area
LIVSPACE	Living room area
KITSPACE	Kitchen room area
DISTKM	Distance in km from the center
ROOMS	Number of rooms
BRICK	Brick flat or not

Table 2.9: The correlation coefficients between the input variables (Table 2.8) and the house price

Variable	TOTSPACE	LIVSPACE	KITSPACE	DISTKM	ROOMS	BRICK
Corr.coef.	0.88	0.85	0.65	-0.38	0.74	0.42

Table 2.10: Degree of monotonicity of the Moscow data compared with benchmark data

Indicators	Moscow data	Benchmark data
Comparable pairs	1 699	1 699
Fraction of monotone pairs	0.81	0.51
Number of points	150	150
Monotone points	25	7.2

where DISTKM$_{max}$ and DISTKM$_{min}$ are the maximal and minimal value of DISTKM in the data. Furthermore, for computational convenience the dependent variable was transformed by taking its logarithm.

The data set thus constructed contain unique points only. Similarly to the bond rating case study, we verify the monotonicity of the Moscow data set by comparing the observed indicators with the benchmark measures; see Table 2.10.

Analogously to the bond rating case study, we perform a statistical test to check the significance of the difference between the two observed and benchmark measures. We again generate a collection of 1000 benchmark data sets and for each of them we generate the empirical distribution of both

indicators (fraction of monotone pairs and number of monotone points); see Figures 2.8 and 2.9. The empirical benchmark measure for the fraction of monotone pairs is 0.51, which is the same as the theoretical measure given in Table 2.10.

Next, we find the critical values of both benchmark measures for monotonicity at 95% and 99% significance levels; see Table 2.11. The results show that both critical values are smaller than the corresponding observed indicators given in Table 2.10, which leads to rejection of the null hypothesis in favor of the alternative. The conclusion drawn from the statistical test is that there is a significant difference between the observed and benchmark measures for the degree of monotonicity. This indicates that the Moscow data set is another example of a monotone problem, which can be used to build monotone models for prediction tasks.

Table 2.11: Critical values of both benchmark measures for monotonicity of the Moscow data based on the empirical distribution of a collection of 1000 data sets

Significance level	Fraction of monotone pairs	Number of monotone points
95 %	0.60	13
99 %	0.63	16

Finally, the greedy algorithm for relabeling is applied to the Moscow data set to obtain monotone data, which led to 54 label changes.

2.6 Conclusion

In this chapter, we discussed the notion of monotonicity in data sets. We started the discussion with the definitions of several concepts, namely monotone point, monotone/non-monotone pair of points, and monotone/non-monotone (noisy) data set. In practice, we wish to know to what extent the monotonicity assumption holds for given real data. For this purpose, we introduced a novel procedure for testing the degree of monotonicity of a real data set. The procedure is based on the comparison between measures computed from the real data and benchmark data. The latter is defined by

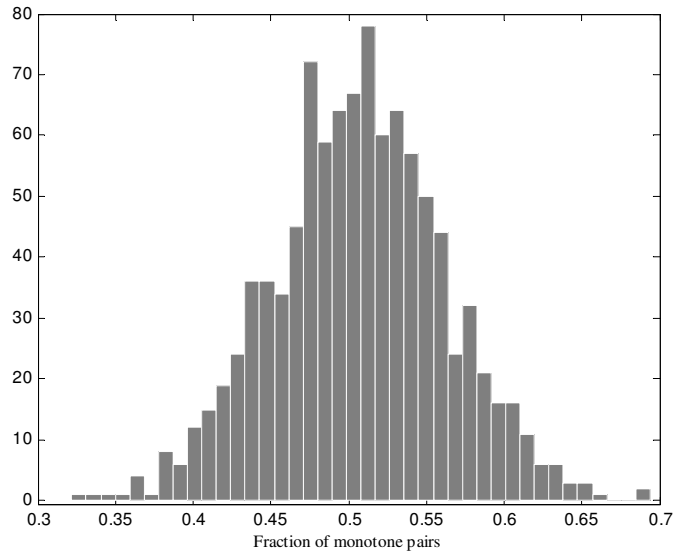


Figure 2.8: Empirical distribution of the fraction of monotone pairs generated from a collection of 1000 benchmark data sets based on the Moscow data

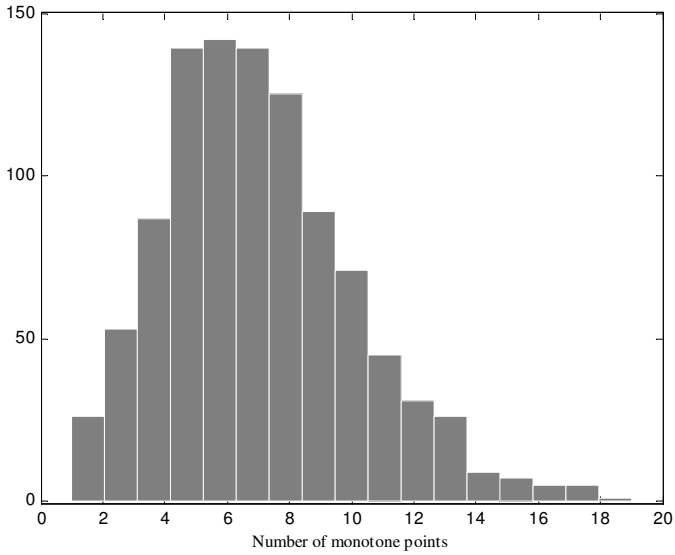


Figure 2.9: Empirical distribution of the number of monotone points generated from a collection of 1000 benchmark data sets based on the Moscow data

taking the same structure of the independent variables as in the real data and adding a random permutation of the set of the original labels. Two measures are computed from the real data–fraction of monotone pairs and number of monotone points. We derive their counterparts from the benchmark data as expected values. By using a statistical test, we check the significance of the difference between the real and benchmark measures. If the real indicators are significantly larger than the benchmarks, then we can conclude that there are monotone relationships established in the real data; otherwise, the monotonicity assumption is questionable.

Given a real data set with a small number of monotonicity violations, another interesting problem is how to make it monotone. The solution we provided in this chapter is the greedy algorithm for relabeling, which modifies the labels of the points participating in non-monotone pairs. The main idea is to make as few label changes as possible so that the original data are preserved. We prove that the greedy algorithm always leads to monotone data, though the solution may not be unique. We conducted simulation studies with artificial data with discrete and continuous labels in order to show that the greedy algorithm is capable of removing noise to a large extent.

Finally, we demonstrated the application of the procedure for testing monotonicity of data and our algorithm for relabeling on two real case studies: bond rating (classification problem) and house pricing (regression problem). The results from the testing procedure confirm our expectations that the monotonicity assumption holds in both data sets. The greedy algorithm for relabeling is used further to make the data sets monotone by resolving the inconsistencies.

Chapter 3

Monotone decision trees

As explained in Chapter 1, decision trees are one of the most popular techniques for prediction in data mining. Given the objectives of our research, in this chapter we discuss *monotone decision trees* as a method to derive monotone prediction models. We begin with an introduction of a standard algorithm for tree construction and definitions of the main concepts, which are used in the follow-up discussion. Then, we provide a general overview of earlier studies that deal with the generation of decision trees with monotonicity constraints, and discuss their main advantages and disadvantages. In the main part of the chapter, we describe our extended version of an algorithm for building monotone decision trees, which is based on an approach proposed by Feelders (2000). By using the two case studies introduced in the previous chapter, we demonstrate the application of the algorithm and draw some general conclusions regarding the performance of the monotone decision tree models. The work presented in this chapter has been published in Velikova and Daniels (2004) and Daniels and Velikova (2006).

3.1 Introduction

Several approaches have been developed for building decision trees. Among the most popular ones are CART (Classification and Regression Trees) developed by Breiman et al. (1984) and ID3 (Induction Decision Tree) with its later versions C4.5 and C5.0 developed by Quinlan (1986, 1993, 2005). Both approaches use similar principles to build a tree.

Now we present the main tree construction scheme, which is primarily

based on CART. Starting from the root of a tree, binary splits are made on each non-terminal node t; terminal nodes (leaves) are without branches and contain the predicted value of the response variable. The splits are based on one input variable x. If x is continuous, the split is of the form $x \leq c$ or $x > c$, for some constant c. If x is categorical, the split is of the form $x \in S$ or $x \notin S$, where S is a non-empty subset of x's possible categories. The variable x and its value for splitting are selected through a criterion $i(t)$, which measures the "impurity" of a node t. The basic idea is to choose a split such that the child nodes are "purer" than their parent node, i.e., they contain objects that have the same or close (for example in least-squares sense) responses.

In practice, various splitting criteria are used depending on the task at hand. In classification problems, the class probabilities $\Pr(\ell|t)$, $\ell \in \mathcal{L}$ in each node t are first estimated by:

$$\Pr(\ell|t) = \frac{1}{N(t)} \sum_{\mathbf{x} \in t} \mathtt{I}\left(\ell_{\mathbf{x}} = \ell\right),$$

where $\mathtt{I}\left(\ell_{\mathbf{x}} = \ell\right) = 1$ if $\ell_{\mathbf{x}} = \ell$ and $\mathtt{I}\left(\ell_{\mathbf{x}} = \ell\right) = 0$ if $\ell_{\mathbf{x}} \neq \ell$; $N(t)$ is the total number of objects belonging to t.

Then, typical splitting criteria are the following:

- *Gini-index*: this measures the variance of the response variable obtained by observing the class label of an example drawn at random (with replacement) from node t. The ideal objective is to obtain pure nodes with zero variance in the class label. The actual variance is

$$i(t) = \sum_{\ell} \Pr(\ell|t)\left(1 - \Pr(\ell|t)\right).$$

- *Entropy*: this measures the average amount of information yielded by observing the class label of an example drawn at random (with replacement) from node t. If a node is pure, this information and thus the entropy is zero. The actual entropy as defined by Shannon (1948) is

$$i(t) = -\sum_{\ell} \Pr(\ell|t) \log \Pr(\ell|t).$$

In regression problems, the most natural and commonly used splitting criterion is the *mean-squared error*, which measures the total squared deviations of the value of the response variable $\ell_{\mathbf{x}}$ of each case $\mathbf{x}$ in node t from the average response value $\bar{\ell}(t)$ for all cases belonging to that node:

$$i(t) = \frac{1}{N(t)} \sum_{\mathbf{x} \in t} \left(\ell_{\mathbf{x}} - \bar{\ell}(t)\right)^2 .$$

Besides the splitting criterion (impurity measure), we also define the quality of a split as the reduction of impurity that the split achieves; that is for split s in node t:

$$\Delta i(s,t) = i(t) - \pi(\lambda) i(\lambda) - \pi(r) i(r),$$

where $\pi(\lambda)$ is the proportion of objects sent to the left by the split, and $\pi(r)$ is the proportion of objects sent to the right.

The partitioning process is applied recursively to each node continuing until either all leaves are pure (i.e., they contain one or more objects with a unique label) or further splits cannot be performed (e.g., identical objects belonging to a node have different labels). Then the resulting tree (almost) perfectly fits the data used for the model construction. However, we are interested in the general prediction capabilities of the tree, which is determined by its performance for new data. Fitting the tree perfectly to the data under study, we have "overfitted" the model; this may result in a high prediction error on new data. Therefore, a crucial issue in tree construction is to determine the right size of the final tree.

One approach is to use a *stopping rule*, which prevents expanding a node if the maximum reduction in the impurity measure for the best split is below some threshold θ. However, it is not trivial to determine the value of θ. On the one hand, too low a value of θ leads to many splits, which results in a large tree. On the other hand, too large a value of θ may result in a node to be declared a terminal node–due to small impurity reduction; its child nodes, however, may have good splits, which will be lost in this case. An alternative stopping rule is defined by the minimal number of observations that fall in a node. In other words, the splitting of a node is terminated when the number of observations is below some threshold. Again, setting the value of the threshold is done in an ad-hoc way, which may complicate the process of finding the right size of a tree. In summary, using stopping

rules to determine whether to grow a new node does not produce satisfactory results in general.

Therefore, a more plausible approach is to apply *pruning* on the initial tree. The idea is to build a large tree with pure nodes first. Next, a sequence of pruned subtrees is generated from this tree by merging the nodes back up to the root of the tree. Finally the right-sized tree is selected from the resulting sequence of trees on the basis of a consistent estimate of the prediction error. The main problem with pruning is that the number of pruned trees may become very large and it would be infeasible to find the best tree.

One of the most popular and efficient pruning procedures applied in practice is so called *cost-complexity pruning* proposed by Breiman et al. (1984). Using similar notations as Breiman et al. (1984), we now give a general description of their approach.

Let T_{max} denote the initial tree obtained from a tree construction algorithm. Furthermore, let T be any pruned subtree of T_{max} (in short $T \leq T_{max}$). Suppose $R(T)$ is the error measure (cost) of T, and $|\tilde{T}|$ is the complexity of T defined as the number of terminal nodes. Then, the cost-complexity measure $C_\alpha(T)$ is defined by

$$C_\alpha(T) = R(T) + \alpha|\tilde{T}|,$$

where $\alpha \geq 0$ is the complexity parameter, which plays the role of a penalty for tree size.

Thus, $C_\alpha(T)$ represents the trade-off between the cost of a tree and its complexity. Now the objective of cost-complexity pruning is to find the sequence of smallest subtrees $T(\alpha) \leq T_{max}$ for different values of α such that $C_\alpha(T)$ is minimized, that is

$$C_\alpha(T(\alpha)) = \min_{T \leq T_{max}} C_\alpha(T).$$

This is done in a recursive manner. We start with T_{max} and find the smallest pruned subtree T_1 of T_{max} at $\alpha_1 = 0$ such that $R(T_1) = R(T_{max})$. Thus we obtain a tree that has the same total cost as T_{max} but because it is smaller it is preferred over T_{max}. Then, for all non-terminal nodes t in T_1, we define the following function $g(t)$:

$$g(t) = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1},$$

where $R(t)$ and $R(T_t)$ are the errors in node t and the tree T_t with root node t, respectively. Next, we select the nodes for which $g(t)$ is the smallest, and prune T_1 in these nodes to obtain T_2, the next tree in the sequence. The new value α_2 of α is set to be

$$\alpha_2 = \min_t g(t).$$

In other words, α_2 is the minimal value of α at which $C_\alpha(t) = C_\alpha(T_t)$. This is easily derived from the definition of $g(t)$.

In the next step, we proceed with T_2 as the current tree to be pruned. This process is repeated, until the root node t_0 is reached. As a result a nested sequence of pruned subtrees of T_{max} is obtained, i.e.,

$$T_1 > T_2 > \ldots > t_0.$$

The final stage selects the optimum-sized tree from the generated tree sequence. The most natural way is to choose the tree with the best performance (lowest error) on new data. As we pointed out in Section 1.3.3, this can be done by using a separate test set held apart from the original data.

The efficiency of the cost-complexity pruning is due to two main factors, as shown by Breiman et al. (1984). First, the authors prove that for each value of α there exists a smallest minimizing subtree. In other words, if there is more than one tree that minimizes the cost-complexity measure, then the smallest one is chosen, which is a subtree of all other minimizing trees. This means that it is impossible to have exactly two trees that minimize the cost-complexity measure but are incomparable, i.e., neither tree is a subtree of the other. This finding leads to the second important result obtained by Breiman et al. (1984), namely that the final outcome of the cost-complexity pruning is a *nested* sequence of trees. It means that the next tree in the sequence can be obtained by pruning the current tree. As a result the number of trees that need to be pruned is considerably reduced, which leads to an efficient pruning procedure.

The tree construction algorithm thus described, in its original form does not guarantee that the constructed tree is monotone, even if the underlying data set is monotone.

Potharst (1999) proposes a straightforward approach for testing the monotonicity of a decision tree. His approach is based on a comparison of the so-called minimal and maximal elements of the leaves in the decision tree,

which are defined as follows. Given a leaf node t of tree T, the subset of the input space $\mathcal{X}$ associated with that node is represented by

$$t = \{\mathbf{x} \in \mathcal{X} : \mathbf{a} \leq \mathbf{x} \leq \mathbf{b}\}, \text{ for } \mathbf{a}, \mathbf{b} \in \bar{\mathcal{X}},$$

where $\bar{\mathcal{X}} = \mathcal{X} \cup \{+\infty\} \cup \{-\infty\}$. Then, $\mathbf{a}(t)$ is the *minimal element* and $\mathbf{b}(t)$ is the *maximal element* of t; both elements are called *corner elements* of t. The test for monotonicity of the tree T is performed by the following algorithm proposed by Potharst (1999), where $\ell(t)$ denotes the label assigned to node t:

Algorithm 3.1 Monotonicity testing of a decision tree

for all pairs of leaves t, t':
  if $(\ell(t) > \ell(t')$ **and** $\mathbf{a}(t) \leq \mathbf{b}(t'))$ **or**
    $(\ell(t') > \ell(t)$ **and** $\mathbf{a}(t') \leq \mathbf{b}(t))$
  then stop: T not monotone

Potharst (1999) proves that if a decision tree T is passed through the above algorithm without stopping, then T is monotone.

The idea of using minimal and maximal elements in the leaves in order to test the monotonicity of a decision tree is very intuitive and therefore, it has been applied in various research studies as described in the next section.

3.2 Related work

There are a number of approaches for constructing decision trees that incorporate monotonicity properties.

One of the earliest approaches is introduced by Ben-David (1995), which is a modification of the traditional tree algorithm ID3. The only difference is the use of a new splitting criterion called the total-ambiguity-score TA; for node t it is defined by adding a non-monotonicity index I_{nm} to the standard entropy-based impurity measure $i(t)$:

$$TA(t) = i(t) + \delta I_{nm}(t),$$

where the weight parameter $\delta > 0$ expresses the relative importance of monotonicity to the prediction accuracy. The non-monotonicity index I_{nm} is computed as the ratio of non-monotone leaf pairs of the tree obtained after

the candidate split is performed to the maximum possible number of non-monotone leaf pairs in the tree. The attribute with the lowest TA-score is selected to split a node.

The advantages of Ben-David's approach are that it balances the accuracy and monotonicity properties of a tree, and it works for both monotone and non-monotone data. The results from the case studies in Ben-David (1995) show that the trees generated with this approach have a significantly lower degree of non-monotonicity–without a significant deterioration in the prediction accuracy. However, a disadvantage is that the monotonicity of the resulting tree is not guaranteed.

Makino et al. (1999) propose a method to construct monotone decision trees for two-class problems with ordinal attributes. It also uses a modification of the splitting criterion in the ID3 algorithm. Furthermore, the monotonicity of a tree is enforced by adding at each step (if necessary) the corner (minimal and maximal) elements of a node with appropriate labels. Their approach has been extended by Potharst and Bioch (2000) for k-class problems; it also deals with continuous attributes. Though the monotonicity of a tree is guaranteed by these methods, the main limitation for their applicability is that they require totally monotone data sets (see Definition 2.1.3), which is seldomly the case in practice.

Bioch and Popova (2002) address the problem of generating monotone binary decision trees from noisy data. Their approach is a modification of the algorithm proposed by Potharst and Bioch (2000). The main idea is to add data points to the original data sample whenever inconsistencies occur during the tree construction. At each step of the algorithm, the corner elements of the node considered for splitting are relabeled with the consistent labels that are calculated from the data. Bioch and Popova (2002) prove that this procedure always generates a monotone decision tree.

Furthermore, Bioch and Popova (2002) note that the presence of noise in the data may require the addition of many new points to the data, which leads to complex (large) trees. In order to remedy this problem, they suggest two methods for pruning the constructed tree. The first method, called pre-pruning, is based on the application of a pruning procedure while growing the tree. This is done by stopping the generation of a branch if the number of points falling in each of the new leaves drops below a pre-defined threshold; the current node then is turned into a leaf. This node is labeled in such a way that monotonicity is preserved. The second method, called post-pruning,

is applied after a large tree has been generated. It is based on pruning back branches from the tree such that the misclassification rate of the new tree is below a pre-defined threshold. Again the monotonicity of the tree is guaranteed by a consistent labeling procedure. Experimental studies with artificial and real data are used to compare both pruning procedures. The advantage of pre-pruning is that it leads to smaller data sets, so it requires less resources for generating and storing the tree. However, with pre-pruning it is more difficult to decide when to stop expanding a node and what label to assign to it. Hence, as noted by Bioch and Popova (2002), for some data sets their post-pruning produces better results by pruning a large part of the tree without increasing the misclassification error.

Strobl et al. (2003) suggest to build monotone decision trees by combining the standard CART algorithm with isotonic regression (see Section 2.2). More specifically, the authors propose an extension of CART by allowing the tree-based algorithm to make multiple (not only binary) splits. The problem, which usually arises with multiple splits, is the fast increasing amount of possible splits that need to be tested and compared during the tree construction; this leads to a considerably slower algorithm and results in very large trees. To solve this problem, the authors use the reduced version of isotonic regression to determine the splits (cutpoints) in the tree. Their idea is that applying the reduced isotonic regression on each node considered for splitting leads to a small number of solution blocks, which act as subnodes after splitting. Furthermore, they suggest a modification of the goodness-of-split criterion, which takes into account the number of subnodes obtained after splitting and provides a fairer comparisons of two or more splits with different independent variables. Their criterion is based on a specially designed likelihood ratio test, which measures the likelihood of the subnodes obtained after splitting–given the parameters of the reduced isotonic regression used to perform the splitting. The lowest p-value of the likelihood ratio test determines the overall best split. By using simulation studies, the authors demonstrate that their approach can indeed find the correct cutpoints in the tree construction. In addition, the reduced isotonic regression could be applied as a stopping rule in the tree generating process. Finally, the authors compare the performance of their non-binary tree approach with the binary standard CART algorithm; they use a real data set on the occurrence of chronic bronchitis (dependent variable) given time and overall dust measure (independent variables). The results show that: (i) both approaches have

comparable performance in terms of misclassification error and can detect similar cutpoints; (ii) the non-binary tree tends to be more balanced (the branches of the tree have similar height) than its binary counterpart. Their enhanced version of the CART algorithm not only guarantees that the generated tree is monotone, but also reduces the computational effort and yields parsimonious models with optimal complexity.

Another tree-based approach that incorporates monotonicity (ordering) constraints is proposed by Cao-Van and De Baets (2003). The authors argue that most of the approaches for monotone classification focus on the prediction accuracy, and tend to ignore acceptability (ease of understanding and interpretability) of the models by the human decision makers. Therefore, Cao-Van and De Baets (2003) suggest a method to derive an interpretable and intuitive rule base for ordinal classification, which is represented by a tree. The method is based on concepts from the field of multicriteria decision aid (MCDA). The monotonicity in the approach is defined by the so-called *principle of partial dominance preservation*, which states that an object $\mathbf{x}$ with (partial) measurements dominating the (partial) measurements of another object $\mathbf{x}'$ should get evaluation (classification) f that is also at least as good, that is

$$\mathbf{x} > \mathbf{x}' \Rightarrow f(\mathbf{x}) \geq f(\mathbf{x}'). \tag{3.1}$$

It should be noted that (3.1) is a weaker version of (1.2) as the latter requires that two objects with identical measurements should have the same evaluation. Furthermore, the term *partial* in the definition of the principle means that the comparison between two objects is based on the information available for them; that is, in order to establish the partial dominance relationship between two objects it is sufficient to have their common subset of independent variables for which measurements are available. This property is especially useful in a tree growing process where at each step only one or a subset of independent variables are considered for splitting. The authors address several typical problems in the standard tree-based approaches for ordinal classification. First, they argue that the choice of split values is not trivial, because often some of the splits violate the monotonicity assumption. Second, the selection of the nodes for expansion is important because the leaves are interconnected by the partial dominance relation. Finally, the resulting tree may contain leaves with ambiguous evaluation or empty leaf

nodes. Their solutions for these problems are demonstrated by a real case study where the objective is to predict the use of contraception (low, moderate, high) based on four criteria: average number of years of education, urbanization, gross national product per capita, and expenditure on family planning. The authors start with growing a ranking tree. At each step of the tree construction procedure, the variable that leads to the smallest number of violations of the (weaker) principle of partial dominance preservation is chosen to split the current node. Furthermore, the order in which the nodes are expanded is determined by the degree of impurity of the nodes, i.e., the most impure node at a given step is selected for splitting. Their rationale is that this procedure will lead to a faster decrease in the overall impurity. Finally, to prevent overfitting, the authors stop growing a node if it contains less than four objects.

Based on the tree thus constructed, the next step is to derive a rule base. The main problem is related to the assignment of labels to the leaves. The authors argue that the traditional misclassification techniques might be inadequate in the context of ranking problems. Therefore, they apply two labeling strategies depending on the number of objects that belong to the leaves. If a leaf is empty, then the principle of partial dominance preservation is used to assign a label that is consistent with the labeling of the remaining leaves. If a leaf contains more objects with different class labels, then the label assigned is not a singleton (usually the most frequent class category) but an interval for the permissible values of the response variable that are consistent with the principle of partial dominance preservation. Further refinement of the derived rule base leads to a parsimonious model for ordinal classification. The final output of the approach is a labeled partial dominance graph that has a semantic interpretation. This property is the main advantage of the method, because it helps to reveal complex interactions between the attributes and their effects on the target variable.

The ultimate objective of the tree-based approaches considered so far is to construct monotone decision trees, which predict the value of the dependent variable. However, Lee et al. (2003) develop an alternative approach for generating monotone decision trees (MDT) for classification tasks. Assuming that all the attributes in the input space are ordinal, the authors aim to predict the implicit ordering between the objects rather than the labels themselves. Although the labels are related to the ordering of the objects, they do not necessarily reflect the original ordinal classification of the ob-

jects. Hence, the proposed approach can generate monotone decision trees even if the underlying data set is non-monotone or inconsistent (two objects with the same attribute values have different labels). The authors try to build a tree that can effectively separate the more dominant elements from the less dominant elements. Instead of using the standard misclassification error, the authors propose two new criteria to measure the quality of the derived ordinal classification model. The first measure provides insight into the similarity between the induced model and the original ordinal classification; it is based on the number of concordant pairs (pairs that exists in both orderings) and discordant pairs of observations. The second measure shows the effectiveness of the tree model to predict the pair ordering; it is computed as a percentage of the net concordant pairs (concordant less discordant pairs) in the ordering induced by the model.

The performance of MDT is tested on eight real data sets and compared with the performance of the standard CART algorithm. The results show that the MDT trees are smaller and hence easier to interpret than the CART trees. Furthermore, the accuracy of MDT is higher than that of CART in 20 out of 24 tests. In addition, MDT guarantees that the trees generated are monotone, which is required in ordinal classification.

A limitation of their approach, however, is that it requires the attributes to be ordinal, i.e., it does not deal with numeric attribute values. Although numeric data can be transformed into ordinal by taking a number of levels, this has several drawbacks: (i) additional data pre-processing is needed; (ii) setting the number of levels is not trivial, and is usually done ad-hoc; (iii) combining several attribute values into one level may lead to loss of important information in the knowledge discovery process.

Most of the approaches considered so far enforce monotonicity during tree construction. Although this strategy guarantees that the generated trees are monotone, Feelders (2000) argues that there are several disadvantages. First, the order in which the nodes are expanded is important; for example, the depth-first search and breadth-first search generally produce different trees (as shown by Bioch and Popova (2002)). Second, a non-monotone tree may become monotone after additional splits. Therefore, Feelders (2000) applies an alternative strategy for generating monotone decision trees. His approach is based on the standard CART algorithm with recording the corner elements of the nodes during tree construction in order to check the monotonicity of the generated tree. The method simply constructs many trees, and checks

whether or not they are monotone. In addition, Feelders (2000) proposes the following improvement of the non-monotonicity index introduced by Ben-David (1995): give each non-monotone leaf pair a weight computed as the proportion of the objects belonging to those leaf nodes. The rationale is that the non-monotone leaf pairs with small weights would violate monotonicity less than the non-monotone leaf pairs with large weights. This weighting procedure provides an upper bound for the degree of non-monotonicity of a tree.

Depending on the nature of the problem and data at hand, Feelders's method might be computationally intensive. However, for monotone prediction problems, it is expected that it will generate more monotone trees. Furthermore, by using this approach one can estimate the degree of non-monotonicity of a tree and check to what extent the assumptions for monotonicity are valid; this is an empirical alternative of the theoretical measures we derived in Section 2.3.

Feelders and Pardoel (2003) provide another study where pruning is applied to generate monotone classification trees. The authors propose a method that is similar to CART for growing the initial tree. As in Feelders (2000), the only difference is that the corner elements of each node are recorded during the tree construction in order to check the monotonicity of the tree. If there are monotonicity violations (i.e., there are non-monotone leaf pairs in the tree), then so-called *fixing methods* are applied to prune the tree such that the inconsistencies are resolved. The basic idea is to make a minimal number of adjustments by pruning parent nodes that have at least one child node participating in a non-monotone leaf pair. Depending on the choice of a parent node for pruning, those authors suggest various fixing methods. The most natural fixing method selects the node that leads to the biggest reduction in the number of non-monotone leaf pairs. In case the fixing method gives equally good fixes at a given step of the procedures, the authors consider an additional heuristic for choosing to prune the parent with the least number of observations. The argument is that the monotonicity violation is expected to be caused by a small number of inconsistent observations.

In addition to these methods, Feelders and Pardoel (2003) suggest various ways to combine their fixing methods with existing pruning techniques such as cost-complexity pruning (Breiman et al., 1984), discussed in the introduction of this chapter. One approach is to switch between the cost-

complexity pruning and fixing steps, which results in a sequence of monotone trees. If a pruned tree is monotone, it is added to the sequence; otherwise, it is fixed (made monotone) and then added. Another approach is to take the sequence of trees generated from cost-complexity pruning, and to apply fixing methods to all trees. As a third method, the authors suggest to take the best tree from the sequence (in terms of smallest misclassification error), and fix only that tree.

Feelders and Pardoel (2003) conduct various experimental studies with artificial and real data sets to check the performance of their fixing methods. The authors use two benchmarks for comparison, namely the best tree (with the smallest error) generated by the standard CART algorithm with cost-complexity pruning and the best monotone tree selected from the tree sequence produced by the standard algorithm. The results show that applying fixing methods to generate monotone trees leads to slightly but not significantly better predictions than the benchmarks. However, their approach has several advantages, as pointed out by Feelders and Pardoel: (i) it guarantees that the generated tree is monotone; (ii) it can be applied to both monotone and non-monotone data; (iii) it leads to monotone trees that are much smaller than the ones built by the standard algorithm.

Given the objectives of our research and the advantages and disadvantages of the discussed tree-based approaches with monotonicity constraints, we found that the simplest approach in practice is the one developed by Feelders (2000). We emphasize that our primary goal is to compare the performance of the monotone models derived from monotone and non-monotone data–showing that the former are better–rather than to build a new method or compare the prediction accuracies of different monotone tree-based algorithms. Hence, any of the developed approaches for generating monotone decision trees would serve our goal, but our choice to use Feelders's algorithm can be justified by the following pragmatic considerations:

- His approach is intuitive and easily applied; it simply generates trees and checks whether they are monotone.
- It is based on the standard CART algorithm; its wide application in practice is evidence of its effectiveness.
- While the approach has been originally developed to deal with classification problems, it is easily extended to regression problems (as shown in the next section).

Therefore, we apply Feelders's algorithm with some extensions to build monotone decision trees for the real case studies in Section 3.3.2. In the next section, we provide a detailed description of the approach with the additional modifications we have made.

3.3 Algorithm for building monotone decision trees

3.3.1 Implementation.

As mentioned in the previous section, the algorithm developed by Feelders (2000) in many respects is similar to the CART program described in Breiman et al. (1984). The original algorithm was created to construct classification trees, whereas we extend it to regression trees as well.

Our algorithm works as follows. Starting from the root node, which contains the full training data set, the program makes binary splits only using as the splitting criterion either the Gini-index in classification trees or the mean-squared error (MSE) in regression trees. The partitioning process is applied recursively to each non-leaf–continuing until all leaves are pure, or further splitting cannot be performed. The final tree is denoted by T_{max}. Since this tree almost certainly overfits the data, cost-complexity pruning is applied next (as described in the introduction of this chapter). This generates a nested sequence of minimizing subtrees, $T_{max} > \ldots > t_0$, where t_0 is the root node of the tree. From this sequence, the best monotone subtree is selected on the basis of its validation set performance (explained below). The monotonicity of a tree is checked by using Algorithm 3.1 presented in the introduction of this chapter; it is based on the comparison of the minimum and maximum elements of the leaf nodes. Finally, the generalization (prediction) error of the chosen model is computed using a separate test set. The algorithm outline is given in Algorithm 3.2.

Depending on the task at hand, we check the performance of the constructed trees on the validation and test sets by using the two measures for prediction accuracy, namely misclassification error in classification trees or mean-squared error in regression trees.

Algorithm 3.2 Derivation of monotone decision trees

`Train_set` = Training data
`Validation_set` = Validation data
`Test_set` = Test data

T_{max} = a large tree built on `Train_set`
$Trseq$ = a nested sequence of minimizing subtrees ($T_{max} > \ldots > t_0$) built by applying cost-complexity pruning on T_{max}
$Bmtr$ = the best monotone tree from $Trseq$ selected on the basis of the `Validation_set` performance
Determine the generalization prediction accuracy of the final model by applying $Bmtr$ on `Test_set`

Modifications in Feelders's algorithm

Given our algorithm, we now present the modifications we made in the original approach developed by Feelders (2000)–to meet the objectives of our research study.

First, as we are interested only in monotone models, we select the *monotone tree* with the best prediction accuracy from the nested sequence of pruned subtrees. This is the main difference with the algorithms in Feelders (2000) and Breiman et al. (1984), where the choice of the best subtree is based only on the test (or validation) set performance irrespective of the type of the tree (monotone/non-monotone).

Furthermore, in our experiments with real data sets (Section 3.3.2) it turns out that the non-monotone trees generated by the tree construction algorithm have comparable performance but are much larger than monotone trees. This is demonstrated in Figure 3.1 (at the end of this section) where both the non-monotone and the monotone tree are derived from the Moscow house pricing data used as a real case study in this research. The finding is also supported by earlier studies for classification trees (Feelders, 2000; Potharst and Feelders, 2002).

In addition, for problems with monotonicity properties (e.g., house pricing), monotone models are easier to understand than their non-monotone counterparts because they agree with the decision makers' expertise. In other words, non-monotone models are much harder to interpret because they present inconsistent and less intuitive dependencies. Therefore, only monotone trees are selected in our tree-based algorithm applied to the present

case studies.

Second, as we consider here both types of prediction problems (classification and regression), we modify Feelders's algorithm, and we build not only classification trees but also regression trees. For this purpose, the splitting criterion we use in growing regression trees is the mean-squared error; we label the leaves with the average response value of the objects belonging to them. Thus, our algorithm allows us to make predictions in an intuitive way in typical monotone regression problems such as house pricing. Feelders (2000) also applies his algorithm to a pricing case study, but first he discretizes the continuous house price in a number of classes. We do not find such a discretization natural; as we discussed in the previous section, it may lead to loss of important information in the knowledge discovery process.

Third, in order to guarantee that each class is properly represented in the training, validation, and test sets, we apply a *stratification* procedure when we split the data during the construction process of classification trees. In this case, each class is represented in approximately the same proportion in the three subsets as in the full data set.

Besides these modifications, we also point out a possible limitation of Feelders's approach, which concerns the number of monotone trees derived. His method guarantees that at least one monotone tree is generated, namely the one-node (root) tree. Due to its simplistic nature, in general the root tree does not perform satisfactorily for new data. Therefore, it is not preferred in practice as a final decision model for making predictions. However, if the data used to build trees are non-monotone, then it is well possible that the root tree is the only monotone tree generated by the algorithm. As a result, the monotone models derived would have bad performance on new data. As a simple solution, we suggest first to make the data monotone by using the greedy algorithm for relabeling (see Section 2.4), and then to apply the modified version of Feelders's tree-based approach as described above. In the next section, we present two cases where we apply this strategy and compare the performance of the models obtained from both monotone and non-monotone data.

3.3.2 Real case studies

Based on the two case studies introduced in Section 2.5, we would like to test the hypothesis that the models derived from the modified data perform

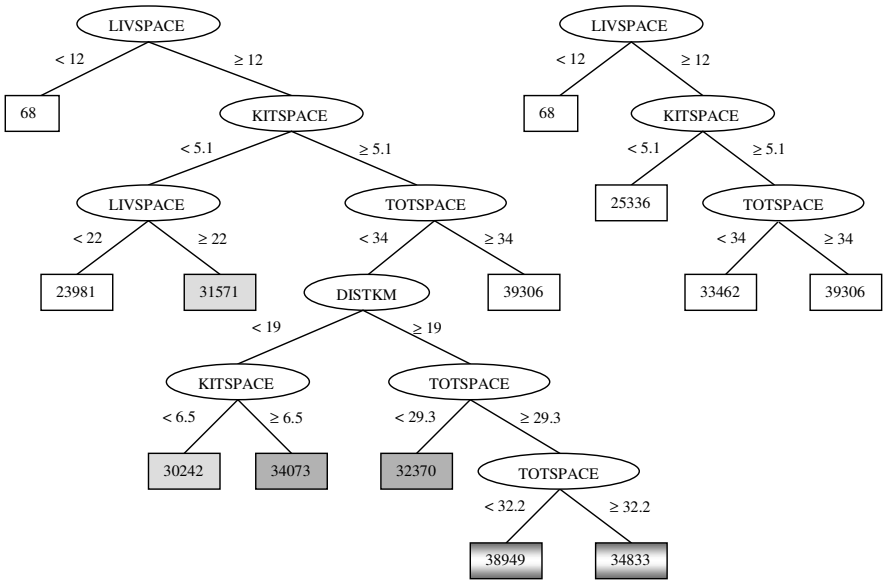


Figure 3.1: Non-monotone (left) and monotone trees for the original Moscow housing data. The shaded leaves represent the non-monotone leaf-pairs in the non-monotone tree. The estimated error of the non-monotone tree is 1.01 and the estimated error of the monotone tree is 1.02.

better than those derived from the original data. Therefore, we construct decision trees from both monotone and non-monotone data sets by using the tree-based algorithm described in the previous section and compare the models' performance measures.

To obtain a statistically sound assessment of our tree-based approach applied on both data sets, the following experiment is carried out 20 times. The original (non-monotone) data set is randomly partitioned into a construction set with 75% of the observations and a test set with 25% of the observations. The construction set is further randomly separated into a training set with 50% of the observations and a validation set with 25% of the observations. The training set is used to generate a tree of maximal size as explained in the introduction of this chapter, and to construct a sequence of subtrees using cost-complexity pruning. From this sequence of trees, the best monotone tree is selected on the basis of the prediction (misclassification or mean-squared) error computed on the validation set; in case of a tie, the smallest tree was chosen. The monotonicity of a tree is checked by using Algorithm 3.1. The random partition into training and validation sets is repeated five times, re-

sulting in a sequence of five trees, from which the one with the lowest error iss chosen as a final tree. In order to evaluate the performance of the final tree, the generalisation error is computed on the test set. The main steps of the experiment are depicted in Figure 3.2.

The same experiment is carried out with the cleaned (monotone) data. The only difference is the way the error is computed: instead of using a test set with 25% of the observations from the cleaned data, we compute the generalisation error on the basis of the same 25% observations from the original data, which are used as the test set in the previous experiment. Thus, the model is constructed from the cleaned data, whereas the performance is measured on the original data.

The prediction error and the size of a tree are popular performance measures for tree classifiers. Given our objective, we add one more measure, namely the number of monotone trees generated after pruning a large tree. The idea is that a higher number of monotone trees is preferred to guarantee that the decision model is monotone.

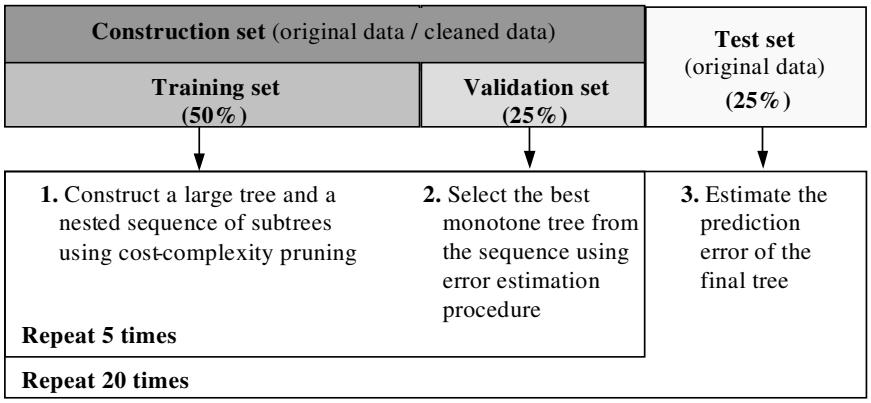


Figure 3.2: Main steps of the experiment conducted with the original and the cleaned real data sets

Software. The experiments with the extended algorithm for generating monotone decision trees were developed in S-PLUS, a software package for statistical and data analysis (see the S-PLUS web-site in the bibliography); the original program was also implemented in S-PLUS.

Below, we present a detailed description of our two case studies; bond rating data are used to construct monotone classification trees and Moscow house pricing data are used to construct monotone regression trees.

A. Bond rating

A summary of the results from the experiments with the bond rating data is given in Table 3.1.

Table 3.1: Experiments with monotone and non-monotone bond rating data based on decision trees

Indicators	Mean		Variance	
	Monotone data	Non-monotone data	Monotone data	Non-monotone data
Error on test set	0.50	0.53	0.003	0.003
Average number of monotone trees generated after pruning a large tree	5.1	4.7	0.195	0.431
Average number of leaves	8.2	7.5	1.958	5.526

To check the significance of the results we performed three t-tests. Since the test set in both experiments is the same, we apply the paired t-test of the null hypothesis that the trees derived from both data sets have the same classification error against the one-sided alternative (the trees derived from the monotone data have a smaller error than the trees derived from the non-monotone data). For the other two indicators (number of monotone trees generated after pruning a large tree, and number of leaf nodes) we use t-tests assuming unequal variances of the null hypotheses that the means are equal against the one-sided alternative hypotheses (the trees generated from the monotone data have larger indicator values than the trees generated from the non-monotone data). Table 3.1 suggests that the differences between the variances for these indicators are significant, which is also confirmed by the p-values of the two F-tests with 19 degrees of freedom, namely 4.6% and 1.4% (so these tests are significant at 5%). The p-value of the F-test for the difference in the errors is 33.4%, which indicates insignificant differences. Furthermore, the significantly lower variances of the number and the size of monotone trees generated from the monotone data show that the mono-

Table 3.2: p-values yielded of statistical t-tests and one-sided confidence intervals for the indicators in the bond rating case study

Indicators	p-value	Confidence intervals	
		95%	90%
Error on test set	6.9%	[-1, 0.003)	[-1, -0.004)
Average number of monotone trees generated after pruning a large tree	2.2%	(0.070, +∞)	(0.137, +∞)
Average number of leaves	13.1%	(-0.330, +∞)	(-0.101, +∞)

tone trees constructed from the cleaned data have less variability than that generated from the raw data.

The p-values obtained from the three t-tests and the respective one-sided confidence intervals at 95% and 90% levels are reported in Table 3.2.

The results show that the first null hypothesis (classification error of trees) can be rejected at a 10% significance level. Furthermore, the average number of monotone trees derived from the monotone data is significantly larger than that for non-monotone data.

B. Moscow house pricing

To generate decision trees from the original and the cleaned Moscow housing data, we apply the same tree-based algorithm and carry out the experiments described in the bond rating case study. A summary of the results is given in Table 3.3.

Table 3.3: Experiments with monotone and non-monotone Moscow housing data based on decision trees

Indicators	Mean		Variance	
	Monotone data	Non-monotone data	Monotone data	Non-monotone data
Error on test set	0.45	0.97	0.365	0.775
Average number of monotone trees generated after pruning a large tree	4.7	2.4	0.876	0.828
Average number of leaves	4.9	2.1	1.568	2.261

To check the significance of the results we again performed three t-tests (as we did for the bond rating case study). So, we use again a paired t-test for the prediction error. For the other two indicators (number of monotone trees generated after pruning a large tree and number of leaf nodes) we use t-tests assuming equal variances; the p-values of 45% and 22% for the two F-tests with 19 degrees of freedom indicate insignificant differences between the variances. The p-value of the F-test for the difference in the errors is 5.4%, which indicates insignificant differences at 5% significance level. The slightly, though not significantly, lower variances of the indicators in Table 3.3 show that the monotone trees constructed from the cleaned data tend to be more stable than that generated from the raw data. The p-values obtained from the three statistical t-tests are reported in Table 3.4.

The results show that the first null hypothesis (error of trees) can be rejected at the 5% significance level. Furthermore, the average number of monotone trees derived from the monotone data is significantly larger than that for the non-monotone data. Actually (not shown in the table) in 45% of cases, the only monotone tree generated by the non-monotone data is the root, which explains the smaller number of leaf nodes in the trees generated by the raw data.

Table 3.4: p-values yielded of statistical t-tests and one-sided confidence intervals for the indicators in the Moscow case study

Indicators	**P-value**	**Confidence intervals**	
		95%	**90%**
Error on test set	0.0%	(-∞, -0.284)	(-∞, -0.340)
Average number of monotone trees generated after pruning a large tree	0.0%	(1.868, +∞)	(1.979, +∞)
Average number of leaves	0.0%	(2.115, +∞)	(2.281, +∞)

Discussion of results

Based on the results obtained from the two case studies we conducted in this chapter, we draw the following conclusions:

1. Our experiments with regression trees support the finding in Feelders (2000) and Potharst and Feelders (2002) for classification trees, namely

monotone trees perform comparably to non-monotone trees, but the former are considerably smaller and therefore easier to interpret by the human decision-makers.

2. The prediction error of the monotone trees generated from the monotone (cleaned) data is significantly smaller than the prediction error of the monotone trees generated from the non-monotone (original) data.

3. The cleaned data yield more monotone trees after pruning the initial large tree than the raw data. This is preferred in monotone problems where it is necessary to guarantee that the prediction model is monotone.

4. The size (measured by the number of leaves) of the monotone trees derived from the original data is smaller than the size of their counterparts derived from the cleaned data; the difference is between one and three leaves on average. This is mainly due to the fact that often the only monotone tree generated from the original (non-monotone) is the one-node (root) tree. This also explains the better performance of the monotone trees obtained from the cleaned data compared with the monotone trees obtained from the raw data.

In summary, for problems that have monotonicity properties in the domain (such as bond rating and house pricing), pre-processing the data by making them monotone leads to considerable improvement in the performance of the monotone models in terms of smaller errors on the test data, less variability, and a larger number of generated monotone models.

3.4 Conclusion

In this chapter, we discussed monotone decision trees as a method to build monotone models for prediction tasks in data mining. At the beginning of the chapter, we introduced the basic terms and concepts concerning monotone trees. We presented a general tree construction procedure that is based on standard algorithms such as CART and ID3. Then, we reviewed previous studies related to generating decision trees with monotonicity constraints, and we discussed their main advantages and disadvantages. Based

on this overview, we justified the choice of a particular approach developed by Feelders (2000) for deriving monotone classification trees, which we further extended for monotone regression problems. The implementation of the algorithm with our own modifications were described in the main part of this chapter. We conducted two case studies on bond rating (a classification problem) and house pricing (a regression problem) to derive monotone decision trees by using our extended version of the tree-based approach. We compared the performance of the monotone trees derived from both the monotone cleaned and non-monotone original data. The results confirm our second hypothesis stated in the introduction of this thesis (Section 1.4), namely monotone models obtained from the monotone (transformed) data outperform monotone models obtained from the original data. Besides the experiments we conducted with monotone models, we also compared the performance of monotone and non-monotone regression trees. Our results support the finding in Feelders (2000) and Potharst and Feelders (2002) for classification trees, namely monotone trees perform comparably to non-monotone trees, but the former are considerably smaller and therefore easier to interpret by the human decision-makers. This result confirms our first hypothesis that for monotone problems monotone models have superior predictive performance to non-monotone models.

Chapter 4

Monotone neural networks

As we mentioned in the introduction of this thesis, neural networks are another popular technique widely applied in data mining prediction problems. In this chapter, we consider *monotone neural networks* to build monotone prediction models. First, we introduce some theoretical concepts about the architecture and functionality of neural networks. Then, we discuss earlier studies related to the development of monotone neural networks. In the main part of this chapter, we consider two approaches for building monotone neural networks. The first approach is proposed by Kay and Ungar (2000), who argue that their type of two-layer neural networks can approximate any monotone function. This proposition, however, is valid only for functions with a one-dimensional input; it does not hold for multi-dimensional spaces, as we show through a counter-example with two inputs. The second approach is developed by Sill (1998). He constructs a special class of three-layer neural networks, and proves that this class can approximate arbitrarily well any monotone function with one or more inputs. Given the universal approximation properties of Sill's networks, we use them–with some modifications–to build monotone models for prediction in this study. To demonstrate the universal approximation capabilities of Sill's networks, we conduct simulation studies with artificial data. Finally, we apply Sill's class of monotone neural networks to build monotone models for prediction in the two case studies that we have presented in Chapter 2, and we draw conclusions about the performance of the models.

4.1 Introduction

A standard feed-forward neural network with a multi-layer architecture is represented as follows.

- One input layer with k nodes, each corresponding to an input variable, and one bias unit set to a constant value of 1.
- One or more hidden layer(s) with a set of $h + 1$ nodes.
- One output layer with one or more nodes.

In the literature, there is no consensus about the total number of layers when specifying the architecture of a particular neural network. For some researchers, the total number of layers includes the input, hidden, and output layers. For others, it represents only the hidden and output layers, whereas the input layer is considered as a link to the external world. We follow the second convention; for example, a three-layer neural network consists of two hidden layers and one output layer.

All the connections between the layers are weighted. Let w_{ij} denote the weight for the connection between input j and hidden unit i, and v_i the weight for the connection between hidden unit i and the output. Then, given an input $\mathbf{x}$, the functional form of the output $O_{\mathbf{x}}$ corresponding to a network with one hidden layer is represented by

$$O_{\mathbf{x}} = \sigma\left(\sum_{i=1}^{h} v_i \sigma\left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i\right) + \theta_0\right) \tag{4.1}$$

where θ_0, θ_i, $i = 1, \ldots, h$ are the bias terms, and σ is the activation function, which is usually taken to be the sigmoid function, $\sigma(u) = 1/(1 + e^{-u})$. For regression problems, the activation function applied at the output of network is usually linear. The class of networks in (4.1) can approximate any continuous function of k input variables on any compact subset of $\Re^k$ (Cybenko, 1989).

The wide and successful application of neural networks is due to their principal capability, namely *learning*. Analogously to the process in a human brain, learning in neural networks is achieved by adjusting the connection weights of the network. In prediction problems, the weight adjustment during the learning process aims at minimizing the difference (error) between

the target (value of the dependent variable in the data) and the network's output corresponding to a particular input. The actual presentation of input and target data is called *training*. A neural network learns by being trained. The presentation of the entire data set to the network is called an *epoch*.

There are several learning algorithms for training a multi-layer feed-forward neural network. The most popular algorithm, outlined already in the introduction of this thesis, is *error backpropagation* (Rumelhart et al., 1986). Because this algorithm has been discussed extensively in the literature, we give only the following brief formal description of its working scheme.

The error backpropagation algorithm is based on the repeated application of the following two passes:

1. *Forward pass*: the network is activated for one input, and the error between the given target and network's actual output is computed.

2. *Backward pass*: the network error is used to update the weights. Starting at the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each neuron; see (4.5) below.

This explains the name of the algorithm, "backwards propagation of the error".

Depending on the type of prediction problem, the network error is computed in different ways. In regression, we have a network with one output, which produces a real value. Then, given the data $D = \left(\mathbf{x}^n, \ell_{\mathbf{x}^n}\right)_{n=1}^N$, we aim to minimize the mean-squared error (MSE) defined by

$$E = \sum_{n=1}^{N} \left(\ell_{\mathbf{x}^n} - O_{\mathbf{x}^n}\right)^2, \tag{4.2}$$

where $O_{\mathbf{x}}^n$ is the network output for the input $\mathbf{x}^n$.

In classification, we typically have a network with a number of output nodes corresponding to the number of class categories. Then, as shown by Bishop (1997), the error function we typically try to minimize is the so-called *cross-entropy function* given by

$$E = -\sum_{n=1}^{N} \sum_{c=1}^{\ell_{max}} \ell_{\mathbf{x}^n}^c \ln\left(\frac{O_{\mathbf{x}^n}^c}{\ell_{\mathbf{x}^n}^c}\right). \tag{4.3}$$

Here the target $\ell^c_{\mathbf{x}^n}$ can be considered as the probability that input $\mathbf{x}^n$ belongs to class ℓ_c. More precisely, the target $\ell^c_{\mathbf{x}^n}$ is represented by a binary vector containing the value one for $c = \ell_{\mathbf{x}^n}$, and zero, otherwise. Hence, the network's output $O^c_{\mathbf{x}^n}$ must also be computed as a probability, which lies in the range (0, 1) and adds up to one. This is achieved by using the *softmax* activation function in the output layer; that is, for class c

$$O^c_{\mathbf{x}^n} = \frac{e^{\mathbf{w}_c \mathbf{x}^n + \theta_c}}{\sum_{c'=1}^{\ell_{max}} e^{\mathbf{w}_{c'} \mathbf{x}^n + \theta_{c'}}}. \tag{4.4}$$

This function is a soft version of the winner-takes-all activation model, which equals one for the largest output, and zero for all other outputs. Thus, the error function in (4.3) is non-negative, and reaches its global minimum when $O^c_{\mathbf{x}^n} = \ell^c_{\mathbf{x}^n}$, for all c and $\mathbf{x}^n$.

In the second step of the backpropagation algorithm, we propagate the error–computed by either (4.2) or (4.3)–in order to update the weights such that they reduce the error. As the derivation of the update rule for the weights has been extensively discussed in the literature (see, for example, Bishop (1997)), we present only the general form of the rule, that is, at step s

$$w^s = w^{s-1} - \eta \sum_{n=1}^{N} \nabla E|_{w^s}, \tag{4.5}$$

where $\nabla E|_{w^s}$ is the gradient (the set of partial derivatives) of the error function E in (4.2) or (4.3) with respect to the weights w^s. The update rule in (4.5) is known as *batch learning*, since the weights are updated after each epoch. In contrast, in *sequential learning* the weights are updated after the presentation of each input.

The parameter η in the update rule for the weights is called the *learning rate*, which determines the step size of the learning process. Finding the optimum value of η is not trivial. If η is relatively small, we expect to obtain the minimum error, but at the cost of very slow learning (i.e., long computation time). If η is large, the learning process is speeded-up, but at the risk of jumping over the minimum, which may result in a large error. Several procedures have been developed to overcome these difficulties (Bishop, 1997). In summary, the backpropagation algorithm tries to minimize the network's error through the negative gradient of E, evaluated at w^s at step s.

As we have already discussed in this thesis, our ultimate goal is to build prediction models with good generalization capability, i.e., the models should not be specialized for the training data only (overfit), and should have reasonable predictive accuracy for new data.

In the introduction of this thesis (p. 12), we mentioned that one of the main drawbacks of neural networks is their tendency to overfit the data, especially for small samples. This is caused by: (i) an excessive number of network parameters (layers, hidden neurons and weight connections); (ii) very large weights. Given these causes, there are two main approaches to remedy this problem: *model selection* and *regularization*.

The objective of *model selection* is to find the model with the appropriate number of network parameters for the particular problem at hand. One method–like in decision trees–is to apply a pruning procedure, i.e., start with a large network, and subsequently remove connections or neurons during the training procedure; the final model is selected on the basis of the lowest estimated prediction error.

Another more ad-hoc approach used for model selection in practice is simply to apply a set of networks with different numbers of parameters on the same data, and then compare their performance; again the model with the smallest prediction error is preferred. Of course, this procedure might be computationally expensive, and does not guarantee that the set of preselected networks will lead to a satisfactory model. This method might be more efficient if we have a priori knowledge, which can guide us to set the parameters of the different networks; for example, a number of natural clusters in the data may play the role of the number of hidden neurons (see Section 4.3.2).

Regularization methods are used for weight restriction in order to improve the generalization capabilities of the network. It is known that large weights lead to network mappings with high curvature, i.e., all the observations in the data are approximated exactly. Two methods are commonly used to prevent the weights from growing too large, and to smooth out the network's output: *stopped training* and *weight decay*.

The idea in *stopped training* is to terminate the training process before convergence is reached. This is done by either reducing the number of epochs or using independent test set to compute the prediction error. The first approach works in an ad-hoc manner, because it is not trivial to determine the appropriate number of epochs; therefore, it is not very applicable. The

second approach is more realistic: it stops training the network as soon as the prediction error on the test set starts increasing.

In *weight decay*, the error function E to be minimized is modified by adding a term to penalize large weights:

$$\tilde{E} = E + \lambda \sum_{ij} w_{ij}^2$$

where λ is the regularization parameter. In other words, minimising $\tilde{E}$ is a trade-off between the goodness of fit and the smoothness of the network mapping. An advantage is that the optimization problem is well defined; a disadvantage is that the additional parameter λ needs to be determined beforehand.

4.2 Related work

In the last few years, several researchers have developed methods for incorporating monotonicity constraints in neural networks.

A monotone unbiased model for two-class problems is presented by Archer and Wang (1993b). They use two neural networks, one generating an optimistic and one generating a pessimistic monotone frontiers. These frontiers determine areas where the classification of a new observation can be specified as "preferred", "questionable", or "not-preferred". The optimistic boundary lies in the "not-preferred" class, whereas the pessimistic boundary lies in the "preferred" class. Now if a new point is below (above) both the optimistic and pessimistic frontiers, its class would be determined as not-preferred (preferred) with a high degree of certainty. However, if the point lies between the frontiers, then its classification is ambiguous and additional information might be used to take a decision. The authors show that their model has the effect of controlling the learning bias, because it provides more complete information than a single neural network.

Archer and Wang (1993a) and Lory and Gietl (2000) present two other neural network approaches that deal with monotonicity constraints for two-group classification problems. Lory and Gietl (2000) study so-called learning vector quantization type of networks where monotonicity is imposed by an appropriate modification of the Euclidean distance between the so-called codebook and input vectors. Archer and Wang (1993a) suggest a monotone

function (MF) model, which is a modification of the backpropagation learning algorithm. First, they pre-process the training data sample by using a linear classification function with monotonicity constraints in order to obtain monotone data. Then, the transformed data are used to train the network. To guarantee that the final model is monotone, the authors constrain the weights to be non-negative during the training process. In simulation studies with artificially generated data, the authors demonstrate that their monotone neural network approach outperforms linear discriminant analysis in terms of lower misclassification rates.

Wang (1994) introduces another approach that enforces monotonicity constraints on the network's weights during the training process. His approach employs a neural-network curve-fitting model that produces density estimation for univariate unimodal data. The topology of the network is a standard backpropagation neural network with one input layer consisting of one input node and one bias node, one hidden layer, and one output. As Wang notes, univariate unimodal cumulative density functions (CDF) are S-shaped, i.e, they are monotone increasing, and concave upwards and downwards on both sides of the mode point. To assure monotonicity, he enforces the derivative of the network output to be non-decreasing. In addition, he requires the second (partial) derivatives to be non-increasing/non-decreasing in order to guarantee that the estimated function is concave downward/upward. Wang demonstrates the application of his approach on real data, which contain measurements of iris sepal width for 150 plants; it is known from previous studies that these data are unimodal. Wang's results show that his approach provides better fit than the traditional density method, because he does not require any a priori assumptions for the functional form and he preserves the unimodality of the data. Wang points out that–like any other approach–his method for density estimation based on a neural network model has the following disadvantages: (i) it is sensitive to the true estimation of the mode in the data, and (ii) the weights in the neural network do not reveal much information about the relationship between the input and the output of the model.

Daniels and Kamp (1999) propose two approaches for the training of neural networks that are monotone by construction; these approaches are modifications of the standard backpropagation algorithm. The increasing monotonicity of the networks in both approaches is guaranteed by enforcing positive weights. The first algorithm sets all negative weights equal to zero

during each training step. The second algorithm modifies the standard error function E by adding a bias term as a penalty for negative weights:

$$E_m = E + \lambda \sum_{ij} (|w_{ij}| - w_{ij})$$

During the training phase, the parameter λ gradually increases until all the weights are non-negative.

Daniels and Kamp apply the first type of monotone neural networks to a house pricing case study, and compare its performance with the performance of ordinary neural networks. As expected, the results show that the monotone networks have better generalization capabilities, if the problem is largely monotone.

Kay and Ungar (2000) present another method with monotonicity constraints enforced through the signs of the weights. They call their method a Monotonic Semi-QUantitative system IDentification method (MSQUID). They aim at estimating monotone functions based on a two-layer backpropagation neural network with non-negative weights. They prove for univariate functions that such a type of neural network can approximate any continuous monotone function. This proposition, however, is not valid for multivariate functions, as we show by a counter-example in Section 4.3.1. In addition, they argue that the estimate computed by the monotone network will not be close to the target, because it is affected by the finite size of the data sample. Therefore, they extend their MSQUID by computing bounding envelopes for all possible functions that could generate the data under study with a certain probability. This is done by first linearizing the functional form of the network's output, and then using an F-statistic to compute the confidence interval of the network's estimate. When using knowledge about the monotone nature of a function, the confidence intervals obtained are smaller and thus the prediction accuracy is better. Kay and Ungar demonstrate the application of MSQUID by a real study for predicting the amount of inflow (dependent variable) in a watershed on the basis of the water level (independent variable) obtained from different streams; the two variables are known to have a monotone relationship. The results show that MSQUID provides a good fit to the data.

Sill and Abu-Mostafa (1997) also consider the incorporation of prior information about the monotone nature of the target function in neural network algorithms, which they call learning from monotonicity hints. Similarly to

Daniels and Kamp (1999), Sill and Abu-Mostafa modify the standard error function by adding a penalty term for deviation from monotonicity. For a candidate function f, they define so-called *monotonicity error*, which is a scalar measure for the degree to which f obeys monotonicity, given a set of input variables. They apply their method to two real cases: credit card applications (classification problem), and medical analysis (regression problem). The results show that using monotonicity hints can significantly improve the performance of a neural network compared with linear models and standard neural networks.

Although the addition of monotonicity error enforces monotonicity on the network and leads to improvement in the network's performance, it does not guarantee that the final model is totally monotone. Furthermore, this approach can also be computationally expensive, because it requires the optimization of a more complex function. To overcome these drawbacks, Sill (1998) develops another type of neural networks with monotonicity preserved by virtue of construction. The structure of these networks is similar to the structure of the so-called *adaptive logic networks*, introduced by Armstrong and Thomas (1997).

The adaptive logic network is a feed-forward multilayer network that uses linear functions in the first hidden layer, and originally used the logic operators AND and OR (they explain the name "logic") in the other hidden layers. Obviously, such a network produces Boolean output. Later on the logic operators were replaced by MAX and MIN functions, which allowed real-valued functions to be computed as well. Due to their architecture, adaptive logic networks have several advantages: (i) computation of the output is simple and fast, due to the limited number of linear unit calculations and simple comparison operators performed. In addition, at each iteration of the training process only the weights of a single linear unit (the active one) are modified, which speeds up network's learning; (ii) by constraining the coefficients of the linear units it is easy to incorporate domain knowledge. Therefore, monotonicity can be easily imposed by restricting the coefficients to be positive or negative; (iii) given an input, the network's output is easy to understand and interpret by the end user as the parameters of the linear units directly reflect the relationships in the data. These networks have been successfully applied in many fields, for example, prediction, data analysis, control problems, robotics, and optimization of communication networks. Yet, their main disadvantage is that they remain accessible commercially

only through "Dendronic Decisions Limited" (see Armstrong (1974)), and are not fully disclosed in an academic setting, which limits their use for research purposes.

Sill (1998) proposes an alternative approach. Using an architecture and properties similar to the adaptive logic networks, he develops a three-layer neural network with monotonicity constraints. In addition, he proves that his type of network has universal approximation capabilities, and can outperform linear models and standard neural networks in real-world problems.

The theoretical and practical advantages of Sill's monotone networks make them attractive for application in monotone prediction problems. Therefore, we apply his type of network to build monotone models. We refer to it as a *Sill network* in the remainder of the thesis. In Section 4.3.2, we formalize the architecture and functionality of Sill networks.

4.3 Algorithms for building monotone neural networks

4.3.1 Two-layer monotone networks

Suppose we have a monotone problem with the dependent variable ℓ being monotone in all independent variables and we want to predict ℓ by using a neural network as defined in (4.1). To preserve the property of ℓ being monotonically increasing in an input x_j, the partial derivative of the output $O_{\mathbf{x}}$ with respect to x_j must be enforced to be non-negative:

$$\frac{\partial O_{\mathbf{x}}}{\partial x_j} = \sigma' \cdot \sum_{i=1}^{h} v_i \sigma' \left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i \right) \cdot w_{ij} \geq 0. \tag{4.6}$$

Because $\sigma' > 0$, (4.6) holds if and only if

$$\sum_{i=1}^{h} v_i \sigma' \left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i \right) \cdot w_{ij} \geq 0.$$

As shown by Kay and Ungar (2000), this condition is equivalent to the constraint

$$\forall_{1\leq i\leq h}\ v_i \cdot w_{ij} \geq 0, \qquad \text{for each input } j,\ j = 1, \ldots, k.$$

In case of decreasing monotonicity, the inequality is reversed. Kay and Ungar prove for the one-dimensional case that a neural network constrained in this manner can approximate any continuous monotone function. This proposition, however, does not hold for multivariate functions. This is illustrated by the counter-example for two dimensions presented on p. 110.

First we show that the class of networks defined in (4.1) has an analogue (see Proposition 4.3.1), which is used in the following counter-example. For simplicity and without loss of generality, the networks considered here do not include a sigmoid function at the final network's output.

Proposition 4.3.1. *The following two classes of neural networks coincide:*

Class-1 *(considered by Kay and Ungar (2000)):*

$$O_{\mathbf{x}} = \sum_{i=1}^{h} v_i \sigma \left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i \right) + \theta_0$$

subject to

$$\forall_{1\leq i\leq h}\ v_i \cdot w_{ij} \geq 0, \qquad \textit{for each input } j,\ j = 1, \ldots, k.$$

Class-2*:*

$$O_{\mathbf{x}} = C + \sum_{i=1}^{h} v_i \sigma \left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i \right) + \theta_0$$

where C is a constant, and

$$\forall_{1\leq i\leq h}\ v_i \geq 0,\ \forall_{1\leq j\leq k}\ w_{ij} \geq 0.$$

Proof. The proof follows from

$$1 - \varphi(x) = -\varphi(-x), \qquad \text{with } \varphi(x) = 1/(1 + \exp^{-x}).$$

The same proposition also holds for ψ, with

$$\psi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

but without the constant C, because $\psi(x) = -\psi(-x)$. □

Counter-example for the approximation capabilities of two-layer monotone neural networks

Consider the Class-2 functions as defined in Proposition 4.3.1. Class-2 corresponds to a neural network with one output, one hidden layer with sigmoid activation function $\sigma(x) = 1/(1 + \exp^{-x})$ and parameters v, w, and θ. The input vector is denoted by $\mathbf{x} = (x_1, x_2 \ldots, x_k)$.

To show that Class-2 cannot approximate every monotone continuous function–except for the one-dimensional case ($k = 1$)–we give a counter-example with $k = 2$ (a counter-example in a slightly different context was originally communicated to us by Dr. A.J.E.M. Janssen of Philips Research Laboratories, Eindhoven, The Netherlands).

Let $\mathcal{X} = \{x_1, x_2\}$ be the two-dimensional input space with values on $[0, 1]^2$, and let $f : \mathcal{X} \to [0, 1]$ be a function defined by three regions: $f_\epsilon = 0$ in region R_1, $f_\epsilon = 1$ in region R_2, and $0 \leq f_\epsilon \leq 1$ in region R_3, for $\epsilon \in (0, 1)$. To determine the boundaries for the three regions, we use the following four linear functions:

$$\begin{array}{rcl} a_1 & = & 3x_1 + x_2 - 1 + \epsilon \\ a_2 & = & x_1 + 3x_2 - 1 + \epsilon \\ a_3 & = & 3x_1 + x_2 - 1 - \epsilon \\ a_4 & = & x_1 + 3x_2 - 1 - \epsilon. \end{array} \tag{4.7}$$

Next, we define three regions in the input space:

$$\begin{array}{ll} R_1: & \big\{(x_1, x_2) \,|\, a_1 < 0 \text{ and } x_1 < x_2\big\} \cup \\ & \big\{(x_1, x_2) \,|\, a_2 < 0 \text{ and } x_1 \geq x_2\big\} \\ R_2: & \big\{(x_1, x_2) \,|\, a_3 > 0 \text{ and } x_1 < x_2\big\} \cup \\ & \big\{(x_1, x_2) \,|\, a_4 > 0 \text{ and } x_1 \geq x_2\big\} \\ R_3: & \mathcal{X} - R_1 \cup R_2, \end{array} \tag{4.8}$$

and define f_ϵ by:

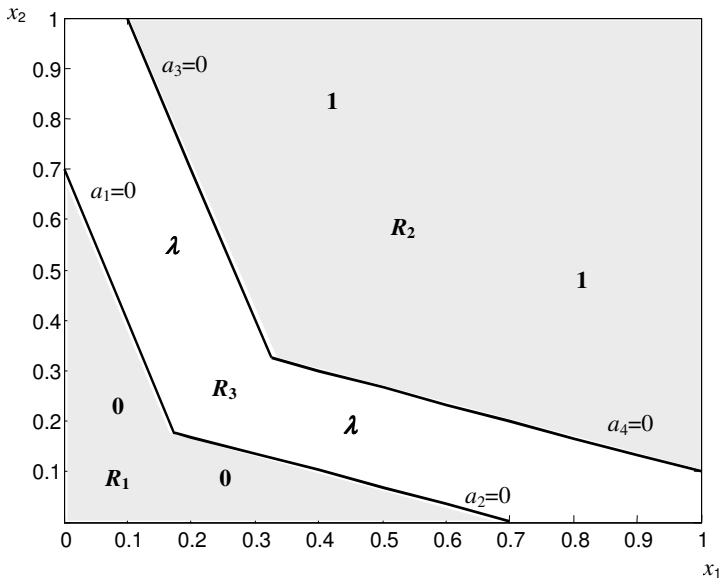


Figure 4.1: Graphical representation of f_ϵ defined in (4.9)

$$f_\epsilon = \begin{cases} 0 & \text{on } R_1 \\ 1 & \text{on } R_2 \\ \lambda & \text{on } R_3, \end{cases} \tag{4.9}$$

where

$$\lambda = \begin{cases} \frac{3x_1+x_2-1+\epsilon}{2\epsilon} & \text{if } x_1 < x_2, \\ \frac{x_1+3x_2-1+\epsilon}{2\epsilon} & \text{otherwise.} \end{cases}$$

The graphical representation of f_ϵ is given in Figure 4.1.

By taking ϵ small enough we can find points A, B, and C on a straight line with $f_\epsilon(A) = 0, f_\epsilon(B) = 1$ and $f_\epsilon(C) = 0$; see Figure 4.2.

Now suppose we have a neural-network approximation $\hat{f}$ of f_ϵ of the form

$$\hat{f}(\mathbf{x}) = C + \sum_{i=1}^{h} v_i \sigma \left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i \right) + \theta_0 \tag{4.10}$$

with $\forall_{1 \le i \le h}\ v_i \ge 0,\ \forall_{1 \le j \le k}\ w_{ij} \ge 0$, and

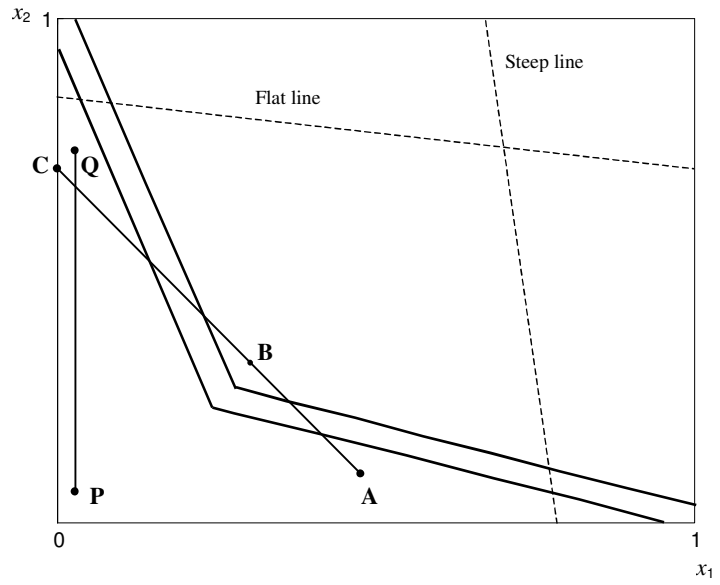


Figure 4.2: Graphical representation of the lines $[A, B, C]$ and $[P, Q]$.

$$\max_{\mathbf{x}\in[0,1]^2} \left| f_\epsilon(\mathbf{x}) - \hat{f}(\mathbf{x}) \right| < \frac{1}{8}. \tag{4.11}$$

We will show that this leads to a contradiction if ϵ is small enough. To do so, we show that the increase in f_ϵ from A to B implies that the neural-network approximation $\hat{f}$ also increases from A to B. However, this causes at least the same increase in $\hat{f}$ from P to Q. Given that $f_\epsilon(P) = f_\epsilon(Q) = 0$, we have a contradiction with the fact that the neural-network approximation $\hat{f}$ is close to the true function f_ϵ in terms of (4.11).

First, consider the term i in (4.10)

$$\hat{f}_i = v_i \sigma \left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i \right).$$

The contour lines of $\hat{f}_i$ are straight lines with normal vector (w_{i1}, w_{i2}) in the positive quadrant since $w_{i1} \geq 0$ and $w_{i2} \geq 0$. So points for which $\hat{f}_i$ is constant must lie on a straight line.

We now split the sum of $\hat{f}$ in (4.10) into the following three terms:

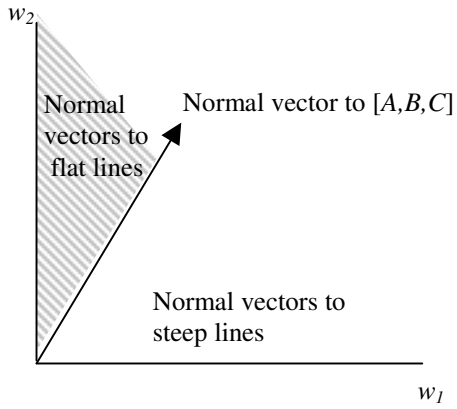


Figure 4.3: Normal vectors to $[A, B, C]$, flat and steep lines.

$$\hat{f} = C + \sum_{i \in Flat} \hat{f}_i + \sum_{i \in Steep} \hat{f}_i,$$

where $Flat$ corresponds to the lines that are "flatter" than or as flat as the line $[A, B, C]$, i.e., the vector (w_{i1}, w_{i2}) perpendicular to the lines is in the shaded area in Figure 4.3. Similarly, $Steep$ corresponds to lines that are "steeper" than or as steep as the line $[A, B, C]$.

The increase in $\hat{f}$ in (4.10) when moving from A to B can be caused by flat lines only. So

$$\sum_{i \in Flat} \hat{f}_i(B) - \sum_{i \in Flat} \hat{f}_i(A) \geq \frac{6}{8},$$

because $f_\epsilon(A) = 0$, $f_\epsilon(B) = 1$ and $|f_\epsilon - \hat{f}| < 1/8$ everywhere.

Now consider the line $[P, Q]$ in Figure 4.2, which is parallel to the x_2-axis. Then

$$\sum_{i \in Flat} \hat{f}_i(Q) - \sum_{i \in Flat} \hat{f}_i(P) \geq \frac{6}{8},$$

since P is below all flat lines that cross the line $[A, B]$ somewhere, and Q is above all lines that cross the line $[A, B]$ somewhere. Hence

$$\begin{aligned}\hat{f}(Q) - \hat{f}(P) = &\sum_{i \in Flat} \hat{f}_i(Q) - \sum_{i \in Flat} \hat{f}_i(P) \\ &+ \sum_{i \in Steep} \hat{f}_i(Q) - \sum_{i \in Steep} \hat{f}_i(P) \\ &\geq \frac{6}{8},\end{aligned}$$

because

$$\sum_{i \in Steep} \hat{f}_i(Q) - \sum_{i \in Steep} \hat{f}_i(P) \geq 0.$$

Note that all terms in $\hat{f}_i$ are non-decreasing on the line $[P, Q]$. However, $f_\epsilon(P) = f_\epsilon(Q) = 0$ and $|f_\epsilon - \hat{f}| > 1/8$, contradicting (4.11).

4.3.2 Three-layer Sill monotone networks

As the counter-example in the previous section proved, two-layer monotone networks cannot approximate all continuous monotone functions.

Therefore, we now consider another class of monotone neural networks, proposed by Sill (1998); these networks prove to have universal approximation capabilities. Below we provide a detailed description of their architecture and training algorithm.

A Sill network has a three-layer architecture (with two hidden layers). Figure 4.4 gives an example of Sill network's architecture. The input layer is connected to the first hidden layer consisting of a set of linear units (hyperplanes), which are combined into several groups (the number of units in each group is not necessarily the same). Corresponding to each group is a second hidden-layer unit, which computes the maximum over all first-layer units within the group. The final output unit computes the minimum over all groups.

In formal notation, a Sill network can be represented as follows. Let R denote the number of nodes in the second hidden layer; that is, the number of groups in the first hidden layer, with outputs $g_1, g_2, \ldots, g_R$. Let h_r denote the number of hyperplanes within group $r, r = 1, 2, \ldots, R$. The parameters (weights) of the hyperplanes in r are k-dimensional vectors denoted by $\mathbf{w}_{(r,1)}$, $\mathbf{w}_{(r,2)}$, ..., $\mathbf{w}_{(r,h_r)}$; the matrix of all weights and biases is denoted by W.

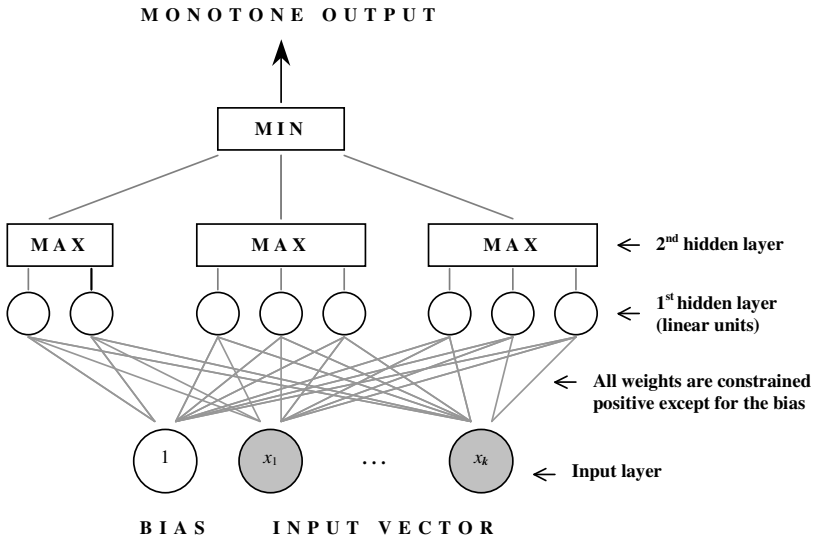


Figure 4.4: An example of Sill network's architecture.

Then, the output at group r is defined by:

$$g_r(\mathbf{x}) = \max_j \left(\mathbf{w}_{(r,j)} \cdot \mathbf{x} + \theta_{(r,j)} \right), \quad 1 \leq j \leq h_r, \tag{4.12}$$

where θ is a bias term.

The final output of the network is given by

$$O_{\mathbf{x}} = \min_r g_r(\mathbf{x}), \tag{4.13}$$

or in classification problems

$$O_{\mathbf{x}} = \min_r \sigma(g_r(\mathbf{x})), \tag{4.14}$$

where σ is the sigmoid function.

From (4.13) and (4.14), it follows that one group and one hyperplane within this group uniquely determine the output of the network for each input vector. Such group and hyperplane are called *active*. In case of ties in the group or network outputs (though this is unlikely, because the outputs are continuous), the choice of the active hyperplane or group is made randomly.

To guarantee that the network output is monotone, all weights for an input to the first hidden layer are constrained to be non-negative (non-

positive), if increasing (decreasing) monotonicity is desired for that input. Here, we enforce the parameters in (4.12) to be non-negative by taking an appropriate transformation such as $w = z^2$, where z is a free parameter.

As proven by Sill (1998), a Sill network is capable of approximating any continuous monotonic function arbitrarily well, given sufficiently many groups and hyperplanes within each group. For completeness, we present the theorem for the universal approximation capability of Sill networks.

Theorem 4.3.1 (Sill, 1998). *Let $m(\mathbf{x})$ be any continuous bounded monotone function with bounded partial derivatives, mapping $[0,1]^k$ to $\Re$. Then, for any $\epsilon > 0$ there exists a function $mnet(\mathbf{x})$ which can be implemented by a monotone network and is such that $|m(\mathbf{x}) - mnet(\mathbf{x})| < \epsilon$, for any $\mathbf{x} \in [0,1]^k$.*

Due to their architecture, Sill networks can approximate convex surfaces by the maximum operator, and concave surfaces by the minimum operator; see Figure 4.5 (the solid lines represent the approximation). The combination of both minimum and maximum operators also enables the Sill networks to approximate any type of monotone function, for example, functions that are neither convex nor concave (see Experiment 1, p. 122).

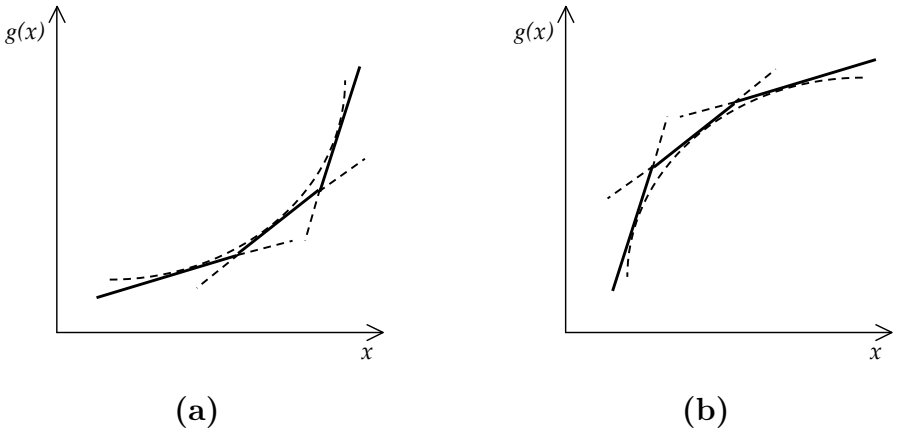


Figure 4.5: Approximation of (a) a convex function by the maximum operator, and (b) a concave function by the minimum operator in a Sill network

Training algorithm

Given the architecture of a Sill network, the training algorithm is easily implemented. We start with W_0, the initial values for W, the matrix of

parameters (weights and biases), which are obtained from the initialization procedure described below. To guarantee that the trained network does not overfit the data, we first randomly partition the whole data into a construction set and a test set. Next, we repeat the following procedure five times. We randomly split the construction set into a training set and a validation set. We run an iterative optimization technique, such as Quasi-Newton line search, on the training set to find the parameters of the network that minimize the prediction error. At each iteration, for all input vectors $\mathbf{x}^n$, $n = 1, \ldots, N$, the network's outputs $O_{\mathbf{x}^n}$ are computed and the network's error E is calculated over all data points. The update of the parameters is done in batch mode, i.e., after each pass of the training data. The iterative process terminates when either convergence is reached or some other stopping criterion is satisfied (e.g., the maximal number of iterations or function evaluations is reached). By using the parameters of the trained network, we compute the prediction error on the validation set. From the five trained networks we select the network with the lowest prediction error on the validation set. The generalization error of the final model is computed on the test set. The training algorithm outline is given in Algorithm 4.1.

Modifications in the architecture and the training algorithm of Sill networks

As we mentioned in the description of a Sill network architecture (p. 116), the network's output is guaranteed to be monotone by enforcing the weights to be non-negative. We take $w = z^2$, which is our simple modification of the Sill network's architecture. In the original approach, Sill guarantees monotonicity by applying an exponential transformation on the weights, namely $w = e^z$. However, this exponential function cannot give a zero value, so it does not allow the weights to be zero. Hence, it is impossible to approximate flat (constant) functions by using the original Sill's approach.

Our second modification concerns the architecture of Sill networks for classification problems. As we discussed in the introduction of this chapter, if the predicted variable is discrete, then the output layer of neural networks usually consists of a number of nodes corresponding to the number of class categories ℓ_{max}. Sill (1998) applies a similar approach in the prediction of a company's bond rating, using his three-layer monotone networks. He uses

Algorithm 4.1 Training of Sill networks

Initialization:
`Construction_set` = Network construction data
`Test_set` = Test data
R= number of groups
h_r= number of hyperplanes in group r
E^*_{val}=∞

Training:
for $i := 1$ to 5 **do**
  `Train_set` = Training data $\subset$ `Construction_set`
  `Validation_set` = `Construction_set` - `Train_set`
  W_0= $R \times h_r \times (k+1)$ matrix of initial network's parameters obtained from the initialization procedure
  W= $\min_W Error$(`Train_set`,W_0,R,h_r) (minimization by Quasi-Newton line search)
  E_{val}=$Error$(`Validation_set`,W)
  if $E^*_{val} > E_{val}$ **then**
    $E^*_{val} = E_{val}$
    $W^* = W$
    $Bmnet$ = SillNet(W^*,R,h_r)
  end if
end for

Determine the generalization prediction accuracy of the final model by applying $Bmnet$ on `Test_set`

a set of networks represented by the same number of input nodes, groups and hyperplanes, and one output corresponding to a particular bond rating (i.e., class category); the only difference between the networks in the set are the weights and the biases assigned to the connections. In this case, given an input, the predicted class is given by the Sill network with the maximum output among the networks in the set.

In our study, however, we take a slightly different approach. The main difference is that we train only one network with one output as defined in (4.13). Furthermore, we present to the network the original discrete labels as targets without applying any transformation to them (e.g., making the labels continuous within the range [0,1]). The idea is that each discrete label ℓ can be considered as the middle of an interval of size one. In other words,

any value ℓ' that is $|\ell' - \ell| < 1/2$ lies on the same interval and $\ell' = \ell$; otherwise, $\ell' \neq \ell$. Hence, the network's error for point $\mathbf{x}^n$ is defined by

if $|\ell_{\mathbf{x}^n} - O_{\mathbf{x}^n}| < 1/2$ **then**
  $E_{\mathbf{x}^n} = [2(\ell_{\mathbf{x}^n} - O_{\mathbf{x}^n})]^4$
else if $|\ell_{\mathbf{x}^n} - O_{\mathbf{x}^n}| \geq \ell_{max}$ **then**
  $E_{\mathbf{x}^n} = \ell_{max}$
else
  $E_{\mathbf{x}^n} = |\ell_{\mathbf{x}^n} - O_{\mathbf{x}^n}|$.
end if

Then the total network's error we try to minimize after one epoch is:

$$E = \sum_{n=1}^{N} E_{\mathbf{x}^n}. \tag{4.15}$$

Note that in the first case when the network approximation $O_{\mathbf{x}^n}$ is within the interval of $\ell_{\mathbf{x}^n}$, i.e., we have correct classification, we still add a penalty term. If $O_{\mathbf{x}^n}$ is close to the middle of the interval ($\ell_{\mathbf{x}^n}$), then the penalty is negligibly small due to the exponent of four. If $O_{\mathbf{x}^n}$ approaches, however, one of the ends of the interval of the true label, then the penalty approaches one. In this way, we enforce the network's output to be close to the target label and thus we stabilize the network's training process. In the second case for the error function when the difference between the true and predicted labels is larger than or equal to the number of class categories ℓ_{max}, we add as a penalty ℓ_{max} only. Thus, we prevent the network of overfitting outliers. In the final case for the error function, we have a misclassified point within the range of class categories and the penalty is the absolute difference between the true and predicted labels.

Our network's output representation has two advantages over Sill's original approach: (i) a single network is trained, which speeds up the learning process; (ii) it requires less storage and update (memory) capacity; if the target has a large number of class categories, and if the network has many groups and hyperplanes, then Sill's original approach would be considerably more expensive than our method.

Like we did for decision trees in Chapter 3, we extend the application of Sill networks to regression problems. This is simply done by using the standard MSE as a measure for the quality of the prediction.

We also modify the training algorithm of Sill networks, i.e., we apply the Quasi-Newton method with line search to minimize the network's error instead of the gradient descent originally proposed by Sill. Note that from a theoretical point of view neither the gradient descent nor the Quasi-Newton optimization technique is suitable for the problem at hand as the functions in (4.13) and (4.14) are non-differentiable. For this type of problems, the simplex search method proposed by Nelder and Mead (1965) is more appropriate technique as it uses only function evaluations. However, in the practical applications presented in this thesis the Quasi-Newton method with the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) update of the Hessian matrix (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) appears to comparably perform to the simplex method but the former is much faster. Therefore, we finally use the Quasi-Newton method in the training algorithm to minimize the network's error.

Finally, we apply a different initialization procedure to set the weights and biases of the network. In his training algorithm, Sill uses the parameters obtained from the linear model fitted to the whole data set, and add a small random perturbation to them in order to initialize the parameters of the hyperplanes in each group. Our initialization procedure is described below.

Initialization of the network's parameters

In practice, the initial values for the network's weights are usually taken at random. Then, however, the network's solutions may differ considerably each time the network is trained. In contrast, appropriate weight initialization may make it more robust and thus, improve the generalization capabilities of the network, and speed up the learning process. Therefore, in this study we apply an initialization procedure for the network's weights, which is based on the training data. Our main objective is to ensure that the training algorithm starts from a reasonable solution. To achieve such a start, we first partition the set of input variable values into a number of clusters (subsets) corresponding to the number of groups in the network. Then, we find the parameters of the linear model that best fits to the data belonging to each cluster. Finally, by adding a small random perturbation to the linear parameters, we initialize the parameters (weights and biases) for each hyperplane in each group.

More formally, we first apply the K-means clustering method (see Duda

and Hart (1973) for details) on the explanatory variables to partition the data into R clusters (subsets), where R corresponds to the number of groups in the network. In this way, we obtain groups of objects with similar attribute values. For each cluster r ($r = 1,\ldots, R$) we find the parameters of the linear model that best fits the data belonging to a particular cluster. To illustrate, let us consider cluster r. The linear model obtained from this cluster has the form:

$$f_r(\mathbf{x}) = \boldsymbol{\beta}_r \cdot \mathbf{x} + \theta_r, \tag{4.16}$$

where $\boldsymbol{\beta}$ is a k-dimensional vector of parameters, θ is a scalar, and $\mathbf{x} = (x_1, \ldots, x_k)$.

Now we can also consider the model in (4.16) as a hyperplane from the network with parameters $|\boldsymbol{\beta}_r| = \mathbf{w}_r = \mathbf{z}_r^2$ and a bias term θ_r. Hence, the initial parameters for each hyperplane j ($j = 1, 2, \ldots h_r$) in cluster r, are obtained by

$$\theta_{(r,j)} = \theta_r + \tau$$

and

$$\mathbf{z}_{(r,j)} = \sqrt{|\boldsymbol{\beta}_r|} + \tau$$

where τ is a small random perturbation, such as $0.01\mathcal{N}(0, 1)$.

Software. The implementation of our training algorithm of Sill networks is done in MATLAB (see the MATLAB web-site in the bibliography). One of the main strengths of MATLAB is its ability to handle large matrices, and thus, to perform complex calculations extremely quickly. In addition, the large number of built-in functions for numerical optimization allow us to develop a fast and efficient algorithm for training Sill networks.

Simulation studies

In his case study on bond rating, Sill (1998) shows that his three-layer monotone networks perform better than linear models and standard neural networks. To better demonstrate the approximation capabilities of Sill networks, we now conduct two simulation studies with artificially generated data, and the results are reported. We emphasize that our main objective is

to check the extent to which Sill networks can approximate a data generating process with a given functional form, rather than to check the generalization prediction accuracy of the networks on new data. Therefore, the functions we use to generate the data in both studies are deterministic, i.e., the data are noise-free. Because the input data for both experiments are the same, we first describe their generating process.

Let x_1 and x_2 be vectors of equispaced N elements taking values on [0,1]; the interelement spacing is δ. Then the input space is defined as the Cartesian product of x_1 and x_2 on grid $[0,1]^2$, that is

$$\Big\{(x_1, x_2)\Big\} = \Big\{0, \delta, 2\delta, \ldots, 1\Big\}^2.$$

In our experiments, we take $\delta = 0.1$, i.e., the two-dimensional input space consists of 121 points.

Experiment 1. As noted in Section 4.3.2, the MAX and MIN operators in the architecture of Sill networks allow the networks to approximate monotone concave and convex functions. In this experimental study, we illustrate that Sill networks can also approximate monotone functions that are neither convex nor concave. Here, for $(x_1, x_2) \in [0,1]^2$ we define such type of function by

$$f(x_1, x_2) = 1 + x_1 + \frac{1}{2}\left(x_2^2 - x_1^2\right). \tag{4.17}$$

It is obvious that $f(x_1, x_2)$ is monotonically increasing in x_1 and x_2, but it is neither convex nor concave; see Figure 4.6.

Now, we aim to build a model that produces a good approximation of f. Given that the function is monotone, a natural solution is to apply Sill networks. However, to better assess the performance, we compare Sill networks to standard two-layer neural networks (in short, NNs).

Since $f(x_1, x_2)$ is noise-free, we use $D = (x_1, x_2, f(x_1, x_2))_{n=1}^{121}$ both as a training set and a test set for the application of Sill networks and NNs. In this case, the standard MSE is going to be very small for both types of networks, so it is difficult to compare them. Therefore, we use an additional criterion, namely the average percentage error ($AvrPE$) defined as

$$AvrPE = 100 \sum_{n=1}^{N} \frac{|\ell_{\mathbf{x}^n} - O_{\mathbf{x}^n}|}{\ell_{\mathbf{x}^n}}.$$

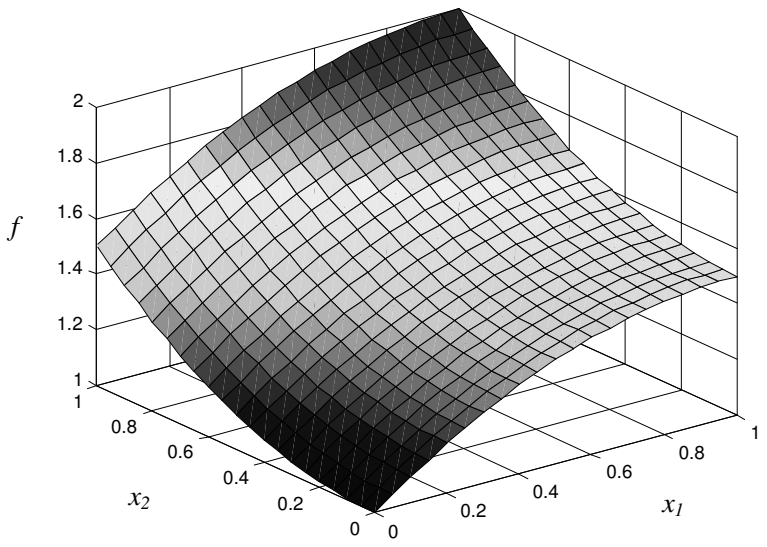


Figure 4.6: Graphical representation of the non-convex non-concave function in (4.17)

Finally, we use various configurations for the architecture of both network types. We repeat twenty times the application of the networks with each configuration and different initial weights and biases, and average the results. Tables 4.1 and 4.2 report the means and variances of the performance measures estimated from the experiments.

For a better illustration of the approximation capabilities of both network types, we provide graphical representations for their best solutions, in the sense of lowest error rates and variances, obtained for a particular network architecture; that is, for a Sill network with four groups and six hyperplanes in each group, and a standard neural network with two hidden nodes; see Figures 4.7–4.10.

The results clearly show that Sill networks outperform standard NNs. First, Sill networks obtain better prediction accuracy with low MSE and AvrPE. Furthermore, they produce solutions with less variability across different runs compared to standard NNs, as the variances of the errors show in Tables 4.1 and 4.2. This implies that Sill networks lead to similar solutions after each run as a result of our initialization procedure, whereas standard

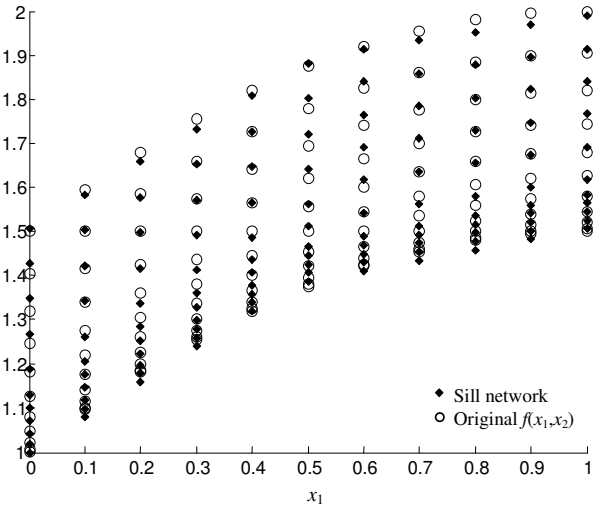


Figure 4.7: Sill network approximation with four groups and six hyperplanes of a non-convex non-concave function; MSE = 0.0002, $AvrPE = 0.80\%$ (plot against x_1)

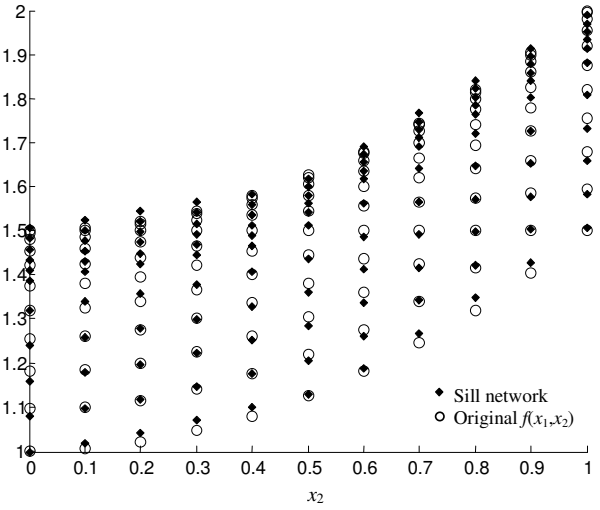


Figure 4.8: Sill network approximation with four groups and six hyperplanes of a non-convex non-concave function; MSE = 0.0002, $AvrPE = 0.80\%$ (plot against x_2)

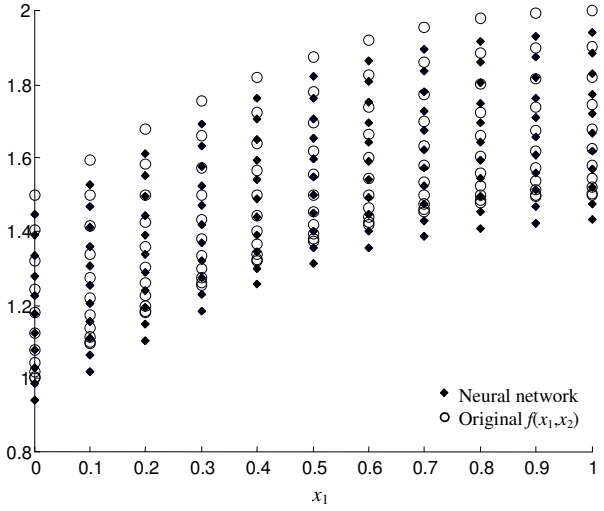


Figure 4.9: Standard neural network approximation with two hidden neurons of a non-convex non-concave function; MSE = 0.0014, $AvrPE = 2.22\%$ (plot against x_1)

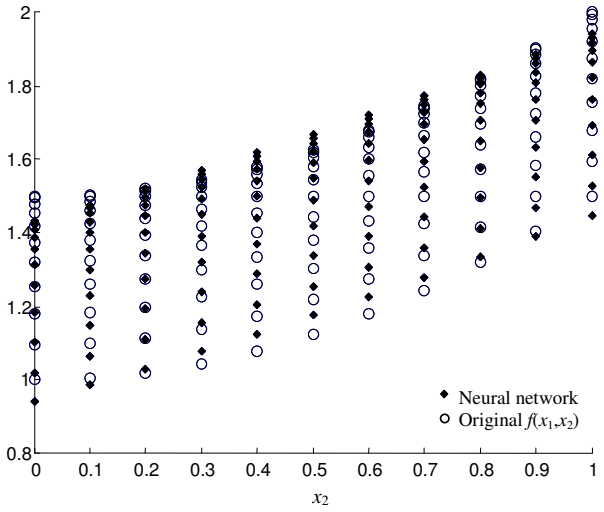


Figure 4.10: Standard neural network approximation with two hidden neurons of a non-convex non-concave function; MSE = 0.0014, $AvrPE = 2.22\%$ (plot against x_2)

Table 4.1: Application of Sill networks to the non-convex non-concave function in (4.17)

Sill network (groups × planes)	**MSE**	**var(MSE)**	**AvrPE (%)**	**var(AvrPE)**
2×2	0.0008	5.3e-07	1.39	0.53
2×4	0.0006	4.7e-07	1.18	0.48
2×6	0.0005	4.6e-07	1.10	0.47
4×2	0.0003	1.1e-09	0.91	0.00
4×4	0.0003	1.9e-09	0.89	0.00
4×6	0.0002	3.2e-09	0.85	0.01
6×2	0.0009	6.6e-07	1.51	0.64
6×4	0.0008	6.4e-07	1.38	0.64
6×6	0.0003	2.0e-07	0.91	0.20

Table 4.2: Application of standard neural networks to the non-convex non-concave function in (4.17)

NNs (hidden nodes)	**MSE**	**var(MSE)**	**AvrPE (%)**	**var(AvrPE)**
2	0.0020	3.7e-06	2.04	2.03
4	0.0021	3.7e-06	2.02	2.30
6	0.0021	3.3e-06	2.08	2.08
8	0.0024	3.1e-06	2.38	1.77

NNs are more sensitive to the initial starting network's parameters and thus they lead to different outcomes. Finally, the results indicate that the various architectures for each type of networks do not have a considerable effect on the performance of the models built.

Experiment 2. Through the counter-example in Section 4.3.1, we showed that *two*-layer monotone neural networks cannot approximate all monotone functions with more than one input. Now we again apply Sill networks to the function defined in (4.9), which has inputs x_1 and x_2–to illustrate the approximation capabilities of *three*-layer monotone neural networks. The function values are computed for $\epsilon = 0.3$. To avoid division by zero in computing the average percentage error in this experiment, we slightly modify the original function, i.e., we add the constant one.

The results are reported in Table 4.3. Given the counter-example in

Section 4.3.1, it is clear that two-layer monotone networks could not approximate f very well; therefore, we do not apply them in this simulation study.

Table 4.3: Application of Sill networks to the monotone step function defined in (4.9)

Sill network (groups × planes)	**MSE**	**var(MSE)**	**AvrPE (%)**	**var(AvrPE)**
2×2	0.0333	8.5e-08	8.17	0.18
2×4	0.0334	1.1e-07	8.51	0.03
2×6	0.0333	9.2e-08	8.51	0.05
4×2	0.0022	1.4e-05	1.13	2.55
4×4	0.0003	1.5e-06	0.41	0.27
4×6	0.0004	1.5e-06	0.37	0.28
6×2	0.0012	5.1e-06	0.82	0.96
6×4	0.0006	2.8e-06	0.46	0.52
6×6	0.0001	2.9e-09	0.29	0.02

To better illustrate Sill network approximations of f, we plot the original function values and the corresponding approximated values given by a Sill network with six groups and six hyperplanes; see Figure 4.11.

The results indicate that Sill networks with sufficient number of groups and hyperplanes can adequately approximate a monotone step function, which cannot be approximated by two-layer neural networks. Table 4.3 shows that networks with only two groups produce a poor approximation, whereas networks with four or six groups lead to considerably better results. This can be explained as follows. On the one hand, in the initialization step of our training algorithm, we find a number of groups in the data corresponding to the number of groups in the network, and for each of these groups we fit a linear model; the parameters of the model with added small perturbations are used to initialize the parameters (weights and biases) of each hyperplane in the group. On the other hand, the monotone step function is defined by four regions: two constant regions given by f=1 and f=2, and two continuous regions. Since the function value is monotone in the input variables, the natural grouping with respect to the input data should represent the different values of the true function. Hence, a Sill network with four or more groups is expected to give a good approximation of the monotone step function. As the results, however, show Sill networks with four or six groups

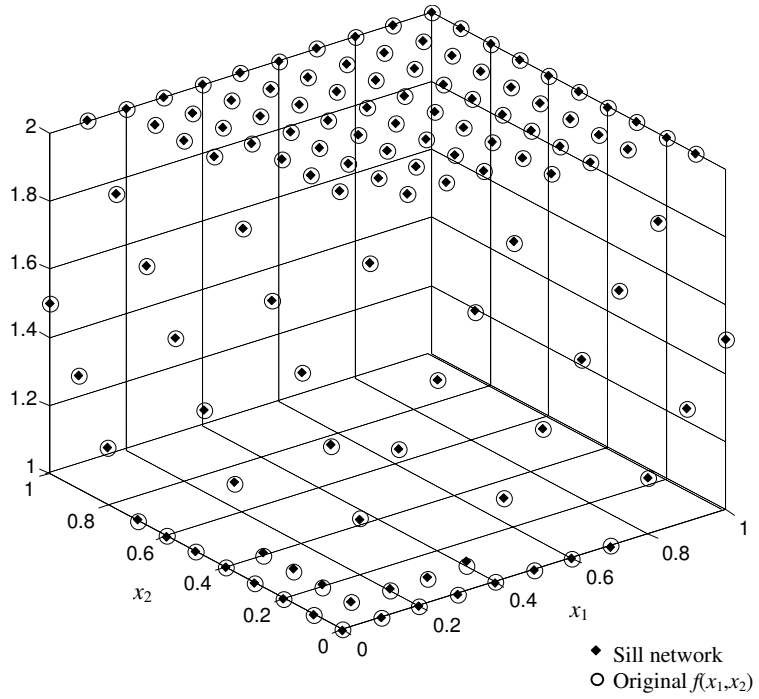


Figure 4.11: Sill network approximation of the monotone step function with six groups and six hyperplanes; MSE = 0.0000045, $AvrPE = 0.0683\%$

and only two hyperplanes in each group do not produce satisfactory results. This is explained by the fact that the function values for the points in the constant regions are also determined by the distribution of the points in the input space, i.e., whether or not $x_1 \geq x_2$. This condition means that more hyperplanes are needed to capture the relationships between the dependent and independent variables. Therefore, Sill networks with four or six groups and four or six hyperplanes in each group lead to better approximations on average (lower error rates).

In summary, the results from our second experiment clearly demonstrate that three-layer Sill networks can approximate monotone functions to any desired degree of accuracy, given sufficient number of groups and hyperplanes; this observation validates the conclusion drawn by Sill (1998).

In general, the results from both experiments show that Sill networks can approximate very well "difficult" functions with a moderate number of groups and hyperplanes. However, we did not find any systematic rule or

method to determine the number of groups and hyperplanes beforehand; this remains a matter of trial and error.

4.3.3 Real case studies

In this section, we report the results from the application of Sill monotone networks in the two case studies introduced in Section 2.5, namely bond rating (a classification problem) and Moscow house pricing (a regression problem).

Our main objective is to build monotone models based on the original data and cleaned data obtained after applying the greedy algorithm for relabeling (Section 2.4), and to compare the models' performance. The experimental set-up is analogous to that used with monotone decision trees (see Section 3.3.2).

We repeat the following experiment 20 times. The raw (non-monotone) data set is randomly partitioned into a construction set with 75% of the observations and a test set with 25% of the observations. The construction set is randomly split into a training set with 50% of the observations, and a validation set with 25% of the observations. The training set is used to train a monotone network, whereas the validation set is used to compute the network's error. We train five monotone networks, and select the one with the lowest error on the validation set. Finally, we compute the prediction error of the model on the test set.

The same experiment is carried out with the cleaned (monotone) data. The only difference is the way the error of the final model is computed: we compute the prediction error on the basis of the same 25% of the observations from the raw data, which were used as the test set in the previous experiment. Thus, the model is constructed from the cleaned data, whereas the performance is measured on the original data.

Finally, in order to provide more general conclusions of the results, for both case studies we use three Sill network topologies with different numbers of groups and hyperplanes in each group; see Table 4.4.

A. Bond rating

The results from the experiments with the monotone and non-monotone bond rating data are reported in Table 4.5.

Table 4.4: Three topologies of Sill networks for two case studies

Number of groups	×	Number of hyperplanes
2	×	2
3	×	3
4	×	4

Table 4.5: Estimated mean and variance of prediction errors of Sill networks for monotone and non-monotone bond rating data

Sill network (groups × planes)	Mean		Variance	
	Monotone data	Non-monotone data	Monotone data	Non-monotone data
2 × 2	0.53	0.54	0.001	0.003
3 × 3	0.49	0.51	0.001	0.001
4 × 4	0.50	0.51	0.002	0.003

The results show that Sill networks trained on the monotone data tend to be more accurate and stable than the networks trained on the non-monotone data. Furthermore, for both cleaned and raw data sets, it is clear that networks with a large number of parameters perform better than networks with fewer parameters. We can notice, however, that across various topologies the networks trained on the non-monotone data have more fluctuating variances compared with the networks trained on the monotone data. This finding indicates that the prediction results obtained from the former networks are more dependent on the topology than those obtained from the latter networks.

Next we test how significant the differences are, given that the test set is the same in the experiments with monotone and non-monotone data, respectively. We conduct paired t-tests of the null hypotheses that the networks built from both data sets have the same prediction error against the one-sided alternatives. Table 4.6 reports the p-values and the confidence intervals at 90% and 95%.

The results from the statistical tests show that for 2×2 and 3×3 networks, the models derived from the monotone data have significantly smaller errors than those derived from the non-monotone data. For 4×4 networks the

Table 4.6: p-values of paired t-tests and one-sided confidence intervals for the difference in error means in the bond rating case study with monotone and non-monotone data

Sill network (groups × planes)	***p*-value**	**Confidence intervals**	
		95%	**90%**
2 × 2	3.0%	[-1, -0.002)	[-1, -0.005)
3 × 3	4.3%	[-1, -0.001)	[-1, -0.004)
4 × 4	16.5%	[-1, 0.008)	[-1, 0.004)

difference in the errors is statistically insignificant at 5% and 10% significance levels.

Finally, we perform F-tests (with 19 degrees of freedom) to check the significance of the differences in variances. As Table 4.5 suggests, we can expect significant differences for the first type of Sill networks (2×2). This is confirmed by the p-value of the test, namely 4.4%. For 3×3 and 4×4 networks the differences are statistically insignificant: the p-values are 50% and 20.1%, respectively.

B. Moscow house pricing

Similar to the bond rating case study, we use monotone and non-monotone Moscow housing data, to build monotone models based on Sill networks. The summary of the results is given in Table 4.7.

Table 4.7: Estimated mean and variance of prediction errors of Sill networks for monotone and non-monotone Moscow housing data

Sill network (groups × planes)	**Mean**		**Variance**	
	Monotone data	**Non-monotone data**	**Monotone data**	**Non-monotone data**
2 × 2	0.48	0.66	0.12	0.23
3 × 3	0.35	0.51	0.12	0.14
4 × 4	0.23	0.44	0.03	0.10

We again compute paired t-statistics to test the significance of the difference in the estimated error means; Table 4.8 reports the p-values and the

one-sided confidence intervals at 90% and 95%.

Table 4.8: p-values of paired t-tests and one-sided confidence intervals for the difference in error means in the Moscow housing case study with monotone and non-monotone data

Sill network (groups × planes)	**P-value**	**Confidence intervals**	
		95%	**90%**
2 × 2	3.0%	(-∞, -0.025)	(-∞, -0.061)
3 × 3	2.3%	(-∞, -0.029)	(-∞, -0.059)
4 × 4	0.0%	(-∞, -0.124)	(-∞, -0.144)

The results clearly indicate that the Sill networks built on the monotone data outperform the Sill networks built on the non-monotone data; the prediction errors of the former are significantly smaller than the errors of the latter.

We again conduct F-tests (with 19 degrees of freedom) for the significance of the difference in variances. The p-values for 2×2 and 3×3 networks are 7.0% and 35.4%, respectively; the p-value of 0.8% for 4×4 shows significantly different variances. Although the differences in the variances for the smaller networks appear to be statistically insignificant at a significance level of 5%, the results show the tendency of the Sill networks trained on the cleaned data to have less variability compared with the Sill networks trained on the raw data.

Discussion of results

Based on the results for the two case studies in this chapter, we draw the following conclusions:

1. The prediction errors of the Sill networks trained on the cleaned data are generally smaller than the prediction errors of the networks trained on the raw data. This finding holds for networks with different topologies (types of architecture).

2. Across the runs with different data samples, monotone models (networks) derived from the monotone data tend to be more stable, i.e., their prediction accuracy varies less, compared with the monotone models derived from the non-monotone data.

Although the results from both case studies indicate that larger networks outperform smaller networks, the problem of setting the right number of groups and hyperplanes in each group remains. On the one hand, as we discussed for the bond rating data, networks with many weights and biases lead to the better predictions but at the cost of increasing the variance. On the other hand, large networks built on the cleaned and raw Moscow housing data, respectively, produced more accurate and stable monotone models. Hence, choosing the number of parameters is domain dependent. However, it is clear that the monotone data lead to models superior to the non-monotone data, irrespective of the problem at hand.

4.4 Conclusion

In this chapter, we discussed monotone neural networks as a method to build monotone models for prediction tasks in data mining. At the beginning of the chapter, we introduced the basic terms and concepts concerning neural networks. We presented the architecture and functionality of a standard neural network, and described backpropagation as the most popular training algorithm. Then, we reviewed previous studies on monotone neural networks, and we discussed their main advantages and disadvantages. In the main part of the chapter, we described two existing approaches for building monotone networks. The first approach is developed by Kay and Ungar (2000); it is based on networks with two layers and positive weights enforced on the connections between the layers. Through a counter-example with two inputs, we demonstrate that their type of network does not have universal approximation capabilities; this disproves the proposition stated by the authors that two-layer monotone networks can approximate any monotone function. Next, we described the architecture and functionality of a three-layer monotone network proposed by Sill (1998). He proves that his type of network can approximate any monotone function to any degree of accuracy. Therefore, we chose Sill's approach to build monotone neural networks (models). Given the objectives of our research, we modified the original architecture, the initialization procedure and training algorithm of Sill networks. Next, we reported the results from our simulation studies with artificially generated data in order to demonstrate the approximation capabilities of Sill networks. First we showed that they can approximate any monotone function including functions that are neither convex nor concave. In the same study we

demonstrated that Sill networks outperform standard neural networks. This finding supports the results reported by Sill (1998), and our first hypothesis stated in the introduction of this thesis (Section 1.4), namely that for monotone problems monotone models have superior predictive performance to non-monotone models. Another simulation study was conducted to illustrate that Sill networks can find adequate approximations of the function used in our counter-example. Finally, we used two case studies, namely bond rating (a classification problem) and house pricing (a regression problem) to build monotone neural networks by using our extended version of the Sill approach. We compared the performance of the Sill networks built on both the cleaned (monotone) and the original (non-monotone) data. The results confirm our second hypothesis, namely monotone models obtained from monotone (transformed) data outperform monotone models obtained from the original data.

Chapter 5

Partial monotonicity

The problems discussed so far in this thesis are based on the assumption that the target function we try to predict is monotone in all explanatory variables. In this chapter we consider *partially monotone prediction problems*, where the dependent variable depends monotonically on some of the independent variables but *not* on all. Our main objective is to construct models for such type of problems. We begin with a simple example of a partially monotone problem, and then we present a formal definition of partial monotonicity. Our main contribution presented in this chapter is a novel method to construct prediction models, where monotone dependences with respect to some of the input variables are preserved by virtue of construction. The basic idea is to convolute Sill monotone networks (see Section 4.3.2) with weight (kernel) functions to make predictions. By using simulation and two new real case studies, we demonstrate the application of our method. We compare the results with standard neural networks and partially monotone linear models. Finally, we give general conclusions about the performance of the models derived from the three methods. The work presented in this chapter has been published in Velikova et al. (2006a, 2006b).

5.1 Introduction

Suppose that for the housing data in Table 2.1 we observe one more variable, namely the number of floors in a house. Then, our data set is represented by Table 5.1.

Table 5.1: Extended house pricing data

No	Number of floors	Area	Number of rooms	Volume	Price (Euro)
1	2	90	2	210	121 000
2	1	86	2	255	130 500
3	3	125	3	320	119 750
4	2	210	4	405	165 200
5	1	174	3	373	190 000

As we noted in the introduction of Chapter 2, common sense suggests that the house price has a monotone increasing dependence on the number of rooms, the total house area and the volume. However, we suspect that a monotone dependency on the number of floors does not necessarily hold; for example, some expensive houses (such as villas) may have only one floor, whereas cheaper houses may have three floors. In other words, the data in Table 5.1 represent an example of a partially monotone problem where the house price depends monotonically on some of the house characteristics but not on all. The question is how to use the prior knowledge about monotone relationships in data to build accurate or easy to interpret prediction models.

It is known that non-monotone functions can often be represented as compositions of monotone functions; for example, unimodal (non-cumulative) probability distribution functions are monotone increasing on the left side of the mode point, and monotone decreasing on the right side (Wang, 1994). This implies that first we can construct a number of monotone models corresponding to the monotone regions in the non-monotone function; then we can combine the local monotone models in order to obtain the global model.

Let us again consider the example in Table 5.1. We form three groups of houses with respect to the three values for the number of floors, namely the first group consists of the second and fifth house, the second group consists of the first and fourth houses, and the third group is based only on the third house. Within these three groups, the other house attributes have a monotone relationship with the house price. We build separate monotone models based only on the latter three attributes; for example, we use monotone neural networks. Finally, to predict the price of a new house, we simply use the monotone model obtained from the group to which the house belongs based on the number of floors.

In the remainder of this section we first introduce some notation and definitions, which are used in the discussion throughout the chapter.

Notation and definitions

For the partially monotone problems defined in the introduction of this thesis (p. 20), we have $\mathcal{X} = \mathcal{X}^m \cup \mathcal{X}^{nm}$ with $\mathcal{X}^m = \prod_{i=1}^{m} \mathcal{X}_i$ and $\mathcal{X}^{nm} = \prod_{i=m+1}^{k} \mathcal{X}_i$ for $1 \leq m < k$.

Furthermore, we have a data set $D = (\mathbf{x}^m, \mathbf{x}^{nm}, \ell_{\mathbf{x}})^N$, where $\mathbf{x}^m \in \mathcal{X}^m$, $\mathbf{x}^{nm} \in \mathcal{X}^{nm}$, and N is the number of observations. A data point $\mathbf{x} \in D$ is represented by $\mathbf{x} = (\mathbf{x}^m, \mathbf{x}^{nm})$; the label of $\mathbf{x}$ is $\ell_{\mathbf{x}}$. We assume that D is generated by the following process

$$\ell_{\mathbf{x}} = f(\mathbf{x}^m, \mathbf{x}^{nm}) + \epsilon, \tag{5.1}$$

where f is a monotone function in $\mathbf{x}^m$ and ϵ is a random error. In regression problems, ϵ has zero mean, whereas in classification problems ϵ is a small probability that the assigned class is incorrect.

The partial monotonicity constraint of f on $\mathbf{x}^m$ is defined by

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}: \quad \mathbf{x}^{nm} = \mathbf{x}'^{nm} \text{ and } \mathbf{x}^m \geq \mathbf{x}'^m \Rightarrow f(\mathbf{x}) \geq f(\mathbf{x}'). \tag{5.2}$$

Henceforth, we call $\mathcal{X}^m$ the *set of monotone variables* and $\mathcal{X}^{nm}$ the *set of non-monotone variables*. By *non-monotone* we mean that it is not known a priori a variable to be monotone. Although, we do not constrain the size of the two sets, our main assumption for the problems considered in this chapter is that we have only a small number of non-monotone variables, and a large number of monotone variables. Thus, monotonicity plays an important role in the data generating process, and needs to be preserved.

Our objective is to find a smooth approximation $\hat{f}$ of $f(\mathbf{x}^m, \mathbf{x}^{nm})$, such that $\hat{f}$ is monotone in $\mathbf{x}^m$, i.e., $\hat{f}$ is a *partially monotone estimator*. As we discussed in the introduction of this thesis, in practice the true function $f(\mathbf{x}^m, \mathbf{x}^{nm})$ is unknown, and therefore we use $\ell_{\mathbf{x}}$ in (5.1) as a close proximity of $f(\mathbf{x}^m, \mathbf{x}^{nm})$ to find $\hat{f}$.

A simple solution is to consider the class of partially monotone linear functions of the form:

$$\hat{f} = a_0 + \sum_{i=1}^{m} a_i x_i^m + \sum_{j=m+1}^{k} a_j x_j^{nm} \quad \text{subject to} \quad a_i \geq 0, i = 1, \ldots, m. \quad (5.3)$$

We expect that the estimate in (5.3) produces good fit for simple (e.g., linear) functions; however, it gives poor approximations for complex functions. Therefore, it is necessary to consider more flexible models for estimating an arbitrary partially monotone function.

Following the definition of partial monotonicity in (5.2), we can simply estimate f based only on the values of the monotone variables $\mathbf{x}^m$ for each possible value of $\mathbf{x}^{nm}$. This approach can only be applied in very simple cases such as the example in the beginning of this chapter. However, for large data sets with multiple or continuous non-monotone variables, the approach is impractical: there are too many values of $\mathbf{x}^{nm}$, and most of them are not observed. To overcome this problem, instead of taking each separate value of $\mathbf{x}^{nm}$, we cluster the observations into groups that are similar with respect to the non-monotone variables. On each group we then build monotone estimations based on the values of the monotone variables only. Finally, we smooth out the resulting estimations by using weight functions (kernels) based on $\mathbf{x}^{nm}$.

As we discussed in Chapter 4, feed-forward neural networks are powerful computational tools that can approximate an arbitrary function to any desired level of accuracy. Therefore, in this study, we consider the outputs of monotone neural networks as monotone function estimations for each group. In the literature, these local networks are often referred to as *experts*, and hence the overall function representation as a *mixture of experts* (Jacobs et al., 1991). In Section 5.3.1, we propose a mixture-of-networks model that preserves partial monotonicity by virtue of construction; we prove that our partially monotone model has universal function approximation capabilities.

Furthermore, our approach is based on the "divide-and-conquer" strategy: the full data set is naturally divided into several subsets (groups) according to knowledge of the problem, and on each subset, a separate network expert is used to solve a particular sub-task. This strategy has several advantages. First, a set of experts are used at the decision level to tackle a complex prediction problem. This approach helps to focus on smaller objectives, which are simpler and thus easier to achieve. In many practical situations

this aspect also leads to a more realistic representation of the data generating process; for example, the distribution of people using their credit cards to make purchases may be modeled by two groups (components): those who are unlikely to use their credit cards and those who do so (Hand et al., 2001). Another useful aspect of mixture-of-experts models is that they can still be applied even if the specific parametric form for modeling the data is uncertain. Finally, the use of a large number of parameters allows extra flexibility and hence better accuracy of mixture models compared with single models.

5.2 Related work

Although a number of recent studies discuss mixture-of-experts models for prediction, they do not deal with incorporating partial monotonicity constraints (Jacobs et al., 1991; Jordan and Jacobs, 1994; Frosyniotis et al., 2003; Suárez-Fariñas and Pedreira, 2003). From this perspective, our algorithm for building partially monotone models, described in the next section, can be considered as a new approach in the field.

An alternative solution based on a single neural-network approach is proposed by William Armstrong, the developer of Adaptive Logic Networks discussed in Section 4.2. His type of networks is a combination of minimum and maximum operators over linear functions. Due to their architecture, it is easy to constrain the weights on the monotone variable(s) to be non-negative, and thus to obtain a partially monotone model. Armstrong claims that his type of networks has universal function approximation capabilities, without providing a formal proof for that. In Appendix B we give such a proof.

5.3 Algorithm for partial monotonicity

5.3.1 Description

The working scheme of our algorithm is as follows. First we find a number of natural groups in the data with respect to the set of non-monotone variables. Next, for each group we apply a Sill network (see Section 4.3.2) to obtain a monotone function estimation based on the set of monotone vari-

ables. Finally, we convolute these monotone estimations with suitable weight functions (kernels) based on the set of non-monotone variables to obtain the overall model.

More formally, in the first step of our approach, we partition the input space with respect to $\mathbf{x}^{nm}$ into a number of disjoint subsets (clusters) by using the so-called agglomerative (merging) type of hierarchical clustering with complete-linkage distance (see Appendix C for more details). The appropriate number of clusters is determined automatically in the following way. We first cut off the hierarchy obtained from the clustering procedure at several levels (from two to ten). Then for each of the partitioning outcomes we compute the "silhouette value" as a measure for the goodness of clustering (ranged from -1 for bad to $+1$ for good) (Rousseeuw, 1987). The outcome with the maximal silhouette value determines the final number of clusters. An additional improvement in the clustering procedure is adding weights $\boldsymbol{\alpha} > 0$ to the variables in the standard Euclidean distance measure we use. In this way, we take into account the significance of each variable on the dissimilarities between the points and the formation of the clusters, respectively. The outline of our clustering procedure is given in Algorithm 5.1.

Algorithm 5.1 Data clustering: CLUSTER(D,$\boldsymbol{\alpha}$)

$dist$(D,$\boldsymbol{\alpha}$) = $N \times N$ dissimilarity matrix containing the Euclidean distances, weighted by $\boldsymbol{\alpha}$, between the points in D
hCl = a hierarchical cluster tree based on $dist(D,\boldsymbol{\alpha})$ and the complete-linkage distance
$sv_{max} = -1$
for $c = 2$ to 10 **do**
  $[D_1, \ldots, D_c, \bar{\mathbf{x}}_1^{nm}, \ldots, \bar{\mathbf{x}}_c^{nm}]$ = disjoint clusters (subsets of D) with their centroids obtained after cutting off hCl into c clusters
  sv_c= silhouette value obtained for c clusters
  if $sv_{max} < sv_c$ **then**
    $sv_{max} = sv_c$
    $[D_1, \ldots, D_C, \bar{\mathbf{x}}_1^{nm}, \ldots, \bar{\mathbf{x}}_C^{nm}] = [D_1, \ldots, D_c, \bar{\mathbf{x}}_1^{nm}, \ldots, \bar{\mathbf{x}}_c^{nm}]$
  end if
end for
return $[D_1, \ldots, D_C, \bar{\mathbf{x}}_1^{nm}, \ldots, \bar{\mathbf{x}}_C^{nm}]$

As a result of this partitioning of the original data D, we obtain a number C of subsets $D_1, \ldots, D_C$; the number of points in the subsets is not necessarily the same. There is no restriction on the minimal number of points in a subset. For each D_c, $c = 1, 2, \ldots, C$, which contains more than one point, the value of the non-monotone variable is fixed to the cluster mean $\bar{\mathbf{x}}_c^{nm}$. Furthermore, an estimate $\hat{f}(\mathbf{x}^m)$ of f is obtained based only on the values of the monotone variable $\mathbf{x}^m$ for the points belonging to D_c. This is done by using Sill networks, which guarantees that the function approximation is monotone within each subset.

If a cluster with only one point is created (i.e., an outlier with respect to the values of the non-monotone variables is detected), then the cluster mean takes the values of the non-monotone variable for that point, and the function approximation is simply the label of the point. The reasoning for not ignoring the one-point clusters is as follows. Suppose we want to predict the label $\ell_{\mathbf{z}}$ of a new point $\mathbf{z}$, which is closer to a one-point cluster than to the other clusters (meaning that the values of the non-monotone variables are similar). Now if $\mathbf{z}$ also has values of the monotone variables that are similar to those of the point in the cluster, then the predicted label is also expected to be close to the label of the point. However if the values of the monotone variables are dissimilar, then $\mathbf{z}$ can be considered as a point without an analog in the data (i.e., outlier) but its label can still be predicted by using the function estimations from all the clusters as described below.

In the next step, we define for each subset D_c, $c = 1, 2, \ldots, C$,

$$\psi_c(\mathbf{x}^{nm}) = \frac{1}{e^{\|\boldsymbol{\alpha}(\mathbf{x}^{nm} - \bar{\mathbf{x}}_c^{nm})\|}}, \tag{5.4}$$

where $\|\cdot\|$ is the Euclidean distance norm weighted by $\boldsymbol{\alpha}$, $\mathbf{x}^{nm} \in D$ are the values of the non-monotone variables, and $\bar{\mathbf{x}}_c^{nm}$ is the mean (centroid) value of the non-monotone variables for the points falling in cluster c. By definition $\psi_c > 0$ and it determines the distance of a point $\mathbf{x}^{nm}$ to the mean $\bar{\mathbf{x}}_c^{nm}$ of cluster c.

By normalizing ψ_c with

$$\varphi_c(\mathbf{x}^{nm}) = \frac{\psi_c(\mathbf{x}^{nm})}{\sum_{c'=1}^{C} \psi_{c'}(\mathbf{x}^{nm})}, \tag{5.5}$$

we obtain a function $\varphi_c > 0$, for which

$$\sum_{c=1}^{C} \varphi_c(\mathbf{x}^{nm}) = 1.$$

Hence, φ_c can be considered as a weight function or kernel in Nadaraya-Watson form (Nadaraya, 1964; Watson, 1964).

Finally, we convolute φ_c with the corresponding monotone approximations $\hat{f}_c(\mathbf{x}^m)$ for all clusters by

$$\hat{f}(\mathbf{x}^m, \mathbf{x}^{nm}) = \sum_{c=1}^{C} \varphi_c(\mathbf{x}^{nm}) \cdot \hat{f}_c(\mathbf{x}^m) \tag{5.6}$$

to obtain the final estimate of f. Note that $\hat{f}(\mathbf{x}^m, \mathbf{x}^{nm})$ is monotone in $\mathbf{x}^m$, and a weighted sum of monotone functions is monotone. So, $\hat{f}(\mathbf{x}^m, \mathbf{x}^{nm})$ is guaranteed to be a partially monotone estimator by construction.

The outline for the algorithm for partial monotonicity is given in Algorithm 5.2.

Algorithm 5.2 Building partially monotone models and prediction

`Construction_set` = Construction data
`Test_set` = Test data
$\mathbf{x}^m$ = set of monotone variables from `Construction_set`
$\mathbf{x}^{nm}$ = set of non-monotone variables from `Construction_set`
$\mathbf{x}^m_{Tset}$ = set of monotone variables from `Test_set`
$\mathbf{x}^{nm}_{Tset}$ = set of non-monotone variables from `Test_set`
$\boldsymbol{\alpha}$ = positive weight coefficients
$[D_1, \ldots, D_C, \bar{\mathbf{x}}_1^{nm}, \ldots, \bar{\mathbf{x}}_C^{nm}]$ = CLUSTER(`Construction_set`($\mathbf{x}^{nm}$),$\boldsymbol{\alpha}$)

for all $\mathbf{x}_{Tset} \in$ `Test_set` **do**
  for $c = 1$ to C **do**
    $MonNet_c(\mathbf{x}^m)$ = Sill network trained on $D_c(\mathbf{x}^m, \ell_{\mathbf{x}})$
    $\hat{f}_c(\mathbf{x}^m_{Tset})$ = output of $MonNet_c(\mathbf{x}^m_{Tset})$
    $\psi_c(\mathbf{x}^{nm}_{Tset}) = 1/e^{\|\boldsymbol{\alpha}(\mathbf{x}^{nm}_{Tset} - \bar{\mathbf{x}}^{nm}_c)\|}$
    $\varphi_c(\mathbf{x}^{nm}_{Tset}) = \psi_c(\mathbf{x}^{nm}_{Tset}) / \sum_{c'=1}^{C} \psi_{c'}(\mathbf{x}^{nm}_{Tset})$
  end for

  $\hat{f}(\mathbf{x}^m_{Tset}, \mathbf{x}^{nm}_{Tset}) = \sum_{c=1}^{C} \varphi_c(\mathbf{x}^{nm}_{Tset}) \cdot \hat{f}_c(\mathbf{x}^m_{Tset})$
end for

In the following theorem we show that our partially monotone estimator has universal approximation capabilities.

Theorem 5.3.1. *Let $\mathcal{X} = \mathcal{X}^m \cup \mathcal{X}^{nm}$ be a closed bounded domain of k inputs, with closed subsets $\mathcal{X}^m = \prod_{i=1}^{m} \mathcal{X}_i$ and $\mathcal{X}^{nm} = \prod_{i=m+1}^{k} \mathcal{X}_i$ for $1 \leq m < k$. Furthermore, we have $\mathbf{x} = (\mathbf{x}^m, \mathbf{x}^{nm})$ with $\mathbf{x}^m \in \mathcal{X}^m$, and $\mathbf{x}^{nm} \in \mathcal{X}^{nm}$. Let $f(\mathbf{x}^m, \mathbf{x}^{nm})$ be a continuous bounded function mapping $\mathcal{X}$ to $\Re^+$, which is monotone in $\mathbf{x}^m$. Then, for any $\epsilon > 0$ there exists a partially monotone estimator $\hat{f}(\mathbf{x}^m, \mathbf{x}^{nm})$ in the form of (5.6) such that $\left|f(\mathbf{x}^m, \mathbf{x}^{nm}) - \hat{f}(\mathbf{x}^m, \mathbf{x}^{nm})\right| < \epsilon$, for any $\mathbf{x} \in \mathcal{X}$.*

Proof. Let $\epsilon > 0$ and let Φ bound the magnitude of f on $\mathcal{X}$. By definition f is uniformly continuous on $\mathcal{X}^{nm}$, so there exists $\delta > 0$ such that for any points $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ with $\|\mathbf{x}^{nm} - \mathbf{x}'^{nm}\| < \delta$, we have $\left|f(\mathbf{x}^m, \mathbf{x}^{nm}) - f(\mathbf{x}^m, \mathbf{x}'^{nm})\right| < \epsilon/4$.

We now define an equispaced grid S of points on $\mathcal{X}^{nm}$ such that the spacing between grid points along each dimension is $\delta/\sqrt{k-m}$.

Next for any grid point $\mathbf{s} \in S$ with value $\mathbf{s}^{nm}$ we construct a Sill network approximation $\hat{f}_{\mathbf{s}}(\mathbf{x}^m)$ based on the values of the monotone variables only, such that $\left|f(\mathbf{x}^m, \mathbf{s}^{nm}) - \hat{f}_{\mathbf{s}}(\mathbf{x}^m)\right| < \epsilon/2$. The existence of such an approximation is guaranteed by the universal approximation capabilities of Sill networks (see Theorem 4.3.1, p. 116).

Now we consider any point $\mathbf{x} = (\mathbf{x}^m, \mathbf{x}^{nm})$, $\mathbf{x} \in \mathcal{X}$. Let $\varphi_{\mathbf{s}}(\mathbf{x}^{nm})$ be a function in the form of (5.5), which measures the distance between $\mathbf{x}^{nm}$ and the grid point $\mathbf{s}$. We take $\boldsymbol{\alpha} = (\alpha, \ldots, \alpha)$. Hence,

$$\psi_{\mathbf{s}}(\mathbf{x}^{nm}) = \frac{1}{e^{\alpha\|\mathbf{x}^{nm} - \mathbf{s}\|}}$$

and

$$\varphi_{\mathbf{s}}(\mathbf{x}^{nm}) = \frac{\psi_{\mathbf{s}}(\mathbf{x}^{nm})}{\sum_{\mathbf{s}' \in S} \psi_{\mathbf{s}'}(\mathbf{x}^{nm})}.$$

Next, as an approximation of $f(\mathbf{x}^m, \mathbf{x}^{nm})$, we take

$$\hat{f}(\mathbf{x}^m, \mathbf{x}^{nm}) = \sum_{\mathbf{s} \in S} \varphi_{\mathbf{s}}(\mathbf{x}^{nm}) \hat{f}_{\mathbf{s}}(\mathbf{x}^m).$$

We will show that $\left|f(\mathbf{x}^m, \mathbf{x}^{nm}) - \hat{f}(\mathbf{x}^m, \mathbf{x}^{nm})\right| < \epsilon$.

First, by the triangle inequality, we have

$$\begin{aligned}\left|f(\mathbf{x}^m,\mathbf{x}^{nm})-\hat{f}(\mathbf{x}^m,\mathbf{x}^{nm})\right| \leq \\ \left|f(\mathbf{x}^m,\mathbf{x}^{nm})-\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})f(\mathbf{x}^m,\mathbf{x}^{nm})\right| \\ +\left|\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})f(\mathbf{x}^m,\mathbf{x}^{nm})-\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})f(\mathbf{x}^m,\mathbf{s}^{nm})\right| \\ +\left|\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})f(\mathbf{x}^m,\mathbf{s}^{nm})-\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})\hat{f}_{\mathbf{s}}(\mathbf{x}^m)\right|.\end{aligned} \tag{5.7}$$

Next we consider the three absolute-value terms on the right-hand side of the inequality in (5.7). For the first term, we have

$$\begin{aligned}\left|f(\mathbf{x}^m,\mathbf{x}^{nm})-\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})f(\mathbf{x}^m,\mathbf{x}^{nm})\right| &= f(\mathbf{x}^m,\mathbf{x}^{nm})\left(1-\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})\right) \\ &= 0.\end{aligned}$$

and for the third term,

$$\begin{aligned}\left|\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})f(\mathbf{x}^m,\mathbf{s}^{nm})-\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})\hat{f}_{\mathbf{s}}(\mathbf{x}^m)\right| = \\ \sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})\left|f(\mathbf{x}^m,\mathbf{s}^{nm})-\hat{f}_{\mathbf{s}}(\mathbf{x}^m)\right| < \sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})\frac{\epsilon}{2}=\frac{\epsilon}{2}.\end{aligned}$$

For the second term in (5.7), we have

$$\begin{aligned}
&\left|\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})f(\mathbf{x}^{m},\mathbf{x}^{nm})-\sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})f(\mathbf{x}^{m},\mathbf{s}^{nm})\right| = \\
&\qquad \sum_{\mathbf{s}\in S}\varphi_{\mathbf{s}}(\mathbf{x}^{nm})\left|f(\mathbf{x}^{m},\mathbf{x}^{nm})-f(\mathbf{x}^{m},\mathbf{s}^{nm})\right| = \\
&\qquad\qquad \sum_{\substack{\mathbf{q}\in S\\ \|\mathbf{x}^{nm}-\mathbf{q}^{nm}\|<\delta}}\varphi_{\mathbf{q}}(\mathbf{x}^{nm})\left|f(\mathbf{x}^{m},\mathbf{x}^{nm})-f(\mathbf{x}^{m},\mathbf{q}^{nm})\right| \\
&\qquad + \sum_{\substack{\mathbf{t}\in S\\ \|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|\geq\delta}}\varphi_{\mathbf{t}}(\mathbf{x}^{nm})\left|f(\mathbf{x}^{m},\mathbf{x}^{nm})-f(\mathbf{x}^{m},\mathbf{t}^{nm})\right|.
\end{aligned} \tag{5.8}$$

Now we consider separately the last two parts in (5.8). By the definition of the grid for the first part, we have

$$\sum_{\substack{\mathbf{q}\in S\\ \|\mathbf{x}^{nm}-\mathbf{q}^{nm}\|<\delta}}\varphi_{\mathbf{q}}(\mathbf{x}^{nm})\left|f(\mathbf{x}^{m},\mathbf{x}^{nm})-f(\mathbf{x}^{m},\mathbf{q}^{nm})\right| < \sum_{\substack{\mathbf{q}\in S\\ \|\mathbf{x}^{nm}-\mathbf{q}^{nm}\|<\delta}}\varphi_{\mathbf{q}}(\mathbf{x}^{nm})\frac{\epsilon}{4} < \frac{\epsilon}{4}.$$

For the second part we reason as follows. First note that for any two grid points $\mathbf{s},\mathbf{p}\in S$ we have

$$\varphi_{\mathbf{s}}(\mathbf{x}^{nm}) = \frac{\psi_{\mathbf{s}}(\mathbf{x}^{nm})}{\sum_{\mathbf{s}'\in S}\psi_{\mathbf{s}'}(\mathbf{x}^{nm})} \leq \frac{\psi_{\mathbf{s}}(\mathbf{x}^{nm})}{\psi_{\mathbf{p}}(\mathbf{x}^{nm})} = e^{-\alpha\left(\|\mathbf{x}^{nm}-\mathbf{s}^{nm}\|-\|\mathbf{x}^{nm}-\mathbf{p}^{nm}\|\right)}.$$

By the definition of the grid, there always exists a grid point $\mathbf{p}$ with $\|\mathbf{x}^{nm}-\mathbf{p}^{nm}\|\leq\delta/2$. Then for any grid point $\mathbf{t}$ with $\|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|\geq\delta$, we have

$$\left(\|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|-\|\mathbf{x}^{nm}-\mathbf{p}^{nm}\|\right)\geq\delta/2.$$

Hence,

$$\sum_{\substack{\mathbf{t}\in S\\ \|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|\geq\delta}} \varphi_{\mathbf{t}}(\mathbf{x}^{nm})\left|f(\mathbf{x}^m,\mathbf{x}^{nm})-f(\mathbf{x}^m,\mathbf{t}^{nm})\right| \leq$$

$$\sum_{\substack{\mathbf{t}\in S\\ \|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|\geq\delta}} \frac{\psi_{\mathbf{t}}(\mathbf{x}^{nm})}{\psi_{\mathbf{p}}(\mathbf{x}^{nm})}\left|f(\mathbf{x}^m,\mathbf{x}^{nm})-f(\mathbf{x}^m,\mathbf{t}^{nm})\right| =$$

$$\sum_{\substack{\mathbf{t}\in S\\ \|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|\geq\delta}} e^{-\alpha\left(\|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|-\|\mathbf{x}^{nm}-\mathbf{p}^{nm}\|\right)}\left|f(\mathbf{x}^m,\mathbf{x}^{nm})-f(\mathbf{x}^m,\mathbf{t}^{nm})\right|.$$

Furthermore, for any term in the last sum, we have

$$e^{-\alpha\left(\|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|-\|\mathbf{x}^{nm}-\mathbf{p}^{nm}\|\right)}\left|f(\mathbf{x}^m,\mathbf{x}^{nm})-f(\mathbf{x}^m,\mathbf{t}^{nm})\right| \leq$$
$$e^{-\alpha\left(\|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|-\|\mathbf{x}^{nm}-\mathbf{p}^{nm}\|\right)}\ \Phi \leq e^{-\alpha\frac{\delta}{2}}\ \Phi$$
$$< \left(\alpha\frac{\delta}{2}\right)^{-1}\ \Phi.$$

Now we take $\alpha = 8\Phi(\mathcal{N}-1)/\delta\epsilon$, where $\mathcal{N}$ is the number of grid points. Then,

$$\sum_{\substack{\mathbf{t}\in S\\ \|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|\geq\delta}} \varphi_{\mathbf{t}}(\mathbf{x}^{nm})\left|f(\mathbf{x}^m,\mathbf{x}^{nm})-f(\mathbf{x}^m,\mathbf{t}^{nm})\right| <$$
$$\sum_{\substack{\mathbf{t}\in S\\ \|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|\geq\delta}} \left(\frac{8\Phi(\mathcal{N}-1)}{\delta\epsilon}\frac{\delta}{2}\right)^{-1}\Phi = \sum_{\substack{\mathbf{t}\in S\\ \|\mathbf{x}^{nm}-\mathbf{t}^{nm}\|\geq\delta}} \frac{\epsilon}{4(\mathcal{N}-1)} \leq \frac{\epsilon}{4}.$$

So, we finally obtain

$$\sum_{\mathbf{s}\in S} \varphi_{\mathbf{s}}(\mathbf{x}^{nm}) \left| f(\mathbf{x}^m, \mathbf{x}^{nm}) - f(\mathbf{x}^m, \mathbf{s}^{nm}) \right| < \frac{\epsilon}{4} + \frac{\epsilon}{4} = \frac{\epsilon}{2}.$$

Hence,

$$\left| f(\mathbf{x}^m, \mathbf{x}^{nm}) - \hat{f}(\mathbf{x}^m, \mathbf{x}^{nm}) \right| < 0 + \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.$$

□

5.3.2 Simulation studies

In this section, we present the results of our simulation studies designed to test the effectiveness of our approach for partially monotone problems. We generate an artificial data set D based on a set of independent variables and a continuous dependent variable computed by applying a function that is monotone only on a subset of the independent variables. Based on D thus generated we build a model for predicting the dependent variable. Since the problem is partially monotone, we apply our approach for partial monotonicity. For practical reasons, we slightly modify the approach.

First we need to determine a priori the weights $\boldsymbol{\alpha}$ measuring the impact of each non-monotone variable on the response variable. To do so, we take the absolute value of the respective coefficients for each non-monotone variable obtained from the linear model fitted to the whole data set $D = (\mathbf{x}^m, \mathbf{x}^{nm}, \ell_{\mathbf{x}})^N$, and we normalize them to sum up to one. Thus we use the information provided by the data to obtain more appropriate values of $\boldsymbol{\alpha}$. More formally, based on D we fit a linear model

$$\ell_{\mathbf{x}} = a_0 + \sum_{i=1}^{m} a_i x_i^m + \sum_{j=m+1}^{k} a_j x_j^{nm}.$$

Next, we compute

$$\boldsymbol{\alpha} = (\alpha_{m+1}, \ldots, \alpha_k), \tag{5.9}$$

where

$$\alpha_j = \frac{|a_j|}{\sum_{m+1}^{k} |a_j|} \quad \text{for } j = m+1, \ldots, k.$$

Second, we simplify the function ψ measuring the distance between $\mathbf{x}^{nm}$ and $\bar{\mathbf{x}}_c^{nm}$, $c = 1, 2, \ldots, C$, by

$$\psi_c(\mathbf{x}^{nm}) = \frac{1}{\|\boldsymbol{\alpha}\,(\mathbf{x}^{nm} - \bar{\mathbf{x}}_c^{nm})\| + \mu}.$$

where μ is a small positive number.

To obtain a sound assessment of the performance of our approach, we use as benchmark methods for comparison standard neural networks with weight decay and partially monotone linear models in the form of (5.3). The standard neural networks consist of an input layer, one hidden layer and one continuous output. In the hidden layer the activation function is sigmoid, whereas in the output it is linear. In addition, weight decay (see Section 4.1) is used as a regularization method to prevent the networks from overfitting. This is done by adding to the mean-squared error the term $\lambda \sum_{ij} w_{ij}^2$ to penalize large weights, where λ is the weight decay parameter.

Given that our target function is continuous, the comparison between our approach and the benchmark methods is based on the mean-squared error (MSE) as a measure for the quality of estimation. In addition, as the data generating process (true function $f(\mathbf{x})$) is known, we use the bias-variance decomposition of MSE to gain more insight into the performance of the methods used in the simulation studies. Recall from the introduction of this thesis (p. 21) that the prediction error (MSE or misclassification error) can be decomposed into three components: squared bias, variance, and irreducible error (variance of the noise term ϵ). Since the last component is independent of the model constructed and does not affect the comparisons, we omit it from the computations of the MSE. Thus, in our simulations for each estimator $\hat{f}_{M_D}(\mathbf{x})$ based on method M_D applied on a data set D, MSE is computed by

$$\text{MSE} = \text{Bias}^2 + \text{Variance},$$

where

$$\text{Bias}^2 = (f(\mathbf{x}) - \mathcal{E}_D[\hat{f}_{M_D}(\mathbf{x})])^2,$$

and

$$\text{Variance} = \mathcal{E}_D[(\hat{f}_{M_D}(\mathbf{x}) - \mathcal{E}_D[\hat{f}_{M_D}(\mathbf{x})])^2].$$

Furthermore, to improve our performance analysis, we conduct the experiments with our approach for partial monotonicity and neural networks with weight decay by using several factors, each with three values; see Table 5.2.

Table 5.2: Factors with their values in simulation experiments on partially monotone problems

Approach for partial monotonicity					Neural networks with weight decay				
Factors		Levels (values)			Factors		Levels (values)		
		1	2	3			1	2	3
1	# points in data	50	150	250	1	# points in data	50	150	250
2	Noise level (σ_ϵ^2)	0.01	0.5	2	2	Noise level (σ_ϵ^2)	0.01	0.5	2
3	# groups in Sill net	2	3	4	3	# hidden neurons	3	9	15
4	# planes in Sill net	2	3	4	4	Weight decay (λ)	0.000001	0.00001	0.0001

All possible combinations of four three-value factors require 81 (3^4) experiments with each method. To reduce the effort and experimental cost in the simulations, we use the so-called *fractional factorial design*, where a smaller number of combinations of factor values are taken to carry out the experiments (Wu and Hamada, 2000). This is done in a systematic way by combining each value of each factor only once with each level of the other factors. In our case the fractional design requires only nine runs (trials) with each method (Wu and Hamada, 2000); see Table 5.3. For example, for trial #1 with the algorithm for partial monotonicity, all four factors are at their first levels, i.e, we generate data sets with 50 data points and noise level $\sigma_\epsilon^2 = 0.01$, and apply Sill networks with two groups and two hyperplanes.

For each run we generate a collection of 100 data samples. For computational convenience the values of the independent variables in each set are fixed, whereas the value of the dependent variable varies across different data samples. The approach for partial monotonicity, neural networks with weight decay and partially monotone linear models are applied on the same collections of data samples.

From the experiments with each method we obtain nine estimates of the two measures (squared bias and variance) of the models. Next, these results are used to compute the expected value $\mathcal{E}(\Theta_{ijkl})$ of each measure Θ for all possible combinations of factor values (i, j, k, l), where i, j, k and l range from one to three. As described by Wu and Hamada (2000), this can be done by fitting the exponential model

Table 5.3: Fractional factorial design for four factors with three levels

Runs (trials)	Factors			
	1	**2**	**3**	**4**
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

$$\mathcal{E}(\Theta_{ijkl}) = \exp\big(\mu + (\mu_i^1 - \mu) + (\mu_j^2 - \mu) + (\mu_k^3 - \mu) + (\mu_l^4 - \mu)\big), \quad (5.10)$$

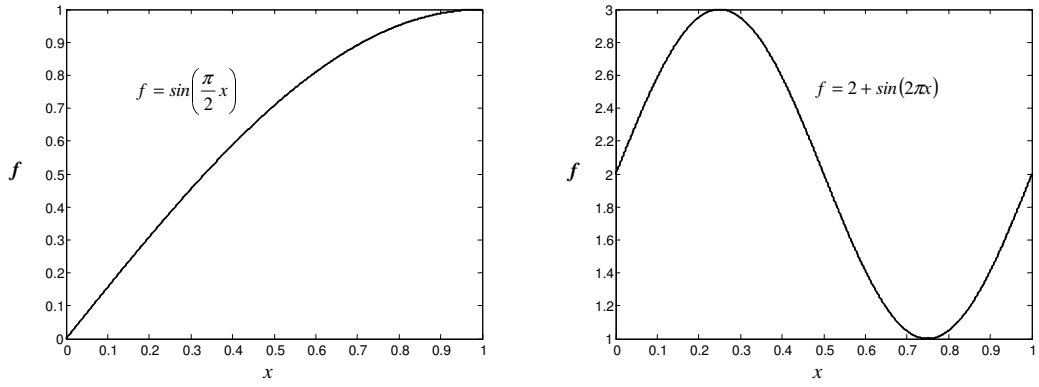
where μ is the total mean computed over all nine estimates, μ_i^1, μ_j^2, μ_k^3 and μ_l^4 are the means for each factor value; the exponential fit guarantees that the estimated $\mathcal{E}(\Theta_{ijkl})$ is positive. For example, for the combination of factor values (50 data points, $\sigma_\epsilon^2 = 0.5$, four groups, three planes), i.e., $(i = 1, j = 2, k = 3, l = 2)$ the approach for partial monotonicity has not been run. To estimate $\mathcal{E}(\Theta_{1232})$, we substitute the estimated means $\hat{\mu}_1^1$, $\hat{\mu}_2^2$, $\hat{\mu}_3^3$ and $\hat{\mu}_2^4$ in (5.10).

Next, we compute MSE by summing up the corresponding estimates of the squared bias and variance for all possible combinations of factor values.

Finally, as there are two factors (number of data points and noise level) that are the same in the experiments, we want to compare the performance of the methods for all combinations of values (i, j) (in total nine) of these two factors. For this purpose, within each (i, j), out of all nine value combinations we take the minimum estimated value with the corresponding variance of MSE over the other two factors.

To draw more general conclusions from our simulation study, we conduct two types of experiments described below.

Experiment 1. First, two vectors of N values, x^m and x^{nm}, are generated independently from each other. The values of vector x^m are drawn



(a) Monotone function (b) Non-monotone function

Figure 5.1: Functions used to generate the data in Experiment 1 for partially monotone problems

from the uniform distribution on [0,1]. The vector x^{nm} is a composition of two sub-vectors each of size $N/2$ points drawn from two normal (Gaussian) distributions: $\mathcal{N}(0.02, 0.05)$ and $\mathcal{N}(0.08, 0.05)$. Finally, we compute the values of a third vector $\ell_{\mathbf{x}}$ by applying a monotone function on x^m and a non-monotone function on x^{nm} plus a random perturbation $\epsilon \sim \mathcal{N}(0, \sigma^2)$:

$$\ell_{\mathbf{x}} = 3 + \sin\left(\frac{\pi}{2}x^m\right)\left(2 + \sin(2\pi x^{nm})\right) + \epsilon. \tag{5.11}$$

The monotone and non-monotone functions are depicted in Figure 5.1.

Hence, we can consider $\mathbf{x} = (x^m, x^{nm})$ as a data point, x^m and x^{nm} as the independent variables and $\ell_{\mathbf{x}}$ as the dependent variable in a data set $D = (x^m,\, x^{nm},\, \ell_{\mathbf{x}})$ of N points.

On the artificial data thus generated, we apply the approach for partial monotonicity (PartMon), neural networks with weight decay (NNet) and partially monotone linear models (PMonLin). The results are summarized in Table 5.4.

We draw the following conclusions from these results:

- Given the universal approximation capabilities of standard neural networks, it is not surprising that they achieve closer approximations (i.e., lower squared bias) than our approach for partial monotonicity and partially linear monotone models.

Table 5.4: Minimum MSE obtained by the three methods in Experiment 1 with partially monotone problems

Method	50 points			150 points			250 points		
	$\sigma_\epsilon^2 = 0.01$	$\sigma_\epsilon^2 = 0.5$	$\sigma_\epsilon^2 = 2$	$\sigma_\epsilon^2 = 0.01$	$\sigma_\epsilon^2 = 0.5$	$\sigma_\epsilon^2 = 2$	$\sigma_\epsilon^2 = 0.01$	$\sigma_\epsilon^2 = 0.5$	$\sigma_\epsilon^2 = 2$
SQUARED BIAS									
PartMon	0.0672	0.0764	0.1090	0.0241	0.0275	0.0392	0.0352	0.0401	0.0572
NNet	0.0065	0.0102	0.0171	0.0047	0.0074	0.0124	0.0042	0.0066	0.0111
PMonLin	0.1324	0.1326	0.1331	0.0952	0.0953	0.0957	0.1170	0.1172	0.1176
VARIANCE									
PartMon	0.0035	0.0604	0.2199	0.0024	0.0403	0.1466	0.0008	0.0141	0.0513
NNet	0.0087	0.1473	0.6510	0.0024	0.0404	0.1783	0.0016	0.0278	0.1228
PMonLin	0.0006	0.0245	0.0918	0.0002	0.0102	0.0381	0.0001	0.0053	0.0197
MINIMUM MSE									
PartMon	0.0707	0.1368	0.3289	0.0265	0.0677	0.1858	0.0361	0.0542	0.1085
NNet	0.0152	0.1575	0.6680	0.0071	0.0477	0.1907	0.0059	0.0344	0.1339
PMonLin	0.1330	0.1571	0.2249	0.0954	0.1055	0.1337	0.1171	0.1224	0.1373

- However, the flexible nature of neural networks and the random initialization of the network weights lead to higher variances across different runs compared with the other two approaches–especially for small or noisy data sets.

- The flexibility of the mixture modeling employed by our approach for partial monotonicity gives considerably smaller squared bias than the partially linear monotone models across data sets with different number of points and noise levels; however, this leads to higher variances of the models.

- Standard neural networks give considerably smaller MSE than the two partially monotone models, for data sets with small noise level ($\sigma_\epsilon^2 = 0.01$). For larger data sets ($N = 150$ or $N = 250$) and moderate noise level ($\sigma_\epsilon^2 = 0.5$), neural networks produce only slightly lower MSE that our approach for partial monotonicity.

- For very noisy data sets with a small number of points ($N < 250$), the minimum MSE is achieved by the partially monotone linear models. This indicates that the other two flexible models overfit the data (by fitting the noise inherent in it). However, for very noisy data sets ($\sigma_\epsilon^2 = 2$) with more data points, the lowest MSE is obtained by our approach for partial monotonicity, which indicates that the true relationship between the dependent and independent variables can be

Table 5.5: Architectures of Sill networks and standard neural networks for which the minimum MSE is obtained by the models in Experiment 1 with partially monotone problems

Approach for partial monotonicity (groups × planes)				Neural networks with weight decay (hidden nodes–weight decay)			
Noise level	Number of points			Noise level	Number of points		
	50	150	250		50	150	250
$\sigma_\varepsilon^2 = 0.01$	3×2	3×2	3×2	$\sigma_\varepsilon^2 = 0.01$	9–0.000001	9–0.000001	9–0.000001
$\sigma_\varepsilon^2 = 0.5$	3×4	3×4	3×4	$\sigma_\varepsilon^2 = 0.5$	3–0.000001	3–0.000001	3–0.000001
$\sigma_\varepsilon^2 = 2$	3×4	3×4	3×4	$\sigma_\varepsilon^2 = 2$	3–0.000001	3–0.000001	3–0.000001

captured in case of a sufficient number of points.

- Standard neural networks perform remarkably bad on small and very noisy data sets. For data sets with $N = 50$, the minimum MSE is almost twice as big as that obtained by our method and it is three times bigger than that obtained by the partially monotone linear models. The explanation is that standard neural networks do not use any prior knowledge about the partial monotonicity of the true function.

Besides the accuracy, we also study the architectures of Sill and standard networks with the minimum MSE; see Table 5.5.

The results show that for all data sets our approach for partial monotonicity reaches the minimum MSE with Sill networks with three groups; for noisier data, however, it requires more hyperplanes to find a close approximation. In contrast, standard neural networks with nine hidden nodes lead to the minimum MSE for data sets with small noise levels, whereas for noisier data three hidden nodes suffice.

Finally, we checked the number of clusters found by our approach for partial monotonicity in each run. Given that the non-monotone variable has been generated by two normal distributions, we expect the approach to find these two distributions. It turned out that our approach indeed detects two clusters in all runs with data sets with 50 and 250 points–irrespective of the noise level. Note that the noise level indirectly affects the clustering procedure through the computation of the weights $\boldsymbol{\alpha}$ in (5.9). For data sets with 150 points, the approach detects ten clusters with respect to the non-monotone variable. A possible explanation could be that once generated,

the non-monotone variable is fixed for a data set with a certain number of points; so, it is very likely that the clustering procedure applied in our approach tends to find the same number of clusters across the runs.

Experiment 2. For our second experiment we first generate five vectors of N points as follows:

- x^1, x^2, x^3 are drawn from the uniform distribution on [0,1],
- x^4 is a composition of two sub-vectors each with $N/2$ points drawn from the two Gaussian distributions $\mathcal{N}(0.02, 0.05)$ and $\mathcal{N}(0.08, 0.05)$, and
- x^5 is drawn from $\mathcal{N}(0.5, 0.1)$.

Next, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$ we generate a vector $\ell_{\mathbf{x}}$ by

$$\ell_{\mathbf{x}} = 3 + \sin\left(\frac{\pi}{2} x^1 x^2 x^3\right)\big(2 + \sin(2\pi x^4 x^5)\big) + \epsilon, \tag{5.12}$$

Hence, $\mathbf{x}^m = (x^1, x^2, x^3)$ and $\mathbf{x}^{nm} = (x^4, x^5)$; $D = (\mathbf{x}^m, \mathbf{x}^{nm}, \ell_{\mathbf{x}})$ is a data set of N points.

Analogously to Experiment 1, we apply three methods: (i) our approach for partial monotonicity, (ii) standard neural networks with weight decay, and (iii) partially monotone linear models. We compare again the three methods by using the same factors with their levels in Table 5.2, and the design in Table 5.3. A summary of the results is given in Table 5.6.

The following conclusions can be drawn:

- Neural networks lead to more accurate models in terms of MSE for data sets with low noise compared with the partially monotone models. Their variances and thus MSE, however, increase considerably for noisier data; although the weight decay is used as a regularization method, overfitting occurs–a common problem in the application of neural networks. The incorporation of monotonicity constraints in the partially monotone models helps to overcome this overfitting problem–capturing the true relationships in the target function.
- For considerably noisy data, especially with fewer data observations, partially monotone linear models tend to give smaller MSE: despite

Table 5.6: Minimum MSE obtained by the three methods in Experiment 2 with partially monotone problems

Method	50 points			150 points			250 points		
	$\sigma_\varepsilon^2 = 0.01$	$\sigma_\varepsilon^2 = 0.5$	$\sigma_\varepsilon^2 = 2$	$\sigma_\varepsilon^2 = 0.01$	$\sigma_\varepsilon^2 = 0.5$	$\sigma_\varepsilon^2 = 2$	$\sigma_\varepsilon^2 = 0.01$	$\sigma_\varepsilon^2 = 0.5$	$\sigma_\varepsilon^2 = 2$
SQUARED BIAS									
PartMon	0.0288	0.0705	0.1290	0.0164	0.0400	0.0731	0.0106	0.0260	0.0476
NNet	0.0094	0.0113	0.0236	0.0037	0.0103	0.0214	0.0036	0.0100	0.0209
PMonLin	0.1282	0.1293	0.1301	0.0841	0.0846	0.0853	0.0799	0.0804	0.0811
VARIANCE									
PartMon	0.0249	0.1369	0.4641	0.0071	0.0389	0.1320	0.0045	0.0249	0.0845
NNet	0.0093	0.2207	1.0666	0.0059	0.0858	0.4149	0.0033	0.0480	0.2320
PMonLin	0.0013	0.0572	0.2378	0.0004	0.0187	0.0771	0.0003	0.0114	0.0468
MINIMUM MSE									
PartMon	0.0537	0.2074	0.5931	0.0235	0.0789	0.2051	0.0151	0.0509	0.1321
NNet	0.0187	0.2320	1.0902	0.0096	0.0961	0.4363	0.0069	0.0580	0.2529
PMonLin	0.1295	0.1865	0.3679	0.0845	0.1033	0.1624	0.0802	0.0918	0.1279

their higher bias, they have much smaller variance across different runs compared with the more flexible counterpart methods.

- Our approach for partial monotonicity leads to models that are more accurate and with low variance for data sets with a moderate noise level and a larger number of data points.

The architectures of Sill and standard neural networks with minimum MSE are reported in Table 5.7. On one hand, Sill networks with three groups and three hyperplanes achieve the best performance–irrespective of the noise level and the size of the data sets. On the other hand, the architectures of standard neural networks vary considerably over different types of data sets. So, our approach dependents less on the network's architecture; standard neural networks are more sensitive to the noise level, the size of the data, and the number of independent variables.

Finally, we again check the number of clusters detected in our approach across the runs; see Table 5.8. In contrast to Experiment 1, for the different types of data sets within the 100 data samples, our approach now finds different numbers of clusters. Because we now have two non-monotone variables drawn from different distributions, the noise level and the size of the data set would affect the clustering procedure for determining the right number of data subsets. Given our data generating procedure, we expect that our approach finds two clusters; their centroids are vectors containing

Table 5.7: Architectures of Sill networks and standard neural networks for which the minimum MSE is obtained by the models in Experiment 2 with partially monotone problems

Approach for partial monotonicity (groups × planes)				Neural networks with weight decay (hidden nodes–weight decay)			
Noise level	Number of points			Noise level	Number of points		
	50	150	250		50	150	250
$\sigma_\varepsilon^2 = 0.01$	3 × 3	3 × 3	3 × 3	$\sigma_\varepsilon^2 = 0.01$	3–0.000001	9–0.000001	9–0.000001
$\sigma_\varepsilon^2 = 0.5$	3 × 3	3 × 3	3 × 3	$\sigma_\varepsilon^2 = 0.5$	15–0.00001	15–0.00001	15–0.00001
$\sigma_\varepsilon^2 = 2$	3 × 3	3 × 3	3 × 3	$\sigma_\varepsilon^2 = 2$	3–0.00001	3–0.00001	3–0.00001

Table 5.8: Number of clusters found by the approach for partial monotonicity at each run

	Number of points		
	50	150	250
Noise level	Number of clusters – Percentage from 100 runs		
$\sigma_\varepsilon^2 = 0.01$	2 - 68 % 7 - 1 % 8 - 3 % 9 - 9 % 10 - 19 %	2 - 89 % 4 - 1 % 5 - 1 % 8 - 5 % 10 - 4 %	2 - 74 % 3 - 18 % 5 - 1 % 7 - 5 % 9 - 1 % 10 - 1 %
$\sigma_\varepsilon^2 = 0.5$	2 - 55 % 4 - 1 % 5 - 1 % 7 - 2 % 8 - 1 % 9 - 11 % 10 - 29 %	2 - 54 % 4 - 7 % 5 - 6 % 6 - 4 % 7 - 4 % 8 - 2 % 9 - 9 % 10 - 14 %	2 - 59 % 3 - 21 % 4 - 1 % 5 - 6 % 6 - 4 % 7 - 4 % 8 - 1 % 9 - 1 % 10 - 3 %
$\sigma_\varepsilon^2 = 2$	2 - 68 % 7 - 1 % 9 - 9 % 10 - 22 %	2 - 44 % 4 - 5 % 5 - 7 % 6 - 1 % 7 - 8 % 8 - 8 % 9 - 18 % 10 - 9 %	2 - 71 % 3 - 13 % 4 - 5 % 5 - 1 % 6 - 4 % 7 - 2 % 9 - 3 % 10 - 1 %

two values corresponding to the means used to generate the non-monotone variables. The results show that for all types of data sets, our approach indeed detects two clusters most; the percentage of two clusters detected is

above 50%, except for noisy data with 150 observations. Not surpisingly, our approach can better find the right number of (two) clusters for less noisy data sets, irrespective of the number of observations. Of course, larger data sets provide more information; indeed, our results show that our approach tends to detect the correct number of clusters: in more than 80% of the runs, it finds two or at most three clusters for data sets with 250 observations with different noise levels.

The results from both simulation studies indicate that there is no method that is superior in all the cases. Depending on the size of and the noise inherent in the data set, any of the three methods can achieve the best performance. In practice, of course, the data generating process is unknown, so we cannot determine beforehand which is the most appropriate method to model the data. However, we expect that in real cases our approach for partial monotonicity and standard neural networks would outperform partially monotone linear models as the former are more flexible. Furthermore, due to the monotonicity constraints our approach would lead to more stable models than the models derived from the unconstrained standard neural networks. The results from the following case studies confirm our expectations.

5.3.3 Real case studies

In this section we present the results for our approach in two real case studies, namely abalone age prediction (classification problem) and Den Bosch house pricing (regression problem). We apply the approach for partial monotonicity with the modifications made for the simulation studies, and we again compare it with standard neural networks with weight decay and partially monotone linear models.

A. Abalone age prediction

The abalone shellfish data set is publicly available at the UCI Repository of machine learning databases (Newman et al., 1998). It has been used as a benchmark to which various machine learning techniques have been applied (Waugh, 1995; Clark et al., 1996; Abdelbar, 1998). The data were originally collected by an agency in the Australian state of Tasmania for ongoing research purposes (Nash et al., 1994). The objective is to predict the age of abalone shellfish from eight physical measurements. Determining the age of abalone in the laboratory needs much time and labor because it requires

Table 5.9: Definition of the variables for the abalone data

Symbol	Definition
SEX	Male, Female, and Infant
LENGTH	Longest shell measurement (mm)
DIAMETER	Perpendicular to length (mm)
HEIGHT	With meat in shell (mm)
WHOLE WEIGHT	Whole abalone (grams)
SHUCKED WEIGHT	Weight of meat (grams)
VISCERA WEIGHT	Gut weight after bleeding (grams)
SHELL WEIGHT	After being dried (grams)

cutting the shell through the cone, staining it, and counting the number of rings through a microscope. Therefore, faster prediction can be done based on physical measurements that are easily obtained, namely the eight measurements (attributes) in Table 5.9. We transformed the nominal-valued sex attribute into continuous-valued by assigning the values of 0.1 for infant, 0.2 for male, and 0.3 for female. We also apply a simple transformation on the attributes WHOLE WEIGHT and SHUCKED WEIGHT to guarantee that all inputs are in the range [0, 1].

The dependent variable is the number of rings (age is easily computed by adding 1.5 to the number of rings); it has 28 values in the data set, ranging from 1 to 29 (28 is missing). Thus, these data can be treated as a regression or a classification problem. In earlier studies, the response variable has been discretized into three classes (age-groups): 1-8, 9-10, 11 or more. Here we adopt the same transformation procedure, and consider the abalone age prediction as a classification problem.

The original data consist of 4 177 observations (no missing attribute values). In our study, we take a random sample of 292 observations, which is 7% of the full data set. The stratified sample is taken such that infants, males, and females, as well as the three classes are represented in the same proportions as in the full data.

In these sampled data, we expect that the age of abalone depends monotonically on some of the measurements, but not on all. For example, if we consider the sex attribute, the discrimination between males and females is not expected to be monotone with age. Furthermore, with the abalone maturity, the abalone age may not necessarily have monotone relationships with some of the other physical measurements. To check for which attributes

Table 5.10: Degree of monotonicity of the abalone data

Removed variable(s)	Comparable pairs	DgrMon
SEX	33708	0.9057
SEX, WHOLE WEIGHT	33715	0.9057
SEX, SHUCKED WEIGHT	34683	0.9069
SEX, WHOLE WEIGHT, SHUCKED WEIGHT	34789	0.9070

monotonicity with age holds, we conduct a test. This is done by using a measure for the degree of monotonicity (DgrMon) of data, namely the fraction of monotone pairs of all comparable pairs in the data. Although the values assigned to the attribute SEX are numeric they do not imply any ordering; so does not make sense to use this attribute in the test for monotone relationships. Therefore, the measure for the degree of monotonicity is computed for the original data without the attribute SEX and for the data sets obtained after removing SEX and one or more of the other variables.

Table 5.10 shows that the removal of SEX, WHOLE WEIGHT and SHUCKED WEIGHT leads to a higher number of monotone pairs in the increased number of comparable pairs; the individual removal of the other attributes, not shown in the table, leads to a decrease in the degree of monotonicity compared with the original data (DgrMon $\leq$ 0.9052). These results indicate that we can consider the abalone data as a partially monotone classification problem where SEX, WHOLE WEIGHT and SHUCKED WEIGHT are the non-monotone variables, whereas the other attributes are the monotone variables.

Therefore, we apply our approach for partial monotonicity. Analogously to the simulation studies, we also use standard neural networks with weight decay and partially monotone linear models as benchmark methods for comparison. To obtain a sound assessment of the generalization capabilities of the model obtained, we randomly split the original data into a training data set with 219 observations (75%) and a test data set with 73 observations (25%). The former is used to build a model, whereas the latter is used to test the model performance measured by the misclassification error. The random partitioning of the data is repeated 20 times. We use nine combinations of parameters for the Sill networks (groups - 2, 3, 4; planes - 2, 3, 4) and standard neural networks (hidden nodes - 3, 6, 9; weight decay - 0.000001, 0.00001, 0.0001) to get better insight into the performance of the models. At each of the twenty runs, we select the model that gives the min-

Table 5.11: Estimated prediction errors of our approach for partial monotonicity (PartMon), standard neural networks with weight decay (NNets), and partially monotone linear models (PMonLin) for abalone data

Method	Minimum error			
	Min	Mean	Max	Variance
PartMon	0.27	0.31	0.36	0.000
NNet	0.25	0.32	0.37	0.002
PMonLin	0.30	0.35	0.38	0.000

imum misclassification error out of the nine parameter combinations with each method. Table 5.11 reports the minimal, mean, and maximal value and the variance of the estimated error across the runs.

The results show that our approach tends to be more accurate on average than standard neural networks and partially monotone linear models. Furthermore, both types of partially monotone models exhibit smaller variances upon repeated sampling than their unconstrained counterpart.

To check the significance of our results, we performed statistical tests. Since the test set in the experiments with the three methods is the same, we conduct three paired t-tests to test the null hypothesis that the models derived from one method have the same error as the models derived from the other methods against the one-sided alternatives. Their p-values are reported in Table 5.12. They show that the differences in errors obtained from our approach and standard neural networks are statistically insignificant at 5% and 10% significance level. Furthermore, the flexible nature of our approach and standard neural networks leads to models that can better capture the true relationships in the data. Hence these models have significantly smaller errors than the partially monotone linear models.

Table 5.11 suggests that the differences between the variances of the partially monotone models and standard neural networks are significant, which is also confirmed by the p-values of the two F-tests with 19 degrees of freedom, namely 0.2% and 0.1%; the difference between the variances of the models derived from our approach and from partially monotone linear models is statistically insignificant (p-value is 42.1%).

We also compare the architectures of the networks for which the minimum MSEs are obtained by the algorithm for partial monotonicity and standard neural networks. In two (out of twenty) runs, the minimum error in our

Table 5.12: p-values of paired t-tests and one-sided confidence intervals for the difference in error means in the abalone case study

Indicator	p-value	Confidence intervals	
		95%	90%
Minimum error (PartMon–NNet)	12.8%	[-1, 0.005)	[-1, 0.001)
Minimum error (PartMon–PMonLin)	0.0%	[-1, -0.024)	[-1, -0.027)
Minimum error (NNet–PMonLin)	0.2%	[-1, -0.011)	[-1, -0.014)

approach is achieved for Sill networks with four groups and four hyperplanes in each group. Also, six times, Sill networks with twelve first-layer nodes led to minimum MSE. In the other twelve runs the minimum is obtained for Sill networks with at most nine nodes in the first hidden layer. For the standard networks, the minimum MSE is obtained ten times with three hidden nodes, six times with six hidden nodes, and four times with nine hidden nodes. These results show that in majority of cases the best prediction accuracy is produced by networks with similar architectures in both algorithms.

Another interesting observation is the variance of the error across different Sill and standard network architectures within a run; see Table 5.13.

Table 5.13: Variance across different network architectures within a run for the abalone data

Method	Variance within a run		
	Min	Mean	Max
PartMon	0.0013	0.0032	0.0054
NNet	0.0003	0.0083	0.0496

The results show that our approach produces models with a lower variance on average across various network architectures compared with standard neural networks with weight decay. In fact, in two out of the twenty runs, standard neural networks with three and six hidden nodes produced models with 100% misclassification rate. This result implies that the models derived from the neural networks have high variability and thus higher dependence on the network architecture.

B. Den Bosch house pricing

The data used in the second case study consist of 119 observations on houses in the Dutch city of Den Bosch. Eleven independent variables describe the characteristics of a house; see Table 5.14. The dependent variable is the house price; for computational convenience it was transformed by taking its logarithm.

Table 5.14: Definition of the variables for the Den Bosch housing data

Symbol	Definition
DISTR	Type of district, 4 categories ranked from bad to good
AREA	Total house area including garden
RM	Number of bedrooms
TYPE	House type, 6 categories, ranked from flat to villa
VOL	Volume of the house
GARD	Type of garden, 4 categories ranked from bad to good
GARG	1-no garage, 2-normal garage, 3-large garage
FLOORS	Number of floors
YEAR	Year of building
X-DIST	Horizontal map location
Y-DIST	Vertical map location

This data set has been used in previous studies (Daniels and Kamp, 1999; Potharst and Feelders, 2002), which deal with incorporating total monotonicity constraints in data mining algorithms. Therefore, in those studies, YEAR, X-DIST and Y-DIST were dropped from the data as variables, because monotone relationships with the house price do not hold. Furthermore, we suspect that the monotonicity dependency on FLOORS does not hold; for example, some expensive houses (such as villas) may have only one floor, whereas cheaper houses may have three floors. Therefore, we conduct the same test as in the abalone case study to check for which variables in the housing data the monotonicity assumption holds; see Table 5.15.

Considering first the individual removal of the four variables, we see that the degree of monotonicity increases most after leaving out Y-DIST and YEAR. Therefore, in the next step these two variables are removed from the data, which leads to an additional increase in the number of monotone pairs. Finally, it is obvious that the maximum degree of monotonicity (namely 0.9659) is achieved after leaving out the four variables, which shows that their dependency with the house price is not necessarily monotone. As a result

Table 5.15: Degree of monotonicity of the Den Bosch housing data

Removed variable(s)	Comparable pairs	DgrMon
- (original data)	314	0.9140
FLOORS	331	0.9184
X-DIST	412	0.9199
Y-DIST	634	0.9495
YEAR	1073	0.9553
Y-DIST,YEAR	1534	0.9615
FLOORS,Y-DIST,YEAR	1620	0.9630
X-DIST,Y-DIST,YEAR	2217	0.9648
FLOORS,X-DIST,Y-DIST,YEAR	2345	0.9659

FLOORS, X-DIST, Y-DIST and YEAR are considered to be non-monotone variables, whereas the other are monotone.

Using this knowledge about the (non)-monotone relationships in the housing data, we apply our approach for partial monotonicity. Analogously to the experiment with the abalone data, we randomly split the original housing data into training data of 89 observations (75%) and test data of 30 observations (25%). The random partition of the data is repeated 20 times. The performance of the models is measured by computing the mean-squared error (MSE). We use again nine combinations of parameters for Sill networks (groups - 2, 3, 4; planes - 2, 3, 4) and standard neural networks (hidden nodes - 5, 13, 20; weight decay - 0.000001, 0.00001, 0.0001). Note that in this case study the standard neural networks have more hidden nodes compared to that used in the abalone case study: the housing data contain more independent variables. At each of the twenty runs, we select the model that obtains the minimum MSE out of the nine parameter combinations with each method. Table 5.16 reports the minimal, mean and maximal value and the variance of the estimated MSE across the runs.

The results show that, in general, the models generated by our approach are more accurate than those models generated by standard neural networks and partially monotone linear models. Furthermore, the variation in the minimum MSE across runs is considerably smaller for our approach than for the benchmark methods, as are the differences between the maximum and minimum value of MSE in Table 5.16.

We again conduct three paired t-tests to check the significance of the differences in the errors. The p-values obtained from the tests and the con-

Table 5.16: Estimated prediction errors of the approach for partial monotonicity (PartMon), standard neural networks with weight decay (NNets), and partially monotone linear models (PMonLin) for Den Bosch housing data

Method	Minimum MSE			
	Min	Mean	Max	Variance
PartMon	0.011	0.016	0.038	0.0000
NNet	0.012	0.020	0.059	0.0001
PMonLin	0.016	0.022	0.053	0.0001

Table 5.17: p-values of paired t-tests and one-sided confidence intervals for the difference in error means in the Den Bosch house pricing case study

Indicator	P-value	Confidence intervals	
		95%	90%
Minimum MSE (PartMon–NNet)	0.5%	(-∞, -0.002)	(-∞, -0.002)
Minimum MSE (PartMon–PMonLin)	0.0%	(-∞, -0.004)	(-∞, -0.005)
Minimum MSE (NNet–PMonLin)	5.1%	(-∞, 0.000)	(-∞, -0.000)

fidence intervals at 95% and 90% are reported in Table 5.17. The results show that our approach leads to models with significantly smaller errors than the errors of the models derived from the standard neural networks and partially monotone linear models. The error difference between standard neural networks and partially monotone linear models is not significant at 5% significance level.

In addition F-tests for the differences between the variances of the models are performed. The p-values of 1.0% and 3.3% obtained from the tests show that our approach has significantly lower variances than standard neural networks and partially monotone linear models, respectively. These results indicate that the models derived from our approach are more stable upon repeated sampling. The differences between the variances of the standard neural networks and partially monotone linear model is insignificant: p-value is 29.8%.

Other results, not reported in Tables 5.16 and 5.17, concern the architectures of the networks for which the minimum MSEs are obtained by the algorithm for partial monotonicity and standard neural networks. In five

Table 5.18: Variance across different network architectures within a run for the Den Bosch housing data

Method	Variance within a run		
	Min	Mean	Max
PartMon	0.00000	0.00003	0.00018
NNet	0.00008	0.00120	0.00503

(out of twenty) runs, the minimum error in our approach is achieved for Sill networks with four groups and four hyperplanes in each group. Also, six times, Sill networks with twelve first-layer nodes led to minimum MSE. In the other nine runs the minimum is obtained for Sill networks with at most nine nodes in the first hidden layer. For the standard networks, the minimum MSE is obtained twelve times with twenty hidden nodes, and seven times with thirteen hidden nodes. In other words, only one time a standard neural network with five hidden nodes produced the best prediction accuracy. These results clearly demonstrate important advantages of our algorithm compared with standard neural networks: (i) our divide-and-conquer approach employs smaller networks that can be trained on smaller subsets of data, and (ii) thus, the execution time and the computational effort can be considerably reduced.

The variances of MSE across different Sill and standard network architectures within a run are reported in Table 5.18.

The results clearly show that our approach has considerably lower variances across various network architectures compared with standard neural networks. This clearly indicates that our models are more stable and less dependent on the network architecture.

Discussion of results

We can draw the following conclusions about the performance of the three methods on real data:

1. Our approach has comparable performance to standard neural networks. However, due to the preservation of monotonicity in the former, our models have (i) a significantly smaller variance upon repeated sampling, and (ii) less variability over different network architectures for a

fixed data sample. Hence, our models would be a preferable building tool in partially monotone prediction problems.

2. The flexible nature of our approach leads to models that are significantly more accurate than the simple partially monotone linear models. The variances of both types of models are comparable.

Furthermore, for data sets with a larger number of independent variables, our approach has an important advantage over standard neural networks– namely, our divide-and-conquer strategy employs smaller networks on subsets of data, and thus the good performance of our models is obtained faster and with less computational efforts.

5.4 Conclusion

In this chapter we considered prediction problems with partial monotonicity, i.e., the response variable depends monotonically on some but not on all predictor variables. We derived an approach for partially monotone models, which are a convolution of weight functions (kernels) based on non-monotone variables and Sill (monotone) networks built only on the monotone variables. Simulation with artificial data and real case studies showed that the overall performance of our approach is better than standard neural networks and partially monotone linear models. First our models outperform the partially monotone linear models in terms of accuracy. Compared to standard neural networks our approach achieves comparable accuracy but significantly smaller variance upon repeated sampling. Hence, the incorporation of partial monotonicity constraints leads not only to models that agree with the decision maker's expertise but also to more stable models. This result supports our third hypothesis stated in the introduction of this thesis (Section 1.4) that for partially monotone problems partially monotone models have superior predictive performance to non-monotone models.

Chapter 6

Conclusions and future research

6.1 Conclusions

The field of *data mining* has emerged from the need for efficient and effective manipulation of large amounts of data; it turns these data into novel valuable knowledge for decision-making. This knowledge discovery is a complex process involving several steps; the successful implementation of any data mining system depends on the successful outcome of each step. By "success", we mean that the results (models) obtained from a data mining process are accurate, comprehensible, and easy to understand by the end user; moreover, they comply with business policies and the expert knowledge encoded in the domain. In practice large data sets are available, but often the models derived purely from standard data mining methods do not have the desired properties, i.e., they contradict the underlying domain knowledge.

To overcome this problem, it is crucial to integrate domain (expert) knowledge into a data mining process. This integration can guide and facilitate the knowledge discovery process by restricting the search space of possible outcomes, thus obtaining not only valid but also faster decisions. This is especially important in *prediction problems*–one of the main data mining tasks ubiquitous in practice–where the objective is to make accurate and plausible future predictions about a certain attribute of analyzed objects in a domain. This prediction is based on a set of other attributes describing the objects. Within the class of prediction problems, we distinguished two subclasses: (i) classification problems where the dependent variable is discrete, and (ii) regression problems where the dependent variable is continuous.

In this thesis, our main objective is to study the incorporation of *monotonicity constraints* as a special class of domain knowledge, into data mining models for both types of prediction problems. The monotonicity constraint simply states that the increase in an input must not lead to a decrease in the output, all other inputs being equal.

With respect to monotonicity, we defined two types of prediction problems: *monotone* and *partially monotone problems*. In monotone prediction problems, we assume that the dependent variable is generated by a function monotone in all independent variables. In partially monotone prediction problems, the assumption is slightly weaker, namely the generating function is monotone on some of the independent variables but not on all. Chapters 2, 3, and 4 deal with monotone problems; Chapter 5 deals with partially monotone problems.

Our research has solid theoretical foundations. It also develops practical computational methods applicable to a wide range of domains. To validate and demonstrate the performance of our methods, we used a number of simulation and real case studies. In some of these studies, we used traditional data mining approaches as benchmark methods resulting in a more reliable assessment of our results.

Research questions and answers

We decomposed our main objective of incorporating monotonicity constraints in data mining for prediction, into two more specific goals corresponding to the two steps in the data mining process–data preprocessing and modeling– where monotonicity can be enforced:

Research objective - 1 Preprocessing (transforming) data such that they obey monotonicity constraints before using the data to build monotone decision models.

Research objective - 2 Enforcing monotonicity in data mining models based on decision trees and neural networks for prediction tasks.

From these two research objectives, we derived a number of research questions. Below we summarize the answers provided by this thesis. Related to *Research objective - 1* we have:

- *How can we measure the degree of monotonicity of a data set?*

Before starting the modeling of the data, the analyst must be sure that the data are suitable for the problem at hand. For monotone prediction problems, this means that the data exhibit monotonicity properties. To check this property, we proposed a novel procedure for testing the degree of monotonicity of the real data in Section 2.3. The procedure is based on two measures for monotonicity: the fraction of monotone pairs and the number of monotone points in the data. These measures are computed from the real data and from benchmark data obtained by taking the same set of independent variables as in the real data set and adding a random permutation of the original labels. In addition, we developed a statistical test to check the significance of the difference between both the observed and the benchmark measures–a significant difference indicates that the data exhibit monotone relationships; otherwise, the data are considered non-monotone. Compared with previous testing approaches, the main advantages of our procedure are that it does not depend on the data generating process and does not require any pre-modeling of the data.

- *How can we transform a data set from non-monotone into monotone?*

  The transformation of non-monotone into monotone data guarantees that the data are consistent, comply with human expertise and domain knowledge, and thus, they are a reliable source for data analysis. To obtain monotone data we proposed a greedy algorithm for relabeling, described in Section 2.4. It assumes that only a few data observations are inconsistent; by appropriately changing their labels (the so-called relabeling process), we can obtain monotone data. Although the algorithm does not provide a unique solution, it guarantees that after a small number of steps the data are monotone. A comparison with a network flow algorithm shows that our number of label changes is the same or very close to the minimum possible number of points that need to be relabeled to get monotone data. Furthermore, compared with previous approaches dealing with monotone data transformation, the main advantage of our approach is that it preserves the majority of the original labels, so the modified data remain close to the real data.

With respect to *Research objective - 2*, our answers to the research questions are as follows.

- *How can we build monotone models?*

  To build monotone models, we used decision trees and neural networks as standard modeling techniques for data mining. For the purposes of our research, we did not develop new approaches but modified and extended two existing methods such that (i) their implementation is improved and (ii) they can be applied to both classification and regression problems. In Section 3.3, we described a tree-based approach proposed by Feelders (2000), which has a similar construction scheme to the standard CART method for building decision trees. The only difference is that our method checks whether the trees derived are monotone. In Chapter 4, we discussed the construction of monotone models based on neural networks. Through a counter-example in two dimensions we showed that two-layer neural networks cannot adequately approximate any function with more than one input. Sill (1998) proves that three-layer monotone networks with combinations of minimum and maximum operators over linear functions do have universal function approximation capabilities. Therefore, we extended his approach to build monotone models in our study.

- *How can we build partially monotone models?*

  For partially monotone problems, we need to build models that preserve the monotone relationships between the dependent and independent variables. In the literature, these problems have not been extensively considered. In Chapter 5 we therefore proposed a novel approach for building partially monotone models. The approach is based on the mixture modeling framework, where Sill (monotone) networks are convoluted with weight (kernel) functions. We showed that the estimator obtained from our approach has universal function approximation capabilities. Furthermore, simulation studies (with artificial data) and case studies (with real data) show that due to its flexibility our approach outperforms partially monotone linear models in terms of accuracy. Our approach performs comparably to standard neural networks, but the former has less variance and thus produces more stable models.

  As an alternative approach for building partially monotone models, we considered three-layer neural networks with combinations of minimum and maximum operators over linear functions (Appendix A).

The weights on the connections to the monotone variables are constrained to be non-negative. We also prove that this type of network can arbitrary well approximate any partially monotone function.

Finally, we defined the following three hypotheses:

Hypothesis - 1 For monotone problems, monotone models have superior predictive performance to non-monotone models.

This hypothesis has been supported by previous studies dealing with monotonicity. For example, Feelders (2000) and Potharst and Feelders (2002) show that for monotone classification problems with real data, monotone trees have comparable accuracy but much smaller size than non-monotone trees. Therefore, the former are easier to interpret by the human decision-makers. Our experiments with regression trees in Chapter 3 also support this finding, which makes the conclusions more sound. Furthermore, Sill (1998) shows that his type of three-layer monotone networks outperform standard neural networks in terms of accuracy and stability on real data for classification problems. To provide generalization of this finding, we apply both types of networks on artificially generated monotone data for regression problems. Our results again confirm the hypothesis that the monotone networks lead to models with better prediction accuracy and smaller variances compared with the models derived from the non-monotone standard networks.

Hypothesis - 2 For monotone problems, monotone models derived from monotone data (i.e., data obtained after the transformation) outperform monotone models derived from the original data, i.e., the former are more accurate and their variance on new data is lower.

To test this hypothesis we used the monotone models built through decision trees in Chapter 3 and neural networks in Chapter 4. We conducted two real case studies representing a classification problem (bond rating) and a regression problem (house pricing). We derived monotone models from both the monotone (cleaned) data and the original data; we compared the models' performance. The results confirmed our hypothesis in the following sense: (i) monotone models derived from the cleaned data have significantly better accuracy than those derived from the original data, and (ii) upon repeated sampling the variances of the former tends to be lower that those of the latter. In other words, resolving the inconsistencies in data beforehand leads to more accurate, stable, and reliable models.

Hypothesis - 3 For partially monotone problems, partially monotone models have superior predictive performance to non-monotone models.

In simulation and real case studies, we used two types of partially monotone models–models derived from the approach for partial monotonicity and partially monotone linear models, which were compared with unconstrained standard neural networks. The results showed that our approach has comparable predictive accuracy to standard neural networks. However, due to the preservation of monotonicity in the former, our models have (i) a significantly smaller variance upon repeated sampling, and (ii) less variability over different network architectures for a fixed data sample. Hence, our models would be a preferable building tool in partially monotone prediction problems. In comparison with partially monotone linear models, unconstrained neural networks with their flexible nature lead to models with smaller errors on average but higher variances upon repeated sampling.

6.2 Future research

Although our research study provides answers to the main research questions posed in this thesis, there are a number of directions for future work in both theoretical and practical terms:

- *Benchmark monotonicity measures*: We considered two indicators to measure the degree of monotonicity of benchmark data: the expected value of the fraction of monotone pairs and the expected value of the number of monotone points. We derived an analytical expression for the first measure, whereas we empirically computed the second measure from generated benchmark data. Hence, the theoretical derivation of the latter remains an open research question.

- *Monotone models*: In this research we used decision trees and neural networks to build monotone models. It would be interesting to extend our study on derivation of monotone models by using other modeling techniques; examples are Bayesian networks, graphical models, decision rules, which have been successfully applied to various knowledge representation problems.

- *Sill (monotone) networks*: Additional simulation studies may give more insight into the problem of determining the appropriate number of

groups and hyperplanes depending on the task at hand; for example, in which cases does increasing the number of groups work better than increasing the number of hyperplanes? Another issue is the effect of interchanging the minimum and maximum operators in the hidden layers: how does it affect the performance of the models derived?

- *Test for partial monotonicity*: In totally monotone problems the monotonicity dependency between the response and predictor variables is explicitly defined. In partially monotone problems this dependency is more difficult to establish, due to the effect of non-monotone variables. Therefore the test for partial monotonicity we used in Chapter 5 is rather empirical, and thus is highly dependent on the data. In some cases, the results for the non-monotone relationships might be doubtful. Then, an alternative empirical strategy is to apply parallel modeling, namely partially monotone approaches on the whole data set, and monotone approaches on the subset of monotone variables only. The results can then be compared, and the analyst can decide which type of technique is more suitable for modeling the data. Of course, a better solution develops a theoretical test that leads to more reliable conclusions regarding the degree of (partial) monotonicity of the data at hand.

- *Algorithm for partial monotonicity*: The appropriate partitioning of data in the initial step of our algorithm plays a crucial role in the building of accurate models. Applying different clustering techniques–such as fuzzy clustering and Expectation-Maximization algorithm–may improve the performance of our partially monotone models. Furthermore, in our current algorithm we use the non-monotone variables to define clusters only. An advanced strategy is to consider these variables as "genuine" predictors in a data mining modeling technique (e.g., standard neural networks). Thus the effects of monotone and non-monotone variables can be combined to obtain better predictive performance of the models derived.

- *Partially monotone neural networks*: Although we proved the universal function approximation capabilities of partially monotone neural networks, it would be interesting to determine their performance on artificial and real data. Similarly to Sill networks and standard neu-

ral networks, we expect that setting the parameter values in partially monotone neural networks may be problematic in practice. Furthermore, a comparison with our approach for partial monotonicity can be useful to demonstrate the strengths and weaknesses of both approaches when solving partially monotone problems.

To conclude, this thesis clearly demonstrates that the incorporation of monotonicity constraints as domain knowledge into a data mining process can be beneficial for both research and practice. The results from our study are based on solid theoretical foundations and the development of computational methods. They show that enforcing monotonicity can considerably improve the knowledge discovery process by deriving more accurate, stable and plausible decision models. Our research also suggests several directions for future investigation on the problem for building monotone prediction models in data mining.

Appendix A

Network flow algorithm for making data monotone

We describe a polynomial-time network flow algorithm to make data monotone as an alternative of the greedy algorithm for relabeling (Section 2.4). The former guarantees finding the minimum number of points that need to be deleted. In Proposition 2.4.1 we showed that this number is equal to the minimum number of points that need to be relabeled to obtain monotone data.

The application of the network flow algorithm is compared with the greedy algorithm for relabeling for the two real case studies presented in Section 2.5. The results show that both algorithms find the same number of points that need to be deleted or relabeled to make the data monotone. Although the greedy algorithm does not guarantee finding the minimum number of label changes (see the example in Figure 2.4 on p. 56), we conjecture that in practice the greedy algorithm makes a number of label changes that is very close to the minimum.

A.1 Description

Our primal problem is how to make data monotone by removing as few points as possible. Instead we can solve the dual problem of finding the maximum number of points that participate in monotone relationships only. In graph theory, this problem is equivalent to the problem of finding the maximum number of non-adjacent vertices in an inconsistency graph, i.e., the graph

representing the non-monotone relationships in a data set (Rademaker et al., 2006). In order to discuss this problem we first introduce some concepts and definitions about graphs.

A *graph* consists of *vertices* (points), representing entities, and *edges* (links) connecting different vertices and representing relationships between these vertices. If the edges have a direction, then the graph is called *directed* or *digraph*; otherwise, the graph is *undirected*.

For the purpose of our study, we consider a special type of graphs that represents the partial order among entities, namely a *comparability graph*. Our description of comparability graphs follows that of Möhring (1985).

A *partial order* is a digraph $P = (V, R)$ with the vertex set V, and the edge set R that is a strict order relation on V, i.e., a transitive and asymmetric binary relation. Transitivity means that for any edges $ab \in R$, and $bc \in R$ there is an edge $ac \in R$. Asymmetry means that $R \cap R^{-1} = \emptyset$, where $R^{-1} = \{ba \mid ab \in R\}$ is the inverse relation of R.

Let $G(P) = (V, E)$ be an undirected graph assigned to P, where $E = R \cup R^{-1}$, i.e., two vertices are adjacent (connected) in G if they are comparable in P. Then, $G(P)$ is a comparability graph of P.

Furthermore, for any graph, a subset of its vertices is an *independent set* if no two vertices in the subset are adjacent. The *independence number* of a graph is the cardinality (size) of the *maximum independent set*. Finding the independence number of arbitrary graphs is known to be NP-complete problem (Garey and Johnson, 1979). However, this number can be determined in polynomial time for comparability graphs due to their associated transitive orientation and underlying structure.

The problem of finding the independence number in a comparability graph is related to the problem of finding the maximum monotone data subset because of the binary nature of monotone relationships. All the points (observations) that participate in non-monotone relationships are represented by vertices in a graph, whereas the non-monotone relationships themselves are represented by edges. Thus, we obtain the so-called *inconsistency graph*. Note that for any non-monotone relationships represented by the edges ab and bc, with $a < b < c$, there is a non-monotone relationship represented by edge ac. Hence, an inconsistency graph is transitive. Furthermore, recall from the introduction of Chapter 2 that every (non)-monotone relationship is comparable. So, an inconsistency graph is a comparability graph. In Theorem 1.25, Möhring (1985) shows that the independence num-

ber of a comparability graph equals the minimum flow value in a flow network that can be obtained by a simple transformation of the graph. Hence, our problem is finally reduced to finding this value.

A *flow network*–as defined by Swamy and Thulasiraman (1981)–is a connected directed graph that has no self-loops, and satisfies the following conditions:

1. There is only one vertex with no incoming edges; this vertex is called the *source* and is denoted by s.

2. There is only one vertex with no outgoing edges; this vertex is called the *sink* and is denoted by t.

3. Each directed edge $e = ab$ in the network is associated with a nonnegative real number called *capacity* or *weight* of the edge; it is denoted by $c(e)$. If there is no edge e directed between a and b, then we define $c(e) = 0$.

Hence, if we give directions on the edges from smaller to larger elements in a comparability graph, and then connect a source to all minimal elements, and a sink to all maximal elements, we obtain a flow network structure.

A flow $\mathtt{f}$ in a flow network is an assignment of a nonnegative real number $\mathtt{f}(e) = \mathtt{f}(ab)$ to each edge $e = ab$ such that the following three conditions are satisfied:

1. Capacity constraint: $\mathtt{f}(e) \leq c(e)$ for every edge e in the network.

2. Skew symmetry: $\mathtt{f}(ab) = -\mathtt{f}(ba)$.

3. Flow conservation: $\sum_b \mathtt{f}(ab) - \sum_b \mathtt{f}(ba) = 0$ for all $a \neq s, t$.

The value of a flow $\mathtt{f}$, denoted by $\mathtt{f}_{val}$, is defined as

$$\mathtt{f}_{val} = \sum_b \mathtt{f}(sb) = \sum_b \mathtt{f}(bt).$$

A flow $\mathtt{f}^*$ in a flow network is said to be *maximum* (*minimum*) if there is no flow $\mathtt{f}$ in the network such that $\mathtt{f}_{val} > \mathtt{f}^*_{val}$ ($\mathtt{f}_{val} < \mathtt{f}^*_{val}$). Hence, one of the most typical network flow problems encountered in practice is the *maximum flow problem*. An example is finding the maximum amount of oil that can

be pumped between an origin node and a destination node in an oil pipeline. More formally, for a flow network $N = (V', E')$, a maximum flow problem is defined as

$$\begin{aligned} \text{maximize} \quad & \mathtt{f}_{val} \\ \text{subject to} \quad & \sum_b \mathtt{f}(ab) - \sum_b \mathtt{f}(ba) = 0, \quad \forall a \neq s, t \\ & 0 \leq \mathtt{f}(e) \leq c(e), \quad e \in E', \end{aligned} \tag{A.1}$$

The minimization analogue of the maximum flow problem is the *minimum flow problem* formally defined as

$$\begin{aligned} \text{minimize} \quad & \mathtt{f}_{val} \\ \text{subject to} \quad & \sum_b \mathtt{f}(ab) - \sum_b \mathtt{f}(ba) = 0, \quad \forall a \neq s, t \\ & lc(e) \leq \mathtt{f}(e) \leq uc(e), \quad e \in E', \end{aligned} \tag{A.2}$$

where $lc(e)$ and $uc(e)$ are the lower and upper bounds for the flow through edge e.

In the next section, we discuss how to use a minimum network flow algorithm to transform non-monotone into monotone data.

A minimum network flow algorithm for making data monotone

Let $D = \left(\mathbf{x}^n, \ell_{\mathbf{x}^n}\right)_{n=1}^N$ denote the original data set, and let $Q(D)$ denote the set of all non-monotone points in D. As we have already noted, the points in $Q(D)$ induce a partial order P, since the non-monotone relations are transitive and asymmetric. Hence, we construct an inconsistency graph $G(P) = (Q(D), E)$ of P with the vertex set $Q(D)$ and the edge set $E = \{\mathbf{x}^i\mathbf{x}^j \mid (\mathbf{x}^i, \mathbf{x}^j) \text{ is a non-monotone pair}\}$, for $i \neq j$ and $1 \leq i, j \leq N$. Next–as described by Möhring (1985)–we transform the inconsistency graph to a flow network $N_P = (V', E')$ with $E' = \{\mathbf{x}^i\mathbf{x}^j \mid \mathbf{x}^i \leq \mathbf{x}^j \text{ and } \ell_{\mathbf{x}^i} > \ell_{\mathbf{x}^j}\}$. To $G(P)$ we add a source node s, a sink node t, edges $s\underline{\mathbf{x}}$ for all minimal vertices $\underline{\mathbf{x}}$, and edges $\overline{\mathbf{x}}t$ for all maximal vertices $\overline{\mathbf{x}}$. These edges and all edges $\mathbf{x}^i\mathbf{x}^j \in E'$ are assigned lower capacities of zero and upper capacities of $+\infty$. Furthermore, except for the source and the sink, all the vertices are assigned weights of one

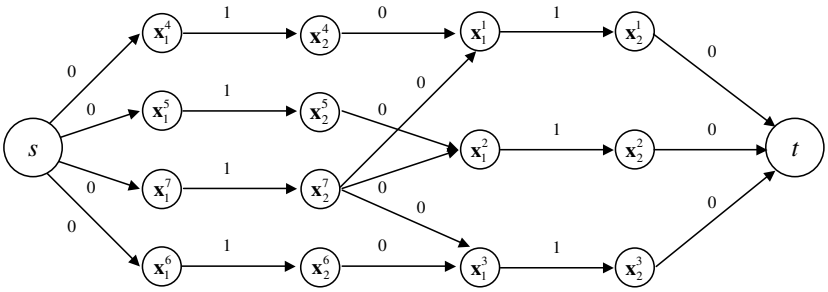


Figure A.1: Flow network based on the non-monotone data in Figure 2.4

corresponding to one point in the data. Since the network flow algorithms deal with capacities on the edges (not on the vertices), we replace each vertex $\mathbf{x} \in N_P$ by an edge $\mathbf{x}_1\mathbf{x}_2$ such that all incoming edges of $\mathbf{x}$ have $\mathbf{x}_1$ as their end point, and all outgoing edges of $\mathbf{x}$ have $\mathbf{x}_2$ as their starting point; the edge $\mathbf{x}_1\mathbf{x}_2$ is assigned lower capacity of one and a upper capacity of $+\infty$.

To illustrate this flow network construction, we consider the data depicted in Figure 2.4 (p. 56). The set of lines E connecting the points indicate the non-monotone relationships; they constitute an inconsistency graph $G(P) = (Q(D), E)$ of the partial order P on $Q(D)$. We have the set of non-monotone points $Q(D) = \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6, \mathbf{x}^7\}$ because all the points participate in non-monotone relationships. Figure A.1 depicts the flow network N_P associated with $G(P)$. Each of the seven points is represented by an edge with a lower capacity of one and a upper capacity of $+\infty$. The connections to the source and the sink, and the edges representing the non-monotone relationships are assigned lower capacities of zero and upper capacities of $+\infty$.

Now the question is what is the minimum cardinality set of points that need to be removed from $Q(D)$ to make D monotone? Alternatively, what is the maximum cardinality set of points that are in monotone relationships only, i.e., the maximum independent set M, $M \subset Q(D)$ in $G(P)$? This problem can be solved by finding the minimum flow value $\mathtt{f}_{val}^{min}$ in N_P. Furthermore, by the min-flow max-cut theorem (Ford and Fulkerson, 1962; Lawler, 1976) $\mathtt{f}_{val}^{min}$ equals the maximum capacity of an s,t-cut (or maximum cut) in N_P, i.e.,

$$\mathtt{f}_{val}^{min} = \max_{S,T} \left[\sum_{\substack{ab \in E' \\ a \in S, b \in T}} lc(ab) - \sum_{\substack{ab \in E' \\ a \in T, b \in S}} uc(ab) \right],$$

where S, T is an s, t-cut of $N_P = (V', E')$, i.e., $V' = S \cup T$, $S \cap T = \emptyset$, $s \in S$, $t \in T$, and where $lc(ab)$ and $uc(ab)$ denote the lower and upper capacity of edge $ab \in E'$.

Once the maximum cut is found, we can use it to derive the maximum independent set as shown by Möhring (1985). This is the set of vertices $\{\mathbf{x} \in Q(D) \mid \mathbf{x}_1 b \in E', \mathbf{x}_1 \in S, b \in T\}$. Finally, the set complement to the maximum independent set is the set of points that need to be deleted (relabeled) to get monotone data.

For the network flow in Figure A.1, we find $\mathtt{f}_{val}^{min} = 4$, i.e., the cardinality (independence number) of the maximum independent set M is 4. This set is obtained by the S, T-cut, where $S = \{s, \mathbf{x}_1^4, \mathbf{x}_1^5, \mathbf{x}_1^6, \mathbf{x}_1^7\}$, $T = V' \backslash S$, and $M = \{\mathbf{x}^4, \mathbf{x}^5, \mathbf{x}^6, \mathbf{x}^7\}$. Hence, we find that the set of points that need to be removed (or relabeled) to make D monotone is $Q(D) \backslash M = \{\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3\}$.

A.2 Implementation

The algorithmic complexity of finding a minimum flow and the associated maximum independent set in a flow network $N_P = (V', E')$ based on a comparability graph $G(P) = (V, E)$ is $O(|V'|^3) = O(|V|^3)$, where $|\cdot|$ denotes the size of the set (Even, 1979). For our problem $V = Q(D)$.

Now we discuss the implementation of the minimum flow problem for making data monotone, and its application to the two real case studies presented in Section 2.5. As we mentioned in the previous section, the minimum flow problem is a minimization analogue of the maximum flow problem; the latter is one of the most typical network flow problems encountered in practice, for which numerous efficient algorithms have been developed. In our study we use implementations by Prof. Dr. S. Iglin (National Technical University KhPI, Kharkiv, Ukraine) in MATLAB; the programs are freely available on the MATLAB Central web-site cited in the bibliography. We modified the original program to solve the minimum flow problem.

First (as described in the previous section) for a given data set, we build a flow network N_P based on the partial order P and inconsistency graph

$G(P)$ of the non-monotone relationships in the data. The lower capacities of the edges representing data points are ones; otherwise, they are zero. The upper capacities on all the edges are $+\infty$. Next we try to find the minimum flow in N_P by solving a network programming problem as defined in (A.2).

We applied our minimum flow algorithm to the bond rating and Moscow housing data to make the data sets monotone. The algorithm yielded the same numbers of points that need to be deleted (relabeled) as the greedy algorithm for relabeling, namely 28 for the bond rating data, and 54 for the Moscow housing data. Furthermore, we compared the sets of points found by both algorithms. It turned out that 79% of the points are the same for the bond rating data and 67% for the Moscow housing data. These are mostly the points relabeled at the beginning of the greedy algorithm, i.e., the points whose relabeling resolves most inconsistencies. This means that both algorithms correctly detect the "outliers", i.e., the points that violate the monotonicity constraint most.

We also tested our minimum flow algorithm on the artificial example depicted in Figure A.1. The algorithm found the correct maximum independent set and its complement set of points that need to be deleted (relabeled) to make the data monotone.

Remark. The results from the experiments with the greedy algorithm for relabeling and the minimum flow algorithm show that the problem of minimizing the number of label changes has several optimal solutions. Thus, we can use a second criterion for optimization such as the sum of the absolute values of all changes. The problem with both criteria can be solved, for example, by using Mixed Integer Linear Programming (Schrijver, 1998). However, there is no guarantee of finding the solution in polynomial time.

Appendix B

Universal approximation theorems for three-layer neural networks

We will prove that three-layer neural networks with a combination of minimum and maximum operators over linear functions have universal function approximation capabilities. We show this property for two types of network: (i) without any constraints on the weights, and (ii) with monotonicity constraints on some of the weights. For fully constrained monotone networks, the proof was given by Sill (1998).

B.1 Unconstrained neural networks

A general proof for the universal function approximation capabilities of unconstrained three-layer neural networks has been communicated to us through e-mail by William Armstrong, the developer of Adaptive Logic Networks discussed in Section 4.2. We use his ideas about the representation of linear functions to build another constructive proof. We also follow the grid representation scheme used in the proof for the universal approximation capabilities of monotone networks given by Sill (1998).

Suppose we have a three-layer neural network with the architecture presented in Figure B.1. Note that the weights between the input and the first hidden layer are unconstrained. Let $O_{\mathbf{x}}$ denote the output of the network for an input $\mathbf{x}$.

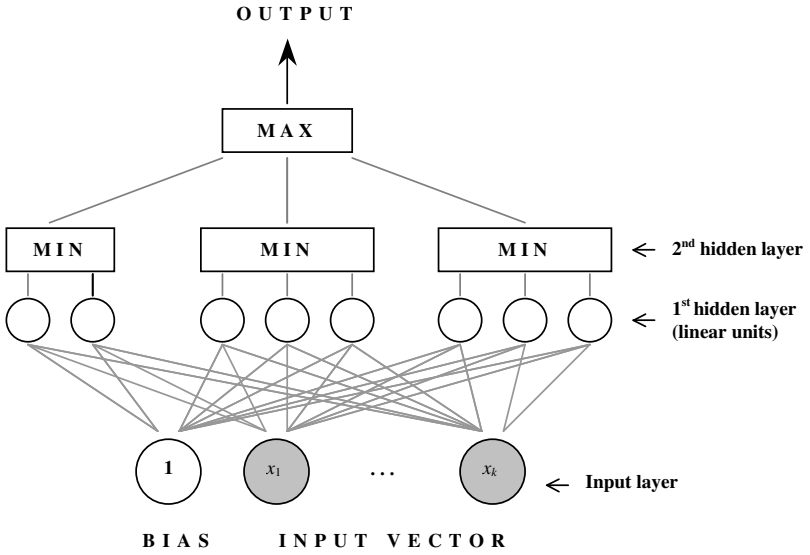


Figure B.1: An architecture of three-layer neural network with MIN and MAX operators.

Theorem B.1.1. *Let $\mathcal{X}$ denote a closed bounded domain of k inputs (dimensions) and f be a continuous bounded function mapping $\mathcal{X}$ to $\Re^+$. Then, for any $\epsilon > 0$ there exists a function $O_{\mathbf{x}}$ that can be implemented by a three-layer network such that $|f(\mathbf{x}) - O_{\mathbf{x}}| < \epsilon$, for any $\mathbf{x} \in \mathcal{X}$.*

Proof. Let $\epsilon > 0$. Define an equispaced grid of points on $\mathcal{X}$ such that the spacing between grid points along each dimension is δ, $\delta > 0$, and it is taken such that for any grid point $\mathbf{s}$ and for any $\mathbf{x}$ with $\|\mathbf{x} - \mathbf{s}\|_\infty < \delta$, we have $|f(\mathbf{x}) - f(\mathbf{s})| < \epsilon$. Here $\|\cdot\|_\infty$ denotes the max-norm distance between two points; for k inputs, this distance is defined by

$$\|\mathbf{x} - \mathbf{x}'\|_\infty = \max_i |x_i - x_i'|, \quad i = 1, 2, \ldots, k.$$

Corresponding to each grid point $\mathbf{s}$, we assign a group consisting of $2k + 1$ hyperplanes: one hyperplane is the constant output $h_{\mathbf{s}} = f(\mathbf{s})$. In addition, along each dimension d we place two hyperplanes:

$$h_{\mathbf{s}}^+(x_d) = \frac{2}{\delta} f(\mathbf{s})(x_d - s_d + \delta) \quad \text{and} \quad h_{\mathbf{s}}^-(x_d) = \frac{2}{\delta} f(\mathbf{s})(s_d - x_d + \delta).$$

Furthermore, we denote

$$\underline{f} = \min_{\mathbf{s}} f(\mathbf{s}) \quad \text{and} \quad \overline{f} = \max_{\mathbf{s}} f(\mathbf{s}), \tag{B.1}$$

where $\mathbf{s}$ are the grid points close to $\mathbf{x}$, i.e., $\|\mathbf{x}-\mathbf{s}\|_\infty < \delta$. The number of these points is at most 2^k.

We will show that

$$\underline{f} \leq O_{\mathbf{x}} \leq \overline{f}. \tag{B.2}$$

First we prove the left-hand side of the inequality in (B.2). It is clear that for any point $\mathbf{x}$, $\mathbf{x} \in \mathcal{X}$, there always exists at least one grid point $\mathbf{q}$ such that $\|\mathbf{x}-\mathbf{q}\|_\infty \leq \delta/2$. Then, by the definition of the hyperplanes the group's output (minimum over the hyperplanes) for $\mathbf{q}$ at $\mathbf{x}$ is $f(\mathbf{q})$. Hence, the final network's output $O_{\mathbf{x}}$ (maximum over all the groups' outputs) at $\mathbf{x}$ is $\geq f(\mathbf{q}) \geq \underline{f}$.

Next we proceed with the proof of the right-hand side of the inequality in (B.2). We consider two sets of grid points and we show that their group's outputs are $\leq \overline{f}$.

First for any grid point $\mathbf{s}$ with $\|\mathbf{x}-\mathbf{s}\|_\infty < \delta$, the group's output for $\mathbf{s}$ at $\mathbf{x}$ lies in the range $(0, f(\mathbf{s})]$, and therefore it is $\leq \overline{f}$.

Now we take a grid point $\mathbf{t}$ with $\|\mathbf{x}-\mathbf{t}\|_\infty \geq \delta$. In other words, there is at least one dimension d for which $|x_d - t_d| \geq \delta$. If $x_d - t_d \geq \delta$, then $h^-_{\mathbf{t}}(x_d) \leq 0$; if $t_d - x_d \geq \delta$, then $h^+_{\mathbf{t}}(x_d) \leq 0$. Since at each group we compute the minimum over all the hyperplanes, the group associated with $\mathbf{t}$ would produce an output at $\mathbf{x}$ that is $\leq 0 < \overline{f}$.

Since the group's outputs at $\mathbf{x}$ for all grid points are $\leq \overline{f}$, it follows that $O_{\mathbf{x}} \leq \overline{f}$.

Finally, by the construction of the grid, for all points $\mathbf{s}$, $\|\mathbf{x}-\mathbf{s}\|_\infty < \delta$, we have $|f(\mathbf{x}) - f(\mathbf{s})| < \epsilon$. Thus, $|f(\mathbf{x}) - \underline{f}| < \epsilon$ and $|f(\mathbf{x}) - \overline{f}| < \epsilon$. Hence, we have $|f(\mathbf{x}) - O_{\mathbf{x}}| < \epsilon$, which completes the proof. □

Remark. The theorem can be generalized to mappings $g : \mathcal{X} \to \Re$. First note that any function g can be represented as $g = f - C$, where f is a positive function and C is a constant. Next, given Theorem B.1.1 for any $\epsilon > 0$ we can always find a Sill network's approximation $\hat{f}$ of f such that $|f - \hat{f}| < \epsilon$. Finally, we apply $\hat{g} = \hat{f} - C$ to obtain the approximation $\hat{g}$ of g. The proof is straightforward.

B.2 Partially monotone neural networks

Now we show that the type of three-layer neural networks presented in Figure B.1 with weights constrained to be non-negative on some of the inputs can arbitrarily well approximate any partially monotone function. Note that the network's output $O_{\mathbf{x}}$ is guaranteed to be partially monotone by construction. Our proof is based on a function that is monotone in one of the inputs, but it is trivial to generalize the results to multiple monotone inputs.

Theorem B.2.1. *Let $\mathcal{X}$ denote a closed bounded domain of k inputs (dimensions) and f be a continuous bounded function mapping $\mathcal{X}$ to $\Re^+$ that is monotonically increasing in x_k. Then, for any $\epsilon > 0$ there exists a partially monotone function $O_{\mathbf{x}}$ that can be implemented by a three-layer network with weights on the k^{th} input constrained to be non-negative, such that $|f(\mathbf{x}) - O_{\mathbf{x}}| < \epsilon$, for any $\mathbf{x} \in \mathcal{X}$.*

Proof. The proof follows the same line of reasoning as the proof of Theorem B.1.1. The difference is that now to each grid point $\mathbf{s}$, along the monotone dimension k, we place only one hyperplane:

$$h_{\mathbf{s}}^{+}(x_k) = \frac{2}{\delta} f(\mathbf{s}) (x_k - s_k + \delta).$$

Thus, at any grid point we have $2k$ hyperplanes in total. We again show that

$$\underline{f} \leq O_{\mathbf{x}} \leq \overline{f},$$

for $\underline{f}$ and $\overline{f}$ defined in (B.1).

The left-hand side of the inequality is proved analogously to the unconstrained case. For the right-hand side, we now consider three sets of grid points and we show that their group's outputs are $\leq \overline{f}$.

First, for any grid point $\mathbf{s}$ with $\|\mathbf{x} - \mathbf{s}\|_\infty < \delta$, the group's output (minimum over the hyperplanes) at $\mathbf{x}$ lies in the range $(0, f(\mathbf{s})]$, so it is $\leq \overline{f}$.

Next, we take any grid point $\mathbf{q}$ with $q_k - x_k \geq \delta$ or $|x_d - q_d| \geq \delta$, for at least one non-monotone dimension d, $d = 1, 2, \ldots, k-1$. Then, by the definition of the group hyperplanes, the group's output for $\mathbf{q}$ at $\mathbf{x}$ is $\leq 0 < \overline{f}$.

Finally, we consider any grid point $\mathbf{t}$ with $x_k - t_k \geq \delta$ and $|x_d - t_d| < \delta$, for $d = 1, 2, \ldots, k-1$. Then, by the definition of the hyperplanes, the group's output for $\mathbf{t}$ at $\mathbf{x}$ lies in the range $(0, f(\mathbf{t})]$. Note that there always exists at least one grid point $\mathbf{s}$ with $\|\mathbf{x} - \mathbf{s}\|_\infty < \delta$ such that $t_k < s_k$ and $t_d = s_d$,

for $d = 1, 2, \ldots, k-1$. Then by monotonicity of f on the k^{th} dimension, we have $f(\mathbf{t}) \leq f(\mathbf{s}) \leq \overline{f}$.

Since the group's outputs at $\mathbf{x}$ for all grid points are $\leq \overline{f}$, it follows that $O_{\mathbf{x}} \leq \overline{f}$. Following the same line of reasoning as in the proof for the unconstrained networks, we can show that $|f(\mathbf{x}) - O_{\mathbf{x}}| < \epsilon$. □

Remark. The proof is analogous if f is monotonically decreasing in one or more inputs. Then, along the monotone dimension(s) we take h^- instead of h^+.

Appendix C

Agglomerative hierarchical clustering

As we discussed in the introduction of this thesis, the objective of clustering is to partition the data into groups of similar objects (points). Three main clustering types are distinguished in the literature: partition-based clustering, hierarchical clustering, and probabilistic model-based clustering (Jain and Dubes, 1988; Hand et al., 2001). Here, we discuss the basic nature of hierarchical clustering, which is applied in our approach for partial monotonicity (see Section 5.3). For more details on clustering and its types, the reader is referred to the aforementioned publications.

Hierarchical clustering builds a cluster hierarchy with tree-like nature, also known as a dendrogram. Every cluster node contains child clusters; sibling clusters partition the points covered by their common parent. Such an approach allows exploring data on different levels of granularity. Hierarchical clustering methods are categorized into agglomerative (bottom-up) and divisive (top-down). An agglomerative clustering starts with one-point (singleton) clusters, and recursively merges two or more most appropriate clusters based on a distance measure. A divisive clustering starts with one cluster of all data points, and recursively splits the most appropriate cluster. The process continues until a stopping criterion (usually, the requested number C of clusters) is achieved. Agglomerative clustering is the more important and widely used of the two. Therefore, we have used it in our approach, and describe it briefly below.

Suppose, we have a data set $D = \left(\mathbf{x}^n, \ell_{\mathbf{x}^n}\right)_{n=1}^{N}$ and a function $dist(c_i, c_j)$ measuring the distance between the two clusters c_i and c_j. Then, the agglo-

merative hierarchical clustering algorithm described by (Hand et al., 2001) is as follows.

for $n = 1$ to N **do**
  $c_n = \{\mathbf{x}^n\}$
end for
while there is more than one cluster left **do**
  $(c_s, c_t) = \min_{i,j} dist(c_i, c_j)$
  $c_s = c_s \cup c_t$
  remove cluster c_t
end while

The distance function $dist(c_i, c_j)$ can be computed in different ways:

- *Single linkage* defines the distance between two clusters as the distance between the two closest points, one from each cluster;

$$dist_{sl}(c_i, c_j) = \min_{\mathbf{x}^i, \mathbf{x}^j} \left\{ d(\mathbf{x}^i, \mathbf{x}^j) | \, \mathbf{x}^i \in c_i, \mathbf{x}^j \in c_j \right\},$$

  where $d(\mathbf{x}^i, \mathbf{x}^j)$ is the distance (usually Euclidean) between objects $\mathbf{x}^i$ and $\mathbf{x}^j$.

- *Complete linkage* defines the distance between two clusters as the distance between the two furthest away points, one from each cluster;

$$dist_{cl}(c_i, c_j) = \max_{\mathbf{x}^i, \mathbf{x}^j} \left\{ d(\mathbf{x}^i, \mathbf{x}^j) | \, \mathbf{x}^i \in c_i, \mathbf{x}^j \in c_j \right\}.$$

- *Average linkage* defines the distance between two clusters as the average of all the distances between pairs of points, one from each cluster;

$$dist_{al}(c_i, c_j) = \text{average}_{\mathbf{x}^i, \mathbf{x}^j} \left\{ d(\mathbf{x}^i, \mathbf{x}^j) | \, \mathbf{x}^i \in c_i, \mathbf{x}^j \in c_j \right\}.$$

To illustrate agglomerative clustering, we use a simple example. Figure C.1(a) represents five objects in a two-dimensional input space. In the first step, the clustering algorithm creates five clusters corresponding to the five objects. Next, it merges the singletons $\mathbf{x}^3$ and $\mathbf{x}^4$ into one cluster, because these are the closest clusters. In the third step, the distance between the

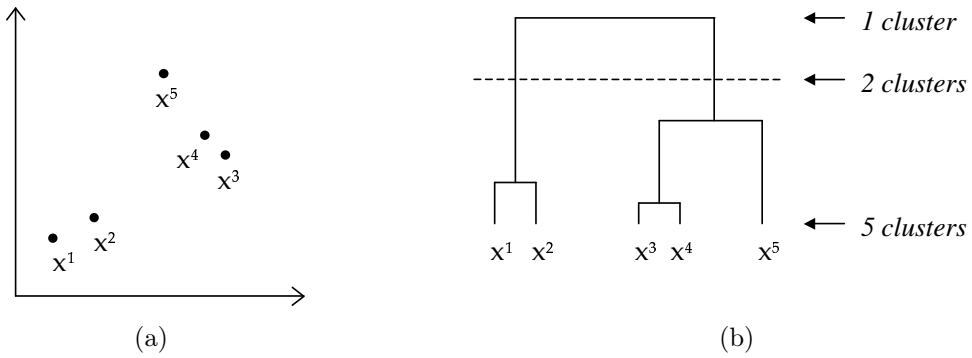


Figure C.1: An example of agglomerative clustering: (a) five objects in a two-dimensional input space and (b) hierarchical clustering tree (dendrogram) built on the five objects

singletons $\mathbf{x}^1$ and $\mathbf{x}^2$ is the smallest, so they are merged. Then, $\mathbf{x}^5$ is merged with the ($\mathbf{x}^3$,$\mathbf{x}^4$) cluster. In the fifth step, ($\mathbf{x}^1$,$\mathbf{x}^2$) and ($\mathbf{x}^3$,$\mathbf{x}^4$,$\mathbf{x}^5$) are merged into one final cluster. The resulting hierarchical clustering tree is depicted in Figure C.1(b).

After the hierarchical clustering tree is built, we can cut off the hierarchy at different levels to obtain different cluster numbers as shown in Figure C.1(b). Usually we try to determine the number of clusters that best represents the natural grouping in the data. This can be done by using the so-called *silhouette value* proposed by Rousseeuw (1987). The silhouette value $s(\mathbf{x}^n)$ for each object $\mathbf{x}^n$, $n = 1, \ldots, N$, is a measure of how similar that object is to objects in its own cluster compared with objects in other clusters:

$$s(\mathbf{x}^n) = \frac{b(\mathbf{x}^n, c) - a(\mathbf{x}^n)}{\max(a(\mathbf{x}^n), b(\mathbf{x}^n, c))}$$

where $a(\mathbf{x}^n)$ is the average distance from the object $\mathbf{x}^n$ to the other objects in its cluster, and $b(\mathbf{x}^n, c)$ is the average distance from the object $\mathbf{x}^n$ to the objects in another cluster c closest to $\mathbf{x}^n$.

The silhouette value ranges from -1 to $+1$. If it is close to 1, the point is assigned to a very appropriate cluster. If the silhouette value is about zero, the point could be assign to another, closest cluster; the point lies equally far away from both clusters. If the silhouette value is close to -1, the point is "badly clustered". The overall silhouette value of the clustering outcome

is simply the average over the silhouette values for all points in the data set.

Hence, we can cut off the hierarchy at different levels corresponding to the number of clusters we want to obtain; for each of the clustering outcomes we compute the overall silhouette value. The number of clusters with the maximum overall average silhouette value is taken as the optimal partitioning of the data.

Samenvatting

Het vakgebied data mining is ontstaan uit de behoefte aan de extractie van waardevolle en nieuwe kennis uit grote gegevensverzamelingen ter ondersteuning van het nemen van beslissingen. Het is hierbij van groot belang dat de verkregen modellen nauwkeurig en begrijpelijk voor de eindgebruiker zijn; tevens dienen ze in overeenstemming te zijn met bedrijfsbeleid en de kennis van deskundigen in het toepassingsgebied. Dit is vooral van belang bij voorspelproblemen–een in de praktijk veel voorkomende data mining taak– waarbij het doel is nauwkeurige en plausibele voorspellingen te doen van een bepaalde variabele in het toepassingsgebied.

In de praktijk zijn vaak wel grote gegevensverzamelingen beschikbaar, maar de modellen die hieruit met standaard data mining technieken worden afgeleid zijn vaak niet in overeenstemming met de kennis van domeindeskundigen of restricties die vanuit toepassingsoogpunt aan het model worden opgelegd. Om dit probleem op te lossen is het van groot belang data mining algoritmen zodanig aan te passen dat dergelijke domeinkennis kan worden gecombineerd met de beschikbare data.

In veel toepassingsgebieden is kennis voorhanden over de richting van de samenhang tussen variabelen. We kunnen deze kennis uitdrukken middels een monotonierestrictie: hoe groter de waarde van een onafhankelijke variabele is, des te groter de waarde van de afhankelijke variabele, verondersteld dat de overige relevante variabelen gelijk blijven. Het is bijvoorbeeld redelijk te veronderstellen dat de vraagprijs van een huis toeneemt met de oppervlakte en het aantal kamers.

In dit proefschrift worden data mining algoritmen ontwikkeld die modellen opleveren die aan deze monotonierestricties voldoen zonder vergaande beperkingen op te leggen aan de relatie tussen de afhankelijke variabele en de onafhankelijke variabelen. We beschouwen hierbij zowel volledig monotone als gedeeltelijk monotone voorspelproblemen. Bij volledig monotone voor-

spelproblemen wordt een stijgend verband verondersteld tussen de afhankelijke variabele en iedere onafhankelijke variabele; bij gedeeltelijk monotone problemen is dit slechts het geval voor een deel van de onafhankelijke variabelen.

Alhoewel er reeds verscheidene studies bestaan over het gebruik van monotonierestricties in data mining algoritmen, draagt dit proefschrift op een aantal manieren bij aan dit gebied.

Ten eerste presenteren we in paragraaf 2.3 een eenvoudige manier om te toetsen of een gegevensverzameling door een monotoon proces is gegenereerd. Deze toets kan worden gebruikt om te bepalen of het al dan niet verantwoord is monotonierestricties aan het model op te leggen. De toets is niet-parametrisch, en is derhalve van toepassing op gegevensverzamelingen met zeer uiteenlopende eigenschappen.

De tweede bijdrage is een gretig algoritme voor het monotoon maken van gegevens; dit algoritme wordt beschreven in paragraaf 2.4. Het monotoon maken van gegevens zorgt ervoor dat ze consistent en in overeenstemming met de kennis van domeindeskundigen zijn. Tevens kan het algoritme worden gebruikt als voorbewerkingsstap voor data mining algoritmen die een monotone gegevensverzameling vereisen. De gegevensverzameling wordt monotoon gemaakt door de waarde van de afhankelijke variabele voor zo min mogelijk observaties aan te passen, teneinde de geobserveerde gegevens zo min mogelijk te vertekenen. De hypothese is dat de monotone gegevensverzameling betere (nauwkeurigere en stabielere) monotone modellen oplevert dan de oorspronkelijke gegevensverzameling. Om na te gaan of dit waar is, bouwen we monotone modellen voor classificatie- en regressieproblemen met behulp van beslisbomen en neurale netwerken (hoofdstuk 3 en 4). Hiertoe hebben we bestaande algoritmen voor het construeren van monotone beslisbomen en monotone neurale netwerken verbeterd; dit is onze derde bijdrage.

De vierde bijdrage, beschreven in hoofdstuk 5, is een methode en bijbehorend algoritme voor het bouwen van gedeeltelijk monotone modellen. De methode is gebaseerd op de convolutie van monotone neurale netwerken gebouwd met de variabelen die een monotoon verband hebben met de afhankelijke variabele en gewichtsfuncties die zijn geconstrueerd op de overige variabelen. Voorzover wij kunnen nagaan is dit de eerste methode die gebruik maakt van de mengsel-van-netwerken aanpak voor problemen met gedeeltelijke monotonie. We bewijzen dat onze gedeeltelijk monotone modellen universele benaderingseigenschappen bezitten. Met behulp van simulatie-

experimenten en experimenten met empirische gegevensverzamelingen tonen we aan dat de modellen die ons algoritme oplevert beter voorspellen dan gedeeltelijk monotone lineaire modellen. We laten zien dat deze verbetering vooral wordt bereikt door vermindering van de variantie in vergelijking met standaard neurale netwerken.

Onze laatste bijdrage is een bewijs, gepresenteerd in Appendix B, van de universele benaderingseigenschappen van drie-laags neurale netwerken met een combinatie van minimum- en maximumoperatoren over lineaire functies. We tonen deze eigenschap aan voor twee typen netwerken: (i) netwerken zonder enige restrictie op de gewichten, en (ii) netwerken met monotonierestricties op een deel van de gewichten. Laatstgenoemde is een alternatieve methode voor het bouwen van gedeeltelijk monotone modellen.

De resultaten van dit proefschrift laten zien dat het afdwingen van monotonierestricties in data mining kan leiden tot stabielere modellen die beter voorspellen, en beter aansluiten bij de kennis van domeindeskundigen.

Bibliography

Abdelbar, A. M. (1998). Achieving superior generalisation with a high order neural network. *Neural Computing & Applications*, *7*(2), 141–146.

Archer, N. P. and Wang, S. (1993a). Application of the backpropagation neural network algorithm with monotonicity constraints for two-group classification problems. *Decision Sciences*, *24*(1), 60–75.

Archer, N. P. and Wang, S. (1993b). Learning bias in neural networks and an approach to controlling its effect in monotonic classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*(9), 962–966.

Armstrong, W. W. (1974). *Dendronic Decisions Limited.* (`http://www.dendronic.com/main.htm`)

Armstrong, W. W. and Thomas, M. M. (1997). Adaptive logic networks. In *Handbook of Neural Computation* (Vol. 10, pages C1.8:1–14). Oxford University Press.

Ayer, M., Brunk, H. D., Ewing, G. M., Reid, W. T., and Silverman, E. (1955). An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics*, *26*(4), 641–647.

Ben-David, A. (1995). Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, *19*(1), 29–43.

Ben-David, A., Sterling, L., and Pao, Y.-H. (1989). Learning and classification of monotonic ordinal concepts. *Computational Intelligence*, *5*(1), 45–49.

Berry, M. J. A. and Linoff, G. (1997). *Data mining techniques for marketing, sales and customer support.* New York: John Wiley & Sons.

Bioch, J. C. and Popova, V. (2002). *Monotone decision trees and noisy data* (ERIM Internal Report No. 206). Erasmus University Rotterdam.

Bishop, C. (1997). *Neural networks for pattern recognition.* Oxford University Press.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees.* Belmont, Calif.: Wadsworth International Group.

Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, *6*, 76–90.

Cao-Van, K. and De Baets, B. (2003). Growing decision trees in an ordinal setting. *International Journal of Intelligent Systems*, *18*, 733–750.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. (2000). *CRISP-DM 1.0 Process and User Guide.* CRISP-DM Consortium. (`http://www.crisp-dm.org`)

Clark, D., Schreter, Z., and Adams, A. (1996). A quantitative comparison of Dystal and backpropagation. In *Proceedings of the seventh Australian Conference Neural Networks, Canberra, Australia* (p. 132–137).

Cybenko, G. (1989). Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*, 303–314.

Daniels, H. A. M. and Kamp, B. (1999). Application of MLP networks to bond rating and house pricing. *Neural Computing & Applications*, *8*(3), 226–234.

Daniels, H. A. M. and Velikova, M. V. (2003). *Derivation of monotone decision models from non-monotone data* (Center Discussion Paper Nos. 2003–30). Tilburg University.

Daniels, H. A. M. and Velikova, M. V. (2006). Derivation of monotone decision models from noisy data. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, *36*(5), 705–710.

Duda, R. O. and Hart, P. E. (1973). *Pattern classification and scene analysis.* New York: John Wiley & Sons.

Even, S. (1979). *Graph algorithms.* London: Pitman.

Feelders, A. (2000). Prior knowledge in economic applications of data mining. In *Principles of Data Mining and Knowledge Discovery, Lecture Notes in Artificial Intelligence* (Vol. 1910, p. 395–400). Springer.

Feelders, A. (2002). Statistical concepts. In M. Berthold and D. Hand (Eds.), *Intelligent data analysis: an introduction* (pages 15–66). Berlin: Springer-Verlag.

Feelders, A., Daniels, H. A. M., and Holsheimer, M. (2000). Methodological and practical aspects of data mining. *Information & Management*, *37*(5), 271–281.

Feelders, A. and Pardoel, M. (2003). Pruning for monotone classification trees. *Lecture Notes in Computer Science*, *2810*, 1–12.

Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., and Samorodnitsky, R. (2002). Monotonicity testing over general poset domains. In *Proceedings of the thirty-fourth annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada* (pages 474–483). ACM Press.

Fletcher, R. (1970). A new approach to variable metric algorithms. *Computer Journal*, *13*, 317–322.

Ford, L. R. and Fulkerson, D. R. (1962). *Flows in networks.* Princeton, NJ: Princeton University Press.

Friedman, J. H. and Tukey, J. W. (1974). A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, *C-23*(9), 881–889.

Frosyniotis, D., Stafylopatis, A., and Likas, A. (2003). A divide-and-conquer method for multi-net classifiers. *Pattern Analysis & Applications*, *6*(1), 32–40.

Galliers, R. (1992). *Information systems research: issues, methods and practical guidelines.* Oxford: Blackwell Scientific Publications.

Gamarnik, D. (1998). Efficient learning of monotone concepts via quadratic optimization. In *Proceedings of the eleventh Annual Conference on Computational Learning Theory, Madison, Wisconsin, United States* (pages 134–143). ACM Press.

Garey, M. and Johnson, D. (1979). *Computers and intractability: a guide to the theory of NP-completeness.* New York: Freeman.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*(1), 1–58.

Giudici, P. (2003). *Applied data mining: statistical methods for business and industry.* Chichester: John Wiley & Sons.

Goldfarb, D. (1970). A family of variable metric updates derived by variational means. *Mathematics of Computing*, *24*, 23–26.

Goldreich, O., Goldwasser, S., Lehman, E., and Ron, D. (1998). Testing monotonicity. In *Proceedings of the thirty-ninth Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA* (pages 426–435). IEEE Computer Society.

Hammersley, J. M. and Handscomb, D. C. (1964). *Monte Carlo methods.* London: Methuen.

Hand, D., Mannila, H., and Smyth, P. (2001). *Principles of data mining.* Cambridge: MIT Press.

Harrison, O. and Rubinfeld, D. (1978). Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, *53*, 81–102.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2001). *The elements of statistical learning: data mining, inference, and prediction.* New York: Springer-Verlag.

Hellerstein, J. (1989). A statistical approach to diagnosing intermittent performance-problems using monotone relationships. In *Proceedings of the 1989 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Oakland, California, United States* (pages 20–28). ACM Press.

Huber, P. J. (1985). Projection pursuit. *The Annals of Statistics*, *13*(2), 435–475.

Insightful Miner 3.0 User's Guide. (2003). Seattle: Insightful Corporation. (`http://www.insightful.com/support/iminer30/uguide.pdf`)

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixture of local experts. *Neural Computation*, *3*, 79–87.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for data clustering.* New Jersey: Prentice Hall.

Jolliffe, I. T. (1986). *Principal component analysis.* New York: Springer-Verlag.

Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181-214.

Karpf, J. (1991). Inductive modeling in law: example based expert systems in administrative law. In *Proceedings of the third International Conference on Artificial Intelligence and Law, Oxford, England* (pages 297–306). ACM Press.

Kay, H. and Ungar, L. H. (2000). Estimating monotonic functions and their bounds. *American Institute of Chemical Engineers (AIChE) Journal*, *46*(12), 2426–2434.

Kleijnen, J. P. C. (2004). Design and analysis of Monte Carlo experiments. In J. Gentle, W. Haerdle, and Y. Mori (Eds.), *Handbook of computational statistics; Volume I: concepts and fundamentals* (pages 497–516). Berlin: Springer.

Kohavi, R. and Wolpert, D. H. (1996). Bias plus variance decomposition for zero-one loss functions. In *Proceedings of the thirteenth International Conference on Machine Learning, Bari, Italy* (p. 275-283). Morgan Kaufmann.

Lawler, E. L. (1976). *Combinatorial optimization: networks and matroids.* New York: Holt, Rinehart and Winston.

Lee, J. W. T., Yeung, D. S., and Wang, X. (2003). Monotonic decision tree for ordinal classification. *IEEE International Conference on Systems, Man and Cybernetics*, *3*, 2623–2628.

Lory, P. and Gietl, D. (2000). Neural networks for two-group classification problems with monotonicity hints. In *Classification and Information Processing at the Turn of the Millennium* (pages 113–118). Springer-Verlag.

Makino, K., Suda, T., Ono, H., and Ibaraki, T. (1999). Data analysis by positive decision trees. *IEICE Transactions on Information and Systems*, *E82-D*(1), 76–88.

MATLAB Central. `http://www.mathworks.com/matlabcentral/`.

MATLAB Software. `http://www.mathworks.com/products/matlab/`.

MIT Technology Review. (2001). Ten emerging technologies that will change the world. *Annual Innovation Issue*, *104*(1), Jan./Feb.

Möhring, R. H. (1985). Algorithmic aspects of comparability graphs and interval graphs. In *Graphs and Order* (pages 41–101). Dordrecht: D. Reidel Publishing Company.

Moshkovich, H. M., Mechitov, A. I., and Olson, D. L. (2002). Rule induction in data mining: effect of ordinal dependencies. *Expert Systems with Applications*, *22*(4), 303–311.

Mukarjee, H. and Stern, S. (1994). Feasible nonparametric estimation of multiargument monotone functions. *Journal of the American Statistical Association*, *89*(425), 77–80.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, *9*(1), 141–142.

Nash, W. J., Sellers, T. L., Talbot, S. R., and Cawthorn, W. B., A. J. anf Ford. (1994). *The Population Biology of Abalone (Haliotis species) in Tasmania. 1. Blacklip Abalone (H. rubra) from the North Coast and the islands of Bass Strait* (Technical Report No. 48). Sea Fisheries Division, Marine Research Laboratories-Taroona, Department of Primary Industry and Fisheries, Tasmania, Australia.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, *7*, 308–313.

Newman, D. J., Hettich, S., Blake, C. L., and Merz, C. J. (1998). *UCI Repository of Machine Learning Databases.* University of California, Irvine, Dept. of Information and Computer Sciences. (`http://www.ics.uci.edu/~mlearn/MLRepository.html`)

Obesity Education Initiative. (1998). *Clinical guidelines on the identification, evaluation, and treatment of overweight and obesity in adults* (NIH Report No. 98-4083). National Institutes of Health, National Heart, Lung, and Blood Institute.

Pidd, M. (1996). *Tools for thinking: modeling in management science.* Chichester: John Wiley & Sons.

Popova, V. (2004). *Knowledge discovery and monotonicity.* PhD thesis, Erasmus University Rotterdam, Rotterdam, The Netherlands.

Potharst, R. (1999). *Classification using decision trees and neural nets.* PhD thesis, Erasmus University Rotterdam, Rotterdam, The Netherlands.

Potharst, R. and Bioch, J. C. (2000). Decision trees for ordinal classification. *Intelligent Data Analysis*, *4*(2), 97–111.

Potharst, R. and Feelders, A. (2002). Classification trees for problems with monotonicity constraints. *SIGKDD Explorations Newsletter*, *4*(1), 1–10.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Los Altos, CA: Morgan Kaufmann.

Quinlan, J. R. (2005). *C5: Data Mining Tool.* (`http://www.rulequest.com/see5-info.html`, visited in September 2005)

Rademaker, M., De Baets, B., and De Meyer, H. (2006). Data sets for supervised ranking: to clean or not to clean. In *Proceedings of the fifteenth Annual Machine Learning Conference of Belgium and The Netherlands: Benelearn 2006, Ghent, Belgium* (p. 139–146).

Raskhodnikova, S. (1999). *Monotonicity testing.* Master thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Robertson, T., Wright, F. T., and Dykstra, R. L. (1988). *Order restricted statistical inference.* Chicester: John Wiley & Sons.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53–65.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructures of cognition* (Vol. 1, pages 318–362). MIT Press.

S-PLUS Software. `http://www.insightful.com/products/splus/default.asp`.

Sarfraz, M., Al-Mulhem, M., and Ashraf, F. (1997). Preserving monotonic shape of the data by using piecewise rational cubic functions. *Computers and Graphics*, *21*, 5–14.

Schell, M. J. and Singh, B. (1997). The reduced monotonic regression method. *Journal of the American Statistical Association*, *92*(437), 128–135.

Schrijver, A. (1998). *Theory of linear and integer programming.* Chichester: John Wiley & Sons.

Shanno, D. F. (1970). Conditioning of Quasi-Newton methods for function minimization. *Mathematics of Computing*, *24*, 647–656.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*, 379–423, 623–656.

Siem, A. Y. D., De Klerk, E., and Den Hertog, D. (2005). *Discrete least-norm approximation by nonnegative (trigonometric) polynomials and rational functions* (Center Discussion Paper Nos. 2005–73). Tilburg University.

Sill, J. (1998). Monotonic networks. In *Advances in Neural Information Processing Systems (NIPS)* (Vol. 10, pages 661–667). MIT Press.

Sill, J. and Abu-Mostafa, Y. S. (1997). Monotonicity hints. In *Advances in Neural Information Processing Systems (NIPS)* (Vol. 9, pages 634–640). MIT Press.

Strobl, R., Salanti, G., and Ulm, K. (2003). *Extension of CART using multiple splits under order restrictions* (SFB 386 Discussion Paper No. 364). LMU München.

Suárez-Fariñas, M. and Pedreira, C. E. (2003). Mixture of experts and local-global neural networks. In *Proceedings of the eleventh European Symposium on Artificial Neural Networks, Bruges, Belgium* (pages 331–336). Brussels: D-Facto.

Swamy, M. N. S. and Thulasiraman, K. (1981). *Graphs, networks, and algorithms.* New York: John Wiley & Sons.

Tuy, H. (2000). Monotonic optimization: problems and solution approaches. *SIAM Journal on Optimization*, *11*(2), 464–494.

Velikova, M. V. and Daniels, H. A. M. (2004). Decision trees for monotone price models. *Computational Management Science*, *1*(3–4), 231–244.

Velikova, M. V., Daniels, H. A. M., and Feelders, A. (2006a). Mixtures of monotone networks for prediction. *International Journal of Computational Intelligence*, *3*(3), 204–214.

Velikova, M. V., Daniels, H. A. M., and Feelders, A. (2006b). Solving partially monotone problems with neural networks. In *Proceedings of the twelfth International Conference on Computer Science, Vienna, Austria* (pages 82–87). Turkey: World Enformatika Society.

Vogel, D. R. and Wetherbe, J. C. (1984). MIS Research: a profile of leading journals and universities. *Data Base*, *16*(1), 3–14.

Wang, S. (1994). A neural network method of density estimation for univariate unimodal data. *Neural Computing & Applications*, *2*(3), 160–167.

Watson, G. S. (1964). Smooth regression analysis. *Sankhya: The Indian Journal of Statistics, Series A*, *26*(4), 359–372.

Waugh, S. (1995). *Extending and benchmarking cascade-correlation.* PhD thesis, University of Tasmania, Tasmania, Australia.

Wu, C. F. J. and Hamada, M. (2000). *Experiments: Planning, analysis, and parameter design optimization* (Wiley Series in Probability and Statistics ed.). New York: John Wiley & Sons.