

DATA MINING IN FINANCE

Advances in Relational and Hybrid Methods

**The Kluwer International Series
in Engineering and Computer Science**

DATA MINING IN FINANCE

Advances in Relational and Hybrid Methods

by

BORIS KOVALERCHUK

Central Washington University, USA

and

EVGENII VITYAEV

Institute of Mathematics

Russian Academy of Sciences, Russia

KLUWER ACADEMIC PUBLISHERS

NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 0-306-47018-7
Print ISBN: 0-792-37804-0

©2002 Kluwer Academic Publishers
New York, Boston, Dordrecht, London, Moscow

Print ©2000 Kluwer Academic Publishers
New York

All rights reserved

No part of this eBook may be reproduced or transmitted in any form or by any means, electronic, mechanical, recording, or otherwise, without written consent from the Publisher

Created in the United States of America

Visit Kluwer Online at: <http://kluweronline.com>
and Kluwer's eBookstore at: <http://ebooks.kluweronline.com>

***To our families and
Klim***

TABLE OF CONTENTS

Foreword by Gregory Piatetsky-Shapiro	xi
Preface	xiii
Acknowledgements	xv
1. The Scope and Methods of the Study	
1.1 Introduction.....	1
1.2 Problem definition	3
1.3 Data mining methodologies	4
1.3.1 Parameters.....	4
1.3.2 Problem ID and profile	6
1.3.3 Comparison of intelligent decision support methods.....	7
1.4 Modern methodologies in financial knowledge discovery	9
1.4.1 Deterministic dynamic system approach	9
1.4.2 Efficient market theory	10
1.4.3 Fundamental and technical analyses	11
1.5 Data mining and database management.....	12
1.6 Data mining: definitions and practice	14
1.7 Learning paradigms for data mining.....	17
1.8 Intellectual challenges in data mining.....	19
2. Numerical Data Mining Models with Financial Applications	
2.1. Statistical, autoregression models	21
2.1.1. ARIMA models.....	22
2.1.2. Steps in developing ARIMA model.....	25
2.1.3. Seasonal ARIMA	27
2.1.4. Exponential smoothing and trading day regression	28
2.1.5. Comparison with other methods.....	28
2.2. Financial applications of autoregression models	30
2.3. Instance-based learning and financial applications	32
2.4. Neural networks	36
2.4.1. Introduction.....	36
2.4.2. Steps	38
2.4.3. Recurrent networks.....	39
2.4.4. Dynamically modifying network structure	40
2.5. Neural networks and hybrid systems in finance	40

2.6. Recurrent neural networks in finance	42
2.7. Modular networks and genetic algorithms.....	44
2.7.1. Mixture of neural networks.....	44
2.7.2. Genetic algorithms for modular neural networks.....	45
2.8. Testing results and the complete round robin method.....	47
2.8.1. Introduction.....	47
2.8.2. Approach and method	47
2.8.3. Multithreaded implementation	52
2.8.4. Experiments with SP500 and neural networks	54
2.9. Expert mining	58
2.10. Interactive learning of monotone Boolean functions	66
2.10.1. Basic definitions and results	66
2.10.2. Algorithm for restoring a monotone Boolean function.....	67
2.10.3. Construction of Hansel chains	69

3. Rule-Based and Hybrid Financial Data Mining

3.1. Decision tree and DNF learning.....	71
3.1.1. Advantages.....	71
3.1.2. Limitation: size of the tree.....	72
3.1.3. Constructing decision trees	81
3.1.4. Ensembles and hybrid methods for decision trees.....	84
3.1.5. Discussion	87
3.2. Decision tree and DNF learning in finance.....	88
3.2.1. Decision-tree methods in finance.....	88
3.2.2. Extracting decision tree and sets of rules for SP500.....	89
3.2.3. Sets of decision trees and DNF learning in finance.....	93
3.3. Extracting decision trees from neural networks.....	95
3.3.1. Approach.....	95
3.3.2. Trepan algorithm.....	96
3.4. Extracting decision trees from neural networks in finance.....	97
3.4.1. Predicting the Dollar-Mark exchange rate.....	97
3.4.2. Comparison of performance	99
3.5. Probabilistic rules and knowledge-based stochastic modeling.....	102
3.5.1. Probabilistic networks and probabilistic rules.....	103
3.5.2. The naïve Bayes classifier	106
3.5.3. The mixture of experts	107
3.5.4. The hidden Markov model	108
3.5.5. Uncertainty of the structure of stochastic models	111
3.6. Knowledge-based stochastic modeling in finance.....	112
3.6.1. Markov chains in finance	112
3.6.2. Hidden Markov models in finance	114

4. Relational Data Mining (RDM)

4.1. Introduction.....	115
4.2. Examples.....	118
4.3. Relational data mining paradigm	123
4.4 Challenges and obstacles in relational data mining.....	127
4.5 Theory of RDM	129
4.5.1 Data types in relational data mining	129
4.5.2 Relational representation of examples.	130
4.5.3 First-order logic and rules.....	135
4.6 Background knowledge	140
4.6.1 Arguments constraints and skipping useless hypotheses.....	140
4.6.2 Initial rules and improving search of hypotheses.....	141
4.6.3 Relational data mining and relational databases	144
4.7 Algorithms: FOIL and FOCL	146
4.7.1 Introduction.....	146
4.7.2 FOIL.....	147
4.7.3 FOCL	150
4.8 Algorithm MMDR	151
4.8.1 Approach.....	151
4.8.2 MMDR algorithm and existence theorem.....	154
4.8.3 Fisher test.....	159
4.8.4 MMDR pseudocode.....	162
4.8.5 Comparison of FOIL and MMDR	165
4.9 Numerical relational data mining	166
4.10 Data types	169
4.10.1 Problem of data types	169
4.10.2 Numerical data type	174
4.10.3 Representative measurement theory.....	174
4.10.4 Critical analysis of data types in ABL	175
4.11 Empirical axiomatic theories: empirical contents of data.....	179
4.11.1 Definitions.	179
4.11.2 Representation of data types in empirical axiomatic theories.	181
4.11.3 Discovering empirical regularities as universal formulas.....	186

5. Financial Applications of Relational Data Mining

5.1. Introduction.....	189
5.2. Transforming numeric data into relations.....	191
5.3. Hypotheses and probabilistic “laws”.....	193
5.4. Markov chains as probabilistic “laws” in finance.....	196
5.5. Learning.....	199
5.6. Method of forecasting	202

5.7. Experiment 1 204

5.7.1. Forecasting Performance for hypotheses H1-H4 204

5.7.2. Forecasting performance for a specific regularity..... 207

5.7.3. Forecasting performance for Markovian expressions..... 209

5.8. Experiment 2..... 212

5.9. Interval stock forecast for portfolio selection 213

5.10. Predicate invention for financial applications: calendar effects.. 215

5.11. Conclusion 218

6 Comparison of Performance of RDM and other methods in financial applications

6.1. Forecasting methods 219

6.2. Approach: measures of performance 220

6.3. Experiment 1: simulated trading performance..... 222

6.4. Experiment 1: comparison with ARIMA..... 225

6.5. Experiment 2: forecast and simulated gain..... 227

6.6. Experiment 2: analysis of performance..... 227

6.7. Conclusion 229

7. Fuzzy logic approach and its financial applications

7.1. Knowledge discovery and fuzzy logic..... 231

7.2. “Human logic” and mathematical principles of uncertainty..... 235

7.3. Difference between fuzzy logic and probability theory 239

7.4. Basic concepts of fuzzy logic 240

7.5. Inference problems and solutions 248

7.6. Constructing coordinated contextual linguistic variables..... 252

7.6.1. Examples..... 252

7.6.2. Context space 259

7.6.3. Acquisition of fuzzy sets and membership function..... 262

7.6.4. Obtaining linguistic variables 265

7.7. Constructing coordinated fuzzy inference 266

7.7.1. Approach..... 266

7.7.2. Example 268

7.7.3. Advantages of "exact complete" context for fuzzy inference.. 270

7.8. Fuzzy logic in finance..... 278

7.8.1. Review of applications of fuzzy logic in finance..... 278

7.8.2. Fuzzy logic and technical analysis..... 281

REFERENCES..... 285

Subject Index..... 299

FOREWORD

Finding Profitable Knowledge

The information revolution is generating mountains of data, from sources as diverse as astronomy observations, credit card transactions, genetics research, telephone calls, and web clickstreams. At the same time, faster and cheaper storage technology allows us to store ever-greater amounts of data online, and better DBMS software provides an easy access to those databases. The web revolution is also expanding the focus of data mining beyond structured databases to the analysis of text, hyperlinked web pages, images, sounds, movies and other multimedia data.

Mining financial data presents special challenges. For one, the rewards for finding successful patterns are potentially enormous, but so are the difficulties and sources of confusions. The efficient market theory states that it is practically impossible to predict financial markets long-term. However, there is good evidence that short-term trends do exist and programs can be written to find them. The data miners' challenge is to find the trends quickly while they are valid, as well as to recognize the time when the trends are no longer effective.

Additional challenges of financial mining are to take into account the abundance of domain knowledge that describes the intricately inter-related world of global financial markets and to deal effectively with time series and calendar effects. For example, Monday and Friday are known to usually have different effects on S&P 500 than other days of the week.

The authors present a comprehensive overview of major algorithmic approaches to predictive data mining, including statistical, neural networks,

rule-based, decision-tree, and fuzzy-logic methods and examine the suitability of these approaches to financial data mining.

They focus especially on relational data mining, which is a learning method able to learn more expressive rules than other symbolic approaches. RDM is thus better suited for financial mining, because it is able to make better use of underlying domain knowledge. Relational data mining also has a better ability to explain the discovered rules -- ability critical for avoiding spurious patterns which inevitably arise when the number of variables examined is very large. The earlier algorithms for relational data mining, also known as ILP -- inductive logic programming, suffer from a well-known inefficiency. The authors introduce a new approach, which combines relational data mining with the analysis of statistical significance of discovered rules. This reduces the search space and speeds up the algorithms.

The authors also introduce a set of interactive tools for "mining" the knowledge from the experts. This helps to further reduce the search space.

The authors' grand tour of the data mining methods contains a number of practical examples of forecasting S&P 500 and exchange rates, and allows interested readers to start building their own models. I expect that this book will be a handy reference to many financially inclined data miners, who will find the volume both interesting and profitable.

Gregory Piatetsky-Shapiro
Boston, Massachusetts

PREFACE

The new generation of computing techniques collectively called data mining methods are now applied to stock market analysis, predictions, and other financial applications. In this book we discuss the relative merits of these methods for financial modeling and present a comprehensive survey of current capabilities of these methods in financial analysis.

The focus is on the specific and highly topical issue of adaptive linear and non-linear “mining” of financial data. Topics are progressively developed. First, we examine the distinction between the use of such methods as ARIMA, neural networks, decision trees, Markov chains, hybrid knowledge-based neural networks, and hybrid relational methods. Later, we focus on examining financial time series, and, finally, modeling and forecasting these financial time series using data mining methods.

Our main purpose is to provide much needed guidance for applying new predictive and decision-enhancing hybrid methods to financial tasks such as capital-market investments, trading, banking services, and many others.

The very complex and challenging problem of forecasting financial time series requires specific methods of data mining. We discuss these requirements and show the relations between problem requirements and the capabilities of different methods. Relational data mining as a hybrid learning method combines the strength of inductive logic programming (ILP) and probabilistic inference to meet this challenge. A special feature of the book is the large number of worked examples illustrating the theoretical concepts discussed.

The book begins with problem definitions, modern methodologies of general data mining and financial knowledge discovery, relations between

data mining and database management, current practice, and intellectual challenges in data mining.

Chapter 2 is devoted to numerical data mining learning models and their financial applications. We consider ARIMA models, Markov chains, instance-based learning, neural networks, methods of learning from experts (“expert” mining”), and new methods for testing the results of data mining.

Chapter 3 presents rule-based and hybrid data mining methods such as learning propositional rules (decision trees and DNF), extracting rules from learned neural networks, learning probabilistic rules, and knowledge-based stochastic modeling (Markov chains and hidden Markov models) in finance.

Chapter 4 describes a new area of data mining and financial applications - relational data mining (RDM) methods. From our viewpoint, this approach will play a key role in future advances in data mining methodology and practice. Topics covered in this chapter include the relational data mining paradigm and current challenges, theory, and algorithms (FOIL, FOCL and MMDR).

Numerical relational data mining methods are especially important for financial analysis where data commonly are numerical financial time series. This subject is developed in chapters 4, 5 and 6 using complex data types and representative measurement theory. The RDM paradigm is based on highly expressive first-order logic language and inductive logic programming. Chapters 5 and 6 cover knowledge representation and financial applications of RDM. Chapter 6 also discusses key performance issues of the selected methods in forecasting financial time series. Chapter 7 presents fuzzy logic methods combined with probabilistic methods, comparison of fuzzy logic and probabilistic methods, and their financial applications.

Well-known and commonly used data mining methods in finance are attribute-based learning methods such as neural networks, the nearest neighbours method, and decision trees. These are relatively simple, efficient, and can handle noisy data. However, these methods have two serious drawbacks: a limited ability to represent background knowledge and the lack of complex relations. The purpose of relational data mining is to overcome these limitations. On the other hand, as Bratko and Muggleton noted [1995], current relational methods (ILP methods) are relatively inefficient and have rather limited facilities for handling numerical data. Biology, pharmacology, and medicine have already benefited significantly from relational data mining. We believe that now is the time for applying these methods to financial analyses. This book is addressed to researchers, consultants, and students interested in the application of mathematics to investment, economics, and management. We also maintain a related website <http://www.cwu.edu/~borisk/finanace>.

ACKNOWLEDGEMENTS

Authors gratefully acknowledge that relational learning methods presented in this book are originated by Professor Klim Samokhvalov in the 70s at the Institute of Mathematics of the Russian Academy of Sciences. His remarkable work has influenced us for more than two decades.

During the same period we have had fruitful discussions with many people from a variety of areas of expertise around the globe, including R. Burright, L. Zadeh, G. Klir, E. Hisdal, B. Mirkin, D. Dubous, G. Piatetsky-Shapiro, S. Kundu, S. Kak, J. Moody, A. Touzilin, A. Logvinenko, N. Zagoruiko, A. Weigend, G. Nakhaeizadeh, and R. Caldwell. These discussions helped us to shape multidisciplinary ideas presented in this book. Many discussions have lasted for years. Sometimes short exchanges of ideas during conferences and review papers have had a long-term effect.

For creating the data sets we investigated, we especially thank Randal Caldwell from the Journal of Computational Intelligence in Finance. We also obtained valuable support from the US National Research Council, the Office of Naval Research (USA), the Royal Society (UK) and the Russian Fund of Fundamental Research for our previous work on relational data mining methods, which allowed us to speed up the current financial data study. Finally, we want to thank James Schwing, Dale Comstock, Barry Donahue, Edward Gellenbeck, and Clayton Todd for their time, valuable and insightful commentary in the final stage of the book preparation. CWU students C. Todd, D. Henderson, and J. Summet provided programming assistance for some computations.

Chapter 1

The scope and methods of the study

October. This is one of the peculiarly dangerous months to speculate in stocks in. The others are July, January, September, April, November, May, March, June, December, August and February

Mark Twain [1894]

1.1 Introduction

Mark Twain's aphorism became increasingly popular in discussions about a new generation of computing techniques called **data mining (DM)** [Sullivan et al., 1998]. These techniques are now applied **to discover hidden trends and patterns in financial databases**, e.g., in stock market data for market prediction. The question in discussions is **how to separate real trends and patterns from mirages**. Otherwise, it is equally dangerous to follow any of them, as noted by Mark Twain more than hundred years ago. This book is intended to address this issue by presenting different methods without advocating any particular calendar dependency like the January stock calendar effect. We use stock market data in this book because, in contrast with other financial data, they are not proprietary and are well understood without extensive explanations.

Data mining draws from two major sources: **database and machine learning technologies** [Fayyad, Piatetsky-Shapiro, Smyth, 1996]. The goal of machine learning is to construct computer programs that automatically **improve with experience** [Mitchell, 1997]. Detecting fraudulent credit card transactions is one of the successful applications of machine learning. Many others are known in finance and other areas [Mitchell, 1999].

Friedman [1997] listed four major technological reasons stimulated data mining development, applications, and public interest:

- the emergence of very large **databases** such as commercial data warehouses and computer automated data recording;
- advances in computer technology such as faster and bigger **computer engines** and **parallel architectures**;
- **fast access** to vast amounts of data, and
- the ability to apply **computationally intensive statistical methodology** to these data.

Currently the methods used in data mining range from classical statistical methods to new inductive logic programming methods. This book introduces data mining methods for financial analysis and forecasting. We overview Fundamental Analysis, Technical Analysis, Autoregression, Neural Networks, Genetic Algorithms, k Nearest neighbours, Markov Chains, Decision Trees, Hybrid methods and Relational Data Mining (RDM).

Our emphasis is on **Relational Data Mining in the financial analysis and forecasting**. Relational Data Mining combines recent advances in such areas as **Inductive Logic Programming (ILP)**, **Probabilistic Inference**, and **Representative Measurement Theory (RMT)**. Relational data mining benefits from noise robust probabilistic inference and highly expressive and understandable first-order logic rules employed in ILP and representative measurement theory.

Because of the interdisciplinary nature of the material, this book makes few assumptions about the background of the reader. Instead, it introduces basic concepts as the need arises. Currently statistical and Artificial Neural Network methods dominate in financial data mining. Alternative **relational (symbolic) data mining** methods have shown their effectiveness in robotics, drug design and other applications [Lavrak et al., 1997, Muggleton, 1999].

Traditionally symbolic methods are used in the areas with a lot of **non-numeric (symbolic) knowledge**. In robot navigation, this is relative location of obstacles (on the right, on the left and so on). At first glance, stock market forecast looks as a pure numeric area irrelevant to symbolic methods. One of our major goals is to show that financial time series can benefit significantly from relational data mining based on symbolic methods.

Typically, general-purpose data mining and machine learning texts describe methods for very different tasks in the same text to show the broad range of potential applications. We believe that an effective way to learn about the relative strength of data mining methods is to view them from one type of application. Through the book, we use the SP500 and other stock market time series to show strength and weakness of different methods.

The book is intended for researchers, consultants, and students interested in the application of mathematics to investment, economics, and management. The book can also serve as a reference work for those who are conducting research in data mining.

1.2 Problem definition

Financial forecasting has been widely studied as a case of time-series prediction problem. The difficulty of this problem is due to the following factors: low signal-to-noise ratio, non-Gaussian noise distribution, nonstationarity, and nonlinearly [Oliker, 1997]. A variety of views exists on this problem, in this book, we try to present a faithful summary of these works.

Deriving relationships that allow one to predict future values of time series is a challenging task when the underlying system is highly non-linear. Usually, the history of the time series is provided and the goal is to extract from that data a **dynamic system**. The dynamic system models the relationship between a window of past values and a value T time steps ahead.

Discovering such a model is difficult in practice since the processes are typically corrupted by noise and can only be partially modelled due to missing information and the overall complexity of the problem. In addition, financial time series are inherently non-stationary so **adaptive forecasting techniques** are required.

Below in Tables 1.1-1.4 we present a list of typical task related to data mining in finance [Loofbourrow and Loofbourrow, 1995].

Table 1.1. Data mining tasks for portfolio managers

Task and expected result	Example
1. Early warning of positions that may be growing dangerous.	Filtering stocks for mutual fund portfolios using neural networks (Fidelity Investment funds). The neural network indicates stocks suitable for further consideration. The system, retrained daily, suggests consideration of 200-400 stocks instead of the initial 2000 stocks. There are about 100 input variables including stock prices, dividends, historical earnings, balance sheet information, and inputs from Fidelity analysts.
2. Automated choosing of securities with certain desired characteristics.	
3. Screening and selecting securities faster and more accurately.	

Various publications have estimated the use of data mining methods like **hybrid architectures of neural networks with genetic algorithms, chaos theory, and fuzzy logic** in finance. “Conservative estimates place about \$5 billion to \$10 billion under the direct management of neural network **trading models**. This amount is growing steadily as more firms experiment with and gain confidence with neural networks techniques and methods” [Loof-

bourrow & Loofbourrow, 1995]. Many other **proprietary financial applications** of data mining exist, but are not reported publicly [Von Altrock, 1997; Groth, 1998].

Table 1.2. Data mining tasks for trading manages and traders

Users	Task and expected data mining result
Traders	Early warning of changing trends.
	Finding small patterns in the market, which might otherwise be lost in a torrent of information.
	Discovering the effects of one market on another.
Trading managers	Improving overall performance of trading operations.
	Monitoring compliance of trading operations.
	Improving the consistency of trading operations.

Table 1.3. Knowledge-based (symbolic) systems for trading managers

Task and expected result	Difficulties	Result
Compliance monitoring, (signaling if portfolio managers or traders try to trade beyond allowed limits and restrictions).	Typically resolved during development.	Reported significant success (Putnam Funds, NYSE, Toronto Stock Exchange).

Table 1.4. Traditional knowledge-based (expert) systems for traders

Task and expected result	Difficulties	Result
1. Early warning of changing trends.	Slowness of the extraction of trading rules and their dynamic correction for changing markets.	Only few rules of trading work consistently well across different kinds of markets, e.g., rules that work well for a bull market may perform miserably in a bear market.
2. Finding small patterns in the market.		
3. Discovering the effects of one market on another.		

1.3 Data mining methodologies

1.3.1 Parameters

There are several parameters to characterize Data Mining methodologies for financial forecasting:

1. **Date types.** There are two major groups of data types **attributes** or **relations**. Usually Data Mining methods follow an **attribute-based approach**, also called **attribute-value** approach. This approach covers a wide range of statistical and connectionist (neural network) methods. Less traditional **relational methods** based on **relational data types** are presented in Chapters 4-6.

2. **Data set.** Two major options exist: use the time series itself or use all variables that may influence the evolution of the time series. Data Mining methods do not restrict themselves to a particular option. They follow a fundamental analysis approach incorporating all available attributes and their values, but they also do not exclude a technical analysis approach, i.e., use only a financial time series itself.
3. **Mathematical algorithm (method, model).** A variety of statistical, neural network, and logical methods has been developed. For example, there are many neural network models, based on different mathematical algorithms, theories, and methodologies. Methods and their specific assumptions are presented in this book.

Combinations of different models may provide a better performance than those provided by individuals [Wai Man Leung et al., 1997]. Often these models are interpreted as **trained “experts”**, for example trained neural networks [Dietterich, 1997], therefore combinations of these **artificial experts** (models) can be organized similar to a consultation of real **human experts**. We discuss this issue in Section 3.1. Moreover, artificial experts can be effectively combined with real experts in this consultation. Another new terminology came from recent advances in Artificial Intelligence. These experts are called **intelligent agents** [Russel, Norvig, 1995]. Even the next level of hierarchy is offered “experts” learning from another already trained artificial experts and human experts. We use the new term **“expert mining”** as an umbrella term for extracting knowledge from “experts”. This issue is covered in Sections 2.7 and 2.8.

Assumptions. Many data mining methods assume a functional form of the relationship being modeled. For instance, the linear discriminant analysis assumes linearity of the border, which discriminates between two classes in the space of attributes. Relational Data Mining (RDM) algorithms (Chapters 4-6) do not assume a **functional form** for the relationship being modeled is known in advance. In addition, RDM algorithms do not assume the existence of **derivatives**. RDM can automatically **learn symbolic relations on numerical data of financial time series**.

Selection of a method for discovering regularities in financial time series is a very complex task. Uncertainty of problem descriptions and method capabilities are among the most obvious difficulties in the process of selection. We argue for relational data mining methods for financial applications using the concept of dimensions developed by Dhar and Stein [1997a, 1997b]. This approach uses a specific set of terms to express advantages and disadvantages of different methods. In Table 1.7, RDM is evaluated using these terms as well as some additional terms.

Bratko and Muggleton [1995] pointed out that **attribute-based learners** typically only accept available (background) knowledge in rather **limited**

form. In contrast **relational learners support general** representation for **background knowledge.**

1.3.2 Problem ID and profile

Dhar and Stein [1997a,b] introduced and applied a unified vocabulary for business computational intelligence problems and methods. A problem is described using a set of **desirable values (problem ID profile)** and a method is described using its **capabilities** in the same terms. Use of unified terms (**dimensions**) for problems and methods allows us to compare alternative methods.

At first glance, such dimensions are not very helpful, because they are vague. Different experts definitely may have different opinions about some dimensions. However, there is consensus between experts about some critical dimensions such as the low explainability of neural networks. Recognition of the importance of introducing dimensions itself accelerates clarification of these dimensions and can help to improve methods. Moreover, the current trend in data mining shows that user prefer to operate completely in terms specific to their own domain. For instance, users wish to send to the data mining system a query like -- what are the characteristics of stocks with the increased price? If the data mining method has a low capacity to explain its discovery, this method is not desirable for that question. Next, users should not be forced to spend time determining a method's capabilities (values of dimensions for the method). This is a task for developers, but users should be able to identify desirable values of dimensions using natural language terms as suggested by Dhar and Stein.

Table 1.5. Comparison of model quality and resources

Dimension	Desirable value for stock price forecast problem	Capability of neural network method
<i>Model Quality</i>		
Accuracy	Moderate	High *
Explainability	Moderate to High	Low
Response speed	Moderate	High
Ease to use logical relations*	High	Low
Ease to use numerical attributes	High	High
<i>Quality of available resources</i>		
Tolerance for noise in data	High	Moderate to high
Tolerance for sparse data	High	Low
Tolerance for complexity**	High	High
Independence from experts	Moderate	High

* With comprehensive training data

** Tolerance for complexity is the degree to which the quality of a system is affected by interactions among the various components of the process.

Neural networks are the most common methods in financial market forecasting. Therefore, we begin for them. Table 1.5 indicates three shortages of neural networks for stock price forecasting related to

- 1. explainability,
- 2. usage of logical relations and
- 3. tolerance for sparse data.

This table is based on the table from [Dhar, Stein, 1997b, p.234] and on our additional feature—usage of logical relations. The last feature is an important for comparison with ILP methods.

Table 1.6. Comparison of engineering and logistical dimensions

Dimension	Desirable value for stock price forecast problem	Capability of neural network method
<i>Quality of system engineering</i>		
Flexibility	High*	High
Scalability	High	Moderate
Compactness	Moderate l l	High
Embeddability	High	High
Independence from experts	High	High
Ease to use	Moderate	Moderate
<i>Logistical constraints</i>		
Computing resources	Low to Moderate	Low
Development speed	Moderate	Moderate

* To be able to adjust to other stocks

Table 1.6 indicates a shortage of neural networks for this problem related to scalability. High scalability means that a system can be relatively easy scaled up to realistic environment from a research prototype. Flexibility means that a system should be relatively easily updated to allow for new investment instruments and financial strategies [Dhar, Stein, 1997a,b].

1.3.3 Comparison of intelligent decision support methods.

Table 1.7 compares different methods in terms of dimensions offered by Dhar and Stein [1997a,b]. We added the gray part to show the importance of relational first-order logic methods. The terms H, M, L represents high, medium and low levels of the dimension respectively.

The abbreviations in the first row represent different methods. IBL means instance-based learning, ILP means inductive logic programming, PILP means probabilistic ILP, NN means neural networks, FL means fuzzy logic. Statistical methods (ARIMA and others) are denoted as ST, DT means decision trees and DR means deductive reasoning (expert systems).

Table 1.7. Comparison of capabilities of methods (adapted from [Dhar, Stein, 1997a,b])

Dimension	NN	IBL	FL	DR	ST	DT	ILP	PILP
1 Accuracy	H ¹	MH	H		MH	MH	H	H
2 Explainability	L ²	M	M	H	MH	MH	H	H
3 Response speed	H	MH	H	LM	MH	H	H	H
4 Scalability	M	H	M	M	MH	MH	M	M
5 Compactness	H	LM	H	L		M	M	M
6 Flexibility	H ⁴	H	H	M	LM	H	H	
7 Embeddability	H	M	M	L	H	MH	MH	MH
8 Tolerance for complexity	H	M	H	L	LM	M	LM	LM
9 Tolerance for noise in data	MH ⁵	M			LM	M	L	H
10 Tolerance for sparse data	L	M				L	L	H
11 Independence from experts	H	MH	M	L	H	M	M	H
12 Development speed	M ⁶		M	M H		M	M	M
13 Used computing resources	LM ⁷		L			M	M	M
14 Ease of use	M		M				M	M
15 Ease of use logical relations	LM	L	M	H	L	L	H	H
16 Ease of use of numerical data	H	H	H	LM	H	H	LM	LM

Comments for Table 1.7 ([Dhar, Stein, 1997a,b]).

¹ Needs comprehensive training data.

² Some methods exist for doing sensitivity analysis and rule extraction.

³ Depends on complexity of problem, availability of data.

⁴ Needs representative training data.

⁵ Preprocessing is useful in dealing with noise.

⁶ Depends on understanding of process, on computer speed, and learning paradigm.

⁷ Scale with respect to amount of data and size of network.

1.4 Modern methodologies in financial knowledge discovery

1.4.1 Deterministic dynamic system approach

Financial data are often represented as a time series of a variety of attributes such as stock prices and indexes. Time series prediction has been one of the ultimate challenges in mathematical modeling for many years [Drake, Kim, 1997]. Currently Data Mining methods try to enhance this study with new approaches.

Dynamic system approach has been developed and applied successfully for many difficult problems in physics. Recently several studies have been accomplished to apply this technique in finance. Table 1.8 presents the major steps of this approach [Alexander and Giblin, 1997].

Selecting attributes (step 1) and discovering the laws (step 2) are largely informal and the success of an entire application depends heavily on this art. The hope of discovering dynamic rules in finance is based on the idea borrowed from physics -- single actions of molecules are not predictable but overall behavior of a gas can be predicted. Similarly, an individual operator in the market is not predictable but general rules governing overall market behavior may exist [Alexander and Giblin, 1997].

Table 1. 8. Steps for development of deterministic dynamic system

Step 1. Development of state space for the dynamic system, i.e., selecting and/or inventing attributes characterizing the system behavior.
Step 2. Discovering the laws that govern the phenomenon, i.e., discovering relations between attributes of current and previous states (state vectors) in the form of differential equations
Step 3. Solving differential equations for identifying the transition function (rules)
Step 4. Use of the transition function as a predictor of the next state of the dynamic system, e.g., next day stock value.

Inferring a set of rules for **dynamic system** assumes that there is:

1. **enough information** in the available data to sufficiently characterize the dynamics of the system with high accuracy,
2. all of **the variables** that influence the time series are **available** or they vary slowly enough that the system can be modeled adaptively,
3. the system has reached some kind of **stationary evolution**, i.e. its trajectory is moving on a well-defined surface in the state space,
4. the system is a **deterministic system**, i.e., can be described by means of differential equations,

5. the evolution of a system can be described by means of a surface in the **space of delayed values**.

There are several applications of these methods to financial time series. However, the literature claims both **for and against** the existence of **chaotic deterministic systems** underlying financial markets [Alexander, Giblin, 1997; LeBaron, 1994].

Table 1.9 summarizes comparison of one of the dynamic systems approach methods (state-space reconstruction technique) [Gershenfeld, Weigend, 1994]) with desirable values for stock market forecast (SP500).

Table 1.9. Comparison of method capability and desirable values

Characteristics	State-space reconstruction technique	Desirable values for stock market forecast (SP500)
Delivering numeric value forecast	Yes	Yes
Inputting samples of the time series	Yes	Yes
Tolerance for significant noise in data	No	Yes
Tolerance for time series generated by high-dimensional differential equations	No	Can be needed
Tolerance for time series generated by a non-differentiable process	No	Yes

State-space reconstruction technique depends on a result in non-linear dynamics called Takens' theorem. This theorem assumes a system of low-dimensional non-linear differential equations that generates a time series. According to this theorem, the whole dynamics of the system can be restored. Thus, the time series can be forecast by solving the differential equations. However, the existence of a low-dimensional system of differential equations is not obvious for financial time series as noted in Table 1.9.

Recent research has focused on methods to distinguish stochastic noise from deterministic chaotic dynamics [Alexander, Giblin, 1997] and more generally on **constructing systems combining deterministic and probabilistic techniques**. Relational Data Mining follows the same direction, moving from classical deterministic first-order logic rules to probabilistic first-order rules to avoid limitations of deterministic systems.

1.4.2 Efficient market theory

The efficient market theory states that it is practically impossible to infer a **fixed long-term global forecasting model** from historical stock market information. This idea is based on the observation that if the market presents some kind of regularity then someone will take advantage of it and the regularity disappears. In other words, according to the efficient market the-

ory, the evolution of the prices for each economic variable is a **random walk**. More formally this means that the variations in price are completely independent from one time step to the next in the long run [Moser, 1994].

This theory does not exclude that hidden **short-term local conditional regularities** may exist. These regularities cannot work “forever,” they should be corrected **frequently**. It has been shown that the financial data are not random and that the efficient market hypothesis is merely a subset of a larger **chaotic market hypothesis** [Drake, Kim, 1997]. This hypothesis does not exclude successful short term forecasting models for prediction of chaotic time series [Casdagli, Eubank, 1992].

Data mining does not try to accept or reject the efficient market theory. Data mining creates **tools**, which can be useful for discovering subtle short-term conditional patterns and trends in wide range of financial data. Moreover, as we already mentioned, we use stock market data in this book not because we reject efficient market theory, but because, in contrast with other financial data, they are not proprietary and are well understood without extensive explanations.

1.4.3 Fundamental and technical analyses

Fundamental and Technical analyses are two widely used techniques in financial markets forecast. A **fundamental analysis** tries to determine **all the econometric variables** that may influence the **dynamics of a given stock price or exchange rate**. For instance, these variables may include unemployment, internal product, assets, debt, productivity, type of production, announcements, interest rates, international wars, government directives, etc. Often it is hard to establish which of these variables are relevant and how to evaluate their effect [Farley, Bornmann, 1997].

A **Technical analysis (TA)** assumes that when the sampling rate of a given economic variable is high, all the information necessary to predict the future values is contained in the **time series itself**. More exactly the technical analyst studies the market for the financial security itself: price, the volume of trading, the open interest, or number of contracts open at any time [Nicholson, 1998; Edwards, Magee, 1997].

There are several difficulties in technical analysis for accurate prediction [Alexander and Giblin, 1997]:

- successive ticks correspond to bids from different sources,
- the correlation between price variations may be low,
- time series are not stationary,
- good statistical indicators may not be known,
- different realizations of the random process may not be available,

- the number of training examples may not be enough to accurately infer rules.

Therefore, the technical analysis can fit short-term predictions for financial time series without great changes in the economic environment between successive ticks. Actually, the technical analysis was more successful in **identifying market trends**, which is much easier than **forecasting the future stock prices** [Nicholson, 1998].

Currently different Data Mining techniques try to incorporate some of the most common technical analysis strategies in pre-processing of data and in the construction of appropriate attributes [Von Altrock, 1997].

1.5 Data mining and database management

Numerous methods for learning from data were developed during the last three decades. However, the interest in data mining has suddenly become intense because of the recent involvement with the field of data base management [Berson, Smith, 1997].

Conventional data base management systems (DBMS) are focused on retrieval of:

1. individual records, e.g., -- Display Mr. Smith's payment on February 5;
2. statistical records, e.g., -- How many foreign investors bought stock X last month?
3. multidimensional data, e.g., -- Display all stocks from the data base with increased price.

Retrieval of individual records is often refereed as **on-line transaction processing** (OLTP). Retrieval of statistical records often is associated with **statistical decision support systems** (DSS) and retrieval of multidimensional data is associated with providing **online analytic processing** (OLAP) and **relational online analytic processing** (ROLAP).

At first glance, the above presented queries are simple, but to be useful for decision-making they should be based on **sophisticated domain knowledge** [Berson, Smith, 1997]. For instance, retrieval of Mr. Smith's payment instead of Mr. Brown's payment can be reasonable if the domain knowledge includes information about previous failures to pay by Mr. Smith and that he is supposed to pay on February 5. Current databases with hundreds of gigabytes make it very hard for users to keep sophisticated domain knowledge updated.

Learning **data mining** methods help to extend the traditional database focus and allows the retrieval of answers for important, but vague questions, which **improve domain knowledge** like:

1. What are the characteristics of stocks with the increased price?
2. What are the characteristics of the Dollar-Mark exchange rate?
3. Can we expect that stock X will go up this week?
4. How many cardholders will not pay their debts this month?
5. What are the characteristics of customers who bought this product?

Answering these questions assumes **discovering** some **regularities** and **forecasting**. For instance, it would be useless to retrieve all attributes of stocks with increased price, because many of them will be the same for stocks with decreased price. Figures 1.1 and 1.2 represent relations between database and data mining technologies more specifically in terms of **data warehouses** and data marts. These new terms reflect the fact that database technology has reached a new level of unification and centralization of very large databases with common format. Smaller specialized databases are called **data marts**.

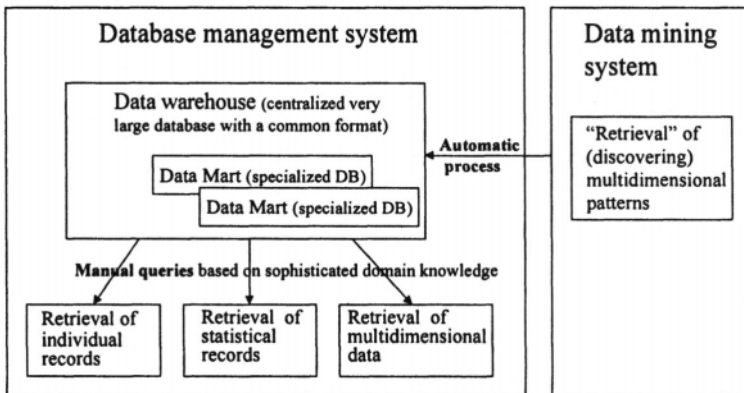


Figure 1.1. Interaction between data warehouse and data mining

Figure 1.1 shows interaction between database and data mining systems. **Online analytic processing** and **relational OLAP** fulfil an important function connecting database and data mining technologies [Groth, 1998]. Currently an OLAP component called the Decision Cube is available in Borland C++ Builder (enterprise edition) as a multi-tier database development tool [DelRossi, 1999]. ROLAP databases are organized by dimension, that is, logical grouping by attributes (variables). This structure is called a **data-cube** [Berson, Smith, 1997]. ROLAP and data mining are intended for **multidimensional analysis**.

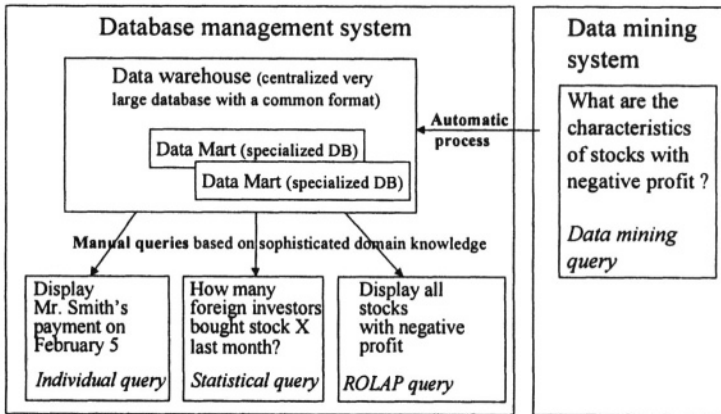


Figure 1.2. Examples of queries

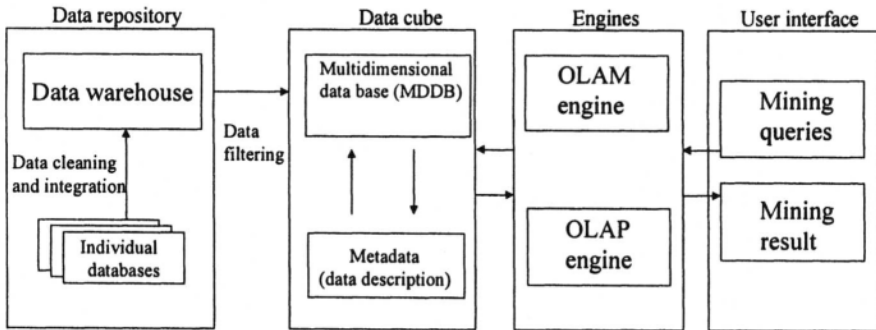


Figure 1.3. Online analytical mining (OLAM)

The next step for the integration of database and data mining technologies is called **online analytical mining (OLAM)** [Han et al, 1999]. This suggests an **automated approach** by combining manual OLAP and fully automatic data mining. The OLAM architecture adapted from [Han et al, 1999] is presented in Figure 1.3.

1.6 Data mining: definitions and practice

Two learning approaches are used in data mining:

1. **supervised (pattern) learning** -- learning with known classes for training examples and
2. **unsupervised (pattern) learning** -- learning without known classes for training examples.

This book is focused on the supervised learning. The common **(attribute-based) representation** of a supervised learning includes [Zighed, 1996]:

- $W=\{w\}$, a sample, called the **training sample**, chosen from a population. Each individual w in W is called a **training example**.
- $X(w)$, the state of n variables known as **attributes** for each training example w .
- $Y(w)$, the **target function** assigning the **target value** for each training example w . Values $Y(w)$ are called **classes** if they represent a classification of training examples.

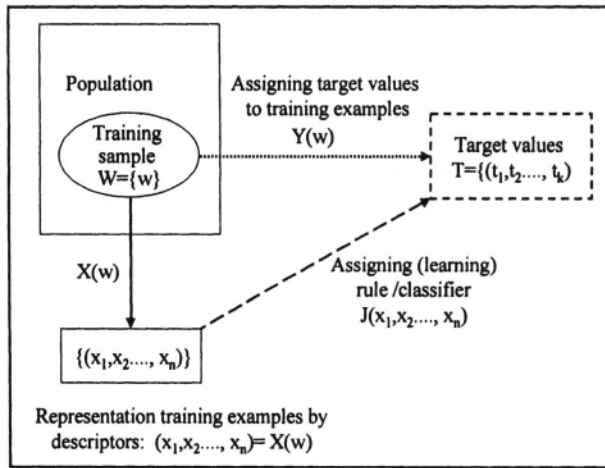


Figure 1.4. Schematic supervised attribute-based data mining model

The aim is to find a **rule (model) J** predicting target the value of the target function $Y(w)$. For example, consider w with unknown value $Y(w)$ but with the state of all its attributes $X(w)$ known:

$$J(X(w))=Y(w),$$

where $J(X(w))$ is a value generated by rule J . It should be done for a **majority** of examples w in W . This scheme adapted from [Zighed, 1996] is shown in 1.4. The choice of a specific data mining method to learn J depends on many factors discussed in this book. The resulting model J can be an algebraic expression, a logic expression, a decision tree, a neural network, a complex algorithm, or a combination of these models.

An **unsupervised data mining model (clustering model)** arises from the diagram in Figure 1.4 by erasing the arrow reflecting $Y(w)$, i.e., the classes are given in advance.

There are several **definitions** of data mining. Friedman [1997] collected them from the data mining literature:

- Data mining is the nontrivial process of identifying valid, novel, potentially useful, and ultimately **understandable patterns** in data. - Fayyad.
- Data mining is the process of extracting previously unknown, comprehensible, and actionable information from large databases and using it to make crucial **business decisions**. - Zekulin.
- Data Mining is a set of methods used in the knowledge discovery process to distinguish previously **unknown relationships** and patterns within data. - Ferruzza.
- Data mining is a decision support process where we look in large databases for unknown and **unexpected patterns** of information. - Parsaye.

Another definition just lists methods of data mining: Decision Trees, Neural Networks, Rule Induction, Nearest Neighbors, Genetic Algorithms.

Less formal, but the most **practical definition** can be taken from the lists of components of current data mining products. There are dozens of products, including, Intelligent Miner (IBM), SAS Enterprise Miner (SAS Corporation), Recon (Lockheed Corporation), MineSet (Silicon Graphics), Relational Data Miner (Tandem), KnowledgeSeeker (Angoss Software), Darwin (Thinking Machines Corporation), ASIC (NeoVista Software), Clementine (ISL Decision Systems, Inc), DataMind Data Cruncher (DataMind Corporation), BrainMaker (California Scientific Software), WizWhy (WizSoft Corporation). For more companies see [Groth, 1998].

The list of components and features of data mining products also collected by [Friedman, 1997] includes attractive GUI to databases (query language), suite of data analysis procedures, windows style interface, flexible convenient input, point and click icons and menus, input dialog boxes, diagrams to describe analyses, sophisticated graphical views of the output, a data plots, slick graphical representations: trees, networks, and flight simulation.

Note that in financial data mining, especially in stock market forecasting and analysis, neural networks and associated methods are used much more often than in other data mining applications. Several of the software packages used in finance include neural networks, Bayesian belief networks (graphical models), genetic algorithms, self organizing maps, neuro-fuzzy systems. Some data mining packages offer traditional statistical methods: hypothesis testing, experimental design, ANOVA, MANOVA, linear regression, ARIMA, discriminant analysis, Markov chains, logistic regression, canonical correlation, principal components and factor analysis.

1.7 Learning paradigms for data mining

Data mining learning paradigms have been derived from machine learning paradigms. In machine learning, the general aim is to improve the performance of some task, and the general approach involves **finding and exploiting regularities in training data** [Langley, Simon, 1995].

Below we describe machine learning paradigms using three components:

- Knowledge representation,
- Forecast performer, and
- Learning mechanism.

Knowledge representation sets a framework for representing prior knowledge. A **forecast performer** serves as a final product, generating a forecast from learned knowledge. A **learning mechanism** produces new knowledge and identifies parameters for the forecast performer using prior knowledge.

Knowledge representation is the major characteristic used to distinguish five known paradigms [Langley, Simon, 1995]:

1. A multilayer network of units. Activation is spread from input nodes to output nodes through internal units (**neural network paradigm**).
2. Specific cases or experiences applied to new situations by matching known cases and experiences with new cases (**instance-based learning, case-based reasoning paradigm**).
3. Binary features used as the conditions and actions of rules (**genetic algorithms paradigm**).
4. Condition-action (IF-THEN) rules, decision trees, or similar knowledge structures. The action sides of the rules or the leaves of the tree contain predictions (classes or numeric predictions) (**rule induction paradigm**).
5. Rules in first-order logic form (Horn clauses as in the Prolog language) (**analytic learning paradigm**).
6. A mixture of the previous representations (**hybrid paradigm**).

The above listed types of knowledge representation largely determine the frameworks for **forecast performers**. These frameworks are presented below [Langley, Simon, 1995]:

1. **Neural networks** use weights on the links to compute the activation level passed on for a given input case through the network. The activation of output nodes is transformed into numeric predictions or discrete decisions about the class of the input.
2. **Instance-based learning** includes one common scheme, it uses the target value of the stored nearest (according to some distance metric) case as a classification or predicted value for the current case.
3. **Genetic Algorithms** share the approach of neural networks and other paradigms, because genetic algorithms are often used to speed up the learning process for other paradigms.

4. **Rule induction.** The performer sorts cases down the branches of the decision tree or finds the rule whose conditions match the cases. The values stored in the if-part of the rules or the leaves of the tree are used as target values (classes or numeric predictions).
5. **Analytical learning.** The forecast is produced through the use of background knowledge to construct a specific combination of rules for a current case. This combination of rules produces a forecast similar to that in rule induction. The process of constructing the combination of rules is called a proof or "explanation" of experience for that case.

The next important component of each of these paradigms is a **learning mechanism**. These mechanisms are very specific for different paradigms. However, search methods like gradient descent search and parallel hill climbing play an essential role in many of these mechanisms.

Figure 1.5 shows the interaction of the components of a learning paradigm. The training data and other available knowledge are embedded into some form of knowledge representation. Then the learning mechanism (method, algorithm) uses them to produce a forecast performer and possibly a separate entity, learned knowledge, which can be communicated to human experts.

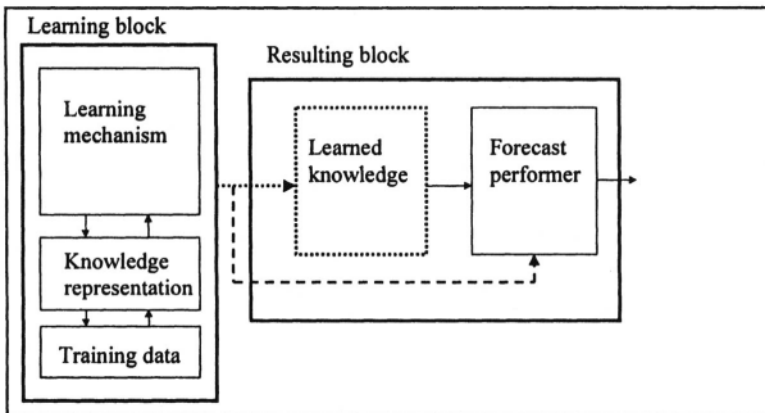


Figure 1.5. Learning paradigm

Neural network learning identifies the forecast performer, but does not produce knowledge in a form understandable by humans, IF-THEN rules. The rule induction paradigm produces learned knowledge in the form of understandable IF-THEN rules and the forecast performer is a derivative from this form of knowledge.

Steps for learning. Langley and Simon [1995] pointed out the general steps of machine learning presented in Figure 1.6. In general, data mining follows these steps in the learning process.

These steps are challenging for many reasons. Collecting training examples has been a bottleneck for many years. Merging database and data mining technologies evidently speeds up collecting the training examples. Currently, the least formalized steps are reformulating the actual problem as a learning problem and identifying an effective knowledge representation.

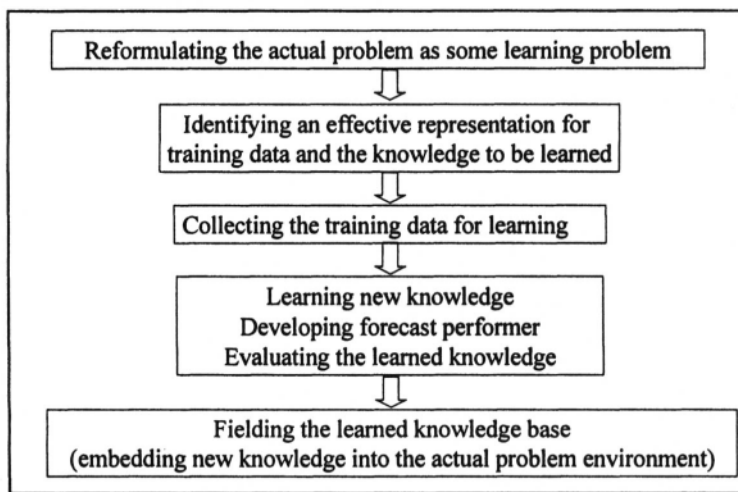


Figure 1.6. Data mining steps

1.8 Intellectual challenges in data mining

The importance of identifying an effective knowledge representation has been hidden by the data-collecting problem. Currently it has become increasingly evident that the effective **knowledge representation** is an important problem for the success of data mining. Close inspection of **successful projects** suggests that much of the power comes not from the specific induction method, but from proper formulation of the problems and from crafting the representation to make learning tractable [Langley, Simon, 1995]. Thus, the **conceptual challenges** in data mining are:

- Proper **formulation of the problems** and
- Crafting the **knowledge representation** to make learning meaningful and tractable.

In this book, we specifically address conceptual challenges related to knowledge representation as related to **relational date mining and date types** in Chapters 4-7.

Available data mining packages implement well-known procedures from the fields of machine learning, pattern recognition, neural networks, and data visualization. These packages emphasize **look and feel** (GUI) and the existence of **functionality**. Most academic research in this area so far has focused on incremental **modifications** to current machine learning methods, and the **speed-up** of existing algorithms [Friedman, 1997].

The current trend shows three new **technological challenges** in data mining [Friedman, 1997]:

- Implementation of data mining tools using **parallel computation of on-line queries**.
- **Direct interface** of DBMS to data mining algorithms.
- **Parallel** implementations of basic **data mining algorithms**.

Some of our advances in parallel data mining are presented in Section 2.8.

Munakata [1999] and Mitchell [1999] point out four especially promising and challenging areas:

- incorporation of background and associated knowledge,
- incorporation of more comprehensible, non-oversimplified, real-world types of data,
- human-computer interaction for extracting background knowledge and guiding data mining, and
- hybrid systems for taking advantage of different methods of data mining.

Fu [1999] noted “Lack of comprehension causes concern about the credibility of the result when neural networks are applied to **risky domains**, such as patient care and financial investment”. Therefore, the development of a special neural network whose knowledge can be decoded faithfully is considered as a promising direction [Fu, 1999].

It is important for the future of data mining that the current growth of this technology is stimulated by requests from the database management area. The database management area is neutral to the learning methods, which will be used. This already has produced an increased interest for hybrid learning methods and cooperation among different professional groups developing and implementing learning methods.

Based upon new demands for data mining and recent achievements in information technology, the significant intellectual and commercial future of the data mining methodology has been pointed out in many recent publications (e.g., [Friedman, 1997; Ramakrishnan, Grama, 1999]). R. Groth [1997] cited “Bank systems and technology” [Jan., 1996] which states that data mining is the most important application in financial services.

Chapter 2

Numerical Data Mining Models and Financial Applications

There's always an easy solution to every human problem -- neat, plausible, and wrong.

Henry Louis Mencken

2.1. Statistical, autoregression models

Traditional attempts to obtain a short-term forecasting model of a particular financial time series are associated with statistical methods such as ARIMA models.

In sections 2.1 and 2.2, **ARIMA regression models** as typical examples of the **statistical approach** to financial data mining [Box, Jenkins, 1976; Montgomery et al, 1990] are discussed.

Section 2.3 contains **instance-based learning (IBL)** methods and their financial applications. Another approach, sometimes called “**regression without models**” [Farlow, 1984] does not assume a class of models. Neural networks described in sections 2.4-2.8 exemplify this approach [Worbois, 1975]. There are sensitive assumptions behind of these approaches. We discuss them and their impact on forecasting in this chapter.

Section 2.9 is devoted to “**expert mining**”, that is, methods for extracting knowledge from experts. Models trained from data can serve as artificial “experts” along with or in place of human experts.

Section 2.10 describes background mathematical facts about the restoration of monotone Boolean functions. This powerful mathematical mechanism is used to speed up the testing of learned models like neural networks in section 2.8 and “expert mining” methods in section 2.9.

2.1.1. ARIMA Models

Flexible ARIMA models were developed by Box and Jenkins [Box, Jenkins, 1976] ARIMA means **AutoRegressive Integrated Moving Average**. This name reflects three components of the ARIMA model. Many data mining and statistical systems such as SPSS and SAS support the computations needed for developing ARIMA models. Brief overview of ARIMA modeling is presented in this chapter. More details can be found in [Box, Jenkins, 1976; Montgomery et al, 1990] and manuals for such systems as SPSS and SAS.

ARIMA models include three processes:

1. **autoregression (AR)**;
2. **differencing** to eliminate the integration (**I**) of series, and
3. **moving average (MA)**.

The general ARIMA model combines autoregression, differencing and moving average models. This model is denoted as **ARIMA(p,d,q)**, where
 p is the **order of autoregression**,
 d is the **degree of differencing**, and
 q is the **order of the moving average**.

Autoregression. An autoregressive process is defined as a linear function matching p preceding values of a time series **V(t-1), V(t-2),...,V(t-p)** with **V(t)**, where **V(t)** is the **value** of the time series at the moment t.

In a first-order autoregressive process, only the preceding value is used. In higher order processes, the p preceding values are used. This is denoted as **AR(p)**. Thus, **AR(1)** is the first-order autoregressive process, where:

$$V(t) = C + g_1 V(t-1) + D(t).$$

Here C is a constant term related to the **mean** of the process, and D(t) is a function of t interpreted as a **disturbance** of the time series at the moment t. The coefficient **g₁** is estimated from the observed series. It shows the **correlation** between V(t) and **V(t-1)**.

Similarly, a **second order autoregression, AR(2)**, takes the form below, where the two preceding values are assumed to be independent of one another:

$$V(t) = C + g_1 V(t-1) + g_2 V(t-2) + D(t).$$

The autoregression model, **AR(p)**, is the same as the **ARIMA(p,0,0)** model:

$$V(t) = C + g_1 V(t-1) + g_2 V(t-2) + \dots + g_p V(t-p) + D(t).$$

Differencing. Differencing substitutes each value by the difference between that value and the preceding value in the time series. The first difference is

$$W(t)=V(t)-V(t-1).$$

The standard notation for models based on the first difference is **I(1)** or **ARIMA(0,1,0)**. Similarly, **I(2)** or **ARIMA(0,2,0)** models are based on the second difference:

$$Z(t)=W(t)-W(t-1).$$

Next we can define the third difference:

$$Y(t)=Z(t)-Z(t-1)$$

for **I(3)**. As we already mentioned, the parameter **d** in **I(d)** is called the **degree of differencing**.

The **stationarity** of the differences is required by ARIMA models. Differencing may provide the stationarity for a derived time series, $W(t)$, $Z(t)$ or $Y(t)$. For some time series, differencing can reflect a meaningful empirical operation with real world objects. These series are called **integrated**. For instance, trade volume measures the cumulative effect of all buy/sell transactions.

An **I(1)** or **ARIMA(0,1,0)** model can be viewed as an **autoregressive model**, **AR(1)** or **ARIMA(1,0,0)**, with a regression coefficient **g=1**:

$$V(t) = gV(t-1) + D(t).$$

In this model (called **random walk**), each next value is only a random step $D(t)$ away from the previous value. See chapter 1 for a financial interpretation of this model.

Moving averages. In a moving-average process, each value is determined by the weighted average of the current **disturbance** and q previous disturbances. This model is denoted as a **MA(q)** or **ARIMA(0,0,q)**. The equation for a first-order moving average process **MA(1)** is:

$$V(t) = C + D(t) + s_1D(t-1),$$

MA(2) is:

$$V(t) = C + D(t) + s_1D(t-1)+s_2D(t-2),$$

and $MA(q)$ is:

$$V(t) = C + D(t) + s_1 D(t-1) + s_2 D(t-2) + \dots + s_q D(t-q)$$

The major difference between $AR(p)$ and $MA(q)$ models is in their components: $AR(p)$ is averaging the p most recent values of the time series while $MA(q)$ is averaging the q most recent random disturbances of the same time series.

The combination of $AR(1)$ and $MA(1)$ creates an **ARMA(1,1)** model, which is the same as the **ARIMA(1,0,1)**, is expressed as

$$V(t) = C + g_1 V(t-1) + s_1 D(t-1) + D(t)$$

The more general model **ARMA(p,q)**, which is the same as **ARIMA(p,0,q)** is:

$$V(t) = C + g_1 V(t-1) + g_2 V(t-2) + \dots + g_p V(t-p) + D(t) + s_1 D(t-1) + s_2 D(t-2) + \dots + s_q D(t-q).$$

Now we can proceed to introduce the third parameter for differencing, for example, the **ARIMA(1,1,1)** model:

$$V(t) - V(t-1) = C + g_1 (V(t-1) - V(t-2)) + D(t) + s_1 D(t-1)$$

or equivalently

$$W(t) = C + g_1 W(t-1) + D(t) + s_1 D(t-1),$$

where $W(t) = V(t) - V(t-1)$ in the **first difference**, that is, **d=1**

Similarly, **ARIMA(1,2,1)** is represented by:

$$Z(t) = C + g_1 Z(t-1) + D(t) + s_1 D(t-1),$$

where

$$Z(t) = W(t) - W(t-1)$$

is the **second difference (d=2)**, and the **ARIMA(1,3,1)** model is

$$Y(t) = C + g_1 Y(t-1) + D(t) + s_1 D(t-1),$$

where $Y(t) = Z(t) - Z(t-1)$ is the **third difference (d=3)**

Generalizing this we see the **ARIMA(p,3,q)** model is:

$$Y(t) = C + g_1 Y(t-1) + g_2 Y(t-2) + \dots + g_p Y(t-p) + D(t) + s_1 D(t-1) + s_2 D(t-2) + \dots + s_q D(t-q) Y(t).$$

In practice, d larger than 2 or 3 are used very rarely [Pankratz, 1983].

2.1.2. Steps in developing ARIMA model

Box and Jenkins [1976] developed a **model-building procedure** that allows one to construct a model for a series. However, this procedure is not a formal computer algorithm. It requires user's decisions in several critical points. The procedure consists of three steps, which can be repeated several times:

- Identification,
- Estimation, and
- Diagnosis.

Identification is the first and most subjective step. The three integers p,d,q in the ARIMA(p,d,q) process generating the series must be determined. In addition, **seasonal variation** parameters can be incorporated into the ARIMA models. Seasonal ARIMA models are discussed later in section 2.1.3.

ARIMA models are applied only to time series that have essentially constant mean and variance through time [Pankratz, 1983]. These series are called **stationary**. **Integrated series are typically non-stationary**. In this case, the time series should be transformed into a stationary one, using differencing or other methods. Logarithmic and square-root transformations are used if the short-term variation of the time series is proportional to the time series value, $V(t)$. Next, we must identify p and q, the order of autoregression and of moving average. In a non-seasonal process:

- both p and q are usually less than 3 and
- the **autocorrelation function (ACF)** and **partial autocorrelation function (PACF)** of a series help to **identify** the p and q.

Below ACF and PACF of a series are described. In practice, ACF and PACF are computed using a given part of the entire time series, therefore, they are merely estimated ACF and PACF.

This is an important note, because if a given part of the time series is not representative for the future part, correlation parameters including ACF and PACF will be **misleading**. ACF is based on the standard Pearson's **correlation coefficients** applied to the time series with a lag. Recall with two inde-

pendent time series x and y , their correlation coefficient $r(x,y)$ is computed, e.g., [Pfaffenberger, Patterson, 1977]:

$$r = r(x, y) = \frac{\sum_{i=1}^n (x_i - M(x))(y_i - M(y))}{\sqrt{\sum_{i=1}^n (x_i - M(x))^2} \sqrt{\sum_{i=1}^n (y_i - M(y))^2}},$$

where $M(x)$ is the mean of $\{x\}$ and $M(y)$ is the mean of $\{y\}$.

Instead, we consider correlation between consecutive values of the same time series with lag k [Pankratz, 1983]:

$$r(k) = r(V(t), V(t+k)) = \frac{\sum_{t=1}^n (V(t) - M(V))(V(t+k) - M(V))}{\sqrt{\sum_{t=1}^n (V(t) - M(V))^2} \sqrt{\sum_{t=1}^n (V(t+k) - M(V))^2}},$$

where $M(V)$ is the mean of $V(t)$, $t=1, \dots, n$.

For large n ($n \gg k$) this formula can be simplified:

$$r(k) = r(V(t), V(t+k)) = \frac{\sum_{t=1}^n (V(t) - M(V))(V(t+k) - M(V))}{\sum_{t=1}^n (V(t) - M(V))^2},$$

Now, if the time series are considered as fixed and k as varying from 1 to $(n-1)$, then r_k or $r(k)=r(V(t), V(t+k))$ is a function of k and is called **estimated ACF**. The idea of PACF is to modify ACF to be able to incorporate not only correlation between $V(t)$ and $V(t+k)$ but also the impact of all values in between $V(t+1)$, $V(t+2)$, ..., $V(t+(k-1))$ [Pankratz, 1983]:

$$\phi(k, k) = \phi(k) = \frac{r(k) - \sum_{j=1}^{k-1} \phi(k-1, j)r(k-j)}{1 - \sum_{j=1}^{k-1} \phi(k-1, j)r(j)} \quad (k = 2, 3, \dots)$$

where $\phi(1,1)=r(1)$ and $\phi(k,j)=\phi(k-1,j)-\phi(k,k)\phi(k-1,k-j)$, $k=3,4,\dots,j=1,2,\dots,k-1$. The function $\phi(k)$ is called an **estimated partial autocorrelation function (PACF)**.

A user can conclude that ARIMA model parameters p , q and d are acceptable by analyzing the behavior of ACF and PACF functions. For more detail see [Pankratz, 1983].

Estimation. Assume that parameters p , d , and q are preliminarily identified. Then coefficients of the ARIMA model are estimated along with the error of the model (residual). These estimates are accompanied by statistical parameters like the confidence limits, the standard error of the coefficients, and the statistical significance of the coefficients. The process can be repeated for different p , d , and q , if the identification of the model is uncertain. We have already mentioned that identification of p , d , and q is the most subjective step, therefore, several alternative values of p , d , and q can be used.

Diagnosis. This step checks to see if the model is appropriate. A user tests that:

- the ACF and PACF are “mostly” close to 0,
- the first- and second-order correlations are “small” (if one of them is large, then the model is probably incorrectly specified),
- the residual time series checked by ACF and PACF shows “no pattern”, that is, white noise (this can be tested by the Box-Ljung Q statistics).

Some data mining tools provide the ARIMA with several criteria for choosing among models. Traditionally, final ARIMA models substitute zero for the computed coefficients when they are not statistically significant. However, recently an alternative approach has come into favor -- accepting the best-fitting model with coefficients that are insignificant according to a simple statistical test. We discuss this issue in section 2.2 below.

2.1.3. Seasonal ARIMA

It is possible that a time series has periodic regularity. For instance, there is the known January effect in the stock market (see chapter 5). Therefore, a period can be 12 as with a year or 7 as with a week.

The XII ARIMA algorithm estimates **seasonal factors** with ARIMA forecasts and back casts. There are three steps in this algorithm:

- Select variables,
- Select multiplicative, additive, or logarithmic seasonal options,
- Adjust extreme components of the process.

Multiplicative Seasonal Adjustment means that the seasonally adjusted series is multiplied to produce the original series.

Additive Seasonal Adjustment means that the seasonal adjustments are added to the seasonally adjusted series to obtain the observed values.

Logarithmic Seasonal Adjustment means that the seasonal adjustments are added to the seasonally adjusted series in the logarithmic scale.

2.1.4. Exponential smoothing and trading day regression

The trend and seasonal components can be specified in custom models available in statistical software like SPSS. For example, exponential smoothing levels irregular components for the time series with the following characteristics:

- a linear trend and no seasonal variation;
- a linear trend and multiplicative seasonal variation;
- the mean level of the series increases at a constant rate with time;
- the mean level of the series increases exponentially with time;
- the mean level of the series increases with time, but the rate of change declines;
- the magnitude of seasonal variation does not depend on the overall level of the series;
- the magnitude of seasonal variation depends on the overall level of the series.

This smoothing is done by optimization of parameters that control trend and seasonal components. A **grid search** is used to find parameters for the **best “smooth” model**. Each node of the grid is associated with a model; therefore, a large number of models should be analyzed to find the best one. Typical **optimization parameters** are:

- The relative weight (α) assigned to recent observations, as opposed to the overall series mean.
- Trend value (β) is the relative weight given to recent observations in estimating the present series trend.
- Seasonal value (δ) is the relative weight given to recent observations in estimating the present seasonality.
- Trend modification value (τ) is the rate at which a trend is reduced in magnitude over time.

Trading Day Regression. Seasonal XII ARIMA model is based on an **even periodicity**. This assumption simplifies the seasonal ARIMA model, but can make this model unrealistic and useless. For instance, the number of **trading days** in different months is not a constant. It can be 20, 21 or 22 days. Day adjustment controls the computation of the trading day ARIMA model and day weights. For instance, trading weekdays may get a weight of 1.4 and Saturday and Sunday may get a weight of 0.

2.1.5. Comparison with other methods.

As we have seen, the use of ARIMA models requires many **individual adjustments** like adjustments to the number of trading days. There are two

ways to design ARIMA models, automatic and custom models, offered by software tools like SPSS. Selecting the best automatic model often means fitting three default ARIMA models to the series and selecting the one that fits best. In custom models, a user specifies the model parameters: autoregressive (p), difference (d), and moving average (q). A user could also specify the corresponding seasonal parameters by entering them into the model. This shows that ARIMA models and statistical models, in general, are **sensitive to expert decisions** about parameters. Table 1.7 summarizes many of the differences between a variety of data mining methods. The ARIMA method was evaluated in the category of statistical methods (SM) in the column marked ST. According to [Dhar, Stein, 1977] this column indicates the two **strongest** features of SM: **embeddability** and **independence of an expert in comparison with other methods**.

Embeddability of SM into application software systems is really its most attractive and indisputable feature. SM software and open codes are widely available and their runtimes are not prohibitive for many real tasks.

Independence of an expert is relatively high in comparison with neuro-fuzzy and some other methods. However, as we discussed above, tuning ARIMA models is an art and an expert is integral and the most important part of this process.

With SM it is well known that it is easy to use **numerical data**. Two other features are indicated as **weakest** features: **flexibility** and use of **logical expressions**.

The last feature is crucial for developing **hybrid methods combining statistical (probabilistic) methods and first-order logic** methods like MMDR (chapters 4 and 5). These methods are marked as PILP (Probabilistic Inductive Logic Programming) methods in table 1.7. Moreover, in tasks with switching regularities, flexibility and logical switches are necessary. For instance, switching from a bull trend to a bear trend requires both of these features.

Other features of ARIMA and other statistical methods are on the average level in comparison with alternative methods (table 1.7 in chapter 1). There is only one feature in table 1.7 making statistical methods unique. This is the possibility to estimate the **statistical significance** of a learned model. If assumptions of the statistical test are reasonable for the task, then a positive test tells us about **predictive power** of the model as well as about its performance on training and testing data.

2.2. Financial applications of autoregression models

ARIMA models can be applied for forecasting a time series if the transformed time series is stationary. However, in financial markets, often time series like stock indices, foreign exchange trends and others are all non-stationary even for the short-term trend [Drake, Kim, 1997]. At present, there is no convenient way to modify or update the estimates of the model parameters as each new observation become available [Montgomery, et al., 1990].

ARIMA models implicitly assume that a strong and relatively simple statistical regularity exists in a time series. Unfortunately, this is not a very realistic assumption for many financial time series. Therefore, the successful use of ARIMA is still an individual art rather than a regular procedure. Moreover, the possibility to present all actual regularities in one simple formula is questionable for many time series.

Let us suppose that there are 130 local regularities in the time series. What is the chance to that one will be able to compress 130 regularities into one simple autoregression formula? To use ARIMA we need to identify these local areas and adjust many parameters. For example, suppose n local intervals $[A_1, B_1], [A_2, B_2], \dots, [A_n, B_n]$ are given along with n autoregression functions $F_1, F_2, \dots, F_n$ such that,

IF $A_i < x < B_i$ THEN $F_i(x)$, $i=1, 2, \dots, n$,

we need to find all A_i and B_i values as parameters. This can be a nontrivial task, especially for a large n .

However, despite these difficulties, ARIMA and related methods have been **competitive** with newer methods like neural networks in financial applications. Examples in open competition using the same data and evaluation criteria can be found in the Journal of Computational Intelligence in Finance [Non-linear Financial Forecasting, 1997].

The important specifics of data mining methods in finance and the stock market, in particular, is that often **forecasting is not the final product**. Forecasting is used as a basis in trading strategies (management decisions). For instance, consider the following trading strategies:

- selling the security and then buying it back at a lower price,
- taking the cash proceeds from the sale and putting them to work in a savings account or any other investment,
- owning the stock long-term (passive buy-and-hold strategy).

The performance of these strategies depends on parameters for overall return like the prices at which the transactions are executed, transaction costs and dividends paid by the stock.

Therefore, **trading (management) performance rather than forecasting accuracy should measure the success of the forecast.** This attractive, objective measure of success has a drawback -- an inappropriate trading strategy weakens the potential gain from an accurate forecast.

On the other hand, even a very subtle forecasting result combined with an appropriate trading strategy can bring a significant profit.

For instance, ARIMA models shown in table 2.1 below have a subtle statistical significance, but some of them (Models 4 and 5) were able to produce correct buy/hold/sell signals in 75-79% of cases in simulated trading for two years.

The ARIMA models examined are presented in Table 2.1, where s is the periodic parameter (**$s=5$ days**), t is the day, $T(t)$ is the target stock for day t , and a, b, c and q are model coefficients. These coefficients were evaluated for models 3-5 using non-linear optimization methods and test data (1995-1996 years). The sign of $T(t+1)-T(t)$ was predicted with high accuracy, but an absolute value was not predicted accurately.

Table 2.1. Experiments with ARIMA models for stock forecasting on 1995-96 data

#	Model	Forecasting performance (correct buy/sell signal)
1	$T(t+1)=T(t)+\varepsilon$, t : all trading days	Not applicable
2	$T(t+1)=aT(t)+b$, t : all trading days	62.58%
3	$T(t+4)=(1q)T(t+3)+qT(t+2)+q^2T(t)+c$, t : all trading days	58.84%
4	$T(t+s)=aT(t)+b$, t : the specific weekday, $s=5$	79.6%
5	$T(t+2s)=aT(t+s)+bT(t)+c$ t : the specific weekday, $s=5$	75.92%

However, correct forecast of the sign is sufficient to form a successful buy/sell trading strategy. The **sign forecast** is simpler than absolute value forecast and first-order logic methods (Chapters 4 and 5) fit to discover sign forecast rules. Model #1, called a random walk model, was reviewed briefly in chapter 1. This model is "...a good ARIMA model for many stock-price series" [Pancratz, 1983, p.410]. Nevertheless, this model is not applicable to interesting trading strategies. It does not produce ups and downs needed for developing those trading strategies. This model has zero difference for all days. Parameters for the most successful Models 4 and 5 were discovered using the relational data mining approach and the MMDR algorithm described in Chapters 4 and 5.

In Chapter 1 (Table 1.7) we presented comparative capabilities of different data mining methods based on [Dhar, Stein, 1997]. According to Dhar and Stein, accuracy, explainability, response speed, and scalability of statistical methods including ARIMA are medium in comparison with other methods. On the other hand, flexibility, tolerance for noise and complexity are rather low in comparison with other methods. Only one parameter, possibility to embed into a larger system, was graded as high.

2.3. Instance-based learning and financial applications

The literature considers two similar concepts **instance-based learning (IBL)** and **case-based reasoning** [Mitchell, 1997]. Instance-based learning covers methods based on objects presented by a **set of numeric attributes** of a fixed size. Case-based reasoning allows one the use of more **general objects**. In this section, we focus on instance-based learning. In ILB objects are usually embedded into the n-dimensional Euclidean space with measuring the distance between data objects

$$D=\{\mathbf{d}_i\}, \mathbf{d}_i=(d_{i1}, d_{i2}, \dots, d_{in}), i=1, \dots, m,$$

by the **Euclidean distance** $\rho(\mathbf{d}_i, \mathbf{d}_j)$:

$$\rho(\mathbf{d}_i, \mathbf{d}_j) = \sqrt{\sum_{s=1}^m (d_{is} - d_{js})^2}.$$

Then the target value for a new object $\mathbf{d}$ is assigned according to the target values of some training objects. Usually, it is done selecting **k nearest neighbours** $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_k$ [Mitchell, 1997], i.e., objects with minimal $\rho(\mathbf{d}, \mathbf{d}_i)$:

$$\text{IF } \mathbf{d}_i \neq \mathbf{d}; \text{ THEN } \rho(\mathbf{d}, \mathbf{d}_i) > \rho(\mathbf{d}, \mathbf{d}_i) \text{ for all } i=1, \dots, k.$$

One of the simplest approaches is to assign a target value for $\mathbf{d}$ by averaging the target values of the k nearest neighbors if the target is continuous variable:

$$T(\mathbf{d}) = (T(\mathbf{d}_1) + T(\mathbf{d}_2) + \dots + T(\mathbf{d}_k)) / k$$

Alternatively, each object $\mathbf{d}_i$ can be weighted proportionally to its distance from $\mathbf{d}$:

$$T(\mathbf{d}) = (w_1 T(\mathbf{d}_1) + w_2 T(\mathbf{d}_2) + \dots + w_k T(\mathbf{d}_k)) / k$$

Similarly, if the target variable presents a discrete classification, then a weighted **majority voting** mechanism is used:

$$\text{class} = \begin{cases} 1, N_1 > N_2 \\ 0, N_2 > N_1 \\ \text{no classification otherwise.} \end{cases}$$

Here $N_1 = w_1 C_{11}(d) + \dots + w_i C_{1i}(d) + \dots + w_k C_{1k}(d)$, where w_i is a weight of the i^{th} classifier and $C_{1i}(d) = 1$ if i^{th} classifier classified d into the first class. Similarly, $N_2 = w_1 C_{21}(d) + \dots + w_i C_{2i}(d) + \dots + w_k C_{2k}(d)$, where $C_{2i}(d) = 1$ if the i^{th} classifier classified d into the second class.

It is well known that the result of instance-based learning can be very **sensitive** to parameters like k and w_i and **normalization** of attributes. For instance, stock prices and trade volumes are measured in dollars. Let $d_1 = \langle 35, 45000 \rangle$ and $d_2 = \langle 45, 56000 \rangle$ represent data for two stocks in the format $\langle \$\text{price}, \$\text{volume} \rangle$. Then $\rho(d_1, d_2) = 11000.00455$, but if we exclude stock prices from d_1 and d_2 descriptions and consider $d_1 = \langle 45000 \rangle$ and $d_2 = \langle 56000 \rangle$ then $\rho(d_1, d_2) = 11000.00$. Therefore contribution of stock price into the Euclidean distance is miserable (less than $5 \cdot 10^{-3}$). Thus, it is obvious that normalization is needed in order to use information about differences in the stock prices, but the result will be sensitive to the particular normalization used.

Two possible normalizations are:

1. normalization based on max values of stock price and volume in the sample:

$$\text{StockPrice}(d_i) / \max \text{StockPrice}, \quad \text{StockVolume}(d_i) / \max \text{StockVolume}$$

2. normalization based on averages in the sample:

$$\text{StockPrice}(d_i) / \text{AverageStockPrice}, \\ \text{StockVolume}(d_i) / \text{AverageStockVolume}.$$

Similarly, specific normalizations and related data coding are needed for many data types. More about different coding schemes associated with meaningful distances and advanced case-based learning methods are presented, for instance, in [Kovalerchuk, 1973, 1975, 1997; Hattoru, Torri, 1993; Kamgar-Parsi, Kanal, 1985].

The voting schemes can be extended outside of the k neighbors to use more complex estimates than presented above. One of methods (**AVO**

method) uses combinatorial estimates for voting [Zhuravlev, Nikiforov, 1971]:

$$\text{class} = \begin{cases} 1, G_1 > G_2 + \delta_1 & \& G_1/(G_1 + G_2) > \delta_2 \\ 0, G_2 > G_1 + \delta_1 & \& G_2/(G_1 + G_2) > \delta_2 \\ \text{no classification, else} \end{cases}$$

where δ_1 and δ_2 are voting thresholds and G_1 is a combinatorial estimate of closeness of object d to the objects of class #1 using distance between d and all objects of the class #1 $\{d_i^1\}$. Similarly, G_2 estimates closeness of d to the class #2. Below we present formulas for G_1 and G_2 , where k is a parameter:

$$G_1 = \sum_{i=1}^{m_1} \binom{n - \rho(d_i^1, d)}{k}, \quad G_2 = \sum_{i=1}^{m_2} \binom{n - \rho(d_i^2, d)}{k}.$$

All distances are viewed as integers, they are computed using **Hamming distance** $H(d, d_i)$ with thresholds $e_1, e_2, \dots, e_n$:

$$H(d_i, d_j) = \sum_{s=1, n} \Delta_s,$$

where Δ_s is a threshold difference:

$$\Delta_s = \begin{cases} 1, \text{ if } |d_{is} - d_{js}| \geq e_s \\ 0, \text{ if } |d_{is} - d_{js}| < e_s \end{cases} \quad (s = 1, 2, \dots, n).$$

This case-based method was studied and used intensively: In particular, it has been shown in [Kovalerchuk, 1977] that this method is sensitive to backcasting. Backcast means the forecast of known attributes values d_{is} which have not been used for training. For instance, a backcast value d_{is} depends on an entered hypothetical target value T . At first glance, it appears to not be a very useful property. However, for non-stationary training data this is the way to “**extend**” the sample. Extending the data allows for a better testing of a discovered regularity. In this way, 10 attributes and 1000 cases can be used to obtain $10 \cdot 1000 = 10000$ values for backcasting, because for the training data any value of the attributes, not only the target, can be backcast. This approach has been used successfully with the AVO instance-based learning method for **classification of court decisions** with a scarce data set [Kovalerchuk et al, 1977].

Case-based reasoning covers instance-based learning relaxing the requirement of embedding data into Euclidean space. Other presentation of data are permitted and alternative measures of closeness can be used [Mitchell, 1997].

Financial Time Series Forecasting using k-Nearest Neighbors Classification. Maggini et al. [1997] transform the prediction problem into a classification task and use an approach based on the k-nearest neighbors algorithm to obtain the **most probable variation with respect to the present price value**. Often predicting the actual price with high accuracy is unrealistic. Therefore, classes of prices are predicted instead of values. It is based on the past variations and other variables that might be correlated. In addition, the classification is based on a set of examples extracted from the previous values of the series. The classification is made adaptively -- new values are inserted in the prototype set when they are available and the old values are discarded. This method learns quickly because it is sufficient to memorize all the values contained in the fitting set [Maggini et al., 1997]

Comparison with other methods. A summary of differences between data mining methods was presented in Table 1.7 (Chapter 1). The instance-based learning methods were evaluated in the column marked IBL. According to [Dhar, Stein, 1977] this column indicates the two **strongest** features of IBR: **scalability and flexibility**. We added to Table 1.7 that it is easy to use **numerical data** with ILB. Two other features are indicated as **weakest** features: **low compactness** and use of **logical expressions**. We added the last feature as crucial for developing explainable models like **hybrid models combining statistical (probabilistic) inference and first-order logic** like MMDR (Chapters 4 and 5). These methods are marked as PILP (Probabilistic Inductive Logic Programming) methods in Table 1.7. The flexibility and scalability of instance-based learning has lead to these methods being widely and successfully used in data mining including financial applications. For instance, adding new data to the k-nearest neighbor method increases the training time and forecasting time reasonably. Independence of an expert when using IBL is relatively high in comparison with neuro-fuzzy and some other methods. In IBL, selection of measure of closeness and data normalization can be challenging. Using Euclidean distance has the obvious advantage of computational simplicity, but it may not be adequate to the problem. This distance is not invariant to the attribute scale transformation. Tuning normalization and selecting a distance measure is an art and an expert is integral to and perhaps the most important part of this process. Embeddability of IBL is quite good -- software and open codes are widely available and runtime is not prohibitive for many real tasks.

Most of the other features of IBL are average in comparison with alternative methods (see Table 1.7). There is one feature of the k-nearest neighbors method that makes it unique.

This is the possibility for an expert to study the behavior of a **real case** with similar attributes and motivate his/her intuition and informal knowledge for the final forecasts. Below some features of IBL based on [Dhar, Stein, 1977] are presented.

Explainability	Medium
Response speed	Medium-High
Tolerance for noise in data	Medium
Tolerance for sparse data	Medium
Ease of use logical relations	Low
Ease of use of numerical data	High

These features are needed to compare IBL with probabilistic ILP. Excepting the feature - “ease of use of numerical data”, the probabilistic ILP methods have advantages over IBL. To use numerical data in a probabilistic ILP, the data should be transformed into relational form (see Chapter 4).

2.4. Neural Networks

2.4.1. Introduction

Neural networks are widely presented in many available publications, therefore in this section we present only a short overview of neural networks based on terms and notation from [Russel, Norvig, 1995] and [Mitchell, 1997]. Section 2.5 is devoted to a new approach for testing neural networks and other data mining methods. Following that, section 2.6 discusses financial applications of neural networks.

A **neural network** can be viewed as consisting of four components:

$$\langle U, L^w, In, g \rangle,$$

where U is a set of **units (nodes, neurons)**, L^w is a set of **weighted links** between units, In is a set of **input functions** and g is an **activation function**. These components allow the network to propagate from input into output. Below these components are described more specifically.

The set of units U consists of three parts:

$$U = \langle U_{in}, U_h, U_{out} \rangle,$$

where

U_{in} , is a set of **input units** called the **input layer**,

U_h , is a set internal (**hidden**) **units** called the **hidden layers** and

U_{out} is a set of **output units**, called the **output layer**.

The set of weighted links, L^w , consists of two components:

$$L^w = \langle L, W \rangle,$$

where L is a set of links between nodes and W is a set of numeric **weights** attached to the links. Each link is given as an ordered pair $\langle j, i \rangle$, where j and i are indexes of nodes U_j and U_i . This pair indicates a directed link from U_j to U_i . Similarly, W_{ji} is a number (weight) attached to the link from U_j and U_i .

The set of **linear input functions**, In , consists of a function in_i for each unit U_i :

$$in_i(a_i) = \sum_j W_{ji} a_j$$

where a_j is the output of the node U_j , which serves as input for node U_i and a_i is the set of all such inputs to the node U_i . Each output a_i is called an **activation level**. This output from U_j is weighted by the W_{ji} . The functions, in_i , are interpreted as the total weighted input for the unit U_i .

An activation level for input nodes, U_{in} , is taken from **external (environment) nodes**, which do not belong to the neural network. Usually, a set of these input values is called an **example (instance)**, e . Similarly, the output nodes, U_{out} , deliver values into some **external nodes**, which also do not belong to the neural network.

The non-linear **activation function**, g , converts the weighted input into the final value that serves as unit's **activation value**, a_i . The **step** and **sigmoid functions** are typical activation functions:

$$s_t(x) = \begin{cases} 1, & \text{if } x \geq t \\ 0, & \text{if } x < t \end{cases}, \quad \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}.$$

The parameter t in the step function, s_t , is a biologically motivated threshold. If a node is interpreted as a neuron t represents the minimum total weighted input needed to fire the neuron.

In these terms, a neural network can be viewed as a set of complete processing **nodes**, $C = \{C_i\}$, where each node C_i consists of three components:

$$C_i = \langle \text{input links, output links, current activation level} \rangle.$$

In this way, information processing is encapsulated in the nodes. Each node uses only local inputs from its neighbors and therefore each node is independent of the rest of the network. This processing node can compute and recompute an activation level many times.

Figure 2.1 presents an example of a neural network used for forecasting financial time series. This example is considered in detail in Section 2.6.

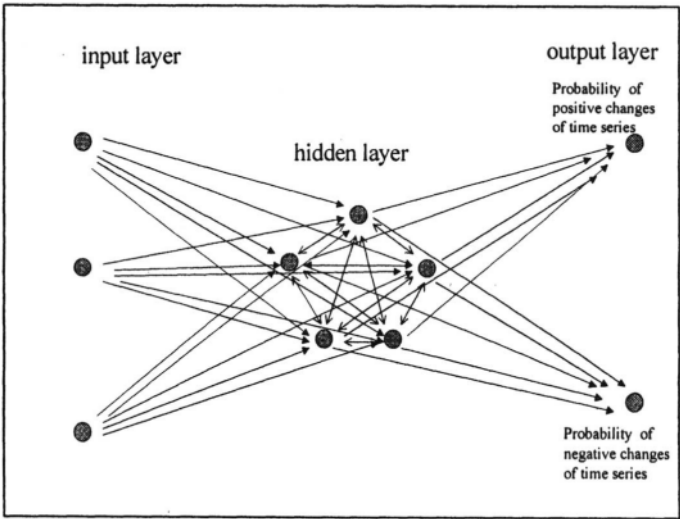


Figure 2. 1. Recurrent Neural Network for financial time series forecasting

2.4.2. Steps

Each processing unit in neural network performs a simple computation. Based on the input signals from its input links, the unit computes a new activation level for its output link. The major steps of the generic neural network learning method are presented in Table 2.2.

Table 2.2. Steps to build a neural network.

1. Define appropriate units (input, output, hidden) and their number. 2. Define connections of units to form a network. 3. Initialize the weights of the network. 4. Select the examples to train the network. 5. Define encoding the examples in terms of inputs and outputs of the network. 6. Adjust (train) the weights using a learning algorithm applied to a set of training examples to minimize forecasting errors.
--

Table 2.3 shows the mechanism for implementing the most important step (#6) of adjusting weights [Russel, Norvig, 1995]. This mechanism of updating a multilayer neural network is called the **backpropagation method**. The method has a tuning parameter α , called **the learning rate**.

Table 2.3. Update the weights in network (backpropagation method)

Step 6.1. Compute the complete output error $\text{Err}^e = T^e - O^e$ for all outputs O^e and errors Δ_j for each unit j in the output layer for an example e , where T^e is the observed output value from example e .

Step 6.2. Update the weights in the network using current weights W_{ji} , the tuning parameter (learning rate), α , the current activation value a_j , the error Err_i^e for example e and the derivative of the activation function $g'(in_i)$ for each unit j leading to the output layer:

$$W_{ji} + \alpha \times a_j \times \text{Err}_i^e \times g'(in_i) \rightarrow W_{ji}$$

Step 6.3. Compute the error at each node j

$$g'(in_j) \sum_i W_{ji} \Delta_i \rightarrow \Delta_j$$

Step 6.4. Update the weights leading into the layer, using values I_k from input layer, corresponding to training examples, e .

$$W_{kj} + \alpha \times I_k \times \Delta_j \rightarrow W_{kj}$$

2.4.3. Recurrent networks

A neural network can be designed with or without loops. Neural networks without loops are called **feedforward networks**. **Recurrent neural networks (RNN)** [Elman, 1991] are artificial neural networks with loops. They use outputs of network units at time t as the input to other units at time $t+1$. Specifically, a recurrent network can be constructed from a feedforward network by adding:

- a new unit b to the **hidden** layer, and
- a new **input** unit $c(t)$.

The value of $c(t)$ is defined as the value of unit b at **time $t-1$** , i.e., $c(t)=b(t-1)$. In this structure, b depends on both the original input $x(t)$ and on the added input $c(t)$. Therefore, it is possible for b to summarize information from earlier values of x that are **arbitrarily distant in time** [Mitchell, 1997].

Mitchell also pointed out that these recurrent networks have an important interpretation in **financial time series forecasting**. Let $T(t+1)$ be the stock price on date $t+1$. This stock price should be predicted using some economic indicator $x(t)$ on the date t and values of x on previous days arbitrarily distant in time. The RNN can be used for developing such forecasting model.

Alternative recurrent network structures can be designed to express more complex recurrent relations, e.g., adding new hidden layers between the input and unit b . Methods of training recurrent networks are described in [Mozer, 1995]. Despite the relative complexity of recurrent networks, they have unique advantages in representing **distant relations in time series**. In

section 2.6, we review some financial applications of recurrent neural networks.

2.4.4. Dynamically modifying network structure

In sections 2.1-2.3, we assumed a **fixed static network structure**, i.e., fixed numbers and types of network units and interconnections. Selection of the structure of a neural network is a most informal, **expert-dependent task**, but a neural network's performance, generality, accuracy, and training efficiency depend heavily on this structure.

The **dynamically modifying network approach** tries to tune a network structure in two opposite ways [Mitchell, 1997]:

1. begin with a network containing no hidden units (**perceptron**), then grow the network by adding hidden units until the training error is reduced to some acceptable level, or
2. begin with a complex network and prune it as certain connections are found to be nonessential.

According to Mitchell [1997], in general, techniques for dynamically modifying network structure have met with mixed success. From our viewpoint, this mixed success can be "credited" partially to the search methods used in the space of "black box" neural networks. In this space, the majority of searched network structures can be irrelevant. And only after a network is found, can it be transformed into a meaningful "IF-THEN" rule form. We consider this matter in later chapters.

2.5. Neural networks and hybrid systems in finance

Referring to the wide use of neural networks Rao and Rao [1993] say: "The vast interest in neural networks during the recent years results from the generalization accepted by (instance) example-based learning systems. It also results from the capability of the networks to form an arbitrarily close approximation to any continuous non-linear mapping."

Indeed, neural networks are widely used in finance. Many reviews and books are available, including [Abu-Mostafa, Moody, Weigend, 1996; Trippi, Turban, 1996; Azoff, 1994; Freedman, et al., 1995; Van Eyden, 1996; Jurik, 1993; Caldwell, 1994ab; Wong, 1994; Deny, 1994; Bandy, 1994; Obradovic, 1997; Pan et al., 1997].

These publications cover basic neural network examples, backpropagation, and data preprocessing as well as more advanced issues. These issues include neural network and fuzzy logic hybrid systems (see chapter 7) and a variety of specific applications: neural network-based financial trading sys-

tems and hybrid neural-fuzzy systems for financial modeling and forecasting. Use of backpropagation neural networks in finance is exemplified in the following study [Rao, Rao, 1993]. The neural network was developed using 14 attributes listed in table 2.4. The network was designed to predict the change in the closing price of SP500 from last week to this week. According to [Rao, Rao, 1993], these 14 delayed attributes used were able to produce 0.6% prediction error on the test data (20 weeks).

Table 2.4. Input delayed attributes for SP500 forecasting

SP500 High, Low
NYSE Advancing/Declining issues
NASDAQ Advancing/Declining issues
NYSE New Highs/New Lows
NASDAQ new Highs/New Lows
NYSE Total Volume
NYSE Advancing/Declining issues volume
NASDAQ Total Volume
NASDAQ Advancing/Declining issues volume
Three-Month Treasury Bill Yield
30-Year Treasury Bond Yield
Gold
SP500 Closing price

This impressive prediction accuracy of 99.4%, raises a question: how good is this **prediction error** for real trading? For example, in a buy/hold/sell trading strategy, we are much more interested in the correct forecast of stock direction (up/down) rather than the error itself. Graphical data from [Rao, Rao, 1993, p.328] shows that in three weeks out of the 20 test weeks the expected change direction was opposite to the actual direction. Therefore, the actual accuracy of the forecast for a buy/hold/sell strategy is 85%, which is really “quite good”, but not as impressive as 99.4%.

Preprocessing is widely used in general and financial neural network applications. A common way of preprocessing is the use of sigmoid and other transformations making values less than 1 [Rao & Rao, 1993]. The purpose is to speed up neural network training. Care needs to be exercised, however, since preprocessing may corrupt a hidden law, that should be discovered and/or used in later analysis and stock forecasting. For example, **independent sigmoid preprocessing** of three stock characteristics, volume (V), number of shares (N) and price per share (P), can violate the property: **$V=N \cdot P$** . This means that having transformed the data, the neural network may not be able to discover a simple property such as:

IF price of stock A is larger than price of stock B and

the number of shares of stock A is larger than number of shares of stock B

THEN volume of stock A is larger than volume of stock B

Relational data mining methods discussed in chapters 4 and 5 do not violate these kinds of properties. Also in chapter 5, we present another neural network constructed for SP500.

General properties of neural networks in comparison with requirements of stock price forecast are shown in tables 1.5 and 1.6 in chapter 1. These tables indicate three inadequacies of neural networks for stock price forecasting related to (1) explainability, (2) usage of logical relations, and (3) tolerance for sparse data. On the other hand, neural networks provide several advantages like high-speed response, tolerance for complexity, relative independence from an expert, flexibility, and compactness.

2.6. Recurrent neural networks in finance

Recurrent Neural Networks (RNN) have been used in several financial applications [Laurence et al, 1996, Giles et al, 1997; Saad, Prokhorov, Wunsch, 1998]. In particular, RNN were developed for the prediction of **daily foreign exchange rates** in combination with other techniques. In [Laurence et al, 1996, Giles et al, 1997], the authors report significant predictability in comprehensive experiments covering five different foreign exchange rates. The data consists of 3645 data points for each exchange rate covering the period from 1973 to 1987. They include daily closing bids for five currencies: (German Mark (DM), Japanese Yen, Swiss Franc, British Pound, and Canadian Dollar) with respect to the US Dollar.

The use of a recurrent neural network is important for two reasons. First, the model addresses the **temporal relationships** within a time series by maintaining an internal state. Secondly, **interpretable** rules (understandable by humans) can be extracted from the trained recurrent network. Specifically the used network consists of:

- Three input neurons. The first input node (neuron) is used to enter a condensed representation of the time series data $\mathbf{x(t)}, \mathbf{x(t-1)}, \mathbf{x(t-2)}, \dots, \mathbf{x(t-k)}$ for k time intervals. Delayed inputs are used for the other two input neurons, aiding in the training process.
- One hidden layer with five fully connected neurons.
- Two output neurons. The first output is trained to predict the probability of a positive change (“up learning”), and the second output is trained to predict the probability of a negative change (“down learning”).

This network is presented in figure 2.1 in section 2.4. The condensed representation, called an index, is used to keep the neural network smaller.

Giles et al. [1997] use the **self-organizing map**, or **SOM** technique [Kohonen, 1995] to get the index. This is an **unsupervised learning** process, which learns the distribution of a set of patterns without any class information. The SOM condenses each instance $\mathbf{x}$ with k components, $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_k$, and presents it as an instance $\mathbf{y}$ with s components, $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \dots, \mathbf{y}_s$, where s is significantly smaller than k . In the process of transformation SOM tries to keep distances between instances in the condensed space Y similar to the distances in the original space X . The networks were trained with backpropagation through time for 500 updates.

Rule Extraction from Recurrent Neural Network. Table 2.5 presents the steps of an algorithm for extracting rules from a recurrent neural network and table 2.6 presents the results of this algorithms -- extracted forecasting rules based on [Laurence at al, 1996, Giles et all, 1997]. This is a set of meaningful symbolic rules extracted from a time series with a significant noise level. Understanding the operation of a neural network can be gained by the extraction of rules.

The steps shown in table 2.5 produce the sets of rules presented in table 2.6. Figure 2.2 shows the third set of rules as graphical structure. This structure is adapted from [Laurence et al, 1996, Giles et al, 1997]. Mathematically each extracted structure is defined by the set ordered triples {state, input, next state}. These triples are called a **Markov chain** or **discrete Markov process** ({state, input, next state}), which is equivalent to **deterministic finite state automata (DFA)**.

Table 2.5. Steps of the algorithm for extracting rules from trained network

Step 1. Cluster the activation values of the recurrent state neurons [Kohonen, 1995].
Step 2. Assign states to the clusters.
Step 3. Insert transitions between the clusters on the relevant input symbols.

Table 2.6. Rules extracted from Recurrent Neural Network.

Set of rules	Extracted forecasting rules
1	Rule 1. IF the last change in the series was negative THEN the next change will be positive. Rule 2. IF the last change in the series was positive THEN the next change will be Negative.
2	Rule 1. IF the last change in the series was Negative THEN the next change will be Positive Rule 2. IF the last change in the series was Positive THEN the next change will be Positive
3	Rule 1. IF the last change in the series was Positive THEN the next change will be Positive. Rule 2. IF the last change in the series was Negative and previous change was not Positive THEN the next change will be Positive.

Figure 2.2 uses notation from [Laurence et al, 1996, Giles et al, 1997]. Transitions marked with a solid line correspond to predicting positive changes and transitions marked with a dotted line correspond to predicting negative changes. State 1 is the starting state of the DFA.

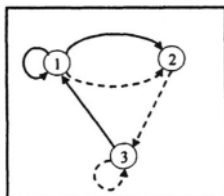


Figure 2.2. Sample deterministic finite state automata (DFA) extracted from trained financial prediction networks

The recurrent neural network combined with extracting deterministic finite state automata and discrete Markov process form a **hybrid approach in data mining**. As we have seen, this approach addresses the **fundamental problem of predicting noisy time series**.

2.7. Modular networks and genetic algorithms

2.7.1. Mixture of neural networks

A neural network is applicable to any data in the domain. In other words, its output can be computed for any input x from that domain. This property leads to both a positive and a negative consequence. On the plus side, the network covers a wide range of data. While alternatively, the design of a single network covering a wide variety of regularities requires a significant amount of training data in order to reach an appropriate accuracy for the whole of the domain. For non-stationary financial data, this is often problematic. Therefore, some hybrid methodologies were offered to **mix smaller networks into financial forecasting** [Wai Man Leung, et al., 1997]. Although these networks may have different architectures, the general design of such a hybrid system is presented on Figure 2.3.

Wai Man Leung, et al. [1997] offer the following training mechanism:

- a) The individual networks are trained before training their combination through gating network.
- b) The gating network is trained with the same set of training data. For each input test data, each network gives an output, which becomes the input to

the gating network. The final combined result is the output of gating network.

The **gated network** is trained to balance contributions of individual networks. We present a mixture of different data mining methods in more detail in chapter 3 (section 3.5.4).

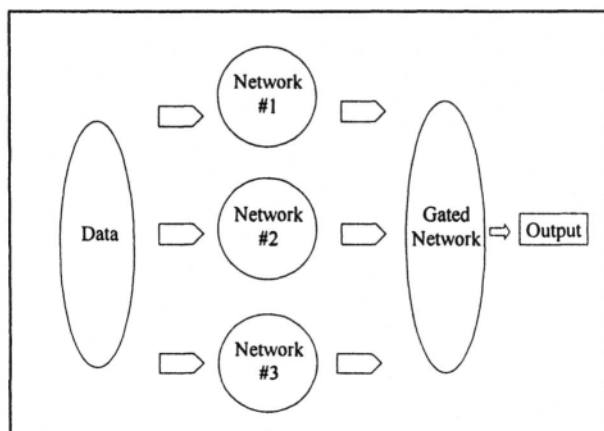


Figure 2.3. Mixture of neural networks

2.7.2. Genetic algorithms for modular neural networks

As we have already discussed, financial time series have specific drawbacks, like poor signal-to-noise ratios, non-Gaussian noise distribution, and **limited training data**. Traditional backpropagation neural networks can address these problems using a mixture of smaller neural networks of different architectures as described in the previous section. However, backpropagation has other drawbacks:

- it does not work if the error functions Δ are not smooth (**differentiable**), and
- it can become trapped in a **local minimum of the error function** and therefore it can miss the best neural network.

Feedforward Modular Neural Networks [Oliker, 1997] were designed to meet these challenges for **financial applications**. Similar to the previously discussed mixture of neural networks, this model consists of modular networks. Every one of the networks is trained by using a different set of features.

The major difference from backpropagation neural networks is in applying **genetic algorithms** [Holland, 1975] to adjust the weights. Genetic algorithms do not require smooth error functions for finding the best neural net-

work as required for the backpropagation method. In table 2.7, these methods are compared for a stock market forecast problem. In [Oliker, 1997], the distributed genetic algorithm is used for a **local search** of both the **connectivity and weights** of each unit in the network. Therefore, the method eliminates the need for a **predetermined network structure**.

Table 2.7. Comparison of methods and problem requirements

Parameters of methods and requirements of the problem	Feedforward modular neural networks	Backpropagation modular neural network	Requirements of stock market forecast problem
Mathematical technique	Distributed genetic algorithm	Backpropagation	
Ability to design smaller networks with a small number of hidden units	Yes	Yes	Desirable property
Ability to work well on time series with significant noise (poor signal-to-noise ratios)	Yes	Yes	Yes
Ability to perform well on time series with non-Gaussian noise distribution	Yes	Yes	Yes
Ability to perform well on Data with non-stationary regularities	Yes	Yes	Yes
Ability to perform well on Data with non-linear regularities	Yes	Yes	Yes
Ability to work in multimodal and non differentiable space	Yes	No	Yes
Free of the need to calculate derivatives of the error function	Yes	No	Desirable property
Free of the need to restrict artificially the number of parameters (layers) of the model	Yes, to some extend	No	Desirable property
Free of the need for a predetermined network structure	Yes, to some extend	No	Desirable property

Genetic algorithms [Holland, 1975] are an evolutionary training approach, attractive because of their ability to handle the **global search problem** in a set of alternative solutions. The traditional sequential search approach (hill-climbing) assumes that the next alternative to be examined is selected using a single preceding alternative. Fundamental advantages of genetic algorithms are based on use of parallel processing and evolutionary

adaptation. Specifically in genetic algorithms, several new alternatives are generated using already examined alternatives. An adaptive process of generating new alternatives is inspired by genetics. Each of the alternatives is viewed as a string of search parameters, called a “**genotype**”. The set of alternatives is considered as a **population**. The initial genotypes of the population are generated randomly. Then objective function values are computed for all initial genotypes. The next set of examples (“next generation of genotypes”) is generated using these values and **transitional operators**. These operators are also inspired by genetics and allow us to combine components of strings similar to genetic processes like **reproduction, crossover and mutation**. See [Holland, 1975; Olikier, 1997] and others for further explanations.

In [Olikier, 1997] each new generation is represented by its set of weights and connectivities within the neural network. The goal is find the network with the best set of weights (string, “genotype”). Each generation produces the network’s current error values and the next generation. Error values are tested against the required network’s accuracy and/or other criteria.

2.8. Testing results and complete round robin method

2.8.1. Introduction

In sections 2.4 through 2.7, we presented an overview of the neural networks and combinations of the neural networks and their financial applications. In this section, we will present an in-depth look at testing results produced by neural networks and other learning methods. In particular, we describe a novel extension of the round robin testing method. The novelty includes a mathematical mechanism based on the theory of monotone Boolean functions. The software implementation itself features important characteristics, which will be discussed below. Finally, we present computational experiments with SP500 using this tool.

2.8.2. Approach and method

The reliability of data mining methods depends on testing discovered patterns in the data before they are used for their intended purpose. One common approach used to test learned neural networks and learned regularities is to **divide** the data set into two parts. For instance, approximately 30% of the data can be chosen at random. This subset becomes the testing

data used to validate the patterns discovered by the data mining method after processing the remaining data.

This process is repeated several times and if results are similar to each other than a discovered regularity can be called reliable for data D. Three major methods for selecting subsets of training data are known as:

1. **Random** selection of subsets,
2. Selection of **disjoint** subsets,
3. Selection of subsets according the **probability distribution**.

These methods are sometimes called, respectively, bootstrap aggregation (bagging), cross-validated committees, and boosting [Dietterich, 1997].

The random selection of subsets, or **bagging** method, uses a training set that consists of a sample of m training examples drawn **randomly** with replacement from the original training set of N items.

The second method divides the training sets into subsets. For example, the training set can be randomly divided into 10 disjoint subsets. Then 10 overlapping training sets can be constructed by deleting a portion of these subsets. The same procedure is employed to construct training sets in 10-fold cross-validation [Dietterich, 1997]. Ensembles constructed in this way are sometimes called **cross-validated committees**.

The third method, computes the probability distribution $p(x)$ over the training data and generates subsamples of size k according to this distribution. Moreover, this distribution is systematically adjusted for paying more attention to cases which failed to learn when using previous subsamples. This method is presented in more detail in chapter 3 (section 3.1.4) and in [Dietterich, 1997].

Problems of subsampling. Different subsamples of a sample, Tr , can be governed by different regularities. Rejecting and accepting regularities heavily depends on the accidental splitting of Tr . This is common for non-stationary financial time series, e.g., bear and bull market trends for different time intervals.

Let subsets A and B be chosen from Tr as testing data and their complements, A' and B' , in Tr be chosen as training data sets:

$$Tr = A' \cup A, Tr = B' \cup B.$$

Further suppose that subsets A and B satisfy two different regularities, and that A and B do not intersect but occupy complimentary parts of Tr as shown in figure 2.4, covering 100% of the sample Tr .

Here regularity #1 discovered on A is useless for A' , which is actually B and thus governed by regularity #2. Similarly, regularity #2 discovered on B can not be confirmed on B' which is governed by regularity #1. In this extreme 50:50 case, the splitting test approach reasonably rejected both regularities

as universal for Tr. Now assume that 70% of Tr is governed by regularity #2 and only 30% by regularity #1. If accidentally, test set A consists of all cases governed by regularity #1, then regularity #2 found on A' will be rejected although it is true for 70% of the sample Tr.

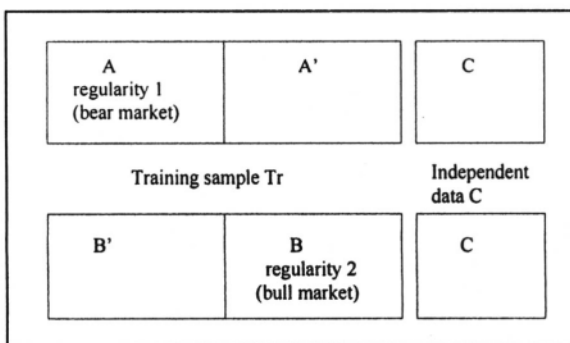


Figure 2.4. Training subsets for non-stationary data

These examples show that test results can be sensitive to a particular splitting of Tr. Therefore, such tests can reflect rather an arbitrary **splitting** instead of the real strength of regularities on data.

A more comprehensive approach is the **Round Robin**. It is designed to eliminate arbitrary splitting by examining several groups of subsets of Tr. If these groups do not cover all possible subsets then the round robin approach faces the problem of selecting independent subsets and determining their sizes.

The **Complete Round Robin** method examines **all groups** of subsets of Tr. The obvious drawback with the complete round robin is that there are 2^n possible subsets, where n is the number of **groups of objects** in the data set. Learning 2^n neural networks is a computational challenge.

Below we present an original implementation of the complete round robin method and techniques to speed up required computations [Kovalerchuk et al, 1996] along with experimental testing of 1024 neural networks constructed using SP500 data. This method uses the concepts of **monotonicity** and **multithreaded parallel processing** for Windows NT. It is applicable to both attribute-based and relational data mining methods.

The method was implemented for backpropagation neural networks with participation a group of Central Washington University computer science students. C. Todd, D. Henderson and J. Summet are major contributors to the project.

Let M be a data mining method and D be a set of N objects, represented by m attributes. $D = \{d_i\}, i=1, \dots, N, d_i = (d_{i1}, d_{i2}, \dots, d_{im})$. Method M is applied

to data D for knowledge discovery. We assume that data are grouped, for example in a stock time series, the first 250 data objects (days) belong to 1980, the next 250 objects (days) belong to 1981, and so on.

To simplify the example, assume we have ten groups (years), $n=10$. Similarly, half years, quarters and other time intervals can be used. There are 2^{10} possible subsets of the data groups. Any of these subsets can be used as training data.

If the data do not represent a time series, then all their complements without constraint can be used as testing data. For instance, by selecting the odd groups #1, #3, #5, #7 and #9 for training, permits use of the even groups #2, #4, #6, #8 and #10 for testing.

Alternatively, when the data represents a time series, it is reasonable to assume that test data should represent a later time than the training groups. For instance, training data can be the groups #1 to #5 and testing data can be the groups #6 to #10.

For our work, we have taken a third path by using the completely independent, later data set C for testing. This allows us to use any of the 2^{10} subsets of data for training.

The hypothesis of monotonicity is used below as a major assumption with the following notation. Let D_1 and D_2 be training data sets and let $Perform_1$ and $Perform_2$ be the performance indicators of learned models using D_1 and D_2 , respectively. We will consider a simplified case of a binary $Perform$ index, where 1 stands for appropriate performance and 0 stands for inappropriate performance. The **hypothesis of monotonicity** (HM) says that:

$$\text{IF } D_1 \supseteq D_2 \text{ THEN } Perform_1 \geq Perform_2. \quad (1)$$

This hypothesis means that IF data set D_1 covers data set D_2 , THEN performance of method M on D_1 should be better or equal to performance of M on D_2 . The hypothesis assumes that extra data bring more useful information than noise for knowledge discovery. Obviously, the hypothesis is not always true. From 1024 subsets of ten years used to generate the neural network, the algorithm found 683 subsets such that $D_i \supseteq D_j$. We expected that the significant number of them would satisfy the **property of monotonicity**

$$P(M, D_i) \geq P(M, D_j)$$

Surprisingly, in this experiment **monotonicity** was observed for **all of 683 combinations** of years. This is strong evidence for the use of monotonicity along with the complete round robin method. In fact, the incomplete round robin method assumes some kind of independence of the training subsets

used. Discovered monotonicity shows that independence at least should be tested.

Next, the concept of monotonicity is defined formally to be able to apply the theory of monotone Boolean functions.

The error Er for data set $D=\{\mathbf{d}_i\}$, $i=1,\dots,N$, is the normalized error of all its components $\mathbf{d}_i$:

$$Er = \frac{\sqrt{\sum_{i=1}^N (T(\mathbf{d}_i) - J(\mathbf{d}_i))^2}}{\sum_{i=1}^N T(\mathbf{d}_i)}$$

Here $T(\mathbf{d}_i)$ is the actual target value for $\mathbf{d}_i$ and $J(\mathbf{d}_i)$ is the target value forecast delivered by discovered model J , i.e., the trained neural network in our case.

Performance is measured by the error tolerance (threshold) Q_0 of error Er :

$$\text{Perform} = \begin{cases} 1, & \text{if } Er \leq Q_0 \\ 0, & \text{if } Er > Q_0 \end{cases}$$

Next, we introduce the hypothesis of monotonicity in terms of binary vectors and the parameter Perform in order to be able to use methods from the theory of monotone Boolean functions. Combinations of years are coded as binary vectors $\mathbf{v}_i=(v_{i1},v_{i2}\dots v_{i10})$ with 10 components from (0000000000) to (1111111111) with total $2^n=1024$ data subsets. In these binary terms, the **hypothesis of monotonicity** can be rewritten as

$$\text{IF } \mathbf{v}_i \succeq \mathbf{v}_j \text{ THEN Perform}_i \geq \text{Perform}_j \quad (2)$$

Here relation $\mathbf{v}_i \succeq \mathbf{v}_j$ (“no greater than”) for binary vectors is defined by the ordinary numeric order relation “ $\succeq$ ” for the components of these vectors:

$$\mathbf{v}_i \succeq \mathbf{v}_j \Leftrightarrow v_{ik} \geq v_{jk} \text{ for all } k=1,\dots,10.$$

Note that not every $\mathbf{v}_i$ and $\mathbf{v}_j$ are comparable with each other by the “ $\succeq$ ” relation. More formally, we will present Perform as a quality indicator Q :

$$Q(M, D, Q_0) = 1 \Leftrightarrow \text{Perform}=1, \quad (3)$$

where Q_0 is some performance limit. In this way, we rewrite (2)

$$\text{IF } \mathbf{v}_i \succeq \mathbf{v}_j \text{ THEN } Q(M, D_i, Q_0) \geq Q(M, D_j, Q_0) \quad (4)$$

and obtain $Q(M, D, Q_0)$ as a **monotone Boolean function** of D .

The theory of monotone Boolean functions [Hansel, 1966; Kovalerchuk et al, 1996] is a well-developed theory having mechanisms to speed up computations. We exploit this theory to speed up the round robin method. Consider a method M and data sets $\mathbf{D}_1$ and $\mathbf{D}_2$ with $\mathbf{D}_2$ contained in $\mathbf{D}_1$. Informally, according to the hypothesis of monotonicity if it is found that the method M does not perform well on the data $\mathbf{D}_1$, then it will not perform well on the data $\mathbf{D}_2$ either. Under this assumption, we do not need to test method M on $\mathbf{D}_2$.

Our experiments with SP500 shows that by eliminating these redundant computations it is possible to run method M 250 times instead of the complete 1024 times. The number of computations depends on a sequence of testing data subsets $\mathbf{D}_i$. To optimize the sequence of testing, Hansel's lemma [Hansel, 1966, Kovalerchuk et al, 1996] from the theory of monotone Boolean functions is applied to so called Hansel's chains of binary vectors. The mathematical monotone Boolean function techniques are presented in section 10.

The general logic of software is the following. The set of Hansel chains is generated first and stored. In the exhaustive case we need (1) to generate 1024 subsets using a file preparation program and (2) compute backpropagation for all of them. Actually, we follow the sequence dictated by the Hansel chains. Therefore, for a given binary vector we produce the corresponding training data and compute backpropagation generating the Perform value. This value is used along with stored Hansel chains to decide which binary vector (i.e., subset of data) will be used next for learning neural networks. We consider next the implementation of this approach.

2.8.3. Multithreaded implementation

The computation process for the round robin method can be decomposed into relatively independent subprocesses. Each subprocess can be matched to learning an individual neural network or a group of the neural networks. The multithreaded application in C++ uses both these decompositions, where each subprocess is implemented as an individual thread. The program is designed in such way that it can run in parallel on several processors to further speed up computations. Screen fragments and diagram of the implementation for the backpropagation Neural Network are presented in figures 2.5, 2.6, and 2.8.

Threads and the user interface. The processes described at the end of the section 2.8.2 lend themselves to implementation by threads. The system creates a new thread for the first vector in each Hansel chain. Each thread accomplishes three tasks: training file preparation, backpropagation using

this file, and computing the value Perform. When the thread starts, it paints a small area within the Thread State Watcher on the form to provide visual feedback to the user. The Perform value is extended, if possible, to the other threads in the chain and the results are printed to a file.

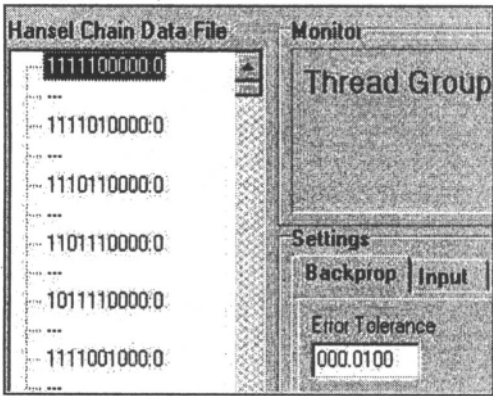


Figure 2.5. Fragment of Hansel chains used to construct threads

Threads are continually checking the critical section and an event state to see if it is permissible for them to run. When a thread is finally to run, a group of them will run at once. Since the starting of all the threads for each vector occurs at the same time, the vectors are split up over several computers.

The main program acts as a client, which accesses servers to run threads on different computers. On initialization, the client gathers user information and sends sets of vectors to various servers. Then the servers initialize only threads corresponding to the vectors they have been given. If a server finishes with all its assigned vectors it starts helping another server with the vectors it has not yet finished. Instead of having one computer run 250+ threads, six computers can run anywhere from 9 to 90 threads. This provides a speed up of approximately 6 times.

The client is made of two main components: the interface and the monitor. Once settings are made through the interface, the simulation is started. The monitor then controls the process by setting up, updating, and sending work to all the connected servers. This information is communicated in the form of groups of vectors. Once the servers receive their information, they spawn threads to work on the assigned group of vectors.

Once the thread has reached a result (Perform value), the information is communicated back to the monitor. The monitor then sends more work to the server if any remains and updates the display to reflect the work that has been done. Servers that are given work and fail to reply, have their work

flagged and once all other results are collected, the flagged work is sent to a different server. If there is still flagged work and no servers are responding, the client itself will execute the remaining work.

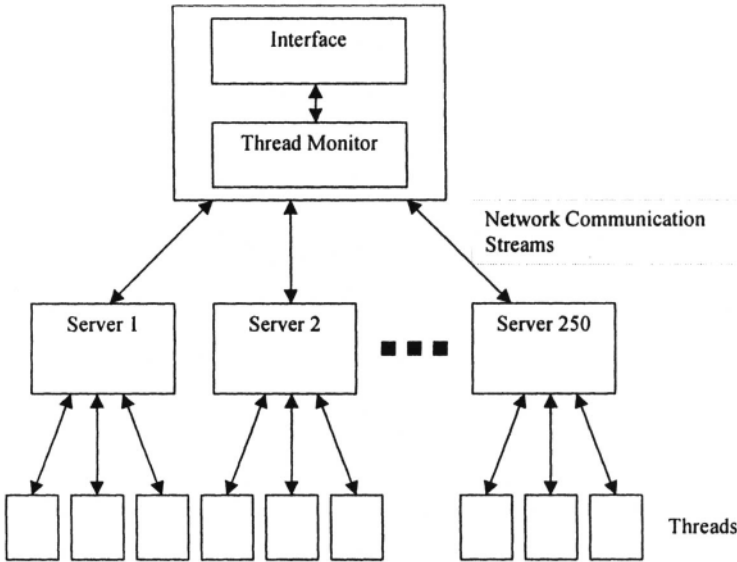


Figure 2.6. Implementation diagram

2.8.4. Experiments with SP500 and neural networks

In section 2.5, the backpropagation neural network of Rao and Rao [1993] which produced a 0.6% prediction error on the test data (50 weeks) with 200 weeks (about four years) of training data was presented. Now we consider the question - is this result reliable? In other words, will it be sustained for wider training and testing data? In an effort to answer these questions, we used all available data associated with SP500 from [Rao, Rao, 1993] as training and testing data:

- Training data -- all trading weeks from 1980 to 1989 and
- Independent testing data -- all trading weeks from 1990-1992.

We generated all 1024 subsets of the ten training years (1980-1989) and computed the corresponding backpropagation neural networks and their Perform values. Table 2.8 shows these results. A satisfactory performance is coded as 1 and non-satisfactory performance is coded as 0. Each of the resulting neural networks was tested on the same fixed independent testing data set (1990-1992) with a 3% error tolerance threshold ($Q_0=0.03$), which is higher than 0.6% used for the smaller data set in [Rao, Rao, 1993].

Table 2.8. Overall performance of 1024 Neural Networks

Performance		Number of neural net- works	% of neural networks
Training	Testing		
0	0	289	28.25
0	1	24	2.35
1	0	24	2.35
1	1	686	67.05

The majority of data subsets (67.05%, 686) satisfied the 3% error tolerance thus demonstrating sound performance of both training and testing data. Unsound performance was demonstrated by 48 subsets (4.68%). Of those 48 cases, 24 had Perform=1 for training and Perform=0 for testing while the other 24 cases had Perform=0 for training and Perform=1 for testing. Of course testing regularities found on any of those 48 subsets will **fail**, even if similar regularities were discovered on the 686 other subsets above. Using any of the remaining 289 subsets (28.25%) as training data would lead to the conclusion that there is insufficient data to discover regularities with a 3% error tolerance.

Therefore, a random choice of data for training from ten-year SP500 data will not produce regularity in 32.95% of cases, although regularities useful for forecasting **do exist**.

Table 2.9 presents a more specific analysis for 9 nested data subsets of the possible 1023 subsets (the trivial empty case (0000000000) is excluded from consideration). Suppose we begin the nested sequence with a single year (1986). Not surprisingly, it turns out that this single year's data is too small to train a neural network to a 3% error tolerance.

Table 2.9. Backpropagation neural networks performance for different data subsets.

Binary code for set of years	Training years										Performance	
	80	81	82	83	84	85	86	87	88	89	Training	Testing 90-92
0000001000							x				0	0
0000001001							x			x	0	0
0000001011							x		x	x	0	0
0000011011						x	x		x	x	1	1
0000111011					x	x	x		x	x	1	1
0001111011				x	x	x	x		x	x	1	1
0011111011			x	x	x	x	x		x	x	1	1
0111111011		x	x	x	x	x	x		x	x	1	1
1111111011	x	x	x	x	x	x	x		x	x	1	1

For the next two subsets, we added the years 1988 and 1989, respectively with the same negative result. However, when a fourth subset was added to

the data, 1985, the error moved below the 3% threshold. The additional subsets with five or more years of data also satisfy the error criteria. This, of course, is not surprising as we expect the monotonicity hypothesis will hold.

Similar computations made for all other 1023 combinations of years have shown that only a few combinations of four years satisfy the 3% error tolerance, but practically all five-year combinations satisfy the 3% threshold. In addition, all combinations of over five years satisfy this threshold.

Let us return to the results of Rao and Rao [1993] about the 0.6% error for 200 weeks (about four years) from the 10-year training data. In general, we see that four-year training data sets produce marginally reliable forecasts. For instance, the four years (1980, 1981, 1986, and 1987) corresponding to the binary vector (1100001100) do not satisfy the 3% error tolerance when used as training data. A neural network trained on them failed when it was tested on 1990-1992 years.

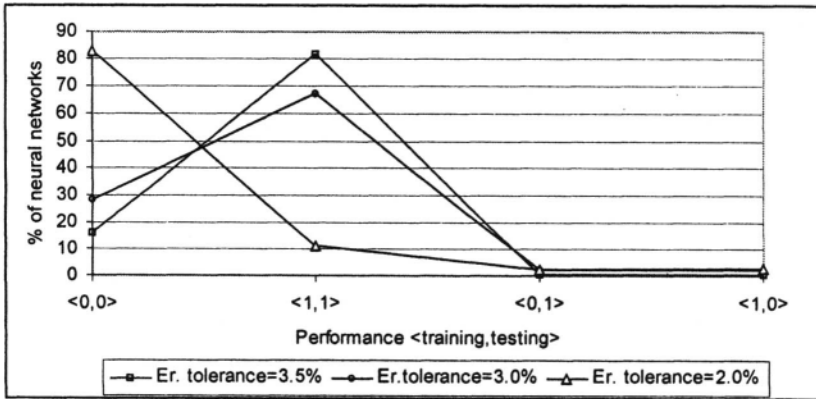


Figure 2.7. Performance of neural networks for error tolerance 3.5%, 3.0% and 2.0%.

Figure 2.7 and table 2.10 present further analyses of the performance of the same 1023 data subsets, but with three different levels of error tolerance, Q_0 : 3.5%, 3.0% and 2.0%. The number of neural networks with sound performance goes down from 81.82% to 11.24% by moving error tolerance from 3.5% to 2%. Therefore, neural networks with 3.5% error tolerance are much more reliable than networks with 2.0 % error tolerance. **A random choice** of training data for 2% error tolerance will more often reject the training data as insufficient. However, this standard approach does not even allow us to know how unreliable the result is without running the complete 1024 subsets in complete round robin method.

Running a complete round robin is a computational challenge. Below we present results of computational experiments showing that monotonicity and multithreading significantly decrease the computation time.

Table 2.10. Overall performance of 1024 neural networks with different error tolerance

Error toler- ance	Performance		Number of neural networks	% of neural networks
	Training	Testing		
3.5%	0	0	167	16.32
	0	1	3	0.293
	1	0	6	0.586
	1	1	837	81.81
3.0%	0	0	289	28.25
	0	1	24	2.35
	1	0	24	2.35
	1	1	686	67.05
2.0%	0	0	845	82.60
	0	1	22	2.150
	1	0	31	3.030
	1	1	115	11.24

Use of monotonicity with 1023 threads decreased average runtime about 3.5 times from 15-20 minutes to 4-6 minutes in order to train 1023 Neural Networks in the case of mixed 1s and 0s for output. Different error tolerance values can change the output and runtime. For instance, we may get extreme cases with all 1's or all 0's as outputs. Table 2.11 shows the result obtained for the extreme case where all 1's were produced as output.

Table 2.11. Runtime for different error tolerance settings

Method	Average time for 1023 Neural Networks 1 Processor, no threads	Average time for 1023 Neural Networks 1 Processor, 1023 threads
Round-Robin with monotonicity for mixed 1's and 0's as output	15-20 min.	6-4 min.
Round-Robin with monotonicity for all 1's as output	10 min.	3.5 min.

In addition, a significant amount of time is taken for file preparation to train 1023 Neural Networks. The largest share of time for file preparation (41.5%) is taken by files using five years in the data subset (table 2.12).

Table 2.12. Time for backpropagation and file preparation

Set of years	% of time in file preparation	% of time in Backpropagation
0000011111	41.5%	58.5%
0000000001	36.4%	63.6%
1111111111	17.8%	82.2%

Backpropagation neural networks also were ran on another set of SP500 data (training data -- daily SP500 data from 1985-1994, and testing data -- daily SP500 data from 1995-1998). Figure 2.8 presents a screen shot of the

performance of 256 backpropagation neural networks for these data. Error tolerance in this test was chosen to be 3%. More than one-thousand (1007) neural networks out of 1023 total satisfied this error tolerance on both training and testing data. The total runtime was 4.14 minutes using a single processor. In this experiment, monotonicity allowed us to run 256 subsets instead of 1023 subsets.

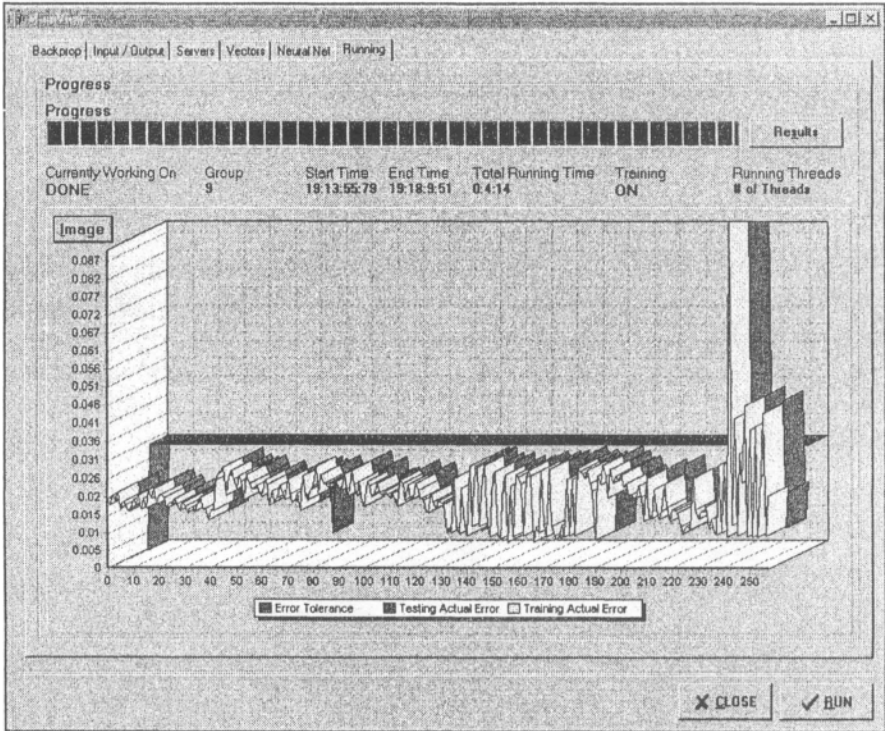


Figure 2.8. Performance of 256 Neural networks on SP500 data.

2.9. Expert mining

Data mining vs. “expert mining”. Data mining has a serious drawback: if data are scarce then data mining methods can not produce useful regularities as discussed in section 8. An expert can be a valuable source of regularities for situations where an explicit set of data either does not exist or is insufficient. The serious drawback of traditional knowledge-based (expert) systems in finance is the **slowness of the knowledge engineering** for changing markets (see table 1.4, chapter 1). This includes the extraction of trading rules and dynamic correction. Few trading rules work consistently well across different kinds of markets, e.g., trading rules, which work well

for a bull market, may perform miserably in a bear market [Loofbourrow and Loofbourrow, 1995].

In chapter 1, we mentioned that any expert can learn from already trained artificial “experts” and human experts. This learning is called “**expert mining**,” an umbrella term for extracting knowledge from “experts”. An example of expert mining is extracting understandable rules from a learned neural network, which serves as an artificial expert [Shavlik, 1994].

In this section, we present a method to help “mine” regularities from an expert and to speed up this process significantly. The method is based on the same mathematical tools of monotone Boolean functions we used in section 8 for testing learned models.

The essence of the property of **monotonicity** for this application is that:

If an expert believes that property T is true for example x and attributes of example y are stronger than attributes of x , then property T is also true for example y .

Here the phrase *attributes are stronger* refers to the property that values of each attributes of x are larger than the corresponding values of y . Informally, larger is interpreted as “better” or “stronger.” Sometimes to be consistent with this idea, we need to transform the coding of attributes.

The mentioned problem, **slowness of learning** of traditional expert systems, means that we need to ask experts too many questions when extracting rules. That is, it takes too much time for real systems with a large amount of attributes. The idea of the approach is to represent the questioning procedure (interviewing) as a restoration of a monotone Boolean function interactively with an “oracle” (expert). In the experiment below, even for a small number of attributes (5), using the method based on monotone Boolean functions, we were able to restrict the number of questions to 60% of the number questions needed for complete search. The difference becomes more significant for larger numbers of attributes. Thus, full restoration of either one of the two functions f_1 and f_2 (considered below) with 11 arguments without any optimization of the interview process would have required up to 2^{11} or 2048 calls to an expert. However, according to the Hansel lemma [Hansel, 1963, Kovalerchuk et al, 1996], under the assumption of monotonicity, an optimal dialogue (i.e. a minimal number of questions) for restoring each of these functions would require at most 924 questions:

$$\binom{11}{5} + \binom{11}{6} = 2 \times 462 = 924,$$

This new value, 924 questions, is 2.36 times smaller than the previous upper limit of 2048 calls. However, this upper limit of 924 questions can be re-

duced even further. In particular, in one of the tasks by using monotonicity and the **hierarchy**, the maximum number of questions needed to restore the monotone Boolean functions was reduced first to 72 questions and then further reduced to 46 questions using the Hansel lemma.

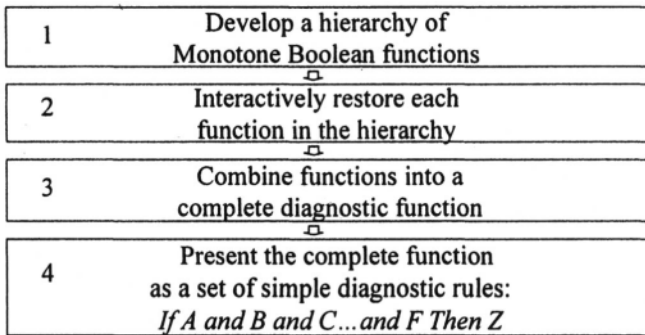


Figure 2.9. Major steps for extraction of expert diagnostic rules

Figure 2.9 presents the major steps of extraction of rules from an expert using this mathematical technique.

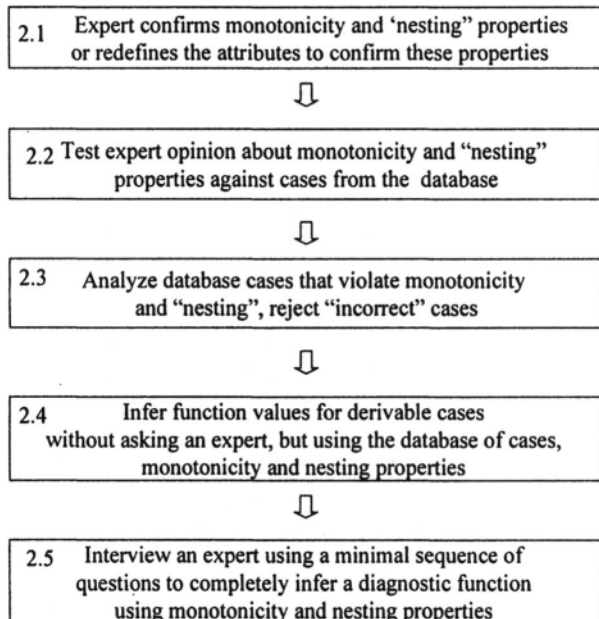


Figure 2.10. Iteratively restoring functions in hierarchy

The actual number of questions asked was about 40 questions for the two nested functions, i.e., about 20 questions per function. This number should be compared with the full search requiring 2^{11} or 2048 questions. Therefore, this procedure allowed us to ask about **100 times fewer questions** without relaxing the requirement of complete restoration of the functions. The formal definitions of concepts from the theory of monotone Boolean functions used to obtain these results are presented in section 2.10.

Figure 2.10 details the sequence of actions taken to accomplish step 2, i.e., restoring each of the monotone Boolean functions with a minimal sequence of questions to the expert. The last block (2.5) in figure 2.10 provides for interviewing an expert with a **minimal dynamic sequence of questions**. This sequence is based on the fundamental Hansel lemma [Hansel, 1966; Kovalerchuk et al., 1996] and the property of **monotonicity**. Table 2.13 shows the general idea of these steps. It represents a complete interactive session. A minimal dynamic sequence of questions means that we reach the minimum of the Shannon Function, i.e., the **minimum number of questions required to restore the most complex monotone Boolean function of n arguments**. This sequence is not a sequence written in advance. It depends on the previous answers of an expert; therefore each subsequent question is defined **dynamically** in order to minimize the number of total questions.

Columns 2, 3 and 4 in table 2.13 present values of the three functions f_1 , f_2 and ψ of five arguments chosen for this example. These functions represent regularities that should be discovered by interviewing an expert.

We assume here that each of these functions has its own target variable. Thus, the first question to the expert in column 2 is: "Does the sequence (01100) represent a case with the target attribute equal to 1 for f_1 ?" Columns 3 and 4 represent expert's answers for functions f_2 and ψ .

For instance, the binary vector (01100) could represent five binary stock attributes, like up (1) and down (0) for the last five years--1995, 1996, 1997, 1998, 1999. These attributes could also be months, weeks, minutes, or other stock features. Recall, in section 8 we considered binary vectors as representatives for **sets of objects**. In this section, binary vectors represent **sets of attributes**. The strength of the monotone Boolean function approach is that it is applicable for both tasks in the same way after entities are coded by binary vectors and monotonicity is interpreted.

In our last example, $(01100)=(x_1, x_2, x_3, x_4, x_5)$ and thus, $x_1=0$. If the answer is "yes" (1), then the next question will be about the target value for the case (01010). If the answer is "No" (0), then the next question will be about the target value for (11100). This sequence of questions is not accidental. As mentioned above, it is inferred from the Hansel lemma.

Table 2.13. Dynamic sequence of interviewing an expert

1	2	3	4	5	6	7	8
Vector	f_1	f_2	ψ	Monotone extension		Chain #	Case #
				1 $\rightarrow$ 1	0 $\rightarrow$ 0		
(01100)	1*	1*	1*	1.2;6.3;7.3	7.1;8.1	Chain 1	1.1
(11100)	1	1	1	6.4;7.4	5.1;3.1		1.2
(01010)	1*	0*	1*	2.2;6.3;8.3	6.1;8.1	Chain 2	2.1
(11010)	1	1*	1	6.4;8.4	3.1;6.1		2.2
(11000)	1*	1*	1*	3.2	8.1;9.1	Chain 3	3.1
(11001)	1	1	1	7.4;8.4	8.2;9.2		3.2
(10010)	1*	0*	1*	4.2;9.3	6.1;9.1	Chain 4	4.1
(10110)	1	1*	1	6.4;9.4	6.2;5.1		4.2
(10100)	1*	1*	1*	5.2	7.1;9.1	Chain 5	5.1
(10101)	1	1	1	7.4;9.4	7.2;9.2		5.2
(00010)	0*	0	0*	6.2;10.3	10.1	Chain 6	6.1
(00110)	1*	1*	0*	6.3;10.4	7.1		6.2
(01110)	1	1	1	6.4;10.5			6.3
(11110)	1	1	1	10.6			6.4
(00100)	1*	1*	0*	7.2;10.4	10.1	Chain 7	7.1
(00101)	1	1	0*	7.3;10.4	10.2		7.2
(01101)	1	1	1*	7.4;10.5	8.2;10.2		7.3
(11101)	1	1	1	5.6			7.4
(01000)	0*	0	1*	8.2	10.1	Chain 8	8.1
(01001)	1*	1*	1	8.3	10.2		8.2
(01011)	1	1	1	8.4	10.3		8.3
(11011)	1	1	1	10.6	9.3		8.4
(10000)	0*	0	1*	9.2	10.1	Chain 9	9.1
(10001)	1*	1*	1	9.3	10.2		9.2
(10011)	1	1	1	9.4	10.3		9.3
(10111)	1	1	1	10.6	10.4		9.4
(00000)	0	0	0	10.2		Chain 10	10.1
(00001)	1*	0*	0	10.3			10.2
(00011)	1	1*	0	10.4			10.3
(00111)	1	1	1	10.5			10.4
(01111)	1	1	1	10.6			10.5
(11111)	1	1	1				10.6
Total Calls	13	13	12				

Columns 5 and 6 list cases for extending values of functions without asking an expert by using the property of monotonicity. Column 5 is for extending values of functions from 1 to 1 and column 6 is for extending them from 0 to 0. In other words, if $f(x)=1$ then column 5 helps to find y such that $f(y)=1$. Similarly, column 6 works for $f(x)=0$ by helping to find y such that $f(y)=0$. Suppose for case #1.1 an expert gave the answer $f_1(01100)=0$, then this 0 value could be extended in column 2 for case #7.1 (00100) and case

#8.1 (01000). These cases are listed in column 6 in the row for case #1.1. There is no need to ask an expert about cases #7.1 and #8.1. Monotonicity is working for them. On the other hand, the negative answer $\mathbf{f}_1(01100)=0$ can not be extended for $\mathbf{f}_1(11100)$. An expert should be asked about $\mathbf{f}_1(11100)$ value. If the answer is negative, i.e., $\mathbf{f}_1(11100)=0$, then this value can be extended for cases #5.1 and #3.1. Similarly to case #1.1 these cases are listed in column 6 for case #1.2. The value of $\mathbf{f}_1$ for cases #5.1 and #3.1 will also be 0, because of monotonicity.

In other words, the values in column 2 for $\mathbf{f}_1$ are derived by **up-down sliding** in table 2.13 according to the following five steps:

Step 1.

Begin from the first case #1.1, here (01100).

Action: Ask the expert about the value of $\mathbf{f}_1(01100)$.

Result: Here the expert reported that $\mathbf{f}_1(01100)=1$.

Step 2.

Action: Write the value for case #1.1. under column 2.

Here a "1*" is recorded next to vector (01100). Recall that an asterisk denotes an answer directly provided by the expert. The case of having the true value corresponds to column 5. (If the reply was false (0), then we write "0" in column 2. The case of having a false value corresponds to column 6.)

Step 3.

Action: Based on the response of true or false by the expert in step 1, check column 5 or 6 respectively to extend the given value. Here we check column 5 due to the response true for case #1.1.

Result: Extend the response to the cases listed. Here cases #1.2, #6.3, and #7.3 are defined as 1. Therefore, in column 2 for cases #1.2, #6.3, and #7.3 the values of the function $\mathbf{f}_1$ must be (1). Note that now no asterisk is used because these are extended values.

Step 4. (Iterate until finished).

Action: Go to the next vector (called sliding down), here case #1.2. Check whether the value of $\mathbf{f}_1$ has already been fixed. If the value of $\mathbf{f}_1$ is not fixed (i.e., it is empty), repeat steps 1-3, above for this new vector. If $\mathbf{f}_1$ is not empty (i.e., it has been already fixed), then apply step 3 and slide down to the next vector. Here cases #6.4 and #7.4 are extended and then we move to case #2.1. Note, that if the value has not been fixed yet, then it will be denoted by $\mathbf{f}_1(\mathbf{x}) = \mathbf{e}$; for empty.

Result: Values of the function are extended. Here since $\mathbf{f}_1(11100) \neq \mathbf{e}$, the values of the function for the cases #6.4 and #7.4 are extended.

Here the total number of cases with an asterisk (*) in column 1 is equal to 13. For columns 3 and 4 the number of asterisks is 13 and 12, respectively. These numbers show that 13 questions are needed to restore each of $\mathbf{f}_1$ and $\mathbf{f}_2$

and 12 questions are needed to restore ψ as functions of five variables. As we have already mentioned, this is only 37.5% of 32 total possible questions and 60% of the potential maximum of 20 questions generated by Hansel lemma. Table 2.14 shows this result more specifically.

Table 2.14. Comparison of search results for five attributes.

Search methods	f_1, f_2	f_2	Decreasing coefficient	
			f_1, f_2	ψ
Non-optimised search (upper limit)	32	32	1	1
Optimal search (upper limit)	20	20	1.6	1.6
Optimal search (actual performance)	13	12	2.5	2.67

The next step is obtaining learned rules from table 2.13. In order to construct rules, one needs to concentrate on the information contained in columns 2, 3 and 4 of table 2.13. One needs to take the first vector marked with "1*" in each one of the chains and construct a conjunction of non-zero components. For instance, for the vector (01010) in chain 2, the corresponding conjunction is x_2x_4 . Similarly, from chain 6 we have taken the "1" components in the vector (00110) form the conjunction x_3x_4 .

Based on these conjunctions, column 4 in table 2.13, and the steps listed below, we obtained this logical expressions for $\psi(x_1, x_2, x_3, x_4, x_5)$:

$$\psi(x) = x_1x_2 \vee x_2x_3 \vee x_2x_4 \vee x_1x_3 \vee x_1x_4 \vee x_2x_3x_4 \vee x_2x_3x_5 \vee x_2 \vee x_1 \vee x_3x_4x_5.$$

1. Find all the **lower units** for all chains as elementary functions.
2. Exclude the redundant terms (conjunctions) from the end formula.

Let us explain the concept of lower unit with an example. In chain 6 in table 2.13 the case #6.2 is a maximal lower unit, because f_1 for this case is equal to 1 and the prior case #6.1 as an f_1 value equal to 0. Similarly, the case #6.1 will be referred to as an **upper zero**. The formula for $\psi(x)$ is simplified to

$$\psi(x) = x_2 \vee x_1 \vee x_3x_4x_5.$$

Similarly, the target functions $f_1(x)$ and $f_2(x)$ can be obtained from columns (2 and 3) in table 2.13 as follows:

$$f_1(x) = x_2x_3 \vee x_2x_4 \vee x_1x_2 \vee x_1x_4 \vee x_1x_3 \vee x_3x_4 \vee x_3 \vee x_2x_5 \vee x_1x_5 \vee x_5,$$

$$f_2(x) = x_2x_3 \vee x_1x_2x_4 \vee x_1x_2 \vee x_1x_3x_4 \vee x_1x_3 \vee x_3x_4 \vee x_3 \vee x_2x_5 \vee x_1x_5 \vee x_4x_5.$$

Hansel chains. This sequence of questions is not accidental. As mentioned above, it is inferred from the Hansel lemma to get a minimal number

of questions in the process of restoring a **complete rule**. Here by complete rule we mean restoring a function for all possible inputs.

Below we consider the general steps of the algorithm for chain construction. All 32 possible cases with five binary attributes ($\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5$) are presented in column 1 in table 2.13. They have been grouped according to the Hansel lemma. These groups are called Hansel chains. The sequence of chains begins from the shortest length chain #1 -- (01100) and (11100). This chain consists only of two ordered cases (vectors), (01100) < (11100) for five binary attributes. Then largest chain #10 consists of 6 ordered cases:

$$(00000) < (00001) < (00011) < (00111) < (01111) < (11111).$$

To construct chains presented in table 2.13 (with five dimensions like $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5$) a sequential process is used, which starts with a single attribute and builds to all five attributes. We use a standard mathematical notation, for example, all five-dimensional vectors are considered as points in 5-dimensional binary “cube”,

$$\mathbf{E}_5 = \{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\}.$$

At first all 1-dimensional chains (in $\mathbf{E}_1 = \{0,1\}$) are generated. Each step of chain generation consists of using current i -dimensional chains to generate $(i+1)$ dimensional chains. Generating of chains for the next dimension $(i+1)$ is four-step “**clone-grow-cut-add**” process. An i -dimensional chain is “cloned” by adding zero to all vectors in the chain. For example, the 1-dimensional chain:

$$(0) < (1)$$

clones to its two-dimensional copy:

$$(00) < (01).$$

Next we **grow** additional chains by changing added zero from cloning to 1. For example cloned chain 1 from above grows to chain 2:

$$\text{Chain 1: } (00) < (01)$$

$$\text{Chain 2: } (10) < (11).$$

Next we **cut** the head case, the largest vector (11), from chain 2 and **add** it as the head of chain 1 producing two Hansel 2-dimencional chains:

**New chain 1: (00) < (01) < (11) and
New chain 2: (10).**

This process continues through the fifth dimension for $\langle x_1, x_2, x_3, x_4, x_5 \rangle$. Table 2.13 presents result of this process. The chains are numbered from 1 to 10 in column 7 and each case number corresponds to its chain number, e.g., #1.2 means the second case in the first chain. Asterisks in columns 2, 3 and 4 mark answers obtained from an expert, e.g., 1* for vector (01100) in column 2 means that the expert answered “yes”. The remaining answers for **the same chain** in column 2 are automatically obtained using monotonicity. The value $f_1(01100)=1$ for case #1.1 is extended for cases #1.2, #6.3 and #7.3 in this way. Hansel chains are derived independently of the particular applied problem, they depend only on the number of attributes (five in this case). The formal definitions of concepts from theory of monotone Boolean functions, which were used to obtain these results and the results described in section 2.8, are presented in section 2.10 below.

2.10. Interactive Learning of Monotone Boolean Functions

2.10.1. Basic definitions and results

Let E_n be the set of all binary vectors of length n . Let x and y be two such vectors. Then, the vector $x = (x_1, x_2, x_3, \dots, x_n)$ **precedes** the vector $y = (y_1, y_2, y_3, \dots, y_n)$ (denoted as: $x \preceq y$) if and only if the following is true: $x_i \leq y_i$, for all $1 \leq i \leq n$.

A Boolean function $f(x)$ is **monotone** if for any vectors $x, y \in E_n$, the relation $f(x) \leq f(y)$ follows from the fact that $x \preceq y$. Let M_n be the set of all monotone Boolean functions defined on n variables.

A binary vector x of length n is said to be an **upper zero** of a function $f(x) \in M_n$, if $f(x) = 0$ and, for any vector y such that $y \succ x$, we have $f(y) = 1$. Also, the term **level** represents the number of units (i.e., the number of the “1” elements) in the vector x and is denoted by $U(x)$.

An upper zero x of a function f is said to be the **maximal upper zero** if $U(y) \leq U(x)$ for any upper zero y of the function f [Kovalerchuk, Lavkov, 1984]. The concepts of **lower unit** and **minimal lower unit** similarly are defined similarly. A binary vector x of length n is said to be a **lower unit** of a function $f(x) \in M_n$, if $f(x) = 1$ and, for any vector y from E_n such that $y \prec x$, we get $f(y) = 0$. A lower unit of a function f is said to be the **minimal lower unit** if $U(x) \leq U(y)$ for any lower unit y of the function f . The **number of monotone Boolean functions** of n variables, $\psi(n)$, is given by:

$$\psi(n) = 2^{\binom{n}{\lfloor n/2 \rfloor}} (1 + \varepsilon(n))$$

where $0 < \varepsilon(n) < c(\log n)/n$ and c is a constant (see [Alekseev, 1988; Kleitman, 1969]). Thus, the number of monotone Boolean functions grows exponentially with n . Let a monotone Boolean function f be defined by using a certain operator $\mathbf{A}_f$ (also called an **oracle**) which takes a vector $\mathbf{x} = (x_1, x_2, x_3, \dots, x_n)$ and returns the value of $f(\mathbf{x})$. Let $\mathbf{F} = \{\mathbf{F}\}$ be the set of all **algorithms** which can solve the above problem and let $\varphi(\mathbf{F}, f)$ be the number of accesses to the operator $\mathbf{A}_f$ required to generate $f(\mathbf{x})$ and completely restore a monotone function $f \in \mathbf{M}_n$.

Next, we introduce the **Shannon function** $\varphi(n)$ [Korobkov, 1965]:

$$\varphi(n) = \min_{\mathbf{F} \in \mathbf{F}} \max_{f \in \mathbf{M}_n} \varphi(\mathbf{F}, f) \quad (4)$$

Consider the problem of finding all the maximal upper zeros (lower units) of an arbitrary function $f \in \mathbf{M}_n$ with by accessing the operator $\mathbf{A}_f$. It is shown in [Hansel, 1966] that for this problem the following relation is true (known as **Hansel's lemma**):

$$\varphi(n) = \binom{n}{\lfloor n/2 \rfloor} + \binom{n}{\lfloor n/2 \rfloor + 1} \quad (5)$$

Here $\lfloor n/2 \rfloor$ is the closest integer number to $n/2$, which is no greater than $n/2$ (floor function). In terms of machine learning, the set of all maximal upper zeros represents the **border elements** of the negative patterns. Similarly, the set of all minimal lower units represents the border of positive patterns. In this way, a monotone Boolean function represents two **compact patterns**. Restoration algorithms for monotone Boolean functions which use Hansel's lemma are optimal in terms of the Shannon function. That is, they minimize the maximum time requirements of any possible restoration algorithm. To the best of our knowledge, Hansel's lemma has not been translated into English, although there are numerous references to it in the non-English literature (Hansel's results were published in French in Paris). This lemma is one of the final results of the long-term efforts in monotone Boolean functions started by Dedekind [1897].

2.10.2. Algorithm for restoring a monotone Boolean function

Next we present algorithm **RESTORE**, for the interactive restoration of a monotone Boolean function, and two procedures **GENERATE** and **EXPAND**, for manipulation of chains.

Algorithm "RESTORE" $f(x_1, x_2, \dots, x_n)$

Input. Dimension n of the binary space and access to an oracle A_f .

Output. A monotone Boolean function restored after a minimal number (according to formula (5)) of calls to the oracle A_f .

Method.

1. Construction of Hansel chains (see section 2.10.3 below).
2. Restoration of a monotone Boolean function starting from chains of minimal length and finishing with chains of maximal length.

This ensures that the number of calls to the oracle A_f is no more than the limit presented in formula (5).

Table 2.15. Procedure RESTORE

```

Set  $i=1$ ; {initialization}
DO WHILE (function  $f(x)$  is not entirely restored)
    Step 1: Use procedure GENERATE to generate element  $\alpha_i$ ; which is a binary vector.
    Step 2: Call oracle  $A_f$  to retrieve the value of  $f(\alpha_i)$ ;
    Step 3: Use procedure EXPAND to deduce the values of other
            Elements in Hansel chains (i.e., sequences of examples in  $E_n$ )
            by using the value of  $f(\alpha_i)$ , the structure of element  $\alpha_i$  and
            the monotonicity property.
    Step 4: Set  $i \leftarrow i+1$ ;
RETURN
  
```

Procedure "GENERATE": Generate i -th element α_i to be classified by the oracle A_f .

Input. The dimension n of the binary space.

Output. The next element to send for classification by the oracle A_f .

Method: Begin with the minimal Hansel chain and proceed to maximal Hansel chains.

Table 2.16. Procedure GENERATE

```

IF  $i=1$  THEN {where  $i$  is the index of the current element}
    Step 1.1: Retrieve all Hansel chains of minimal length;
    Step 1.2: Randomly choose the first chain  $C_l$  among the chains retrieved
              in step 1.1;
    Step 1.3: Set the first element  $\alpha_i$  as the minimal element of chain  $C_l$ ;
ELSE
    Set  $k=1$  {where  $k$  is the index number of a Hansel chain};
    DO WHILE (NOT all Hansel chains are tested)
        Step 2.1: Find the largest element  $\alpha_i$  of chain  $C_k$ , which still has no  $f(\alpha_i)$  value;
        Step 2.2: If step 2.1 did not return an element  $\alpha_i$ , then randomly select the
                  Next Hansel Chain  $C_{k+l}$  of the same length  $l$  as the one of the current
                  chain  $C_k$ ;
        Step 2.3: Find the least element  $\alpha_i$  from chain  $C_{k+l}$ , which still has no  $f(\alpha_i)$ 
                  value;
        Step 2.4: If Step 2.3 did not return an element  $\alpha_i$ , then randomly choose chain
                   $C_{k+l}$  of the next available length  $(l+1)$ ;
        Step 2.5: Set  $k \leftarrow k+1$ ;
  
```

Procedure "EXPAND": Assign values of $f(x)$ for $x < \alpha_i$ or $x > \alpha_i$ in chains of the given length l and in chains of the next length $l+2$. According to the Hansel lemma if for n the first chain has even length then all other chains for this n will be even. A similar result holds for odd lengths.

Input. The $f(\alpha_i)$ value.

Output. An extended set of elements with known $f(x)$ values.

Method: The method is based on monotone properties:

IF $x > \alpha_i$ and $f(\alpha_i)=1$, THEN $f(x)=1$; and

IF $x < \alpha_i$ and $f(\alpha_i)=0$, THEN $f(x)=0$.

Table 2.17. Procedure EXPAND

Step 1.	Obtain x such that $x < \alpha_i$ or $x > \alpha_i$ and x is in a chain of the lengths l or $l+2$.
Step 2.	IF $f(\alpha_i)=1$, THEN $\forall x (x > \alpha_i)$ set $f(x)=1$; IF $f(\alpha_i)=0$, THEN $\forall x (x < \alpha_i)$ set $f(x)=0$;
Step 3:	Store the $f(x)$ values which were obtained in step 2;

2.10.3. Construction of Hansel Chains

Several steps in the previous algorithms deal with Hansel chains. Next we describe how to construct all Hansel chains for a given space E_n of dimension n . First, we formally define a chain. A **chain** is a sequence of binary vectors (examples) $\alpha_1, \alpha_2, \dots, \alpha_i, \alpha_{i+1}, \dots, \alpha_l$ such that α_{i+1} is obtained from α_i by changing a "0" element to a "1". That is, there is an index k such that $\alpha_{i,k} = 0, \alpha_{i+1,k} = 1$ and for any $t \neq k$, the following is true $\alpha_{i,t} = \alpha_{i+1,t}$. For instance, the following list $\langle 01000, 01100, 01110 \rangle$ of three vectors is a chain. To construct all Hansel chains an iterative procedure is used. Let $E_n = \{0,1\}^n$ be the n -dimensional binary cube. All chains for E_n are constructed from chains for E_{n-1} . Therefore, we begin the construction with E_1 .

Chains for E_1 .

For E_1 there is only a single (trivial) chain and it is $\langle (0), (1) \rangle$.

Chains for E_2 .

First we consider E_1 and add at the beginning of each one of its chains the element (0). Thus, we obtain the set $\{00, 01\}$. This set is called E_2^{min} . In addition, by changing the first "0" to "1" in E_2^{min} , we construct the set $E_2^{max} = \{10, 11\}$. To simplify notation, we will usually omit "()" for vectors as (10) and (11). Both E_2^{min} and E_2^{max} are isomorphic to E_1 , clearly,

$$E_2 = E_2^{min} \cup E_2^{max}.$$

To obtain Hansel chains the chain $\langle 00, 01 \rangle$ should be adjusted by adding the maximum element (11) from the chain $\langle 10, 11 \rangle$. Thus, we obtain a new chain $\langle 00, 01, 11 \rangle$. Then element (11) is removed from the chain $\langle 10, 11 \rangle$.

Hence, the two new chains: $\langle 00, 01, 11 \rangle$ and $\langle 10 \rangle$ are obtained. These chains are the **Hansel chains** for E_2 , i.e., $E_2 = \{\langle 00, 01, 11 \rangle, \langle 10 \rangle\}$.

Chains for E_3 .

The Hansel chains for E_3 are constructed in a manner similar to the chains for E_2 . First, we double (**clone**) and adjust (**grow**) the Hansel chains of E_2 to obtain E_3^{min} and E_3^{max} . The following relations is also true:

$$E_3 = E_3^{min} \cup E_3^{max},$$

where $E_3^{min} = \{\langle 000, 001, 011 \rangle, \langle 010 \rangle\}$, $E_3^{max} = \{\langle 100, 101, 111 \rangle, \langle 110 \rangle\}$.

We then proceed with the same chain modification as for E_2 . That is, first we choose two isomorphic chains. Let it be two maximal length chains

$$\langle 000, 001, 011 \rangle \text{ and } \langle 100, 101, 111 \rangle.$$

Next the maximal element (111) from $\langle 100, 101, 111 \rangle$ is added to $\langle 000, 001, 011 \rangle$ and drop it from $\langle 100, 101, 111 \rangle$. Thus, the two new chains are obtained:

$$\langle 000, 001, 011, 111 \rangle \text{ and } \langle 100, 101 \rangle.$$

This procedure is repeated for the rest of the isomorphic chains $\langle 010 \rangle$ and $\langle 110 \rangle$. In this simple case just one new chain $\langle 010, 110 \rangle$ exists (the second chain is empty). Therefore, E_3 consists of the three Hansel chains

$$\langle 010, 110 \rangle, \langle 100, 101 \rangle \text{ and } \langle 000, 001, 011, 111 \rangle.$$

Figure 2.11 depicts the above issues. Similarly, the Hansel chains for $E_4, \dots, E_n$ are constructed recursively using the Hansel chains of $E_3, \dots, E_{n-1}$.

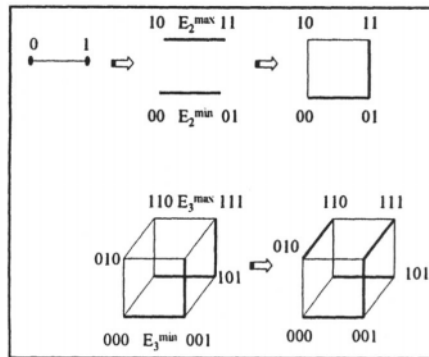


Figure 2.11. Construction of Hansel chains for E_3

Chapter 3

Rule-Based and Hybrid Financial Data Mining

Math is like love -- a simple idea but it can get complicated

R. Drabek

3.1. Decision tree and DNF learning

3.1.1. Advantages

Neural Networks provide a high level of predictive accuracy in many applications. However, their learned solutions do not facilitate **human inspection** or **understanding**. In contrast, “**symbolic**” **learning systems** usually produce solutions much more amenable to human comprehension [Graven, Shavlik, 1997]. They are based on discovering rules such as those that induce decision trees. For instance, rule R:

IF attribute V_1 (stock price) is greater than \$70 and
attribute V_2 (trade volume) is greater than \$500,000
THEN attribute V_3 (stock growth) will be positive ($V_3 > 0$)
for the next day.

is much more comprehensible than equation E:

$$3.5678V_1(t) + 0.0367V_2(t) + 0.00754V_1(t)V_2(t) + 0.001267(V_1(t))^2 = V_3(t+1).$$

Both can be wrong, but an expert can more easily test rule R. A correct rule itself has a value for an expert because it provides some explanation power. It is hard to use equation E for explanation without additional studies like the **sensitivity analysis** of output in econometrics with changing inputs.

Nevertheless, sensitivity analysis does not have the explanation power of logic rules. Rules can be constructed using sensitivity analysis, but it seems much more **natural** to begin by directly constructing rules. If this attempt fails, then equations could be constructed. Therefore, there are two major advantages of the comprehensibility of learned solutions:

- The **confidence** of users about the system's decisions (e.g., investment decision without confidence about the system is questionable);
- The importance of **understandable** rules that are behind the system's decisions. New rules may be even more valuable for a user than decisions themselves. They generate decisions and give generalized knowledge.

Many decision tree learning methods were developed over the last 30 years [Mitchell, 1997, Quinlan, 1993] and successfully used in financial and banking applications such as assessing the credit risk of loan applicants. Actually, each learned decision tree is a set of **human readable IF-THEN rules**. The term “decision tree” reflects a specific way of discovering and representing rules. This way of discovering creates an important advantage of decision tree learning – **discovered rules are consistent**. There is **no contradiction between rules** extracted from any given learned decision tree. On the other hand, rules discovered independently may contradict each other. This is typical if the training data contain significant noise. Therefore, readability and consistency are valuable properties of decision tree learning.

3.1.2. Limitation: size of the tree

In this section, we discuss the **limited expressive power** of decision trees. There are many consistent sets of rules, which cannot be represented by a relatively small single decision tree [Parsaye, 1997]. For example having data from table 3.1 we may wish to discover rules, which will describe the behavior of target T as a function of x_1, x_2, x_3 and x_4 .

Target T in table 3.1 is governed by simple **rule R1**:

IF ($x_1=1$ & $x_2=1$) OR ($x_3=1$ & $x_4=1$) THEN $T=1$

The decision tree equivalent to this rule is shown in figure 3.1. This kind of rule is called a **propositional rule** [Mitchell, 1997]. Propositional rules operate only with **logical constants** in contrast with first-order rules operating with **logical variables**. We will discuss first order rules in chapter 4, here we focus on decision trees. Rule R1 and its equivalent decision tree in figure 3.1 belong to the relatively restricted language of propositional logic.

Let's take $\mathbf{x}=(x_1, x_2, x_3, x_4)=(0, 0, 0, 0)$. This $\mathbf{x}$ is shown in the first line in table 3.1. We feed the decision tree in figure 3.1 with this $\mathbf{x}$. Node x_1 called

the **root** is the beginning point. Here $\mathbf{x_1=0}$, therefore we should traverse to the left from the root. Then we test the value of $\mathbf{x_2}$, which is also equal to 0. Next $\mathbf{x_3}$ and $\mathbf{x_4}$ are tested sequentially. Finally, we come to the leftmost **terminal node** with a 0 value for the target attribute.

Table 3.1. Training data

x_1	x_2	x_3	x_4	Target T
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

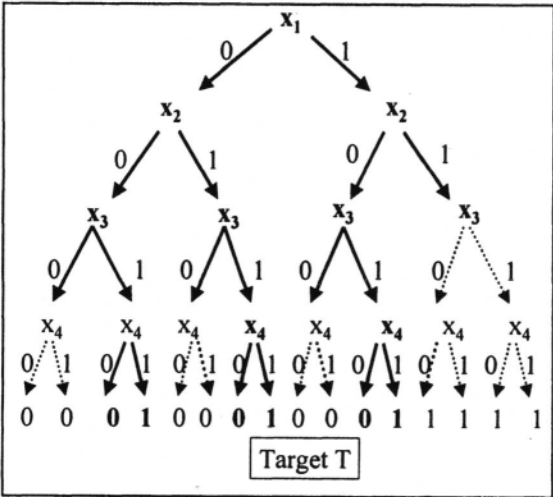


Figure 3.1. Decision tree

The tree in figure 3.1 has 31 nodes and can be simplified. For instance, the four rightmost terminal nodes on the bottom line produce the same value, 1. We merge them by assigning the rightmost node x_3 (in the third line) as a new terminal node. This node is marked in bold and all edges descending from this x_3 are eliminated (represented by dotted lines in figure 3.1). Similarly, other redundant nodes are merged. Finally, the decision tree has 19 nodes shown in bold in 1.1. This is significantly less than the total number of initial nodes, 31. Moreover, the new tree is also redundant. Two subtrees beginning from the leftmost x_2 on the line 2 are identical, i.e., independent of the values of x_2 , the target values are the same. Therefore, x_2 can be eliminated. The new simplified tree has 13 nodes. It is shown in bold in figure 3.2.

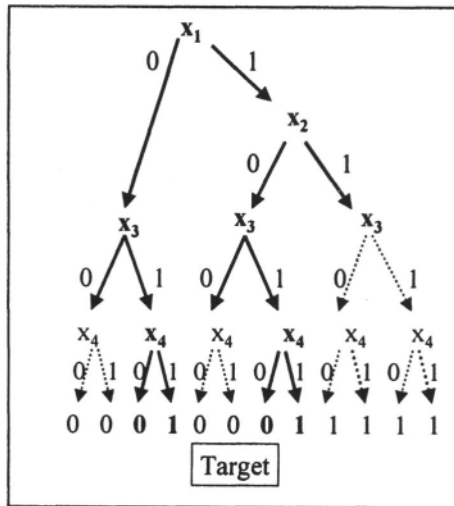


Figure 3.2. Simplified decision tree

The tree in figure 3.2 can be written as a set of rules. Each rule represents a path from the root to a terminal node. The IF-part ANDs all nodes of the branch except the leaf and the THEN-part consists of the leaf (terminal node). For example, the right-most branch of the tree in figure 3.2 can be written as follows:

IF ($x_1=1$ & $x_2=1$ & $x_3=1$ & $x_4=1$) THEN $T=1$

This tree returns a solution (a 0 or 1 value of the target T in a terminal node) for **all possible combinations** of the four binary attributes. We say that a decision tree produces a **complete solution** if it delivers an output

value for all possible input alternatives. There are many other trees, which do not deliver complete solutions. For example, we can remove the right-most branch of the tree in figure 3.2. The resulting shortened tree will not deliver solutions for combinations of attributes $\mathbf{x}=(x_1, x_2, x_3, x_4)$ with $x_1=1$.

Often decision tree methods implicitly assume a **close world**: all positive instances (i.e., with target values equal to 1, $T=1$) are presented in training data. Obviously, this is a questionable assumption for many applications. There is no guarantee that we are able to collect **all** positive instances. The decision tree in figure 3.2 delivers a complete solution; therefore, the close world assumption is satisfied for this tree. Hence it is enough to write only three rules with positive solutions ($T=1$) to represent all solutions given by this tree completely:

IF ($x_1=1$ & $x_3=1$ & $x_4=1$) THEN $T=1$

IF ($x_1=1$ & $x_2=0$ & $x_3=1$ & $x_4=1$) THEN $T=1$

IF ($x_1=1$ & $x_2=1$) THEN $T=1$.

We also can combine these rules into one **rule**,
Rule RT:

IF ($x_1=1$ & $x_3=1$ & $x_4=1$) OR
 ($x_1=1$ & $x_2=0$ & $x_3=1$ & $x_4=1$) OR
 ($x_1=1$ & $x_2=1$)
 THEN $T=1$.

This single rule represents the tree from figure 3.2. It is called the **disjunctive normal form (DNF)** of the rule. It is a special property of decision tree methods that all rules have a **specific DNF form**, i.e., all AND clauses (**conjunctions**) include a value of the attribute assigned to the **root**. In rule RT all AND clauses include the root x_1 . More generally in this type of rule, many AND clauses have significant overlapping. For example in rule RT, clauses ($x_1=1$ & $x_3=1$ & $x_4=1$), ($x_1=1$ & $x_2=0$ & $x_3=1$ & $x_4=1$) differ only in $x_2=0$. We call this specific DNF form of rule presentation the **tree form of a rule**. The tree form of rules tends to be long for some simple non-redundant DNF rules like **Rule RS**:

IF ($x_1=1$ & $x_2=1$) OR ($x_3=1$ & $x_4=1$) THEN $T=1$.

The last form of rule is called a **minimal DNF**. The corresponding tree has 4 components (tree edges) like $x_1=1$ and $x_4=1$ in contrast with 9 components

in the tree form of the same rule, RT. We show the comparative growth of rule size in more detail below. Decision tree algorithms need a special criterion to stop the search for the best tree, because the search time is growing exponentially as a function of the number of nodes and edges in a tree.

The standard stopping criterion is the number of components (nodes, edges) of the tree [Graven, Shavlik, 1997]. Therefore, if the search for rule RT will be restricted to six components, rule RT will not be discovered by the decision tree method, since rule RT has nine components in its IF-part.

Discovering rules without requiring the tree form of rules is called **discovering the disjunctive normal form (DNF) of rules**. The MMDR method (chapter 4) and the R-MINI method (section 3.2.3 in this chapter) belong to this class.

Rule RS is represented using only four components in the IF-part (comparisons of values of attributes with 1): $x_1=1, x_2=1, x_3=1, x_4=1$. It can be drawn as two small trees with five nodes each and 10 nodes total representing the rules:

IF ($x_1=1$ & $x_2=1$) THEN $T=1$,

IF ($x_3=1$ & $x_4=1$) THEN $T=1$.

These rules are shown in bold in figure 3.3.

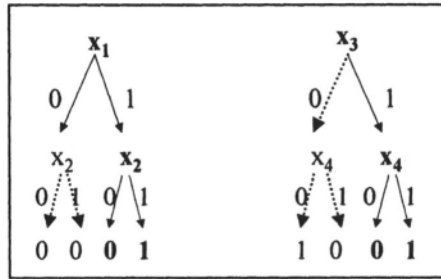


Figure 3.3. Two shorter decision trees

At first glance, the difference between trees in figures 3.2 and 3.3 is not large. However, this difference will grow very quickly with the number of pairs of attributes involved (see table 3.2).

For example to express **rule RG**

**IF ($x_1=1$ & $x_2=1$) OR ($x_3=1$ & $x_4=1$) OR
 ($x_5=1$ & $x_6=1$) OR ($x_7=1$ & $x_8=1$)
 THEN $T=1$**

a tree having 8 levels and 61 nodes is needed in contrast to 20 nodes for the set of separate small trees (see Table 3.2).

Table 3.2. Comparison of the number of nodes

Number of pairs	Number of nodes in the single tree	Number of nodes in the set of separate trees	Number of pairs	Number of nodes in the single tree	Number of nodes in the set of separate trees
1	5	5	16	262141	80
2	13	10	17	524285	85
3	29	15	18	1048573	90
4	61	20	19	2097149	95
5	125	25	20	4194301	100
6	253	30	21	8388605	105
7	509	35	22	16777213	110
8	1021	40	23	33554429	115
9	2045	45	24	67108861	120
10	4093	50	25	134217725	125
11	8189	55	26	268435453	130
12	16381	60	27	536870909	135
13	32765	65	28	1073741821	140
14	65533	70	29	2147483645	145
15	131069	75	30	4294967293	150

The larger **rule RL**:

IF ($x_1=1$ & $x_2=1$) **OR** ($x_3=1$ & $x_4=1$) **OR** ($x_5=1$ & $x_6=1$) **OR**
 ($x_7=1$ & $x_8=1$) **OR** ($x_i=1$ & $x_{i+1}=1$) **OR** ...**OR** ($x_n=1$ & $x_n=1$)
THEN $T=1$

requires a still larger tree. The general formula for the number of nodes $N(k)$ for one tree is:

$$N(k)=2N(k-1)+3,$$

and for the set of trees it is:

$$G(k)=5*k,$$

where k is the number of pairs such as ($x_i=1$ & $x_{i+1}=1$). These formulas are tabulated in table 3.2 for computing the number of nodes for a single tree and a set of decision trees equivalent to the single tree.

Table 3.2 shows that to represent a rule like RL with 5 pairs (10 nodes) we need 125 nodes in a single tree and only 25 nodes in a set of trees. For 10 pairs (20 nodes) this difference is much more significant: 4093 nodes vs. 50 nodes. For 30 pairs (60 nodes) we have about $4.29 \cdot 10^9$ vs. 150 nodes.

There are rules (**propositional rules**), which are more complex, e.g.,

IF $(x_1 < 5) \& (x_2 < 3)$ OR $(x_2 < 7) \& (x_3 > 9)$ THEN Target=1.

Having a more complex task with more than two values for each attribute, the numbers in table 3.2 become even larger for a single decision tree and grow exponentially. Therefore, although a decision tree can represent any propositional rule, this representation can be too large for practical use. Other more general propositional and relational methods can produce smaller rules in these cases (see Section 3.2.3 in this chapter and Chapter 4).

Table 3.3. Stock price data

Stock price On date t	Fore- casted stock price on date t+1	Target indica- tor	Signal on date t	Stock price on date t	Fore- casted stock price on date t+1	Target indica- tor	Signal on date t
17.70	17.60	1	Sell	17.92	18.09	-1	Buy
17.60	17.72	-1	Buy	18.09	18.08	1	Sell
17.72	17.70	1	Sell	18.08	17.90	1	Sell
17.70	17.71	-1	Buy	17.90	17.75	1	Sell
17.71	17.94	-1	Buy	17.75	17.84	-1	Buy
17.94	18.08	-1	Buy	17.84	17.97	-1	Buy
18.08	18.16	-1	Buy	17.97	18.08	-1	Buy
18.16	18.02	1	Sell	18.08	18.10	-1	Buy
18.02	17.92	1	Sell	18.10	18.11	-1	Buy

Let us consider an illustrative example of discovering three simple trading rules using data from Table 3.3:

1. IF the stock price today (date t) is greater than the forecast price for tomorrow (date t+1) THEN sell today;
2. IF the stock price today is less than the forecast price for tomorrow THEN buy today;
3. IF the stock price today is equal to the forecast price for tomorrow THEN hold.

In Table 3.3, a value of 1 indicates sell, while -1 indicates buy and 0 indicates hold. This example allows us to illustrate the difference between the methods. First, we discover rules 1-3 using Table 3.3 and the propositional logic approach. Table 3.3 allows the extraction of a propositional rule PR.

Propositional rule (PR):

IF ($a \leq 17.71 \& b > 17.7$) **OR** [$(17.71 < a \leq 17.9) \& (b > 17.8)$]
OR [$(17.9 < a \leq 18.06) \& (b > 18.0)$]
OR [$(18.06 < a \leq 18.2) \& (b > 18.08)$]
THEN “sell shares on date t.”

The bold line in Figure 3.4 illustrates this rule. The area above the line corresponds to buy, the area below corresponds to sell and the line itself represents the hold.

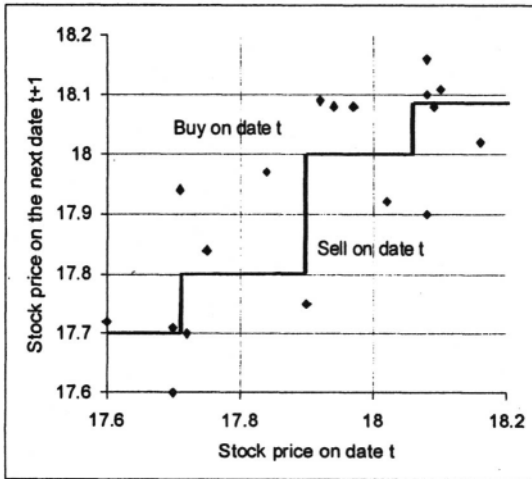


Figure 3.4. Diagram for decision tree boundary

We denote stock prices on date t (today) as a and stock prices on date $t+1$ (tomorrow) as b . Let's apply this rule for $(a, b) = (17.8, 17.5)$. All conjunctions of the rule PR are false for $(17.8, 17.5)$. Therefore, the rule does not deliver a sell signal, which is wrong. Propositional rule PR can also be discovered as a **decision tree** (Figure 3.5).

This tree is mathematically equivalent to propositional rule PR and gives the same incorrect buy/sell/hold signal as rule PR for $a=17.8$ and $b=17.5$. Obviously, the correct answer should be “sell shares on date t ” for these data.

This example shows that decision tree learning **cannot discover the simple relational rule**:

IF $S(t+1) > S(t)$ THEN sell.

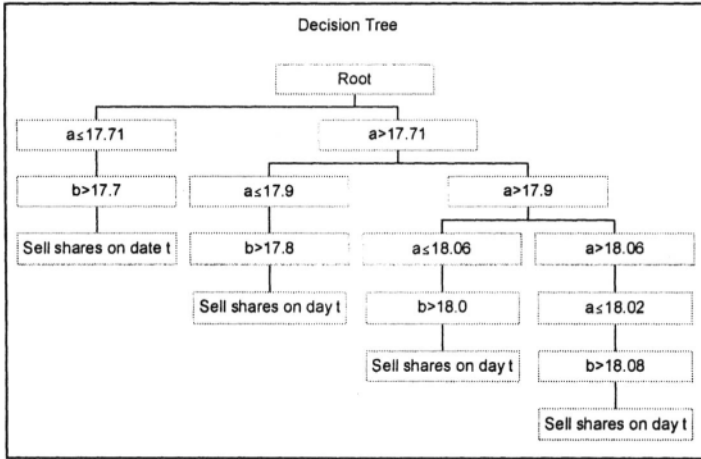


Figure 3.5. Example of decision tree

The reason is that decision tree learning methods work only with **only one attribute value** at a time by comparing this attribute value with a **constant** (e.g., $S(t) > 17.8$). Decision trees do not compare **two attribute** values of actual cases, for example $S(t+1) > S(t)$. **Relational methods** described in chapter 4 specifically address this issue.

Dietterich (1997) discusses similar constraints of the decision tree method. Different training samples will shift the locations of the staircase approximation shown in figure 3.4. By generating different approximations $F_i(\mathbf{x})$, $i=1, \dots, n$, a better classifier can be constructed to the diagonal decision boundary through combinations of these approximations. There are many ways for combining approximations. One of them would be **majority voting** $M(\mathbf{x})$:

$$M(\mathbf{x}) = \begin{cases} 1, & N_1 > N_0 \\ 0, & N_0 > N_1 \\ \text{no value} & \text{otherwise} \end{cases},$$

where N_1 is the number of approximations which classify $\mathbf{x}$ as a member of class #1 ($F_i(\mathbf{x})=1$ $i=1, \dots, n$) and N_0 is the number of approximations which classify $\mathbf{x}$ as a member of class #0 ($F_i(\mathbf{x})=0$, $i=1, \dots, n$).

Dietterich [1997] noted that these improved staircase approximations are equivalent to very **complex decision trees**. Those trees are too large to be included in a practical hypothesis space H . The space would be far too large for the available training data

The method of **voting ensembles of decision trees** is one the approaches to resolve size and accuracy problems of traditional decision trees in finan-

cial applications. On the other hand there are two current **limitations** of this approach pointed out by Dietterich [1997]:

1. Voting ensembles can require **large amounts of memory** to store and large amounts of computation to apply. For example, 200 decision trees require 59 megabytes of storage, which makes them less practical than other methods.
2. A second difficulty with voting ensemble classifiers is that a voting ensemble **provides little insight into how it makes its decisions**. A single decision tree can often be interpreted by human users, but an ensemble of 200 voted decision trees is much more difficult to understand.

Another alternative in finance is discovering **sets of rules**, i.e., rules in **propositional disjunctive normal form** (section 3.2.3 and [Apte, Hong, 1996]. The third approach is discovering **sets of first-order probabilistic rules** in financial time series (chapter 4 and [Kovalerchuk, Vityaev, 1998, 1999]).

3.1.3. Constructing decision trees

As we have seen in examples of decision trees above, the decision trees are built by sequentially partitioning the input space. A **splitting criterion (test)** is the core design element of a decision tree. A specific splitting test is matched to each **internal node**. In addition, **leafs** (terminal nodes) show **predicted target values** (classes). Thus three major decisions are made in the process of designing a decision tree:

1. Selecting the **root attribute** (e.g., x_1).
2. Selecting **splitting criteria** for the root and each of the internal nodes (e.g., test= $x_1 \leq 17.71$ for x_1 can produce branch 1 with $x_1 \leq 17.71$ and branch 2 with $x_1 > 17.71$).
3. Selecting **performance criterion**, e.g., the ratio of right and wrong classified instances for training and test data.

The typical algorithm stores:

1. a subset of the training examples associated with the node, e.g., set of all training instances such that $x_1 \leq 17.71$ for branch 1.
2. a set of constraints, representing the splitting test, e.g., $5 < x_1 < 10$, here 5 and 10 are constraints.

A typical decision tree algorithm uses this information for tuning splitting tests. The goal is to improve the value of a performance criterion. There are two types of **splitting tests**:

1. **Single attribute** splitting test, used in C4.5 algorithms [Quinlan, 1993].
2. **Multiple-attribute** splitting test, used in an enhanced version of ID3-of-3 and Trepan algorithms [Murphy, Pazzani, 1991; Graven, Shavlik, 1997].

Figure 3.5 illustrates the idea of single-attribute splitting tests like $x_1 \leq 17.71$. Another example of a single-attribute test is $5 < x_1 < 10$. The multiple-attribute splitting test in ID2-of-3 uses **m-of-n** expressions for tests at its internal nodes. The multiple-attribute splitting test can discover rules such

Rule MFR:

IF ($x_1=1$ & $x_2=1$) OR ($x_3=1$ & $x_4=1$) OR ($x_1=1$ & $x_3=1$) OR
 ($x_1=1$ & $x_4=1$) OR ($x_2=1$ & $x_3=1$) OR ($x_2=1$ & $x_4=1$)
 THEN T=1.

This rule is a 2-of-4 type rule, i.e., the rule is true if any two of four attributes are true.

Each m-of-n tree can be viewed as a set of ordinary trees connected with OR. For example, rule MFR can be written as three rules MFR1, MFR2 and MFR3 connected with OR:

Rule MFR1:

IF ($x_1=1$ & $x_2=1$) OR ($x_1=1$ & $x_3=1$) OR ($x_1=1$ & $x_4=1$)
 THEN T=1.

Rule MFR2:

IF OR ($x_3=1$ & $x_4=1$) OR ($x_1=1$ & $x_3=1$) OR ($x_2=1$ & $x_3=1$)
 THEN T=1.

Rule MFR3

IF ($x_3=1$ & $x_4=1$) OR ($x_2=1$ & $x_4=1$)
 THEN T=1.

Rule MFR1 can be presented as an ordinary tree with root x_1 and branches going to x_2 , x_3 and x_4 . Similarly, rule MFR2 has root x_3 and rule MFR3 has root x_4 .

In section 3.1.2, we discussed rule R1:

IF ($x_1=1$ & $x_2=1$) OR ($x_3=1$ & $x_4=1$) THEN T=1.

This rule is true if any of two **specific pairs** of attributes are true. A compact 2-of-m rule type does not cover this rule. We will call this type of rule **2s-of-m** in order to distinguish it from non-specific 2-of-m rules. Another language that is more general is needed for the compact presentation of these and many other rules. This form is discussed in Section 3.2.3.

It is important to mention that m-of-n rules follow a form similar to extracting rules from neural networks. In essence, **activation functions for nodes of a neural network work as m-of-n nodes in decision trees**. The advantage of using m-of-n tests is that they often result in trees that are more concise. Originally, Murphy and Pazzani [1991] introduced the idea of using m-of-n expressions as splitting tests in decision trees. Methods such as Trepan extract a decision tree from a trained neural network by identifying m and n. C4.S, ID2-of-3, and Trepan algorithms use a heuristic search process to construct their splitting tests.

Evaluating decision tree criteria. Two criteria are critical for evaluating the learned decision trees:

1. Predictive accuracy and
2. Comprehensibility.

Usually **accuracy** is measured using the examples in the test set as the percentage of test set examples that are correctly classified. There is no direct measure of the comprehensibility of a decision tree. Therefore, different indirect measures are used to represent comprehensibility of trees such as:

- the number of internal (i.e., non-leaf) nodes in the tree, and
- the number of attribute references used in the splitting tests of the tree.

The test is called a **single-attribute test** if only one value of the attribute is involved, for instance, $S(t) > 17$. A single attribute test is counted as **one attribute** reference for computing the complexity of the tree. An m-of-n test is counted as n attribute references, because it lists n attribute values [Graven, Shavlik, 1997]. Table 3.2 shows how quickly the number of nodes of the tree grows for 2s-of-m rules. It is very difficult to observe and comprehend a tree with 100 or more nodes even if smaller parts of the tree are understandable. Table 3.2 shows that a **set of separate trees** for rules like R1 based on 20 pairs of attributes requires 100 nodes. In contrast, a **single tree** with only five pairs of attributes requires even more nodes (125 nodes). Therefore, by measuring the **comprehensibility** of a tree by its **complexity**, we may conclude that a rule with only 10 attributes (5 pairs) is not comprehensible. This example shows that complexity can be the **wrong measure** of the comprehensibility of a rule.

Use of individual attribute distribution. A distribution of values of an individual attribute in training data can be used to set up the splitting test. For example, the distribution could show that most of the instances with $x_1 \leq 17.71$ belong to the class 1 (“buy”) and the most of the instances with $x_1 > 17.71$ belong to class 2 (“sell”). However, there is a trap in this approach—such a distribution does not take into account **dependencies** among attributes. This approach can extract rules, which do not reflect the essential attribute dependencies. Table 3.3 and Figures 3.4 and 3.5 represent this case. There are some possibilities to capture conditional dependencies, and thus to

construct a more accurate model of the true distribution. One of them is to compute conditional probabilities for particular nodes. In some cases, they may instead provide **worse estimates** because they are based on less data. To handle this trade-off, several methods apply **statistical tests** to decide whether to use the local conditional probability for a node [Graven, Shavlik, 1997].

Stopping Criteria. A typical algorithm tries many partitions to find a better one. Often a new partition is the result of merging or splitting groups in the current partition. The process stops when any merge or split does not form a better partition according to local and/or global stopping criteria. A **local criterion** uses characteristics of **a single node** and a **global criterion** uses characteristics of the **entire tree**. For instance, the local criterion used by Trepan [Graven, Shavlik, 1997] is based on the **purity** of the set of examples covered by a node. Purity controls the proportion between training examples from different classes that reached the node. In addition, Trepan employs two global stopping criteria:

- a limit on the size of the tree and
- a statistical **significance test** on a validation set.

The first global criterion is intended to control the comprehensibility of the tree and the second one is intended to control accuracy of the tree.

3.1.4. Ensembles and Hybrid methods for decision trees

Many studies have shown advantages of ensembles of different decision trees and hybrids of decision trees with other methods when compared to individual decision tree design. In this section we compare an individual decision tree constructed using the **C4.5 algorithm** [Quinlan, 1993] and different ensembles of decision trees. C4.5 algorithm is one of the most widely used decision tree algorithms. C4.5 consists of the following steps:

1. Choosing the attribute to be a starting node (root) in the decision tree.
2. Generating various possible attribute-splitting tests at the node (e.g., creating branches $x < 17.7$ and $x \geq 17.7$).
3. Ranking generated attribute tests using the Information Gain Ratio Criterion (IGRC).
4. Choosing the attribute-value splitting test top-ranked according to IGRC.
5. Extending the decision tree with new nodes corresponding to the best test.
6. Repeating steps 2-4 for each internal node of the decision tree.

The algorithm splits discrete valued attribute and real-valued attributes differently. If attribute S has V discrete values then the data can be split into V subsets. Each subset contains objects with one of V values. If any real

number can be a value of attribute S then the data is split using some threshold T into 2 subsets ($S < T$ and $S \geq T$).

Ensembles. The first approach to create ensembles of decision trees is to run a decision tree learning algorithm several times, each time with a different subset of the training examples (subsamples). We already briefly discussed three subsampling methods in Section 2.8.1:

4. **Random selection** of subsets (bootstrap aggregation -- bagging),
5. Selection of **disjoint subsets** (cross-validated committees),
6. Selection of subsets according **probability distribution**.

Sometimes, methods which use random selections of subsets are called bagged methods, e.g., **bagged C4.5**.

There are several versions of these methods, which are summarized below in table 3.4 following to [Dietterich, 1997].

Table 3.4. Subsampling methods for the training set

Method name	Subsampling mechanism	Example of subsample parameters
1. Bootstrap aggregation (bagging)	Draw randomly s examples with replacement.	About 60 % of the training set. Examples can appear multiple times.
2. Selection of disjoint subsets (cross validated committees, k -fold cross validation)	Divide the training set into k disjoint subsets. Construct k overlapping training sets by dropping out a different one of these k subsets.	$k=10$
3. Boosting (AdaBoost)	Draw s examples with replacement according the probability distribution $p_i(x)$.	60-70% of the training set

For example, the AdaBoost boosting method uses different probability distributions to better match an ensemble of classifiers with the most difficult training data. Below steps of this algorithm are presented [Feund, Schapire, 1995,1996; Dietterich, 1997]:

- Step 1. Compute a probability distribution $p_i(x)$ over the training examples Tr .
- Step 2. Draw a training set $A_{i,s}$ of size s according the probability distribution p_i .
- Step 3. Produce a decision tree (classifier h_i) using the decision tree learning algorithm.
- Step 4. Compute the weighted error rate (Er) of classifier h_i on the training examples using $p_i(x)$: $Er = \sum_{i=1, N} p_i(x_i) Er(x_i)$, where $Er(x_i)$ is error of classifier h_i for example x_i .
- Step 5. **Adjust** the probability distribution $p_i(x)$ on the training examples

(examples with higher error rate $\mathbf{Er}(\mathbf{x}_i)$ obtain higher probability values).

Step 6. Generate a new training subsample $\mathbf{A}_{i,k}$ of size k with replacement according to the **adjusted probability distribution** $\mathbf{p}_i$ and repeat step beginning from step 3.

The final classifier, $\mathbf{h}_f$, is constructed by a weighted vote of the individual classifiers, $\mathbf{h}_i$. Each classifier is weighted according to its accuracy for the distribution $\mathbf{p}_i$ that it was trained on.

Table 3.5, condensed from descriptions in [Dietterich, 1997], summarizes performance of different decision tree methods in comparison with their ensembles. Most of these comparisons were done using the C4.S algorithm as a benchmark method [Quintan, 1993] with one exception for the option method [Buntine, 1990]. The option method produces decision trees where an internal node may contain several **alternative splits** (each producing its own sub-decision tree). Actually, an option tree is a set of voted conventional sub-decision trees.

Table 3.5. Comparison of C4.5 decision trees and ensembles of decision trees.

Underlying benchmark algorithm	Ensemble	Results
C4.5	C4.5 with AdaBoost.M1	The ensemble outperformed the underlying algorithm. [Feund, Schapire, 1995, 1996]
C4.5	C4.5 with a weighted training sample	The ensemble outperformed the underlying algorithm [Quinlan, 1996]
C4.5	Bagged C4.5	The ensemble outperformed the underlying algorithm [Dietterich, 1997]
C4.5 with injected randomness	Bagged C4.5	The ensemble outperformed the underlying algorithm [Dietterich, 1997]
Option tree	Bagged C4.5	The results are comparable while the ensemble produces a more understandable result [Kohavi, Kunz, 1997]

An alternative approach to creating ensembles of decision trees is to **change a tree** already discovered by using **transition probabilities** to go from one tree into another one. For instance, the **Markov Chain Monte Carlo (MCMC) method** [Dietterich, 1997] interchanges a parent and a child node in the tree or replaces one node with another. Each tree is associate with some probability $\mathbf{P}(\mathbf{h}_i)$. Then these trees are combined by weighted vote to produce a forecast. Probabilities $\mathbf{P}(\mathbf{h}_i)$ can be assigned using some prior probabilities and training data in the Bayesian approach. The process of generating decision trees and assigning probabilities as transition prob-

abilities from one tree to another one is modeled as a Markov process. More about general Markov processes can be found in [Hiller, Lieberman, 1995].

3.1.5. Discussion

A **learning algorithm** is **unstable** if its forecast is altered significantly by a small change in training data. Many examples of such instabilities are known for decision tree, neural network, and rule learning methods. The linear regression, k nearest neighbor, and the linear discriminant methods suffer less from these instabilities. Dietterich [1997] argues that voting ensembles of unstable decision trees learning methods are much more **stable** than the underlying decision tree methods. In addition to better stability of the forecast, Table 3.5 shows that such a forecast **fits** the real target values **better**, producing less forecasting errors.

Why do individual decision trees often perform worse than the voting ensembles built on them? There are at least three reasons: insufficient training data, difficult search problems, and inadequate hypotheses space [Dietterich, 1997]:

1. **Insufficient training data.** Usually several hypotheses are confirmed on training data. It would be unwise to prefer one of them and reject others with the same performance knowing that the data are insufficient for this preference.

2. **Search problems.** It is computational challenge to find the smallest possible decision trees or neural networks consistent with a training set. Both problems are NP-hard [Hyafil, Rivest, 1976; Blum, Rivest, 1988]. Therefore, **search heuristics** became common for finding small decision trees. Similarly, practical neural network algorithms search only locally optimal weights for the network. These heuristics have a chance to produce better solutions (decision trees or neural networks) if they use slightly different data as done with ensembles of classifiers.

3. **Inadequate hypothesis space.** It is possible that an hypothesis space H does not contain the **actual function** f , but several acceptable approximations to f . **Weighted combinations** of them can lead to an acceptable representation of such f .

There are two potential problems with having an inadequate hypothesis space. H may not contain a weighted combination of decision trees close to f . Alternatively, such a combination can be too large to be practical. In both cases ensembles do not help to find f . Therefore, a **wider hypothesis space** and more expressive hypothesis language are needed. The DNF and first order languages provide this option.

The fight with insufficient training data is also not so straightforward -- add as many additional data as possible. It is common that the real shortage is in so called “border” data -- data, which are representing the **border** between classes. For instance, in figure 3.4, the diagonal is the actual border between two classes, but available training data are not sufficient to identify the diagonal unambiguously. These data permit several significantly different borderlines. Indeed, having thousands of training examples located far away from the border does not permit the border to be identified with the desired accuracy. In Figure 3.4, the simple linear discriminant function has enough expressive power to solve this problem completely. Just a few points on the border (“hold” state) or a few points really close to this border between classes “buy”, “sell”, are needed. Nevertheless, thousands of additional examples representing “buy” and “sell” situations far away from the actual border are useless for identifying this border.

3.2. Decision tree and DNF learning in finance

3.2.1. Decision-tree methods in finance

Langley and Simon [1995] listed some typical financial systems, developed with decision tree methods:

- Making **credit card decisions** for card issuing companies through the evaluation of credit card applications.
- Advice on **share trading** for security dealers in six European countries.
- Prediction of which **overdue mortgages** are likely to be paid.
- **Monitoring excessive claims** in health insurance from both clients and providers for different medical treatments.

Graven and Shavlik [1997] used the decision tree algorithm Trepan in predicting whether the **currency exchange rate** will go up or down on the next day.

Let us illustrate the “borderline” problem from the previous section with the loan making example found in [Michie, 1989; Langley and Simon, 1995]. A typical decision of loan companies is to accept or reject a loan application using information about an applicant. The trained statistical decision making system at American Express, UK was able to process from 85-90% of the applications. The remaining 10-15% “borderline” applications were analyzed by a loan officer, who forecast whether these borderline applicants would default on their loans. These loan officers were at most 50% accurate in such prediction.

This example shows the need for special attention to “**borderline**” **performance** for the complex tasks. The standard performance criterion described in section 3.1.3 is the ratio of right and wrong classified instances for training and test data. A more advanced criterion would give higher weights to examples which are closer to the ‘borderline’. The decision tree (figures 3.4 and 3.5) performs well under the standard criterion and fails under the borderline criterion.

American Express, UK motivated by the borderline problem developed a decision tree based on **1014 training cases and 18 attributes** (such as age and years with an employer). The tree contains around **20 nodes** and 10 of the original 20 attributes that made correct predictions on 70% of the **borderline applicants**. In addition, the company has benefited from the explanatory ability of the decision tree, because by using the tree and underlying rules the company could explain the reasons for decisions to applicants. This exploratory project took less than **a week** to designing the tree, but American Express UK was so impressed that they put the resulting knowledge base into use without further development [Michie, 1989; Langley and Simon, 1995].

It is important to note that Michie et al. were able to construct a small tree (around 20 nodes) having 10 original attributes. Note, just for comparison, that the complete binary decision tree for 10 binary attributes consists of about 2000 nodes. That is, the discovered tree is a hundred times smaller than the complete tree. Unfortunately, experience shows that the chance of discovering small trees for complex problems is not high. The next section demonstrates this for a complex financial time series like SP500C.

3.2.2. Extracting decision tree and sets of rules for SP500

In this section, we consider an example of forecasting the up-down direction of SP500C using its ten-year daily data (1985-1994) -- 2513 training examples. SP500C data for 1995-1998 are used for testing -- 938 testing examples. Each example is a ($\mathbf{v}$, y) pair, where $\mathbf{v}$ represents a SP500C ten-day history and y is its up/down indicator for the following day. The y indicator is equal to 1 (class 1) if SP500C goes down and **$y=2$** (class 2) if SP500C goes up.

The task is to study the possibility of discovering a small decision tree capable forecasting the y value using $\mathbf{v}$. More precisely, $\mathbf{v}$ represents a vector $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{10})$, where each $\mathbf{v}_i$ is the relative difference of SP500C for the current date i with the SP500C for the previous trading day $(i-1)$:

$$\mathbf{v}_i = (\text{SP500}(i) - \text{SP500}(i-1)) / \text{SP500}(i).$$

These numbers are small. They were scaled using a linear transformation (av_i+b). Therefore, figures below show a discovered decision tree and rules using a scaled v_i .

A fragment of the extracted decision tree (the first 9 of 25 levels) is presented in Figure 3.6. This tree has 112 nodes, 37 leaves, and a maximum of depth 25. It was extracted using the Sipina_W method [Zighed, 1992] and software [<http://www.epcad.fr/sipina/sipinaE.html>],

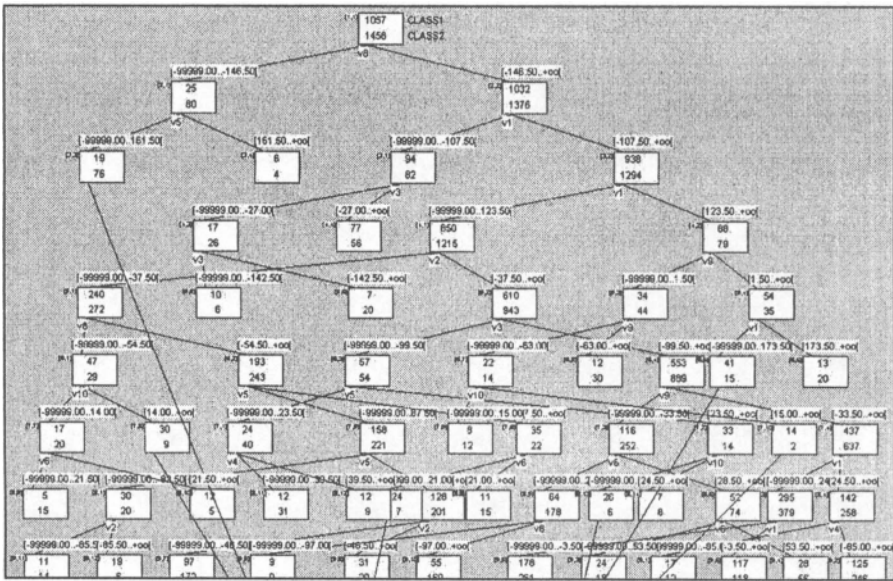


Figure 3.6. Fragment of decision tree for SP300 (first 9 levels from 25 levels)

Two essential notions are used in Figure 3.6: vertices and partitions. The root (node 1.1) shows all 2513 training examples (1057 downs and 1456 ups) before partitioning. The first partition deals with attribute v_1 . This attribute is split by $v_1 = -146.5$ producing node 2.1 with threshold $v_1 < -146.5$ and node 2.2 with threshold $v_1 \geq -146.5$. Node 2.1 is relatively effective node -- 76.2% of the examples in this node belong to "up" class for SP500C.

Node 2.2 needs further partitioning. In the process of this partitioning, an effective node 7.9 on 7th level is produced (87.5% of examples in the node are from the "up" class. That is, two examples are from the "down" class and 14 examples are from the "up" class. Further partitioning is needed for other nodes.

In total Sipina_W extracted 85 simplified propositional rules from the decision tree: 36 rules for the first class and 49 rules for the second class.

Propositional rules extracted from the decision tree were simplified using the C4.5 algorithm along with finding the best pessimistic error rate on each rule [Quinlan, 1993]. A total accuracy of 67.5% was reached on the training data (see table 3.6) by 32 of the 85 rules. These 32 rules cover 105 examples per rule on the average. Figure 3.7 shows their distribution.

Table 3.6. Performance of the decision tree on training data 1985-1994 (112 nodes, 37 leaves, max depth 25.)

	Forecast class 1 (down)	Forecast class 2 (up)	Unclassi- fied	Accuracy rate
Actual class 1 (down)	460	581	16	43.7%
Actual class 2 (up)	203	1237	16	84.9%
Total	663	1818	32	67.5%

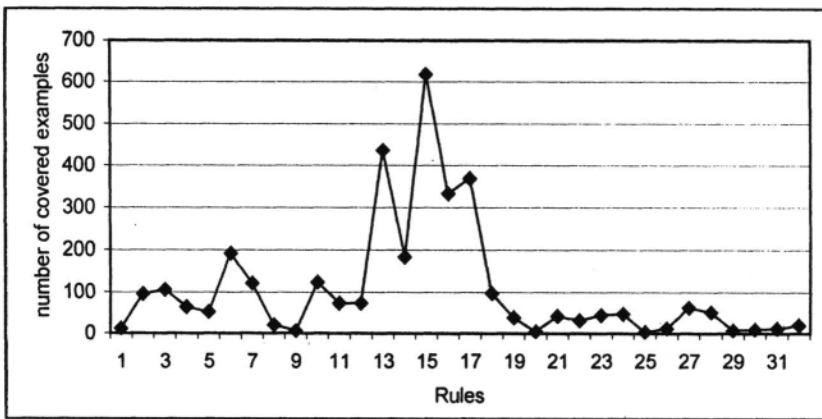


Figure 3.7. Number of examples covered by the rules with error rate better than 67.5%

There are only five completely correct rules (5.8%) out of the 85 rules extracted from the decision tree. They cover $9+4+3+7+6=29$ of the 2513 training examples (1.15%) or about six training examples per rule (see table 3.7). We call these rules **error-free rules**.

With this ratio of error-free rules, it is not surprising that the decision tree and its 85 rules failed to forecast the “down” direction of SP500 for the testing data. The results are 58.1% for 1995-1996 and 52.8% for 1997-1998 (see tables 3.8 and 3.9). The average performance on all the test data (1995-1998) for both “up” and “down” classes is 55.6% with a relatively high accuracy for the “up” class (77.1% and 70.3%).

Table 3.7. Error-free propositional rules extracted by C4.5 algorithm from a decision tree

Category	Expression	Cove- red exam- ples	Statistical signifi- cance
IF part for Class 1 (down stock direc- tion)	1. $v1 \in [22.50, 123.50)$ and $v2 \geq -37.50$ and $v4 < -80.00$ and $v9 \geq -91.50$ and $v10 < -27.50$	9	0.995
	2. $v1 \in [22.50, 123.50)$ and $v2 \geq -37.50$ and $v4 \in [-80.00, 4.50)$	4	0.901
	and $v7 \geq 83.00$ and $v8 \geq -14.00$ and $v10 < -27.50$	3	0.824
	3. $v1 \in [-107.50, 22.50)$ and $v2 < -37.50$ and $v3 \geq 120.50$ and $v7 \geq 83.00$ and $v8 \in [-4.50, 14.00)$	7	0.983
	4. $v2 < -37.50$ and $v3 \geq 20.50$ and $v4 \geq 4.50$ and $v7 \geq 83.00$		
IF part for Class 2 (up stock direction)	5. $v1 \in [9.50, 22.50)$ and $v2 \geq -37.50$ and $v4 \in [-46.50, 4.50)$ and $v5 < -73.50$ and $v7 < 83.00$	6	0.920

Table 3.8. Performance of the decision tree on testing data 1995-1996 (112 nodes, 37 leaves, max depth 25)

	Forecast class 1 (down)	Forecast class 2 (up)	Unclassified	Accuracy rate
Actual class 1 (down)	28	126	5	18.2%
Actual class 2 (up)	74	249	9	77.1%
Total	102	375	14	58.1%

Table 3.9. Performance of the decision tree on testing data 1997-1998 (112 nodes, 37 leaves, max depth 25)

	Forecast class 1 (down)	Forecast class 2 (up)	Unclassified	Accuracy rate
Actual class 1 (down)	49	120	5	29.0%
Actual class 2 (up)	74	175	9	70.3%
Total	129	295	14	52.8%

Thus, only a small fraction of rules produced by the decision-tree method is fully consistent with training data. Moreover, these rules do not form a short decision tree. They actually come from very different parts of a large extracted decision tree with 112 nodes, 37 leaves, and a depth of 25. The first three error-free rules from table 3.9 can be combined into a short tree beginning from $v1$, but rules 4 and 5 do not belong to that tree. Note that the successful use of the decision tree method described in section 3.2.1 required only 20 nodes with 10 attributes.

This consideration illustrates our claim in section 3.1.2 that for many applications, the decision tree language is too **restrictive**.

Recognition of this fact led to extending decision tree software systems to more **general rule** extraction and to creating tools to combine rules extracted from different trees (e.g., Sipina_W [Zighed, 1992], C4.5 [Quinlan, 1993]). Extracting rules in disjunctive normal form (DNF) is an alternative generic approach to this problem. We discuss DNF success in finance in the next section.

3.2.3. Sets of decision trees and DNF learning in finance

Examples in sections 3.1.2, 3.1.3, and 3.2.2 in this chapter show that the attempt to construct a single decision tree can produce very large tree (table 3.2) or/and inaccuracies. Different ways to overcome these problems have been suggested. For example in [Heath et al, 1993] a randomized decision tree induction algorithm was suggested. The algorithm generates different decision trees every time it is run. Each tree represents a different **artificial expert** (decision maker). These trees are combined using a **majority-voting** scheme in order to overcome the **small errors** that occur in individual trees. These methods are also called leaning **voting ensembles** or learning a committee of decision trees. A majority-voting scheme was defined in section 2.3 and further discussed in section 3.1.4.

There are two extreme cases with sets of decision trees:

- all trees are applicable for all data instances (complete overlapping case),
 - each tree has its own area in the attribute space (non-overlapping case).
- The majority-voting scheme is reasonable for the cases with significant overlap. The major **disadvantage** of voting schemes is that they provide interpretations that are less clear than individual rules [Dietterich, 1997].

Another approach is to **generalize** the concept of a decision tree. We have already discussed one of the generalization methods – the m-of-n form of a decision tree. They do not solve the problem completely. There are many sets of rules, which cannot be represented by relatively small m-of-n rules. Propositional and first-order logic methods for discovering rules in disjunctive normal form (DNF) have expressive power to address the size limitation of decision trees in way that is much more general. We present details about first-order logic methods in chapter 4.

Next we discuss results obtained at IBM Research in discovering DNF rules for financial applications using the R-MINI method [Apte, Hong, 1996]. The rule is in **disjunctive normal form** when the IF-part is presented as a set of **AND clauses** (statements) connected with ORs. For example: the rule

IF ($x_1=1$ & $x_2=1$) OR ($x_3=1$ & $x_4=1$) THEN T=1

consists of two AND clauses $(x_1=1 \ \& \ x_2=1)$, $(x_3=1 \ \& \ x_4=1)$ connected with an OR. Several approaches have been suggested for generating DNF rules from data [Michalski et al., 1986; Clark, Niblett, 1989; Weiss, Indurkha, 1993]. Generating DNF also is called **learning sets of rules** [Mitchell, 1997]. These methods create one rule at a time. Then examples covered by this rule are deleted from the training data, before repeating the construction step. The process is continued to find the least complex rule (DNF) covering the training set. The **complexity** of a DNF is measured by the total number of AND clauses and total number of tests. In Apte and Hong [1996] 569 AND clauses and related rules were generated for SP500 using 4901 training examples represented with 30 attributes each. Specifically, the task was to generate **minimal DNF rules**. These numbers show, however, that even with this purpose in mind, the actual number of rules is still large (one rule for each 8.6 examples on the average).

This amount is close to the six training examples per error-free rule in Section 3.2.2 for SP500. Apte and Hong do not present a distribution like that given in Figure 3.7 about the number of examples actually covered by each rule. They mentioned that rules were cut if they cover three or less examples. It was assumed that such rules “discover” the noise component of the data.

The purpose of rule discovering in [Apte, Hong, 1996] was **managing equity investment portfolios**. A portfolio management scheme based upon these rules was constructed and compared with a simulated SP500 index fund performance. The authors report that the simulated prediction-based portfolio **returns** 54% per year in comparison with the SP500 index return of 22%. The following trading strategy was used:

- Sell all securities whose predicted excess return is less than -6%,
- Buy all securities whose predicted excess return is greater than 6%.

In both cases a 0.5% transaction fee is applied to every trade.

This strategy gives a specific treatment to **borderline instances** -- buy all securities only if the predicted excess return is greater than 6%. In this study, five-year data are used to simulate return. The method used by Apte and Hong can be called a **hybrid method**, because to get the final decision it is combined with other methods. It is important to note that in this study, the **rule-based method outperformed the benchmarks**. Apte and Hong optimistically concluded that capital market domains could be effectively modeled by DNF classification rules induced from available historical data for making gainful predictions about equity investments.

3.3. Extracting decision trees from neural networks

3.3.1. Approach

The rule-based “**symbolic**” approach described in Sections 3.1 and 3.2 represents learned solutions in an **understandable form**. Therefore, end users can **examine** the learned solutions independent of their understanding of the underlying mathematical tools. This type of representation is difficult to obtain for neural networks. Neural network representations are usually incomprehensible to humans [Shavlik, 1994; Graven, Shavlik, 1997]. However, the pure rule-based approach ignores valuable findings produced by neural networks. A hybrid approach **extracts symbolic representations** like decision trees from a trained neural network and combines both discoveries in comprehensible manner.

In this approach, a trained neural network is used as **input** for a rule discovery algorithm. The algorithm produces as output an approximate, **symbolic representation** of the concept represented by the trained network. Over the last decade, several studies have been done in this area [Andrews et al., 1995; Shavlik, 1994]. One of the studies was specifically devoted to a financial application – predicting the **Dollar-Mark exchange rate** using the Trepan algorithm [Graven, Shavlik, 1997].

Two types of methods for extracting rules from the underlying neural network are possible:

1. **structure specific** and
2. **structure independent**.

Structure independent methods have an obvious advantage - they can use an **arbitrary network** as a black box (“**expert**”) for obtaining simulated target values. Meanwhile, the structure specific methods may more efficiently extract rules from a neural network by taking advantage of the particular structure.

The next classification of rule-extracting methods is based on the input data types. Three types of **inputs** and **outputs** make a significant difference for rule-extracting algorithms:

1. **discrete valued inputs** and outputs are used in classification trees;
2. **continuous valued inputs** and outputs are used in regression trees;
3. **mixture** of discrete- and continuous valued inputs and outputs are used in mixed trees.

It is also important to note that decision trees can be extracted from:

- training data directly,
- a trained network, and
- both a trained network and training data.

In the second and third cases, a neural network serves as an “**oracle**”, which can produce target values for many **thousands** of artificial **training examples**. If the “oracle” is well trained then these additional examples can help significantly in designing a decision tree. This means that there is **no significant limitation** on the amount of available (artificial) training data for training a decision tree.

3.3.2. Trepan algorithm

The Trepan algorithm exemplifies extraction of a decision tree from a trained neural network. The general logic of the Trepan algorithm is presented in figure 3.8. There are many similarities between the Trepan and conventional decision tree algorithms, that is, learning directly from a training set (see for instance, CART [Breiman et al., 1984] and C4.5 [Quinlan, 1993]). The major difference is that the Trepan interacts with the trained neural network along with the training set used. The neural network provides predicted target values for **artificial training examples**. It is not necessary that these predicted target values are the same as actual target values, especially if the network was trained using insufficient data.

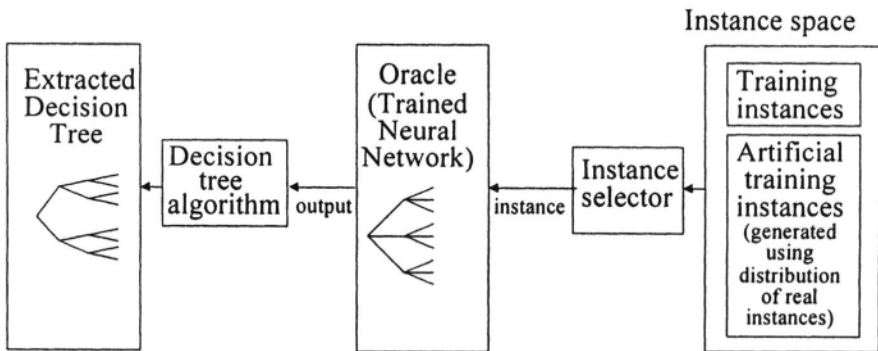


Figure 3.8. Extraction of Decision Tree from Neural Network

However, the Trepan algorithm treats the network's forecast as “ground truth”. The Trepan tree grows by selecting the best nodes first (best-first search method, [Russel, Norvig, 1995]) according to the **node evaluation function** $f(N)$:

$$F(N) = \text{reach}(N) * (1 - \text{fidelity}(N)),$$

where $\text{reach}(N)$ is the estimated fraction of instances that reached the node and $\text{fidelity}(N)$ is a measure of closeness between the tree's outputs and the network's outputs for real and generated examples. The splitting test in each

node is tested against a threshold on the minimal number of examples at the node. This threshold (called `min_sample`) controls the reliability of the tree. Input examples for the algorithm are selected using the underlying distribution of training data.

Graven and Shavlik [1997] noted some difficulties in extracting rules with high fidelity with respect to their target networks using Trepan. They suggested two ways to address this problem:

- Use **languages** other than decision trees to represent the extracted models;
- Apply **truth-preserving transformations** for simplifying the extracted decision trees.

Relational methods, which we discuss in chapter 4, offer this language--first order language. In addition, the representative measurement theory presented in chapter 4 specifically addresses truth-preserving transformations for continuous valued attributes.

3.4. Extracting decision trees from neural networks in finance

3.4.1. Predicting the Dollar-Mark exchange rate

In this section, we discuss **predicting the daily Dollar-Mark exchange rate** using a decision tree extracted from an independently trained Neural Network. According to Graven and Shavlik [1997] the decision tree produced from the training set without requests to the “oracle” (learned neural network) is much less accurate and comprehensible than the tree extracted by the Trepan algorithm. Formally, three evaluation parameters have been used to evaluate the decision tree: complexity, predictive accuracy, and fidelity to the network. The **fidelity** parameter measures how close the forecast of the decision tree is to the forecast by the underlying neural network.

Data. The data for the task consist of daily values of foreign exchange rate between the U.S. Dollar and the German Mark from January 15, 1985 through January 27, 1994 [Weigend et al., 1996] where:

1. the test set consisted of the last 216 days;
2. the validation set consisted of 535 days (every fourth day from the remaining data);
3. the training set consisted of the remaining 1607 days.

The neural network used was trained by Weigend et al [1996] independent of the decision tree constructed by Graven and Shavlik [1997]. The net-

work is composed of 69 input units, a single hidden layer of 15 units, and 3 output units.

The **input units** represent two types of information:

- Information derived from the time series itself such as a relative strength index, skewness, point and figure chart indicators, etc. (12 inputs).
- Fundamental information beyond the series itself such as indicators dependent on exchange rates between different countries, interest rates, stock indices, currency futures, etc. (57 inputs).

Three **output units** reflect a forecast of the exchange rate and two characteristics representing **turning points** in the exchange rate, specifically:

- a normalized version of the return (the logarithm of the ratio of tomorrow's price to today's price divided by the standard deviation computed over the last 10 trading days),
- the number of days to the next turning point where the exchange rate will reverse direction,
- the return between today and the next turning point.

This task was converted into the classification task of predicting whether the current day's price is **going up or down**. The network was trained using the technique of “cleaning” [Weigend et al., 1996]. The obtained cleaned network contained connections to 15 of the real valued attributes and five of the discrete valued attributes from original the 69 attributes. In computational experiments, two sets of attributes were involved:

- the 20 attributes taken from the cleaned network and
- the entire set of 69 attributes.

The **cleanning method** involves simultaneously cleaning the data and learning the underlying structure of the data. Specifically, cleaning uses a **cost function** that consists of two terms:

$$C = 0.5\eta(y-y^d)^2 + 0.5k(x-x^d)^2$$

The first term, $0.5\eta(y-y^d)^2$, measures the **cost of learning** by weighting the squared deviation between the network's output y and the target value y^d . The second term, $0.5k(x-x^d)^2$, measures the **cost of cleaning** by squaring the deviation between the cleaned input x and the actual input x^d . The parameters η and k are the learning rate and the cleaning rate, respectively.

The cleaning technique assumes that both inputs and outputs are corrupted by noise. Correcting the inputs and outputs can **suppress the noise** level and help to find a simpler regularity by training a network. A simpler regularity suffers less from over fitting the output, therefore the found regularity can be more reliable for out-of-sample forecasting.

Algorithm specifics. Trepan uses a given validation test set to measure the closeness of the decision tree output to the neural network output, this is

called **fidelity**. Trepan generates the tree with the highest value of fidelity to the target network. This is a fundamental property of the approach. If the network does not catch the **essence** of the underlying financial process, the same will happen with an extracted decision tree.

For the `min_sample` threshold (see Section 3.3.2), values of 1000, 5000, and 10000 are tried. Let us remember that the training set consists only of 1607 days. There is a big difference in training the decision tree directly from 1607 actual samples and training with **10000 artificial examples** generated by the neural network.

3.4.2. Comparison of performance

Two methods for discovering decision trees directly from data were used for comparison with Trepan; namely, C4.5 [Quinlan, 1993] and an enhanced version of ID2-of-3 [Murphy & Pazzani, 1991; Graven, Shavlik, 1997]. In addition to these methods, a **naïve prediction** algorithm was run for comparison with Trepan. This algorithm simply reproduces the current up/down direction as a forecast. Below we analyze experimental results reported by Graven and Shavlik. These results are summarized in Table 3.10.

The cleaned network provides the most accurate predictions and the m-of-n tree extracted from the neural network made similar predictions. The naïve rule and conventional decision trees produce the worst predictions.

In addition, Graven and Shavlik match Trepan's output and the output from the neural network. Trepan was able to reproduce the network's up/down output in about 80% of the cases, but only 60.6% of Trepan's outputs match to real up/down behavior of the exchange rate. On the other hand, methods extracting rules directly from data (C4.5, ID2-of-3+) did not reach this level of accuracy.

Table 3.10. Comparison of methods [Graven, Shavlik, 1997]

Method	Test-set accuracy (%)
Naïve rule (IFup THEN up, IF down THEN down)	52.8
Extracting conventional decision tree from data (C4.5)	52.8
Extracting m-of-n decision tree (ID2-of-3+)	59.3
Extracting m-of-n decision tree from neural network (Trepan)	60.6
Cleaned neural network	61.6

Table 3.10 shows that the accuracy of the naïve prediction, C4.5, and ID2-of-3+ ranges from 52.8% to 59.3%. This performance is similar to the average 55.6% for extracting a decision tree from data without use of a neural network obtained by Sipina_W for SP500 in Section 3.2.2.

It is important to note here that one can not expect that an extracted decision tree would outperform the underlying neural network significantly, be-

cause Trepan is designed to mimic the behavior of the network. Table 3.10 confirms this statement.

Figure 3.9 shows the tree extracted by Trepan from the cleaned neural network. This tree has one m-of-n test, and four ordinary splitting tests. Table 3.11 [Graven, Shavlik, 1997] provides brief descriptions of the attributes that are incorporated into the depicted tree. For example, v_7 and v_9 represent different functions of the Mark-Dollar exchange rate. Each leaf in figure 3.9 predicts the up/down direction of the exchange rate.

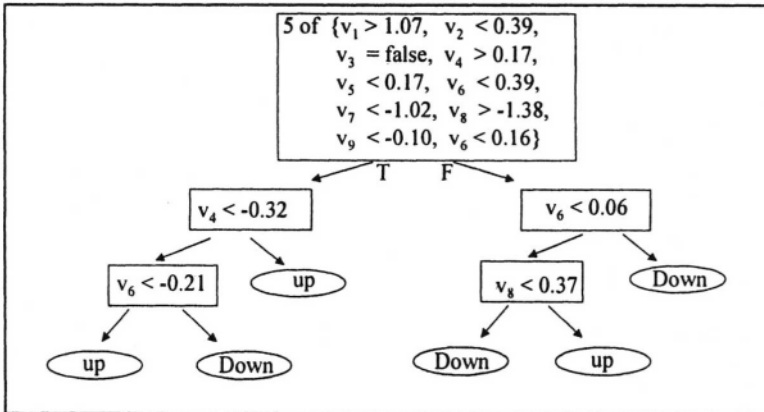


Figure 3.9. The Trepan tree [Graven, Shavlik, 1997]

The beginning node of the tree (the root) is an m-of-n node with $m=5$ and $n=10$. It means that any five of 10 properties (tests) listed in this node such as $v_1 > 1.07$, should be satisfied to get T (“true” output) and otherwise this node leads to F output.

Table 3.11. The attributes incorporated into the Trepan tree

Notation	Description
v_1	A comparison of the return on investment of investing Marks in the U.S. stock market versus investing Marks in the French stock market.
v_2	A measure of the Yen-Dollar exchange rate versus Yen futures.
v_3	A comparison of the Mark-Dollar exchange rate to the German interest rate.
v_4	A measures of the Mark-Dollar exchange rate versus Mark futures.
v_5	A measures of the Mark-Dollar exchange rate versus Mark futures.
v_6	A measure of the Swiss Franc-Dollar exchange rate versus Swiss Franc futures.
v_7	A function of the Mark-Dollar exchange rate.
v_8	A function of Deutsche Mark futures.
v_9	A function of the Mark-Dollar exchange rate.

The trees extracted from the neural networks are considerably simpler than trees extracted from data directly, although the latter trees were simplified. Based on these results, Graven and Shavlik concluded that the tree extracted from the neural network is **more comprehensible** than the trees learned from data directly. Remember that these authors measure comprehensibility of the decision tree by the **number of nodes**.

However, the single root node in Figure 3.9, which is a **compact 5-of-10 rule** extracted from the neural network, is equivalent to the set of all combination 5 of 10 tests, i.e., **252 conventional rules**. The conventional decision tree equivalent to this set of rules will not be smaller than decision trees discovered from data directly by C4.5 and ID2-of-3+ algorithms. For instance, C4.5 produced 103 internal nodes [Graven, Shavlik, 1997].

Graven and Shavlik report that the first node (5-of-10 rule) in the tree is able to mimic nearly 80% of the neural network's behavior and **successive nodes** added to the tree do not significantly improve this fidelity value. They express concern that Trepan possibly fails to explain the underlying regularity in the financial process discovered by the neural network. These observations raise the questions:

- Is this 5-of-10 rule really more comprehensible than the set of rules extracted from data directly?
- Does this 5-of-10 rule provide enough explanation of the underlying regularity in the financial process discovered by the neural network?

Conventional decision trees and sets of rules have an obvious advantage in comprehensibility. We think that this non-conventional 5-of-10 rule and other similar m-of-n rules need to be further analyzed to extract more **understandable, conventional IF-THEN rules**. Alternatively, appropriate rules can be selected from the already identified 256 conventional rules.

It was noted in [Graven, Shavlik, 1997] that Trepan discovered the overall behavior of the target in the regions that correspond to up or down trends. However, it was not able to catch all target fluctuations in the regions with small values of up and down. Apte and Hong (see Section 3.2.3) avoided this problem by putting a threshold of 6% for up/down in their trading strategy. Therefore, the trading strategy chosen can eliminate some problems of matching the neural network and the decision tree.

From our viewpoint, these experiments show that to this point m-of-n rule language has not helped uncover really **comprehensible** rules behind regularities discovered by neural networks in finance. The language of m-of-n rules is better understandable than the language of neural networks, if m and n are small. However, in the described experiments relatively large **m=5** and **n=10** were found. Therefore, the search for an appropriate language should continue. Meanwhile, these experiments show that a decision tree can be extracted from neural networks in finance with a close **approximation** to

the network. This can be valuable if the underlying neural network performs well.

3.5. Probabilistic rules and knowledge-based stochastic modeling

Learning methods can be classified into two groups:

- Data-based learning methods and
- Knowledge-based methods.

The **data-based learning methods** rely on training data as a major or sole source for discovering regularities. Decision trees, neural networks and other general-purpose learning methods belong to this class. **Knowledge-based methods** rely on training data and **prior** knowledge in variety of other forms, which is routinely ignored by the data-based methods. Ignoring other forms of available knowledge is one of the reasons it is difficult to interpret the **internal structure** of a classifier produced by the data-based methods. It may not have any correspondence to the real-world process that is generating the training data [Duetterich, 1997]. In this section we discuss one of the knowledge-based approaches -- **generative stochastic modeling** [Jensen, 1996, Castillo et al, 1997, Dietterich, 1997, D'Ambrosio, 1993, Jensen et al, 1990] also known as **causal modeling**.

The original idea of the approach was to represent a **causal mechanism** of generating the training data including the output. For example in physics, it would be a mechanism describing the energy level of particles (output) having as input (training data) the energy distribution of particles in prior moments. Outside of physics, it is very hard to produce a meaningful causal mechanism which actually generates training data. This is particularly true for finance. Therefore, producing a generative causal mechanism is usually replaced by a **less representative** generative mechanism without firm causal relations between components, but with **probabilistic dependencies**. Technically, it means the development of a **probabilistic network** – a network of nodes associated with conditional probability distributions between nodes.

This approach fits financial applications. Figure 3.10 shows an illustrative example of this type of generative mechanism (process) for stock direction forecast. Many useful probabilistic relations can be discovered between the blocks in this figure. In this example, the prior knowledge is a network of blocks with transition probabilities between them. For instance, the probability of stock going up today if yesterday stock went down -- $P(\text{up/down})$ for blocks F and G -- can be very useful for stock direction forecasting for tomorrow (block H).

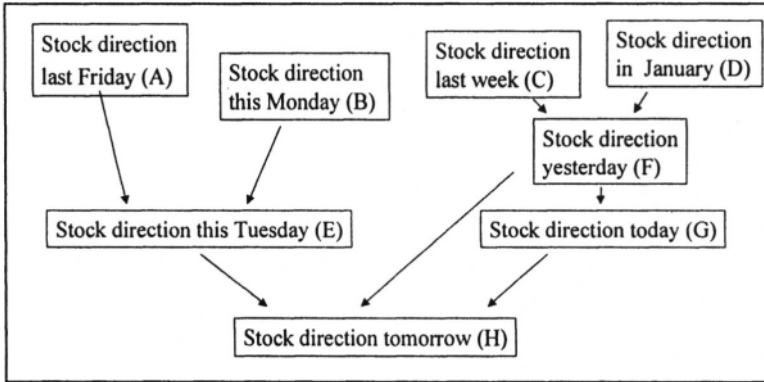


Figure 3.10. A probabilistic network for stock forecast

An important attribute of this prior knowledge is that it can be used for **different targets**. For instance, if tomorrow is Tuesday then the probabilistic relations between blocks A, B and E can be used. Below in section six we analyze the progress that has been achieved in implementing a knowledge-based approach in finance using Markov processes.

Solving learning problems uses a stochastic knowledge-based approach with four major steps [Dietterich, 1997]:

1. Designing the structure of the network,
2. Selecting the forms of the probability distributions,
3. Identifying parameters of the node probability distributions, and
4. Solving forecasting tasks using probabilistic inference with the stochastic model.

Training data are used for identifying parameters of the node probability distributions in step 3 after a user has accomplished steps 1 and 2. Finally, the learned network performs step 4. The most intuitive and informal step is designing the structure of the network. In finance, especially in stock market forecasting, this is a serious problem [Weigend, Shi, 1997, 1998]. Relational data mining methods discussed in Chapter 4 could be viewed as a **generic tool** for discovering a graphical structure.

3.5.1. Probabilistic Networks and Probabilistic Rules

In this section, we continue to illustrate the stochastic knowledge-based approach with the probabilistic network presented in figure 3.10. Eight variables (with their abbreviations from A to H) are depicted in this figure:

1. Stock direction last Friday (A),
2. Stock direction this Monday (B),

- 3. Stock direction last week (C),
- 4. Stock direction in January (D),
- 5. Stock direction this Tuesday (E),
- 6. Stock direction yesterday (F),
- 7. Stock direction today (G),
- 8. Stock direction tomorrow (H).

The variables 1-7 are used to estimate the probability of H - that the stock will go up tomorrow. The essence of the approach is to evaluate the joint probability $P(A,B,C,D,E,F,G,H)$ of all 8 variables as a function of a few simpler probability distributions like

$$P(A)*P(B) *P(E|A,B)*P(C)*P(D)*P(F|C,D)*P(G|F)*P(H|E,G,F)$$

Each smaller distribution is associated with a node of the network and relates the node with its predecessor.

Next, we need to represent each probability distribution using a small number of values to make the problem tractable. For instance, we can discretize stock direction into the values $\{<-6\%, -6 - 0\%, 0 - 6\%, >6\%\}$ or consider just {down, up}, coding them as 0 and 1.

Then Tables 3.12-3.14 could represent the probability distributions for those nodes. Each of these probabilities should be learned to get a **learned probabilistic network**.

Table 3.12. Two-dimensional up-down probability distribution table for node H

Stock direction yesterday (F)	Stock direction today G)	Probability of stock directions tomorrow $P(H F,G)$	
		Down	Up
Down	Up	P_{11}	P_{12}
Up	Up	P_{21}	P_{22}
Up	Down	P_{31}	P_{32}
Down	Down	P_{41}	P_{42}

Table 3.13. One-dimensional up-down probability distribution table for node F

Stock direction yesterday (F)	Probability of stock directions yesterday $P(F)$
Down	P_1
Up	P_2

Table 3.14. One-dimensional up-down probability distribution table for node G

Stock direction today (G)	Probability of stock directions today $P(G)$
Down	P_3
Up	P_4

Probabilistic information from table 3.12 has an equivalent representation in the form of the set of **propositional probabilistic rules**:

Rule PR1:**IF F="down" & G="down" THEN H="down"****with PROBABILITY $P(H="down"|F="down",G="down")=P_{41}$** **Rule PR2:****IF F="down" & G="down" THEN H="up"****with PROBABILITY $P(H="up"|F="down",G="down")=P_{42}$** **Rule PR3:****IF F="up" & G="down" THEN H="up"****with PROBABILITY $P(H="down"|F="up",G="down")=P_{31}$** **Rule PR4:****IF F="up" & G="down" THEN H="up"****with PROBABILITY $P(H="up",F="up",G="down")=P_{32}$**

These rules are prepositional because they do not include variables.

There are software systems, which support computations of probabilities in probabilistic networks such as Microsoft Belief Network (MSBN), see <http://research.microsoft.com/msbn>. The full set of probabilistic rules for F, G and H consists of eight rules. Given a set of training examples, this learning problem is very easy to solve. Each probability can be computed directly from the training data. For example, the cell $P(F="down")$ can be computed as the number of cases in the sample that had a downward stock direction. The value $P(H="down"|F="up", G="down")$ is the fraction of training examples with a downward stock direction on date t, an upward stock direction on date (t-1), and an upward stock direction on date (t-2) for each date t in the training data. Formally, using a sample, we can only get the maximum likelihood estimates of each of these probabilities. These estimates can be far away from the actual probability if there are very few examples. One practical approach is to interpolate probabilities for "adjacent" points using the assumption of monotonicity. For example, we might require that $P(H = "down \text{ more than } 6\%" | F = "up \text{ less than } 6\%", G = "down \text{ more than } 6\%")$ have a value similar to $P(H = "down \text{ more than } 6\%" | F = "up \text{ less than } 6\%", \text{down } 6\%", G = "down 6\%")$.

The final step is predicting whether stock will go up for a new case. Often technically this means that we need to find the conditional probability distribution $P(H|A,B,C,D,E,F,G)$ and to select the hypothesis H with the highest conditional probability for given A,B,C,D,E,F and G.

This can be done in advance (off-line) before getting A,B,C,D,E,F and G values. However, it will take a lot of storage space. An alternative way is to

get first A,B,C,D,E,F and G and then compute the only one row of the probability table (on-line computation). Several algorithms have been developed for off-line and on-line computation [Jensen, Lauritzen, and Olesen, 1990, D'Ambrosio, 1993; Diettrich, 1997].

3.5.2. The naïve Bayes classifier

The “naïve” Bayes classifier [Duda & Hart, 1973] exemplifies stochastic modeling from another viewpoint -- it makes assumptions about the probabilities. The major assumption is **independence** -- the attributes are generated independently according to their distributions $P(x_i|y)$ for each example.

Then real-valued attributes are discretized into a small number of values and the probability distributions are computed from the training data by using the fractions of examples in each class that take on a given attribute value. See tables shown in Section 3.5.1. For classes “down” (class code 0) and “up” (class code 1), the decision rule which classifies a new example (x,y) as follows:

$$class = \begin{cases} 0, & \text{if } P(y = 0 / x) > P(y = 1 / x) \\ 1, & \text{if } P(y = 1 / x) > P(y = 0 / x) \\ \text{no classification} & \text{otherwise} \end{cases}$$

By using **Bayes' rule** from probability theory,

$$P(y=0|x) = (P(x|y=0)P(y=0))/P(x)$$

$$P(y=1|x) = (P(x|y=1)P(y=1))/P(x),$$

one can write probabilistic propositional rules B0 and B1:

Rule B0: **IF x THEN y=0 with**
 PROBABILITY $(P(x|y=0) * P(y=0))/P(x)$

Rule B1: **IF x THEN y=1 with**
 PROBABILITY $(P(x|y=1) * P(y=1))/P(x)$

If input x is represented by several attributes x_i , i.e., $x=(x_1, x_2, \dots, x_n)$ the Bayesian classification decision rule should classify an example into class 0 if and only if

$$[P(x_1|y=0) * P(x_2|y=0) * \dots * P(x_n|y=0)P(y=0)] / [P(x_1|y=1) * P(x_2|y=1) * \dots * P(x_n|y=1) * P(y=1)] > 1.$$

In this form, the rule means that

IF $(x_1, x_2, \dots, x_n)$ **THEN** $y=0$ with
PROBABILITY $P(x_1|y=0) * P(x_2|y=0) * \dots * P(x_n|y=0) * P(y=0),$

IF $(x_1, x_2, \dots, x_n)$ **THEN** $y=1$ with
PROBABILITY $P(x_1|y=1) * P(x_2|y=1) * \dots * P(x_n|y=1) * P(y=1).$

This leads to the final decision rule:

If the probability of rule B0 is greater than the probability of rule B1 then the forecast is 0 (“stock will go down”).

Several experiments have shown that the naïve Bayes model performed well in comparison with the popular decision tree algorithm C4.5 [Quinlan, 1993] on 28 benchmark tasks and was robust with respect to violations of the independence of attributes [Domingos, Pazzani, 1996, Diettrich, 1997].

Stochastic Bayes models are also important for **unsupervised learning**, where the goal is to classify examples without having a target value. This is also called **data clustering**.

3.5.3. The mixture of experts

Below we consider another set of stochastic models: the **Hierarchical Mixture of Experts (HME)** model, the **Hidden Markov Model (HMM)** and the **Dynamic Probabilistic Network (DPN)**.

The Hierarchical Mixture of Experts model assumes that several processes (“experts”) contribute to the final decision [Jordan & Jacobs, 1994]. For example, in financial applications, we might assume two underlying processes: bull market and bear market as the generating the processes. The particular trends “down-up-up”, “down-up-down” and “up-up-down” could be viewed as a mixture of “bear” and “bull” trends with the generating processes “down-down” and “up-up”. Trends that are more complex may require a more complex mixture.

The simplified version of the HME model, called a **switching gated model**, works as follows [Diettrich, 1997]:

1. Generate a training example $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i$ is a set of attributes and $\mathbf{y}_i$ is the target value.
2. Choose an “expert” $\mathbf{E}_i$ stochastically (depending on the value of $\mathbf{x}_i$);

3. Compute conditional probability $P(y|x, E_i)$, where E_i is an expert;
4. Combine experts according to their probability distribution:

$$P(y|x) = \sum_i g_i(x) p_i(y|x),$$

where $g_i(x)$ is the output of the gating network for expert E_i (weight assigned to the expert E_i) and $p_i(y|x)$ is the conditional probability distribution over the various output values y produced by the expert E_i .

5. Forecast y using the distribution of $P(y|x)$.

Experts can also be organized hierarchically, such model is called a **hierarchical mixture model**. An important limitation of this approach follows from the complexity of the identification of $P(y|x, E_i)$. Practically $P(y|x, E_i)$ is identified only under strong assumptions, which may not be relevant to a particular forecasting problem.

3.5.4. The hidden Markov model

Each node in the Markov network can be in different states S and transition between states is governed by the **Markovian law (property)**. This property sets relations between probabilities of transition from one state to another state:

The conditional probability of transition from a state S in one node of the network (a parent node, R) to another node (a child node, C) depends only on the parent node and does not depend upon a “grandparent” like node G ,

$$P(S(C)|S(R), S(G)) = P(S(C)|S(R)).$$

Below figure 3.11 shows a diagram modified from figure 3.10. This diagram satisfies Markovian property because all links to “grandparent” nodes are deleted and all connections are made time sequential.

Each block in figure 3.11 has a single input and produces a single output. For instance, block G is associated with: (1) F -- “stock direction yesterday”, (2) its current state G -- “stock direction today” and (3) H -- “stock direction tomorrow”, with a probability distributions for all these transitions. The stock direction tomorrow (H) can be viewed as an output for node G and also as the next state after G . Such a process can be repeated daily by updating the value of today’s state of the stock. This iterative process is called a **Markov process**, where each state of the stock is associated with a set of its outputs (next states) and transition probabilities to each output state.

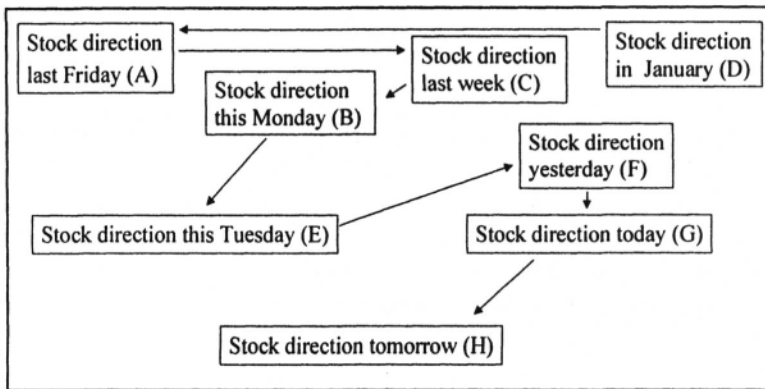


Figure 3.11. An illustrative probabilistic network for stock forecast

The problem of identifying (discovering) and explicitly describing states of such a **Markov probabilistic network** is challenging in financial applications. If states are not presented explicitly, they are called **hidden states**. For instance, the state of the market can be viewed as a hidden state, because we can not measure today directly such market parameters as investor's expectations and intentions. However, we are able to observe their consequences such as changes in prices and trade volume. A Markov model, which is able to operate with hidden states, is called a **Hidden Markov Model (HMM)**. In this model, each **hidden state** $s(t)$ is associated with an **observable output** $o(t)$, which is generated according to the conditional output probability distribution $P(o(t)|s(t))$. Similarly to conventional Markov model, HMM assumes that each state $s(t)$ moves to the next state $s(t+1)$ according to the **transition conditional probability** distribution is $P(s(t+1)|s(t))$. Beginning from some initial state $s(1)$ HMM will generate the **sequences of observable outputs** $o(1), o(2), \dots, o(n)$. Such observable sequences are considered as **training examples** and they can be of different length n . The set of possible values of observable outputs is called an **alphabet**. Each individual value o_i from the alphabet is called a **letter** and a sequence of letters is called a **word**. Similarly, a sequence of words is called a **sentence**. HMM are described in [Rabiner, 1989].

These terms came from the speech recognition [Rabiner, 1989; Dietterich, 1997], where the **alphabet** of letters consists of "frames" of the speech signal. Let W be the set of all words in the language. We are able to observe them, but the states generating them are hidden. Each word from W is modeled as a HMM.

There are two major steps in using HMM with given probability distributions $P(o(t)|s(t))$ and $P(s(t+1)|s(t))$ to recognize a spoken word:

- compute the **likelihood** that each of the word HMM's generated that spoken word w ,
- match the most likely word with the spoken word w .

Financial time series can be interpreted similarly. Actually, this idea has been already applied in finance, see Section 3.6 and [Weigend, Shi, 1997, 1998]. To be able to follow these steps a HMM should be learned using a set of training examples. Generating of training examples assumes that **variables** describing hidden states are identified first. Weigend and Shi [1997,1998] used unsupervised learning (clustering) to identify these variables. The general logic of HMM is show in Figure 3.12.

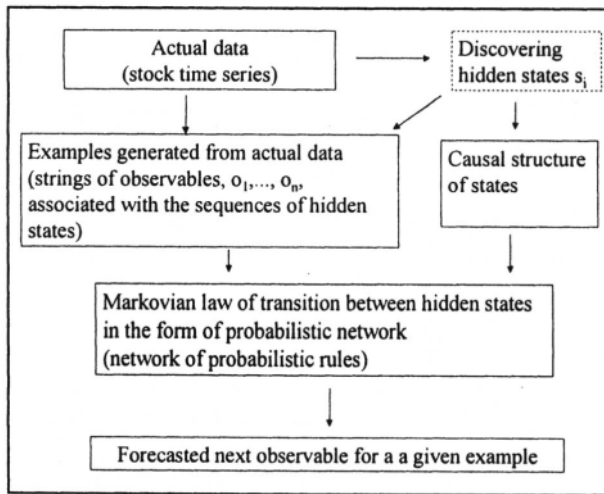


Figure 3.12. Hidden Markov Model diagram

The fitting networks with identified hidden variables is accomplished by several algorithms under different assumptions about statistical distributions. For instance, the **Expectation-Maximization** (EM) algorithm [Dempster et al, 1976] assumes the exponential family of distributions (binomial, multinomial, exponential, Poisson, and normal distributions, and many others) [Dietterich, 1997]. The EM algorithm is also called the Baum-Welch or Forward-Backward algorithm in HMM.

The EM algorithm consists of two steps E and M:

1. E-step -- adding to each training example statistics describing a sequence of states that probably generated the example.
2. M-step -- re-evaluating the probability distributions using the result of E-step.

For more detail see [Dempster et al, 1976; Weigend, Shi, 1998, Dietterich, 1997].

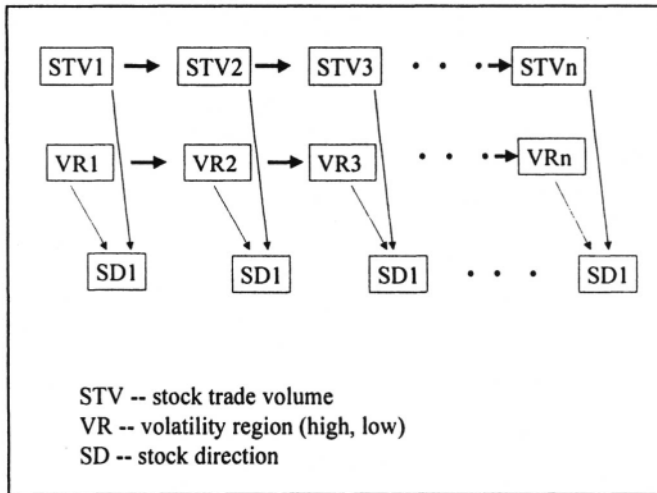


Figure 3.13. A simple dynamic probabilistic network for stock direction.

Dynamic probabilistic (stochastic) network. In the hidden Markov model, the number of values for the state variable at each point in time can become very large. Consequently, the number of parameters in the state transition probability $P(s_i | s_{i-1})$ can become intractably large. One solution is to represent the causal structure within each state by a model. For example, the **hidden state** can be represented by two separate relatively independent state variables: stock trade volume and volatility region (high, low). Figure 3.13 shows the resulting model. This type of models is reviewed in [Smyth et al, 1997]. They are known as a **dynamic probabilistic network** [DPN, Kanazawa et al, 1995], a **dynamic belief network** [DBN, Dean, Kanazawa, 1989], and a **factorial HMM** [Ghahramani, Jordan, 1996].

3.5.5. Uncertainty of the structure of stochastic models

In sections 3.5.2-3.5.4 we have discussed **learning the parameters** of a stochastic network with a given structure. It was assumed that nodes are known and their relations are established. However, there are many problems where these structural components are uncertain and should be learned from available data.

In table 3.15 we summarize some representative structure learning algorithms described in [Dietterich, 1977]. Other algorithms can be found in [Verma, Pearl, 1990; Spiegelhalter et al, 1993; Spirtes et al, 1993].

Table 3.15. Summary of algorithms for learning of the network structure

Structure of the network learned	Algorithm heuristics and assumptions	Algorithm procedure	Reference
Directed tree. Nodes --variables, Edges -- mutual information be- tween nodes	Chooses a root node arbitrarily	Polynomial algorithm starts with a complete graph, finds maximum weighted spanning tree.	Chow, Liu, 1968
Directed tree	Starts with a naive Bayes network	Algorithm called TAN is a modification of the Chow and Liu. Adds arcs to im- prove the posterior probab- ility of the network	Friedman, Goldszmidt 1996
Bayesian frame- work	All variables are independent, ob- served in the training data and are ordered by the user.	Algorithm called K2 evalu- ates the posterior probability of adding each possible sin- gle arc and makes the high- est-ranking addition.	Cooper, Herskovits, 1992
Bayesian frame- work	Starts with a prior probability distribu- tion of a prior net- work	Modification to the K2 to provide a good starting point using the prior network	Heckerman et al, 1995

This area is important for financial applications, but studies in this area have just begun [Weigend, Shi, 1997, 1998; Kovalerchuk, Vityaev, 1998, 1999]. See also sections 3.6.2 and 5.8. More experience has been gained in medical applications compared to what has been done in finance. In one of the experiments with the TAN algorithm, relatively **accurate** classifications with understandable arcs (directed network connections) were found. However, some of these arcs had **wrong directions** [Friedman and Goldszmidt, 1996, Dietterich, 1997]. From our viewpoint, this result clearly shows the difference between **learning understandable** and **interpretable structures**. The TAN algorithm produced an understandable, but not interpretable structure. This raises an important and **open question** -- how to learn successful, understandable and interpretable structures.

3.6. Knowledge-based stochastic modeling in finance

3.6.1. Markov chains in finance

Transition probabilities. Many well-known prediction methods used in stock market studies can be presented in the form of knowledge-based stochastic models. Below we show this for Markov chains. Two simple

Markov chains are presented in [Hiller, Lieberman, 1995]. Chain 1 has four states with conditional probabilities presented in table 3.16.

Table 3.16. Transition matrix for Markov chain 1

	The stock increases today	The stock decreases today
The stock increased yesterday	0.7	0.3
The stock decreased yesterday	0.5	0.5

In particular, table 3.16 gives the rule:

IF the stock increased yesterday
THEN the stock will increase today with probability 0.7.

Chain 2 has more states and is presented in table 3.17. This chain has a similar interpretation. For instance, table 3.17 gives the rule:

IF the stock increases today and decreased yesterday
THEN the stock will increase tomorrow with probability 0.6.

This rule is a combination of the bold cells in table 3.17 where the IF-part is taken from the first column. Similarly, the bold cell on the first row is used as the THEN-part of the rule. The probability 0.6 can be found in the intersection of the respective row and column.

Table 3.17. Transition matrix for Markov chain 2

	The stock increases both tomorrow and today	The stock increases tomorrow and decreases today	The stock decreases tomorrow and increases today	The stock decreases both tomorrow and today
The stock increased both today and yesterday	0.9	0	0.1	0
The stock increased today and decreased yesterday	0.6	0	0.4	0
The stock decreased today and increased yesterday	0	0.5	0	0.5
The stock decreased both today and yesterday	0	0.3	0	0.7

3.6.2. Hidden Markov models in finance

A model for **predicting the daily probability distribution** of SP500 returns is developed in [Weigend, Shi, 1997, 1998]. The goal of predicting a probability distribution is significantly different from the typical goal in finance -- **predicting the next value** of the time series. The probability distribution delivers a wider picture of the possible future of the stock market.

The full probability density function predictions are composed by a weighted superposition of the individual densities. These individual distributions are obtained using the Hidden Markov Model (HMM) method.

The resulting individual distributions have the following interpretation and scope:

- Distribution 1: Low volatility regions,
- Distribution 2: High volatility regions,
- Distribution 3: Collector of the outliers.

These distributions are identified using SP500 training data. Borders between three regions are vague and distributions are mixed in the border areas. Moreover, this mixture may be changed dynamically. Parameters of the dynamic mixture are also identified with training data. The training SP500 set contained data from 01.12.73 to 12.31.86 and the test set contained data from 01.02.87 to 12.29.94.

Weigend and Shi [1997] report that on these data simulated profit and Sharpe ratio are better for the Hidden Markov Model than for the benchmarks (neural network, linear regression model and simple buy-and-hold strategy) working without restoring a probability distribution.

Rule-based methodology for enhancing the HMM approach. This methodology assumes that each forecast is based on generating IF-THEN rules. Neural networks and many other learning techniques produce **forecast generators** without **IF-parts**. This creates significant **difficulties** in applying these methods for **non-homogeneous data** (e.g., data with high and low volatility areas) and blending them. A learned neural network is formally applicable to any combination of input values.

A **general probabilistic inference approach** suggests that:

- a set of forecast generators with IF-parts is learned;
- a probability distribution is associated with each individual generator and
- a blend of individual distributions for these generators is identified.

In HMM, forecast generators are called **experts**, in chapter 4, we call them **probabilistic laws**. Probabilistic laws based on first-order logic developed in the framework of this general probabilistic methodology are described in Chapters 4-6.

Chapter 4

Relational Data Mining (RDM)

To "be" means to be related.

Alfred Korzybski: Science and Sanity, 1933

4.1. Introduction

Data Mining methods map objects onto target values by discovered regularities in the data. These objects and mappings should be represented formally in some formal language. The selection of a language and a method for discovering regularities is a serious challenge. The uncertainty of problem description and method capability is among the most obvious difficulties in the process of selection.

Historically, methods based on **attribute-value languages (AVLs)** have been most popular in applications of learning algorithms. **Neural networks** (Chapter 2) and **decision trees** (Chapter 3) are typical examples of methods based on AVLs. They are relatively simple, efficient, and can handle noisy data. However, these methods have serious **limitations** in how they represent knowledge.

The purpose of **Inductive Logic Programming (ILP)** is to overcome these limitations. **ILP** learning systems naturally incorporate background knowledge and relations between objects into the learning process. Table 4.1 summarizes the advantages and disadvantages of AVL-based methods and first order logic ILP methods [Bratko, Muggleton, 1995]. First-order logic concepts are described in section 4.5.3. Before that we present examples and discuss these concepts informally. However, it may be useful for some readers to read section 4.5.3 now. Bratko and Muggleton [1995] pointed out that existing ILP systems are relatively inefficient and have rather **limited facilities for handling numerical data**.

Table 4.1. Comparison of AVL-based methods and first-order logic ILP methods

Method	Advantages for the learning process	Disadvantages for the learning process
Methods based on attribute-value languages	Simple , efficient, and handle noisy data .	Limited form of background knowledge . Lack of relations in the concept description language.
Inductive logic programming methods	Appropriate learning time with a large number of training examples . Solid theoretical basis (first-order logic, logic programming). Flexible Background knowledge, problem representation, and problem-specific constraints. Understandable representation of background knowledge, and relations between examples.	Inappropriate learning time with a large number of arguments in the relations. Weak facilities for processing numerical data .

The purpose of **Relational Data Mining (RDM)** is to overcome these limitations. We use this new term in parallel with the earlier term Inductive Logic Programming (ILP) to emphasize the goal -- discovering relations. The term ILP reflects the technique for discovering relations -- logic programming. In particular, discovering relational regularities can be done **without logical inference**. Therefore, we define **Relational Data Mining** as

Discovering hidden relations (general first-order logic relations) in numerical and symbolic data using background knowledge (domain theory).

Briefly, the advantages of RDM in comparison with other methods are presented in Chapter 1 using dimensions as suggested by Dhar and Stein [1997]. In some publications (e.g., [Mitchell, 1997]) background knowledge is called **domain knowledge (theory)**. In this chapter, we outline the traditional ILP approach and describe a **hybrid relational and probabilistic technique** that handles numerical data efficiently [Kovalerchuk, Vityaev, 1998; Vityaev et al, 1995; Vityaev, Moskvitin, 1993; Vityaev E., 1983]. This technique is called **MMDR (Machine Methods for Discovering Regularities)**. In Chapter 5, the MMDR method is applied to predict SP500C time series and to develop a trading strategy. This method outperformed several other strategies in simulated trading.

ILP systems have a mechanism to represent background financial knowledge in **human-readable and understandable form**. This is important for investors. Obviously, understandable rules have advantages over a stock market forecast without explanations.

If a learning system uses only a set of hypotheses (**hypothesis space H**) and **training examples** (TR) to find a hypothesis (regularity h) from H consistent with the training examples, then this learning system is called an **inductive learning system**. Examples consistent with hypothesis h , obtained by such a system, are called **induced examples**. If a learning system uses a set of hypotheses H and training examples TR along with **generalized background knowledge B**, then this system is called an **analytical (or explanation-based) learning system**. Examples consistent with background knowledge are said to be **explained** by that background knowledge. Thus, in an ideal world, the target value can be inferred from background knowledge for any training example.

However, in the real world, RDM should handle imperfect (noisy) data and in particular **imperfect numerical data**. This is one of the active topics of modern RDM research [e.g., Bratko, Muggleton, 1995; Kovalerchuk, Vityaev, 1998; Vityaev et al., 1995]. In Section 4.8, we describe the MMDR approach to address this problem.

The next open question for ILP is the computational complexity of ILP methods as noted in Table 4.1. In this chapter, we describe a way to handle this challenge using directed semantic inference. Connections between relational data mining and **relational databases** are also considered in this chapter. There are similarities as well as significant differences in interpretation of the term “**relational**” in these two techniques.

In practice, learning systems based on first-order representations have been successfully applied to many problems in chemistry, physics, medicine and other fields [Bratko et al., 1992, Muggleton et al., 1992 Muggleton, 1999; Bratko, 1993; Dzeroski et al., 1994; Kovalerchuk et al., 1997; Paz-zani, 1997]. Financial applications can specifically benefit from the RDM's predicate logic descriptions and from the background facility in RDM. Chapters 5 and 6 are devoted to these applications.

Dzeroski [1996], Bratko, Muggleton [1995], Muggleton [1999] and Paz-zani [1997] listed some major successful applications of ILP. It was stated in these publications that the results obtained with relational methods using real industrial or environmental data are better than with any other known approach, with or without machine learning. Such tasks as mesh design, mutagenicity, and river water quality exemplifies successful applications. It is especially important that domain specialists appreciate that the learned regularities are **understandable** directly in domain terms.

4.2. Examples

In this section, several examples illustrate the difference between relational and attribute-value languages used in Data Mining. In an attribute-value language, objects are described by tuples of attribute-value pairs, where each attribute represents some characteristic of the object, e.g., share price, volume, etc.

Neural networks and many other attribute-value learning systems have been used in financial forecasting for years (see chapter 3). Alternative attribute-value learning systems producing decision rules, such as decision trees (see chapter 3) also have been successfully applied in finance.

There are two types of attribute-value methods. The first one is based on numerical expressions and the second one is based on logical expressions and operations.

Neural networks and autoregression methods (chapter 2) exemplify the first type of methods based on **numerical representation** and DNF methods (chapter 3) exemplify the second type based on **logical expressions**.

Example 1.

Table 4.2 illustrates an attribute-value object representation. The first two lines represent some objects from a training data set. The last line represents an object without a value for the target attribute. The target value needs to be predicted for this object, i.e., stock price for the next day, 01.05.99. Each attribute-value pair can also be written as a name of an attribute and its value; for example, the first object can be written as follows:

```
<date, 01.02.99>;
<stock price on 01.02.99, $60.6>;
<volume of shares traded on 01.02.99, 1,000,000>;
<target--stock price on 01.03.99, $53.8>.
```

Table 4.2. Attribute-value object presentation

Attribute: date	Attribute 1: Stock price on date t	Attribute 2: Volume (number of shares) traded on date t	Attribute 3: Target-- stock price on date t+1
Value: 01.02.99	Value: \$ 60.6	Value: 1,000,000	\$53.8
Value: 01.02.99	Value: \$ 53.8	Value: 700,000	\$54.6
Value: 01.03.99	Value: \$ 54.6	Value: 800,000	\$56.3
Value: 01.04.99	Value: \$ 56.3	Value: 840,000	

For instance, the following **rule 1** can be extracted from table 4.2:

IF stock price today is more than \$60 and
trade volume today is greater than 900,000
THEN tomorrow stock will go down.

This rule can be written more formally:

IF StockPrice(t)>\$60 AND StockTradeVolume(t)> 900,000
THEN Greater(StockPrice(t+1), StockPrice(t))

Rule 2 is also true for table 4.2:

IF stock price today is greater than stock price yesterday and
trade volume today is greater than yesterday
THEN tomorrow stock price will go up.

Rule 2 also can be written more formally:

IF Greater(StockPrice(t), StockPrice(t-1)) AND
Greater(StockTradeVolume(t), StockTradeVolume(t-1))
THEN StockPrice(t+1)>StockPrice(t)

Note, actually rule 2 is true for table 4.2 because table 4.2 does not have examples contradicting this rule. However, table 4.2 has only one example (**t=01.03.99**) confirming this rule. Obviously, table 4.2 is too small to derive reliable rules. Table 4.2 and presented rules are used just for illustrating that **attribute-value methods can not discover rules 1 and 2** from table 4.2 directly. Both rules involve relations between **two objects** (records for two trading days t and **(t+1)**):

StockPrice(t+1)>StockPrice(t)
Greater(StockTradeVolume(t), StockTradeVolume(t-1)).

Special **preprocessing** is needed to **create additional attributes**, such as.

$$\text{StockUp}(t) = \begin{cases} 1, & \text{StockPrice}(t-1) < \text{StockPrice}(t) \\ 0, & \text{StockPrice}(t-1) \geq \text{StockPrice}(t) \end{cases}$$

There is a logical equivalency between attribute StockUp(t) and relation Greater(StockPrice(t), StockPrice(t-1)) used in rule 1:

StockUp(t) $\Leftrightarrow$ Greater(StockPrice(t), StockPrice(t-1)).

Similarly to be able to discover rule 2 with attribute-value methods we need an additional attribute:

VolumeUp(t) $\Leftrightarrow$ Greater(StockTradeVolume(t), StockTradeVolume(t-1))

Let us try to add relations like

Greater(StockTradeVolume(t), StockTradeVolume(t-i))

with 2, 3, ...i days ahead to the set of attributes. In this case, we need to generate many attributes such as Volume_iUp(t) similar to VolumeUp(t) used for one day ahead. In this way, a very small task can become huge. In logic terms attributes StockUp(t) and VolumeUp(t) are **monadic (unary) predicates** (Boolean functions with only one argument). In other words, languages of attribute-value methods are languages of functions of one variables (e.g., StockPrice(t)) and monadic predicates (e.g., StockUp(t)).

Functions of one variable are called **monadic (unary) functions**. The first order language formally described in section 4.4.3 differs from a propositional logic language mainly by the **presence of variables**. Therefore, a language of monadic functions and predicates is a first order logic language, but a very restricted language.

A language of monadic functions and predicates was not designed to represent relations that involve two, three or more objects. The domain (background) knowledge that can be used in the learning process of attribute-value methods is of a very restricted form. Moreover, other relations from a database cannot be used in the learning process if they are not incorporated into a single attribute-value table [Dzeroski, 1996].

Example 2.

There is a lot of confusion about the difference between logical attribute-value methods and relational methods. At first glance, they do the same thing -- produce "IF-Then" rules and use logical expressions. Dzeroski [1995] presented a fragment of a relational database for the potential customers of an enterprise to illustrate the difference. Table 4.3 presents a similar fragment of a relational database for corporate credit card holders. We wish to discover patterns in this table useful for distinguishing potential new cardholders from those who are not.

An attribute-value learning system may use Age, Sex and the Number of supervised associates from table 4.3. In this way, the following two patterns could be discovered with **monadic functions** Num_of_Supervised(Person) and Potential-Cardholder(Person):

Rule 1:
IF Num_of_Supervised(Person) ≥ 100
THEN Corporate_Cardholder(Person)

Rule 2:
IF Sex(Person)=F AND Age(Person) ≥ 38
THEN Corporate_Cardholder(Person)

Table 4.3. Database Relation “Potential-Corporate-Credit-Cardholder” (Attribute-value table)

Person	Age	Sex	Number of supervised associates	Corporate card-holder
Diana Right	39	F	10	Yes
Carol Peterson	49	F	1000	Yes
Barbara Walker	24	F	20	No
Cindy Peck	47	F	20	Yes
Peter Cooper	35	M	100	Yes
Stephen Baker	54	M	200	Yes

Table 4.4. Database relation “Colleague-of” (Attribute-value table)

Person (CEO)	Colleague (CFO)
Peter Cooper	Diana Right
Stephen Baker	Cindy Peck

Using a first-order language with a two-argument predicate Colleague-of(person, colleague) the following pattern can found:

Rule 3:
IF Colleague-Of(Person, Colleague) AND
Corporate_Cardholder(Person)
THEN Corporate_Cardholder(Colleague).

The last rule is much more meaningful, than the first two formal rules. Rules 1 and 2 are discovered in an isolated file (table 4.3), but rule 3 is discovered using two files simultaneously. Table 4.3 represents a **single relation** in relational database terms. Table 4.4 represents another single relation. To find regularity involving **records from both tables** we need to use more expressive **first-order language**. Mathematically, first-order languages generate such relations with two, three and more **variables**.

Example 3. This example is adapted from [Mitchell, 1997], where it is noted that relational assertions can be **conveniently expressed** using first-order representations, while they are **very difficult** to describe using propositional representations. Propositional representation requires expressing

rules only using constants. Mitchell suggests considering well-known rule learners such as AQ [Michalski, 1969] and CN2 [Clark, Niblet, 1989] as generators of **propositional rules**. More generally, the **decision tree method** is considered a propositional method [Mitchell, 1997, pp. 275, 279, 283]. Our opinion is that for many applications the decision tree method should be viewed as a **special type** of first-order logic method, restricted with monadic (unary) functions and predicates as we already have shown in example 1. Below, we show this for Mitchell's example. Let $\{(x,y)\}$ be a training set and $\text{Father}(x,y)$ is the relation "x is the father of y". For each pair of individuals it is known if $\text{Father}(x,y)$ is true. Similarly, we have truth-values for "y is a female" ($\text{Female}(y)$) and "y is the daughter of x" ($\text{Daughter}(x,y)$).

An algorithm using **first-order representations** could learn the following general rule from these training data:

IF $\text{Father}(x,y) \ \& \ \text{Female}(y)$ THEN $\text{Daughter}(x,y)$,

where x and y are variables that can be bound to any person. To do the same using a propositional rule learner, the data should be represented as shown in table 4.5.

**IF ($\text{Father}_1=\text{Bob}$) & ($\text{Name}_2=\text{Bob}$) & ($\text{Female}_1=\text{False}$)
THEN $\text{Daughter}_{1,2}=\text{True}$.**

Although it is correct, this rule is so specific that it will rarely, if ever, be useful in classifying future pairs of people [Mitchell, 1977]. First-order logic rules have an advantage in discovering relational assertions because they capture relations directly, e.g., $\text{Father}(x,y)$.

Table 4.5 allows us to discover a first-order logic rule with monadic predicates:

IF ($\text{Father}_1(x)=\text{Name}_2(x)$) & ($\text{Female}_1=\text{True}$) THEN $\text{Daughter}_{1,2}(x)=\text{True}$.

Each row in Table 4.5 is viewed as a variable x, which represents information about two persons. Indexes 1 and 2 indicate these persons. This is not a very economical or natural way to store data, but it allows one to discover a rule that is more useful than the propositional rule shown above. Therefore, we believe that attribute-based logical methods should not be reduced to propositional rules. These methods have restricted expressive power, but they are able to discover monadic first-order rules.

Table 4.5. Illustrative data for rule discovery

Name ₁	Fa-ther ₁	Mother ₁	Fe-male ₁	Name ₂	Fa-ther ₂	Mother ₂	Fe-male ₂	Daugh-ter ₁₂
Sharon	Bob	Louise	True	Bob	Victor	Nora	True	True
Bob	Victor	Nora	False					

First-order rules allow one to express naturally other more general hypotheses, not only the relation between pairs of attributes. In particular, in Chapter 5 we consider relations between three attributes related to SP500. Moreover, these rules are able to capture Markov chain types of models, which are used for financial time series forecasting.

4.3. Relational data mining paradigm

As we discussed in the previous section, attribute-value languages are quite restrictive in inducing relations between different objects explicitly. Therefore, richer languages were proposed to express relations between objects and to operate with objects more complex than a single tuple of attributes. Lists, sets, graphs and composite types exemplify complex objects.

These more expressive languages belong to the class of first order logic languages (see for definitions section 4.4.3 and [Russell, Norvig, 1995; Dzeroski, 1996; Mitchell, 1977]). These languages support **variables**, **relations**, and **complex expressions**. As we already mentioned, relational data mining is based on first-order logic.

ILP can discover regularities using several tables (relations) in a database, such as Tables 4.3 and 4.4, but the **propositional approach** requires creating a **single table**, called a **universal relation** (relation in the sense of relational databases) [Dzeroski [1995]. The following fields form the universal relation in example 2 presented above:

Age, Sex, Num_of_Supervised, Corporate_cardholder

along with

Colleague_Age, Colleague_Sex, Colleague_Num_of_Supervised, and Colleague_Corporate_cardholder.

Thus, features of a colleague are included in the list of features of a person. Table 4.6 illustrates this combined (universal) relation. Table 4.6 is larger than Tables 4.2 and 4.3 together. One of the reasons is that the first and the last lines actually present the same information, just in reverse order. The universal relation can be very large and, therefore, inconvenient and

impractical [Dzeroski [1995]. First-order logic rules have an advantage in discovering relational assertions because they capture relations directly.

Table 4.6. Combined (universal) database relation (Attribute-value table)

Diana Right	39	F	10	Y	Peter Cooper	35	M	100	Y
Carol Peterson	49	F	1000	Y					
Barbara Walker	24	F	20	No					
Cindy Peck	47	F	20	Y	Stephen Baker	54	M	200	Y
Stephen Baker	54	M	200	Y	Cindy Peck	47	F	20	Y
Peter Cooper	35	M	100	Y	Diana Right	39	F	10	Y

The typical Inductive Logic Programming task is a **classification** task [Dzeroski, 1996]] with **background knowledge B** expressed as:

- a set of predicate definitions and properties,
- positive examples **E⁺** for some class (pattern) and,
- negative examples **E⁻** for the same class (pattern).

Using this background knowledge an ILP system will construct a predicate logic formula H such that:

*All the examples in **E⁺** can be logically derived from B and H, and no negative example in **E⁻** can be logically derived from B and H. Formula H expresses some **regularity** that existed in the background knowledge. This formula could be discovered by ILP methods. ILP assumes a predicate logic representation of B and H.*

In the example above, records like Corporate_Cardholder(Diana Right)=Yes form the set of positive examples for the pattern (class) “potential customer” and records like Corporate_Cardholder(Barbara Walker)=No form the set of negative examples for this pattern. Usually background knowledge B, formula H, and positive and negative examples **E⁺** and **E⁻** are all written as programs in the Prolog programming language. Prolog was designed to support rich logical expressions, relations, and inference. This language differs significantly from common programming languages like C++ and Java. A search for regularity H is organized using logical inference implemented in Prolog. Convenience of logical inference in Prolog also has a drawback. It may require a lot of memory and disk space for storing examples and background knowledge. Examples E and background knowledge B can be represented in different ways, from very compact to very space

consuming. Therefore, a search for regularity H can be complex and ineffective.

The application of ILP involves **three steps**:

- development of an effective **representation** of the examples
- development of relevant **background knowledge**, and
- use of a general purpose **ILP system**.

There is a practical need for generalizing the goal. Relational data mining methods should be able to solve numerical **and interval forecasting tasks** along with classification tasks such as presented above. This requires modifying the concept of training examples E^+ and E^- and modifying the concept of deriving (inferring) training examples from background knowledge and a predicate. Section 4.8 presents a relational algorithm, called MMDR, which is able to solve numerical and interval forecasting tasks. This algorithm operates with a set of training examples E . Each example is amended with a target value like is done in Table 4.2, where attribute #3 is a target attribute--stock price for the next day. This is a numerical value. There is no need for MMDR to make this target discrete to get a classification task. Therefore, more generally, a **relational data mining (RDM)** mechanism is designed for forecasting tasks, including classification, interval and numerical forecasting. Similar to the definition given for classification tasks, for general RDM **background knowledge B** is expressed as:

- a set of predicate definitions,
- training examples E expanded with target values T (nominal or numeric), and
- set of **hypotheses $\{G_k\}$** expressed in terms of predicate definitions.

Using this background knowledge a RDM system will construct a set of predicate logic formulas $\{H_i\}$ such that:

The target forecast for all the examples in E can be **logically derived** from B and the appropriate H_i , i.e., from B and H_i .

Example 4. Let us consider Rule 2 discovered from Table 4.2 (see Example 1, Section 4.2).

```
IF Greater(StockPrice(t), StockPrice(t-1)) AND
Greater(StockTradeVolume(t), StockTradeVolume(t-1))
THEN StockPrice(t+1)>StockPrice(t)
```

This rule represents logical formula H_2 , and table 4.2 represents training examples E . These two sources allow us to derive the following logically for date $(t+1)=(01.04.99)$:

StockPrice(01.04.99)>54.6

(1)

assuming that $t=(01.03.99)$. This is consistent with actual Stock-Price(01.04.99)=56.3 for date 01.03.99. Rule 1 from the same Example 1 in Section 4.1.2 represents logical formula H_1 , but this rule is not applicable to $t=(01.04.99)$. In addition, other rules can be discovered from Table 4.2. For instance,

IF StockPrice(t)<\$60 AND StockTradeVolume(t)< \$90000
THEN Greater(\$60,StockPrice(t+1))

This rule allows us to infer

StockPrice(01.04.99)< 60 (2)

Combining (1) and (2) we obtain

60>StockPrice(01.04.99)>54.6 (3)

With more data we can narrow the interval (54.6, 60) for $t=(01.04.99)$. A similar logical inference mechanism can be applied for $t=(01.04.99)$ to produce a forecast for $(t+1)=(01.05.99)$.

The next generalization of relational data mining methods should handle (1) **classification**, (2) **interval** and (3) **numerical forecasting tasks with noise**. This is especially important in financial applications with numerical data and a high level of noise.

Hybridizing the pure logical RDM with a probabilistic approach (“probabilistic laws”) is a promising direction here. This is done by introducing probabilities over logical formulas [Carnap, 1962; Fenstad, 1967; Vityaev E. 1983; Halpern, 1990, Vityaev, Moskvitin, 1993; Muggleton, 1994, Vityaev et al, 1995; Kovalerchuk, Vityaev, 1998].

In contrast with the deterministic approach, in **Hybrid Probabilistic Relational Data Mining** background knowledge B is expressed as:

- A set of predicate definitions,
- Training examples E expanded with target values (nominal or numeric), and
- A set of **probabilistic hypotheses** $\{G_k\}$ expressed in terms of predicate definitions.

Using this background knowledge a system constructs a set of predicate logic formulas $\{H_i\}$ such that:

Any example in E is derived from B and the appropriate H_i probabilistically, i.e., statistically significantly.

Applying this approach to (3)

$60 > \text{StockPrice}(01.04.99) > 54.6,$

we may conclude that although this inequality is true and is derived from table 4.2 it is not a statistically significant conclusion. It may be a property of a training sample, which is too small. Therefore, it is risky to rely on statistically insignificant forecasting rules to derive this inequality.

The MMDR method (section 4.8) is one of the few Hybrid Probabilistic Relational Data Mining methods developed [Muggleton, 1994, Vityaev et al, 1995; Kovalerchuk, Vityaev, 1998] and probably the only one which has been applied to financial data.

4.4. Challenges and obstacles in relational data mining

One of the major obstacles to more effective use of the ILP methods is their limited facility for handling **numerical data** [Bratko Muggleton, 1995] (see table 4.1).

There are two types of numerical data in data mining:

- Numerical attributes used to describe objects and discover patterns.
- The numerical target variable and

Traditionally ILP solves only classification tasks without direct operations on numerical data. The MMDR method (section 4.8) handles an interval forecast of numeric variables with continuous values like prices along with solving classification tasks. In addition, MMDR handles **numerical time series** using the first-order logic technique, which is not typical for ILP applications. Historically, ILP was a pure **deterministic logic technique**, which originated in logic programming. There are well-known problems with deterministic methods--handling data with a significant level of **noise**. This is especially important for financial data, which typically have a very high level of noise. The MMDR method addresses this issue by introducing **probabilistic first-order rules**.

Statistical significance is another challenge for deterministic methods. Statistically significant rules have an advantage in comparison with rules tested only for their performance on training and test data [Mitchell, 1997]. Training and testing data can be too limited and/or not representative. If rules rely only on them then there are more chances that these rules will not deliver a correct forecast on other data. This is a hard problem for any data mining method and especially for deterministic methods like ILP. We address this problem in Section 4.8, developing rules tested on their statistical significance. Intensive studies are being conducted for incorporating a probabilistic mechanism into ILP [Muggleton, 1994].

Knowledge Representation is an important and informal initial step in relational data mining. In attribute-based methods, the attribute form of data actually dictates the form of knowledge representation. Relational data mining has many more options for knowledge representation. For example, attribute-based stock market information such as stock prices, indexes, and volume of trading should be transformed into the first-order logic form. This knowledge includes much more than only values of attributes. There are many ways to represent knowledge in the first order logic language. One of them can skip **important information**; another one can hide it. Therefore, data mining algorithms may work too long to “dig” relevant information or even may produce inappropriate rules. Introducing **data types** [Flash et al., 1998] and concepts of **representative measurement theory** [Krantz et al., 1971, 1989, 1990, Narens, 1985; Pfanzagl, 1968] (see section 4.9) into the knowledge representation process helps to address this representation problem.

It is well known that the general problem of rule generating and testing is **NP-complete** [Hyafil, Rivest, 1976]. Therefore, the discussion above is closely related to the following questions. What determines the number of rules? When do we stop generating rules? In Section 4.8.2, we discuss this issue.

The number of hypotheses is another important parameter. It has already been mentioned that RDM with first-order rules allows one to express naturally a large variety of general hypotheses, not only the relation between pairs of attributes. These more general rules can be used for classification problems as well as for an interval forecast of a continuous variable. RDM algorithms face exponential growth in the number of combinations of predicates to be tested. A mechanism to decrease this set of combinations is needed. Section 4.9 addresses this issue using a higher-level language, a data type system and the representative measurement theory approach. Type systems and measurement theory approaches provide better ways to generate only **meaningful hypotheses** using syntactic information.

A probabilistic approach also naturally addresses knowledge discovery in situations with **incomplete or incorrect domain** knowledge. Properties of individual examples are not generalized beyond the limits of statistically significant rules.

Predicate Development is needed for relational data mining. To utilize advantages of human-readable forecasting rules produced in relational data mining, **logical relations (predicates)** should be developed for financial problems. These predicates should be **interpretable** in ordinary financial terms like stock prices, interest rates, trading days, and so on. In this way, relational methods produce valuable understandable rules in addition to the forecast. A financial specialist can **evaluate the performance of the fore-**

cast as well as a forecasting rule. The problem of inventing predicates is addressed in Chapter 5.

4.5. Theory of RDM

4.5.1. Data types in relational data mining

A **data type** (type for short) in modern object-oriented programming (OOP) languages is a rich data structure, $\langle A, P, F \rangle$. It consists of **elements** $A = \{a_1, a_2, \dots, a_n\}$, **relations** between elements (**predicates**) $P = \{P_1, P_2, \dots, P_m\}$ and meaningful **operations** with elements $F = \{F_1, F_2, \dots, F_k\}$. Operations may include two, three or more elements, e.g., $c = a \# b$, where $\#$ is an operation on elements a and b producing element c . This definition of data type formalizes the concept of a **single-level data type**. For instance, a single-level graph structure (“stock price” data type) can be created with nodes reflecting individual stock prices and edges reflecting relations between stock prices ($<, =, >$). These graph structures (values of the data type) can be produced for each trading day -- $StPr(1), StPr(2), \dots, StPr(t)$ -- generating a time series of graph structures. A **multilevel data type** can be defined by considering each element a_i from A as a composite data structure (data type) instead of as an atom. To introduce a multilevel stock price data type, stocks are grouped into categories such as high-tech, banking and so on. Then relations ($<, =, >$) between the average prices of these groups are defined. Traditional attribute-value languages operate with much simpler single-level data types.

Implicitly, **each attribute** in attribute-value languages reflects a **type**, which can take a number of possible values. These values are elements of A . For instance, attribute “date” has 365 (366) elements from 01.01.99 to 12.31.99. There are several meaningful relations and operations with dates: $<, =, >$, and $middle(a, b)$. For instance, the operation $middle(a, b)$ produces the middle date $c = 01.05.99$ for inputs $a = 01.03.99$ and $b = 01.07.99$. It is common in attribute-value languages that a data type such as a date is given as an **implicit data type**. (see example 5). Usually in AVLs, relations P and operations F are not expressed explicitly. However, such data types can be embedded **explicitly** into **attribute-value languages**.

Example 5. Let us consider data type “trading weekdays”, where a set of elements A consists of $\{Mon, Tue, Wed, Thu, Fri\}$. We may code these days as $\{1, 2, 3, 4, 5\}$ and introduce a distance $\rho(a, b) = |a - b|$ between them using these numeric codes. For instance,

$$\rho(Mon, Tue) = \rho(1, 2) = |1 - 2| = 1 \text{ and } \rho(Fri, Mon) = \rho(5, 1) = |5 - 1| = 4.$$

The last distance is natural if both Friday and Monday belong to **the same week**, but if Monday belongs to the next week it would be more reasonable to assign $\rho(\text{Fri}, \text{Mon})=1$, because Monday is the next trading day after Friday. This is a property of **cyclical scales**. Different properties of cyclical scales are studied in representative measurement theory [Krantz, et al. 1971, 1979, 1980]. The “trading weekdays” data type is a **cyclical data type**. This distance has several properties which are unusual for distances. For instance, it is possible that

$$\rho(a, b) \neq \rho(b, a),$$

Let us assume that weekday *a* always precedes weekday *b*. Under this assumption $\rho(\text{Fri}, \text{Mon})$ means a distance between current Friday and Monday next week, but $\rho(\text{Mon}, \text{Fri})$ means a distance between Mon and Fri during the same week. In this example the requirement that *a* precedes *b* was not defined explicitly. In [Kovalerchuk, 1975, 1976] we studied cyclical scales and suggested numeric and binary coding schemes preserving this property for a variety of cyclical scales.

In AVLs each attribute is a type and any object is a sequence of values of these attributes. This is the base for efficient data mining in simple data represented in an AVL. On the other hand, as Flach et al [1998] noted, the Prolog language alleviates many of the **limitations** of attribute-value languages related to data structure.

However, the traditional application of Prolog within ILP has also caused the loss of one critical element inherent in attribute-value languages: the **notion of type**. The **Prolog** language used in ILP has **no type system**. All characteristics of objects are captured by predicates. Therefore, a number of **ad hoc mechanisms** (e.g., linked clauses, mode declarations, determinacy, etc) are used to constrain a search.

A new **strongly typed programming language Escher** was developed to meet this challenge [Flach et al, 1998]. The Escher language is an important tool, which allows users to incorporate a variety of explicit data types developed in representative measurement theory into the programming environment. On the other hand, RDM can be successfully implemented using common languages like Pascal and C ++ [Vityaev, Moskvitin, 1993; Vityaev et al, 1995].

4.5.2. Relational representation of examples.

Relational representation of examples is the key to relational data mining. If examples are already given in relational form, relational methods can be applied directly. For attribute-based examples, this is not the case. We need to express **attribute-based examples** and their **data types** in relational

form. There are two major ways to express **attribute-based examples** using predicates:

- **generate predicates** for each value and
- **use projection functions**.

Table 4.7 presents an attribute-based data example for a stock.

Table 4.7. Attribute-based data example

Stock price, \$	Volume, x1000	Date	Weekday	Stock Event
54.6	3067.54	01.04.99	Monday	New product

Generating predicates for each value. To express stock price \$54.60 from Table 4.7 in predicate form, we may generate predicate $P546(x)$, such that **$P546(x)=true$** if and only if the stock price is equal to \$54.60. In this way, we would be forced to generate about 1000 predicates if prices are expressed from \$1 to \$100 with a \$0.10 step. In this case, the ILP problem will be intractable. Moreover, the stock price data type has not yet been presented with the $P546(x)$ predicate. Therefore, additional relations to express this data type should be introduced. For example, it can be a relation between predicates $P546(x)$ and $P478(x)$, expressing a property that stock price 54.6 is greater than 47.8.

To avoid this problem and to constrain the hypothesis language for RDM, the **projection function** was introduced [Flach et al, 1998]. This concept is described below.

Representation of background knowledge. ILP systems use two sorts of background knowledge: objects and relations between those objects. For example, objects are named by constants a,b,c and relations are expressed using these names -- **$P(a,b)=true$** and **$P(c,b)=false$** . Use of constants is not very helpful because normally names do not carry properties of objects useful for faster data mining. In the approach suggested in [Flach et al, 1998], this is avoided. An object is “named” by the collection of all of its characteristics (**terms**).

For instance, term representation of stock information on 01.03.1999 can be written as follows:

```

StockDate(w)=01.03.1999
& StockPrice(w)=$54.60
& StockVolume(w)=3,067,540
& StockWeekday(w)=Mon
& StockEvent(w)="new product".

```

Here StockPrice is a **projection function** which outputs stock price (value of StockPrice attribute).

Only naming of subterms is needed. This representation of objects (examples) is convenient for adding new information about an object (e.g., **data types**) and **localizing** information. For instance, subterm “StockEvent” permits one to localize such entities as reported profit, new products, competitor activity, and government activity.

In the example above the following **data types** are used:

- type weekday = {Mon, Tue, Wed, Thu, Fri},
- type price,
- type volume,
- type date,
- type event = {reported profit, new product, competitor’s activity, government activity,},
- type stock = {price, volume, date, weekday, event}.

Type event brings a description of event related to the stock, e.g., published three month profit, new product, competitor’s activity. This can be as a simple text file as a structured data type.

The representation of an example then becomes the **term**

Stock (54.6, 3067.54, 01.04.99, Mon, new product).

Notice that when using projection functions in addition to predicates it is possible, without the use of variables, to represent relational information such as the equality of the values of two attributes. E.g., projection function StockEvent together with the equality relation (**=**) are equivalent to predicate SameEvent(w,x):

SameEvent(w,x) ⇔ StockEvent(x) = StockEvent(w).

Thus, the distinction between different propositional and first-order learning tasks depends in part on the representation formalism.

Strongly typed languages. ILP systems use types to provide labels attached to logical variables. However, these are not the data type systems found in modern programming languages. All available literals in the Prolog language will be considered for inclusion if a naive refinement operator is used for Prolog [Flash et al, 1998]. These authors developed a new strongly typed ILP language, **Escher**, which employs a complex data type system and restricts the set of hypotheses by **ruling out many useless hypotheses**. The MMDR method (Section 4.8) employs another way to incorporate data types into data mining by adding a data type structure (relational system) into the background knowledge. Such a relational system is based on representative measurement theory (Section 4.10).

Complex data types and selector functions. Each data type is associated with a relational system, which includes:

- cardinality,
- permissible operations with data type elements, and
- permissible relations between data type elements.

In turn, each data type element may consist of its own subelements with their types. **Selector functions** [Flash et al, 1998] serve for extracting subterms from terms. Without selector functions, the internal structure of the type could not be accessed. Projection for selecting the i -th attribute requires the tuple type and a list of components (attributes) of the tuple. A list of components (attributes) requires the length of the list and the set of types of components.

The number of hypotheses. The most important feature of strongly typed languages is that they not only restrict possible values of variables, but more importantly **constrain the hypothesis language**.

Table 4.8 summarizes information about data type features supported by different languages: ordinary attribute-based languages, attribute-based languages with types, first-order logic languages with types and ILP languages based on Prolog. This table is based on analysis from [Flach et al, 1998],

Table 4.8. Data types supported by data mining languages

Supported features of object representation	Attribute-based language	Attribute-based Language with types	First-order language with types	ILP based on Prolog language
Formally expressed data type context	No	Yes	Yes	Yes
Attribute-value tuples	Yes	Yes	Yes	No
Explicitly induced relations between tuples	No	Yes	Yes	Yes
Data types of attributes expressed as in modern object-oriented programming languages	No	Yes	Yes	No
Mechanism to restrict the set of possible hypotheses using data types	No	Yes	Yes	No
Representing objects by terms using projection function	No	Yes	Yes	No

Strongly typed languages for numerical data are especially important for financial applications with prevailing numeric data.

Single Argument Constraints. Consider an example, the term $\text{stock}(A,B,C,D,E)$ has a type definition of

$\text{stock}(\text{price}, \text{volume}, \text{date}, \text{weekday}, \text{event}).$

Having this type definition, testing rules with arguments like

$(25.7, 90000, 01.04.99, 67.3, \text{new product})$

is avoided because 67.3 does not belong to weekday type. Thus, this typing information is a useful simple form of background knowledge. Algorithms FOCL (Section 4.6.3) and MMDR (Section 4.8) take advantage of typing information. On the other hand, the well-known FOIL algorithm (Section 4.6.2) does not use type constraints to eliminate literals from consideration.

Typing can be combined with **localized predicates** to **reduce the search space**. For instance, a localized relation $\text{Greater_dates}(A,B)$ can be introduced to compare only dates with type information $\text{Greater_dates}(\text{date}, \text{date})$ instead of a universal relation $\text{Greater}(\text{item}, \text{item})$. Similarly, a localized relation $\text{Greater_}\$(A,B)$, type information $\text{Greater_}\$(\text{price}, \text{price})$ can be introduced and applied for prices. This localized typing avoids the testing of some arguments (literals). For instance the localized predicate $\text{Greater_dates}(A, B)$ should not be tested for literals of types such as

$\text{Greater_dates}(\text{stockprice}, \text{stockprice}),$
 $\text{Greater_dates}(\text{stockprice}, \text{date}),$
 $\text{Greater_dates}(\text{date}, \text{stockprice})$

More generally, let $\{\mathbf{T}_i\}$ be the types of already used variables $\{\mathbf{x}_i\}$ in predicate P . Predicate P should be tested for different sequences of arguments. If the type $\mathbf{T}_i$ of the already used i -th argument of P contradicts the type of an argument $\mathbf{y}_i$ suggested for testing P , then the testing of the sequence which involves $\mathbf{y}_i$ can be eliminated. This is a correct procedure only if a predicate is **completely localized**, i.e., only one type of argument is allowed for $\mathbf{y}_i$. It is the case for the predicate Greater_dates , but it is not for the original predicate Greater defined for any items. This consideration shows that typing information **improves background knowledge** in two ways: (1) adding predicates and clauses about data types themselves and (2) refining and adding predicates and clauses about objects (examples). In such situations, typing can in the best case exponentially reduce the search space [Flach et al, 1998]. FOCL and FOIL algorithms (Section 4.6) illustrate the benefit of typing. FOCL algorithm tested 3240 units and 242,982 tuples using typing

as compared to 10,366 units and 820,030 tuples without typing. The task contained [Pazzani, Kibler, 1992]:

- learning a predicate with six variables of different types and
- 641 randomly selected training examples (233 positive and 408 negative training examples).

Typing is very useful for data mining tasks with limited training data, because it can improve the **accuracy of the hypothesis** produced without enlarging the data set. However, this **effect of typing is reduced as the number of examples increases** [Flash et al, 1998; Pazzani, Kibler, 1992].

Existential variables. Consider the hypothesis:

IF (there exists stock w such that StockEvent(x)=StockEvent(w))
 AND (Some other statement)
 THEN StockPrice(x)>StockPrice(w).

and

IF ($\exists w, z$ StockEvent(x)=StockEvent(w)=StockEvent(z))
 THEN StockPriceP(x)>StockPriceP(z).

The variables w and z are called **existential variables**. The **number of existential variables** like w and z provides one of the measurements of the **complexity of the learning task**. Usually the search for regularities with existential variables is a computational challenge.

4.5.3. First-order logic and rules

This chapter defines basic concepts of first order logic such as: predicates, functional expressions, terms, atoms, and quantifiers. More details and advanced issues are covered in [Russel and Norvig, 1995; Mitchell, 1997; Halpern, 1990; Krantz, Luce, Suppes and Tversky, 1971, 1989, 1990].

A **predicate** is defined as a binary function or a subset of a set $D=D_1 \times D_2 \times \dots \times D_n$, where D_1 can be a set of stock prices at moment **t=1** and D_2 can be stock price at moment **t=2** and so on. Predicates can be defined **extensionally**, as a list of tuples for which the predicate is true, or **intensionally**, as a set of (**Horn**) **clauses** for computing whether the predicate is true. Let stock(t) be a stock price at t, and consider the predicate

UpDown(stock(t), stock(t+1), stock(t+2)),

which is true if stock goes up from date t to date **t+1** and goes down from date **t+1** to date **t+2**. This predicate is presented extensionally in Table 4.9

Table 4.9. UpDown predicate

Stock(t)	Stock(t+1)	Stock(t+2)	UpDown(, ,)
\$34	\$38	\$35	True
\$38	\$35	\$35.50	False
\$35.50	\$36	\$34	True
\$36	\$37	\$38	False

and **intensionally** using two other predicates Up and Down:

$$\text{Up}(\text{stock}(t), \text{stock}(t+1)) \& \text{Down}(\text{stock}(t+1), \text{stock}(t+2)) \\ \rightarrow \text{UpDown}(\text{stock}(t), \text{stock}(t+1), \text{stock}(t+2)).$$

where $\text{Up}(\text{stock}(t), \text{stock}(t+1)) \Leftrightarrow \text{Stock}(t+1) \geq \text{Stock}(t)$, and

$$\text{Down}(\text{stock}(t), \text{stock}(t+1)) \Leftrightarrow \text{Stock}(t) \geq \text{Stock}(t+1).$$

Predicates Up and Down are given extensionally in Table 4.10.

Table 4.10. Predicates Up and Down

Stock(t)	Stock(t+1)	Up(,)	Down(,)
\$34	\$38	True	False
\$38	\$35.50	False	True
\$35.50	\$36	True	False
\$36	\$37	False	True

A **literal** is a **predicate** A or its **negation** ($\neg A$). The last one is called a **negative literal**. An unnegated predicate is called a **positive literal**. A **clause body** is a conjunction $A_1 \& A_2 \& \dots \& A_i$ of literals $A_1, A_2, \dots, A_i$. Often we will omit $\&$ operator and write $A_1 \& A_2 \& \dots \& A_i$ as $A_1 A_2 \dots A_i$.

A **Horn clause** consists of two components: a **clause head** (A_0) and a **clause body** ($A_1 A_2 \dots A_i \dots A_t$). A clause head, A_0 , is defined as a single predicate. A Horn clause is written in two equivalent forms:

$$A_0 \leftarrow A_1 A_2 \dots A_i \dots A_t, \text{ or } A_1 A_2 \dots A_i \dots A_t \rightarrow A_0,$$

where each A_i is a literal. The second form is traditional for mathematical logic and the first form is more common in applications.

A **collection** of Horn clauses with the same head A_0 is called a **rule**. The collection can consist of a single Horn clause; therefore, a **single Horn clause** is also called a rule. Mathematically the term collection is equivalent to the OR operator ($\vee$), therefore the rule with two bodies $A_1 A_2 \dots A_i$ and $B_1 B_2 \dots B_t$ can be written as

$$A_0 \leftarrow (A_1 A_2 \dots A_i \vee B_1 B_2 \dots B_t)$$

A **k-tuple**, a **functional expression**, and a **term** are the next concepts used in relational approach. A finite sequence of k constants, denoted by $\langle a_1, \dots, a_k \rangle$ is called a **k-tuple** of constants. A function applied to k-tuples is called a **functional expression**. A **term** is

- a constant,
- functional expression.
- variable or

Examples of terms are given in table 4.11.

Table 4.11. Examples of terms

Expression	Comment	Term ?
x	Variable --stock x	Yes
MSFT	Constant (specific stock/index)	Yes
StockPrice(x)	Functional expression	Yes
TradeVolume(x)	Functional expression	Yes
StockPrice(x)*TradeVolume(x)	Functional expression	Yes
Nasdaq(x)>StockPrice(x)	Incorrect	No
NASDAQ(x)	Predicate, literal (Stock x is traded on NASDAQ)	No
StockPrice(x)>StockPrice(y)	Predicate(x,y), literal	No

A k-tuple of terms can be constructed as a sequence of k terms. These concepts are used to define the concept of atom. An **atom** is a predicate symbol applied to a k-tuple of terms. For example, a predicate symbol P can be applied to 2-tuple of terms (v,w), producing an atom P(v,w) of arity 2.

If P is predicate “>” (greater), $v = \text{StockPrice}(x)$ and $w = \text{StockPrice}(y)$ are two terms then they produce an atom:

$\text{StockPrice}(x) > \text{StockPrice}(y),$

that is, price of stock x is greater than price of stock y.

Predicate P uses two terms v and w as its arguments. The number two is the **arity** of this predicate. If a predicate or function has k arguments, the number k is called **arity** of the predicate or function symbol. By convention, **function and predicate symbols** are denoted by Name/Arity. Functions may have **variety of values**, but predicates may have only Boolean values **true and false**. The meaning of the rule for a k-arity predicate is the set of k-tuples that satisfy the predicate. A tuple satisfies a rule if it satisfies one of the Horn clauses that define the rule.

A **unary (monadic) predicate** is a predicate with arity 1. For example, NASDAQ(x) is unary predicate.

Quantifiers.

$\forall$ means “for all”. For example, $\forall x \text{ StockPrice}(x) \geq 0$ means that for all stocks, stock prices are non-negative.

$\exists$ means “there exists”. It is a way of stating the existence of some object in the world without explicitly identifying it. For example, $\exists x \text{ P}(\text{SP500}, y)$ means that

$$\exists x \text{ StockPrice}(\text{SP500}) \geq \text{StockPrice}(x),$$

i.e., there is stock x less expensive than SP500 for a given time.

Using the introduced notation, the following **clause** can be written:

$$\exists x (\text{StockPrice}(x) < \$100 \leftarrow \text{TradeVolume}(x) < 100,000)$$

This is a notation typical for Prolog language. As we already mentioned, more traditional logic notation uses the opposite sequence of expressions:

$$\exists x (\text{TradeVolume}(x) < 100,000 \rightarrow \text{StockPrice}(x) < \$100)$$

Both clauses are equivalent to the statement: “There is a stock such that if its trade volume is less than 100,000 per day than its price is less than \$100 per share. Combining two quantifiers, a more complex clause can be written:

$$\forall x \exists y (\text{StockPrice}(y) < 100 \leftarrow \text{TradeVolume}(x) < 100,000)$$

Predicates defined by a collection of examples are called **extensionally defined predicates**, and predicates defined by a rule are called **intensionally defined predicates**. If predicates defined by rules then inference based on these predicates can be explained in terms of these rules. Similarly, the **extensionally defined predicates** correspond to the **observable facts** (or the **operational predicates**) [Mitchell, Keller, & Kedar-Cabelli, 1986]. A collection of intensionally defined predicates is also called **domain knowledge** or **domain theory**.

Statements about a particular stock MSFT for a particular trading day can be written as:

StockPrice(MSFT) > 83,
 NASDAQ(MSFT),
 TradeVolume(MSFT) = 24,229,000.

These statements can be written in clause notation also assuming empty if-part:

```

StockPrice(MSFT)>83 ←
NASDAQ(MSFT) ←
TradeVolume(MSFT)=24,229,000 ←

```

To execute a logic program in Prolog, a set of clauses *S* like that presented below should be entered:

```

Picks (Bill, MSFT) ←
Picks(Mary, INTC) ←
Picks(Bill, AMD) ←
Picks(Bill, INTC) ←
Picks(Paul, RNWK) ←

```

If a program is capable of executing logic programs, the following query will get an automatic answer from the program:

```
Picks(Bill; x)?
```

This query is equivalent to the question: "What stock does Bill pick?"

The logic program using a set of clauses *S* as background knowledge will produce a number of answers:

```
x=MSFT, x=AMD, x=INTC
```

Similarly *Picks(Steve,x)?* and *Picks(Mary,x)?* can be asked. This will produce an answer: *x = INTC*.

For now, consider *Picks(Steve,x)?* There are several steps to get this kind of inference:

1. Start search from the first clause;
2. Find any clause whose head has predicate *Picks(,)* and the first argument is *Steve*;
3. If no clause is found return, otherwise go to 4;
4. Associate *x* with the 2nd argument of the head literal (the value associated with *x* is output); mark this clause.
5. Repeat 2-4 for unmarked clauses.

4.6. Background knowledge

4.6.1. Arguments constraints and skipping useless hypotheses

Background knowledge fulfills a variety of functions in the data mining process. One of the most important is reducing the number of hypotheses to be tested to speed up learning and make this process tractable.

There are several approaches to reduce the size of the hypothesis space. Below two of them are presented. They use constraints on arguments of predicates from background knowledge B. The difference is that the first approach uses **constraints on a single argument** and the second one uses **constraints on several arguments** of a predicate defined in B.

The first approach called **typing** approach is based on information about individual data types of arguments of a predicate. For instance, suppose only an integer can be the first argument of predicate P and only the date (M/D/Y) can be the second argument of this predicate. It would be wasteful to test hypotheses with the following typing P(date, integer), P(integer, integer) and P(date, integer). The only one correct type here is P(integer, date).

The second approach is called **inter-argument constraints approach**. For example, predicate Equal(x,x) is always true if both arguments are the same. Similarly, it is possible that for some predicate P for all x **$P(x,x)=0$** . Therefore, testing hypotheses extended by adding Equal(x,x) or P(x,x) should be avoided and the **size of the hypothesis space explored can be reduced**.

The value of inter-argument constraints is illustrated by the experimental fact that the FOCL algorithm, using **typing and inter-argument constraints**, was able to test 2.3 times less literals and examples than using only typing. [Pazzani, Kibler, 1992]. Table 4.12 summarizes properties of the two discussed approaches for reducing the number of hypotheses.

Table 4.12. Approaches for reducing hypothesis space

	Approach 1: Implementing a single argument constraint (typing)	Approach 2: Implementing inter-argument constraints
Definition	Properties of an individual argu- ment of the predicate.	A relationship between different arguments of a predicate
Example of constraints	Only an integer can be the first argument of a predicate. Only date (M/D/Y) can be the sec- ond argument of the predicate.	All of the variables in one predicate should be different, i.e., a hypothesis should not include predicate P(x,x), but may include P(x,y)
Experiment	FOCL algorithm, using typing and inter-argument constraints, was able to test two times less literals and examples than using only typing [M.Pazzani D. Kibler, 1992].	

4.6.2. Initial rules and improving search of hypotheses

This section considers another useful sort of background knowledge--a **(possibly incorrect) partial initial rule** that **approximates the concept (rule)** to be learned. There are two basic forms of this initial rule:

- extensional form and
- intensional form.

If a **predicate is defined by other predicates**, we say the definition is **intensional**. Otherwise, a predicate given by example is called **extensional**. It is also possible that background knowledge B contains a predicate in a **mixed way** partially by examples and partially by other predicates. In general, background knowledge presented in a mixed way reduces the search. [Passani, Kibler, 1992].

Learning using initial extensional rule. An expert or another learning system can provide an initial extensional rule [Widmer, 1990]. Then this rule (initial concept) is refined by adding clauses [Passani, Kibler, 1992]:

1. An algorithm computes the criterion of optimality (usually information gain) of each clause in **the initial concept**.
2. The literal (or conjunction of literals) with the maximum gain is added to the end of the current clause (start clause can be null).
3. If the current clause covers some negative tuples (examples), additional literals are added to rule out the negative tuples.

Learning using initial intensional rules. Next, consider domain knowledge defined in terms of extensional and intensional initial predicates. Systems such as CIGOL [Muggleton & Buntine, 1988] make use of (or **invent**) **background knowledge** of this form. For example, if an extensional definition of the predicate GrowingStock(x,y,z) is not given, it could be defined in terms of the intensional predicate GreaterPrice by:

GrowingStock(x,y,z) ← GreaterPrice(x,y), GreaterPrice(y,z),

where x, y, and z are prices of the stock for days t, **t+1**, and **t+2**, respectively.

It is possible that the intensional predicate GrowingStock(x,y,z) added to the hypothesis improves it, but each of predicates GreaterPrice(x,y) and GreaterPrice(y,z) does not improve the hypothesis. Therefore, common search methods may not discover a valuable stock regularity.

Pazzani and Kibler [1992] suggested that if the literal with the maximum of the optimality criterion (gain) is intensional, then the literal is made extensional and the extensional definition is added to the clause under construction. Table 4.13 shows this idea more specifically. The process is called **operationalization**.

Note that computation of the optimality criterion, which guides the search, is different for extensional and intensional predicates. For intensional predicates it usually involves a Prolog proof.

Table 4.13. Operationalization [Pazzani, Kibler, 1992]

```

Procedure: Operationalize(Predicate, Pos, Neg)
Initialize ClauseBody to the empty set.
For each clause in the definition of Predicate
    Compute_Gain(clause, Pos, Neg).
For the clause with the maximum gain,
    for each literal T in the clause,
        if T is extensional, add T to ClauseBody
        else add Operationalize(T, Pos, Neg) to ClauseBody.

```

Potentially operationalization can generate very long rules, similarly to large decision trees discussed in Chapter 3, when this extensional approach was illustrated versus short intensional formulas.

Table 4.14. Partial background knowledge for stock market

Definition of target predicate to be learned:

Up(Stock(t), Stock(t+1), Stock(t+2)).

IF Stock(t+2) < Stock(t+3) THEN this predicate should be true and the predicate is false

If Stock(t+2) ≥ Stock(t+3).

Up(Stock(t), Stock(t+1), Stock(t+2)) ⇔ Stock(t+2) < Stock(t+3)

To compute this predicate only stock prices Stock(t), Stock(t+1) and Stock(t+2) can be used. Actually the predicate should forecast stock price for date t+3, having stock prices for the three preceding days. The learning algorithm should learn the predicate Up, .i.e., generate a logical rule combining Stock(t), Stock(t+1), Stock(t+2) such that Up(Stock(t), Stock(t+1), Stock(t+2)) ⇔ Stock(t+2) < Stock(t+3) for all training data.

Type : UP(float, float, float, float)

Positive examples, Pos: Ex1--(34.0, 35.1, 36.2, 37.4), Ex2--(37, 38.1, 34.4, 35.7)

Negative examples, Neg: Ex3--(33.2, 32.1, 33.7, 31.6), Ex4--(30.8 29.3, 28.8 27.9)

Intensional Predicate(s):

Q(Stock(t), Stock(t+1), Stock(t+2)) ⇔ Stock(t+1) - Stock(t) < Stock(t+2) - Stock(t+1)

Type : Q(float, float, float);

Extensional Predicates:

Monday (t). t type: date. This predicate is true for Mondays.

Pos : (04.05.99)(04.12.99)(04.19.99)...(11.01.99)

Tuesday(t). t type: date. This predicate is true for Tuesdays.

Pos : (04.06.99)(04.13.99)(04.20.99)...(11.02.99)

Learning using initial intensional and extensional rules. The previous consideration has shown that adding background knowledge can increase the ability of algorithms to find solutions. Table 4.14 shows an example of a partial background knowledge for a stock market forecast. It consists of

- a definition of the target predicate $UP(x,y,w,z)$ with four arguments to be learned,
- typing information about x,y,w and z ,
- intensional predicate $Q(x,y,w)$ with three arguments to be used for discovering predicate $Up(x,y,w,z)$, and
- extensional predicates $Monday(t)$ and $Tuesday(t)$ to be used for discovering $Up(x,y,w,z)$.

In addition, Table 4.15 provides an initial intensional rule for the target concept $Up(x,y,w,z)$

Table 4.15. Intensional initial rule for the target concept

$Up(Stock(t); Stock(t+1); Stock(t+2) \leftarrow Q(Stock(t+1), Stock(t), Stock(t+2), Stock(t+1))$
--

This rule assumes that if growth was accelerated from date t to $t+2$ then the stock will grow further on date $t+3$.

Background knowledge is called **extended background knowledge** if it includes:

- extensional knowledge (training examples and extensional predicates),
- initial rules,
- intensional target concept definition.

Pazzani and Kibler [1992] found in experiments that **extended background knowledge** with a **correct intensional target** definition avoids exhaustive **testing** every variable of every predicate and increases the speed of the **search**. In their experiments, a correct extensional definition of the target concept was found by testing only 2.35% of literals needed for rule discovery if the target concept is not provided. However, the same research has shown that extended background knowledge

- can increase the **search space**,
- can decrease the **accuracy** of the resulting hypothesis, if the background knowledge is partially irrelevant to the task, and
- can increase the number of **training examples** required to achieve a given accuracy.

These observations show the need for **balancing initial intensional and extensional predicates** in background knowledge. One of them can be more accurate and can speed up the search for regularity more than other. Therefore, the following procedure will be more efficient:

- 1) Compare accuracy of intensional and extensional knowledge.

- 2) Include a more accurate one in the background knowledge.
- 3) Discover regularities using the most accurate background knowledge from 2).
- 4) Discover regularities using all background knowledge.

The modification of this mechanism includes use of **probabilities** assigned to all types of background knowledge (see Section 4.8).

There are several ways to combine extensional and intensional knowledge in discovering regularities. One of them is converting initial rules (predicates) into extensional form (**operationalize a clause**) if it has positive information gain.

The extensional predicates are compared to the induced literal with the maximum information gain. This approach is used in the FOIL algorithm (see Section 4.6.3).

In an **explanation-based learning approach**, the target concept is assumed a **correct, intensional definition** of the concept to be learned and the domain knowledge is assumed correct as well. An approach that is more realistic is implemented in algorithms such as FOCL and MMDR. These methods relax the assumption that the target concept and the domain knowledge are correct.

4.6.3. Relational data mining and relational databases

Relational data mining and relational databases use different terms. Translation of first-order logic terms used in RDM to terms used with relational databases is given below [Dzeroski, 1996].

Consider n sets (domains) $D_1, D_2, \dots, D_i, \dots, D_n$ and their Cartesian product

$$D = D_1 \times D_2 \times \dots \times D_i \times \dots \times D_n,$$

i.e., the set of all possible n -tuples with i -th component from D_i ($i=1, \dots, n$).

An **n -ary relation p** (in database terms) is a subset of such a Cartesian product, $p \subset D$, i.e., p can be defined by some set of n -tuples from D . A **relational database (RDB)** consists of a set of relations [Ullman, 1988].

The database term **n -ary relation p** corresponds to the concept of **n -ary predicate p** in first-order logic. **Attributes** of the **relation p** are called **arguments of the predicate**. In ILP and first-order logic, a **tuple**

$$\langle a_1, \dots, a_n \rangle$$

is called a **ground fact $p(a_1, \dots, a_n)$** . An n -ary relation p in a relational database is actually a table of tuples like Table 4.16, which is an extended version of Table 4.7.

Table 4.16. Attribute-based data example

Stock price, \$	Volume, x1000	Date	Weekday	Stock Event
54.6	3067.48	01.04.99	Monday	New product
55.3	35654.31	01.05.99	Tuesday	Reported profit
56.1	3832.11	01.06.99	Wednesday	Government activity

In Table 4.16 a tuples (term)

<54.6, 3067.48, 01.04.99, Monday, New product>

is an argument of predicate Stock

Stock(54.6, 3067.48, 01.04.99, Monday, New product).

which is true. Similarly for tuple

<55.3, 35654.31, 01.05.99, Tuesday, Reported profit>

predicate Stock is true:

Stock(55.3, 35654.31, 01.05.99, Tuesday, Reported profit)=True.

This example and the definition above show that a whole term like

<55.3, 35654.31, 01.05.99, Tuesday, Reported profit>

is a single argument x of predicate Stock(x). Database technology using mechanism of attributes allows a user to access an individual attribute x_i of $\mathbf{x}=\langle x_1,x_2,x_3,x_4,x_5\rangle$.

To be able to do the same in RDM the mechanism of **projection functions** is used (see Section 4.4.2). Each projection function and attribute can be associated with its own set of predicates like P546(x) shown in Section 4.4.2.

Let us consider predicate GreaterPrice(x,y):

GreaterPrice(Stock(t),Stock($t+1$)) $\Leftrightarrow$ StockPriceP(t)<StockPriceP($t+1$).

for stock prices from Table 4.16: \$54.60, \$55.30 and \$56.10.

This predicate can be expressed in two different table forms:

- object-object (Table 4.17), which is a standard predicate table form and
- database object-attribute form (Table 4.18).

Table 4.17. Object-object table for predicate GreaterPrice(x,y)

	Object 01.04.99	Object 01.05.99	Object 01.06.99
Object 01.04.99	0	0	0
Object 01.05.99	1	0	0
Object 01.06.99	1	1	0

Table 4.18. Attribute-based table equivalent to predicate GreaterPrice(x,y)

	Attribute Greater_01.04.99	Attribute Greater_01.05.99	Attribute Greater_01.06.99
Object 01.04.99	0	0	0
Object 01.05.99	1	0	0
Object 01.06.99	1	1	0

A relational database language supports only the object-attribute form presented in Table 4.18, therefore the move from the standard predicate presentation in Table 4.17 requires transformation to Table 4.18. Both of these extensional presentations are very space consuming. Having 10,000 objects, each of these tables will contain 10^8 elements. This simple example shows that it is much more reasonable to store definitions of this kind of predicate in intensional form:

$$\text{GreaterPrice}(\text{Stock}(t), \text{Stock}(t+1)) \Leftrightarrow \text{StockPriceP}(t) < \text{StockPriceP}(t+1)$$

as a short program in Prolog or other languages. However, some ILP software can work only with extensional predicates like in Tables 4.17 and 4.18, processing relatively small tasks.

4.7. Algorithms: FOIL and FOCL

4.7.1. Introduction

A variety of relational machine learning systems have been developed in recent years [Mitchell, 1997]. Theoretically, these systems have many advantages. In practice though, the complexity of the language must be severely restricted, reducing their applicability. For example, some systems require that the concept definition be expressed in terms of **attribute-value pairs** [Lebowitz, 1986; Danyluk, 1989] or only in terms of **unary predicates** [Hirsh, 1989; Mooney, Ourston, 1989; Katz, 1989; Shavlik, Towell, 1989; Pazzani, 1989; Sarrett, Pazzani, 1989]. The systems that allow actual **relational concept definitions** (e.g., OCCAM [Pazzani, 1990], IOE [Flann & Dietterich, 1989], ML-SMART [Bergadano et al., 1989]) place strong

restrictions on the form of induction and the initial knowledge that is provided to the system [Pazzani, Kibler, 1992],

In this section, we present three relational data mining methods: FOIL, FOCL and MMDR. Algorithm FOIL [Quinlan, 1989; Quinlan, 1990] learns constant-free Horn clauses, a useful subset of first-order predicate calculus. Later FOIL was extended to use a variety of types of background knowledge to increase the class of problems that can be solved, to decrease the hypothesis space explored, and to increase the accuracy of learned rules.

Algorithm FOCL [Pazzani, Kibler, 1992], already mentioned several times, extends FOIL. FOCL uses first order logic and FOIL's information-based optimality metric in combination with background knowledge (details are presented in Sections 4.7.2. and 4.7.3). This is reflected in its full name -- First Order Combined Learner. FOCL has been tested on a variety of problems [Pazzani, 1997] that includes a domain theory describing when a **student loan** is required to be repaid [Pazzani & Brunk, 1990].

It is well known that the general problem of rule generating and testing is NP-complete [Hyafil, Rivest, 1976]. Therefore, we face the problem of designing NP-complete algorithms. There are several related questions. What determines the number of rules to be tested? When should one stop generating rules? What is the justification for specifying particular expressions instead of any other expressions? FOCL, FOIL and MMDR use different stop criteria and different mechanisms to generate rules for testing (details are presented in Sections 4.7.2. and 4.7.3). MMDR selects rules which are simplest and consistent with measurement scales (section 4.9.2, [Krantz et al., 1971, 1989, 1990]) for a particular task. The algorithm stops generating new rules when the rules become too complex (i.e., statistically insignificant for the data) in spite of the possibly high accuracy of the rules when applied to training data. The obvious other stop criterion is time limitation. FOIL and FOCL are based on the information gain criterion.

4.7.2. FOIL

The description of FOIL and FOCL below is based on [Pazzani, Kibler 1992]. FOIL uses positive and negative examples $\{\mathbf{e}^+\}$, $\{\mathbf{e}^-\}$ for some concept C , and related (background) predicates. FOIL tries to generate a rule R combining these predicates in such a way that R is true for positive examples, $R(\mathbf{e}^+)=1$, and false for negative examples, $R(\mathbf{e}^-)=0$. This rule should not contain constant and function symbols, but can contain **negated predicates** in both FOIL and FOCL. FOIL design is shown in Tables 4.19 and 4.20 [Pazzani, Kibler, 1992].

FOIL has two major stages:

- the **separate stage** begins a new clause, and
- the **conquer stage** constructs a conjunction of literals to serve as the body of the clause.

Table 4.19. FOIL Design 1

```

Let POS be the positive examples.
Let NEG be the negative examples.
Set NewClauseBody to empty.
Until POS is empty do:
  Separate: (begins new clauses)
    Remove from POS all examples that satisfy the NewClauseBody.
    Reset NEG to the original negative examples.
    Reset NewClauseBody to empty.
  Until NEG is empty do:
    Conquer: (build clause body)
      Choose a literal L.
      Conjoin L to NewClauseBody.
      Remove from NEG examples that do not satisfy L.

```

Each clause describes some subset of the positive examples and no negative examples. FOIL uses two **operators**:

1. Start a new, empty clause, and
2. Add a literal to the end of the current clause.

Adding literals continues until no negative example is covered by the clause. These literals are added to the end of the current clause. FOIL starts new clauses until all positive examples are covered by some clause.

Adding literals assumes a mechanism to **generate literals**, i.e., a particular combination of variables and predicate names. If a predicate (predicate name) is already selected the choice of variables is called a **variablization** (of the predicate) [Pazzani, Kibler, 1992]. If the variable chosen already occurs in an unnegated literal of the rule, then the variable is called **old**. Otherwise, the variable is called **new**. FOIL and FOCL require at least one old variable. This old variable can be in either the head or the current body of the rule (Horn clause).

FOIL uses hill climbing optimization approach to add the literal with the maximum information gain to a clause (rule). This requires computing the information gain for each variablization of each predicate P. The **information gain metric** used by FOIL is

$$\text{Gain(Literal)} = T^{-*} (\log_2 (P_1/P_1+N_1) - \log_2 (P_0/P_0+N_0)),$$

where

P_0 and N_0 are the numbers of positive and negative tuples **before** adding the literal to the clause,

P_1 and N_1 are the numbers of positive and negative tuples **after** adding the literal to the clause, and

T^{++} is the number of positive tuples before adding the literal that has at least one corresponding extension in the positive tuples after adding the literal [Quinlan, 1990].

Cost. A hill-climbing search used by FOIL depends on the branching factor of the search tree. This branching factor is an exponential function:

(1) of the arity of the available predicates, (2) of the arity of the predicate to be learned, and (3) of the length of the clause that is being learned [Pazzani, Kibler, 1992]. Two measures estimate the cost of FOIL computation:

- the theory-cost--**the number of different literals** that can be chosen to extend the body of the given clause (does not depends on the number of
- evaluation-cost --**the cost of computing the information gain** of each training examples),
literal (depends on the number of training examples).

Table 4.20. FOIL Design II [Pazzani, Kibler, 1992].

```

Let POS be the positive tuples.
Let NEG be the negative tuples.
Set NewClauseBody to empty.
Until POS is empty do:
  Separate: (begins new clauses)
    Remove from POS all tuples that satisfy the NewClauseBody.
    Reset Old to be those variables used in P.
    Reset NEG to the original negative examples.
    Reset NewClauseBody to empty.
    Until NEG is empty do:
      Conquer: (refines clause body)
        Choose a predicate P.
        Choose a variablization of the predicate.
        Compute information gain of P and its negation.
        Choose literal L with the most information gain.
        Conjoin the literal with NewClauseBody.
        Add any new variables to Old
        Let POS be all extensions of POS that are satisfied by the literal.
        Let NEG be all extensions of NEG that are satisfied by the literal.

```

Heuristic. Testing the variablizations of some predicates is avoided by a branch-and-bound pruning heuristic in FOIL. The idea is that some variablization can be more specific than another. In a more general variablization, an old variable is replaced with a new variable. The heuristic prefers a more general variablization, computing maximum possible information gain of a predicate with this variablization.

An additional stopping criterion allows FOIL to learn from **noisy data**.

4.7.3. FOCL

Algorithm FOCL [Pazzani, 1997, Pazzani, Kibler, 1992] extends and modifies FOIL to permit the various forms of **background knowledge**:

- **constraints** to limit the search space,
- **predicates defined by a rule** in addition to predicates defined by a collection of examples,
- **input a partial, possibly incorrect rule** that is an initial approximation of the predicate to be learned.

These extensions guide construction of a clause by selecting literals to test. FOCL attempts to constrain search by using variable **typing**, **in-ter-argument constraints**, and an **iterative-widening approach** to add new variables. FOCL specification is presented in Table 4.21.

Table 4.21. FOCL Specification [Pazzani, Kibler, 1992].

Given:
1. The name of a predicate of known arity.
2. A set of positive tuples.
3. A set of negative tuples.
4. A set of extensionally defined predicates.
5. A set of intensionally defined predicates (optional).
6. A set of constraints (e.g., typing) on the intensional and extensional predicates (optional).
7. An initial (operational or non-operational) rule (optional).
Create: A rule in terms of the extensional predicates such that no clause covers any negative examples and some clause covers every positive example

Summary of FOCL. Authors of FOCL draw a number of important conclusions about the complexity of learning rules and the value of different sorts of knowledge. Some of these conclusions are summarized here:

1. The branching factor grows exponentially in the arity of the available predicates and the predicate to be learned.
2. The branching factor grows exponentially in the number of new variables introduced.
3. The difficulty in learning a rule is linearly proportional to the number of clauses in the rule.
4. Knowledge about data types provides an exponential decrease for a search necessary to find a rule.
5. Any method (argument constraints, semantic constraints, typing, symmetry, etc.) that eliminates fruitless paths decreases the search cost and increases the accuracy.
6. The uniform evaluation function allows FOCL to tolerate domain theories that are both incorrect and incomplete.

Advantages of FOCL were experimentally confirmed by Pazzani and Kibler (see Section 4.5.2).

4.8. Algorithm MMDR

4.8.1. Approach

A **Machine Method for Discovering Regularities (MMDR)** contains several extensions over other RDM algorithms. It permits various forms of **background knowledge** to be exploited. The goal of the MMDR algorithm is to create probabilistic rules in terms of the relations (predicates and literals) defined by a collection of examples and other forms of background knowledge.

MMDR as well as FOCL has several advantages over FOIL:

- Limits the search space by using **constraints**.
- Improves the search of hypotheses by using background knowledge with **predicates defined by a rule directly** in addition to predicates defined by a collection of examples.
- Improves the search of hypotheses by accepting as input **a partial, possibly incorrect rule** that is an initial approximation of the predicate to be learned.

There are also advantages of MMRD over FOCL:

- Limits the search space by using the **statistical significance** of hypotheses.
- Limits the search space by using the strength of **data types scales**.
- Shortens the final discovered rule by using the initial set of hypotheses in intensional form directly (without operationalization).

The advantages above represent a way of generalization used in MMDR. Generalization is the critical issue in applying data-driven forecasting systems. The MMDR method generalizes data through “**lawlike**” **logical probabilistic rules** presented in first order logic (Section 4.8.2).

Theoretical advantages of MMDR generalization are presented in [Vityaev, 1976, 1983, 1992, Vityaev, Moskvitin, 1993, Vityaev et al, 1995, Kovalerchuk, 1973, Zagoruiko, 1976, Samokhvalov, 1973]. This approach has some similarity with the hint approach [Abu-Mostafa, 1990]. “A hint may take the form of a global constraint on f , such as a symmetry property or an invariance. It may also be partial information about the implementation of f .” [Abu-Mostafa, 1990]. The main source for hints in first-order logic rules is representative measurement theory [Krantz et al., 1971, 1989, 1990]. Note that a class of general propositional and first-order logic rules, covered by MMDR is wider than a class of decision trees (see Chapter 3).

MMDR selects rules, which are simplest and consistent with measurement scales for a particular task. Initial rule/hypotheses generation for further selection is problem-dependent. In Chapter 5, we present a set of rules/hypotheses specifically generated as an initial set of hypotheses for financial time series. This set of hypotheses can serve as a catalogue of initial rules/hypotheses to be tested (learned) for stock market forecasts. Detailed discussion about a mechanism of initial rule selection using measurement theory [Krantz et al., 1971, 1989, 1990] viewpoint is presented in Section 4.10.2.

The steps of MMDR are described in Figure 4.1. The first step selects and/or generates a class of logical rules suitable for a particular task. The next step learns the particular first-order logic rules using available training data. Then the first-order logic rules on training data using Fisher statistical test [Kendall, Stuart, 1977; Cramer, 1998] are tested. After that statistically significant rules are selected and Occam's razor principle is applied: the simplest hypothesis (rule) that fits the data is preferred [Mitchell, 1997, p. 65]. The last step creates interval and threshold forecasts using selected logical rules: IF $A(x,y,...,z)$ THEN $B(x,y,...,z)$.

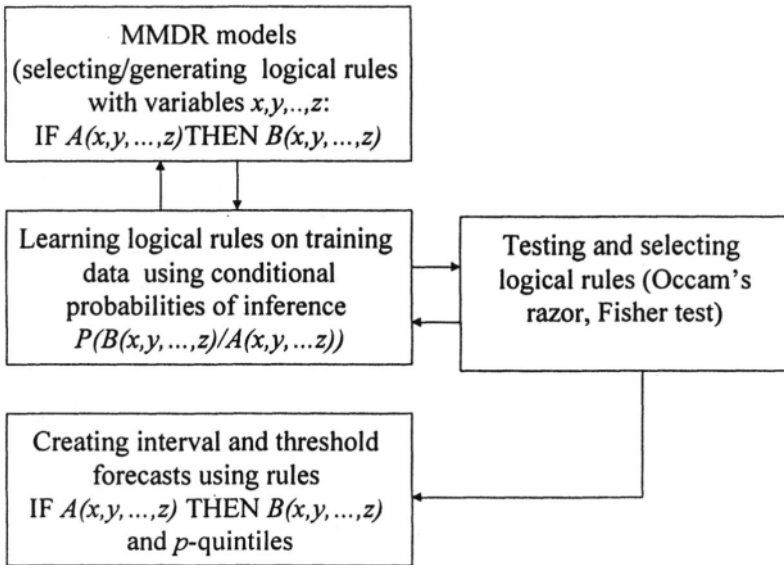


Figure 4.1. Flow diagram for MMDR: steps and technique

It is well known that the general problem of rule generating and testing is NP-complete. MMDR addresses this issue and stops generating new rules

when rules become too complex (i.e., statistically insignificant for the data) in spite of possible high accuracy of these rules for the training data. In this way, the problem becomes tractable. The obvious other stop criterion is a time limitation.

Probabilistic and other first-order methods in machine learning were developed in parallel in the West and in Russia [e.g., Samokhvalov, 1973; Zagoruiko, 1979; Vityaev E. 1983; Int. J. of Pattern Recognition and Artificial Intelligence, 1989; Vityaev, 1976, 1983; Kovalerchuk, 1973,1976] for two decades. Due to many historical reasons, the work in Russia was not well known in the West. The research was concentrated in Pattern Recognition and Applied Logic laboratories in the Institute of Mathematics of the Russian Academy of Sciences (Akademgorodok, Novosibirsk).

What is the difference of MMDR from other data mining learning methods dealing with first-order logic [Mitchell, 1997; Russel and Norvig, 1995]? From our viewpoint, the main emphasis in other first-order methods [Mitchell, 1997; Russel and Norvig, 1995] is on two computational complexity issues:

- how wide is the class of hypotheses tested by the particular data mining algorithms, and
- how does one construct a learning algorithm to find deterministic rules limiting a search space.

The emphasis of MMDR is on **probabilistic first-order rules and measurement** issues for numerical relational methods i.e., how can one move from a real measurement to a first-order logic representation. This is a non-trivial task [Krantz et al., 1971, 1989, 1990]. For example, how does one represent temperature measurement in terms of first-order logic without losing the essence of the attribute (temperature in this case) and without in-putting unnecessary conventional properties? For instance, Fahrenheit and Celsius zeros of temperature are arbitrary conventions in contrast with the Kelvin scale where zero is the lowest possible temperature (the physical zero). Therefore incorporating properties of the Fahrenheit zero into first-order rules may force us to discover/learn properties of this convention along with more significant scale invariant forecasting rules. Learning algorithms in the space with those kind of arbitrary properties may be very time consuming and may produce inappropriate rules.

MMDR uses hypothesis/rule generation and selection processes, based on fundamental representative measurement theory [Krantz, Luce, Suppes and Tversky, 1971, 1989, 1990.] Basic concepts of this theory are presented in Section 4.9.3. The original challenge for MMDR was the simulation of discovering **scientific laws** from empirical data in chemistry and physics. There is a well-know difference between “black box” models and fundamental models (laws) in modern physics. The latter have much longer life,

wider scope, and a solid background. There is reason to believe that MMDR caught some important features of discovering these regularities (“laws”). As was already mentioned, this is an area of extensive research during the last decades.

4.8.2. MMDR algorithm and existence theorem

MMDR expresses patterns in first order logic and assigns probabilities to rules generated by composing patterns. As any technique based on logic rules, this technique allows one to obtain **human-readable forecasting rules** that are **interpretable** in financial language and it provides a forecast. An expert can evaluate the correctness of the forecast as well as forecasting rules. MMDR and related “Discovery” software systems [Vityaev E. 1983; Vityaev, Moskvitin, 1993] generalize data through “law-like” logical probabilistic rules.

Conceptually, **law-like rules** came from the philosophy of science. These rules attempt to mathematically capture the essential features of **scientific laws**: (4) high level of generalization, (2) simplicity (Occam’s razor), and (3) refutability. The first feature -- generalization -- means that any other regularity covering the same events would be less general, i.e., applicable only to the part of testing examples covered by the law-like regularity. The second feature -- simplicity -- reflects the fact that a law-like rule is shorter than other rules. The law-like rule, R1, is more refutable than another rule, R2, if there are more testing examples which refute R1 than R2, but the examples fail to refute R1.

Formally, an IF-THEN rule C is

$$A_1 \& \dots \& A_k \Rightarrow A_0,$$

where the IF-part, $A_1 \& \dots \& A_k$, consists of true/false logical statements $A_1, \dots, A_k$, and the THEN-part consists of a single logical statement A_0 . Statements A_i are some given refutable statements or their negations, which are also refutable. Sub-rules can be generated by truncating the IF-part, e.g.,

$$A_1 \& A_2 \Rightarrow A_0, A_1 \& A_2 \& A_3 \Rightarrow A_0.$$

For rule C its conditional probability

$$\text{Prob}(C) = \text{Prob}(A_0 / A_1 \& \dots \& A_k)$$

is defined. Similarly conditional probabilities

$$\text{Prob}(A_0/A_{i1}\&\dots\&A_{ih})$$

are defined for sub-rules C_i of the form

$$A_{i1}\& \dots\&A_{ih} \Rightarrow A_0, \text{ where } \{A_{i1}, \dots, A_{ih}\} \subset \{A_1, \dots, A_k\}$$

Conditional probability $\text{Prob}(C) = \text{Prob}(A_0/A_1\&\dots\&A_k)$ is used to estimate the forecasting power of the rule to predict A_0 . The rule is “law-like” iff all of its sub-rules have less conditional probability than the rule, and the statistical significance of it is established. Each sub-rule C_i generalizes rule C , i.e., potentially C_i is true for a larger set of instances [Mitchell, 1997]. Another definition of “law-like” rules can be stated in terms of generalization. The rule is a “**law-like**” rule iff it cannot be generalized without producing a statistically significant reduction in its conditional probability. “Law-like” rules defined in this way hold all three properties of scientific laws. They are: (1) general rules from a logical perspective, (2) simple, and (3) refutable. Chapter 5 presents some rules extracted using this approach.

MMDR and the “Discovery” software [Vityaev E. 1983; Vityaev, Moskvitin, 1993; Vityaev 1992] searches all chains

$$C_1, C_2, \dots, C_{m-1}, C_m$$

of nested “law-like” subrules, where C_1 is a sub-rule of rule C_2 , $C_1 = \text{sub}(C_2)$, C_2 is a sub-rule of rule C_3 , $C_2 = \text{sub}(C_3)$ and finally C_{m-1} is a sub-rule of rule C_m , $C_{m-1} = \text{sub}(C_m)$. Also,

$$\text{Prob}(C_1) < \text{Prob}(C_2), \dots, \text{Prob}(C_{m-1}) < \text{Prob}(C_m).$$

There is a **theorem** [Vityaev, 1992] that **all rules, which have a maximum value of conditional probability, can be found at the end of such chains**. The algorithm stops generating new rules when they become too complex (i.e., statistically insignificant for the data) even if the rules are highly accurate on training data. The Fisher statistical test is used in this algorithm for testing statistical significance. The obvious other stop criterion is the time limitation.

Below MMDR is presented more formally using standard first-order logic terms described in Section 4.4.3. Let us consider the set RL of all possible rules of the form:

$$A_1\&\dots\&A_n \Rightarrow A_0 \tag{4}$$

where $A_0, A_1, \dots, A_n$ are atomic formulas in one of the following forms

$P^e(t_1, \dots, t_k)$ and $(t=g)^e$,

Here $t(z^1_1, \dots, z^1_l)$, $g(z^g_1, \dots, z^g_{lg})$, $t_1(z^1_1, \dots, z^1_{l1})$, $\dots$, $t_k(z^k_{k1}, \dots, z^k_{lk})$ are terms,

and $z^1_1, \dots, z^1_l$, $z^g_1, \dots, z^g_{lg}$, $z^1_1, \dots, z^1_{l1}$, $\dots$, $z^k_{k1}, \dots, z^k_{lk}$

are individual constants. Also $e = 1(0)$ is an indicator if there is a negation of the predicate. If $e=1$ then there is no negation of a predicate and if $e=0$ then a predicate is negated. For instance, $(t=g)^0$ means that $t \neq g$, and similarly,

$$P^0(t_1, \dots, t_k) \Leftrightarrow \neg P(t_1, \dots, t_k).$$

The formula (4) means that for every substitution of instances for the individual constants, if the IF-part is true then the THEN-part will also be true.

In this substitution, objects from data D represent individual constants.

The goal of MMDR is finding BST rules, defined below. BST stands for “best”.

Definition 1. Rule $C = B_1, \dots, B_l \Rightarrow A$, where $l > 1$ and $\mu(B_1, \dots, B_l) > 0$ is called a **BST rule** for atomic formula A and data D if and only if:

$$1. \mu(C) = \mu(A/B_1 \& \dots \& B_l) > \mu(A),$$

where $B_1 \& \dots \& B_l$ is a condition generated using data D,

2. Rule C has maximum of conditional probability $\mu(C)$ among rules satisfying condition 1 and generated by the same data, and
3. For any rule C^* satisfying conditions 1 and 2, the property $C \Rightarrow C^*$ is true.

This means that BST rule C is the strongest rule.

Condition 1 means that rule C has a **reasonable If-part** (premise), i.e., the conditional probability μ of rule C is greater than the probability of atomic formula A itself. If this condition is not satisfied then there is no reason to add the premise for forecasting A. If atom A has a high probability itself then it can be predicted without a premise.

Condition 2 brings the “**strongest**” rule, i.e., the rule with maximum conditional probability among rules satisfying condition 1 above for the same data.

Condition 3 means that a BST rule is the **most “general”** among rules satisfying conditions 1 and 2, i.e., a BST rule covers the widest set of cases from D for which it can be applied. Formally, this idea is presented in Definition 2.

Definition 2. Relation $C \rightarrow C'$ is true for rules C and C' if

$$\{A_1, \dots, A_n\} \subset \{A'_1, \dots, A'_{n'}\},$$

where $C = A_1 \& \dots \& A_n \Rightarrow A_0$ and $C' = A'_1 \& \dots \& A'_{n'} \Rightarrow A_0$, $n, n' > 0$.

Now the task is to find the set of all BST rules for data D . It is well known that the exhaustive search of all rules (4) for finding the set of BST rules is practically impossible if predicates with more than one variable are used. Therefore, the search should be constrained to a lesser set of rules, but still allowing the discovery of BST rules. To accomplish this task use the following definition and theorem.

Definition 3. Rule (4) is called a **regularity** if it satisfies the following conditions:

$$\mu(A_1 \& \dots \& A_n) > 0,$$

$$\mu(A_0/A_1 \& \dots \& A_n) > \mu(A_0/\text{SubConjunction}(A_1 \& \dots \& A_n))$$

where μ is the probability of an expression, and

$$\text{SubConjunction}(A_1 \& \dots \& A_n) = A'_1 \& \dots \& A'_k,$$

$$\{A'_1, \dots, A'_k\} \subset \{A_1, \dots, A_n\}, \{A'_1, \dots, A'_k\} \neq \emptyset.$$

Theorem 1. If rule R is a BST rule for data D , then rule R is regularity on data D [Vityaev, 1992].

Let us denote the set of all regularities (rules) by RG . It follows from Theorem 1 that the task now is to find the set of rules RG . Having RG we will be able to make all “best predictions” using regularities from RG .

Partition the set of regularities RG to get chains, **called chains of semantic probabilistic inference** [Vityaev, 1992],

$$C_n \succ C_{n-1} \succ \dots \succ C_2 \succ C_1 \succ A,$$

where $\succ$ is a symbol for semantic probabilistic inference.

Actually, the task consists of **finding all semantic probabilistic inferences**. The following **heuristic rule** is used as the base for a practical implementation of semantic probabilistic inference:

all regularities can be found by searching all chains beginning with a regularity with a short If-part.

This heuristic is used in MMDR for arranging the search of regularities. In spite of the use of a heuristic no one BST rule will be lost. MMDR like FOIL and FOCL operates with Horn clauses (Section 4.4.3). However, the theory was developed for a general case (see the mentioned theorem [Vityaev, 1992, Vityaev et al, 1995]).

Below we use Horn clauses

$$A_1 \& \dots \& A_n \Rightarrow A_0 \quad (5)$$

where $A_0, A_1, \dots, A_n$ are atomic formulas (literals) of the form

$$P_t^e(z_1^t, \dots, z_{l_t}^t), t = 0, 1, \dots, n.$$

Each P_t^e is a predicate and $z_1^t, \dots, z_{l_t}^t, t = 0, 1, \dots, n$ are **individual constants (constants for short)**. Two formulas of type (5) are equivalent if each of them can be obtained from the other by a one-to-one replacement of constants. We will denote all non-equivalent formulas of type (5) as a **RuleSet**.

The concept of **Data Type** was defined in Section 4.9. Each data type is associated with a specific set of predicates. Actually, these predicates and their properties define a data type. Several examples of these predicates are presented in Section 4.10.2.

Let $Pr = \{P_i \in I\}$ be a set of predicates associated with a specific data type and let $Z = \{z_1, z_2, \dots, z_n\}$ of individual constants. Atomic formulas A , which are predicates from Pr or their negations defined on a set of constants from Z are used. Below are presented core concepts used in MMRD.

Regularity: This is a special type of formulas from the **RuleSet**. For instance, in a financial time series, constants $z_1^t, \dots, z_{l_t}^t$ can be represented by days with properties:

$$(z_1 <_{S\&P} z_2) \& (z_2 <_{S\&P} z_3) \& \dots \& (z_{n-1} <_{S\&P} z_n) \rightarrow (z_n <_{S\&P} z_{forecast}),$$

where $z_1 <_{S\&P} z_2$ means that the SP500C is smaller on day z_1 than on day z_2 and that this rule forecasts $z_n <_{S\&P} z_{forecast}$, i.e., SP500C for the next day will be greater than SP500C on day z_n if the given IF-part of the rule is satisfied. More regularities for financial applications are presented in Chapter 5.

Next a formal concept of **regularity type, allowing us to distinguish regularities from other formulas** is also introduced. The general intuitive idea of a formal concept of regularity is that regularity represents the structure of relations between components. It is important to note that the regularity type is defined not as a property of an individual formula itself, but as **relations of this formula with other formulas**. In particular, a chain of

formulas is introduced as a part of regularity type. These formulas are ordered by a “**semantic probabilistic inference**”(➤) relation like

$$C_n \triangleright \dots C_i \dots \triangleright C_2 \triangleright C_1 \triangleright A .$$

All rules C_i are regularities. The last rule C_n is called a **BST regularity**. This ordering already was called a semantic probabilistic inference, because

$$C_{i+1} \triangleright C_i$$

means that probability of C_{i+1} is greater than the probability of C_i and there is a logical relation (sub-rule) between IF-parts of C_i and C_{i+1}

In this way, a heuristic is introduced into a search process, which makes the search in the hypothesis space tractable. The MMDR begins from simplest rules and generates a tree of these chains implementing a kind of **Branch-and-Bound** method. This method begins from Breadth-First search among formulas of a limited length. This set of formulas is called an **initial set of rules**. Suppose that initial rule C_n is obtained. This rule is a tail of some chain like

$$C_n \triangleright \dots C_i \dots \triangleright C_2 \triangleright C_1 \triangleright A .$$

It is possible, in the simplest case, to have just $C_n \triangleright A$. Then a set of possible additions $\{C_{k+1}^{\text{Possible}}\}$ for C_k are generated by calling a function **Specilize_rule(C_k)**, which generates possible additions

$$\text{Specilize_rule}(C_k) = \{C_{k+1}^{\text{Possible}}\}.$$

Specialization using function **Specilize_rule(C_k)** is guided by the strength of attribute scales (expressed in data type). At first, MMDR adds predicates expressing the simplest (rough) data types like nominal scale. Further properties of more complex scales like ordering and interval scales to refine a rule are added. We also need a Boolean function **IsRegularity(R, D)** which will test if R satisfies the definition of a regularity for data D . This function is true if R satisfies the definition and it is false if R does not satisfy the definition.

4.8.3. Fisher test

In Definition 1 in the previous section, conditions for BST rules (probabilistic “laws”) were defined. These conditions are tested by the Fisher test

[Kendall, Stuart, 1977; Cramer, 1998], which is presented in this section for logical expressions. Consider a rule

$$A_1 \Rightarrow A_0.$$

The IF-part of this rule contains only one predicate symbol A_1 , which potentially may be deleted in the generalization process. This generalization can be justified only if the probability of rule

$$C = (\Rightarrow A_0)$$

with empty IF-part is greater than the probability of initial rule

$$A_1 \Rightarrow A_0, \text{ i.e., } \mu(A_0/A_1) > \mu(A_0).$$

To test Condition 2 from Definition 1 the last inequality should be tested. This inequality can be rewritten as follows

$$\mu(A_0/A_1) * \mu(A_1) > \mu(A_0) * \mu(A_1),$$

using a known probabilistic property $\mu(A_0/A_1) * \mu(A_1) = \mu(A_0 \& A_1)$.

The inequality is rewritten again to obtain its equivalent form

$$\mu(A_0 \& A_1) > \mu(A_0) * \mu(A_1).$$

To test the last inequality statistically, the hypothesis H_0 about statistical independence of predicates A_1 and A_0 is generated

$$H_0: \mu(A_0 \& A_1) = \mu(A_0) * \mu(A_1).$$

Next H_0 is tested against the alternative hypothesis H_1

$$H_1: \mu(A_0 \& A_1) \neq \mu(A_0) * \mu(A_1),$$

Mathematical properties of the null hypothesis H_0 are described in [Kendall, Stuart, 1977]. If H_0 is confirmed, then A_1 and A_0 are independent. Thus, inequality $\mu(A_0/A_1) > \mu(A_0)$ for conditional probability $\mu(A_0/A_1)$ is not true. Therefore, formula $A_1 \Rightarrow A_0$ is not a probabilistic “law” (BST rule).

If hypothesis H_0 is rejected then the alternative hypothesis H_1 is true. Hence, A_1 and A_0 are dependent. H_0 is also a hypothesis about equality of probabilities in two sets:

$$H_0: \mu(A_0 \& A_1) = \mu(A_0) * \mu(A_1)$$

against the alternative hypothesis

$$H_1: \mu(A_0 \& A_1) \neq \mu(A_0) * \mu(A_1).$$

If H_0 is rejected then $\mu(A_0/A_1) > \mu(A_0)$ or $\mu(A_0/A_1) < \mu(A_0)$. If the first inequality is true than the tested formula (rule) $A_1 \Rightarrow A_0$ is stronger than A_0 itself. To decide if $\mu(A_0/A_1) > \mu(A_0)$ is true, one can determine which of the following inequalities is true

$$n(A_0 \& A_1) > (n(A_0) * n(A_1))/N, \text{ and } n(A_0 \& A_1) < (n(A_0) * n(A_1))/N,$$

where N is the total number examples, $n(A_1)$, $n(A_0)$, $n(A_1 \& A_0)$ and $n(A_0 \& A_1)$ the number of examples with true A_1 , A_0 , $A_1 \& A_0$ and $A_0 \& A_1$ respectively.

To test hypothesis H_0 against H_1 the exact Fisher test [Kendall, Stuart, 1977] is used. If, using this test, the null hypothesis H_0 is accepted at some significance level α , then predicates A_1 and A_0 are independent. Therefore, there is no regularity. If H_0 is rejected, then H_1 is accepted. If H_1 corresponds to $\mu(A_0/A_1) > \mu(A_0)$, then the tested formula (rule) $A_1 \Rightarrow A_0$ is accepted at the significance level α .

This means that potentially rule $A_1 \Rightarrow A_0$ can be a BST rule if its conditional probability will be highest among other rules.

Consider a more general hypothesis (rule) $C = (A_1 \& \dots \& A_n \Rightarrow A_0) \in S$. This case can be solved using the same method.

Let $DC = \{A_1, \dots, A_n\}$, $D \subset DC$ (D is a proper subset of DC), and $DC^* = A_1 \& \dots \& A_n$, D^* be the conjunction of literals from D .

To test if rule C is a BST the following inequality should be tested:

$$\mu(A_0/DC^*) > \mu(A_0/D^*)$$

for every subset of D (including $\emptyset$).

Consider conjunction D^* as a formula R_1 and consider conjunction of literals from $DC \setminus D$ as another formula R_2 . If $D = \emptyset$, then $R_1 = \text{true}$, and $\mu(P^{\text{test}}_0/D^*) = \mu(P^{\text{test}}_0)$.

For the testing of rule C the inequality should be considered:

$$\mu(A_0/R_1 \& R_2) > \mu(A_0/R_1).$$

Having $\mu(A_0/R_1 \& R_2) = \mu(A_0 \& R_1 \& R_2) / \mu(R_1 \& R_2) = \mu(A_0 \& R_2 / R_1) / \mu(R_2 / R_1)$, the previous inequality is transformed to the inequality

$$\mu(A_0 \& R_2 / R_1) > \mu(R_2 / R_1) * \mu(A_0 / R_1).$$

$\mu(DC^k) > 0$, $\mu(R_1) > 0$, and $\mu(R_2) > 0$ because $n[A_1 \& \dots \& A_n] > 0$, and $D \subset DC$ and $DC \setminus D \subset DC$. To test the last inequality consider the null hypothesis about independence

$$H_0: \mu(A_0 \& R_2 / R_1) = \mu(R_2 / R_1) * \mu(A_0 / R_1)$$

against the alternative hypothesis

$$H_1: \mu(A_0 \& R_2 / R_1) \neq \mu(R_2 / R_1) * \mu(A_0 / R_1).$$

Considering only tuples where formula R_1 is true a subalgebra $\mathfrak{R}$ of Boolean algebra of all sets of tuples is identified. A probabilistic measure $\mu'(A) = \mu(A \& R_1) / \mu(R_1)$ is defined for any atom A . Then hypotheses H_0 and H_1 can be written

$$H_0: \mu'(A_0 \& R_2) = \mu'(R_2) * \mu'(A_0), \text{ and}$$

$$H_1: \mu'(A_0 \& R_2) \neq \mu'(R_2) * \mu'(A_0).$$

Hypothesis H_0 is also tested with the Fisher test at some significance level α . Rule C will be a regularity at the significance level α , if the null hypothesis H_0 is rejected at the level α for any subset $D \subset DC$ and all hypotheses H_1 are accepted with inequalities corresponding to $\mu(A_0 / A_1) > \mu(A_0)$. If C is rejected (as regularity), then we need to test if any more general part of C can be accepted at the same significance level. To generate these parts we use as DC all sequentially possible subsets $D \subset DC$ (proper subsets) of the IF-part of the rule. For each $D' \subset D \subset DC$ all hypotheses are tested again to find if the rule with IF-part D is a regularity. As was mentioned above BST rules are also “probabilistic laws” (regularities).

4.8.4. MMDR pseudocode

MMDR pseudocode is presented in this Section in C++ style as a value returning function `MMDR(D, InitialRuleSet)` with two parameters: data (D) and set of initial rules (InitialRuleSet). This function returns a learned rule (Learned_rule) as a set of Horn clauses.

Table 4.22. MMDR function returning a learned rule

```

MMDR(DataType D, RuleTypeSet InitialRuleSet)
  • Learned_rule ← {}
  • i = 1 // set up the first level (depth of search tree to find a rule)
  • while InitialRuleSet, do
    • NewRule(i) ← Some_rule_from(InitialRuleSet)
    • If (Is_Regularity(NewRule(i),D))
    • then
      • Learned_Rule = AddNewRuleToLearnedRules(Learned_Rule, NewRule(i))
      • RuleSet(I) = Spec.Specialize_rule(NewRule(i))
      • while (i > 0) do
        • while RuleSet(i) do
          • Spec.Rule ←
Some_rule_from(Spec.Specialize_rule(NewRule(i)))
          // Function Some_rule...selects the next rule
          // NewRule(i) is a Rule of i-th depth level
          • If (Is_Regularity(Spec.Rule,D))
          • then
            • Learned_Rule=AddNewRuleToLearnedRule(Learned_Rule,
              Spec.Rule)
            • i++ // move to the next i+1 level of the search tree.
            • NewRule(i) = Spec.Rule
            • RuleSet(i) = Spec.Specialize_rule(NewRule(i))
            // changes in while condition.
          • else
            • RuleSet(i) = Delete_tested_NewRule(RuleSet(i), Spec.Rule)
          • i-- // Go to the previous i-1 level and
            // change the rule set RuleSet(i)
        • InitialRuleSet=Delete_tested_NewRule(InitialRuleSet, NewRule(i))
      • Return Learned_Rule

```

This pseudocode includes declarations of five top-level components:

- **Data D** (as an object of the type **dataType**),
- **An initial set of rules**, consecutive sets of rules and finally a set of learned_rule (as objects of the type **ruleSetType**),
- a **rule R** (as an object of the type **ruleType**),
- a **value returning function Specialize_rule(Rule)** for generating a more specialized rule R' from Rule R ,

a **Boolean function** (predicate) **Is Regularity(Rule(i),D)**. This function tests if rule $R(i)$ is a regularity on data D , i.e., it is statistically significant on D and has higher probability than its subrules on D . Functions **Specialize_rule** and

Is_Regularity used in MMDR pseudocode are presented in Tables 4.23, 4.24.

Table 4.23. Rule returning function **Specialize_rule**

```

RuleTypeSet Specialize_rule(Rule)      // Rule =  $A_1 \& \dots \& A_n \rightarrow A_0$ 
While Pr do // Pr =  $\{P_{i \in I}\}$  – the set of all Boolean member functions of DataType class.
    •  $P \leftarrow Pr$ 
    • NewRule = AddNewPredicateToThePremiseOfTheRule(P,Rule)
      // NewRule =  $(P \& A_1 \& \dots \& A_n \rightarrow A_0)$ 
    • If Is_RuleType(NewRule)
      then AddNewRuleToSpecializeRule(NewRule, Specialize_rule)
    • NewRule = AddNewPredicateToThePremiseOfTheRule(NotP,Rule)
      // NewRule =  $(\text{Not}P \& A_1 \& \dots \& A_n \rightarrow A_0)$ 
    • If Is_RuleType(NewRule)
      then AddNewRuleToSpecializeRule(NewRule, Specialize_rule)
    • Return Specialize_rule

```

Table 4.24. Boolean function **Is_Regularity**

```

Boolean Is_Regularity(Rule(i),D)      // Rule(i) =  $A_1 \& \dots \& A_i \rightarrow A_0$ 
// This procedure is detailed in the previous section.
If according to the Fisher test with some confidence level  $\alpha$  the following conditions
are true,
    •  $\mu(A_1 \& \dots \& A_n) > 0$ ;
    •  $\mu(A_0 / A_1 \& \dots \& A_n) > \mu(A_0 / \text{SubConjunction}(A_1 \& \dots \& A_n))$ ;
    where  $\mu$  is a probability of expression,
    SubConjunction( $A_1 \& \dots \& A_n$ ) =  $A'_1 \& \dots \& A'_k$ ,
     $\{A'_1, \dots, A'_k\} \subset \{A_1, \dots, A_n\}$ , and  $\{A'_1, \dots, A'_k\} \neq \emptyset$ ;
    then Return true
else Return false

```

Above components of MMDR are supported by the declarations:

class DataType// this class defines all attributes used in the learning task

```

{
Public:
class attribute1;
class attribute2;
class attribute3;
...
class attribute_n;
char z1,z2,z3,...zn; // individual constants--names for available data
                      //objects/individuals

```

class attribute //this class describes properties of an attribute

```
{
public:
Boolean memberfunction_1( $\mathbf{z}_1^1, \dots, \mathbf{z}_n^1$ )
// member functions depend on a specific attribute
// examples are presented in Section 4.10.2.
Boolean memberfunction_2( $\mathbf{z}_1^2, \dots, \mathbf{z}_n^2$ )
Boolean memberfunction_3( $\mathbf{z}_1^3, \dots, \mathbf{z}_n^3$ )
...
Boolean memberfunction_m( $\mathbf{z}_1^m, \dots, \mathbf{z}_n^m$ )
}
```

Class RuleType // this class is domain specific

```
{
// There is no generic format for RuleType.
// Financial examples are presented in Chapter 5 and
// in several other places in the book.
}
```

Global variables:

Pr = $\{\mathbf{P}_i \in \mathbf{I}\}$ // the set of all Boolean member functions of Data Type
//class. These Boolean member functions are called
// predicates in first-order logic and ILP.

Z = $\{\mathbf{z}_1, \mathbf{z}_2, \dots\}$ – the set of constants.

Z(Rule) – all constants of the **Rule**.

4.8.5. Comparison of FOIL and MMDR

In Table 4.25 a comparison of major features of the FOIL and MMDR algorithms are presented. Advantages of MMDR include the possibility of value forecast and evaluate a learned rule statistically, i.e., evaluate its predictive power. In addition, the search mechanism does not miss the best rules.

A number of other RDM methods have been developed during the last few decades. Several methods are reviewed in [Mitchell, 1997] and [Paz-zani, Kibler, 1992]. Some operate in overly general background knowledge (e.g., IOU system [Mooney Ourston, 1989], A-EBL [Cohen, 1990]). Some permit specific background knowledge domains (e.g., EITHER [Ourston, Mooney, 1990], ML-SMART [Bergadano, Giordana, 1988]). Recent study concentrates on RDM methods handling a variety of qualitative and quantitative data types in combination with a probabilistic approach.

Table 4.25. Comparison of FOIL and MMDR

FOIL	MMDR
Solves classification tasks.	Solves classification and value forecasting tasks.
Evaluation function is a one-criterion function.	Evaluation function is a two-criterion function.
Criteria in the evaluation function is the minimum number of bits needed to encode the classification of all positive bindings of the rule.	The first evaluation criterion is a conditional probability of the rule. The second evaluation criterion is the statistical significance of increasing the first criterion if a new literal is added.
Evaluation function does not indicate a prediction power of the rule.	Evaluation function indicates a prediction power of the rule (probabilistic estimates).
May lose the best rules. Search is not complete. Speeds up search by restricting search (deletes all covered positive examples from further search after finding a rule). Best rules are defined in terms of FOIL's evaluation function.	The search mechanism does not lose the best rules. The best rules are defined in terms of MMDR's evaluation criterion--max of conditional probability.

4.9. Numerical relational data mining

Rapid growth of databases is accompanied by growing variety of types of these data. Relational data mining has unique capabilities for discovering a wide range of human-readable, understandable regularities in databases with various data types. However, the use of **symbolic relational data for numerical forecast** and discovery regularities in numerical data requires the solution of two problems.

Problem 1 is the **development of a mechanism for transformation between numerical and relational data presentations**. In particular, numerical stock market time series should be transformed into symbolic predicate form for discovering relational regularities.

Problem 2 is the discovery of rules **computationally tractable** in a relational form.

Representative measurement theory (RMT) is a powerful mechanism for approaching both problems. Below we review RMT theory from this viewpoint. RMP was originated by P. Suppes and other scientists at Stanford University [Scott, Suppes, 1958; Suppes, Zines, 1963]. For more than three decades, this theory produced many important results summarized in [Krantz et al., 1971, 1981, 1990, Narens, 1985; Pfanzagl, 1968]. Some related to data mining are presented below. RMT was motivated by the intention to formalize measurement in psychology to a level comparable with physics. Further study has shown that the measurement in physics itself

should be formalized. Finally a formal concept of **measurement scale** was developed. This concept expresses what we call a **data type** in data mining. Connections between learning methods and the measurement theory were established in the 70s [Samokhvalov, 1973; Vityaev 1976; Zagoruiko, 1979; Kovalerchuk, 1973; Lbov et al, 1973; Vityaev, 1983]. First of all, RMT yields the result that

*any numerical type of data can be transformed into a relational form with complete preservation of the **relevant** properties of a numeric data type.*

It means that all regularities that can be discovered with numeric data presentation also can be discovered using relational data presentation. Many theorems [Krantz, et al, 1971, 1981, 1990; Kovalerchuk, 1975] support this property mathematically. These theorems are called homomorphism theorems, because they match (set homomorphism) numerical properties and predicates. Actually, this is a **theoretical basis** of logical data presentations without losing empirical information and without generating meaningless predicates and numerical relations. Moreover, RMT changes the paradigm of data types. What is the primary numerical or relational data form? RMT argues that a numerical presentation is a secondary one. It is a derivative presentation of an **empirical relational data type**. The theorems mentioned support this idea.

The next critical result from measurement theory for learning algorithms is that the **ordering relation** is the most important relation for this transformation. **Most practical regularities can be written in this form and discovered using an ordering relation.** This relation is a central relation in transformation between numerical and relational data presentation.

The third idea prompted by RMT is that the hierarchy of data types developed in RMT can help to **speed up the search for regularities**. The search begins with testing rules based on properties of weaker scales and finishes with properties of stronger scales as defined in RMT. The number of search computations for weaker scales is smaller than for stronger scales. This idea is actually implemented in MMDR method (Section 4.8). MMDR begins by discovering regularities with monadic (unary) predicates; e.g., x is larger than constant 5, $x > 5$ and then discovers regularities with ordering relation $x > y$ with two variables. In other words, MMDR discovers regularities similar to decision-tree type regularities and then MMDR discovers regularities based on ordering relations which are more complex first order logic relations.

Another way to speed up the search for regularities is **to reduce the space of hypotheses**. Measurement theory prompts us to search in a smaller space of hypotheses. The reason is that there is **no need to assume any par-**

ticular class of numerical functions to select a forecasting function in this class. In contrast, numerical interpolation approaches assume a class of numerical functions. Neural networks assume this, defining the network architecture, activation functions, and other components. In contrast relational data mining tests a property of monotonicity just for one predicate instead of thousands different monotone numerical functions. We actually have used this advantage in [Kovalerchuk, Vityaev, Ruiz, 1996,1997]. RMT has several theorems, which match classes of numeric functions with relations.

The next benefit from RMT is that the theory prompts us to **avoid incorrect data preprocessing** which may violate data type properties. This is an especially sensitive issue if preprocessing involves combinations of several features. For instance, results of case-based reasoning are very sensitive to preprocessing. Case-based reasoning methods like k-neighbors compute the nearest k objects and assign target value to the current object according to target values of these k-neighbors. Usually a discovered regularity is changed significantly if an uncoordinated preprocessing of attributes is applied. In this way, patterns can be corrupted. In particular, measurement theory can help to select appropriate preprocessing mechanisms for stock market data and a better **preprocessing mechanism speeds up the search for regularities**.

The discussion above shows that there should be two steps in the transformation of numerical data into relational form:

- extracting, generating, and discovering essential predicates (relations), and
- matching these essential predicates with numerical data.

Sometimes this is straightforward. Sometimes this is a much more complex task, especially taking into account that computational complexity of the problem is growing exponentially with the number of new predicates. Relational data mining can be viewed also as a **tool for discovering predicates**, which will be used for solving a target problem by some other methods. In this way, the whole data mining area can be considered as a **predicate discovery technology**.

Several studies have shown the actual success of discovery of understandable regularities in databases significantly depends on use of **data type information**. Date type information is the first source to get predicates for relational data presentation (see Section 4.10).

A formal mechanism for **describing data types** is well developed in Object-Oriented Programming (OOP) and is implemented in all modern programming languages. An abstract data type in C++ includes private elements and member functions. These functions describe appropriate operations with elements and relations between elements. However, OOP does not help much in generating these operations and relations themselves. The

OOP's intent is to provide a tool for describing **already generated abstract data types**.

In contrast, RMT is intended for **generating operations and predicates** as a description of a data type. This RMT mechanism is called an **empirical axiomatic theory**. A variety of data types as matrices of comparisons of pairs, and multiple comparisons, attribute-based matrices, matrices of ordered data, and matrices of closeness can be represented in this way (see below). Then the relational representation of data and data types are used for discovering understandable regularities. We argue that current learning methods utilize only part of data type information actually presented in data. They either lose a part of data type information or add some non-interpretable information.

The **language of empirical axiomatic theories** is an adequate lossless language to reach this goal. There is an important difference between language of empirical axiomatic theories and first-order logic language. The first-order logic language does not indicate anything about real world entities. Meanwhile the language of empirical axiomatic theories uses first order logic language as a mathematical mechanism, but it also incorporates additional concepts to meet the strong requirement of empirical interpretation of relations and operations.

4.10. Data types

4.10.1. Problem of data types

The design of a particular data mining system implies the selection of the set of data types supported by that system. In Object-Oriented Programming (OOP), this is a part of the software design. Data types are declared in the process of software development. If data types of a particular learning problem are **out of the range** of the data mining system, users have two options: to redesign a system or to corrupt types of input training data for the system. The first option often does not exist at all, but the second produces an inappropriate result.

There are two solutions for this problem. The first is to develop data type **conversion mechanisms** which may work correctly within a data mining tool with a limited number of data types. For example, if input data are of a cyclical data type [Krantz et al, 1970, 1981, 1990; Kovalerchuk, 1973] and only linear data types are supported by the DM tool, one may develop a coding of the cyclical data such that a linear mechanism will process the data correctly.

Another solution would be to develop universal DM tools to **handle any data type**. In this case, member functions of a data type should be input information along with the data (MMDR implements this approach). This problem is more general than the problem of a specific DM tool. Let a relative difference for the stock price be

$$\Delta(t)=[(\text{StockPrice}(t) - \text{StockPrice}(t-1))/\text{StockPrice}(t)]$$

a “float” data type. This is correct for a computer memory allocation, but it does not help to decide if all operations with float numbers are applicable for $\Delta(t)$. For instance, what does it mean to add one relative difference $\Delta(x)$ to another $\Delta(y)$? There is no empirical procedure matching this sum operation. However, the comparison operation makes sense, e.g.,

$$\Delta(x) < \Delta(y)$$

means faster growth of stock price on date y than on date x. This relation also helps interpret a relation “ $\Delta(w)$ between $\Delta(x)$ and $\Delta(w)$ ” as

$$\Delta(x) < \Delta(w) \text{ and } \Delta(w) < \Delta(y) \text{ or } \Delta(y) < \Delta(w) \text{ and } \Delta(w) < \Delta(x)$$

Both of these relations are already interpreted empirically.

Therefore, Δ values can be compared, but one probably should avoid an addition operation (+) if the goal is to produce an interpretable learned rule. If one decides to ignore these observations and applies operations formally proper for float numbers in programming languages, then a learned rule will be difficult to interpret. As was already mentioned, these difficulties arose from the uncertainty of the set of interpretable operations and predicates for these data, i.e., uncertainty of **empirical contents of data**. The precise definition of empirical content of data will be given in terms of the empirical axiomatic theory below.

Relational data types

A data type is relational if it is described in terms of the set of relations (predicates). Some basic relational data types include:

- Equivalence data type,
- Tolerance data type,
- Partial ordering data type,
- Weak ordering data type,
- Strong ordering data type,
- Semi-ordering data type,

- Interval ordering data type,
 - Tree data type.
- Next, we define these data types.

An **equivalence data type** is based on an equivalence relation E :

for any $a, b, c \in A$,

1. $E(a, a) = 1$ (true),
2. $E(a, b) \Leftrightarrow E(b, a)$, and
3. $E(a, b) \& E(b, c) \Rightarrow E(a, c)$.

Further, we will often omit the predicate truth value assuming that $P(x, y)$ is the same as $P(x, y) = 1$.

The pair $\langle A, E \rangle$ is called an equivalence data type, where A is a set of possible values of an attribute and E is an equivalence relation. An equivalence relation partitions set A . This data type is called a **nominal scale** in representative measurement theory. An equivalence data type can be presented as a **class** in programming languages like C++ with a Boolean member function $E(x, y)$.

A **tolerance data type** is based on a tolerance relation L :

for any $a, b \in A$,

1. $L(a, a) = 1$, and
2. $L(a, b) \Leftrightarrow L(b, a)$.

A tolerance relation L is weaker than an equivalence relation E . Intervals satisfy the tolerance relation but not the equivalence relation.

Example. Let $L(a, b) \Leftrightarrow |a - b| < 1$. If $a = 0$, $b = 0.5$ and $c = 1.5$ then $L(a, b) = 1$, but $L(a, c) = 0$ (false), because $|a - c| = 1.5$.

A **partial ordering data type** is based on a partial ordering relation P :

for any $a, b, c \in A$,

1. $P(a, a) = 1$,
2. $P(a, b) \& P(b, c) \Rightarrow P(a, c)$.

This relation is also called a **quasi-ordering relation**. There is no numeric coding (representation) for elements of A completely consistent with relation P , i.e., there is not a numeric relation equivalent to P . The formal concepts of consistency and equivalency will be discussed in the next section.

A **weak ordering data type** is based on a weak ordering relation P :

for any $a, b, c \in A$,

1. $P(a, b) \vee P(b, a)$,
2. $P(a, b) \& P(b, c) \Rightarrow P(a, c)$.

The difference between partial ordering and weak ordering is illustrated by noting that any a and b can be compared by a weak ordering, but in partial ordering some a and b may not be comparable.

A **strong ordering data type** is based on a strong ordering relation P :
for any $a, b, c \in A$,

1. $P(a, a) = 0$,
2. $P(a, b) \vee P(b, a)$, and
3. $P(a, b) \& P(b, c) \Rightarrow P(a, c)$.

The difference between strong ordering and weak ordering can be illustrated with two relations: “ $\geq$ ” and “ $>$ ”. For instance, $a \geq a$ is true, but $a > a$ is false, and $P(a, a) = 0$. It is proved in [Pfanzagl, 1971] that for strong and weak ordering data types $\langle A, P \rangle$, A can be coded by numbers with preservation of properties 1-3 if A is countable.

A **semi-ordering data type** is based on a semi-ordering relation P :
for any $a, b, c \in A$,

$$P(a, b) \& P(b, c) \Rightarrow \forall d \in A \text{ such that } P(a, d) \vee P(d, c).$$

This relation P can be illustrated with intervals. Let a function U be a coding function each element of A as a real number, $U: A \rightarrow \mathbf{Re}$. Then $P(a, b)$ is defined as $P(a, b) \Leftrightarrow U(a) + 1 < U(b)$.

The function U is called a **numeric representation** of A . It is not necessary that a numeric representation exist for any data type. For instance, a partial ordering relation does not have a numeric representation. At first glance this seems strange, because we always can code elements of A with numbers. However, there is no **numerical coding** consistent with a partial ordering relation P . Consider an a and b which are not comparable, i.e., $P(a, b) = 0$, but a and b are coded by numbers. Any numbers are obviously comparable, e.g., $4 < 5$.

Therefore, a non-interpretable property is brought to A by numerical coding of its elements. However, if numerical comparison is not used for constructing a learned rule, any one-to-one numerical coding can be used. If a numerical order is used in learning a rule, then this rule can be non-interpretable.

An **interval ordering data type** is based on an interval-ordering relation P : for any $a, b, c, d \in A$,

1. $P(a, a)$,
2. $P(a, b) \& P(c, d) \Rightarrow (P(a, d) \vee P(c, b))$.

It is proved in [Fishbern, 1970] that a numeric representation for $\langle A, P \rangle$ exists completely consistent with properties 1 and 2. There are functions U and S , $U, S: A \rightarrow \mathbf{Re}^+$, such that for any $a, b \in A$,

$$P(a,b) \Leftrightarrow U(a) + S(a) < U(b).$$

A **tree-type data type** is based on a tree-type relation P:

for any $a,b,c,d \in A$,

1. $P(a,a)=0$,
2. $P(a,b) \& P(a,c) \Rightarrow P(b,c) \vee P(c,b)$.

There is no numerical representation completely consistent with these properties.

If some ordering relation is interpretable in terms of domain background knowledge, then it can be included in background knowledge and used for discovering rules in RDM directly. Partial ordering and tree-type relations often appear in hierarchical classifications and in decision trees. In financial applications, usually the data are presented as numeric attributes, but often relations are not presented explicitly. More precisely, these attributes are coded with numbers, but applicability of number relations and operations must be confirmed. Let us illustrate use of relations defined in this section for identifying data type for stock relative difference $\Delta(x)$. Let $PS(x,y)$ be defined as follows

$$PS(x,y) \Leftrightarrow \Delta(x) > \Delta(y)$$

and a financial expert agrees that $PS(x,y)$ makes sense. Now we can identify its type. This is a strong ordering relation. Therefore, we can identify the Δ attribute as an attribute of the strong relational data type. Similarly we can define $PW(x,y) \Leftrightarrow \Delta(x) \geq \Delta(y)$ and identify Δ as an attribute of the weak relational data type. One can continue identify Δ as belonging to other data types listed in this section as well. One may wish to produce a predicate $PM(x,y,z)$,

$$PM(x,y,z) \Leftrightarrow \Delta(x) + \Delta(y) = \Delta(z).$$

There is little financial sense in this predicate, because the operation (+) is not financially interpreted for Δ . The above considerations show that there are relations without a numeric representation. Therefore, **the relational data representation in the first order logic is more general than a numeric representation**. What is the traditional way of processing binary relations? Traditionally numerical methods use distance functions (from a metric space) between matrices of relations (binary relations). These distance functions are defined axiomatically or using some statistical assumptions associated with coefficients of Stuart, Yule and Kendall, information gain measures, etc. Obviously, these assumptions restrict the areas of applicability for these methods.

4.10.2. Numerical data type

In previous sections, the term numerical data was used without formal definition. Actually, there are several different numerical data types. The strongest one is called **absolute data type (absolute scale)**. Having this data type in background knowledge B of a learning problem, we can use most of the known **numerical** relations and operations for discovering a rule. The weakest numerical data type is **nominal data type (nominal scale)**. This data type is the same as the **equivalence data type** defined in Section 4.9.1. It allows us only one interpretable relation $Q(x,y)$. In other words, it is known if x and y are equal. In between there is a spectrum of data types allowing one to compare values with ordering relations, to add, multiply, divide values and so on. Stevens [1946] suggested classification of these data types. They are presented in Table 4.26. The basis of this classification is a transformation group. The strongest absolute data type does not permit to transform data at all, and the weakest nominal data type permits any one-to-one transformation. This data type permits one-to-one coding of data entities by numbers. Intermediate data types permit different transformations such as positive monotone, linear and others (see Table 4.26) [Krantz, et al, 1990].

Table 4.26. Numerical data types (Stevens' classification of scale types)

Transformation	Transformation Group	Data type (scale)
$X \rightarrow f(x),$	$F : Re \rightarrow (onto)Re, 1 \rightarrow 1$ transformation group	Nominal
$X \rightarrow f(x),$	$F : Re \rightarrow (onto)Re$ homeomorphism group	Order
$X \rightarrow rx + s, r > 0$	Positive affine group	Interval
$X \rightarrow tx^r, \quad t,r > 0$	Power group	Log-interval
$X \rightarrow x + s$	Translation group	Difference
$X \rightarrow tx, t > 0$	Similarity group	Ratio
$X \rightarrow x$	Identity group	Absolute

4.10.3. Representative measurement theory

The main definitions from representative measurement theory are reviewed in this section. A **relational structure A** consists of a set A and relations $S_1, \dots, S_n$ defined on A

$$A = \langle A, S_1, \dots, S_n \rangle.$$

Each relation S_i is a Boolean function (predicate) with n_i arguments from A. The relational structure $A = \langle A, S_1, \dots, S_n \rangle$ is considered along with a relational structure of the same type

$$R = \langle R, T_1, \dots, T_n \rangle.$$

Usually the set R is a subset of $\mathbf{Re}^m$, $m \geq 1$, where $\mathbf{Re}^m$ is a set of m -tuples of real numbers and each relation T_i has the same n_i as the corresponding relation S_i . T_i and S_i are called a k -ary relation on R . Theoretically, it is not a formal requirement that R be numerical.

Next, the relational system A is interpreted as an **empirical real-world system** and R is interpreted as a **numerical system** designed as a **numerical representation** of A . To formalize the idea of numeric representation, we define a homomorphism ϕ as a mapping from A to R .

A mapping $\phi: A \rightarrow R$ is called a **homomorphism** if for all i ($i = 1, \dots, n$),

$$(a_1, \dots, a_{k(i)}) \in S_i \Leftrightarrow (\phi(a_1), \dots, \phi(a_{k(i)})) \in T_i.$$

In other notation,

$$S_i(a_1, \dots, a_{k(i)}) \Leftrightarrow T_i(\phi(a_1), \dots, \phi(a_{k(i)})).$$

Let $\Phi(A, R)$ be the set of all homomorphisms for A and R . It is possible that $\Phi(A, R)$ is empty or contains a variety of representations. Several theorems are proved in RMT about the contents of $\Phi(A, R)$. These theorems involve: (1) whether $\Phi(A, R)$ is empty, and (2) the size of $\Phi(A, R)$. The first theorems are called **representation theorems**. The second theorems are called **uniqueness theorems**.

Using the set of homomorphisms $\Phi(A, R)$ we can define the notion of permissible transformations and the data type (scale types). The most natural concept of permissible transformations is a mapping of the numerical set R into itself, which should bring a “good” representation. More precisely, γ is **permissible** for $\Phi(A, R)$ if γ maps R into itself, and for every ϕ in $\Phi(A, R)$, $\gamma\phi$ is also in $\Phi(A, R)$. For instance, the permissible transformations could be transformations, $x \rightarrow rx$ or monotone transformations $x \rightarrow \gamma(x)$.

4.10.4. Critical analysis of data types in ABL

The empirical status of data types for different learning tasks and their treatment by known numerical data mining methods such as regression, correlation, covariance analysis, and analysis of variance applications are examined in this section. In the previous section, it was required that a relational system A representing a data type should be interpreted as an **empirical real-world system**, i.e., A should be included in the domain background knowledge of a learning task. Numerical methods such as regression, correlation, covariance analysis and analysis of variance assume that any numerical standard mathematical operations (+, -, *, / and so on) can be used despite

their possible non-interpretability. In this way, a non-interpretable learned rule can be obtained as well. Let us consider this situation in more detail for six different cases.

Case 1. Physical data types in physical problems. Multidimensional data contain only physical quantities and the learning task itself belongs to physics, where data types and measurement procedures are well developed. In this case, the measurement theory [Krantz et al, 1970, 1981, 1990] provides formalized empirical systems and the groups of permissible transformations of all quantities. The use of the mentioned methods is most appropriate. However, the following problems remain:

a) To establish the invariance of the data mining methods within permissible transformations of quantities of the data is needed. The invariance of the methods is a necessary condition of their meaningfulness. The invariance means that the resulting learned rule should not depend on choosing particular numerical representations and measurement units. As is shown in [Kuzmin et al., 1977; Orlov, 1979, 1977; Terekhina, 1973; Tyurin et al., 1981; Roberts et al., 1976], the proof of the invariance of the methods is a rather difficult task and most methods are not invariant for permissible transformations.

b) Nevertheless, the **invariance is not a sufficient condition for a rule to be meaningful**. Even if a method is invariant for the permissible transformations of a data type, this does not mean that the results are interpretable in terms of empirical systems [Luce et al, 1971, 1981, 1990; Pfanzagl J. 1971; Roberts F.S. et al, 1976]. However, for many practical tasks this strong interpretability is needed to obtain valuable knowledge.

c) The methods, which are invariant, satisfy a weaker condition of interpretability. For instance, we may be able to interpret relation “=” if we discover that $\mathbf{y} = \mathbf{f}(\mathbf{x}_1, \dots, \mathbf{x}_n)$ for data D, but this does not mean that we will also be able to interpret the function $\mathbf{f}$ itself. The function $\mathbf{f}$ can use non-interpretable relations and operations, but be invariant for permissible transformations. In other words, it is possible that we can not interpret function $\mathbf{f}$, but we are able to interpret “=” for values of this function. For example, in “black box” approaches such as neural networks, we can not interpret a particular network $\mathbf{f}$, but we can interpret that NN produced the same correct pattern for two different inputs.

Case 2. Physical data types for non-physical problems. Multidimensional data contain physical quantities, but the task is not from the physical area. The task may belong to finance, geology, medicine, and other areas. In this case, actual data types are not known even when they represent physical quantities. If the quantity is physical, then we know the empirical system

from the measurement theory. Nevertheless, the relations of that empirical system must be physically interpretable. If the task is physical also, then these relations are interpretable in the physics system of notions.

If the task is not physical, we need to check if relations from the empirical system are interpretable in that area. It may turn out that some of the relations can not be interpreted in that area. Then these relations need to be removed from the empirical system.

For example, for many physical quantities there is an empirically interpretable operation $\bullet$, matched to the formal numeric additive operation $+$ and its properties. However, the use of the same attribute in finance, medicine, and other fields **can change its data type**. For instance, there is no known medical procedure on a patient that gives us temperature t_3 from two patient's temperatures t_1 and t_2 , $t_3 = t_1 \bullet t_2$. Therefore, the operation $\bullet$ doesn't have interpretation in medicine.

On the other hand, the relation " $<$ " makes sense in both areas. Everyone knows the meaning of increasing temperature. It means that temperature data type in physics differs from temperature data type in medicine and we do not know exactly what the medical temperature data type (empirical system of temperature) is. The group of permissible transformations of temperature is not defined in medicine. Hence, we have no necessary criterion of meaningfulness of discovered knowledge. The situation in finance is similar. "**Overheating of the market**" is not measured by the physical temperature data type.

This consideration shows the need for **discovering and testing a data type** for a specific domain. MMDR method (Section 4.8) is able to do this. After the data type is discovered it can be used by variety of methods for rule discovery. What can data mining methods produce for the Case 2? Potentially the target value can be predicted correctly, but **without interpretation of the discovered decision rule**. Thus, the prediction task can be solved in Case 2, but a **regularity-discovering task** will not be solved. The interpretability of the results is critical for such tasks. We cannot use non-interpretable rules except for predictions. These rules can not be added to background domain knowledge.

Case 3: Non-physical data types for non-physical tasks. For non-physical quantities, data types are practically unknown. The situation is similar to the case when we use physical quantities in non-physical areas, because the groups of permissible transformations are unknown. Hence, we come to the same conclusion as in the previous case.

The example for this is shown below for financial applications. Let us try to identify the data type of the directional indicator $DI(t)$ for stock $S(t)$ for

date t used in [Von Altrock, 1997] and described in Chapter 7. Here $SMax$, $SMin$ and $SClose$ are max, min, and close values of the stock, respectively.

$$DI(t) = DM(t) / TR(t),$$

where

$$DM(t) = \begin{cases} SMax(t) - SClose(t-1), & \text{if } SMax(t) > SClose(t-1) \\ 0, & \text{if } SMax(t) < SClose(t-1) \end{cases}$$

and

$$TR(t) = \max(|SMax(t) - SMin(t)|, |SMax(t) - SClose(t-1)|, |SMin(t) - SClose(t-1)|)$$

$DM(t)$ produces today's highest stock price, $SMax(t)$ minus the closing price of last trade day, $SClose(t-1)$. The value is zero if the closing price of the last trade day is greater than today's highest price. Therefore, this formula is equal to zero if the stock does not grow, otherwise it shows the value of the growth.

The formula for $TR(t)$ (true range of stock) is computed as the largest of
 (a) the distance between today's high and today's low,
 (b) the distance between today's high and yesterday's close, or
 (c) the distance between today's low and yesterday's close.

Finally Directional Indicator, DI , shows relative last stock growth. Interpretable operations with DI are not clear. For instance, what does the sum mean for two DI s? However, comparison of different DI s makes sense. If $DI(v) < DI(w)$, then v has less evident growth than w from yesterday's close to today's current price. The simple example in Table 4.27 shows this. Here $DI(v) = 1/2$ and $DI(w) = 1$.

Table 4.27. Example of positive stock direction indicator DI

Date	Price Max	Price Min	Price Close	Max(t)-Close(t-1)	Max(t)-Min(t)	Min(t)-Close(t-1)	DM	TR	D I
v-1	36	31	32						
V	33	30	30	33-32=1	1	30-32=-2	1	2	1/2
W	35	30	33	35-30=5	35-30=5	30-30=0	5	5	1
W+1	35	30	34	35-30=5	35-30=5	30-31=0	5	5	1

This example shows that a hypothesis constructed using relation $DI(v) < DI(w)$, e.g.

IF $DI(v) < DI(w)$ THEN $PriceClose(v) < PriceClose(w)$

is much better interpreted and user-understood than a hypothesis such as

IF $(3*DI(v)+4*DI(w))>5$ THEN $PriceClose(v)<PriceClose(w)$.

Case 4. Nominal discrete data types (multidimensional data containing only discrete nominal data). In this case, all data are interpretable in empirical systems because there is **no difference between the numerical and empirical systems**. All numbers are only names, and names can be easily represented as predicates with one variable. The group of permissible transformations for nominal data type consists of all possible one-to-one transformations without restriction.

All methods working with these predicates are invariant relative to this group of permissible transformations. They also satisfy a stronger condition of meaningfulness of the methods – the interpretability in terms of empirical systems and the system of notions of domain knowledge.

Case 5. Non-quantitative and non-discrete data types. Multidimensional data contain no quantities and discrete variables, but do contain ranks, **orders** and other nonstandard data types. This case is similar to the third case. The only difference is that such data usually are made discrete using various calibrations without losing useful information.

Case 6. Mix of data types (multidimensional data containing mixture of various data). All mentioned difficulties arise in this case. There are several methods, which can work with some specific types of mixed data [Lbov et al, 1973; Manohin, 1976; Zagoruiko, Elkina, 1976; Mirkin, 1976].

To be able to work with all sorts of data type mixes, a new approach is needed. Relational data mining implements this approach using relational presentation of data types and the concept of empirical axiomatic theories. In particular, the MMDR method at first describes data types in relational form, and then discovers regularities using these predicates.

4.11. Empirical axiomatic theories: empirical contents of data

4.11.1. Definitions.

The concept of empirical axiomatic theory is a formal representation of the “**empirical content of data**”. This concept originates from the logic of

empirical theories [Pzelecki, 1969; Samokhvalov, 1973; Zagoruiko, Samokhvalov, et al, 1978] and from the critical analysis of representative measurement theory [Krantz et al, 1971, 1989, 1990].

Definition. **Empirical axiomatic theory** is a set M of four components:

$M = \langle \text{Obs}^V, V, W, S \rangle$, where

Obs^V is a **measurement procedure**,

$V = \{P_1, \dots, P_{n1}\}$ is a set of **empirical predicates** (we assume that the equality “=” belongs to V),

$W = \{Q_1, \dots, Q_{n2}\}$ is a set of **theoretical predicates**, where the predicates from W are idealizations of the empirical predicates from V , and

S is a **set of axioms** in the $V \cup W$.

The set of axioms S consists of axioms S^V, S^W for V and W and for mapping rules $S^{V \cup W}$ [Pzelecki, (1969)]. These rules may be derived from the domain knowledge of measurement procedure Obs^V and predicates from V . If there is no **mapping rule** between V and W , then actually there is theoretical knowledge and sets $W, S^W \cup S^{V \cup W}$ are empty. In this case, empirical axiomatic theory M consists of only three components:

$M = \langle \text{Obs}^V, V, S^V \rangle$.

Measurement procedure, Obs^V , **interprets** the empirical predicates from V . If this procedure is applied to a set of objects $A = \{a_1, \dots, a_m\}$ then a formal protocol pr^V of observations is produced. This protocol includes symbols for objects $a_1, \dots, a_m$, symbols of predicates (from V), and possibly some other symbols. It is assumed that measurement procedure Obs^V can be applied to any set of objects A . It can be done by introducing a third truth value (not defined) for the predicates from V . Next, for simplicity of consideration, it is assumed that Obs^V produces only one **protocol of observations** for a given A . Hence, the procedure Obs^V defines a mapping from the set of objects A to protocols: $\text{Obs}^V(A) = \text{pr}^V$.

The set of all formulas in the set V , which is true for all protocols of observation $\text{pr}^V = \text{Obs}^V(A)$ is called **empirical dependency**.

We say that an empirical axiomatic theory has an **empirical interpretation**, if all parts of that theory are interpretable in the domain theory (background knowledge): measurement procedure Obs^V , protocol of observations pr^V , predicates from V and W , and axioms S .

The concept of **empirical system** can be defined in terms of empirical axiomatic system as a non-reducible model [Pfanzagl, 1971] of the set of axioms S^W . This means that the model does not merge objects, which are different for predicates from W .

4.11.2. Representation of data types in empirical axiomatic theories

The first step of the analysis of empirical content of data consists of the representation of data in empirical axiomatic theories. Below are considered several known data types such as comparisons, binary matrices, matrices of orderings, matrices of proximity and attribute-based matrix. Moreover, empirical axiomatic theories are the most general representation of various data types. They represent well known data types, mixture of various data types, and data types that have no numerical representation at all.

Next, we review existing methods for processing these data types. This includes determination of assumptions of existing methods. Finally, we show that relational methods such as MMDR do not require restrictive assumptions about data types. Any data type can be accommodated and even discovered in training data. It is known that humans answer more precisely for qualitative and comparative questions than for quantitative questions. Therefore, the representation of these data types in axiomatic empirical theories is convenient.

Attribute-based matrix (table). Data D can be represented as an attribute-based matrix (x_{ij}) , $i = 1, \dots, m$; $j = 1, \dots, n$; where x_{ij} is the numerical value of j attribute on i -th object. Attributes may be qualitative and quantitative. The fact that numerical values of attributes exist means that there are n measurement procedures, which produce them. Let us denote these procedures as $x_j(a)$, $j = 1, \dots, n$, where a is an empirical entity. Then, $x_{ij} = x_j(a_i)$.

Attribute-value methods such as neural networks deal with this type of data. These methods are limited by the **assumption** that data types are stronger than interval and log-interval data types, which is not always true. For definitions of these data types, see Section 4.9.2.

Let us determine an empirical axiomatic system for attribute-based matrices. At first, a set of empirical predicates V_i for each attribute x_i needs to be defined. There are two cases:

1. The measurement procedure x_i is well known and an empirical system is known from measurement theory. Therefore, the set of all empirical predicates contains predicates given in V_i $i=1, \dots, n$.

2. An empirical system of the measurement procedure x_i is not completely defined. In this case, we have a measurement procedure, but we do not have an empirical system. The measurement procedure in the second case is called a **measurer**. The examples of measurers are psychological tests, stock market indicators, questionnaires, and physical measurers used in non-physical areas.

Let us define a set of empirical predicates V_i for measurer procedure x_i . For any numerical relation $R(y_1, \dots, y_k)$ in Re^k (Re - the set of all real numbers), we can define the following empirical relation on A^k .

$$P^R(a_1, \dots, a_k) \Leftrightarrow R(x_i(a_1), \dots, x_i(a_k)).$$

The measurer x_i obviously has an empirical interpretation, but relation P^R may not. We need to find such relations R that have empirical interpretations, i.e., relation $R(x_i(a_1), \dots, x_i(a_k))$ is interpretable in the terms of domain theory.

Suppose that $\{R_1, \dots, R_k\}$ is a set of the most common numerical relations and some (relations $P_j^{R_1}, \dots, P_j^{R_k}$) have an empirical interpretation. This set of relations is not empty, because at least the relation P_j^- (equivalence) has an empirical interpretation:

$$P_j^-(a_1, a_2) \Leftrightarrow x_j(a_1) = x_j(a_2).$$

In measurement theory, there are many sets of axioms based on just ordering and equivalence relations. Nevertheless, these sets of axioms establish **strong data types**. A strong data type is a result of **interaction of the quantities with individual weak data types** such as ordering and equivalence. For instance, having one weak order relation (for attribute y) and n equivalence relations

$$\{\leq_y, =_{x_1}, \dots, =_{x_i}, \dots, =_{x_n}\}$$

for attributes $x_1, \dots, x_n$, we can construct a **complex relation** between y and $x_1, \dots, x_n$ given by

$$G(y, x_1, \dots, x_n) \Leftrightarrow y = f(x_1, \dots, x_n),$$

where $f(x_1, \dots, x_n)$ is a polynomial [Krantz et al, 1971].

This is a very strong result. To construct a polynomial we need the sum operation, but this operation is not defined for $x_1, \dots, x_n$. However, relation G is equivalent to polynomial f if a certain set of axioms expressed in terms of order relation ($\leq_y$) presented above for y , and equality relations ($=$) are true for x_i .

This fundamental result serves as a critical **justification** for using ordering relations as a base for **generating relational hypotheses in financial applications** (Chapter 5). Ordering relations usually are empirically interpretable in finance.

Multivariate and pair comparisons [Torgerson, 1952, 1958, Shmerling D.S. 1978]. Consider set of objects $A = \{a_1, \dots, a_m\}$ and set of all tuples A^k of k objects from A . A group of n experts are asked to order objects in all tuples $\langle a_1, a_2, \dots, a_k \rangle$ from A^k in accordance with some preference relation.

Let $a_i^{ts_q}$ be an object i from tuple $\langle a_1, a_2, \dots, a_k \rangle$, where t is an entity to be evaluated, s is an expert and q is a preference rank given by an expert s to the entity t , $i = 1, \dots, m$; $s = 1, \dots, n$; $t = 1, \dots, C_m^k$; $q = 1, \dots, k$. The set of all ordered tuples is denoted by

$$R = \{ \langle a_{i1}^{ts_1}, a_{i2}^{ts_2}, \dots, a_{ik}^{ts_k} \rangle \}.$$

The typical goal of pair and multivariate comparison methods is to order all tuples. Known methods are based on some a priori assumptions [Torgerson, 1952, 1958; Shmerling D.S. 1978] which determine the areas of applicability.

Let us define for every expert s the preference relation

$$P_s(a_{i1}^{ts_1}, a_{i2}^{ts_2}) \Leftrightarrow i_1 < i_2.$$

Also, define the two equivalence relations $\sim$, $\sim_t$ and equivalence relation = by

$$a_{i1}^{ts_1} \sim a_{i2}^{ts_2} \Leftrightarrow i_1 = i_2,$$

$$a_{i1}^{ts_1} \sim_t a_{i2}^{ts_2} \Leftrightarrow t_1 = t_2, \text{ and}$$

$$a_{i1}^{ts_1} = a_{i2}^{ts_2} \Leftrightarrow \text{objects } a_{i1}^{ts_1}, a_{i2}^{ts_2} \text{ are the same.}$$

Therefore, we obtain a set of empirical predicates

$$V = \{=, \sim, \sim_t, P_1, \dots, P_n\}$$

and, thus, a data type represented with the set of empirical predicates V .

Matrix representation of binary relations. A binary relation $P(a,b)$ may be defined on the set of objects $A = \{a_1, \dots, a_m\}$ by the matrix (c_{ij}) , $i, j = 1, \dots, m$ where $c_{ij} = 1(0)$ means that relation $P(a_i, a_j)$ is true (false). Using such matrices, any binary relation on the set A can be defined. This representation of binary relations is widely used [Terehina, 1973; Tyrin et al 1977, 1979, 1981; Mirkin 1976, 1980; Drobishev 1980; Kupershtoh et al, 1976]. The most common binary relations are equivalence, order, quasi-order, partial order (see Section 4.9.1), and lexicographical orders.

Matrices of orderings. Let (r_{ij}) , $i = 1, \dots, m$; $j = 1, \dots, n$, where r_{ij} - rank of i -th object on j -th attribute, be a matrix of orderings. Any matrix can be either the matrix of orderings of m objects by n experts or the matrix of n different orderings on m objects. Methods of multidimensional scaling or

ranking [Tyurin et al, 1977, 1979, 1981; Kamenskii, 1977] consider these matrices.

Let us define relation P_j for an attribute j :

$$P_j(a_{i1}, a_{i2}) \Leftrightarrow r_{i1j} < r_{i2j},$$

where a_{i1}, a_{i2} are objects with numbers $i1, i2$ from $A = \{a_1, \dots, a_m\}$.

In this way we obtain the **set of empirical predicates** $V = \{P_1, \dots, P_n\}$. In these terms, protocol pr^V of observations of the predicates from V on the set of objects A is the relational structure

$$pr^V = \langle A, P_1, \dots, P_n \rangle.$$

Matrices of closeness. A matrix of closeness for set of objects

$$A = \{a_1, \dots, a_m\}$$

is the matrix (r_{ij}) , $i, j = 1, \dots, m$, where r_{ij} is a numerical estimate of **closeness** (similarity or difference) of objects i and j from A by an expert using an ordering scale.

These matrices are processed by multidimensional scaling methods [Holman, 1978, Satarov, Kamenskii, 1977, Tyurin et al, 1979, 1981]. They represent objects from set A as points in some metric space (Euclidean or Riemannian). This space is a space of minimal dimension which preserves original distances $\{t_{ij}\}$ between objects in the original space as much as possible. These methods have general limitations. There is no criterion for matching a data matrix and a multidimensional scaling method, and some matrices of closeness can not be embedded in a metric space. Thus, we can consider these embedded data as **an attribute-based matrix**. Let us represent the matrix of closeness in relational terms of **empirical axiomatic theories**.

Let $P(a_{i1}, a_{i2}, a_{i3}, a_{i4}) \Leftrightarrow r_{i1i2} < r_{i3i4}$. This relation is defined on entire set A . Hence, the **protocol** pr^V in the set of predicates $V = \{P\}$ will be the relational system

$$pr^V = \langle A, V \rangle.$$

In measurement theory such **empirical systems** are denoted by $M = \langle A^*; \leq \rangle$, where $A^* \subset A \times A$, and $\leq$ is a binary ordering relation defined on A^* . Let us describe some results from measurement theory related to such empirical systems.

Scale of positive differences [Krantz et al 1971,p.147]. There exists a mapping

$$\phi: A^* \rightarrow \mathbb{R}, A \neq \emptyset,$$

such that for every pair (a,b), (b,c), (c,d) from A^* ,

- a) $(a,b) \leq (c,d) \Leftrightarrow \phi(a,b) \leq \phi(c,d)$, and
- b) $\phi(a,c) = \phi(a,b) + \phi(b,c)$.

This scale means that for an empirical ordering relation for pairs of empirical objects, we can find a numerical function ϕ with an interpretable sum operation (+) on its values. The difference between a and c can be obtained as the sum of differences between (a, b) and (b,c). See property b).

The sum is interpretable because differences (a,c), (a,b) and (b,c) are interpretable. It is important to mention that original differences (a,c), (b,c) and (a,c) have no numeric values. We only know from the ordering relation that, for instance, $(a,b) \leq (c,d)$. if there are two pairs of stocks and an expert expresses his/her opinion that

$$(\text{Microsoft, Intel}) \leq (\text{Microsoft, Sun}),$$

i.e., from his/her viewpoint behavior of the Microsoft stock is closer to behavior of the Intel stock than to behavior of the Sun stock. Figure 4.2 shows actual data for these three companies for July 99 normalized by the max stock prices for that month.

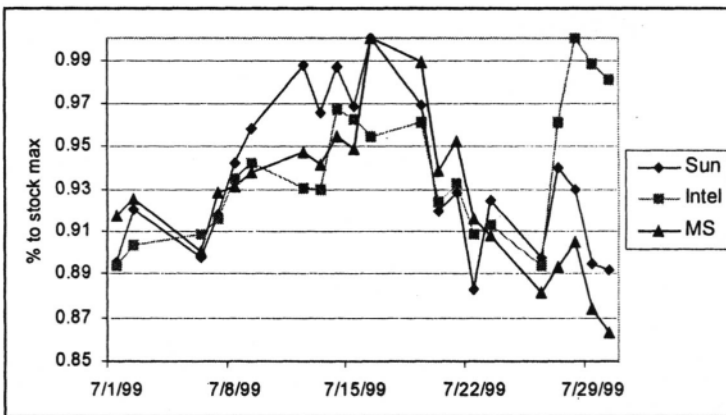


Figure 4.2. Normalized stock data

There is no numeric measure of closeness between stocks yet. It is convenient to introduce a numeric measure of closeness ϕ between stocks. We may wish to introduce ϕ such that, for instance,

$$\phi(\text{Microsoft, Sun}) = \phi(\text{Microsoft, Intel}) + \phi(\text{Intel, Sun}).$$

If we are able to find such measure of closeness ϕ consistent with the ordering relation ($\leq$) given by an expert, then we can use the sum operation (+) in **numeric data mining methods** for discovering regularities.

This sum will be interpretable in contrast with other functions ϕ , which can be invented. An interpretable mapping ϕ is a **homomorphism**. For the formal definition of homomorphism, see Section 4.9.3. We can also view homomorphism ϕ as a way of **transforming a numeric presentation** to a relational presentation for using **relational data mining methods** in discovering regularities.

For a scale of positive differences, a mapping ϕ is unique within a positive numerical multiplier q , i.e., any other interpretable mapping ϕ' can be obtained from ϕ by $\phi' = q\phi$. This means that ϕ is measured in the **ratio scale**.

Scale of algebraic differences [Krantz et al, 1971, p.151].

Let $A^* = A \times A$. There exists a homomorphism

$\phi: A \rightarrow \mathbb{R}$ such that for every $a, b, c, d \in A$:

$$(a, b) < (c, d) \Leftrightarrow (\phi(a) - \phi(b)) < (\phi(c) - \phi(d)), \text{ for } (a, b), (c, d) \in A^*.$$

This representation is unique within the log-linear transformations (**log-interval scale**).

Scale of equal finite intervals [Krantz et al, 1971, p.168].

Let $A^* = A \times A$, A is finite set, $A^* \neq \emptyset$. There exists a homomorphisms $\phi: A \rightarrow \mathbb{N}$ ($\mathbb{N}$ – natural numbers), such that for every $a, b, c, d \in A$:

$$(a, b) \leq (c, d) \Leftrightarrow \phi(a) - \phi(b) \leq \phi(c) - \phi(d), \text{ for } (a, b), (c, d) \in A^*.$$

This representation is unique within the linear transformations (**interval scale**). Similarly, other data types can be converted into the relational form of axiomatic empirical systems.

4.11.3. Discovering empirical regularities as universal formulas

In this section, we consider the possibility of developing a rather general method of rule discovery in first order languages. The most general ap-

proach would operate with all possible hypotheses in the first order language. This is impractical for many reasons. We favor two other alternative approaches:

- restricting the set of hypotheses by formulas with universal quantifiers called **universal formulas** (see Section 4.4.3 for definitions) and
- probabilistic regularities described in Section 4.8 with the MMDR method.

We consider only universal formulas for expressing empirical axioms assuming that these axioms do not contain existential quantifiers and functions (see Section 4.4.3 for definitions). Only empirical axioms can be tested on real data. If empirical axioms are expressed with existential quantifiers, then these quantifiers can be eliminated using Skolem functions [Mal'tsev, 1970]. Processing of other axioms is described in [Krantz, et al 1990]. Along with **empirical axioms**, measurement theory also uses **technical axioms** and **idealized axioms** [Krantz et al 1990, p.251] to simplify numerical representation of data.

At this point, we determine empirically interpretable properties of a measurement procedure Obs^V . These interpretable properties of Obs^V give us the criteria for discovering universal formulas. One of these properties is **inheritance** of observations as defined below. It will be shown that the inheritance of observations is the **necessary and sufficient condition** for expressing empirical dependencies by **universal formulas**.

Let us specify two concepts: measurement procedure Obs^V and the **protocol of observations** pr^V . The protocol of observations is a relational system or model [Mal'tsev, 1970] $\text{pr}^V = \langle A; V \rangle = \text{Obs}^V(A)$, where $A = \{a_1, \dots, a_m\}$ is a set of objects and $V = \{P_1, \dots, P_n\}$ is a set of empirical predicates. We also assume the standard definition of truth for logical expressions on relational system pr^V [Mal'tsev, 1970]. A protocol of observations pr^V can be viewed as **data** D . The set of all formulas in the set V which are true on all protocols of observation $\text{pr}^V = \text{Obs}^V(A)$ is called a set of **empirical dependencies**. Let us show that the inheritance of observational results is the necessary and sufficient condition of expressing empirical dependencies by universal formulas.

We use several formal concepts:

- an **empirical axiomatic theory** $M = \langle \text{Obs}^V, V, W, S \rangle$ (section 4.10.1)
- a set of all finite relational systems $\text{PR} = \{\text{pr}^V\}$, where each relational system $\text{pr}^V = \langle A; V \rangle$ is a result of applying **measurement (observation) procedure** Obs^V to a set of objects A , $\text{pr}^V = \langle A; V \rangle = \text{Obs}^V(A)$, and
- an abstract class T of all finite relational systems of the type $\text{pr}^V = \langle A; V \rangle$.

In these logical terms, our purpose is to find necessary and sufficient conditions of axiomatizability the set PR in class T .

The set PR is called a **universal axiomatizable (UA)** in class T, if there exists a set of universal formulas W, which are true only on relational systems from T, which belong to PR. In that case, the set W will be a system of axioms for our observation procedure $\mathbf{Obs}^V$, producing the set PR.

Theorem 1 (Los'-Tarski) [Mal'tsev, 1970]. Class PR is UA class in class T if and only if PR is locally close in class T.

The mathematical property of local closeness is difficult to interpret, therefore, another condition of universal axiomatizability should be found.

Definition 1 [Mal'tsev, 1970]. Subclass PR is called inherited in class T if any relational subsystem in class T of some relational system from PR belongs to PR.

Proposition 1 [Mal'tsev, 1970]. For the classes of finite relational systems PR and T, the local closeness of class PR in class T follows from the inheritance of class PR in class T.

Definition 2. The **inheritance of observational procedure**: for every protocol $\mathbf{pr}_A^V = \langle A; V \rangle = \mathbf{Obs}^V(A) \in \mathbf{PR}$ and any subset $B \subset A$ the protocol $\mathbf{pr}_B^V = \langle B; V \rangle = \mathbf{Obs}^V(B) \in \mathbf{PR}$ is a subsystem of the protocol $\mathbf{pr}_A^V$.

The empirical interpretation of this definition is that: the truth-values of the relations from V in the protocol $\mathbf{pr}_A^V$ depend only on the set of objects $\mathbf{Ob} \subset A$. Set Ob is the set where the relations are defined. For physical experiments, this property is obviously fulfilled. It explains why most physical laws are universally axiomatizable. Nevertheless, if we consider human responses for a set of stimuli Ob from A, we see that answers depend on the remaining stimuli from $A \setminus \mathbf{Ob}$.

Theorem 2 [Vityaev, 1983]. If observational procedure $\mathbf{Obs}^V$ satisfies the inheritance property, then empirical dependencies are universally axiomatizable.

Chapter 5

Financial Applications of Relational Data Mining

Logic is a system whereby one may go wrong with confidence.

Charles F. Kettering

5.1. Introduction

This chapter is devoted to discovering regularities in financial time series combining mathematical logic and probability theory. Specifically the relational data mining method MMDR (Section 4.8) and the related "Discovery" software system [Vityaev et al., 1992, 1993; Kovalerchuk, Vityaev, 1998] are used. Discovered regularities were used to forecast the target variable, representing the relative difference in percent between today's closing price and the price five days ahead, along with next day's forecast. Below we describe types of regularities found and analyzed, statistical characteristics of these regularities on the training and test data, and the percentage of true and false predictions on the test data. There are more than 130 discovered regularities in 1985-1994 data. The best of these regularities indicates about 75 % correct forecasts in 1995-1996 test data. The target variable (a specific stock data provided by the Journal of Computational Intelligence in Finance) was predicted using separately SP500C (S&P 500 trading day close) and the target variable's own history. Active trading strategy based on discovered rules, outperformed buy-and-hold strategy and strategies based on several other models in simulated trading for 1995-1998. A separate computational experiment was conducted for forecasting SP500 for comparison with other methods.

To the best of our knowledge, this is the **first financial application of relational data mining** and, in particular, for the analysis of SP500C and

other stock market data. In Chapter 6, these results are compared with the performance of other methods: ARIMA, FOIL, backpropagation neural networks, decision trees, linear adaptive methods, buy-and-hold and risk-free investment.

Most of these methods excluding FOIL are attribute-based data mining methods. As has been discussed, these methods are relatively simple, efficient, and can handle noisy data. However, (1) the restricted form of the **background knowledge** and (2) the **lack of relations** limit possibilities of applying these methods in some domains. Relational Data Mining such as ILP methods overcome these limitations, but currently have difficulty **handling numerical data and processing large data and predicate sets** [Bratko, Muggleton, 1995].

The relational method **MMDR** handles a wide range of numeric data, in particular, the relative difference in percent between today's closing price and the price five days ahead. The Standard and Poor's 500 close (SP500C) is also used as a direct target variable with some additional features:

- weekday (indication of a particular day of the week, i.e., Monday, Tuesday, Wednesday, Thursday, Friday) for each value of studied variables, and
- the first and second differences for variables (prices and SP500C and DJIA indexes) for various weekdays, which are similar to first and second derivatives.

All this information was transformed into a first order logic presentation with probabilities on logical expressions, as described below.

Traditionally Inductive Logic Programming is used for classification tasks, which includes:

- a representation of the positive and negative **examples**, and
- relevant **background knowledge** in the forms of predicates (logical relations with variables).

Numerical prediction of values of financial time series is not a classification task, therefore, it should be described differently in terms of predicates. It involves development of a good **representation** of the entire time series in predicate terms, together with relevant **background knowledge** about this time series. Often relevant background knowledge covers the entire time series. Therefore, generally we do not need to distinguish these two components. However, we need to distinguish sets of **hypotheses about the regularities** in the time series in predicate terms and a **standard numerical description** of the time series. The second part may require some traditional preprocessing. The first part is the most innovative in this study. It requires **inventing both predicates and hypotheses** in terms of these predicates.

These predicates and hypotheses are developed for financial series and described in this chapter (Sections 5.2 and 5.7).

In Section 5.2, all hypotheses are combinations of relations (predicates)

$$P(x,y)=\text{True} \Leftrightarrow t(x) \geq t(y),$$

where $t(x)$ and $t(y)$ are values of the time series, or their absolute or relative differences for dates x and y . Thousands of these hypotheses are tested for discovering “probabilistic laws” in computational experiments.

5.2. Transforming numeric data into relations

Variables. Two time series TR (**training set**) and CT (**control/test set**) of the target variable are used to train and evaluate a forecasting algorithm, where

$$\text{TR} = \{a_1, \dots, a_{tr}\}$$

is ten years of data (1985-1994, $tr = 2528$ trading days) and

$$\text{CT} = \{a_1, \dots, a_{ct}\}$$

is two years of data (1995-1996, $ct = 506$ trading days).

Five sequential days are used as the main forecast unit (an **object**)

$$\mathbf{a}_t = (a_t^1, a_t^2, a_t^3, a_t^4, a_t^5),$$

where $\mathbf{a}_t^j$ is j -th day of the five-day object $\mathbf{a}_t$. We also use another notation

$$\mathbf{a}_t = (a_t, a_{t+1}, a_{t+2}, a_{t+3}, a_{t+4})$$

with the correspondence between notations

$$a_{(t-1)+j} = a_t^j$$

for all five days of $\mathbf{a}_t$ ($j=1,\dots,5$). Actually, **index** t indicates the **first day** of a five-day object. A **Weekday**(a_t) **function** has five values: 1,2,3,4,5, where **Weekday**(a_t)=1 indicates that day a_t is Monday and **Weekday**(a_t)=5 indicates a_t is Friday. For instance, IF $a_t = \text{“March 3, 1998”}$, THEN **Weekday**(a_t)=2, i.e., Tuesday. We do not consider Saturdays and Sundays and holidays in this study, because the stock market is closed these days.

Several sets of variables were generated from SP500C.

Set 1. First relative differences:

$$\Delta_{ij}(a_i) = (SP500C(a_i^j) - SP500C(a_i^1)) / SP500C(a_i^1), i < j, i, j = 1, \dots, 5.$$

This variable represents the difference between SP500C for the i -th and the j -th days, normalized by SP500C for the i -th day.

Example. Let $i=1, j=2, t = \text{"March 3, 1998"} ,$ then

$$a_i = \langle \text{March 3, 1998, March 4, 1998, March 5, 1998, March 6, 1998, March 9, 1998} \rangle,$$

where

$$\begin{aligned} a_i &= a_i^1 = \text{"March 3, 1998"}, a_{i+1} = a_i^2 = \text{"March 4, 1998"}, \\ a_{i+2} &= a_i^3 = \text{"March 5, 1998"}, a_{i+3} = a_i^4 = \text{"March 6, 1998"}, \\ a_{i+4} &= a_i^5 = \text{"March 9, 1998"}. \end{aligned}$$

Therefore,

$$\begin{aligned} \Delta_{12}(a_i) &= (SP500C(a_i^2) - SP500C(a_i^1)) / SP500C(a_i^1) \\ &= \frac{SP500C(\text{March 4, 1998}) - SP500C(\text{March 3, 1998})}{SP500C(\text{March 3, 1998})}. \end{aligned}$$

Set 2. Differences between two relative differences:

$$\Delta_{ijk}(a_i) = \Delta_{jk}(a_i) - \Delta_{ij}(a_i)$$

This difference is based on previous relative differences.

Example. Let $k=3$, then $\Delta_{ijk}(a_i) = \Delta_{jk}(a_i) - \Delta_{ij}(a_i)$ can be written as

$$\begin{aligned} \Delta_{123}(a_i) &= \frac{SP500C(\text{March 5, 1998}) - SP500C(\text{March 4, 1998})}{SP500C(\text{March 4, 1998})} \\ &\quad - \frac{SP500C(\text{March 4, 1998}) - SP500C(\text{March 3, 1998})}{SP500C(\text{March 3, 1998})}. \end{aligned}$$

Set 3. Cyclic permutations $\{\pi\}$ of length 5 for object a and function $wd(a)$. Function $wd(a)$ maps five calendar days to five weekdays.

For instance,

$$\mathbf{wd}(a) = \langle 1, 2, 3, 4, 5 \rangle$$

means that a represents normal five weekdays from Monday to Friday, and

$$\mathbf{wd}(b) = \langle d_1, \dots, d_5 \rangle = \langle 2, 3, 4, 5, 1 \rangle = \langle \text{Tue, Wed, Thu, Fri, Mon} \rangle$$

shows that five-day object b begins from Tuesday to next Monday as the last day of b . Using permutation π we can transform sequences of days. For example,

$$\pi(\text{Mon, Tue, Wed, Thu, Fri}) = (\text{Tue, Wed, Thu, Fri, Mon}) = \langle d_1, d_2, d_3, d_4, d_5 \rangle.$$

Thus, π is a cyclic permutation, which changes the set of weekdays $\langle d_1, d_2, d_3, d_4, d_5 \rangle$ under consideration for rule discovery when considering pairs a and b . Formally, the vector function

$$\mathbf{wd}(b) = \langle d_1, \dots, d_5 \rangle$$

is equivalent to the expression

$$(\text{Weekday}(b^1) = d_1) \ \& \ (\text{Weekday}(b^2) = d_2) \ \& \dots \ \& \ (\text{Weekday}(b^5) = d_5)$$

with the scalar function **Weekday**(b^i) defined at the beginning of this section. Note that it is possible to generate hundreds of similar variables in addition to Sets 1-3. In the experimental study below, we use variables of types 1-3 for SP500C, their analogues for the target and DJIA.

The first two variables catch properties similar to the first and second derivatives of the original time series. Analysis of more variables requires more runtime. The goal of the current study is primarily to show the applicability of the method and its capability as a knowledge acquisition tool for financial time series.

5.3. Hypotheses and probabilistic “laws”

The next step in formulating hypotheses to be tested is to find probabilistic laws. The concept of a probabilistic law was defined in Chapter 4. Let's introduce a simplified frame notation for any five-day objects a and b and their relations, omitting indices:

$$(\Delta(a) \leq \Delta(b))^e$$

is **any of inequalities** such as

$$(\Delta_{ij}(a) \leq \Delta_{ij}(b))^e, (\Delta_{ijk}(a) \leq \Delta_{ijk}(b)), i < j < k; i, j, k = 1, \dots, 5,$$

where $e \in \{0, 1\}$, and $e=1$ means that the expression $\Delta_{ij}(a) \leq \Delta_{ij}(b)$ is not negated, and $e=0$ means negation of this expression, i.e.,

$$\Delta_{ij}(a) > \Delta_{ij}(b).$$

Also symbols $e0, e1, e3, \dots, ek$ are used with e if there are more than one expression under consideration.

The following sets of hypotheses, H1-H4, are tested to find probabilistic regularities ("laws").

Set of Hypotheses H1:

$$(wd(a) = wd(b) = \langle d_1, \dots, d_5 \rangle) \& (\Delta(a) \leq \Delta(b))^{e1} \rightarrow ((target(a^5) \leq target(b^5))^{e0};$$

where $e1 = 0$ means that the relation $\Delta(a) \leq \Delta(b)$ is not negated and $e0=1$ means negation of the relation $target(a^5) \leq target(b^5)$, i.e.,

$$target(a^5) > target(b^5).$$

Example: Let **a** and **b** are two five-day objects from March, 1998:

$$a = \langle \text{March 3; March 4, March 5, March 6, March 9} \rangle,$$

$$b = \langle \text{March 10, March 11, March 12, March 13, March 16} \rangle.$$

Let also

$$\Delta(a) = \Delta_{12}(a_t), \quad \Delta(b) = \Delta_{12}(b_t) \quad \langle d_1, \dots, d_5 \rangle = \langle \text{Tue, Wed, Thu, Fri, Mon} \rangle,$$

with March 3, 1998 as 3.3.98. We use similar notation for other days. Therefore, the tested rule/hypothesis in this example is

$$\begin{aligned} & [wd(3.3.98, 3.4.98, 3.5.98, 3.6.98, 3.9.98) \\ & = wd(3.10.98, 3.11.98, 3.12.98, 3.13.98, 3.16.98) \\ & = \langle \text{Tue, Wed, Thu, Fri, Mon} \rangle] \& (\Delta(a) \leq \Delta(b)) \rightarrow target(a^5) > target(b^5). \end{aligned}$$

This means testing all five-day objects beginning on Tuesday. The tested statement is

IF for any five-day objects **a** and **b** beginning from Tuesday,
the SP500C difference $\Delta_{12}(a_t)$ is smaller than $\Delta_{12}(b_t)$,
THEN the target stock for the last day of **a** will be greater than for the
last day of **b**.

Set of Hypotheses H2.

$[wd(a) = wd(b) = \langle d_1, \dots, d_5 \rangle] \ \& \ [\Delta(a) \leq \Delta(b)]^{e1} \ \& \ [\Delta(a) \leq \Delta(b)]^{e2}$
 $\rightarrow [target(a^5) \leq target(b^5)]^{e0};$

This set of hypotheses has a similar interpretation. The only difference from H1 is that now we consider two differences in the rules. For example, one of the tested statements is

IF for any five-day objects **a** and **b** with weekdays $\langle d_1, \dots, d_5 \rangle$,
the SP500C difference $\Delta_{12}(a_t)$ is smaller than $\Delta_{12}(b_t)$ AND the
SP500C difference $\Delta_{23}(a_t)$ is greater than $\Delta_{23}(b_t)$,
THEN the target stock for the last day of **a** will be greater than for the
last day of **b**.

Set of Hypotheses H3.

$[wd(a) = wd(b) = \langle d_1, \dots, d_5 \rangle] \ \& \ [\Delta(a) \leq \Delta(b)]^{e1} \ \& \ [\Delta(a) \leq \Delta(b)]^{e2} \ \& \ [\Delta(a) \leq \Delta(b)]^{e3}$
 $\rightarrow [target(a^5) \leq target(b^5)]^{e0}.$

These hypotheses have a similar interpretation. The only difference from H2 is that now we consider three differences in the rules. For example, one of the tested statements is

IF for any five-day objects **a** and **b** with weekdays $\langle d_1, \dots, d_5 \rangle$, the SP500C
difference $\Delta_{12}(a_t)$ is smaller than $\Delta_{12}(b_t)$
AND the SP500C difference $\Delta_{23}(a_t)$ is greater than $\Delta_{23}(b_t)$
AND the SP500C difference $\Delta_{123}(a_t)$ is greater than $\Delta_{123}(b_t)$,
THEN the target stock for the last day of **a** will be greater then for the last
day of **b**.

Set of hypotheses H4.

$$[wd(a) = wd(b) = < d_1,...,d_5>] \& [\Delta(a) \leq \Delta(b)]^{e1} \& ... \& [(\Delta(a) \leq \Delta(b))]^{ek} \\ \rightarrow [target(a^5) \leq target(b^5)]^{e0}.$$

These hypotheses allow us generate more than three relations including $\Delta_{ijk}(a_i)$. For example, one of the tested statements is

IF for any five-day objects **a** and **b** with weekdays $<d_1,...,d_5>$,
the SP500C difference $\Delta_{12}(a_i)$ is smaller than $\Delta_{12}(b_i)$
AND the SP500C difference $\Delta_{23}(a_i)$ is greater than $\Delta_{23}(b_i)$
AND the SP500C difference $\Delta_{123}(a_i)$ is greater than $\Delta_{123}(b_i)$
AND ...
THEN the target stock for the last day of **a** will be greater then
for the last day of **b**.

Table 5.1 shows examples of hypotheses H1-H4 in the usual financial terms.

Table 5.1. Example of rule consistent with hypotheses H1-H4

IF
<i>Current 5 days end on Monday and there are some other ("old") five days (from the history of years 1984-1996) that end on Monday too</i>
AND
<i>the relative SP500C difference between Tuesday and Thursday for the old five days is no greater than between Tuesday and Thursday for the current five days</i>
AND
<i>the relative SP500C difference between Tuesday and Monday for the old five days is greater than between Tuesday and Monday for the current five days</i>
AND
<i>the relative difference between SP500C differences for Tuesday, Wednesday and Wednesday, Thursday) for the old five days is no greater than for the pairs of days for the current five days</i>
AND $<$ we omit linguistic description of $(\Delta_{245}(a) > \Delta_{245}(b))$, which is similar to previous one $>$
THEN
<i>the target value for Monday from the current 5 days should be no greater than the target value for the Monday from the old five days, i.e., we forecast that a target stock five days ahead from the current Monday will grow no greater than it was five days ahead from the old Monday.</i>

5.4. Markov chains as probabilistic “laws” in finance

Many well-known prediction methods used for stock market study can be written in terms similar to H1-H4. Markov chains and other methods ex-

exploiting conditional probabilities (transition probabilities) exemplify these methods. Two simple financial Markov chains are shown in Chapter 3 in Tables 3.16 and 3.17. Figure 5.1 illustrates rules from Table 3.16, e.g.,

IF the stock increased yesterday,
THEN the stock increases today with probability 0.7.

Similarly, Markov chain 2 presented in Table 3.17 produces 16 rules such as

IF the stock increases today and decreases yesterday,
THEN the stock will increase tomorrow with probability 0.6.

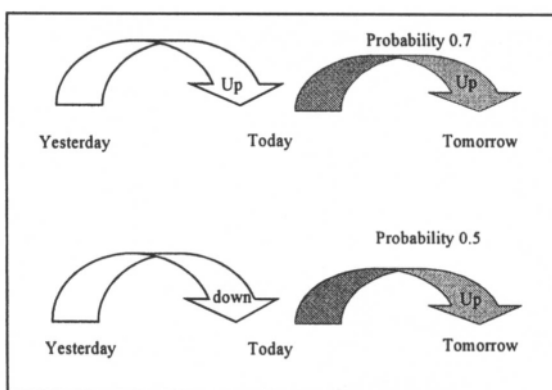


Figure 5.1. Example of Markov chain rules

This type of models can be embedded into first-order logic rules and discovered using this technique. Expressions (hypotheses) H1-H4 are evaluated on training and test data using conditional probabilities. **Six-day objects** are used instead of five-day objects:

$\langle d_1, \dots, d_5, d_6 \rangle = \langle \text{Mon, Tue, Wed, Thu, Fri, Mon} \rangle,$

$(wd(a) = wd(b) = \langle d_1, \dots, d_5, d_6 \rangle, a = a_t, a_t^6 = a_{t+1}^1 = b_t^1,$

i.e., a is some six days and b is the next six days excluding Saturday and Sunday overlapping the end of a and the beginning of b . Next the first relative difference for the same target stock price (S) is generated:

$$\Delta_{ij}(a_t) = (S(a_t^j) - S(a_t^i)) / S(a_t^i).$$

This variable is equal to $\text{target}(\mathbf{a}_t)$ five days earlier. The target represents a five-day forecast in contrast with $\Delta_{ij}(\mathbf{a}_t)$ representing the current dynamics of the stock.

Example. Suppose that the following conditional probabilities are computed on the training set TR:

- 0.31 for *Rule1*: $(\Delta_{ij}(\mathbf{a}_t) < \Delta_{ij}(\mathbf{a}_{t+1}) \rightarrow (\text{target}(\mathbf{a}_t^6) < \text{target}(\mathbf{a}_{t+1}^6)),$
 0.69 for *Rule2*: $(\Delta_{ij}(\mathbf{a}_t) < \Delta_{ij}(\mathbf{a}_{t+1}) \rightarrow \neg (\text{target}(\mathbf{a}_t^6) < \text{target}(\mathbf{a}_{t+1}^6)),$
 0.65 for *Rule3*: $\neg (\Delta_{ij}(\mathbf{a}_t) < \Delta_{ij}(\mathbf{a}_{t+1}) \rightarrow (\text{target}(\mathbf{a}_t^6) < \text{target}(\mathbf{a}_{t+1}^6)),$
 0.35 for *Rule4*: $\neg (\Delta_{ij}(\mathbf{a}_t) < \Delta_{ij}(\mathbf{a}_{t+1}) \rightarrow \neg (\text{target}(\mathbf{a}_t^6) < \text{target}(\mathbf{a}_{t+1}^6)).$

The symbol “ $\neg$ ” is used for negation as usual. These rules can be represented with a matrix of transition probabilities used in Markov chains for forecasting:

		Target	
		0	1
Delta (Δ)	0	0.31	0.69
	1	0.65	0.35

Here, 1 denotes “up” for target and delta (Δ), i.e.,

$$(\Delta_{ij}(\mathbf{a}_t) < \Delta_{ij}(\mathbf{a}_{t+1}), (\text{target}(\mathbf{a}_t^6) < \text{target}(\mathbf{a}_{t+1}^6)),$$

respectively. Similarly, 0 denotes “down” for target and Δ , i.e.,

$$(\Delta_{ij}(\mathbf{a}_t) > \Delta_{ij}(\mathbf{a}_{t+1}) \text{ and } (\text{target}(\mathbf{a}_t^6) > \text{target}(\mathbf{a}_{t+1}^6)).$$

For simplicity, we ignore cases with $\Delta_{ij}(\mathbf{a}_t) = \Delta_{ij}(\mathbf{a}_{t+1})$ and $\text{target}(\mathbf{a}_t^6) = \text{target}(\mathbf{a}_{t+1}^6)$. To incorporate this, an additional state and a larger table with three rows and three columns will be needed. In this way, refined probabilistic rules can be discovered:

- IF $\Delta_{ij}(\mathbf{a}_t) = \Delta_{ij}(\mathbf{a}_{t+1})$, THEN $(\text{target}(\mathbf{a}_t^6) < \text{target}(\mathbf{a}_{t+1}^6))$ with probability 0.65.
 IF $\Delta_{ij}(\mathbf{a}_t) = \Delta_{ij}(\mathbf{a}_{t+1})$, THEN $(\text{target}(\mathbf{a}_t^6) > \text{target}(\mathbf{a}_{t+1}^6))$ with probability 0.30.
 IF $\Delta_{ij}(\mathbf{a}_t) = \Delta_{ij}(\mathbf{a}_{t+1})$, THEN $(\text{target}(\mathbf{a}_t^6) = \text{target}(\mathbf{a}_{t+1}^6))$ with probability 0.05.

Rule 2 can be described using the usual language:

IF delta goes up, THEN target goes down probability 0.69.

Several of these expressions were used to study a forecast horizon for consecutive days and weeks by changing $\langle d_1, \dots, d_k \rangle$ and the i, j days, where $\langle d_1, \dots, d_k \rangle$ is extended from 5 days to 12 weeks.

5.5. Learning

Scheme. An estimate of conditional probability, $P(A_0/A_1 \& \dots \& A_k)$, for each probabilistic law

$$C = (A_1(x, y, \dots, z) \& \dots \& A_k(x, y, \dots, z) \rightarrow A_0(x, y, \dots, z))$$

is computed using training data. Remember that all expressions A_i in first-order rules depend on variables $x, y, \dots, z$, i.e., $A_i = A_i(x, y, \dots, z)$ in contrast with propositional logic expressions, which do not have variables $x, y, \dots, z$.

Values of conditional probabilities are used as **evaluation functions** combined with a test for their statistical significance. This is a relatively common way of designing an evaluation function. The relative frequency is used in the AQ method and statistical significance is evaluated in the CN2 method, but for entropy [Mitchell, 1997]. The MMDR method exploits an original search mechanism to select appropriate expressions (Chapter 4) using mentioned evaluation criteria: conditional probability $P(A_0/A_1 \& \dots \& A_k)$ and Fisher statistical significance test. This search is in line with “current-best-hypothesis” search and “least-commitment” search [Russel and Norvig, 1995], but it is applied for probabilistic hypotheses which are more complex. Specifically, the search was arranged in accordance with a definition of semantic probabilistic inference (Section 4.8.2). After finding several nested probabilistic financial “laws”

$$C_{k-1} \triangleright C_{k-2} \triangleright \dots \triangleright C_2 \triangleright C_1,$$

the search for a new one is done by adding to the If-part of the rule C_{k-1} , a new **atomic logical expression** $(\Delta(a) \leq \Delta(b))^c$. This addition is also known as **specialization** [Russel and Norvig, 1995]. We find mentioned a new logical expression using the search of logical expressions H1-H4.

The Fisher F-statistic (see Section 4.8.3, [Kendall, Stuart, 1977; Bovas, Ledolter, 1983]) was used on each step to test statistically whether each generated hypothesis H1-H4 is a probabilistic “law”. This test requires to

delete some atom and to test if the remainder of the expression has less conditional probability. If the conditional probability decreases with statistical significance (with a certain level of confidence), then the tested hypothesis as a **probabilistic “law”** is accepted. This resembles the idea of finding a generalization of a hypothesis [Russel and Norvig, 1995].

Sets of hypotheses H1-H4 are tested using the training set $TR = \{a_1, \dots, a_n\}$ and randomly chosen pairs of objects **a**, **b** from TR by the software "Discovery" system. To test hypotheses all sequential pairs of objects from TR are used. The result of learning is a set **Law** of possible probabilistic “laws” found on TR. Each of these probabilistic “laws” was described with its conditional probability on TR.

To test if a “law” is stable, its conditional probability on the control set CT is evaluated. However, we did not use these conditional probabilities to choose preferred “laws” for forecasting. Therefore, the independence of the test is preserved.

Examples of discovered rules (“laws”). Let us give three examples of laws with relatively high conditional probabilities for both training and test/control sets TR and CT:

Example 1.

$$\begin{aligned} & [wd(a) = wd(b) = \langle 2, 3, 4, 5, 1 \rangle] \& (\Delta_{13}(a) \leq \Delta_{13}(b)) \& \\ & [\Delta_{15}(a) > \Delta_{15}(b)] \& [\Delta_{234}(a) \leq \Delta_{234}(b)] \& [\Delta_{245}(a) > \Delta_{245}(b)] \\ & \rightarrow target(a^5) \leq target(b^5). \end{aligned}$$

For this rule, we obtained frequency on TR equal to 0.64 and frequency on CT equal to 0.76.

This “law” can be translated to the normal financial language (see Table 5.1). That statement is true only statistically as it reflects frequencies: the frequency on TR equals 0.64 and the frequency on CT equals 0.76. This means that for **about 70%** of those cases, we have found an upper limit for the target value, which is the target value for the old (previous) Monday.

Let us suppose that the last target value is -3%, i.e., the closing price for the old five days decreased. This means that one has a decrease from the current Tuesday to the current Monday and the amount of decrease will be greater than -3% with probability 0.7. For example, it can be -5%.

We present the next two examples without a linguistic description.

Example 2.

$$(wd(a) = wd(b) = \langle 2, 3, 4, 5, 1 \rangle) \& (\Delta_{24}(a) \leq \Delta_{24}(b)) \& (\Delta_{145}(a) \leq \Delta_{145}(b)) \&$$

$$(\Delta_{234}(a) > \Delta_{234}(b)) \& (\Delta_{235}(a) \leq \Delta_{235}(b)) \rightarrow (\text{target}(a^5) > \text{target}(b^5));$$

This rule has frequency 0.63 on TR and has 0.66 on CT.

Example 3.

$$(\text{wd}(a) = \text{wd}(b) = \langle 2, 3, 4, 5, 1 \rangle) \& (\Delta_{25}(a) \leq \Delta_{25}(b)) \& (\Delta_{45}(a) > \Delta_{45}(b)) \& (\Delta_{124}(a) > \Delta_{124}(b)) \rightarrow (\text{target}(a^5) > \text{target}(b^5));$$

A total of **134 regularities** (“laws”) have been found connecting SP500C and the target.

The process of generating new rules is completed when there is no rule with **higher conditional probability** and statistically significant. Note this stop criterion does not require itself to restrict the set of tested rules a priori. The restriction can be based on the volume of available data, acceptable levels of conditional probabilities and significance. For practical computations often computations stop earlier, reaching some runtime limit and/or acceptable level of conditional probabilities. The average of conditional probabilities of these regularities in training data TR is 0.5813 and the average of conditional probabilities on test data CT is 0.5759. All conditional probabilities are evaluated as relative frequencies on TR and CT, respectively, as is common in Machine Learning [Mitchell, 1997].

At first glance, 58% is discouraging. However, this accuracy is statistically significant. It is possible to reach much higher, but statistically insignificant performance on training data, and as a result, to obtain a very low performance on test data. This is called “**overfitting**” and is a well-known problem in neural networks, getting insignificant high performance. In our case, performance (conditional probability) is sufficiently **stable** when moving from training to test data. The difference is **0.0054=0.5813-0.5759**, i.e., **0.54%**. Nevertheless, this difference has a variation. Typical difference is no greater than **±3%** (53 regularities, i.e., 40%). There are also regularities with significantly higher differences. This indicates some regularities became stronger and some weaker in the financial time series for the last two years. Sometimes frequencies dropped by 50%. This can mean changing market conditions, business strategy of the target company, stockholders’ behavior or even that regularities have become known and people used them. Thus, there are three types of regularities:

- Regularities/rules with similar performance on training and test data. Frequency difference range is **±3%** (53 regularities, 40%) with only 0.14% of the average decrease of frequencies;

- Regularities/rules with increasing performance on test data. Frequency increased on 38 regularities (28%) with 5.8% of the average increase of frequency;
- Regularities/rules with decreasing performance on test data. Frequency decreases on 43 regularities (32%) with 6.6% of average decrease of frequency.

This data show that the **majority (40%+22%=68%)** of 134 regularities performed on testing data similar or better than this majority performed on training data. Therefore, forecast can be based only on the rules with the best performance on TR. Other rules can be ignored.

Noise issue. It is possible that the discovered rules will be corrupted by noise in training data and, therefore, will show poor performance on data outside of the training sample. This is a common problem of all forecast methods. Probably MMDR suffers less from noise than other methods. If MMDR captured a “critical mass” of noise, this noise would be a part of a **statistically significant** rule (MMDR selects only statistically significant rules). So it is questionable whether it should be called noise. We interpret this situation as **discovering different laws** on different data as is common in scientific discovery. For example, some laws of physics, identified using data from Earth, do not work on Moon or Mars with other gravitational levels.

Often the reason that rules may not work outside the sample is that the method is very sensitive to initial assumptions. In neural networks initial assumptions include such parameters as weight functions, number of layers, and so on. MMDR is relatively robust in this sense, because MMDR pays special attention to minimize the set of assumptions.

5.6. Method of forecasting

We can use regularities from **Law** set for forecasting only if we know right-side (**target(a⁵)**) or left-side (**target(b⁵)**) value of inequality

$$(\text{target}(\mathbf{a}^5) \leq \text{target}(\mathbf{b}^5))^{\mathbf{e0}},$$

which is a part of a found regularity. For instance, IF **target(b⁵)=45** and **e0=0** THEN we can forecast that **target(a⁵) > 45**. If we take both objects **a** and **b** from CT, then a forecast is impossible, because both target values are unknown. Taking, for example, object **a** from TR and object **b** from CT then we will have a lower bound for unknown target (**b⁵**) if **e0=1**, and it will be an upper bound if **e0=0**, because the value of **target(a⁵)** is known. Taking object **a** from CT and object **b** from TR, we will have an upper bound if **e0**

= 1 and it will be a lower bound if $e0 = 0$ for unknown value of $target(a^5)$. In the IF-part of the rule in example 1 in section 5

$$(\Delta(a) \leq \Delta(b))^{e1} \& \dots \& (\Delta(a) \leq \Delta(b))^{ek}$$

values of all inequalities for objects **a** and **b** are defined in $TR \cup CT$, the union of TR and CT and this part of the rule is an expression, which relates training and test/control objects. This expression shows similarity of objects **a** and **b**.

A target value for object **a** from CT is predicted by applying all regularities (rules) from the **Law** set to two sets of pairs of objects

$$\{<a,b> | b \in TR\} \text{ and } \{<b,a> | b \in TR\}.$$

For each rule, the first of these sets gives **upper bounds**

$$Up1(a^5) = \{target(b^5)\},$$

If $e0 = 1$ and a set of lower bounds $Low1(a^5) = \{target(b^5)\}$ if $e0 = 0$ for unknown value of $target(a^5)$. Similarly, the second of these sets $\{<b,a> | b \in TR\}$ produces **lower bounds**

$$Low2(a^5) = \{target(b^5)\}$$

if $e0 = 1$ and a set of unner bounds $Up2(a^5) = \{target(b^5)\}$, if $e0 = 0$ for an unknown value of $target(a^5)$. The whole sets of upper and lower bounds

$$Up1(a^5), Up2(a^5), Low1(a^5), Low2(a^5)$$

for $target(a^5)$ are obtained by joining these bounds for all individual regularities.

The considered regularities provide a forecast for the last day of a five-day cycle (not necessarily Friday) using data from preceding days, which could be holidays. In this case, the forecast can not be computed. Therefore, the forecast was made for 442 days from 506 on CT. This is not a genuine restriction of the method. Regularities could be discovered with missing days, but it would take more runtime. Analysis of the found regularities has shown that regularities without identification of a particular **day of the week** have significantly less prediction power.

Next, the order statistics with a confidence level is used to set **forecasting intervals** and their upper and lower bounds. The problem is that sets of

bounds $Up1(a^5)$, $Up2(a^5)$, $Low1(a^5)$ and $Low2(a^5)$ can overlap and can not be used as forecasting intervals directly.

We compute **p-quintile** ($p = 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90$) for the **upper bound** of $target(a^5)$ and **(1-p)-quintile** for the **lower bound** of $target(a^5)$. For each value of p-quintile ($p = 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90$) there are the upper bound $Up_p(a^5)$ and the lower bound $Low_p(a^5)$ for $target(a^5)$, taken respectively from

$$Up1(a^5) \cup Up2(a^5), \quad Low1(a^5) \cup Low2(a^5).$$

By default $Low_p(a^5) = -\infty$ for large p values (e.g. 0.80, 0.90, 0.95) if (1-p)-quintile is less than the least value of the lower bound for $target(a^5)$. Similarly, $Up_p(a^5) = +\infty$ for large p values (e.g. 0.80, 0.90, 0.95), if p-quintile is greater than the largest value of the respective upper bound. There is no forecast if the lower bound $Low_p(a^5)$ is greater than the upper bound $Up_p(a^5)$. It took place sometimes for small p (e.g., 0.55, 0.60, 0.65). Also the forecast is not computed if obtained p-interval is $[-\infty, +\infty]$. Note that the p-intervals

$$[Low_p(a^5), Up_p(a^5)]$$

for an unknown value of $target(a^5)$ are nested for growing p values, i.e.,

$$Low_{p1}(a^5) \leq Low_{p2}(a^5), \quad Up_{p1}(a^5) \geq Up_{p2}(a^5), \quad \text{if } p1 > p2.$$

5.7. Experiment 1

5.7.1. Forecasting Performance for hypotheses H1-H4

We have evaluated the performance of the forecast for each p-quintile and for all objects from CT using six parameters:

- percentage of Rejections,
- percentage of Errors,
- percentage of Right forecasts,
- mean length of the p-intervals for all (right and wrong) forecasts (ML)
- mean length of the p-intervals for all right forecasts (MLR) and
- bound forecast mean square error (BF MSE), i.e. mean square difference between the forecast and the nearest p-interval bound for forecasts which are out of p-interval.

For cases when one of the bounds is not defined (a “good” regularity was not found for that bound) we took a doubled distance from $\text{target}(\mathbf{a}^5)$ obtained by forecast and a known bound, i.e., $2 \cdot (\text{target}(\mathbf{a}^5) - \text{Low}_p(\mathbf{a}^5))$, if the lower bound is found. If the upper bound is known then $2 \cdot (\text{Up}_p(\mathbf{a}^5) - \text{target}(\mathbf{a}^5))$ is used. Table 5.2 and Figure 5.2 show performance metrics for test set CT of listed parameters. Figure 5.2 graphically represents first four columns of Table 5.2. It reflects that with growth of p percent of correct forecast is growing too. Figure 5.3 presents the generalized information from the last three columns of Table 5.2. It shows forecast intervals and their standard deviation for different p and found regularities.

Table 5.2. Performance metrics for a set of regularities

p-value	Rejections	Errors	Right Forecast	ML	MLR	BF MSE
0.55	102 (23%)	268 (61%)	72 (16%)	0.54	1.21	2.01
0.60	17 (4%)	315 (71%)	110 (25%)	0.82	1.33	1.59
0.65	4 (0.9%)	279 (61%)	168 (38%)	1.24	1.57	1.75
0.70	4 (0.9%)	215 (49%)	223 (50%)	1.76	2.01	1.99
0.75	3 (0.7%)	176 (40%)	263 (59%)	2.33	2.58	1.72
0.80	3 (0.7%)	125 (28%)	314 (71%)	3.03	3.24	1.38
0.85	3 (0.7%)	71 (16%)	368 (83%)	3.94	4.09	1.22
0.90	10 (2.2%)	35 (7.9%)	397 (90%)	5.19	5.25	1.10

ML is the mean length of the p-intervals for all (right and wrong) forecasts.
MLR is the mean length of the p-intervals for all right forecasts.
BF is MSE Bound Forecast Mean Square Error.

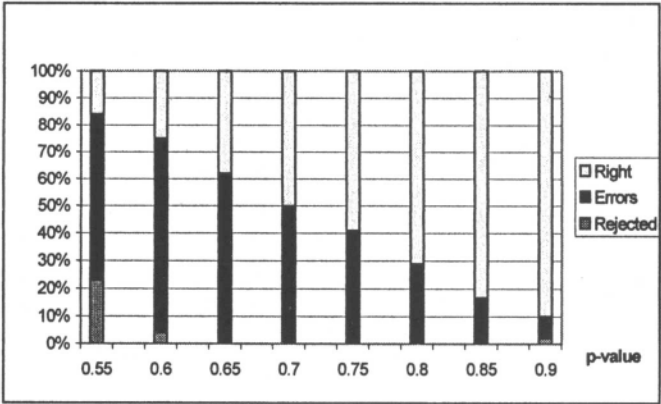


Figure 5.2. Performance of found regularities on test data.

Table 5.3 contains the forecast for the first 15 test objects. Predicted intervals are presented as two sequential numbers, e.g., 0.38 0.73. The following notation is used: “-” means that a predicted interval does not cover the actual target value, “+” means that a predicted interval covers the actual target value.

Table 5.3. Forecast performance for the first 15 objects

#	p= 0.55		p= 0.60		p= 0.65		p= 0.70		p= 0.75		p= 0.80		p= 0.85		p= 0.90		fact
1	0.50 0.49	R	0.38 0.73	-	0.09 0.93	-	-0.05 1.24	-	-0.38 1.57	-	-0.73 2.08	+	-1.36 2.53	+	-1.80 2.87	+	1.86
2	0.40 0.52	-	0.34 0.69	-	0.15 0.95	-	-0.17 1.11	-	-0.41 1.30	-	-0.77 1.47	-	-1.12 1.85	+	-1.63 2.52	+	1.81
3	0.06 0.67	-	-0.02 0.84	-	-0.25 0.99	-	-0.25 1.20	-	-0.42 1.44	-	-0.93 1.54	-	-1.24 1.73	-	-3.21 2.75	+	1.74
4	0.32 0.22	R	0.04 0.38	-	-0.07 0.97	-	-0.26 1.27	+	-0.43 1.77	+	-0.65 2.22	+	-1.14 3.06	+	-1.92 4.64	+	1.15
5	0.39- 0.22	R	0.25- 0.00	R	0.04 0.32	-	-0.26 0.62	+	-0.62 0.74	+	-2.11 1.07	+	-2.11 1.50	+	-3.16 2.23	+	-0.26
6	0.22 0.42	-	0.08 0.73	-	-0.05 1.07	-	-0.32 1.30	-	-0.72 1.85	-	-1.07 2.31	+	-1.69 2.84	+	-2.16 3.17	+	-0.76
7	0.38 0.52	-	0.31 0.79	-	0.07 1.05	-	-0.39 1.13	-	-0.63 1.42	-	-0.81 1.64	-	-1.44 1.80	+	-1.69 2.57	+	-0.89
8	0.17 0.26	-	0.03 0.40	-	-0.34 1.20	-	-0.43 1.38	-	-0.88 2.63	+	-1.04 2.77	+	-1.36 2.77	+	-1.97 2.77	+	-0.49
9	0.06 0.51	+	-0.26 0.87	+	-0.26 0.97	+	-0.43 1.27	+	-0.65 1.77	+	-1.13 2.38	+	-2.68 2.59	+	-3.58 3.59	+	0.29
10	0.04 0.75	-	-0.21 0.77	-	-0.36 2.43	+	-0.56 2.43	+	-1.35 2.43	+	-1.72 2.43	+	-2.29 3.55	+	-3.17 ---	+	1.21
11	0.20 0.57	-	0.08 0.82	+	-0.06 1.18	+	-0.35 1.37	+	-0.73 1.76	+	-1.19 2.23	+	-1.69 2.54	+	-2.15 2.94	+	0.58
12	0.54 0.52	R	0.38 0.79		0.19 1.01	+	0.15 1.13	+	-0.13 1.41	+	-0.39 1.72	+	-0.65 2.07	+	-1.48 2.66	+	0.98
13	0.06 0.62	-	0.06 0.84	+	0.06 1.08	+	0.06 1.42	+	0.06 1.51	+	-0.25 1.73	+	-1.24 2.25	+	-1.24 2.77	+	0.63
14	0.04 1.18	+	-0.09 1.18	+	-0.43 1.62	+	-0.56 1.77	+	-0.77 1.77	+	-1.39 2.15	+	-1.85 2.41	+	-2.32 3.76	+	0.95
15	0.56 0.58	-	-2.11 0.73	-	-2.11 0.92	-	-2.11 1.07	-	-2.11 1.30	-	--- 1.85	+	--- 2.17	+	--- 2.38	+	1.76

In addition, “R” means rejection, i.e., the system refused to predict using available data. If predicted lower and upper bounds cannot form the interval (e.g., we have a pair 0.50, 0.49) then we reject the forecast for this case, which is marked as R. Let us comment Table 5.3 for line 1.

There is no forecast for object #1 (five days) for $p=0.55$ because of inconsistent boundaries [0.50, 0.49]. Here lower bound is larger than upper bound. In addition, the forecast is wrong for $p=0.6$, $p=0.65$, $p=0.70$, $p=0.75$, because the actual value 1.86 is not covered by the intervals. The forecasts are correct for $p=0.85$ and $p=0.9$, i.e., inside of the intervals: [-1.36, 2.53] and [-1.80, 2.87]. This is a natural result. Moving to higher p means using a wider interval. There is no natural way to measure performance with Mean Square Error (MSE) in this situation. The interval forecast does not give us a particular predicted value.

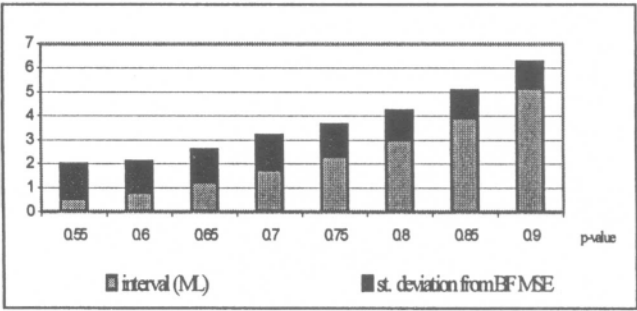


Figure 5.3. Forecast intervals for found regularities.

There is no one value of the distance from an actual value to a predicted one. We predict an interval of possible target values. Therefore, the distance to the nearest interval boundary is estimated. The distances from 1.86 to the nearest boundary (2.53) for $p=0.85$ is 0.67 and for $p=0.9$ this distance is 1.01, i.e. about 1%. These data are generalized in Table 5.3 for all test objects (CT set). For $p=0.85$ we have 0.7% of rejections from forecast, 16 % of errors and 83% right interval forecasts.

5.7.2. Forecasting performance for a specific regularity

The regularity in Example 1 (section 5, this chapter) was identified with 440 objects from training set TR. There are also 89 five-day sequences available in test set CT to test this regularity. We considered different p -values and found the number of objects from those 89 objects, which are related to a particular p -value. For example $p=0.55$ brings us 58 objects and 28 of them were predicted correctly (in relatively narrow forecast interval).

See Table 5.4. Increasing p allowed us to get up to 100% correct forecast, but with a wider forecast interval and less number of objects (see Figure 5.4 and Table 5.4). It means that for practical forecast some intermediate acceptable level of p must be chosen. Figure 5.4 shows approximately equal number of right forecasts, wrong forecasts and rejections for $p=0.55$ and growth of rejections and decreasing the number of wrong forecasts.

Table 5.4. Performance for regularity from Example 1

p-value	Right forecast	ML	MLR	BFMSE
0.55	28 from 58 (48.3%)	2.806	0.269	2.640
0.60	36 from 62 (58.1%)	3.111	0.925	3.347
0.65	34 from 56 (60.7%)	3.471	1.386	2.146
0.70	30 from 46 (65.2%)	4.081	2.119	1.989
0.75	26 from 37 (70.3%)	5.059	3.172	0.604
0.80	24 from 29 (82.8%)	4.962	4.013	0.114
0.85	16 from 18 (88.9%)	6.129	5.411	0.029
0.90	8 from 8 (100%)	6.221	6.221	0.000

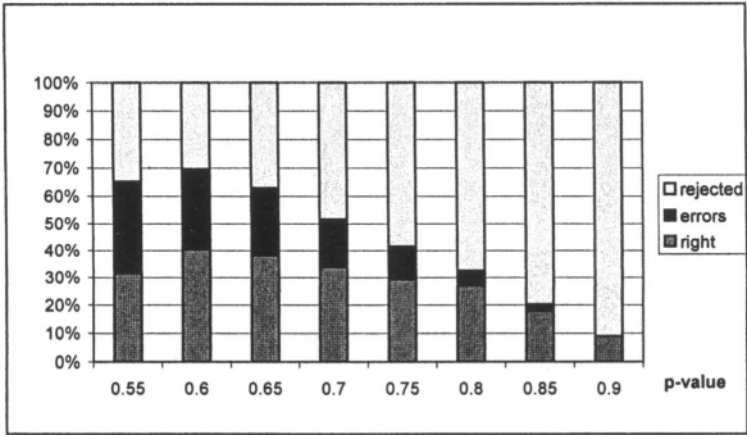


Figure 5.4. Performance of an individual regularity for 89 test objects

This choice depends on investor’s individual purposes, acceptable risk level and environment. Therefore, it should be a part of a trading strategy, which requires a special study probably similar to portfolio selection with risky securities [Hiller, Lieberman, 1995, pp.561-563]. We leave the systematic study of this issue out of this book. Without that analysis we assume that reasonable level of p -value for data presented in Table 5.4 would be [0.65, 0.75].

Let us comment on the advantage of predicting the target using a particular regularity like hypotheses H1-H4. If we exploit all 134 found regu-

larities, the target can be predicted practically for **all possible objects**, but for some of them, the forecast interval can be very large and useless. Using a particular regularity from H1-H4 often the target can be predicted only for a **few specific objects** but much more accurately. Those specific objects are selected by testing the statement Q of the regularity (IF Q then T). Only If the Q statement is true for those specific objects will T be applied. This means that relational regularities refuse to make any stock market decision for objects where there is insufficient information for an accurate forecast. This approach seems more rational than other approaches, which deliver forecasts always using **one “universal” formula (rule)** for all objects. In sections 7.1 and 7.2, SP500C was used to predict the target. In the next section, values of the target itself are used to predict the target. This is a typical approach in Markov chains models (see Section 5.4).

5.7.3. Forecasting performance for Markovian expressions

This section shows that first-order logic can help in discovering Markov chain type models **automatically**. Traditionally, states for a Markov chain are designed **manually** and then a forecasting Markov chain model is constructed (see Section 5.4). This process is very informal and its success heavily depends on the correct choice of states.

Consider the rules:

- a) IF the target decreases from the previous Monday to the current Monday THEN the target will increase for the next Tuesday with probability P_1 .
- b) IF the target increases from the previous Monday to the current Monday THEN the target will decrease for the next Tuesday with probability P_2 .

These rules match a Markov chain similar to that shown in Section 5.4. There are four states in these rules:

- 1. previous Monday,
- 2. current Monday,
- 3. previous Tuesday,
- 4. current Tuesday.

If at least one of the probabilities P_1 or P_2 is large enough the model can be used for stock forecasting. If both probabilities P_1 and P_2 are close to 0.5 (a 50/50 chance) the model does not have predictive power at all. In this case, another combination of weekdays can be tried. There are many of these combinations and their number grows very fast especially if each state will consist of several weekdays. For example, the Markov chain depicted in Table 3.17 consists of states with two weekdays each.

A mechanism to find the best set of states automatically is discussed below. This relational data mining mechanism is based on **four principles**.

The first one is Occam’s razor principle: **prefer the simplest rules** that fits the data [Mitchell, 1977]. What does it mean fit the data? We use a specific version of Occam’s razor: prefer the simplest rules with maximum expected forecasting stability when moving from sample to a real forecast. This expected stability is evaluated using a statistical test.

The second one is: **prefer simplest first-order rules and avoid complex propositional rules** (see Chapters 3 and 4).

The third principle is: **prefer rules based on interpretable relations** (predicates). Relations are interpretable if they are constructed in accordance with fundamental representative measurement theory [Krantz et al, 1971, 1989, 1990]. As we discussed in Chapter 4, any fundamental measurement can be presented using first-order logic assertions. Krantz et al [1971, 1989, 1990] developed these presentations for the most popular scales as order, interval, relational and absolute scales.

The fourth principle is: **prefer rules with directly evaluated performance of the forecast**. Conditional probabilities found in Markov chain type of models have direct relationship with performance of the forecast, i.e., how these rules are confirmed on control/test data (CT). For testing rules, we only need to find these probabilities on CT and compare them with probabilities on training data (TR). If probabilities on CT are similar or higher than on TR then rules are **confirmed**.

Using these principles in the frame of the MMDR relational data mining method, one of the found pairs of these regularities has relatively good predictive power (**probabilities 0.66 and 0.75 on test set CT**). Such regularities can be tested for practical forecast of the target.

The matrix of conditional probabilities is similar to a *confusion matrix* used in [Swanson, White, 1995] as a measure of forecast performance that is calculated how well a given forecasting procedure identifies the direction of change in the spot rate. We get probabilities dividing values on total number of cases for each row of this matrix. For example, a hypothetical confusion matrix

		Actual	
		up	down
Predicted	up	36	15
	down	12	26

presented in [Swanson, White, 1995] can be naturally transformed to a transition probability matrix:

		Actual	
		up	down
Predicted	up	36/(36+15)	15/(36+15)
	down	12/(12+26)	26/(12+26)

with values 0.7 and 0.3 on the first row and 0.31 and 0.69 on the second row. This way, the diagonal cell corresponds to correct directional predictions and off-diagonal cell corresponds to incorrect predictions. Swanson and White measures overall performance with model’s confusion rate, the sum of the off-diagonal elements, divided by the sum of all elements and statistical criterion offered by Henrikson and Merton [1981]. In transition probability matrixes (see Section 4.4 and Table 5.5), we use prediction rules:

IF delta goes up THEN target goes down,
IF delta goes down THEN target goes up.

Therefore, anti-diagonal elements in conditional probability matrices measure overall performance of these prediction rules. Anti-diagonal elements should be close to 1 for a good performance. This is only one technical difference from confusion matrix [Swanson, White, 1995] where off-diagonal elements are used. We obtained A normalized confusion matrix was obtained for Table 5.5 and is shown below:

		Actual	
		up	down
Predicted	up	0.7	0.3
	down	0.16	0.84

The natural buy/sell prediction strategy is based on this directional forecast [Cheng and Wagner, 1996]

$$\text{Prediction} = \begin{cases} \text{Buy, if UP} \\ \text{Sell, if DOWN} \end{cases}$$

The horizon of that forecast was studied (see figure 5.5). Regularities (a) and (b) are restricted. They are not applicable for all days, e.g., (a) and (b) are not applicable for Wednesday and Thursday forecast. The most promising is the forecast for one week among found regularities. There are also some lower chances for success for 5 and 8 weeks, but the majority of probabilities beyond one-week horizon is too close to 50:50. In Figure 5.6 the upper

line shows transition probabilities for **up→up** and lower line shows transition **up→down**. Direct testing of regularities on the test data are given in Table 5.5 (one-week horizon). This testing confirms the discovered regularities. In fact, they are even more confirmed in 1995-1996 data than in the training data (1985-1994).

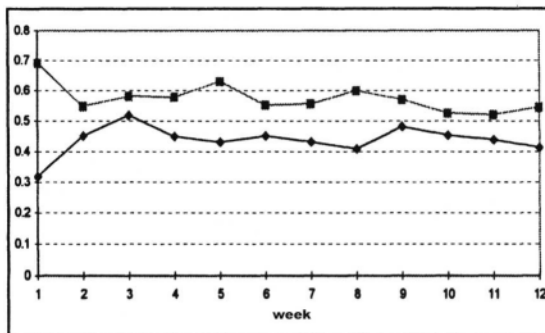


Figure 5.5. Transition probabilities for 12 weeks (training data)

In Table 5.5, we use notation from section 5.4. Transition probability for **training data** is 0.69 for transition from up to down and 0.65 is for transition from down to up. Table 5.5 shows probabilities **0.7** and **0.84**, respectively on the **testing data** (1995-1996).

Table 5.5. Transition probabilities for test data

Delta	Target	
	Up	Down
Up	0.3	0.7
Down	0.84	0.16

5.8. Experiment 2

This experiment uses daily data of SP500 for ten years as a training set (1984-1994) and daily data for four years (1995-1998) as test data. Test data were divided in two separate test sets (1995-1996) and (1997-1998) and thousands of hypotheses about the structure of the time series were tested (see examples in figure 5.6), e.g., structure 1 in Figure 5.6 means that

IF SP500C went up from Friday three weeks ago to the Wednesday two weeks ago
 AND went down from that day to Monday of the current week
 THEN the SP500C will go up to the next Monday.

Structures 2,3 and 4 have similar description. Structure 1 was discovered in training data (1985-1994) and was confirmed on **testing data** (1995-1996) in **78%** of cases. Similar estimates are presented in figure 5.6 for the rest of these rules. Term anchor is used in Figure 5.6 to show points structured relations between which are discovered.

MMDR method outperformed the risk free investment for both test periods 1995-1996 and 1997-1998. The simulated annual returns are 143.83% (1997-1998) and 126,69% (1995-1996) of initial investment in contrast with 103.05% in the risk free investment (see Chapter 6).

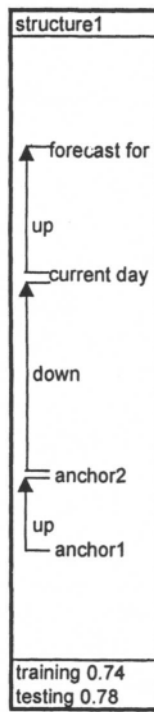
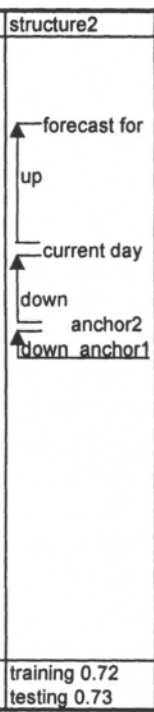
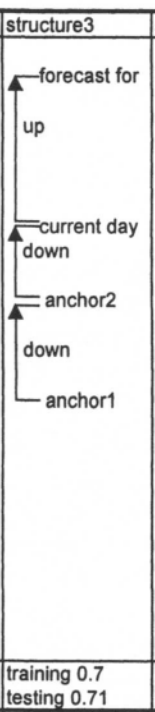
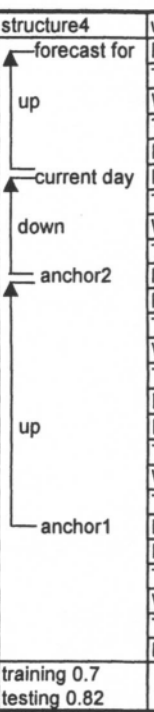
structure1	structure2	structure3	structure4	weekday	week
				Friday	forecast week
				Thursday	forecast week
				Wednesday	forecast week
				Tuesday	forecast week
				Monday	forecast week
				Friday	current week
				Thursday	current week
				Wednesday	current week
				Tuesday	current week
				Monday	current week
				Friday	one week ago
				Thursday	one week ago
				Wednesday	one week ago
				Tuesday	one week ago
				Monday	one week ago
				Friday	two weeks ago
				Thursday	two weeks ago
				Wednesday	two weeks ago
				Tuesday	two weeks ago
				Monday	two weeks ago
				Friday	three weeks ago
				Thursday	three weeks ago
				Wednesday	three weeks ago
				Tuesday	three weeks ago
				Monday	three weeks ago
training 0.74 testing 0.78	training 0.72 testing 0.73	training 0.7 testing 0.71	training 0.7 testing 0.82		

Figure 5.6. Examples of discovered structures (forecasting rules)

5.9. Interval stock forecast for portfolio selection

The interval stock forecast can be viewed as an integral part of portfolio selection with risky securities. One of the known non-linear optimization models for portfolio selection is based on the study done by Markowitz and Sharpe. This study is related to the work for which they won the 1991 Nobel

Prize in Economics Let us present the basic elements of that model [Hiller and Lieberman, 1995].

There are n stocks considered for inclusion in the portfolio $\mathbf{x}$ and $\mathbf{x}_j$ is the number of shares of stock j , $\mathbf{x}=(\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_n)$. For each stock j the estimated mean return on one share μ_j is computed using historical data. Similarly, the variance σ_{jj} and the covariance σ_{ij} of return on one share are estimated for all stocks i and j . The value σ_{ij} measures the risk of the stock j . The two functions R and V are introduced:

$$R(\mathbf{x}) = \sum_{j=1}^n \mu_j x_j, \quad V(\mathbf{x}) = \sum_{i=1}^n \sum_{j=1}^n \sigma_{ij} x_i x_j$$

The first one represents the total **return** and the second one the **risk** associated with the portfolio. The objective function to be maximized is

$$f(\mathbf{x})=R(\mathbf{x})-\beta V(\mathbf{x}),$$

where the parameter β is a nonnegative constant that reflects the investor's desired trade-off between expected return and risk. Choosing $\beta=0$ implies that risk should be ignored completely, whereas choosing a large value for β places a heavy weight on minimizing risk. This way investor's expected utility can be maximized if the model captured investor's utility function (relative value to the investor of different total returns) [Bazaraa et al, 1993].

There is a **bottleneck** in this model to identify β , $\{\mu_j\}$ and $\{\sigma_{ij}\}$. Therefore, it is common to use a parametric (nonlinear) programming approach to generate the optimal solution as a function of β over a wide range of values of β . The next step is to examine the values of $R(\mathbf{x})$ and $V(\mathbf{x})$ for those solutions that are optimal for some value of β and then to choose the solution that seems to give the best trade-off between these two quantities. This procedure is referred to as generating the solutions $\mathbf{x}$ on the **efficient frontier** [Hiller, Lieberman, 1995] and as a **Pareto solution**.

The **drawback** related to $\{\mu_j\}$ and $\{\sigma_{ij}\}$ is that values are only mean and variance over a period of time. Different periods may produce different mean and variance. It is not clear which of them will have similar values for portfolio selection term. Therefore, the usage of **predicted values** of mean and variance appears a natural extension of that model. Predicted intervals for a stock j can **substitute** the stock variance and the middle of that interval can substitute mean μ_j . Also, having probabilities and respective forecast for discovered rules ("laws"), it is possible to incorporate them into decision analysis models to maximize payoff over possible alternatives [Hiller, Lieberman, 1995] in **stochastic programming** setting.

5.10. Predicate invention for financial applications: calendar effects

Discovering relational regularities in finance assumes that relations (predicates) are already formulated. Below the term, **invention of predicates** is used for the **process of formulating predicates**. The predicates and rules presented below are generated using calendar effects listed in financial publications. Table 5.6 present rules for **in the month effects**. Tables 5.7-5.10 show effects related to **day-of-the-week effects**, a **month and a year** in predicate terms. Notation is presented in Table 5.11. All these tables use term week. We follow the definition used by [Sullivan et al, 1998]. The “weeks” are constructed such that the first trading day of the month always occurs in the first week. If the first trading day of the month is a Friday (and there is no Saturday trading), then the first week will only contain 1 day; the following Monday will be part of week 2, and so forth.

Table 5.6. In the month effects

Possible Rules for testing	Description
Day of the month effects IF IsFri(x) & IsMon(y) & InJan(x) & InJan(y) THEN ReturnClose(x)<ReturnClose(y)	In January Monday returns are positive, while they become negative during the remaining part of the year [Keim and Stambaugh 1984]
Week of the month IF IsInSecondWeek(x) & IsInFirstWeek(y) & InJan(x) & InJan(y) THEN ReturnClose(x)>ReturnClose(y)	There are 60 rules, which are long (short) on each of the five weeks while being neutral otherwise, and rules, which are neutral on each of the five weeks while being long (short) otherwise [Sullivan et al, 1998].
Half of the month IF [IsInFirstHalfMonth(x)OR IsImmedPriorFirstHalfMonth(x)]& IsInSecondHalfPreviousMonth(y) THEN ReturnClose(x)>ReturnClose(y)	Mean stock returns are positive only for days immediately prior to or during the first half of calendar months [Ariel, 1987] There is a difference between returns during the first and second half of the month [Lakonishok and Smidt, 1988]
Beginning of the month IF BeginMonth(x) & InJan(x) THEN ReturnSmallFirmClose(x)>ReturnLargeFirmClose(x)	Small firms have paid higher mean returns than large ones at the beginning of January in 32 out of 33 years, c.f., Kamara [1998]
Day of the week of the month IF IsFri(x) & IsFri(y) & IsInSecondWeek(x)& IsInFirstWeek(y) & InJan(x) & InJan(y) THEN ReturnClose(x)>ReturnClose(y)	For details see [Wang, Li, Erickson [1997].

Table 5.7. Day of the week effects

Possible Rules for testing	Description
IF IsFri(x) & IsMon(y) & NextTradDay(x,y) THEN ReturnCloseSP500(x)>ReturnCloseSP500(y)	Returns on the S&P 500 tend to be negative from Friday's close to Monday's close [French, 1980]
IF IsFri(x) & IsMon(y) & NextTradDay(x,y) THEN ReturnClose(x)>ReturnOpen(y)	Negative average returns from Friday's close to Monday's open [Smirlock, Starks, 1986]
IF IsMon(x) THEN ReturnOpen(x)> Return1HourAfterOpen(x); IF IsMon(x) THEN Return1HourAfterOpen(x) > Return2HourAfterOpen(x)	Negative returns occur in every hour of trading on Mondays [Smirlock and Starks, 1986]
IF InMon(x) THEN BeNeutral(x)	Optimal calendar rule is to be neutral on Mondays and be back in the market from Tuesdays to Fridays [Sullivan et al, 1998]

Table 5.8. The month effects

Possible Rule	Description
Month of the year	
IF InJan(x)&InDec(y) THEN ReturnSmall- Firm(x)>ReturnSmallFirm(y)	The high returns in January, are largely associated with small or foreign firms' equities [Sullivan et al, 1998]. Hypothetical rules which are long (short) on each of the twelve months while being neutral otherwise, and rules which are neutral on each of the twelve months while being long (short) otherwise. Keim and Stambaugh [1984], Roll [1983], and Rozeff and Kinney [1976] investigate these calendar rules. The full universe number of these rules is 8,188 [Sullivan et al, 1998].
IF InJan(x) & InDec(y) THEN ReturnForeignFirm(x)> ReturnForeignFirm(y)	
IF InJan(x)&InDec(y) THEN ReturnForeign- Firm(x)>ReturnLargeFirm(y)	
IF InJan(x)&InDec(y) THEN ReturnSmall- Firm(x)>ReturnLargeFirm(y)	
Turn of the month	
IF LastDayMonth(y)& InFirstThreeDaysSubseqMonth(x) THEN Return(x)>Return(y)	Very strong turn-of-the-month effect, especially between the last day of the month and the first three days of the subsequent month [Lakonishok and Smidt, 1988]

Table 5.10 presents explanation related to some of these effects and Tables 5.11 and 5.12 show predicates for discovering calendar effects.

Table 5.9. The year effects

Possible Rule	Description
Turn of the year IF LastTradingDayYear(x) THEN ReturnClose(x)>ReturnOpen(x)	Very high abnormal returns for the ending on the last trading day of the year [Lakonishok and Smidt, 1988]
Holiday IF PreChristmas(y) & NotLastPreCristmas(y) & LastPreChristmas(x) THEN Return(x)>Return(y)	Very high abnormal returns for the period beginning on the last pre-Christmas trading [Lakonishok and Smidt, 1988]

Table 5.10. Explanations of rules

Effect	Explanations
Day of the week	Explanations offered for the strong Monday effect in stock returns data include delays between trading and settlements in stocks [Lakonishok and Levi, 1982], Monday effect actually lies outside the specificity of Mondays and rather has to do with the very large number of rules considered besides the Monday rule [Sullivan at al, 1998]
Month of the year	The January effect has been linked to year-end tax-loss selling pressure that could suppress stock prices in December, only for them to bounce back in early January [Silluvan at al, 1998]

Table 5.11. Predicates for calendar indicators, return indicators and trade signals

Indicators	Number	Predicates
Days (Mon,Tues,Wed,Thur,Fri)	5	IsMon(x), IsTues(x), IsWed(x) IsThur(x), IsFri(x)
Weeks (first, second, third, forth, fifth)	5	InFirstWeek(x) InSecondWeek(x) InThirdWeek(x) InFourthWeek(x) InFirstWeek(x)
Semi-months	2	InFirstHalf(x), InSecondHalf(x)
Months	12	InJan(x), InFeb(x),InMarch(x)...., InDec(x)
Holidays (Christmas, New Year, President's Day, Independence Day)	2	BeforeHoliday(x) AfterHoliday(x)
Pre Christmas day	10	PreChristmas(y)
Next Trading Day		NextTradDay(x,y) x is the next trading day after y
Last Pre Christmas Day	1	LastPreChristmas(x)
Last Day of the Month	1	LastDayMonth(y)

Sullivan et al. [1998] estimated the number of rules for some of these cases. There are thousands of them to be tested using data mining methods.

Table 5.12. More predicates for calendar indicators, return indicators and trade signals

Indicators	Number	Predicates
In First Three Days of Subsequent Month	3	InFirstThreeDaysSubseqMonth(x)
ReturnOpen(x)	8	ReturnOpen(x)
Return1HourAfterOpen(x)		Return1HourAfterOpen(x)
Return2HourAfterOpen(x)		Return2HourAfterOpen(x)
Return3HourAfterOpen(x)		Return3HourAfterOpen(x)
.....		
ReturnClose(x)		ReturnClose(x)
Type of Securities	4	Stock(x), Bond(x),...
BeNeutral(x)	1	BeNeutral(x)
BeLong(x)	1	BeLong(x)
BeShort(x)	1	BeShort(x)

5.11. Conclusion

Relational data mining based on inductive logic programming, first-order logic and probabilistic estimates has several important advantages known from theoretical viewpoint. Computational experiments presented in this chapter have shown these advantages practically for real financial data.

Relational data mining methods and MMDR method, in particular, are able to discover useful regularities in financial time series for stock market prediction. In the time frames of the current study we obtained positive results using separately, SP500C and history of target itself for target forecast. The best of these regularities had shown about 75 % of correct forecasts on test data (1995-1996). The target variable was predicted using separately SP500 (close) and the target variable's own history. Comparison of performance with other methods is presented in the next chapter.

Chapter 6

Comparison of Performance of RDM and other methods in financial applications

An economist is an expert who will know tomorrow why the things he predicted yesterday didn't happen today.

Laurence J. Peter

6.1. Forecasting methods

In this chapter, we compare performance relational methods (MMDR and FOIL, Chapter 4) with neural networks (Chapter 2), ARIMA (Chapter 2), decision trees (Chapter 3) and linear adaptive forecasting methods (this section). Along with these methods, different trading strategies are exploited to simulate trading gain/loss. Active trading strategies are produced using these methods. Passive strategies do not assume regular trading. Passive strategies like buy-and-hold, and risk free investment with 3% interest are considered as benchmarks. Methods are compared on the same financial data as were used in Experiments 1 and 2 in Chapter 5.

Adaptive Linear Forecast. A simple adaptive linear forecast is defined as follows: $y_{i+1} = y_i + \epsilon$, where y_{i+1} , is a predicted stock price, $\epsilon = y_i - y_{i-1}$ ($i > 1$) and y_i , and y_{i-1} are stock prices for consecutive days used for forecasting y_{i+1} . This strategy means that the forecast $y_{i+1} = y_i + \epsilon$ for the next day ($i+1$) is computed using the current stock value y_i and the current change of the price ϵ as a difference from the previous day to the current day's price, $\epsilon = y_i - y_{i-1}$.

This simple strategy is computationally attractive. It does not require any sophisticated computing resources. In spite of simplicity, this strategy delivered about 120% of annual return, as we show below in two test periods in Experiment 2.

In the same experiment, MMDR method outperformed the risk free investment for both test periods 1995-1996 and 1997-1998. The simulated annual returns are 143.83% (1997-1998) and 126.69% (1995-1996) of initial investment in contrast with 3.05% in the risk free investment.

6.2. Approach: measures of performance

Comparable output. Outputs of different methods are not unified, but this is the first requirement for comparing the performance of different methods. For instance, regularities H1H3 (Section 5.3) deliver **interval** forecasts. Regularities H4 (Section 5.3) deliver **threshold forecasts**, e.g., stock price S will be no less than the threshold C ($S \geq C$). There are also **“point” forecasts**, delivering a particular value of the stock. It is not a trivial task to measure which one is closer to the actual value of a stock. For instance, a point forecast delivered a value 56.4 instead of 57.2 with a 0.8 difference between these numbers. An interval forecast delivered a correct but wide interval [56.9, 58.5] with a 0.3 difference from the lower limit and with a 1.3 difference from the upper limits. The average distance (0.8) from the actual value 57.2 to the limits 56.9 and 58.5 is the same as for a point forecast. A similar problem exists for comparing interval and point forecasts with a threshold forecast. For instance, a threshold forecast can deliver the statement **StockPrice(t+1) > 57.1** with differences ranging from 0.1 to a maximum possible difference, say 10.0.

Simulated trading. Fortunately, different forecasts can be compared using a simulated trading performance. A forecast giving the best performance obviously has an advantage. In this way, a stock forecast is generated. Then a trading signal is produced for this forecast and gain/loss is computed using actual stock data.

However, measuring simulated trading performance has a drawback. This test of a time series forecast requires a trading strategy. Therefore, forecast is tested together with a **trading strategy**. The forecast can be wrong or inefficient as well as the trading strategy. Therefore, this comparison can not be a **final comparison** of forecast methods, but it gives a useful output about the **practical value** of a forecast method in trade. This issue is discussed in [Bengio, 1997].

In Experiment 1 a simulated trading over 1995-1996 years for a specific stock was conducted for regularities in the form of H4 (Section 5.3).

In Experiment 2 a forecast delivers a daily closing value for SP500. Then simulated trading mechanisms deliver gain/loss over 1995-1998 years by comparing results of simulated trading with actual stock prices.

Trading strategies. The formula below shows signals of the trading strategy based on a linear forecast of y_t

$$y'_i = \begin{cases} \text{buy on date } i, \text{ if } y_{i+1} > y_i \text{ (predicts an up market)} \\ \text{sell on date } i, \text{ if } y_i > y_{i+1} \text{ (predicts a down market)} \end{cases} \quad (1)$$

Here to simplify consideration we omitted the case with equal stock prices $y_i = y_{i+1}$. Formula (1) means making a profit buying a stock today (date i) if its price will be higher tomorrow (**date $i+1$**) according to the forecast. Similarly, the stock is sold today if its predicted price for tomorrow is less than today's price. Apte and Hong [1996] used an alternative trading strategy with a 6% threshold (see Section 3.2):

- Sell all securities from sorted list whose predicted excess return is less than -6%, applying a 0.5% transaction fee to every trade (because of the price decline).
- Buy all securities from sorted list whose predicted excess return is greater than 6%, applying a 0.5% transaction fee to every trade.

The last strategy works with numeric **“point” forecasts**, but does not work for **up/down forecasts** without a special pre-processing, which changes the target variable. For instance, in pre-processing, the target variable $T(t)$ can be generated from actual stock prices $S(t)$ using the formula:

$$T(t) = \begin{cases} 1, & \text{if } ((S(t) - S(t-1))/S(t-1)) \geq 0.06 \text{ (buy)} \\ 0, & \text{if } ((S(t) - S(t-1))/S(t-1)) < 0.06 \text{ (hold)} \\ -1, & \text{if } ((S(t) - S(t-1))/S(t-1)) \leq -0.06 \text{ (sell)} \end{cases}$$

An **interval forecast** can be associated with several trading strategies such as:

$$y'_i = \begin{cases} \text{buy on date } i, \text{ if } \text{middle_of_interval} > y_i \text{ (up market)} \\ \text{sell on date } i, \text{ if } y_i > \text{middle_of_interval} \text{ (down market)}. \end{cases}$$

Similar strategies can be produced using lower and upper bounds of the interval. Strategies can also differ in the use of gain. In Section 2.2 three options were discussed:

- a) The investor sells the security and then buys it back at a lower price.
- b) The investor takes the cash proceeds from the sale and puts them to work in a savings account or any other investment.
- c) The investor wants to own the stock long term (passive buy-and-hold strategy).

Performance of these strategies depends on the transaction prices, costs and stock dividends. In this chapter, analysis is based on options (a) and (c).

Measures of performance. There are several measures of performance of the simulated trading [Caldwell, 1997]. The Sharpe Ratio includes a component of volatility or risk as the standard deviation of actual returns,

rather than the actual process. The standard deviation is computed across a 20-day sliding window (a trading month) of return. The Sharpe Ratio deducts from accumulated returns (over a defined period, e.g., 20 days) those returns that would result from an appropriate risk-free investment. The risk-free investment is represented by assigning a 3.0% annual return. Also a transaction cost of 0.1% of the price is applied [Caldwell, 1997].

The Sharpe ratio catches many important characteristics of simulated trading and forecasting methods, but it is not so intuitive for investors as an annual gain/loss (G). The **gross gain/loss** (GG) before any taxes is defined as a percentage of initial investment:

$$GG = 100 * (\text{final capital} - \text{initial invested capital}) / (\text{initial invested capital}).$$

Similarly $G = GG/N$, where G is an **annual gain/loss** and N is the number of years of investment.

6.3. Experiment 1: simulated trading performance

In this section, we discuss testing the discovered regularities on the test data (1995-1996) using simulated trading performance. For other details of Experiment 1, see Section 5.7. A simulated trading performance for the target (T) was evaluated on the test data (1995-1996). The target was scaled using the formula $T' = 10 * (T + 5)$ to get more convenient larger numbers. The scaling does not change the performance. An active trade strategy was compared with a buy-and-hold strategy for the 1995-1996 years (Table 6.1 and Figures 6.1 and 6.2). Buy-and-hold strategy means in our simulation “buying” *n* shares at the first trading day of 1995 and “selling” them at the last trading day of 1996. In this way, 48 shares were “bought” for 55.6 each (total investment 2668.7) on January 3, 1995 and “sold” for 60.36 on December 31, 1996 with gain of 228.44 (8.56% of the initial buy-and-hold investment).

Table 6.1. Simulated trading performance for 1995-1996.

Characteristics	Active trading	Buy-and-hold
Average investment for 1995-1996	994.53	2668.7
Final number of shares	48	48
Gain for 1995-1996	1059.87	228.44
Gain (% to the final capital)	52.92%	7.88%
Gain (% to the average active trading investment)	106.57%	Not applicable
Gain (% to the initial buy-and-hold investment)	Not applicable	8.56%

The active trading delivered a simulated gain of 1059.87 (for 48 shares) in contrast with 228.37 in the buy-and-hold strategy for the same 48 shares (Table 6.1). To simplify consideration all taxes are ignored. The initial investment used in the active strategy is much smaller (169.68) with average investment over two years equal to 994.53 in contrast with 2668.7 in the buy-and-hold strategy. It means that the active strategy does not require

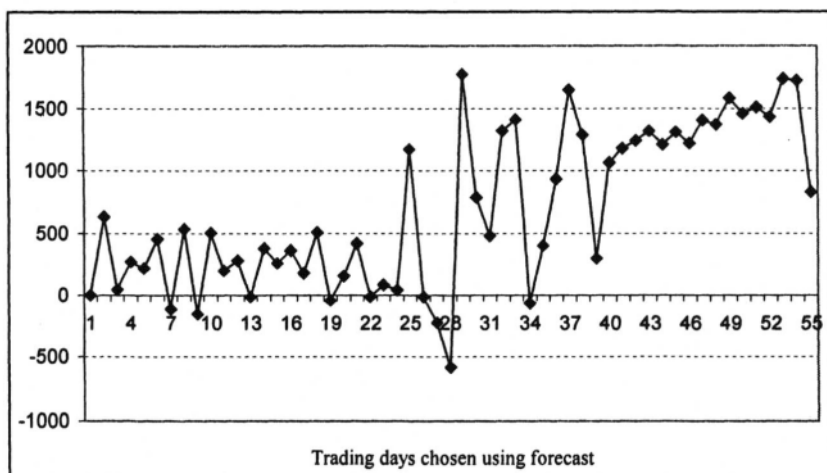


Figure 6.1. Difference in performance between active trading and buy-and-hold (1995-1996).

“tying up” 2668.7 in shares for two years. The gain is 52.92% of the final capital for the active strategy and 7.88% gain of the final capital for the buy-and-hold strategy (Table 6.1). Therefore, the active strategy outperformed the buy-and-hold strategy. All taxes were ignored as we mentioned before.

Figures 6.1 and 6.2 show gain/loss dynamics for the 1995-1996 years. Figure 6.1 shows how the active strategy outperformed the buy-and-hold strategy. Figure 6.2 shows performance of both strategies, which is the basis for Figure 6.1. Trading days are numbered on these figures from 1 to 55. These days were chosen over 1995-1996 using discovered rules and forecasts based on these rules. The rules used are applicable only for these days of 1995-1996.

To illustrate how this result was reached and how the trading dates were chosen, simulated trade for January 1995 is presented in more detail. The real trading interval in the active trading strategy is shorter. It begins on January 16, 1995. How was January 16 selected instead of the first trading day in 1995? January 16 is the first day with sufficient data to use tested regularities. These data are applicable only for trading days with particular properties. Therefore, the scope of these regularities can be relatively nar-

row. Nevertheless the mentioned regularities can deliver more exact forecasts than a regularity applicable for all trading days, because deeper and more specific properties are captured. An extended set of rules can be discovered in the same way and be applicable for a wider set of trading days. Meanwhile, we show that an active strategy with discovered regularities already outperforms a buy-and-hold strategy.

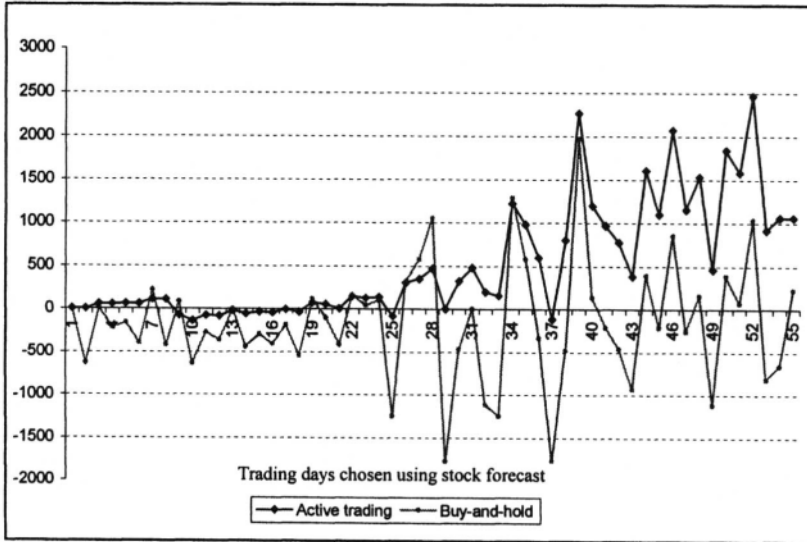


Figure 6.2. Simulated trading performance (1995-1996)

There is a mechanism within an active trade whether to chose a trade on a particular day, e.g., January 3. In the buy-and-hold strategy, this date is chosen formally as the beginning of the year. In the active strategy, rules discovered on TC (trading data) were applied, i.e., data from 1985-1994. Only data of 1995-1996 were used to make trading decisions using these rules. Only on January 16 are enough data obtained for applying discovered rules (IF-part of the rule is equal to 1). On January 16 the applicable rules forecast that the price will go up on January 23. Therefore, January 16 is the time to buy shares and January 23 is the time to sell if the forecast is correct. So, “buy” shares on January 16. On January 23 the rules forecast that the price will go down on January 30 with high probability. Thus, January 23 is the time to sell, but there is also a relatively high probability that the price will continue to grow. Therefore, “sell” only half of the shares. This can give extra gain, but force the use of a more complex active strategy. Here a simpler active strategy is followed -- “to sell all shares just bought”. In this

way, a gain is obtained on January 23. As was mentioned, the forecast for January 30 is that share prices will go down. Thus, the shares should be sold, but already all shares have been sold. Therefore, there is no trade on January 30. If we had more shares on that day an extra gain could be achieved, because the share's value is higher on January 30, 1995. The next forecast on February 6 indicates that share prices will go up on February 13, so February 6 is the time to buy. The further dynamics of gain/losses is shown in Figures 6.1 and 6.2. These figures illustrate that the active strategy, using discovered regularities and forecast, outperformed the buy-and-hold strategy for the test years (1995-1996).

6.4. Experiment 1: comparison with ARIMA

Experiments with ARIMA models were summarized in Table 2.1 (Section 2.2). Forecasting performance of these models ranges from 62.58% to 79.6% on testing data (1995-1996). Parameters for the most successful models 4 and 5 were discovered using the relational data mining approach and the MMDR algorithm described in Chapters 4 and 5.

As was discussed in Section 2.2 for target T the **sign** of $T(t+1)-T(t)$ was predicted with high accuracy, but not a **value**. A correct forecast of the sign is **sufficient** to form a successful buy/hold/sell strategy. The sign forecast is simpler than the absolute value forecast and first-order logic methods specifically fit to discover sign forecast rules.

One can not say that models presented in Section 2.2 are the best possible ARIMA models for the studied data. There are many ways for adjusting ARIMA parameters. In one of the ARIMA packages available for investors ("Forecast Expert", SBS Inc.), we experimented with an automatic adjustment mechanism. The result is 58% of the correct sign forecasts of the Dow-Jones Industrial Average (forecast 25 days ahead during 1994). This result indicates that there is room to improve an adjustment mechanism in ARIMA. The MMDR and other first order logic methods can be effective for these purposes. In particular, **parameters** of the best ARIMA models #4 and #5 were discovered by MMDR.

General problems of the ARIMA model are presented in Montgomery et al [1990]. "...there is not at present a convenient way to modify or update the estimates of the model parameters as each new observations become available, such as there is in direct smoothing. Future evolution of the time series will be identical to the past, that is, the form of the model will not change over time."

Table 6.1. ARIMA performance.

Model #	Forecasting performance (correct buy/sell signal)	Comment
1	No forecast	Buy/sell strategy is based on non-zero difference between $T(t+1)$ and $T(t)$. This random walk model has zero difference for all days. Only random advice is possible here with 50% of success.
2	62.58%	This model was selected without any connections with MMDR. Its forecast is less precise than produced by MMDR (0.7 and 0.84, respectively, correct up-down and down-up forecasts).
3	58.84%	This model was selected without any connections with MMDR. It is less precise than the rules produced by MMDR.
4	79.6%	This simple Markov process was identified by MMDR search approach. All weekdays t (Mon., Tue., Wed., Thu. and Fri.) are tested for discovering values of parameters s and t .
5	75.92%	This model exploits parameters prompted by rules discovered by MMDR. Performance 75.92% is fully consistent with MMDR performance (70% and 84%, respectively, correct up-down and down-up forecasts).

The most significant advantage of the first order methods and MMDR, in particular, is that they can **forecast directly the sign of the difference instead of the value** as ARIMA does. ARIMA can generate a sign forecast using a predicted value. The forecast of a value is more complex and available data may not fit for value forecast. The value forecast can be inaccurate and statistically insignificant, but the forecast of the sign can be accurate and statistically significant for the same data.

A similar problem exists for neural networks. Chang and Wagner [1996] noted that their individual neural networks were capable of sign forecast, but not for the forecast of the absolute magnitude. "Research results demonstrated that individual networks were able to predict bond direction much better than the magnitude of the price variance. This indicated that the networks were capable of matching the frequency and the phase of the actual bond signal (yield and price), but not the absolute magnitude." Chang and Wagner [1996] effectively used this property for the forecast. Rather than use the exact network output values, a mapping model like formulas (1) and (2) in Section 2 was developed.

The first-order logic methods allow one to go further. These methods find y'_i (the indicator of the sign of the difference) directly without generating network output y_i (see testing hypotheses H1-H4 in Section 5.3).

6.5. Experiment 2: forecast and simulated gain

Comparison of forecasting performance obtained by use of different methods in the second experiment is presented in Table 6.3. Data for this experiment are described in Section 5.8. Table 6.3 shows that MMDR outperformed other methods.

Table 6.2. Forecast performance of different methods on test data

Method	% of correct sign (up/down) forecast of SP500C		
	1995-1996	1997-1998*	Average 1995-1998
Risk Free (3%)	N/A	N/A	
Neural network 1 (with preprocessing)	68%	57	62.5%
Rules extracted from NN 1(indirect estimate)	≤ 68%	≤ 57%	≤ 62.5%
Decision tree (Sipina with C4.5 simplification)	67%	60%	64%
First-Order logic with probability (MMDR)	78%	85%	81.5%
First-order logic method (FOIL)	50.50%	45.40%	47.95%

* Data for 1998 are used from 01.01.98 to 10.31.98.

Table 6.3. Simulated gain per year for SP500

Method	Gain per year in simulated trading (% of initial investment)		
	1995-1996	1997-1998	Average 1995-1998
Adaptive Linear	21.9	18.28	20.09
MMDR	26.69	43.83	35.26
Buy-and-Hold	30.39	20.56	25.47
Risk-Free	3.05	3.05	3.05
Neural Network	18.94	16.07	17.5

The most interesting is comparison of the MMDR with the Buy-and-Hold (B&H) strategy. B&HT strategy slightly outperformed MMDR for 1995-1996 (30.39% for B&H and 26.69% for MMDR, see Table 6.4 and Figure 6.3). On the other hand MMDR significantly outperformed Buy-and-Hold for 1997-1998 (43.83% for MMDR and 20.56% for B&H.)

6.6. Experiment 2: analysis of performance

Consider the reasons for different returns. Figures 6.4 and 6.5 show the dynamics of SP500. During 1995-1996 SP500 has had almost a linear growth trend (see Figure 6.4), but for 1997-1998 this was not the case (see Figure 6.5). It is easy to show that B&H is nearly optimal for such data. Therefore, getting a return close to that given by B&H means that MMDR is also close to the best return (26.69% of MMDR gain and 30.39% for B&H, Table 6.4).

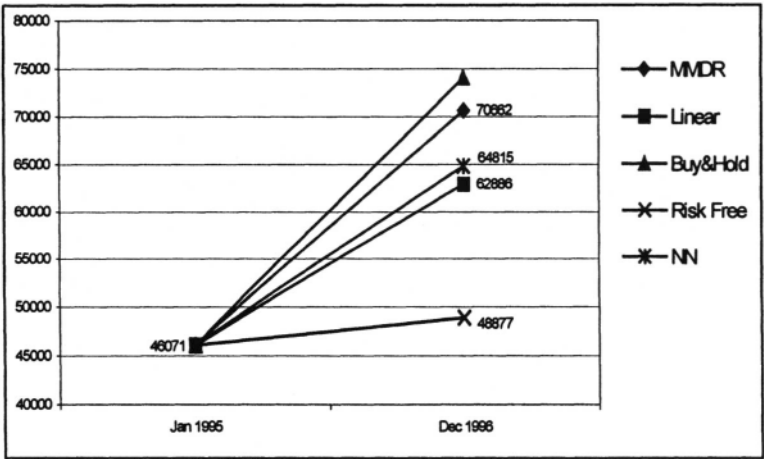


Figure 6.3. Comparison of methods in simulated trading SP500 (test data 1995-1996)

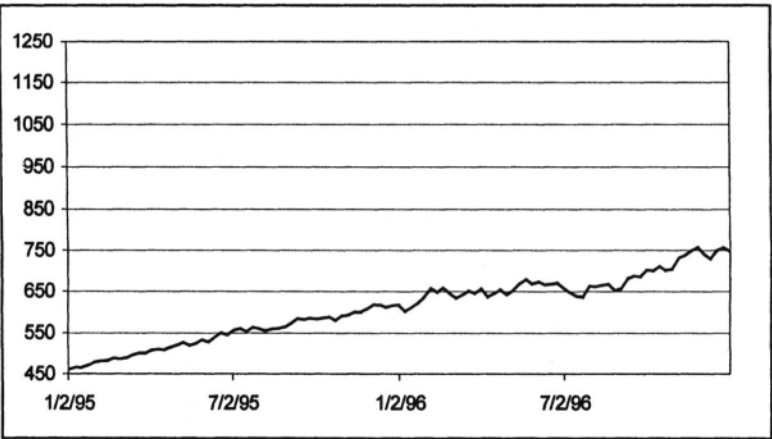


Figure 6.4. SP500C (1995-1996)

For 1997-1998, the situation is significantly different. SP500 has had much more volatility for 1997-1998 than for 1995-1996. This data constitute a much harder test for a buy-and-hold strategy. Obviously, buy-and-hold does not deliver the maximum return for such data. Buy-and-hold does not have a mechanism to adapt to a new trend, but MMDR has these capabilities. Therefore, MMDR use current information effectively applying appropriate discovered rules. Actually, these capabilities yielded a significant gain (43.83% per year).

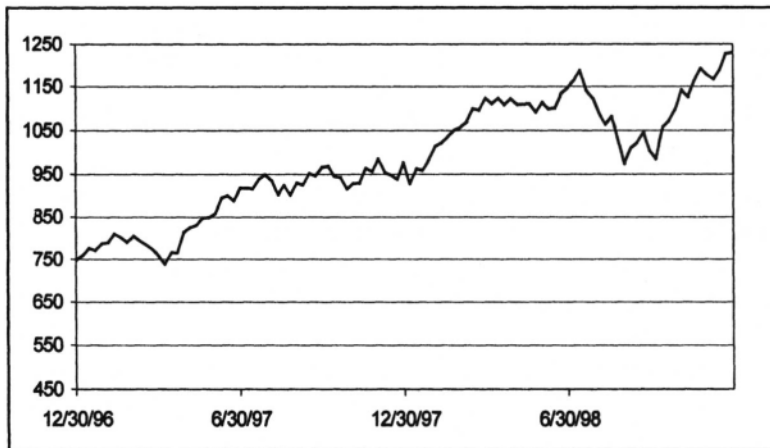


Figure 6.5. SP500 (1997-1998)

6.7. Conclusion

During many years first order logic methods were applied for other areas outside of finance such as ecology, medicine, drag design, and engineering [Russell, Norvig, 1995; Dzeroski, 1996; Mitchell, 1997, 1999]. Computational experiments presented in this chapter show that first-order logic data mining methods are able to discover regularities in financial time series. These financial tasks present a serious challenge for all learning methods [Freedman et. al, 1995].

The target variable (stock) was predicted using separately SP500C and the history of the target stock. MMDR outperformed in **accuracy of sign (up/down) prediction** such methods as neural networks, decision trees and benchmark relational method -- FOIL (Section 4.7).

Active trading strategy based on discovered rules outperformed a buy-and-hold strategy and strategies based on several ARIMA models in simulated trading for test data 1995-1996. An ARIMA model constructed using rules discovered by MMDR had shown the best performance among tested ARIMA models.

Relational data mining based on inductive logic programming and first-order logic has several important advantages known from a theoretical viewpoint. Presented computational experiments with simulated trading of SP500C have shown these advantages for real financial data. A trading strategy based on a relational data mining method (MMRD) outperformed on average the trading strategies based on other methods such as backpropagation neural network or simple linear adaptive method. It also outper-

formed benchmark trading strategies such as risk-free and buy-and-hold strategies. These experiments show that the use of relational data mining methods can benefit financial applications. Combined usage of SP500C, target history, DJIA and other indicators can produce regularities that are more powerful and forecast better.

Relational data mining methods have unrestricted capabilities for combined use of indicators, which are needed for real trading systems. Moreover, relational methods provide nearly unlimited capabilities to formulate and test hypotheses, because of the power of the first-order logic languages. The class of hypotheses H4 already has shown advantages over hypotheses tested in other methods. However, this class of hypotheses represents only the very first step in predicate and hypothesis invention in finance. An intensive growth of a new area of research and applications of relational methods is expected in coming years [Mitchell, 1999].

Chapter 7

Fuzzy logic approach and its financial applications

As far as the laws of mathematics refer to reality, they are not certain, and as far as they are certain, they do not refer to reality.

Albert Einstein

7.1. Knowledge discovery and fuzzy logic

In this chapter, fuzzy logic is presented as a part of decision-making processes in finance. Three basic types of decision-making methods are used in applications:

1. Model-based;
2. Data-based;
3. Expert-based (“expert mining”).

The model-based methods are usually associated with a **known model** and solid theoretical background, which is typical for physics. This way of modeling is the most attractive, but it does not contain much reliable knowledge outside of the hard sciences.

Data-based methods cover tasks **without a model**, but with **sufficient training data** to discover hidden regularities. These methods include **classical interpolation methods** (polynomial interpolation, spline functions, piecewise linear interpolation, etc.) and **data mining methods** (neural networks, decision trees, nearest neighbors, and so on).

“Expert mining” methods cover decision-making tasks **without a model** and **without sufficient training data**, but **with known or extractable expert linguistic rules**. Fuzzy logic methods fit these tasks very well.

Unfortunately, there is no way to know in advance if the training data are sufficient to choose a data-based or expert-based method. Usually, it becomes clear only after testing with large independent test data or several attempts to use the system. In [Mouzouris, Mendel, 1996] it was shown that fuzzy linguistic information becomes less important after enlarging the training data set.

Furthermore, there are many **intermediate tasks** with some training data and some expert rules. Such training data can be insufficient or corrupted and expert rules can also be insufficient or corrupted. These mixed cases require **mixed (hybrid) approaches**. In particular, the neuro-fuzzy approach combines **data-based** and **expert-based approaches** (see Chapter 2 Section 9). Specifically in finance, fuzzy logic is combined with neural networks. The fuzzy logic (**FL**) mechanism serves for “expert mining” and the neural network (**NN**) mechanism serves for data mining in fuzzy-neural hybrid systems. Typically these studies resulted in faster and more accurate learning of a neural network [JCIF, 1994]. Two major ways of combining fuzzy logic and neural networks parts in hybrid systems are presented in Figure 7.1. The first begins from the top left block in Figure 7.1. It **adjusts inputs and parameters of neural networks with expert information**. It is well known that neural networks are very sensitive to initial weights and normalization of input data. Fuzzy logic helps to rationalize this step of neural networks. Then neural networks discover regularity on the data and rules can be extracted from neural networks. For example, instead of entering into a neural network the federal reserve discount rate (%), we can enter this rate as a set of new variables in categories such as very accommodative, accommodative, tight, and very tight. In this way, expert information will be available to the neural networks. They are much more representative than the original variable discount rate. An alternative way would be to keep only one new variable, e.g., accommodative with different degrees. This will make neural networks smaller and faster [Von Altrock, 1997].

The second way begins from the bottom left block in Figure 7.1. It produces a preliminary set of rules using expert **information to initialize the structure of neural networks**. It is well known that neural network’s output is also very sensitive to its structure. Fuzzy logic helps to rationalize this step of neural networks. The further steps are the same in the first way--neural networks discover regularity on the data and rules can be extracted from neural networks. Similarly, fuzzy logic can be combined into a hybrid system with other data mining tools, not only neural networks. In Figure 7.1, fuzzy logic fulfils an important but auxiliary function. The actual knowledge

discovery engine is a neural network. In the opposite approach, their functions are swapped -- fuzzy logic serves as an actual knowledge discovery engine and a neural network provides an auxiliary service. This approach is depicted in Figure 7.2. Steps for implementing **knowledge discovery based on fuzzy logic** are presented in Figure 7.3.

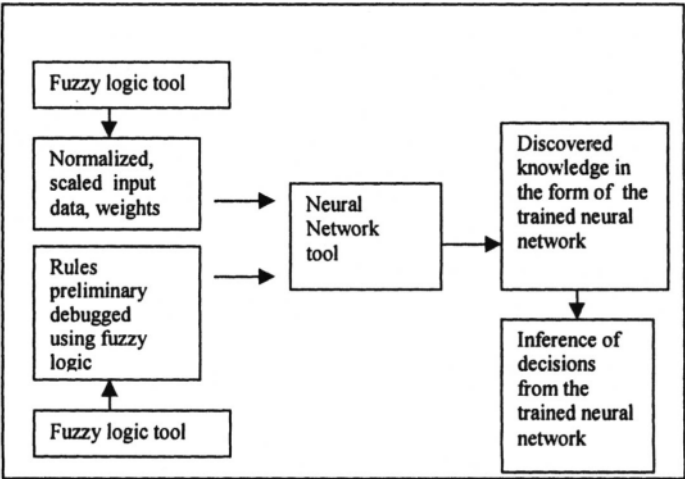


Figure 7.1. Hybrid neuro-fuzzy approach in knowledge discovery

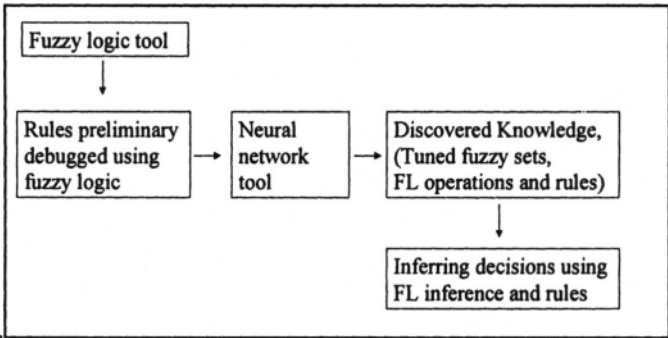


Figure 7.2. Hybrid fuzzy-neuro approach in knowledge discovery

These steps can be implemented in two ways:

- Coordinating **inputs and outputs**, i.e., the output of one step matches the input of the next step.
- Coordinating **contents** of steps and their inputs and outputs, i.e., operations in different steps are consistent.

The coordination of inputs and outputs is much simpler to implement than coordination of contents. However, without coordination of contents, the resulting rules often are much less coherent and require more work to tune them than if the steps have been coordinated from the very beginning. For instance tuned rules can become uninterpretable [Pedrycz, Valente, 1993].

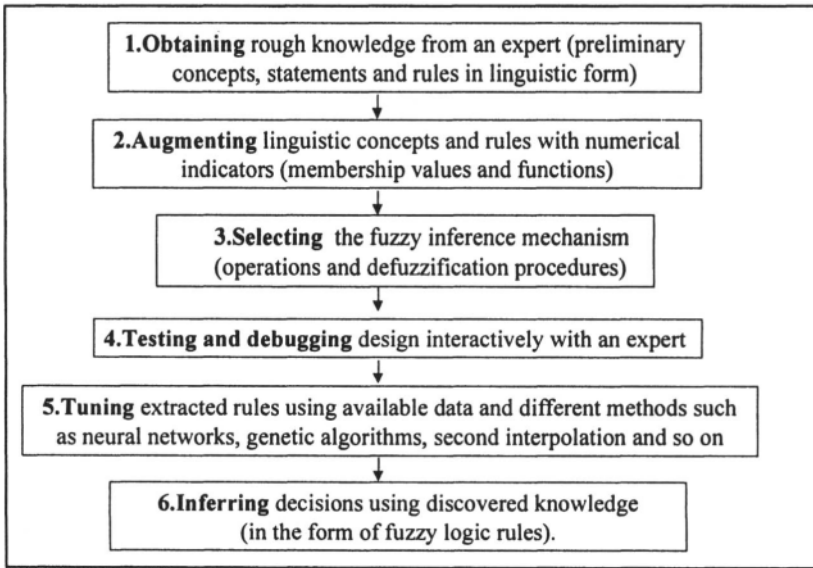


Figure 7.3. Knowledge discovery based on fuzzy logic

Losing meaningful interpretation of rules means losing one of the main advantages of the fuzzy logic approach in comparison with the “black box” data mining and forecast. Below several examples are presented to illustrate the importance of the coordinated approach. Then a **mechanism to implement these steps in a coordinated way based on concepts of context space and the second interpolation** is suggested. Description of the approach consists of six sections (Sections 7.2 through 7.7). In Section 7.2, we review mathematical principles for modeling “**human logic under uncertainty**” comparing probabilistic, stochastic and fuzzy logic approaches. Then in Sections 7.3 and 7.4 basic concepts of fuzzy logic are introduced. Sections 7.5-7.7 are devoted to constructing coordinated contextual fuzzy inference using concepts of context space and second interpolation. The context space concept addresses coordinating steps 1,2 and 3. The second interpolation addresses coordinating steps 4 and 5 listed in Figure 7.3. Section 7.8 describes financial applications of fuzzy sets.

7.2. “Human logic” and mathematical principles of uncertainty

Modern approaches to modeling “human logic” under uncertainty range from classic probability theory to such newer theories as fuzzy logic, Dempster-Shafer theory, rough sets, probabilistic networks, etc. All such approaches are oriented to somewhat different contexts. However, the appropriate context for a given application often is not clearly formulated, and thus it is very difficult to (*a priori*) select one of the approaches in favor of another in many situations. Table 7.1 illustrates differences between **stochastic and lexical/linguistic uncertainty** informally using examples from [Von Altrock, 1997] and [Cheeseman, 1985]. Expert-based decision making typically is inferred from uncertain lexical/linguistic statements like “We will probably have a successful financial year” [Von Altrock, 1997] (see Table 7.1, statement 2). Having several such uncertain statements, an inference system should be able to produce a decision. Actually, the challenge is that the system should mimic **human decision and evaluation processes in a mathematical model**. Experimental research did not confirm **initial expectations** about modeling human logic using fuzzy logic based on min and product operations, e.g., [Thole, Zimmerman and Zisno, 1979; Kovalerchuk, Taliani, 1992]. Therefore, **ad hoc tuning** became common [Kosko, 1997; Nauck et al., 1997; Von Altrock, 1997].

This made fuzzy logic theory somewhat different from probability theory, where no one tunes in ad hoc way definitions of **fundamental operations** such as union and intersection in the course of a particular study. Numerous debates can be found in the literature [e.g., Cheeseman, 1986; Dubois, Prade, 1990; Hisdal, 1998; Kovalerchuk, 1996a] about relations between fuzzy logic and probability theory **Fuzzy Logic vs. Probability Theory**. “Especially people working extensively with probability theory have denied the usefulness of fuzzy logic in applications. They claim that all kinds of uncertainty can be expressed with probability theory...**Stochastic uncertainty** deals with the uncertainty of whether a certain event will take place and probability theory lets you model this. In contrast, **lexical uncertainty** deals with the uncertainty of the definition of the event itself. *Probability theory cannot be used to model this, as the combination of subjective categories in human decision processes does not follow its axioms*” [Von Altrock, 1997, p.25]. The same author argues that stochastic uncertainty and linguistic uncertainty are of different nature.

An alternative view on the possibility to model linguistic uncertainties using the probability theory are presented in Hisdal [1998] and some other publications. We summarize arguments from various discussions in Table 7.2 and comment on them below.

Table 7.1. Comparison of stochastic and linguistic uncertainties

Feature	Stochastic uncertainty	Linguistic uncertainty
Examples of statements	Statement 1: "The probability of DJIA to be between 11000 and 11100 on June 1 st is 0.7" Statement 3: "It is possible to put n passengers into Carole's car" [Cheeseman, 1985]	Statement 2: "We will probably have a successful financial year" [Von Altrock, 1997]. Statement 4: "It is possible that shareholders will be satisfied with the mutual fund performance"
Clarity of event definition	Clearly defined events: "DJIA will be between 11000 and 11100 on June 1 st ", "put n passengers into Carole's car"	Not clearly defined events: "successful financial year", "satisfied shareholders of the mutual fund"
Exactness of resulting probability	High	Low
Importance of background of a person making an evaluation	Relatively low	High
Type of concept	Objective	Subjective
Source of uncertainty	Occurrence of event	Imprecision of human language

Let us begin analysis from the statement in Table 7.2 that stochastic and lexical uncertainties have a different nature and, therefore, require different mathematical models. There are numerous models applied to objects of a different nature, e.g., hydrodynamics and aerodynamics. In ancient time people were able count three stones, three leaves, three trees, but it took a lot of time and effort to come to an abstract concept "three". Note that today we use very different physical techniques for counting entities of different nature: stars in the galaxy, mushrooms in woods and bugs in software. However, it does not change the arithmetic with them.

Therefore, the differences in the nature of the entities and stochastic and lexical uncertainties do not mean automatically mat unified mathematics with them is impossible. It is true that the probability theory has an origin in stochastic uncertainty as a theory of chances and frequencies back in the 18th century. Nevertheless from 1933 when A. Kolmogorov published an axiomatic probability theory [Kolmogorov, 1956, English reprint], the probability theory moved onto much more abstract level. Actually currently, there are two areas:

- **abstract probability theory** as a part of mathematical measure theory and
- **mathematical statistics** as an area dealing with stochastic uncertainty in the real world.

Table 7.2. Comparison of Extreme Probabilistic and Fuzzy Logic positions

<i>Probabilistic Position</i>	<i>Fuzzy Logic Position</i>
All kinds of uncertainty can be expressed with probability theory.	Stochastic and lexical uncertainties have different nature and therefore require different mathematical models.
Probability theory can model stochastic uncertainty, that a certain event will take place.	Probability theory can model only stochastic uncertainty, that a certain event will take place.
Probability theory can model lexical uncertainty with the uncertainty of the definition of the event itself.	Probability theory can not model lexical uncertainty with the uncertainty of the definition of the event itself.
Fuzzy logic may produce useful results, but they currently are based upon weak foundations.	Fuzzy logic often produce useful results
Combination of subjective categories in human decision processes does not follow axioms of fuzzy logic theory.	Combination of subjective categories in human decision processes does not follow axioms of probability theory.

These two areas should not be mixed. **Mathematical statistics** matches **stochastic uncertainties** with abstract probability theory, but it does not prohibit **matching other linguistic and subjective uncertainties** with the abstract probability theory. This is actually done with development of **subjective probability theory**, e.g., [Wright, Ayton, 1994]. Moreover, a **probabilistic linguistic uncertainty theory** was developed [Hisdal, 1998; Kovalerchuk, Shapiro, 1988; Kovalerchuk, 1996a] inspired by a very productive concept of linguistic variables developed by Zadeh [1977]. This third approach switches the **focus** from discussing differences in nature of uncertainty to **formalizing contexts of lexical uncertainty**.

The formal construction of **probability space** was the principal achievement of probability theory and allowed for the expression of an appropriate **context** for use with probabilities especially as stochastic uncertainties expressed within relative frequencies. Clearly, context is important when using linguistic concepts also [Zadeh, 1977] and it is here where fuzzy sets and membership functions typically are used. Therefore, we argue that an *analog* (but by no means an identity) of probability space as an expression of context is critically important for firmly founding and more ably advancing the theory and use of fuzzy sets.

For at least two centuries, probability calculations were made without a strict context base. Obviously, many mistakes occurred as a result of such "context-free" calculations of complicated probabilities, and many results were debated. Without more clear and strict contextual bases, it is impossible to properly verify or falsify such results. Currently, the situation for "non-probabilistic inference" (fuzzy logic, et al.) is quite analogous to the one that probability theory was in prior to Kolmogorov's "corrective medicine" which introduced the concept of probability space. That is, fuzzy logic

may often produce useful results, but they currently are based upon weak foundations. Below we present a way to strengthen those foundations.

L. Zadeh [1988] supposes that inference under linguistic uncertainty ("lexical imprecision") is precisely the field in which fuzzy logic can and should be appropriately and effectively employed. P. Cheeseman [1985, 1986] uses conditional probabilities and probabilistic measures for specific kinds of propositions, and Z. Pawlak, et al.[1988] use lower and upper probabilistic approximations as generalizations of rough sets to work with linguistic uncertainties.

The main idea of Cheeseman's interpretation is demonstrated in the following: The fuzzy logic "possibility distribution" is just the **probability assigned to each of the propositions** – "it is possible to put n passengers into Carole's car" -- even though, for some values of n , there is considerable uncertainty (i.e., the probability is not close to either 0 or 1) [Cheeseman, 1985]. However, Cheeseman did not construct any **exact probability space** for his example. Therefore, in these terms, several questions must be answered: what is the reference set (set of elementary events) for the needed space? How can we obtain the reference set, probabilities of its elements and their combinations? What are the appropriate distribution and density functions?

In response to Cheeseman, Dubois & Prade [1990] draw attention to these weaknesses. That is, they emphasize that in many Artificial Intelligence applications, for example, it is not realistic to find answers for some such questions. Dubois & Prade do not debate Cheeseman's probabilistic approach; they just assert that in many real cases we have **no knowledge about the components of a complete probability space**. This may suggest that the fuzzy logic theory addresses this problem.

It is our contention that the situation is more complicated. For instance, we may have sufficient data to construct probability spaces, but fuzzy logic **Membership Functions (MFs)** could still be needed. In our context space approach, MFs are **not** equivalent to probability distributions. Thus, we clearly do not agree with Cheeseman that the concept of MF always would be useless, whenever there are **fully correct and required probabilities available**. On the other hand, we disagree with a "dynamic approach", which motivates the concept of and need for MFs by **dynamic changes** in, and/or absence of, a complete probability space. Thus, context spaces for linguistic concepts reduce the apparent contradictions between the already debated approaches of Cheeseman vs. Dubois & Prade.

More specifically, context space is considered as a class of connected measure spaces represented correctly and compactly by MFs. Here then, **an MF is not a probability distribution, but each separate value of $m(x)$ is a probability**. That is, an MF is viewed instead as a "cross section" of

probability distributions [Kovalerchuk, 1996a]. For instance, each of membership functions $m_{\text{low-rate}}(x)$, $m_{\text{medium-rate}}(x)$, $m_{\text{high-rate}}(x)$ are not probability density functions with respect to x , but their cross sections for $x=0.04$ (4% rate) could be a probability density function $f_{0.04}(r)$ with respect to r , where r is a linguistic term, $r \in \{\text{low-rate, medium-rate, high-rate}\}$:

$$\text{Sum}_{0.04}=f_{0.04}(\text{low-rate})+f_{0.04}(\text{medium-rate})+ f_{0.04}(\text{high-rate})=1,$$

where $f_{0.04}(\text{low-rate})=m_{\text{low-rate}}(0.04)$, $f_{0.04}(\text{medium-rate})=m_{\text{medium-rate}}(0.04)$ and $f_{0.04}(\text{high-rate})=m_{\text{high-rate}}(0.04)$. Such an interpretation is acceptable for many fuzzy-inference decisions; indeed, experts often construct such "**exact complete context spaces**" (ECCSs) with $\text{sum}_x=1$ informally for specific problems. Thus, we argue that the description of linguistic context, in a natural way, cannot be achieved with the use of **single probability spaces** alone. Each interest rate x corresponds to its own small probability space over the set of linguistic terms like $\Lambda=\{\text{low-rate, medium-rate, high-rate}\}$ [Kovalerchuk, 1996a].

7.3. Difference between fuzzy logic and probability theory

Figure 7.4 presents **interpreting mechanisms** available in classical probability and fuzzy set theories to match these theories with real world entities. The major theoretical concept of probability theory to be interpreted is the **concept of probability $p(x)$** and the major theoretical concept of fuzzy logic to be interpreted is the **concept of membership function $m(x)$** . Both concepts measure uncertainty of real-world entities. Interpreting mechanisms should also interpret operations with $p(x)$ and $m(x)$ as meaningful operations for real-world entities.

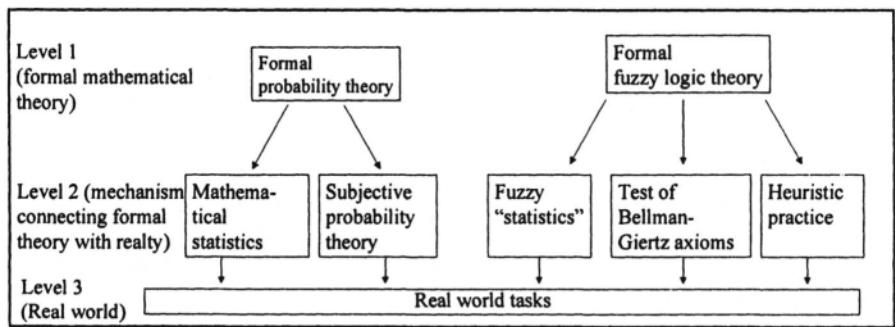


Figure 7.4. Mechanisms for acquisition probabilities and membership functions

An interpreting mechanism should set **empirical procedures** for obtaining values of $p(x)$, $m(x)$ and meaningful operations with them. Technical details of these procedures are not critical but their context dependencies and justification are of critical importance. Heuristic practice shown in Figure 7.4 usually is “context-free” with little justification, which means that ad hoc tuning is necessary.

Context dependence of empirical procedures means that the value of $m(x)$ is assigned in a co-ordination with assigning values of m for other entities $y, z, \dots$: $m(y)$, $m(z)$,

Context independence of computed operations means that:

$$m(a) \& m(b) = f(m(a), m(b)), \quad m(a) \vee m(b) = g(m(a), m(b));$$

i.e., the operations are functions of only those variables $m(a)$, $m(b)$. For instance, **the most common operations** in fuzzy logic **min** and **max** representing **AND** and **OR**, respectively, are **context independent**:

$$a \& b = \min(a, b), \quad a \vee b = \max(a, b)$$

where $x = m(a)$, $y = m(b)$. These operations are also called **truth-functional** [Russell, Norvig, 1995]. Note that in contrast in probability theory, $\&$ operation is not truth-functional, because

$$p(a \& b) = p(b)p(a/b),$$

where the conditional probability $p(a/b)$ is a function of the two variables (a, b) , and the $p(b)$ is a function of just one variable. Thus, **probability $p(a \& b)$ is context dependent**. When we speak of $p(a/b)$, b is a context for a ; if b changes, so does the conditional probability. (For more details, see Gaines [1984] and Kovalerchuk [1990]).

7.4. Basic concepts of fuzzy logic

A fuzzy set and membership functions. We begin from an example. The interest rate 0.02 is considered as a prototype for “low interest rate” and 0.06 is considered as definitely being outside of the “low interest rate” set of rates. A particular rate (e.g., 0.04) can be compared with 0.02 and 0.06 and the result can be expressed as a number reflecting expert’s opinion about **degree of membership** of 0.04 to the set of “low interest rates”. This number is called a value of the **membership function (MF)** $m(x)$ of the fuzzy set “low interest rate”. Figure 7.5 gives an example of such membership function.

Through this chapter, a financial demonstration example from [Von Altrock, 1997] is used. In particular, the membership function presented in Figure 7.6 is extracted from that demonstration. In this way, we show advantages and disadvantages of usage of fuzzy logic in finance. In particular Section 7.5 shows the inconsistency of standard “context-free” (truth-functional) operations. Then context spaces are used to fix this inconsistency.

Let X be a set of all possible interest rates from 0.0 to 1.0. The set of interest rates $\{x\}$ is called the **universe X** and x is called a “Base variable”. The **support of the fuzzy set** is defined as a set of x such that $m(x) > 0$. The degree of membership m covers the entire interval $[0,1]$, where 0 corresponds to absolutely no membership in the set, 1 corresponds to complete membership and 0.5 usually corresponds to the most uncertain degree of membership.

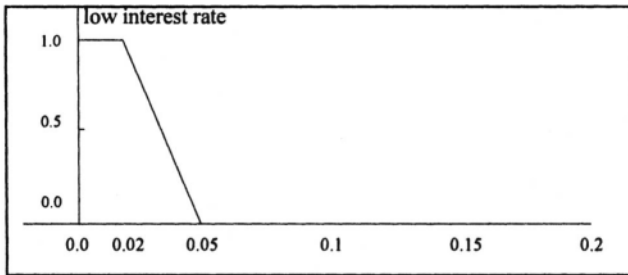


Figure 7.5. Fuzzy set “low interest rate” (“low rate”)

Some examples of membership values are presented below:

$$\begin{array}{lll}
 m_{\text{low-rate}}(1.0) = 0 & m_{\text{low-rate}}(0.05) = 0.022 & m_{\text{low-rate}}(0.025) = 0.9 \\
 m_{\text{low-rate}}(0.5) = 0 & m_{\text{low-rate}}(0.04) = 0.5 & m_{\text{low-rate}}(0.02) = 1 \\
 m_{\text{low-rate}}(0.06) = 0 & m_{\text{low-rate}}(0.03) = 0.72 & m_{\text{low-rate}}(0.01) = 1
 \end{array}$$

Fuzzy sets generalize conventional “crisp” sets with only two values, 1 and 0. Formally the pair

$$\langle X, \{m_{\text{low-rate}}(x): x \in X\} \rangle$$

is called a **fuzzy set of low rates**, i.e., to define a fuzzy set we need three components: linguistic term (“low rate”), universe ($X=[0,1]$, i.e., all possible rates), and a membership function. **Fuzzy logic** combines fuzzy sets **to infer conclusions** from fuzzy logic expressions using membership functions. For example, fuzzy logic assigns a truth value to the expression “the interest rate

0.03 is low AND the interest rate 0.24 is low". It is based on the expression "the interest rate 0.03 is low" with the truth value 0.72 and the expression "the interest rate 0.024 is low" with the truth 0.82.

Linguistic Variables. The most productive concept of fuzzy logic is the concept of **linguistic variable** [Zadeh, 1977]. **Linguistic variable** is a set of fuzzy sets defined on the same universe X for related **linguistic terms** like low, medium, high. See Figure 7.6 for membership functions for these terms [Von Altrock, 1997].

At first glance, any related fuzzy sets can and should be used to create a linguistic variable. However, assigning membership degree $m_{\text{medium}}(0.03)$ for rate 0.03 without coordinating this value with already assigned value $m_{\text{low}}(0.03)$ would be **non-contextual**. In Figure 7.6 it is not the case, here $m_{\text{medium}}(0.03)=1-m_{\text{low}}(0.03)$.

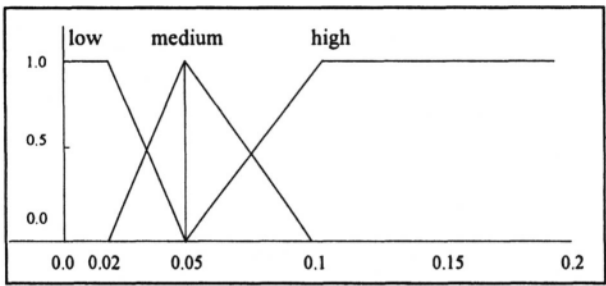


Figure 7.6. Membership functions “low”, “medium”, “high” for linguistic variable “interest rate”

Fuzzification. The process of computing values of membership functions of fuzzy sets for given values of base variables is called fuzzification. Example 1. Let "interest rate" = 0.03. The result of fuzzification for 0.03 could be (Figure 7.6):

low	truth value = 0.72	($m_{\text{low}}(0.03)=0.72$)
medium	truth value = 0.28	($m_{\text{medium}}(0.03)=0.28$)
high	truth value = 0.0	($m_{\text{high}}(0.03)=0.0$)

Example 2. Let "Trade fee" = 0.015. The result of fuzzification for 0.015 could be (Figure 7.7):

low	truth value = 0.78
significant	truth value = 0.22

The term set {low, medium, high} for the linguistic variable “interest rate” does not allow us to express linguistically an interest rate of 0.03. The closest term is “low” (0.72). A larger term set, which will include the term “al-

most low rate, just slightly greater” can be developed. In this way, 0.03 can be represented by a specific linguistic term. Otherwise fuzzification will represent the same idea numerically as three numbers (0.72, 0.28, 0), respectively for low, medium and high rates. The result of fuzzification is used as input for the fuzzy rules.

Fuzzy Rules. Background knowledge of the system is expressed in fuzzy logic in the form of “If-Then” rules. These rules connect linguistic terms of several linguistic variables to infer decision (outputs). The **If-part** of a rule is called the precondition or antecedent and the **Then-part** is called conclusion or consequent. The If-part can consist of several preconditions joined by linguistic connectors like AND and OR.

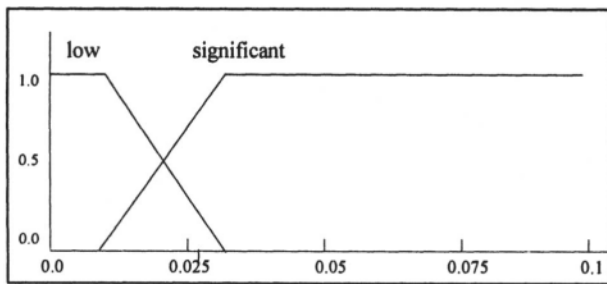


Figure 7.7. Linguistic variable “trade fee”

Fuzzy Rule Inference. Let us begin from examples.

Rule 1:

IF "interest rate" = low AND "trade fee" = low
THEN "environment (trade environment)" = positive

Rule 2:

IF "interest rate" = low AND "trade fee" = significant
THEN "environment" = positive

Rule 3:

IF "interest rate" = high AND "trade fee" = significant
THEN "environment" = negative

Rule 4:

IF "interest rate" = medium AND "trade fee" = significant
THEN "environment" = indifferent

These rules use the AND operator in their IF-parts. Consider a trade in which the interest rate is 0.03 and the trade fee is 0.015. Fuzzification of this data would establish for rule 1 that $m_{\text{low-interest-rate}}(0.03)=0.72$ for the first precondition and $m_{\text{trade-fee}}(0.015)=0.78$ for the second precondition.

Fuzzy rule inference consists of two parts: **aggregation** (computing the truth value of the IF-part of each rule) and composition (computing the truth value of the **conclusion** of the set of rules).

The typical fuzzy logic assumption in aggregation is **truth-functionality**: “the truth of complex sentences can be computed from the **truth of the components**. Probability combination does not work this way, except under strong **independence** assumptions” [Russell, Norvig, 1995] (see also Section 7.3). Several operators were suggested for combining components, for instance, Min and Product are used as AND operation.

The minimum (MIN) operator used to compute the truth value of the entire condition in rule 1 produces the following truth value:

$$\text{MIN}\{\text{Truth value ("interest rate" = low), Truth value ("trade fee" =low)}\} \\ =\text{MIN}\{0.72,0.78\} = 0.72.$$

Similarly, truth value of rule 2 can be calculated:

$$\text{MIN}\{\text{Truth value ("interest rate" = low), Truth value ("trade fee" =significant)}\} = \text{MIN}\{0.72,0.22\} = 0.22.$$

If more than one rule produce the same conclusion (e.g. "environment" = positive), the **maximum of truth values** of the conclusions is selected for all further processing. This is actually representation for logical OR operator. Example:

Rules 1 and 2 yield the same result for "environment", but with different truth values of the conditions:

$$\begin{aligned} m_{\text{positive-environment}}(0.03, 0.015) &= \min(0.72,0.78)=0.72 && \text{(rule 1)} \\ m_{\text{positive-environment}}(0.03, 0.015) &= \min(0.72,0.22)=0.22 && \text{(rule 2)} \end{aligned}$$

The composition step will produce the truth value **MAX(0.72,0.22)=0.72** for the pair ((0.03, 0.015) of interest rate and trade fee. The above used combination of max-min operators is called **MAX-MIN inference**.

Another standard for fuzzy logic is **MAX-PROD**. This method produces the following output for the previous example:

$$\begin{aligned} m_{\text{positive-environment}}(0.03, 0.015) &= 0.72*0.78=0.56 && \text{(rule 1)} \\ m_{\text{positive-environment}}(0.03, 0.015) &= 0.72*0.22=0.16 && \text{(rule 2)} \end{aligned}$$

The third operation is called **BSUM (bounded sum)**. The result is equal to 1, if the sum exceeds 1, else it is equal to the sum. For the same example this method delivers:

$$\begin{aligned} m_{\text{positive-environment}}(0.03, 0.015) &= \text{BSUM}(0.72, 0.78) = 1 && \text{(rule 1)} \\ m_{\text{positive-environment}}(0.03, 0.015) &= 0.72 + 0.22 = 0.94 && \text{(rule 2)} \end{aligned}$$

Many practical applications have shown that these inference methods can be used more or less **interchangeably**, depending on which defuzzification method is used [Von Altmann, 1997]. This empirical observation has some theoretical explanation [Kovalerchuk, Dalabaev, 1994]. It was found under some assumptions that different inference methods produce close final orderings of alternatives (x,y) with a difference about 10%. In the example above, we need to order alternatives as a pair (interest rate, trade fee) with respect to the ordering relation “better trade environment” ($<_{\text{trade-environment}}$). For instance,

$$(0.06, 0.02) <_{\text{trade-environment}} (0.03, 0.015)$$

means that (0.03, 0.015) corresponds to a better trade environment than (0.06, 0.02).

Fuzzy Operators. All listed operators are truth-functional as we already mentioned. However, in real financial and many other applications, this assumption may not be true. Therefore, several other operators were suggested. These operators like the GAMMA operator have adjustable parameters to fit a particular decision-making task. These operators are called **compensatory operators**. GAMMA and MIN-AVG belong to this class of operators. For instance, the following compensatory operator $g(x,y)$ can be considered:

$$G_{\alpha,\beta}(x,y) = \alpha * \text{MIN}(x,y) + \beta * \text{MAX}(x,y)$$

Linguistic result design. At the end of fuzzy rule inference, all output variables are associated with a fuzzy value. To exemplify this, the following truth values are assigned to the trade environment:

$$\begin{aligned} m_{\text{Positive-trade-environment}}(0.03, 0.015) &= 0.72 \\ m_{\text{Indifferent-trade-environment}}(0.03, 0.015) &= 0.2 \\ m_{\text{Negative-trade-environment}}(0.03, 0.015) &= 0.0 \end{aligned}$$

Next, an extended set of linguistic terms can be developed to capture linguistically trade environment expressed with three numbers (0.72; 0.2; 0). There can be the term “positive, but slightly indifferent trade environment”.

Rule Definition

There are two major stages in defining rules:

- formulate pure linguistic rule (prototypes) and
- formulate the prototype as a fuzzy rule.

Example. Consider a statement:

IF the trade period has low interest rate and low trade fee
THEN the trade environment is positive.

This can be formulated as a fuzzy rule:

IF "trade rate" = low AND "trade fee" = low
THEN "environment" = positive.

Rule development is an iterative process, which involves tuning rules and the development of similar rules for other terms. For example, the following rule could be defined: "IF trade period has medium interest rate and it has the trade fee higher than the low trade fee then the trade environment is indifferent". This can be formulated as a fuzzy rule too:

IF "interest rate" = medium AND "trade fee" = significant
THEN "environment" = indifferent"

The next important concept in fuzzy logic is a **matrix of linguistic rules (rule matrix)**. This matrix is presented in Table 7.3. This matrix is coded with numbers in Table 7.4. Code low interest rate as 1, medium as 0 and high as -1. For trade fee use code 1 for low and 0 for significant. For environment use 1 for positive, 0 for indifferent and -1 for negative. It seems that this coding is inconsistent.

Table 7.3. Matrix of linguistic rules

If		Then
Interest Rate	Trade Fee	Environment
Low	Low	Positive
Medium	Low	Positive
High	Low	Indifferent
Low	Significant	Positive
Medium	Significant	Indifferent
High	Significant	Negative

We do not code highest degrees for interest and trade fee with highest number (1). This is done deliberately. The suggested coding scheme allows us to keep the **meaningful property of monotonicity -- larger numbers reflects better interest rate, trade fee and trade environment.**

The information from Table 7.4 is visualized in Figure 7.8. Here 11 on the left side represent the **IF-part** of rule (the interest rate is low and the trade fee is low); 01 means that the interest rate is medium and the trade fee is low. The right lattice in Figure 7.8 shows in each node the **IF-part** of rules along with their **THEN-part** (environment value) too.

Table 7.4. Numerical rule table

If		Then
Interest Rate	Trade Fee	Environment
1	1	1
-1	0	-1
0	0	0
1	0	1
0	1	1
-1	1	0

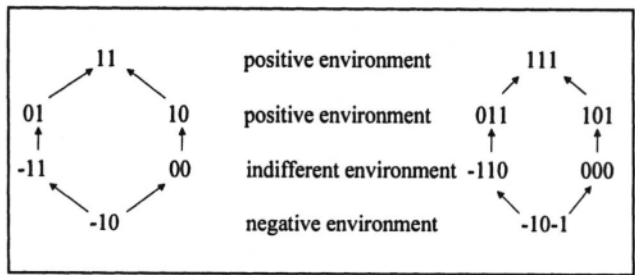


Figure 7.8. Lattice of rules

Another representation of this matrix is given in Table 7.5.

Table 7.5. Rule matrix: relational form

Interest rate	Trade fee	
	1 (low)	0 (significant)
1 (low)	1	1
0 (medium)	1	0
-1 (high)	0	-1

Defuzzification. Fuzzy rules, in contrast with rules in classical logic, produce fuzzy output. It can be a set of values of the membership functions values or a linguistic term. For example, the result could be equivalent to a complex linguistic statement "Trading environment is mostly positive for buying, but it is also slightly indifferent for buying". Obviously, program trade systems or traders cannot interpret such linguistic commands in the same way as crisp buy/hold/sell signals. In fuzzy logic, membership functions are used to retranslate the fuzzy output into a crisp value. This **re-**

translating is known as **defuzzification** summarized in Table 7.6 [Nauck et al, 1997; Passino, Yurkovich, 1998, Von Altröck, 1997].

Table 7.6. Procedures for defuzzification

Abbreviation	Name	Algorithm	Comment
CoA, CoG	Center-of-Area, Center-of-Gravity	Compute x such that areas on both sides of x are equal.	Slow computation
CoM	The Center of Maximum	A weighted mean of the term membership maxima, weighted by the inference results.	Fast, but neglects overlapping approximating CoA (CoG).
CoA BSUM		Boundary sum variant of CoA	Optimized for efficient VLSI implementation
MoM	The Mean-of-Maximum	Computes the mean of the truth values for the term with highest resulting truth value.	
MoM BSUM		BSUM Variant of MoM	Optimized for efficient VLSI implementation

7.5. Inference problems and solutions

In previous sections, we presented common fuzzy logic methods. These methods have been shown to be effective in many applications. However, there are some inference problems in applying these methods. Below we show:

- typical fuzzy inference problems using a financial example and
- a possible way to solve these problems.

Let us consider rules 1 and 2 from Table 7.3,

rule 1:

IF "interest rate" = low AND "trade fee" = low
THEN "environment" = positive

rule 2:

IF "interest rate" = low AND "trade fee" = significant
THEN "environment" = positive
The last two rules imply

rule 5:

IF ("interest rate" = low OR "interest rate" = medium) AND
"trade fee" = low
THEN "environment" = positive

The following notation is used below for a membership function $m_A(R,T)$ of fuzzy set A . For instance, if A ="indifferent trade environment" then $m_{\text{indifferent}}(R,T)=1$ means that the membership degree is equal to 1 for pair (R,T) for the fuzzy set "indifferent trade environment". For simplicity, sometimes we will omit arguments like (R,T) in the membership functions.

Consider some values of interest rate R and trade fee F such that $R=0.04$ and $F=0.01$. According to [Von Altrock, 1997] $m_{\text{low}}(R)=m_{\text{medium}}(R)=0.5$ and $m_{\text{low}}(T)=1$, i.e., F is definitely considered as low fee and R as something between low and medium interest rate (see Figures 7.6 and 7.7). According to rule 5 the **environment** for pair $\langle R, T \rangle = \langle 0.04, 0.01 \rangle$ is **definitely positive**, i.e.,

$$m_{\text{positive-environment}}(R, T) = 1.$$

Computation of the m value using traditional fuzzy MIN-MAX inference (suggested for this task in [Von Altrock, 1997]) produces

$$m_{\text{positive-environment}}(R, F) = 0.5.$$

We also obtained similar result (0.51) for inputs 0.04 and 0.01 using Fuzzytech software [Von Altrock, 1997]. As was already mentioned, in fuzzy logic, membership function value around 0.5 is interpreted as the highest level of uncertainty of the conclusion.

This fuzzy inference conclusion means that there is no assurance that given R and T fit the positive trade environment. This contradicts **rule 5**, which states that the given R and T definitely fit the positive trade environment. The last one is consistent with both logical inference and intuitive expectations. In [Von Altrock, 1997] several ways are offered to change results in the case of such an obstacle. The approach is called “**debugging**”. Fuzzytech has a convenient tool for this. However, the problem of debugging of rules is challenging even for sophisticated users. In particular, the author of the respected Fuzzytech software himself had not debugged the rules completely before publishing the example with MAX-MIN fuzzy rule inference [Von Altrock, 1997, p. 217]. This inference actually has caused the contradiction with logical and intuitive expectations. It shows again that the problem is not trivial. We used an alternative BSUM defuzzification available in Fuzzytech. BSUM has produced

$$MF_{\text{positive-environment}}(R, T) = 1.$$

This output is fully consistent with logical and intuitive expectations. See also output section in Figure 7.9.

We strongly believe that these difficulties can be effectively avoided not through error-prone debugging of very heuristic procedures, but by a thorough **analysis of context space**. This can be performed by designing the fuzzy system in compliance with recent developments in the **second inter-**

polation approach of fuzzy set theory. Below we outline some major concepts of this approach (for more detail see Kovalerchuk [1996]).

Before moving to more theoretical study, let us analyze rules 2 and 4 from the same example. This consideration shows that even if debugging on the intermediate step has shown consistency of the rules they still may be inconsistent in a further inference.

Rule 2: IF “interest rate” = low AND “trade fee” = significant
THEN "environment" =positive

Rule 4: IF "interest rate" = medium AND "trade fee" = significant
THEN "environment" = indifferent

Logically these rules produce

Rule 6: IF ("interest rate" - low OR "interest rate" = medium)
AND "trade fee" = significant
THEN “environment”=indifferent

Given the values of the interest rate R and trade fee **F**: **R=0.04** and **F=0.03**, i.e., exactly between picks of membership functions for low and medium interest rates, i.e.,

$$m_{\text{low-interest}}(R) = m_{\text{medium-interest}}(R) = 0.5, \quad m_{\text{significant-fee}}(F) = 1.$$

In this case, F definitely represents a significant fee and R is something between low and medium interest rate (see Figures 7.6 and 7.7).

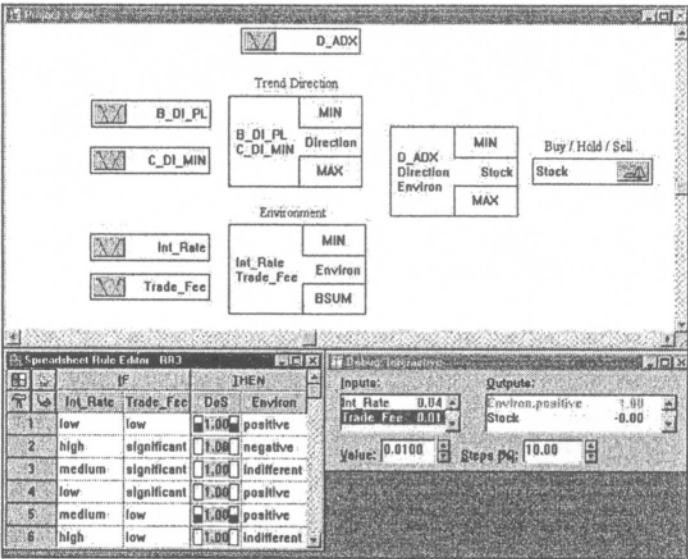


Figure 7.9. BSUM inference

MF were computed using Min-Max and FuzzyTech software [Von Altrock, 1997]. Both computations have produced the same

$$MF_{\text{positive-environment}}(R,F)=0.5.$$

Then MAX-MIN defuzzification delivered intuitively and logically consistent results for positive and indifferent trade environments separately. However, joint consideration of positive and indifferent trade environments produces inconsistent results. Rule 6 actually can be viewed as an OR combination (disjunction) of positive and indifferent trade environments. According to rule 6

$$MF_{\text{positive-or-indifferent-environment}}(R,F) = 1,$$

but MAX-MIN fuzzy inference can produce only 0.5.

Another inference, which uses BSUM handles this situation correctly. Nevertheless, it does not mean that BSUM is the right choice. An adequate analysis should involve a careful study of context. One of the reasonable **alternatives to heuristic debugging** is the use of **interpolation based on picks of membership functions** and values of output variable for these picks in three-dimensional space (see Figure 7.10).

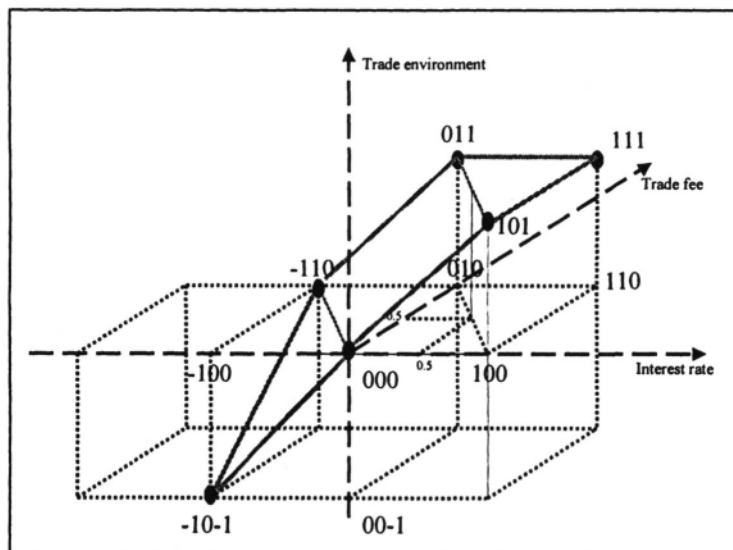


Figure 7.10. Surface presentation of rules

This idea is illustrated with the same example. Six rules presented in Table 7.5 can be converted into points in three-dimensional space. For in-

stance, 111 means that MF values for interest rate, trade fee and environment all equal to 1 and that if the first two MFs are equal to 1 then the third should be 1 also. Use axes for the interest rate, trade fee and trade environment. This also shows the simple surface connecting these points representing the degree of positive environment for every value of the interest rate and trade fees not only integer values. This means a defuzzified output is delivered directly. There are simple formulas to define this surface analytically. The given surface does not suffer from problems of inconsistency known for standard fuzzy inference. The surface delivers

$MF_{\text{positive-environment}}(R,F)=1$, for $MF_{\text{low-interest}}(R)=0.5$ and $MF_{\text{low fee}}(F)=0.5$, see Figure 7.10. Mathematical formalism, detailed justification and discussion of this interpolation method called the **second interpolation** are presented in [Kovalerchuk, 1996b] and briefly we present this concept in Section 7.7 in this chapter.

7.6. Constructing coordinated linguistic variables

7.6.1. Examples

In Section 7.1, six steps of implementing knowledge discovery based on fuzzy logic were presented. Below **coordinated contextural implementation** of the first four steps is presented:

1. **Obtaining** some rough knowledge (rules, statements) in linguistic form from an expert,
2. **Augmenting** linguistic concepts, statements and rules with numerical indicators (membership values and functions),
3. **Selecting** fuzzy inference mechanism (operations and defuzzification procedures), and
4. **Testing and debugging** rules and indicators interactively with an expert.

Although implementation of these steps is simpler without coordination of their contents, the resulting rules often are much less coherent. These rules require more work to test, debug, and tune them than if the steps have been coordinated from the very beginning. Below several examples illustrating the importance of coordinated approach are presented. Then a **mechanism** to implement steps 1-4 in a coordinated way based on concepts of the **context space** is suggested.

Example 1 (Inconsistency).

Let the *first step* (obtaining a linguistic statement) be ended with **statement S1**:

“A person of age 59 is Almost Old *OR* Old”.

Then in the *second step* the linguistic concepts “Almost Old” and “Old” are augmented with numerical indicators of their truth for age 59:

$$m(59 \text{ is Old}) = 0.55,$$

$$m(59 \text{ is Almost Old}) = 0.45,$$

and the statement S1 is augmented with a numerical indicator of the degree of its truth:

$$m(59 \text{ is Old } OR \ 59 \text{ is Almost Old}) = 1.00 \text{ (true)}. \quad (1)$$

Assume also that the *third step* ends with the standard fuzzy logic max operation for *OR*. Using OL for Old and AO for Almost Old, it produces (2):

$$\begin{aligned} m(59 \text{ is Old } OR \ 59 \text{ is Almost Old}) &= \max\{OL(59), AO(59)\} \\ &= \max\{0.55; 0.45\} = 0.55, \end{aligned} \quad (2)$$

i.e., truth of the statement “59 is Old *OR* 59 is Almost Old” is very questionable (truth degree 0.55). Nevertheless, more naturally statement S1 should be true or nearly true. Thus, steps 2 and 3 produced different results (1) and (2). Therefore, the *fourth step* (debugging) is needed, e.g., substituting max for SUM. Here (1) is supported intuitively and experimentally [Hall, et al, 1986], reflecting the given linguistic statement: “A person of age 59 is Almost Old or Old”, but (2) is based only on a formal (and not necessarily universal) fuzzy logic definition. The source of the **inconsistent output (2)** is in **implementation of steps 2 and 3 without coordination** of their contents and contexts. Max operation was chosen without analysis of the results of steps 1 and 2. Unfortunately, this is a standard practice in fuzzy inference. In other words, independently defining membership function *m* and *OR* operation as max created inconsistency. The negative aspect of debugging is that it is done ad hoc and no one can guarantee that with more age concepts such as “Not Old” or “Very Old” a similar problem will not raise.

The concern is that SUM could be just a first step in debugging and we can not universally substitute max for SUM. This is confirmed in the next example.

Example 2 (Case sensitive debugging).

Assume that we have substituted max operation for sum operation in the debugging of Example 1. Let us consider **statement S2**:

“A person of age 59 is Almost Old *OR* Old *OR* Very Old *OR* Not Old”

as a true statement (Step 1). Step 2 produced according to the figures in [Hall, et al., 1986] where:

$$\begin{array}{ll} m(59 \text{ is Old}) = 0.55, & m(59 \text{ is Almost Old}) = 0.45, \\ m(59 \text{ is Not Old}) = 0.35, & m(59 \text{ is Very Old}) = 0.06. \end{array}$$

Step 3 delivers for statement 2:

$$OL(59)+AO(59)+NO(59)+VO(59)=0.45+0.35+0.55+0.06=1.41 \quad (3)$$

Thus, the truth-value 1.41 in (3) instead of 1.0 also should be debugged.

Example 3 (Case sensitive debugging).

Step 1. Suppose we need m for the expression E , where E is a conditional (i.e., "if-then") statement such that

IF the age of 32 years is Not Old (NO)
THEN the age of 59 is either Almost Old *OR* Old.

Intuitively, this statement seems to be approximately true.

Step 2. Hall et al [1986] provide the following experimental values:

$$NO(32)=0.95, AO(32)=0.04, OL(32)=0.01.$$

Also we assume from (1) that $m(\text{age of 59 years is } AO \text{ OR } OL)=1$.

Step 3. The standard logical approach is to consider that E statement $(C \Rightarrow D)$ is equivalent logically to $(\text{not } C) \vee D$.

Here, $D = \{\text{age of 59 years is } AO \text{ OR } OL\}$;

and in our context, $(\text{not } C) = \{\text{age of 32 years is } AO \text{ OR } OL\}$.

This negation differs from a simple negation of $\{\text{age of 32 years is } NO\}$, i.e., it is not simply $\{\text{age of 32 years is } OL\}$. For the other context of $(\text{not } C)$, the other set of linguistic terms may take place.

Using the standard max-min fuzzy logic approach (max for *OR* and min for *AND*):

$$\begin{aligned} m(E) &= m((\text{not } C) \vee D) \\ &= m(\{\text{age of 32 years is } AO \text{ OR } OL\} \text{ or } \{\text{age of 59 years is } OL\}) \\ &= \max(m(\{\text{age of 32 years is } AO \text{ OR } OL\}), m(\{\text{age of 59 years is } AO \text{ OR } OL\})) \\ &= \max(\max[m(\text{age of 32 years is } AO), m(\text{age of 32 years is } OL)], \\ &\quad \max[m(\text{age of 59 years is } AO), m(\text{age of 59 years is } OL)]) \\ &= \max(\max(0.04, 0.01), \max(0.55, 0.45)) = 0.55. \end{aligned}$$

Therefore, the use max-mix operation gives only a formally "correct" answer (**m=0.55**) that certainly is NOT intuitively "correct". The reason for inconsistency is the same -- max-min operations were selected in step 3 without coordinating with steps 1 and 2. In addition, the use of the sum produces results, which should be debugged ($m > 1$):

$$\begin{aligned}
 m(E) &= m((\text{not } C) \vee D) \\
 &= m(\{\text{age of 32 years is AO OR OL}\} \text{ or } \{\text{age of 59 years is OL}\}) \\
 &= \text{sum}(m(\{\text{age of 32 years is AO OR OL}\}), m(\{\text{age of 59 years is AO OR OL}\})) \\
 &= \text{sum}(\text{sum}[m(\text{age of 32 years is AO}), m(\text{age of 32 years is OL})], \text{sum}[m(\text{age of 59 years is AO}), m(\text{age of 59 years is OL})]) \\
 &= \text{sum}(\text{sum}(0.04, 0.01), \text{sum}(0.45, 0.55)) = \text{sum}(0.05, 1.0) = 1.05
 \end{aligned}$$

Analysis of Examples 1-3. Debugging of fuzzy logic operations is needed in all these Examples 1-3. Example 1 brought us to substituting max for sum. Truth-value greater than 1 in Examples 2 and 3 can be fixed by substituting bounded sum for the sum -- BSUM (all values over 1 are cut on the level 1),

$$\text{BSUM}(x, y) = \min(\text{sum}(x, y), 1).$$

Our concern is that the BSUM may not be the final universal substitution for max either. The SUM was not the final one after Example 1 was debugged. Examples 2 and 3 required more debugging. The negative aspect of debugging is that it is done *ad hoc* and there is no guarantee that with more concepts such as "not young" and other ages such as 65 a similar problem will not rise. Examples 1-3 illustrate the coordination problem of linguistic statements and the third example illustrates coordination problem for rules. It is also important to note that **probability theory does not require a debugging** operation in similar situations. This is illustrated in Example 4 below.

Example 4 (Linguistic fuzzy-probabilistic approach).

In this example we apply fuzzy-probabilistic approach to Example 3 -- to find $m(\text{age of 32 is NO AND age of 59 is AO})$. We operate in the Cartesian product of the two probability spaces:

$$\{\text{NO, AO, OL}\} \times \{\text{AO, OL}\},$$

which is equivalent to all pairs

(NO,AO), (AO,AO), (OL,AO), (NO,OL), (AO,OL), (OL,OL).

Then we find membership functions (MFs) of those pairs with the first component for age 32 years and the second component for age 59 years.

According to the already cited data from [Hall, et al., 1986]: **NO(32) = 0.95**, and **AO(59) = 0.55**. If the given probability spaces are **independent**, then:

$$m(32 \text{ is NO AND } 59 \text{ is AO}) = NO(32) * AO(59) = 0.95 * 0.55 = 0.52.$$

Cheeseman [1985, 1986] made this kind of supposition in a similar situation. Thus, under independence assumptions of spaces for ages 32 and 59 years:

$$\begin{aligned} m(E) &= m((\text{not } C) \vee D) \\ &= m(\{\text{age of 32 is AO OR OL}\} \text{ OR } \{\text{age of 59 years is AO OR OL}\}) \\ &= [AO(32) + OL(32)] + [AO(59) + OL(59)] - [AO(32) + OL(32)] * [AO(59) + OL(59)] \\ &= [0.04 + 0.01 + 0.55 + 0.45] - [0.05 * 1.00] = [1.05] - [0.05] \\ &= 1.00. \end{aligned}$$

This **intuitively acceptable output (m(E)=1)** does not requires debugging of standard probabilistic operations for independent events – “+” for *OR* and “*” for *AND*. If the independence assumption is not true, then we should use conditional probability:

$$NO(32) * AO(59 / [NO(32)]) = 1.00.$$

Here, **AO(59/[NO(32)]) = 1.00**

means that 59 years of age IS "Almost Old", GIVEN THAT 32 years of age IS "Not Old".

Example 5 (Dependent terms). Let us compute truth-value for **statement 4**:

B = (age of 32 years is *NO AND* age of 59 years is AO)
 OR (age of 32 years is NO *AND* age of 59 years is NO)
 OR (age of 32 years is NO *AND* age of 59 years is OL).

Use of standard probabilistic operations under the same supposition of independence as in Example 4 produces:

$$m(B) = (0.95 \times 0.55) + (0.95 \times 0.35) + (0.95 \times 0.45) = 0.52 + 0.33 + 0.42 = 1.27,$$

i.e., debugging is obviously needed to get a truth value not greater than 1.

This example shows that a **simple substitution of fuzzy operations for probabilistic operations does not solve the problem of debugging** completely. This substitution as well as previous ones ignores **context** of steps 2 and 3. Actually the term Not Old can not be considered as independent from terms Almost Old and Old for age 59 as shown in experiments in [Hall et al, 1986].

Example 6 (Nested, complementary and overlapping linguistic terms). Below the assumption of Example 4 are used.

Step 1. First, two pairs of term sets are introduced for the ages of 32 years and 59 years:

$$T(32) = \{NO, AL\}, \quad T'(32) = \{NO, NO \text{ OR } AL\},$$

$$T(59) = \{AO, OL\}, \quad T'(59) = \{AO, AO \text{ OR } OL\}$$

Assume that two statements are true

Age of 32 is Not Old *OR* (Not Old *OR* Almost Old),
 Age of 59 is Almost Old *OR* (Almost Old *OR* Old)

For simplicity, NO *OR* AL will be denoted as NOAL and AO *OR* OL will be denoted as AOOL.

Step 2. Let $NO(32)=0.95$ and $AO(59)=0.55$ as in previous examples. It is also assumed that

$$m(\text{age 32 is NOAL})=1, \\ m(\text{age 59 is AOOL})=1.$$

We will call $T(32)$ and $T(59)$ **exact complete term sets** for 32 and 59, respectively, because of these properties. The $T(32)$ and $T(59)$ terms express **distinct concepts** and the prime terms in $T'(32)$ and $T'(59)$ express **nested concepts**. The set of exact complete terms is the base concept of an **exact complete context space**.

Step 3. Select sum to represent *OR* operation.

The sums for nested T' are more than 1:

$$m(\text{Age of 32 is Not Old OR (Not Old OR Almost Old)})=0.95+1=1.95 \\ m(\text{Age of 59 is Almost Old OR (Almost Old OR Old)})=0.55+1=1.55.$$

Therefore, debugging is needed for T' to suppress these sums to 1.
Step 4. One way of debugging is to return to max operation:

$$\begin{aligned} & m(\text{Age of 32 is Not Old OR (Not Old OR Almost Old)}) \\ &= \max(m(\text{age of 32 years is NO}), m(\text{age of 32 years is NO ORAL})) \\ &= \max(0.95, 1) = 1. \end{aligned}$$

Similarly,

$$\begin{aligned} & m(\text{Age of 59 is Almost Old OR (Almost Old OR Old)}) \\ &= \max(m(\text{age of 59 years is AO}), m(\text{age of 59 years is AO OR OL})) \\ &= \max(0.55, 1) = 1. \end{aligned}$$

Analysis of Example 5. Now we have obtained a corrupt loop -- to debug max we moved to the sum and to debug the sum we moved to max again. This example shows that simple “**context-free**” debugging trying different operations helps to solve one problem, but can create another. Therefore, **contextual debugging** or in other words, better contextual designing steps 1-4 is needed. Contextual debugging here would begin with noticing that the intuitively incorrect result obtained in Example 5 before debugging is entirely the result of having used “**context free**” computations. It happened because of **mixing distinct and nested terms** in a single context space and using the same space and term set for different ages (32 and 59 years). In other words, step 1 generated several statements, which require different operation for distinct and nested terms. For example, in the statement “A person of age 59 is Almost Old OR Old OR Very Old OR Not Old” terms Almost Old and Old are **distinct** ones, but terms Old and Very Old are **nested**. For **nested** terms, if someone is called Very Old he/she also can be called Old, i.e., here Very Old is interpreted as nested to Old. This is not the case for **distinct** terms Almost Old and Old. We can not say that if someone is Almost Old he/she is also Old or if he/she is old that he/she is almost old. Nested terms may require min operation, but distinct terms may require sum.

In the **mixed space** with nested and distinct terms, **there is no way to find one “context-free” operation in the process of debugging**. Switching between these operations would be needed. However, this is only a partial solution. The number of operations and switches can be as large as the number of statements in the domain. Moreover, if we combine two statements and they use two different operations, the new combined statement may require a third operation. Therefore, a context solution would be to analyze the term space type if it is a space of nested terms, a space of distinct terms or a mixture. **If the space is nested then use min-max and if space is dis-**

inct then use sum with some additional context restrictions. If it is a mix of terms then a transformation of the space to one of the previous types will be needed. To avoid all these complex problems it would be more natural to **develop a distinct or a nested context space in advance.** In the next section, we discuss known methods of developing these spaces.

Let us produce a space based on nested terms for the statement “A person of age 59 is Almost Old *OR* Old *OR* Very Old *OR* Not Old”. We may have five nested terms

“Almost Old *OR* Old *OR* very Old *OR* Not Old”
 $\supset$ “Almost Old *OR* Old *OR* very Old”
 $\supset$ “Old *OR* very Old” $\supset$ “Old” $\supset$ “Very Old”.

These nested terms are given experimentally only for “Old” and “Very Old”. Therefore, decomposition of the needed term “Almost Old *OR* Old *OR* very Old *OR* Not Old” to compute MF can not be done this way, but it works consistently for “Old *OR* very Old”:

$$m(\text{Old } OR \text{ very Old}) = \max(m(\text{Old}), m(\text{Very Old}))$$

Also, we can transform the term set from the statement “A person of age 59 is Almost Old *OR* Old *OR* Very Old *OR* Not Old” in to the set of distinct terms in “**exact complete context space (ECCS)**”.

For instance, let us consider terms C1-C4:

C1=“Almost Old”,

C2=“Old”,

C3= “Very Old and not Old” and

C4=“Not Old” and not (“Almost Old *OR* Old *OR* Very Old”).

Then knowing from THE mentioned experimental study that C1 and C2 form a complete space for age 59 we obtain a meaningful result:

$$\begin{aligned} & m(\text{“A person of age 59 is Almost Old } OR \text{ Old } OR \text{ Very Old } OR \text{ Not Old”}) \\ &= m(\text{“A person of age 59 is Almost Old”}) + m(\text{“A person of age 59 is Old”}) \\ &+ m(\text{“A person of age 59 is “Very Old and not Old”}) \\ &+ m(\text{“A person of age 59 is Not Old and not(Almost Old } OR \text{ Old } OR \\ &\quad \text{Very Old”}) \\ &= 0.55 + 0.45 + 0 + 0 \end{aligned}$$

7.6.2. Context space

Examples in the previous section show that the max-min fuzzy logic approach produces the same result as the “exact complete” context space ap-

proach if (and only if) **nested spaces** are accurately constructed with a one-to-one correspondence to the **ECCS**.

Thus, whenever natural language terms are mostly nested for a given applied problem, we should try to construct the complete nested term set (sets) within the whole context. That is, the entire context should be equivalent to an ECCS. The correspondence between ECCS and a nested context is close to correspondence between a **density function** and a **distribution function** in probability theory.

The above examples should suffice to show that probability spaces could be constructed for the linguistic uncertainties usually given empirically in situations involving fuzzy sets. Thus, in this regard, Cheeseman's probabilistic position [Cheeseman, 1985, 1986] is justified. We should also emphasize that this is a typical situation for many applications of fuzzy logic. We can construct exact complete context space(s) (as in examples, above) without serious problems, and such contexts likely do not change during their "lifetimes".

On the other hand, as Dubois and Prade noted [1990], in dealing with knowledge base problems, it is not uncommon to have to deal with the complete probability spaces. Thus, in such cases, it is **often impossible to specify probability spaces**, i.e., they may well be dynamically changing and/or do not provide a "complete" context. However, it will be important to have some real examples from knowledge base applications where it is truly impossible to construct "complete" context spaces. Such examples would provide a justification of the Dubois and Prade "impossibility" position [1990].

Nonetheless, the examples offered in the previous section show that without specification of "exact" complete context space, many mistakes can be made in calculating compound membership functions (cf. Examples 2-5 above).

The complete context is of importance and has considerable applied interest for fuzzy inference. The combination of "nested" T(32) and T(59) with negation is "exact complete". T(32) introduced in example 5 gives an intuitively correct result using the favored use of max-min in fuzzy logic.

We certainly can suppose that the sum will be about 1.00 for the offered T', and thus the "over completeness" of natural language can (and should be) "corrected" for the given example. Such a correction is especially effective whenever we construct artificial linguistic variables in fuzzy inference applications.

Financial applications grant a wide range possibilities to generate new linguistic variables like those presented in Figures 7.6 and 7.7 for interest rate, trade fee and related trading environment linguistic variable.

An **"over complete"** space will be generated if we will use a linguistic space {NO, AO, OL} for both ages and all pairs:

$$\{\text{NO, AO, OL}\}_{32} \times \{\text{NO, AO, OL}\}_{59}$$

The term Not Old (NO) is redundant for age 59 and term Old (OL) is **redundant** for age 32.

For "incomplete" spaces, however, we cannot even calculate m for complex expressions because not all of the required components are available. Furthermore, we should not use the idea of independence [Cheeseman, 1985,1986] simply because we "don't know" what else to assume. In addition, clearly, we cannot give equal probabilities to additional formal elements of the space, as was shown by Dubois and Prade [1990].

Let us define concepts of **context space** more specifically. Consider the following examples as providing a possible empirical base for context spaces. These cases were generated from Hall, Szabo, Kandel [1986]. Given the probability space for the age of 59 years, with additivity [Hall, et al. 1986; Figures 1-4]:

$$\text{OL}(59) + \text{AO}(59) = 0.55 + 0.45 = 1.00$$

where:

$$\text{OL}(59) = m(59/\text{Old}), \text{ and } \text{AO}(59) = m(59/\text{Almost Old}).$$

That is, OL(59) means that 59 years of age is taken to be "Old", and AO(59) means that 59 years is considered "Almost Old". Here, the pair {AO, OL} will be considered as providing a **"complete"** (**"exact complete"**) **context** for the age of 59 years. Let us add a new term Not Old (NO). Then, the context specified by the triplet (3-tuple) {NO, AO, OL} would be designated as an **"over complete"** **context**, and the context designated by {OL} would be an **"incomplete"** **context**. Thus, given the following with respect to the age of 32 years:

$$\text{NO}(32) + \text{AO}(32) + \text{OL}(32) = 0.95 + 0.04 + 0.01 = 1.00,$$

$$\text{NO}(32) + \text{AO}(32) + \text{OL}(32) + \text{VO}(32) = 1.00$$

By definition, the contexts {NO, AO, OL} and {NO, AO, OL, VO} would be "exact complete context spaces" for the age of 32 years. Finally according to the figures in Hall, et al. [1986], contexts are incomplete for ages 18 and 24 years, and are over complete for ages 40, 51, 59, 68, 77. For example,

$$OL(59)+NO(59)+AO(59)+VO(59)=0.45+0.35+0.55+0.06=1.41. \quad (4)$$

More formal definition of a context space is given in [Kovalerchuk, 1996a].

7.6.3. Acquisition of fuzzy sets and membership function

One of the most important problems in fuzzy set theory is the problem of obtaining appropriate MFs. Currently there are many known ways of obtaining MFs; but sometimes they have been a source of misunderstanding. For example, consider the statement: "Fuzzy set theory does not require the sum of m values on X to be equal to 1.00". Indeed, for many purposes there is no problem if this sum exceeds 1.00. Clearly, however, the important question is what we want to do with m values. For instance, let $\mathbf{m}_A(\mathbf{x}) = 0.70$, and $\mathbf{m}_A(\mathbf{y}) = 0.90$; if we only want to conclude that the "degree of belief for x is less than that for y for A to be the case, then there is no problem with the sum here being > 1.00 . How m 's are obtained and how we use them is the critical issue. In this present example, we can use any monotone transformation of the initial m values with impunity. For example, it could be that $\mathbf{m}_A(\mathbf{x}) = 0.95$ and $\mathbf{m}_A(\mathbf{y}) = 0.99$, or even that $\mathbf{m}_A(\mathbf{y}) = 1.20$.

However, consider a fuzzy inference that computes the center-of-gravity (CoG) to create a defuzzified decision, a procedure that requires the addition of m values weighted by w_x and w_y . Now it becomes important to justify addition and multiplication operations with the available m 's; i.e.,

$$w_x \mathbf{m}_A(\mathbf{x}) + w_y \mathbf{m}_A(\mathbf{y}).$$

Of course, the center-of-gravity will differ under any of the various monotone transformations just suggested in the prior example, and now it clearly is **necessary to find a procedure for obtaining values for m that appropriately allow addition and multiplication**. We suppose that this procedure should be operational, i.e., close to ideas of Bridgeman's [1927] operationalism in physics.

Fuzzy statistics [Hall, *et al.*, 1986; Hisdal, 1984] provide several approaches to the acquisition of MFs. Let's consider the following:

- a) Values for specific linguistic statements such as " x is a low rate" are obtained by asking members of a group to register their agreement/disagreement with the statement. Responses are **binary** (i.e., "yes" or "no").
- b) Subjects are asked to rate their agreement/disagreement with the statement on a **continuous scale** of 0 to 1. Then, using mathematical techniques, statistical characteristics of the group of subjects are computed.

Values obtained in this way are sometimes referred to as **"fuzzy expected (average) values"** (FEVs).

More exact definitions of the **operational approaches** for obtaining values of m 's via both "semantic procedures" [cf., Hisdal, 1998] and "modal logic" [cf., Resconi, *et al.* 1992, 1993] will be given in the following sections. We will show that these two approaches, when used in conjunction with the concept of exact complete context space, will allow for the appropriate and effective employment of fuzzy logic in many situations that might otherwise be highly controversial, at best. Hisdal [1998] introduced **semantic operational procedures** for defining fuzzy sets. It is necessary here for at least two persons to participate: The **"knowledge engineer"** (E), who gives instructions to the **subject(s) S** ($S_1, S_2, \dots$), and every procedure is performed on a set of **objects**. For example, the "objects" could also be people. Every semantic procedure must have a **reference label set** $\Lambda = \{\lambda\}$. For example, $\Lambda = \{\text{young, middle-aged, old}\}$, and the λ_i 's are called labels.

Labeling (LB) procedure. In psychophysics, this is a forced- (multi-) choice procedure.

Example: E instructs S to answer the question **"What is today's interest rate?"** by choosing one of the labels -- $\Lambda = \{\text{low, medium, high}\}$. (The requirement of referring to Λ pertains to the next three definitions as well.)

Example: E instructs S to answer the question "What is John's age?" by choosing one of the labels -- $\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_i, \dots, \lambda_L\}$. (The requirement of referring to Λ pertains to the next three definitions as well.)

Yes-No (YN) procedure. In psychophysics, this is **forced two-choice** procedure.

Example: E asks, **"Is today's rate high?"** S answers either "yes" or "no".

Example: E asks, "Is John old?" S answers either "yes" or "no".

LB-MU procedure In psychophysics, this is a kind of **direct scaling procedure**.

Example: E asks S to **specify** the **"degree"** (L) to which each of the **labels** from Λ is **appropriate** for today's interest rate. That is $m_{\lambda_i}(\text{today}) \in [0; 1]$,

where $m_{\lambda_1}(\text{today}) = m_{\text{low-int-rate}}(\text{today})$, $m_{\lambda_2}(\text{today}) = m_{\text{medium-int-rate}}(\text{today})$ and $m_{\lambda_3}(\text{today}) = m_{\text{high-int-rate}}(\text{today})$).

Example: E asks S to specify the "degree" (L) to which each of the labels from Λ is appropriate to John (i.e., $m_{\lambda_i}(\text{John}) \in [0, 1]$).

YN-MU procedure. In psychophysics, this is a kind of **direct scaling procedure**.

Example: E instructs S to **specify the degree $m_{\text{yes-high-rate}}(\text{today}) \in [0,1]$ to which the answer "yes" is correct to the question "Is interest rate high today?"**

Example: E instructs S to specify the degree $m_{\text{yes-young}}(\text{John}) \in [0,1]$ to which the answer "yes" is correct to the question "Is John young?"

An alternative way to construct a membership function is to use the **modal logic approach** [Resconi, et al,1992]. First, consider a finite universal set X and a set of propositions

$$\langle X, \{e_A(x)\} \rangle,$$

where $e_A(x)$ is of the form

$e_A(x)$: "Given element x is classified in set A ",

with $x \in X$, $A \in P(X)$, and $P(X)$ is the power set of X .

Also, let

w_i be the "world",

$v_i(e_A)$ be the truth value assigned to the proposition e_A for a given set $A \in P(X)$ in the world w_i , and

- $T[e_A]$ be the number of worlds in which $v_i(e_A) = T$; so, $T[e_\emptyset] = 0$, $T[e_X] = n$.

Now, a function m , defined by $m_A(x) = T[e_A]/n$, is called a **membership function (MF)**.

Matching semantic operational procedures and modal logic. Despite significant differences between the language of semantic operational procedures and modal logic language, these approaches lead to similar membership functions concepts. There are two ways to match these concepts:

- 1) set up a one-to-one correspondence between **subjects**, S_i in semantic approach and **worlds**, w_i , in modal logic approach.
- 2) set up a one-to-one correspondence between a **class of worlds** $\{w_i\}$ and a given subject, S . In the latter case, it is viewed as if a given subject (S) is providing estimates for **different conditions** ("worlds"), $\{w_i\}$. The second approach was developed in [Kovalerchuk, Berezin, 1993].

7.6.4. Obtaining linguistic variables

First, it should be noted that the reference to Λ pertains to all four procedures defined in the semantic approach. The "novelty" of the critical concept we are presenting is the requirement that Λ forms an ECCS; i.e., the following property holds for all x from X :

$$m(e_{\lambda_1}(x)) + m(e_{\lambda_2}(x)) + \dots + m(e_{\lambda_n}(x)) = 1.00.$$

For one approach to finding this kind of set Λ , see Example 5 in Section 7.6.1 above. Specifically, such an approach allows the LB (labeling) procedure to provide a probability space for each object x . Sometimes, when actually using the YN (yes-no) procedure, E (knowledge engineer) asks, "Is interest rate high today?", without reference to Λ -- but that produces highly undesirable results. Consider the following two sets of labels, T and T'' :

$T = \{\text{very very low, very low, low, medium, high, very high, very very high}\},$

$T'' = \{\text{low, medium, high}\}.$

Here, low and high in T and T'' are different. So, if E asks: "Is interest rate high today?", **without establishing context term sets T or T''** (i.e., the Λ), the results will clearly be non-interpretable (see the examples in Section 7.6.1). Therefore obtaining linguistic variables should consist of the following steps:

Step 1. Determine a reference set Λ .

Step 2. Define initial type of the reference set Λ -- distinct, nested or mixed.

Step 3. Correct Λ to bring it into distinct or nested type.

Step 4. Obtain MFs for all terms of corrected Λ using semantic or modal logic operational procedures.

Step 5. Define the type of the corrected reference set Λ , i.e., distinct, nested or mixed.

Step 6. Correct Λ again to bring it into distinct or nested types repeating steps 4,5 and 6 until getting an acceptable exact complete context space.

Conclusion. For correct inference under linguistic uncertainty in applications, it is very useful and necessary to construct exact complete context or their nested equivalent spaces. Unless one specifies the appropriate, **necessary context space** in which s/he is working, correct inferential solutions cannot be clearly arrived at, nor can one establish a clear foundation upon which to debate the efficacy of the problems under consideration. Further

development of the approach outlined in this section should deal with additional requirements on the components of specifiable context spaces.

7.7. Constructing coordinated fuzzy inference

7.7.1. Approach

Six steps of implementing knowledge discovery based on fuzzy logic were presented in Section 7.1. These steps include:

- **Selection and debugging of a defuzzification operator** and
- **Tuning membership functions** of fuzzy operations used in rules.

Often selection, debugging and tuning are done by "blind" optimization. Several studies have shown the disadvantages of "blind" optimization, including neural networks and genetic algorithms [Pfeiffer, Isermann, 1993; Pedrycz, Valente de Oliveira, 1993]. It was shown that simultaneous optimization of both Input/Output interfaces and a linguistic subsystem without integrity constraints can generate meaningless linguistic terms [Pedrycz, Valente de Oliveira, 1993]. Complicated defuzzification using CoG increases the problems of effective computation [Pfeiffer, Isermann, 1993; Bugarin et al, 1993, Tilli, 1993]. Our intention is: (1) to **eliminate heuristic procedures in debugging and tuning** as much as possible and (2) to **simplify the related computations**. Elimination of critical heuristic procedures will help to improve the general design methodology. Simplification of these procedures will allow more effective implementation of fuzzy inferences, which includes decreasing runtime. In Section 7.5 intuitively unacceptable

$$MF_{\text{positive-environment}}(R,F) = 0.5$$

for trading environment was debugged and acceptable

$$MF_{\text{positive-environment}}(R,F) = 1$$

was obtained after debugging. **Heuristic debugging** in that example consists of **trying different fuzzy operations** and finally substituting MAX for BSUM. This trial approach was implemented in FuzzyTech [Von Altrock, 1977]. Even if BSUM debugs $MF_{\text{positive-environment}}(R,F)$ perfectly for given R and T there is no guarantee that the need for further heuristic debugging is eliminated for other R and T. For that reason, a better way for solving the problem would be a **theoretical justification** for a fuzzy operation such as

sum or BSUM along with their empirical confirmation. This avenue is supported by the theoretically justified sum for an exact complete context space (ECCS) discussed in Section 7.6. Therefore, one of the solutions for well-grounded debugging would be **debugging MFs and linguistic variables for obtaining ECCS instead of heuristically changing an operation**. In this approach, the debugging rules will be needed only for correcting a linguistic rule table. For instance, in Table 7.7 an expert may figure out that Then-part “indifferent” should be substituted for “positive”.

Another reasonable **alternative to heuristic debugging and tuning is the use of interpolation based on picks of membership functions** and values of the output variable for these picks. In Section 7.5, we already have shown that an acceptable $MF_{\text{positive-environment}}(R,F)=1$ was obtained after this interpolation (see Figure 7.10).

Justification. There are two major reasons for using the interpolation: to make defuzzification intuitively consistent and to simplify the computation of fuzzy inference. The common in fuzzy logic CoG defuzzification method produces an intuitively **inconsistent wave** in the output function [Kovalerchuk, 1996b]. There is no meaning for this wave for many applications, in particular, for the trading environment as a function of R (interest rate) and T (trade fee). The wave is inconsistent with the intuitive idea of sustained monotone growth of a trading environment indicator with improvement of R and T . Even if the wave is small, it is a warning about potential undiscovered problems.

The **interpolation** shown in Figure 7.10 **does not have a wave** at all. Therefore, it is better justified intuitively. Moreover, it is much **simpler** than CoG for computation. As we already mentioned this method of interpolation is called the second interpolation [Kovalerchuk, 1996b].

Debugging MFs and linguistic variables by an expert can be insufficient and could require the use of training data for **tuning fuzzy inference**. In many **cases**, training data should be generated actively to improve tuning.

Algorithm. The algorithm for debugging and tuning rules based on the second interpolation and ECCS consists of several procedures:

Procedure 1. Identify Exact Complete Context Space (ECCS) with triangular membership functions (see Section 7.6)

Procedure 2. Construct symmetrical ECCS.

Procedure 3. Identify monotone rules.

Procedure 4. Transform ECCS to meet requirement of monotonicity.

Procedure 5. Generate training data for pick points, 0.5-points and 0-points of input MFs and linguistic variables in symmetric ECCS.

Procedure 6. Test linguistic rules against training data.

Procedure 7. Tune linguistic rules for matching training data.

Procedure 8. Interpolate pick points of debugged output MFs of ECCS (see Figure 7.10).

7.7.2. Example

Below we present main procedures for debugging and tuning based on interpolation using the same example as in Section 7.5.

Procedure 1. Identify Exact Complete Context Space with triangular membership functions. The output linguistic variable "trading environment" presented in Figure 7.11 was designed in the forms of ECCS. It does not require any additional work to build ECCS.

Also both input linguistic variables "interest rate" (Figure 7.6) and "trade fee" (Figure 7.7) satisfy ECCS requirements by design. For example, according to Figure 7.11 IF "trading environment" = 0.3 THEN

$$MF_{\text{negative-trad-env}}(0.3) + MF_{\text{indiff-trad-env}}(0.3) + MF_{\text{positive-trad-env}}(0.3) = 0 + 0.7 + 0.3 = 1$$

Procedure 2. Construct symmetrical ECCS. The output linguistic variable "trading environment" (Figure 7.11) and one the input linguistic "trade fee" were designed with symmetrical slopes, but linguistic variable "interest rate" does not have symmetrical slopes (Figure 7.6). This variable is transformed to an artificial scale by one-to-one mapping, where 0.02 is matched with -1, 0.05 with 0 and 0.1 with 1.

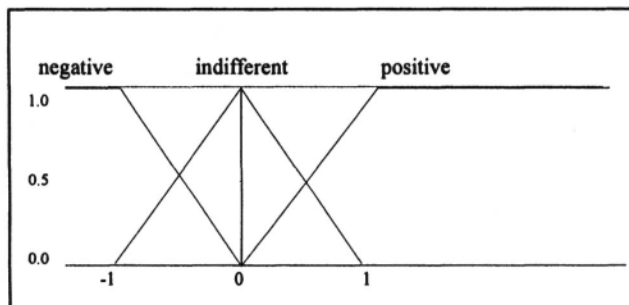


Figure 7.11. Membership functions for linguistic variable "trading environment"

Procedure 3. Identify monotone rules. Tables 7.3, 7.4 and 7.5 and Figure 7.8 in Section 7.4 already have identified monotonic rules.

Procedure 4. Transform ECCS to meet requirement of monotonicity. Technically, it requires correcting only MFs for "interest rate" linguistic variable obtained in procedure 2 by rotating MFs as shown in Figure 7.12.

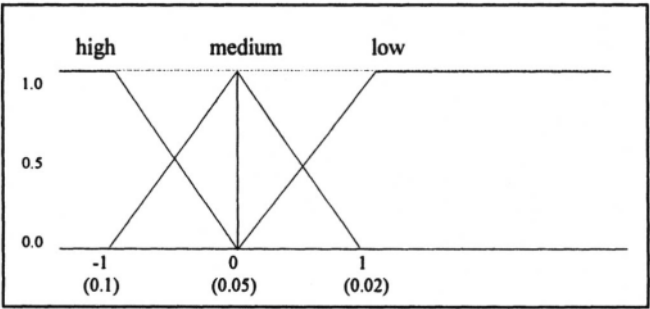


Figure 7.12. Transformed membership functions “low”, “medium”, “high” for linguistic variable “Interest rate”

Procedure 5. Generate training data for pick points, 0.5-points and 0-points of input MFs and linguistic variables in symmetric ECCS.

For this example, it is assumed that training data are presented in Table 7.7.

Procedure 6. Test linguistic rules against training data.

Training data in Table 7.7 show some difference with originally suggested rules (Tables 7.3 and 7.4). This difference is in brackets in the environment column in Table 7.7.

Table 7.7. Training data for numerical rule table

IF-part		Then-part
Interest Rate	Trade Fee	Environment
1	1	1
-1	0	-1
0	0	0
1	0	1
0	1	(0)
-1	1	0

Procedure 7. Tune linguistic rules for matching training data. Now one has an option to tune original rules in Table 7.4 and make them consistent with the training data (Table 7.7) if one trusts these training data more than the original rules.

This is not obvious, because training data can be corrupted by noise. In this example, we assume that data are really corrupted by noise and Table 7.4 should not be changed for Table 7.7.

Procedure 8. Interpolate pick points of debugged output MFs of ECCS. The interpolation for rules from Table 7.4 is already presented in Figure 7.10.

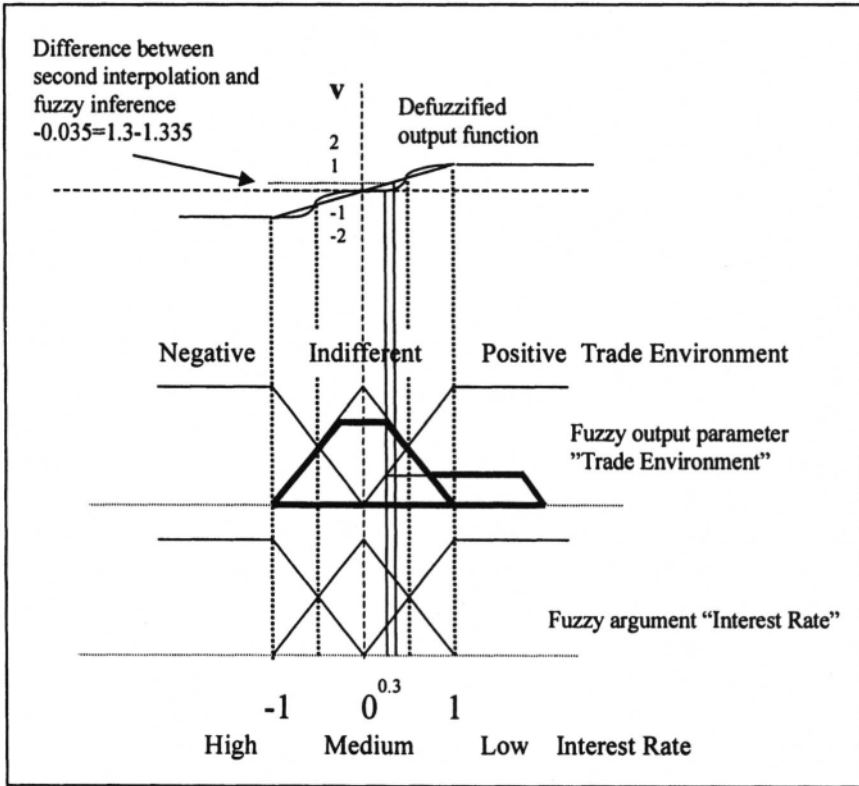


Figure 7.13. Output function for exact complete fuzzy sets context for significant trade fee.

Figures 7.13 and 7.14 show projections of that interpolation. The first projection is for the significant level of trade fee and the second one is for low trade fee. Both figures show interpolations and CoG output function with waves.

7.7.3. Advantages of "exact complete" context for fuzzy inference

There are several advantages of using the exact complete context space concept to obtain fuzzy sets and linguistic variables:

- time and memory for input data,
- simple Input-Output function,
- complete set of "If-Then" rules,
- reliability of the output.

Below these advantages are analyzed using several examples.

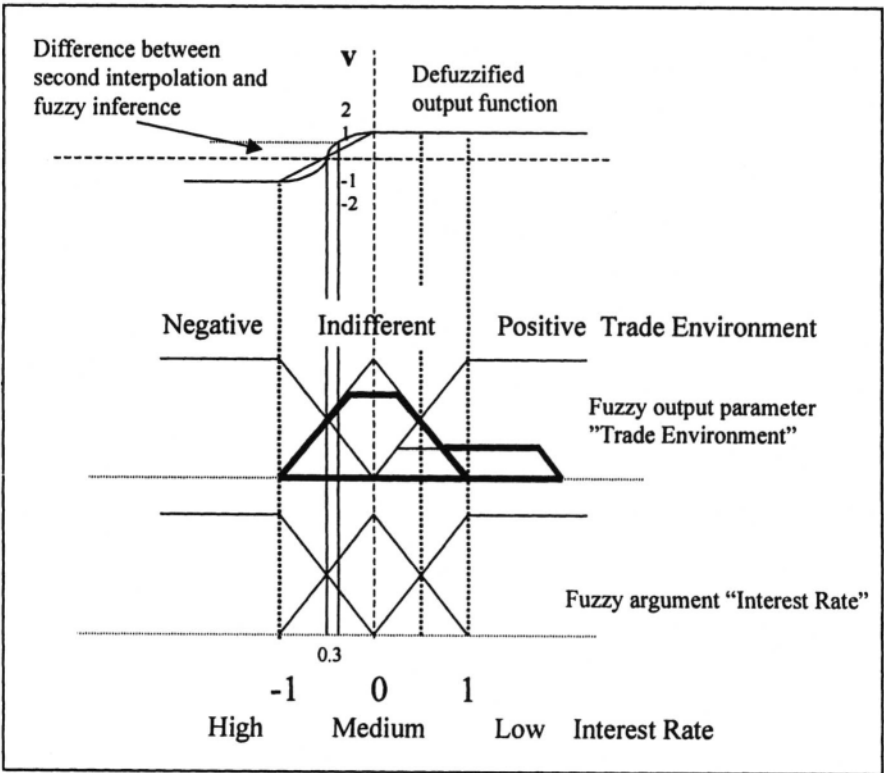


Figure 7.14. Output function for exact complete fuzzy sets context for low trade fee

Time and Memory for Input Data

Example. As noted earlier, context for output and input parameters often is represented as shown in Figure 7.15. In Figure 7.15, NL is "negative large"; NM is "negative medium"; NS is "negative small"; ZR is "approximately zero"; PS, PM, and PL are similarly abbreviated for the positive statements. Notice that each $x \in X$ has an "exact complete" context. For example, for x shown in Figure 7.15:

$$NM(x) + NS(x) = 1.00;$$

and, for this x , for all other fuzzy sets,

$$NL(x) = ZR(x) = PS(x) = PM(x) = PL(x) = 0.$$

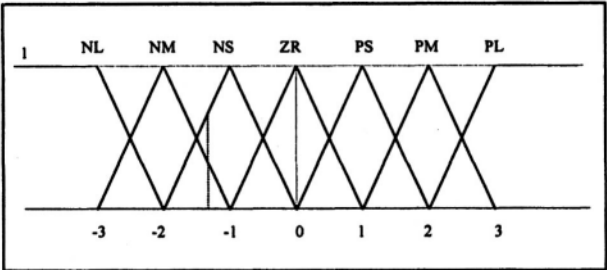


Figure 7.15. Example of "exact complete" context.

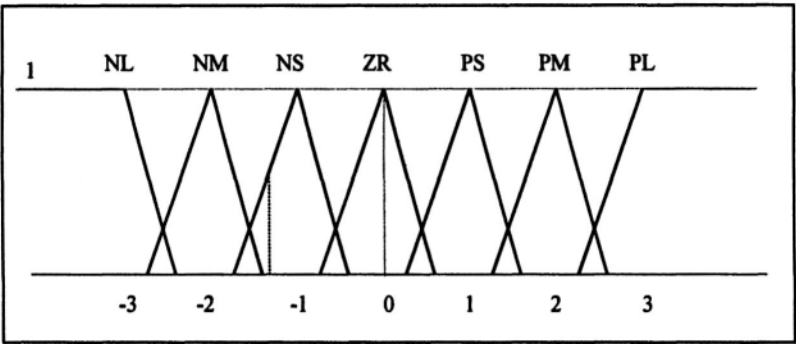


Figure 7.16. Partially-overlapping fuzzy sets (incomplete context space)

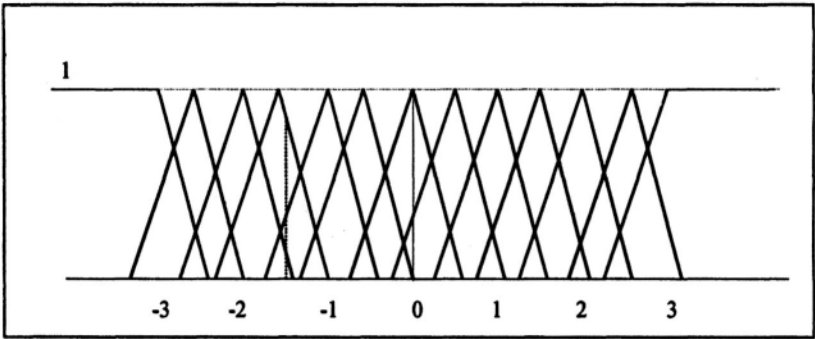


Figure 7.17. Overlapping fuzzy sets (over complete context space)

NL(x) and NS(x) can be interpreted as "subjective probabilities" which form a probability space for a given x. Thus, the first advantage of this is that the same level of objectivity, stability, and context dependence is achieved here as in probability theory. The seven membership functions (MFs) shown in

Figure 7.15 represent all classes of such simple probability spaces. We say that all seven of these fuzzy sets form context (context space) which we define more formally below.

Linguistic variables shown in Figures 7.16 and 7.17 have no this property, they represent incomplete context space ($\text{sum} < 1$) and over-complete spaces ($\text{sum} > 1$).

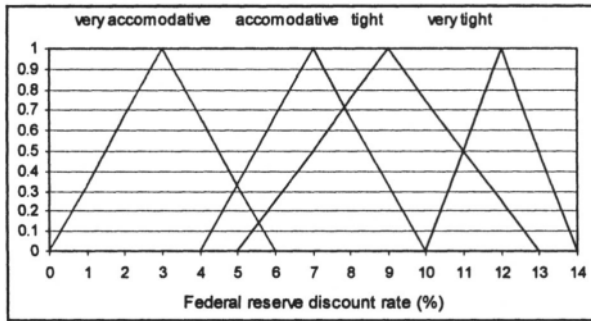


Figure 7.18. Linguistic variable discount rate (mixed context space)

Figure 7.18 represents practical assignment of MFs [Von Altrock, 1997]. This is a typical mixed case. For example for rates under 5% and from 13 to 14% the space is incomplete, from 7% to 9% it is over complete and for 11% it is exact complete context space.

A second advantage of this approach is the very compact representation of the class of probability spaces needed. Let us illustrate this point. Let X be a set of 100 grades (elements); now, instead of 100 separate probability spaces with probabilities of two elementary events each, we simply use seven MFs. In the first case, we would have to store at least 100 numbers. In the second case, we need store only seven numbers $(-3, -2, -1, 0, 1, 2, 3)$. All other numbers are computed via linear functions based on these seven. Because of such compact representation, much less computer memory is needed. We also can easily compute values of MFs for intermediate points, which are not included in our 100 points.

A third advantage is the time saved by the expert who has to provide m values. An expert would have to give answers about elementary probabilities for 200 events in the first case, but for the triangular MFs in Figure 7.19 the expert needs to only give seven numbers. While linguistic variables that are used in fuzzy inference can be represented by "over complete" or "incomplete" parameters, the majority of real fuzzy control systems are based on "**exact complete**" context space (ECCS), as shown in Figure 7.15. This fact clearly relates to the advantages already noted, as well as with some others that will be shown below.

Input-Output Function. Let us show the **simplicity** of the input-output function of a CoG max-min inference method based on an ECCS. Such an input-output function is a quasi-linear, piece-wise function [cf., Kovalerchuk et al, 1993,1994; Raymond et al, 1993] with fewer nodes than would be required for "incomplete" and "over complete" context spaces. It also is similar to a linear function between nodes. Thus, the usual CoG input-output function can be changed into a piecewise, quasi-linear function, and therefore, reject the complicated and heuristic CoG defuzzification procedure typically applied. Consequently, run-time and cost of fuzzy inference can be significantly reduced with such changes.

Figures 7.19-7.21 show the three different context spaces in the bottom and output interpolation functions in the top. The context in Figure 7.19 is presented with crisp not fuzzy sets. These sets do not overlap. They produce a step-function as an output. Figure 7.20 presents context with partially overlapping fuzzy sets. These sets produce a piece-wise linear interpolation with turning points at the beginning and end of overlapping areas. Figure 7.21 presents overlapping of MFs for exact complete context space (ECCS). These sets produce a **simplest** linear interpolation. This interpolation is the simplest and it is very close to the wave in the same Figure 7.21, which is the output for CoG max-min fuzzy inference.

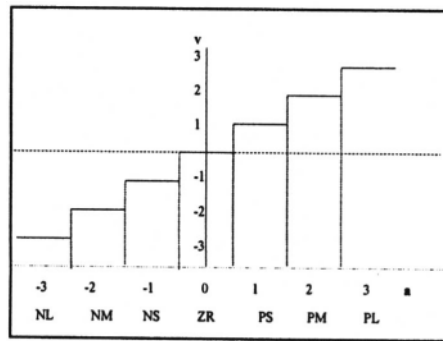


Figure 7.19. Output function for complete crisp sets context

Figure 7.19 shows a CoG output function for non-overlapping intervals. It is a simple step function. In this example, any input is presented as an interval and any output as an exact number. Also, output rules are: "IF argument a is about zero (i.e., $[-0.5, +0.5]$ covers a), THEN output u should be 0"; "IF argument a is positive small (i.e., $[+0.5, +1.0]$ covers a), THEN output u should be 1". The other rules are formulated similarly. In this simple case, the fuzzy inference method and the pure interpolation method give the same output function. Then we do not have a problem choosing one of them.

A fuzzy output function is presented for partially overlapping fuzzy sets in Figure 7.20. Here output rules are slightly different. Let us describe one of them: "IF argument a is positive small THEN output u should be also positive small". The term "positive small" (PS) for argument a is formalized with a fuzzy set in the bottom of Figure 7.20. The term "positive small" for output u is also formalized with a fuzzy set. This fuzzy set is presented above PS fuzzy set for argument a in Figure 7.20. The next output rule is

IF argument a is positive medium
THEN output u should also be positive medium.

The term "positive medium" (PM) for argument a is formalized with a fuzzy set in the bottom of Figure 7.20 next to the fuzzy set for PS. The term "positive medium" (PM) for output u is also formalized with a fuzzy set. This fuzzy set is presented above the PM fuzzy set for argument a in Figure 7.20. The other output rules are defined similarly. Figure 7.20 shows that the fuzzy sets "positive small" and "positive medium" are overlapping on the part of their supports. Similarly, Figure 7.21 shows MFs with ECCS overlapping. Output functions for the fuzzy inference method and the pure interpolation are shown above the fuzzy sets. The output function for the fuzzy inference method has a small wave in the area where PS and PM fuzzy sets are overlapped. The pure interpolation method gives a straight line in this area. Out of the overlapping areas, the fuzzy inference method and the pure interpolation method give the same linear pieces.

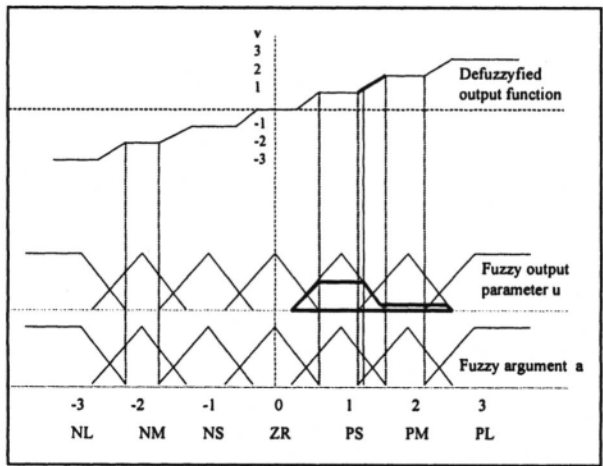


Figure 7.20. Output function for incomplete fuzzy sets context

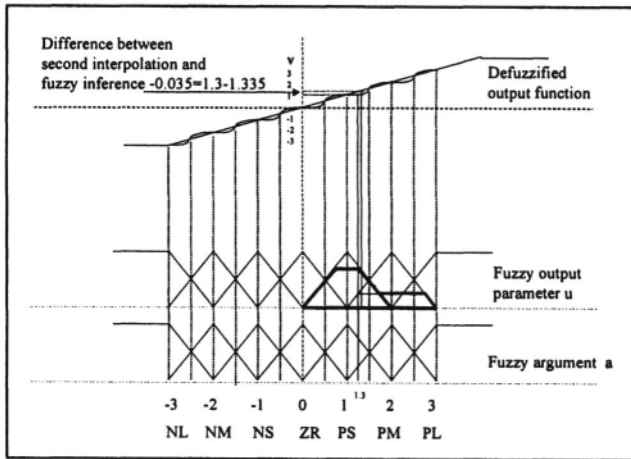


Figure 7.21. Output function for exact complete fuzzy sets context

The difference between these two interpolations in the overlapping area is described with the direct computational formula [Kovalerchuk, 1996b]:

$$u = [(1 - (\rho + \varepsilon))^2 + \rho(1 + \varepsilon)[2 + (1 + \varepsilon - \rho)]] / [(1 - (\rho + \varepsilon))^2 + \rho(1 + \varepsilon)],$$

where ε is the distance between the peak point of PS fuzzy set and the beginning of the PM fuzzy set and ρ is the distance from the beginning of the PM fuzzy set to the input point a , for which we compute a value u of the output function. The last formula is important to show that the difference between the fuzzy inference and the pure interpolation is no more than 2.07% of the length of the support of the used fuzzy sets for this single input single output (SISO) case. The pure interpolation method gives a single straight line. The difference between these two interpolations in the overlapping area is described with the formula with $\rho \leq 0.5$:

$$u = (2 - 3\rho^2 + 5\rho) / 2(1 - \rho^2 + \rho),$$

where p is the distance from the beginning of the fuzzy set to the input point a , for which we compute a value u of the output function [Kovalerchuk, 1996b]. This maximum deviation 2.07% is reached for $\varepsilon = 0$ and two values of ρ : $\rho_1 = 0.197200388$ and $\rho_2 = 0.8022799611$.

For two inputs and single output (TISO) case the difference between a fuzzy output function and a pure interpolation is no more than 5.05% of the support [Kovalerchuk, 1996b]. This study shows how to combine fuzzy inference and pure interpolation to construct simple output functions. If fuzzy sets have a large support then the difference between these two inter-

polations can be significant and the "wave" on Figures 7.20 and 7.21 can be relatively large. In this case, we need to decide which of the two output functions (CoG output function or the second interpolation) should be used. We argue that a piecewise linear interpolation between peak points of fuzzy sets has an important advantage. We consider an output task where all fuzzy sets are equal and symmetrical. How can fuzzy logic procedures generate a wave? How to explain the wave in CoG output function in terms of a particular applied inference task? There is no such explanation. The only explanation is out of context of the particular task. The source of the wave is CoG defuzzification procedure. Piecewise linear interpolation between peak points does not have this weakness.

Let us summarize. In many applications there is model and relevant data are very restricted. In this situation, it is practically reasonable to combine fuzzy inference and pure interpolation methods. Fuzzy inference methods are used to choose interpolation points and pure interpolation methods are used to interpolate between these points. With fuzzy inference methods, one extracts linguistic rules, construct respective fuzzy sets preferably as in Figure 7.21, identifying their peaks. Then interpolation methods are used to identify an output function interpolating between these peak points. If the constructed fuzzy sets meet the above-mentioned requirements, we do not need CoG procedure to have practically the same output function as CoG output function.

Figures 7.19-7.21 illustrate conditions when the second interpolation can simplify a fuzzy inference, thus **substituting the fuzzy inference**. Some fuzzy logic practitioners noticed that quite often output of CoG fuzzy inference and piecewise linear interpolations are very close. We estimated this difference explicitly in terms of the length of the fuzzy sets for one and two-dimensional cases. What is new in this study if it is already known? Now the phenomenon has explanation and conditions are known when this phenomenon takes place. For instance, 5.05% of the length of the fuzzy set can be checked if it is too large for a particular application. Therefore, this study gives a way to evaluate the **number of needed linguistic terms** for constructing fuzzy sets. Next, it explains the reason why CoG fuzzy inference is effective in very many applications with sufficient training data. If there is sufficient training data, pure interpolation methods yield satisfactory output function. Actually, **sufficient training data** and **small deviation** is the main reason that the most common CoG fuzzy inference is acceptable for such tasks. This is an important **theoretical result explaining the source of success**. The main practical conclusion from this study is that CoG max-min fuzzy inference can be substituted by a much simpler one, i.e., piecewise linear interpolation under these conditions.

Complete number of "IF-THEN" rules. The number of nodes involved in the input-output function is directly related to the number of "IF-THEN" rules. To decrease time of design and run-time of the fuzzy inference we must exclude "over complete" fuzzy sets. However, if we change to "incomplete" sets, then reliability of the decision will decrease. Furthermore, some parameter values for "incomplete" contexts have no fuzzy sets and no "IF-THEN" rules. We will consider these points below in discussing the reliability issue.

Reliability of the output of modified CoG inference. An output membership function value measures reliability of fuzzy rule inference. That is, small value of MF indicates an unreliable conclusion. Such conclusion is too risky to be used. This statement is true only if the output MF is well-founded, otherwise such MF's value can mislead a user to discard a valuable conclusion (discovery). In fuzzy inference, CoG defuzzification procedure computes the **output MF** with Center-of-Gravity of the area:

$$[\int u \cdot m^*(u) du] / \int m^*(u) du,$$

where $m^*(u) = [m(u/NL) \text{ OR } m(u/NS)] = \max[m(u/NL), m(u/NS)]$. This function is just one of the possible alternatives, which should be justified. On the other hand, if $m(u/NL)$ and $m(u/NS)$ are obtained using jointly the approach of ECCS and fuzzy statistics (Section 7.6.3 and 7.6.4), a well-founded formula for m^* exists -- it is simply the sum, i.e.,

$$m^*(u) = m(u/NL) + m(u/NS).$$

Note that $m(u/NL) + m(u/NS) \geq \max[m(u/NL), m(u/NS)]$. Thus, the modified CoG inference based on ECCS produces a greater value of the output MF than the usual CoG inference based on the same ECCS. Therefore, the modified CoG inference has better capabilities to produce reliable conclusions.

7.8. Fuzzy logic in finance

7.8.1. Review of applications of fuzzy logic in finance

This section covers a number of successful applications of fuzzy logic in finance. For example, Yamaichi Secures of Tokyo uses fuzzy logic to make decisions for an **investment fund**, and Nikko Secures of Yokohama uses a NeuroFuzzy system for a **bond rating program** [Houlder, 1994]. Several

authors noticed that many financial institutions consider their systems based on fuzzy logic a proprietary technology and do not publicize details, or even the fact of implementation and use [Lee, Smith, 1995; Von Altrock, 1997]. However, many uses of fuzzy logic in finance are published in the literature, including those listed below:

- Foreign-exchange trade support system in Japan with approximately 5000 fuzzy rules derived from a backpropagation neural network [Rao, Rao, 1993]. Fuzzy logic has been used in a foreign exchange trading system to predict the Yen-Dollar exchange rate [Yuize, 1991].
- Analysis of market psychology using a fuzzy expert system with data fuzzification and evaluation [Deny, 1993].
- Insider Trading Surveillance [Moulder, 1994].
- Fuzzy logic and variables in investing and trading [Caldwell, 1994].
- Neural network and fuzzy logic hybrid system in finance [Derry, 1994, WongF, 1994].
- Interpretation of neural network outputs using fuzzy logic, a fuzzy expert system is applied to the task of interpreting multiple outputs from a neural network designed to generate signals for trading the S&P 500 index [Caldwell, 1994b].
- A portfolio insurance strategy of Japanese stocks based on Nikkei Stock Index Futures using fuzzy logic. The system assists in deciding when to rebalance the replicating portfolio [Kay-Hwang and Woon-Seng Gan, 1996].
- Financial modeling and forecasting using a hybrid Neural-Fuzzy system. The model's performance is compared with a random walk model, an ARIMA model, and a variety of regression models [Pan et al, 1997].
- Fuzzy Scoring for Mortgage Applicants.
- Creditworthiness Assessment and Fraud Detection.
- Investor Classification [Von Altrock, 1997].
- Cash Supply Optimization [Von Altrock, 1997].
- Prediction of stock market direction using fuzzy logic [Von Altrock, 1997]
- Rating bonds [Loofbourrow, Loofbourrow, 1995].

Some other applications of fuzzy logic in finance are presented in [Jansen, Ruelas 1996; Rast, 1997, 1999; Golan, Edwards, 1993; Hiemstra, 1994; Severwright, 1997; Lee, Smith 1995; Kim et al., 1998; Sun, Wu, 1996; Wang, 1993]. Some of the mentioned works are described below.

Investor Classification. Many investment institutions classify customers and investments into three risk groups:

- a) conservative and security-oriented (risk shy),
- b) growth-oriented and dynamic (risk neutral), and
- c) chance-oriented and progressive (risk happy).

The fuzzy logic system was designed to evaluate how well a customer fits into these three groups. Each customer was represented by the set of 25 answers. Each question represents an attribute with five values from 1 to 5. The questions include personal background (age, marital state, number of children, job type, education type, etc.) the customer's expectation from an investment (capital protection, tax shelter, liquid assets, etc.) and others [Von Altrock, 1997].

Insider Trading Surveillance. Houlder [1994] describes a system developed for the London Stock Exchange. The goal of the system is to automatically detect insider dealing and market manipulation using a combination of fuzzy logic, neural nets, and genetic algorithms. The system tries to detect suspicious rings of individuals with several accounts in a vast amount of electronic camouflage.

Foreign Exchange Trading. Fuzzy logic has been used to predict the Yen-Dollar exchange rate [Yuize, 1991]. The system uses fuzzy logic rules to make inferences based on economic **news** events that may affect the currency market. This news is "translated" into the fuzzy logic system's input format by **domain experts**.

Cash Supply Optimization [Von Altrock, 1997]. Banks are interested in reaching two contradictory goals for each branch and ATM:

1. minimize **unused cash** and
2. minimize the rate of **out of cash** situations.

Cash surplus could be used for other profitable operations or/and decrease the cost of cash supply for branches and ATM. On the other hand, if the bank is able to minimize out of cash situations, it can better compete with other banks. The traditional expert solution is to set the **minimum amount of cash** for each branch and ATM. However, this minimum is not static, bank business conditions are changed dynamically for each individual branch and ATM due to:

- seasonal factors (week, month, year) and
- environmental factors (new shops, offices, banks nearby and so on)

Suppose the bank takes into account five such factors with only two values for each of them. This means that the bank should analyze regularly $1000 \cdot 2^5 = 32000$ alternatives to set up minimum amount of cash for its 1000 units.

“In a project of a European bank, fuzzy logic was used to recompute the minimum cash amount of each branch and ATM **daily**. The system was able to reduce the average cash supply in the branches and ATMs by 7.1% without increasing the rate of situations where the branch or ATM ran out of cash. For a bank with about 450 branches and 1270 ATMs, this results in an average total of \$3.8M less in cash supply” [Von Altrock, 1997].

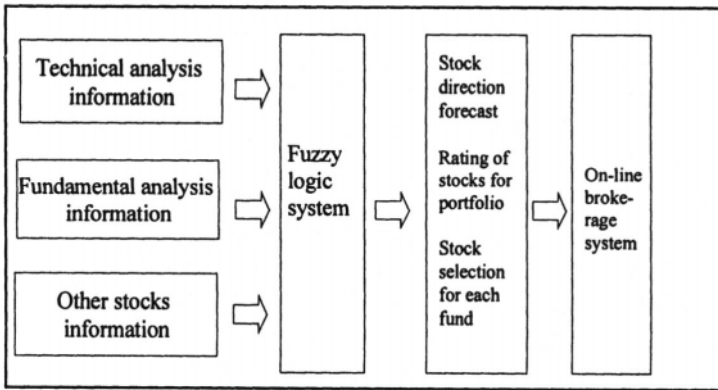


Figure 7.22. Fuzzy logic based system for trading

The system is based on three sources of information:

- 1) the past cash flow of the branches and ATMs,
- 2) the lowest cash amount suggested by the bank experts for each unit,
- 3) classification of ATMs and branches according to the properties of the neighborhoods.

Stock evaluation for trading linked to an online brokering system.

This system uses a similar approach for the **technical analysis** of stocks (see chapter 1). However, the buy/hold/sell strategy relies also on the **fundamental position** of the stock-issuing company and the situation of **similar companies**. Figure 7.22 shows the structure of the implemented fuzzy logic system for trading which evaluates about **1500 stocks in less than a minute** [Van Altroek, 1997].

7.8.2. Fuzzy logic and technical analysis

In Section 7.1, different aspects of a fuzzy logic system for trading were discussed. In particular rules related to inference of trading environment from interest rate and trade fee were presented. In this section, we present a general design of that system built on technical analysis information – various stock market trend indicators. Two decision-making characteristics were generated for the each stock [Von Altroek, 1997]:

- trading suitability rating and
- assessment of how well the considered stock fits with the scope of a particular fund.

The first indicator is based on the forecast of current stock direction (up/down) and the second indicator is based on a description of stock categories suitable for a given fund and a type of stock-issuing company. Therefore, there are two problems solved:

- developing evaluation indices and
 - generating a trading signal (buy/sell/hold) for each stock and fund.
- The solution uses a fuzzy logic **linguistic rule-based approach** for both problems, assuming that a financial expert can produce linguistic rules, e.g.,

IF there is **high probability** that the stock X is going up AND it **fits well** to the scope of the fund THEN buy stock X **almost certainly**.

IF there is **extremely high probability** that the stock X is going up AND it **rather does not fit** to the scope of the fund THEN **buy stock X certainly**.

IF there are **very good chances** that the stock is going up AND it **fits very well** to the scope of the fund THEN buy stock X **certainly**.

IF there are **very low chances** that the stock X is going up AND it **fits very well** to the scope of the fund THEN **consider sell or hold** stock X.

IF there are **some chances** that the stock X is going up AND it **fits very well** to the scope of the fund THEN it is **very risky** to buy stock X.

On a conceptual level this fuzzy logic system uses numerical indicators and evaluation functions (membership functions of fuzzy sets) to represent linguistic categories such as good chances. In addition, a fuzzy logic inference mechanism is used to get a buy/hold/sell signal with different degrees of confidence. A numerical indicator serves as a base scale for the linguistic concept and a membership function of a fuzzy set serves as a formal presentation of uncertain linguistic concepts within this scale.

In [Von Altrock, 1997] two indicators are chosen for the stock market trend: the Directional Movement Indicator (DMI) and Average Directional Movement (ADX) indicator, which is derivable from the first one. These indicators, together with the trends of the day, are used to represent the scales for informal linguistic concepts. Six concepts are considered:

- High chance that the stock will go up,
- Medium chance that the stock will go up,
- Low chance that the stock will go up,
- High chance that the stock will go down,
- Medium chance that the stock will go down,
- Low chance that the stock will go down.

Data. System uses indicators and indices based on stock price and volume:

1. Number of past days for which interest has to be computed.
2. Difference between today's closing highest stock price and closing price of the last trading day.
3. Difference between today's closing stock price and last trading day's.
4. Parameter (D4). Today's highest stock price minus the closing price of last trading day's. The value is zero if the closing price of the last trading day is lower than today's highest price.

5. Parameter (D5). Today's lowest stock price minus the closing price of the last trading day. The value is zero if the closing price of the last trade day is lower than today's lowest price.
6. True Range (TR) defined as the largest of: (a) the distance between today's high and today's low, (b) the distance between today's high and last trading day's close, or (c) the distance between today's low and last trading day's close.
7. 7-day average of parameter #4.
8. 7-day average of parameter #5.
9. 7-day average of parameter #6.
10. **Positive directional indicator** (+DI), computed as $D4/TR$ (see 4 and 6).
11. **Negative directional indicator** (-DI), computed as DS/TR (see 5 and 6).
12. Intermediate value to compute ADX, $DI_{diff} = \text{abs}(+DI7 - (-DI7))$.
13. Intermediate value to compute ADX, $DI_{sum} = (+DI7) + (-DI7)$.
14. Intermediate value to compute ADX, $DX7 = DI_{sum} / DI_{diff}7$.
15. **Average directional indicator**, ADX, computed as the seven-day average of DX7 (the membership functions for directional indicator is presented in figure 7.23).
16. Interest rate of the market rate account.
17. Trade fee rate.

Linguistic rules. Below we present some linguistic rules from the system implemented in [Von Altrock, 1997]. **Trend direction rules:**

IF the positive directional indicator is low and negative directional indicator (-DI) is high THEN the stock direction is negative.

IF the positive directional indicator is high and negative directional indicator is high THEN there is no stock trend. IF the interest rate is low and trade fee is low or significant

THEN <trading> environment is positive.

IF the interest rate is high and trade fee is significant

THEN <trading> environment is negative.

Trading (buy/hold/tell) rules

IF the average directional indicator is high and direction is positive THEN buy.

IF the average directional indicator is high and there is no trend THEN hold.

IF the direction is negative and the environment is negative

THEN sell.

The five variables (positive and negative directional indicators, average directional indicator, interest rate and trade fee) form input variables of this fuzzy logic system. The output variable, stock, has only the three values "buy"(1), "hold"(0) and "sell"(-1) after defuzzification.

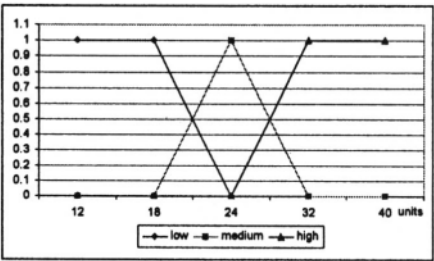


Figure 7.23. Directional indicator

Three rule blocks are used to produce the final solution (figure 7.24). The final rule block computes the value of a stock by analyzing trend strength, trend direction, and environment. The trend strength is presented by the average directional indicator. Trend direction is computed by another rule block from the positive and negative directional indicators. Trading environment is computed by a third rule block from the interest rate and trade fee variables. The trading environment assessment indicates the attractiveness of putting money in a market rate account.

Profit. The system uses the following mechanism to generate profit:

$$\text{investment} = \begin{cases} \text{stock investment, if signal} = 1 \text{ (Buy)} \\ \text{market rate account investment, if signal} = -1 \text{ (Sell)} \end{cases}$$

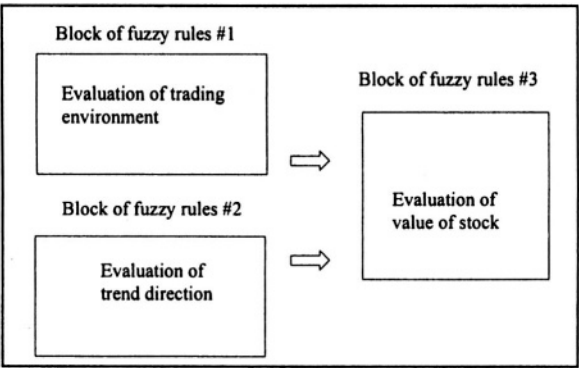


Figure 7.24. Blocks of fuzzy rules

Van Altrock [1997] noted that even this fuzzy logic system can not guarantee a successful investment, the profit rate of 18.92% using fuzzy logic was significantly higher than the fixed interest rate of 7% on the market rate account.

REFERENCES

- Abu-Mostafa (1990): Learning from hints in neural networks. *Journal of complexity* 6: 192-198.
- Abu-Mostafa Y, Moody J, Weigend AS, Eds (1996): *Neural Networks in Financial Engineering*, World Scientific, Singapore.
- Alekseev V (1988): Monotone Boolean functions. *Encyclopedia of Mathematics* v.6, Kluwer Academic Publishers, 306-307.
- Alexander C, Giblin I (1997): Multivariate embedding methods: forecasting high-frequency financial data. In: *Nonlinear Financial Forecasting (Proceedings of the First INFFC, Finance&Technology Pub.)*. Haymarket, USA, 51-74.
- Alexander JA, Mozer MC (1995): Template-based algorithms for connectionist rule extraction. In: *Advances in Neural Information Processing Systems (volume 7)*, Tesauro G, Touretzky D, Leen T, Eds. MIT Press, Cambridge, MA.
- Angluin D (1988): Queries and concept learning. *Machine Learning* 2:319-342.
- Andrews R, Diederich J, Tickle AB (1995): A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*. (<ftp://ftp.fit.qut.edu.au/pub/NRC/AUSIM/doc/QUTNRC-95-01-02.ps.Z>)
- Apte C, Hong SJ (1996): Predicting Equity Returns from Securities Data with Minimal Rule Generation. *Advances in knowledge discovery and data mining*, Eds. Fayyad U, Shapiro-Piatetsky G, Smyth P, Uthurusamy R. MIT, 541-560.
- Azoff E (1994): *Neural Network Time Series Forecasting of Financial Markets*, Wiley, NY.
- Bandy H: Thoughts on Desirable Features for a Neural Network-based Financial Trading System, *Journal of Computational Intelligence in Finance* 2 (3): 19-24.
- Bazaraa M., Sherali H., Shetty C. (1993) *Non-linear Programming: theory and Algorithms*, Wiley, NY.
- Bengio Y (1997): Using a financial training criterion rather than a prediction criterion. *International Journal of Neural Systems special issue on Noisy Time Series* 8(4), 433-444.
- Bergadano, F., & Giordana, A. (1988): A knowledge intensive approach to concept induction. *Proceedings of the Fifth International Conference on Machine Learning* (305-317). Ann Arbor, MI: Morgan Kaufmann.
- Bergadano, F., Giordana, A., & Ponsero, S. (1989): Deduction in top-down inductive learning. *Proceedings of the Sixth International Workshop on Machine Learning* (23-25). Ithaca, NY: Morgan Kaufmann.
- Berson A, Smith S (1997): *Data Warehousing, Data Mining and OLAP*. McCraw-Hill.
- Bovas A., Ledolter J. (1983), *Statistical methods for forecasting*, Wiley, NY.
- Box G, Jenkins G (1976): *Time series analysis: forecasting and control*. 2nd Ed. Holden-Day, San Francisco.
- Bratko, I. (1993): *Applications of machine learning: Towards knowledge synthesis. New Generation Computing 11*, (1993).
- Bratko, I. (1993): Innovative design as learning from examples. In *Proceedings of the International Conference on Design to Manufacture in Modern Industries*, Bled, Slovenia.

- Bratko, I., Muggleton, S., Varvsek, (1992): A. Learning qualitative models of dynamic systems. In *Inductive Logic Programming*, S. Muggleton, Ed. Academic Press, London.
- Bratko I, Muggleton S (1995): Applications of inductive logic programming. *Communications of ACM* 38 (11):65-70.
- Breiman L (1996): Bagging predictors. *Machine Learning* 24 (2):23-140.
- Breiman L, Friedman J, Olshen R, Stone C. (1984): Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA.
- Bridgeman PW (1927): The logic of modern physics. New York, Macmillan.
- Blum A, Rivest RL (1988): Training a 3-node neural network is NP-Complete (Extended abstract). In: Proceedings of the 1988 Workshop on Computational Learning Theory, San Francisco, CA. Morgan Kaufmann, 9—18.
- Bugarin A, Barro S, Regueiro CV (1993): Saving computation time through compaction of chained. In: Fifth IFSA World Congress. Seoul, Korea 2:1543-1549.
- Buntine WL (1990): A theory of learning classification rules. Ph.D. thesis, University of Technology, School of Computing Science, Sydney, Australia.
- Buntine W (1996): A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering* 8:195-210.
- Caldwell R (1994): Interpretation of Neural Network Outputs using Fuzzy Logic. *Journal of Computational Intelligence in Finance* 2 (3): 15-18.
- Caldwell R (1994): Book Review: State-of-the-Art Portfolio Selection, *Journal of Computational Intelligence in Finance* 2 (2):25-28.
- Caldwell R (1997): An Overview of the INFFC: from Organisation to Results. In: Nonlinear financial forecasting (Proc. of the first INFFC). Finance and Technology, 9-22.
- Carnap, R., Logical foundations of probability, Chicago, University of Chicago Press, 1962.
- Casdagli M, Eubank S, Eds (1992): Nonlinear Modeling and Forecasting. Addison-Wesley Publishing Company, Reading, MA.
- Castillo E, Gutierrez JM, Hadi A (1997): Expert Systems and Probabilistic Network Models. Springer Verlag, NY.
- Cheeseman P (1985): In defense of probability. In: Proceedings of International Joint Conference on Artificial Intelligence. Los Angeles, 1002-1009.
- Cheeseman P (1986): Probabilistic vs. fuzzy reasoning. In: Uncertainty in Artificial Intelligence L. Kanal & J. Lemmer Eds. North-Holland, Amsterdam, 85-102.
- Cheng, W., Wagner L. (1996): Forecasting the 30-year U.S. Treasury Bond with a system of neural networks, *Journal of Computational Intelligence in Finance*, v.4, n.1, 10-15.
- Chow C, Liu C (1968): Approximating discrete probability distributions with dependence Trees. *IEEE Transactions on Information Theory* 14:462-467.
- Clark P, Niblett T (1989): The CN2 induction algorithm. *Machine Learning* 3:261 - 283.
- Cohen, W. (1990): Abductive explanation-based learning: A solution to the multiple explanation-problem (ML-TR-29). New Brunswick, NJ: Rutgers University.
- Cooper GF, Herskovits E (1992): A Bayesian method for the induction of probabilistic networks from data. *Machine Learning* 9:309-347.
- Cramer D. (1998): Fundamental Statistics for Social Research, Step-by-step calculations and computer technique using SPSS for Windows, Routledge, London, NY

- Craven M, Shavlik J (1997): Understanding Time-Series Networks: A Case Study in Rule extraction. *International Journal of Neural Systems*, special issue on Noisy Time Series 8 (4):374-384.
- Craven MW, Shavlik JW (1996): Extracting tree-structured representations from trained networks. In: Touretzky DS, Mozer MC, Hasselmo ME (Eds.), *Advances in Neural Information Processing Systems* 8:24—30. Cambridge, MA. MIT Press.
- D'Ambrosio B (1993): Incremental probabilistic inference. In: Ninth Annual Conference on Uncertainty on AI, Heckerman D, Mamdani A (Eds.), Morgan Kaufmann, 301-308.
- Danyluk, A. (1989): Finding new rules for incomplete theories: Explicit biases for induction with contextual information. Proceedings of the Sixth International Workshop on Machine Learning (34-36). Ithaca, NY: Morgan Kaufmann.
- Dean T, Kanazawa K (1989): A model for reasoning about persistence and causation. *Computational Intelligence* 5 (3): 142-150.
- Dedekind (1897): Rueber Zerlegungen von Zahlen durch ihre grossten gemeinsamen Teiler. Festschrift Hoch. Braunschweig (in German), u.ges. Werke, II. 103-148.
- Derry JA (1993): Fuzzy Expert System and Market Psychology: A Primer (Part 1). *Journal of Computational Intelligence in Finance* 1 (1):10-13.
- Derry J (1994): Neurofuzzy Hybrid Systems. *Journal of Computational Intelligence in Finance* 2 (3):11-14.
- Dhar V, Stein R (1997): Intelligent decision support methods. Prentice Hall.
- Dhar V, Stein R (1997): Seven methods for transforming corporate data into business intelligence. Prentice Hall.
- Dietterich TG (1997): Machine Learning Research: Four Current Directions. *AI Magazine* 18 (4):97-136. Preprint <ftp://ftp.cs.orst.edu/pub/tgd/papers/aimag-survey.ps.gz>
- Domingos P, Pazzani M (1996): Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In: Proceedings of the Thirteenth International Conference on Machine Learning, (Ed) Saitta L, San Francisco, CA. Morgan Kaufmann, 105-112.
- Drake K, Kim Y (1997): Abductive Information Modelling Applied to Financial Time Series Forecasting. In: Nonlinear Financial Forecasting (Proceedings of the First INFFC). *Finance & Technology Publishing*, 95-108.
- Drobnishev Y.P. (1980): Data Analysis methods. In: Mathematical Problems of Data Analysis, Novosibirsk, p.6-14. (In Russian)
- Duda RÖ, Hart PE (1973): Pattern Classification and Scene Analysis. Wiley.
- DelRossi R (1999): C++ Programming Tools, *Software Development* 7 (8):49-53.
- Dhar V, Stein R (1997a): Intelligent decision support methods. Prentice Hall.
- Dhar V, Stein R (1997b): Seven methods for transforming corporate data into business intelligence, Prentice Hall.
- Drake K, Kim Y (1997): Abductive Information Modeling Applied to Financial Time Series
- Dubois D, Prade H (1990): Coping with uncertain knowledge - in defense of possibility and evidence theories. *Computers & Artificial Intelligence* 9:115-144.
- Dzeroski, S., DeHaspe, L., Ruck, B.M., and Walley, W.J. Classification of river water quality data using machine learning. In: Proceedings of the Fifth International Conference on the Development and Application of Computer Techniques to Environmental Studies (ENVIROSOFT'94), 1994.

- Dzeroski S (1996): Inductive Logic Programming and Knowledge Discovery in Databases. In: Advances in Knowledge Discovery and Data Mining, Eds. U. Fayad, G., Piatetsky-Shapiro, P. Smyth, R. Uthurusamy. AAAI Press, The MIT Press, 117-152.
- Edwards R, Magee J (1997): Technical Analysis of Stock Trends. Amacom, NY.
- Elman J (1991): Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning* 7 (2/3): 195-226.
- Farley A, Bornmann C (1997) Using a Genetic Algorithm to Determine a Prediction Function for Stock Prices. In: Nonlinear Financial Forecasting. (Proceedings of the First INFFC), Finance & Technology Publishing, 109-122.
- Farlow S (Ed, 1984): Self-organizing Methods in modeling: SMDH-Type Algorithms. Marcel Dekker, NY.
- Fayyad UM, Irani KB (1993): Multi-interval discretization of continuous-valued attributes for classification learning. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence. San Francisco. Morgan Kaufmann, 1022-1027.
- Fayyad U, Piatetsky-Shapiro G, Smyth P, Eds (1996): Advances in Knowledge Discovery and Data Mining, MIT Press.
- Fenstad, J.I. Representation of probabilities defined on first order languages. In: J.N.Crossley, ed., Sets, Models and Recursion Theory: Proceedings of the Summer School in Mathematical Logic and Tenth Logic Colloquium (1967) 156-172.
- Fisher RA (1936): The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7 (2):179-188.
- Fishburn PC (1970): Utility Theory for Decision Making. NY-London, J.Wiley&Sons.
- Fisher, D. (1987): Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2, 139-172.
- Flach, P., Giraud-Carrier C., and Lloyd J.W. (1998): Strongly Typed Inductive Concept Learning. In Proceedings of the Eighth International Conference on Inductive Logic Programming (ILP'98), 185-194.
- Flann, N., & Dietterich, T. (1989): A study of explanation-based methods for inductive learning. *Machine Learning*, 4, 187-226.
- Freedman R, Klein R, Lederman J (1995): Artificial Intelligence in the Capital Markets. Irwin, Chicago.
- French, K.R., Stock Returns and the Weekend Effect, *Journal of Financial Economics* 8, 55-70, 1980.
- Freund Y, Schapire RE (1995): A decision-theoretic generalization of on-line learning and an application to boosting. Tech. rep., AT&T Bell Laboratories, Murray Hill, NJ.
- Freund Y, Schapire RE (1996): Experiments with a new boosting algorithm. In: Proceedings of the Thirteenth International Conference on Machine Learning (Ed.) Saitta L. San Francisco, CA. Morgan Kaufmann 148-156.
- Friedman JH: Data Mining and Statistics: What's the Connection? Dept. of Statistics and Stanford Linear Accelerator Center, Stanford University, Stanford, CA, <http://stat.stanford.edu/~jhf/ftp/dm-stat.ps>
- Friedman N, Goldszmidt M (1996): Building classifiers using Bayesian networks. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence, Cambridge, MA, AAAI Press, 1277-1284.
- Fu LiMin (1999): Knowledge Discovery Based on Neural Networks, *Communications of ACM*, vol. 42, N11, 47-50.

- Gaines B (1984): Fundamentals of decision: Probabilistic, possibilistic, and other forms of uncertainty in decision analysis. *Studies in Management Sciences* 20:47-65.
- Ghahramani Z, Jordan MI (1996): Factorial hidden Markov models. In: Touretzky DS, Mozer MC, Hasselmo ME (Eds.). *Advances in Neural Information Processing Systems* 8:472-478 Cambridge, MA. MIT Press.
- Groth R (1998): Data Mining. Prentice Hall.
- Gershenfeld N, Weigend A (1994): The future of time series: Learning and Understanding. In: Time Series Prediction: forecasting the future and Understanding the Past. Addison-Wesley, Reading, Mass, 1-70.
- Giles C, Lee, Lawrence S, Tsoi AC (1997): Rule Inference for Financial Prediction using Recurrent Neural Networks. In: Proceedings of IEEE/IAFE Conference on Computational Intelligence for Financial Engineering (CIFER). IEEE, Piscataway, NJ, 253-259.
- Hall L, Szabo S, Kandel A (1986): On the derivation of memberships for fuzzy sets in expert systems. *Information Science* 40:39-52.
- Halpern JY: An analysis of first-order logic of probability. *Artificial Intelligence* 46: 311-350, 1990.
- Han J, Laks VS, Raymond T (1999): Constraint-based multidimensional data mining. *Computer* 8:46-50.
- Hansel G (1966): Sur le nombre des fonctions Booleenes monotones den variables. *C.R. Acad. Sci.*, Paris (in French), 262(20): 1088-1090.
- Hattoru K, Torri Y (1993): Effective algorithms for the nearest neighbour method in the clustering problem. *Pattern Recognition* 26 (5):741-746.
- Heath D, Kasif S, Salzberg S (1993): Learning Oblique Decision Trees. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 93), 1002-1007.
- Heckerman D, Geiger D, Chickering DM (1995): Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning* 20:197-243.
- Henriksson R., Merton R. On market timing and investment performance. II. Statistical procedures for evaluating forecasting skills, *Journal of Business*, 54, 513-533, 1981.
- Hienstra YA (1994): Stock Market Forecasting Support System Based on Fuzzy Logic, In: 27th Hawaii international conference. System sciences 3; Information systems: decision support and knowledge-based systems, IEEE Computer Society Press, 281-288.
- Hiller F, Lieberman (1995): Introduction to Operations Research. McGraw-Hill, NY.
- Hirsh, H. (1989): Combining empirical and analytical learning with version spaces.
- Hisdal E (1998): Logical Structures for representation of Knowledge and Uncertainty. Springer, NY.
- Holland J (1975): Adaptation in natural and artificial systems. University of Michigan Press, (reprinted in 1992 by MIT Press, Cambridge, MA).
- Holman E.W. (1978): Completely Nonmetric Multidimensional scaling. - *J. Math. Psychol.*, v.18, N1, p. 39-51.
- Houlder V (1994): Tackling Insider Dealing with Fuzzy Logic. *Financial Times*, September 29, 16.
- Hyafil L, Rivest RL (1976): Constructing optimal binary decision trees is NP-Complete. *Information Processing Letters* 5 (1): 15-17.
- Intern. Journal of Pattern Recognition and Artificial Intelligence* (1989): v.3, N. 1.

- Janssen J, Ruelas G (1996): Iteration of Human Knowledge, Fuzzy set and Bayesian Networks for the Stock Market Timing Forecast In: Proc. EXPERSYS '96: expert systems applications and artificial intelligence 1996 Oct. Paris, Gournay sur Marne; IITT, 139-144.
- JCIF (1994): Neural and Fuzzy Systems. *Journal of Computational Intelligence in Finance* 2 (3).
- Jensen FV, Lauritzen SL, Olesen KG (1990): Bayesian updating in recursive graphical models by local computations. *Computational Statistical Quarterly* 4:269-282.
- Jensen F (1996): An Introduction to Bayesian Networks. Springer, New York.
- Jordan MI, Jacobs RA (1994): Hierarchical mixtures of experts and the EM algorithm. *Neural Computation* 6 (2): 181-214.
- Jurik M. (1993): A Primer on Market Forecasting with Neural Networks. *Journal of Computational Intelligence in Finance* 1 (1):6-13.
- Kamara, A., Time Variation in the Turn-of-the-Year Seasonal, University of Washington and University of California, Los Angeles, Mimeo, 1998.
- Kamenskii V.S. (1977): Models and methods of nonmetric multidimensional scaling (review), *Automatics and telemechanics*, v.8, p.118-156 (In Russian)
- Kanazawa K, Koller D, Russell S (1995): Stochastic simulation algorithms for dynamic probabilistic networks. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. San Francisco, CA. Morgan Kaufmann, 346-351.
- Katz, B.(1989): Integrating learning in a neural network. Proceedings of the Sixth international Workshop on Machine Learning (69-71). Ithaca, NY: Morgan Kaufmann.
- Kay-Hwang, Woon-Seng Gan (1996): Back to Basics: Using a Fuzzy Logic Model for Portfolio Insurance of Japanese Stocks. *Journal of Computational Intelligence in Finance* 4(3):16-23.
- Keim, D.B., and R.F. Stambaugh, A Further Investigation of the Weekend Effect in Stock Returns, *Journal of Finance* 39, 819-835, 1984.
- Kendall M.G., Stuart A. (1977) *The advanced theory of statistics*, 4th ed., v.1. Charles Griffin & Co LTD, London.
- Kim M-J, Han I, Lee K (1998): The Integration of Machine and Human Knowledge by Fuzzy Logic for the Prediction of Stock Price Index. In: 5th Pacific Rim international conference. NY, Springer, 260-271. (LECTURE NOTES IN COMPUTER SCIENCE 1998, No 1531)
- Kohavi R, Kunz C (1997): Option decision trees with majority votes. In: Proceedings of the Fourteenth International Conference on Machine Learning. San Francisco, CA, Morgan Kaufmann.
- Kohonen T (1995): Self-Organizing Maps. Springer-Verlag, Berlin, Germany.
- Kamgar-Parsi B, Kanal LN (1985): An improved branch-and-bound algorithm for computing k-nearest neighbors. *Pattern Recognition Letters* 3:7-12.
- Kleitman D (1969): On Dedekind's problem: The number of monotone Boolean functions. *5-th Proceedings of the American Mathematics Society* 21:677-682.
- Kolmogorov AN (1956): Foundations of the theory of probability (translation edited by Nathan Morrison. 2nd English Ed.), New York, Chelsea Pub. Co.
- Korobkov V (1965): On monotone Boolean functions of algebraic logic. *Problemy Cybernetiki* 13:5-28, Nauka, Moscow, (in Russian):
- Kosko B (1997): Fuzzy Engineering. Prentice Hall, NJ.

- Kovalerchuk B (1973): Classification invariant to coding of objects. *Comp. Syst.* 55:90-97, Institute of Mathematics, Novosibirsk, (in Russian):
- Kovalerchuk B (1975): On cyclical scales. *Comp. Syst.* 61:51-59, Novosibirsk, Institute of Mathematics. (in Russian):
- Kovalerchuk, B. (1976), Coordinating methods for decision rules and training data in pattern recognition. Ph. D. Diss., Institute of Mathematics, USSR Academy of Sciences, Novosibirsk, 146 p. (in Russian):
- Kovalerchuk B (1977): Method for improving performance of decision rules performance. *Comp. syst.* 69:130-146, Novosibirsk, Institute of Mathematics, (in Russian):
- Kovalerchuk B (1990): Analysis of Gaines' logic of uncertainty. In: Proceedings of NAFIPS'90, Eds I.B.Turksen. v.2, Toronto, Canada, 293-295.
- Kovalerchuk B, Berezin S (1993): Matrix conjunction as generalized fuzzy and probabilistic conjunction. In: First European Congress on Fuzzy and Intelligent Technologies. ELITE-Foundation, Aachen, Germany, 1531-1536.
- Kovalerchuk B, (1994): Advantages of Exact Complete Context for Fuzzy Control. In: Joint Int. Conference on Information Science. Duke Univ., NC, 448-449
- Kovalerchuk B (1996a): Context spaces as necessary frames for correct approximate reasoning. *International Journal of General Systems* 25 (1):61-80.
- Kovalerchuk B (1996b): Second interpolation for fuzzy control. In: Fifth IEEE International Conference on Fuzzy Systems. New Orleans, 150-155.
- Kovalerchuk B, Galperin L, Kovalerchuk Ya (1977): Using pattern recognition methods for arbitration. *Voprosy kibernetiki, Cybernetics in Jurisprudence* 40:107-112, Moscow, Academy of Sciences, (in Russian):
- Kovalerchuk B, Dalabaev B (1994): T-norms as scales. In: First European Congress on Fuzzy and Intelligent Technologies, Aachen, Germany, 1993.1482-1487.
- Kovalerchuk B, Lavkov V (1984): Retrieval of the maximum upper zero for minimizing the number of attributes in regression analysis. *USSR Computational Mathematics and Mathematical Physics* 24 (4): 170-175.
- Kovalerchuk B, Shapiro D (1988): On the relation of probability theory and fuzzy-set theory foundations. *Computers and Artificial Intelligence* 7:385-396.
- Kovalerchuk B, Talianski V (1992): Comparison of empirical and computed values of fuzzy conjunction. *Fuzzy sets and Systems* 46:49-53.
- Kovalerchuk B, Triantaphyllou E, Ruiz J. (1996) Monotonicity and logical analysis of data: a mechanism for evaluation of mammographic and clinical data. In Kilcoyne RF, Lear JL, Rowberg AH (eds): Computer applications to assist radiology, Carlsbad, CA, Symposia Foundation, 1996, 191-196.
- Kovalerchuk B, Triantaphyllou E, Despande A, Vityaev E (1996): Interactive Learning of Monotone Boolean Function. *Information Sciences* 94 (1-4):87-118.
- Kovalerchuk B, Triantaphyllou E, Vityaev E (1995): Monotone Boolean functions learning techniques integrated with user interaction. In: Proceedings of the Workshop "Learning from examples vs. programming by demonstration" (12-th International Conference on Machine Learning), Tahoe City, CA, 41-48.
- Kovalerchuk B, Vityaev E (1998): Discovering Lawlike Regularities in Financial Time Series. *Journal of Computational Intelligence in Finance* 6 (3):12-26.
- Kovalerchuk B, Vityaev E (1999a): Comparison of relational and attribute-based methods for data mining in intelligent systems. In: 1999 IEEE International

- Symposium on Intelligent Control/Intelligent Systems and Semiotics. Cambridge, MA, IEEE.
- Kovalerchuk B, Vityaev E (1999b): Inductive Logic Programming for Discovering Financial Regularities, Chapter in: Data mining in Finance, Springer.
- Kovalerchuk B, Vityaev E, Ruiz JF. (1997): Design of consistent system for radiologists to support breast cancer diagnosis. *Joint Conf. of Information Sciences*, Duke University, NC, 2:118-121, 1997.
- Kovalerchuk B., Vityaev E., Ruiz J. (1999b) Consistent knowledge discovery in medical diagnosis, IEEE Engineering in Medicine and Biology.
- Kovalerchuk B, Yusupov H (1993): Fuzzy control as interpolation. In: Fifth IFSA World Congress. Seoul, S.Korea, 1151-1154.
- Kovalerchuk B, Yusupov H, Kovalerchuk N (1994b): Comparison of interpolations in fuzzy control. In: 2nd IFAC workshop on computer software structures integrating AI/KBS systems in process control. Preprints, IFAC, Lund University, Lund, Sweden, pp 76-80.
- Krantz DH, Luce RD, Suppes P, and Tversky A: Foundations of Measurement V.1-3, Acad. Press, NY, London. 1971, 1989, 1990.
- Kuo RJ, Lee LC, Lee CF (1996): Intelligent Stock Market Forecasting System Through Artificial Neural Networks And Fuzzy Delphi. In: Neural networks, World congress, San Diego; CA INNS Press, 886-892.
- Kupershtoh V.L., Mirkin B.C., Trofimov V.A. (1976): Least square method in analysis of qualitative features. In: Problems of discrete information analysis, Novosibirsk. (In Russian)
- Kuzmin V.B. Orlov A.I. (1977): On average values invariant relative to the permissible transformations of scales. In: Statistical expert analysis methods, Moscow, p.220-227. (In Russian)
- Lakonishok, J., and Levi M. Weekend Effects on Stock Returns, *Journal of Finance* 37, 883-889, 1982.
- Lakonishok, J. and S. Smidt, Are Seasonal Anomalies Real? A Ninety-Year Perspective, *Review of Financial Studies* 1 (4), 403-425, 1988.
- Langley P, Simon HA (1995): Applications of machine learning and rule induction. *Communications of the ACM* 38 (November):55-64.
- Lavrac N, Keravnou E, Zupan B Eds (1997): Intelligent Data Analysis in Medicine and Pharmacology, Kluwer Academic Publishers, Boston.
- Lbov, G., Katyukov, V., Manokhin, A., (1973): On pattern recognition algorithm in the space of attributes of different types, (Comp. Syst., #55:98-107). (In Russian)
- Lebowitz, M. (1986): Integrated learning: Controlling explanation. *Cognitive Science*, 10.
- LeBaron B (1994): Nonlinear Diagnostics and Simple Trading Rules for High-Frequency Foreign Exchange Rates. In: Time Series Prediction: forecasting the future and Understanding the Past (Eds. Gershenfeld N, Weigend A), Addison-Wesley, Reading, Mass.
- Lee MA, Smith MH (1995): Handling Uncertainty in Finance Applications Using Soft Computing. In: ISUMA-NAFIPS '95. College Park, MD, IEEE Computer Society Press, 384-389
- Loofbourrow J, Loofbourrow T (1995): What AI brings to trading and portfolio management. In: AI in the Capital markets, Eds. Freedman R, Klein R, Lederman J, Probus Publ, 3-28.
- Lawrence S, Tsoi A, Giles C, Lee C (1996): Symbolic representation and recurrent neural network grammatical inference. Technical Report, UMIACS-TR-96-27

- and CS-TR-3625. Institute for Advanced Computer Studies, University of Maryland, MD 20742.
- Maggini M, Giles CL, Home B (1997): Financial Time Series Forecasting Using K-Nearest Neighbors Classification. In: Nonlinear Financial Forecasting (Proceedings of the First INFFC). Ed Randall B Caldwell, *Finance & Technology Publishing*, 169-182.
- Mal'tsev A.I. (1970): Algebraic systems. Moscow, Nauka, 392p. (In Russian)
- Manohin A.N. (1976): Pattern recognition methods, using logical decision functions. (Comp. Syst. #67), Novosibirsk, p.42-53. (In Russian)
- Michalski, R. (1969): On the quasi-minimal solution of the general covering problem. In: Proc. of the First International Symposium on Information Processing, Bled, Yugoslavia, p. 125-128
- Michalski, R. (1980): Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2, 349-361.
- Michalski R, Mozetic I, Hong J, Lavrac N (1986): The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In: Proc. of Fifth National Conf. on AI. Philadelphia, Morgan-Kaufmann, 1041-1045.
- Michie D (1989): Problems of computer-aided concept formation. In: Quinlan JR (Ed.), *Applications of expert systems* (Vol. 2). Wokingham, UK, Addison-Wesley.
- Mirkin B.G. (1980): Analysis of qualitative features and structures. Moscow, Statistics, 316p. (In Russian)
- Mirkin B.G. (1976): Analysis of qualitative features. Moscow, Statistics, 166p. (In Russian)
- Mitchell T. (1997): *Machine Learning*, Prentice Hall.
- Mitchell T. (1999): Machine Learning and Data Mining. *Communications of ACM* 42(11):31-36.
- Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986): Explanation-based learning: A unifying view. *Machine Learning*, 1, 47-80.
- McMillan C, Mozer MC, Smolensky P (1992): Rule induction through integrated symbolic and sub-symbolic processing. In: Moody J, Hanson S, Lippmann R, (Eds), *Advances in Neural Information Processing Systems* (volume 4). Morgan Kaufmann, San Mateo, CA.
- Montgomery D, Johnson L, Gardner J (1990): *Forecasting & time series analysis*. 2nd Ed. McGraw-Hill, NY.
- Mooney, R., & Ourston, D. (1989): Induction over the unexplained: Integrated learning of concepts with both explainable and conventional aspects. *Proceedings of the Sixth International Workshop on Machine Learning* (5-7). Ithaca, NY: Morgan Kaufmann.
- Moser MC (1994): Neural net architecture for temporal sequence processing. In: *Time Series Prediction: Forecasting the Future and Understanding the Past*, A. Weigend and N. Gershenfeld (Eds). Reading, MA, Addison-Wesley.
- Moser MC (1995): Neural net architecture for temporal sequence processing. In: *Predicting the future and understanding the past*. Eds. Weigend A, Gershenfeld N, Redwood City, CA, Addison-Wesley.
- Mouzouris G, Mendel J (1996): Designing fuzzy logic systems for uncertain environments using a singular-value-QR decomposition method. In: Fifth IEEE international conference on fuzzy systems. New Orleans, September, 295-301.

- Muggleton, S., & Buntine, W. (1988): Machine invention of first-order predicates by inverting resolution. *Proceedings of the Fifth International Workshop on Machine Learning* (339-352). Ann Arbor, MI: Morgan Kaufmann.
- Muggleton, S., Bain, M., Hayes-Michie, J., & Michie, D. (1989): An experimental comparison of human and machine learning formalisms. *Proceedings of the Sixth International Workshop on Machine Learning* (115-118). Ithaca, NY: Morgan Kaufmann.
- Muggleton, S., Ed. (1992): *Inductive Logic Programming*. Academic Press, London.
- Muggleton, S., King, R.D. and Sternberg, M.J.E. (1992) Protein secondary structure prediction using logic. *Prot. Eng.* 5, 7), 647-657
- Muggleton S. Bayesian inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning* W. Cohen and H. Hirsh, Eds. (1994), 371-379.
- Muggleton S (1999): Scientific Knowledge Discovery Using Inductive Logic Programming, *Communications of ACM*, vol. 42, N11, 43-46.
- Munakata T (1999): Knowledge Discovery, *Communications of ACM*, vol. 42, N11, 27-29.
- Murphy PM, Pazzani MJ (1991): ID2-of-3: Constructive induction of M-of-N concepts for discriminators in decision trees. In: *Proceedings of the Eighth International Machine Learning Workshop*. Evanston, IL, Morgan Kaufmann, 183-187.
- Narens L. (1985), *Abstract Measurement Theory*, MIT Press, Cambridge.
- Nauck D, Klawonn F, Krause R (1997): *Foundation of neuro-fuzzy systems*. Wiley, NY.
- Nicholson C (1998): What Is Technical Analysis and Why Use It? http://www.ataa.com.au/a_cn_ta.htm
- Nonlinear Financial Forecasting (1997): *Proceedings of the First INFFC*, Edited by Randall B. Caldwell, *Finance & Technology Publishing*.
- Obradovic Z (Guest Editor, 1997): Hybrid Neural Networks for Financial Forecasting. *Journal of Computational Intelligence in Finance* 5 (1).
- Oliker S (1997): A Distributed Genetic Algorithm for Designing and Training Modular Neural Networks in Financial Prediction. In: *Nonlinear Financial Forecasting (Proceedings of the First INFFC)*. *Finance & Technology Publishing*, 183-190.
- Omlin CW, Giles CL (1996): Extraction of rules from discrete-time recurrent neural networks. *Neural Networks* 9 (1):41-52.
- Orlov A.I. (1979): Permissible averages in expert estimations and quality index aggregating. In: *Multidimensional statistical analysis in social-economic studies*. Moscow: Nauka. (In Russian)
- Orlov A.I. (1977): Stability in social-economic models. Moscow. (In Russian).
- Ourston, D., Mooney, R. (1990): Chaining the rules: A comprehensive approach to theory refinement. *Proceedings of the Eighth National Conference on Artificial Intelligence* (815-820). Boston, MA: Morgan Kaufmann.
- Pan Z, Liu X, Mejabi O (1997): A Neural-Fuzzy System for Financial Forecasting. *Journal of Computational Intelligence in Finance* 5 (1):7-15.
- Pankratz A (1983): *Forecasting with Univariate Box-Jenkins Models*. Wiley&Sons.
- Parsaye K (1997): (Prepositional) Rules are Much More than Decision Trees. *The Journal of Data Warehousing*, n.1.
- Passino K, Yurkovich S (1998): *Fuzzy Control*. Addison-Wesley, Menlo Park, CA.

- Pawlak Z, Wong SKM, Ziarko W (1988): Rough sets: Probability versus deterministic approach. *International Journal of Man-Machine Studies* 29:81-95.
- Pazzani, M. (1989): Explanation-based learning with weak domain theories. Proceedings of the Sixth International Workshop on Machine Learning (72-74). Ithaca, NY: Morgan Kaufmann.
- Pazzani, M. J. (1990): Creating a memory of causal relationships: An integration of empirical and explanation-based learning methods. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Pazzani, M., Brunk, C. (1990), Detecting and correcting errors in rule-based expert systems: An integration of empirical and explanation-based learning. Proceedings of the Workshop on Knowledge Acquisition for Knowledge-Based System. Banff, Canada.
- Pazzani, M., Kibler, D. (1992): The utility of prior knowledge in inductive learning. *Machine Learning*, 9, 54-97
- Pazzani, M., (1997), Comprehensible Knowledge Discovery: Gaining Insight from Data. First Federal Data Mining Conference and Exposition, 73-82. Washington, DC
- Pedrycz W, Valente de Oliveira J (1993): Optimization of fuzzy relational models. In: Fifth IFSA World Congress, v.2, 1187-1190.
- Pfaffenberger R, Patterson J (1977): Statistical Methods for Business and Economics. Irwin.
- Pfanzagl J. (1971): Theory of measurement (in cooperation with V.Baumann, H.Huber) 2nd ed. Physica-Verlag.
- Pfeiffer BM, Isermann R (1993): Criteria for Successful Application of Fuzzy Control. In: First European Congress on Fuzzy and Intelligent Technologies v.2, Aachen, Germany, 1403-1409.
- Proceedings of the Sixth International Workshop on Machine Learning (29-33). Ithaca, NY: Morgan Kaufmann.
- Pzelecki M. (1969): The logic of empirical theories. - London: Routledge Kogan Paul.
- Quinlan, J. R. (1986): Induction of decision trees. *Machine Learning*, 1, p.81-106
- Quinlan, J. R. (1989): Learning relations: Comparison of a symbolic and a connectionist approach (Technical Report). Sydney, Australia: University of Sydney.
- Quinlan, J. R. (1990): Learning logical definitions from relations. *Machine Learning*, 5, 239-266.
- Quinlan J (1993): C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA.
- Quinlan JR (1996): Bagging, boosting, and C4.5. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence. Cambridge, MA, AAAI/MIT Press, 725-730.
- Rabiner LR (1989): A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77 (2):257-286.
- Ramakrishnan N, Grama AY (1999): Data Mining: from serendipity to Science. *Computer* 8:34-37.
- Rao VB, Rao HV (1993): C++ Neural Networks and Fuzzy Logic. Management Information Source Press, NY.
- Rast M (1999): Forecasting with Fuzzy Neural Networks: A Case Study in Stock Market Crash Situations. In: Proc. North American Fuzzy Information Process-

- ing Society: Real world applications of fuzzy logic and soft computing, NY. NY, IEEE, 418-420.
- Rast M (1998): Predicting Time-Series with a Committee of Independent Experts Based on Fuzzy Rules. In: International conference: Decision technologies for computational finance, Neural networks in the capital markets. Dordrecht, London; Kluwer Academic Publisher, 433-438.
- Raymond C, Boverie S, LeQuellec J (1993): Practical realization of fuzzy controllers, comparison with conventional methods. In: First European Congress on Fuzzy and Intelligent Technologies. ELITE-Foundation, Aachen, Germany, 149-155.
- Resconi G (1993): Vectorial and tensorial representation of uncertainty by nodal logic. In: Fifth IFSA World Congress. Seoul, S. Korea, 92-95.
- Resconi G, Klir G, St. Clair, U. (1992): Hierarchical uncertainty metatheory based upon modal logic. *International Journal of General Systems* 21:23-50.
- Roberts F.S., Franke C.H. (1976) On the theory of Uniqueness in Measurement. - *J. Math. Psychol.*, v.14, n3, p.211-218.
- Roll, R., Was ist Das? (1983): The Turn-of-the-Year Effects and the Return Premia of Small Firms, *Journal of Portfolio Management*, Winter, 18-28.
- Rozeff, M.S. and Kinney, Jr., W.R. (1976): Capital Market Seasonality: The Case of Stock Returns, *Journal of Financial Economics* 3, 379-402.
- Russel S, Norvig P (1995): Artificial Intelligence. A Modern Approach, Prentice Hall.
- Saad E, Prokhorov D, Wunsch D (1998): Comparative Study of Stock Trend Prediction Using Time Delay. In: Recurrent and Probabilistic Neural Networks, *IEEE Trans. on Neural Networks* 9:1456-1470.
- Samokhvalov, K., (1973): On theory of empirical prediction, (Comp. Syst, #55), 3-35. (In Russian)
- Sarrett, W., Pazzani, M. (1989): One-sided algorithms for integrating empirical and explanation-based learning. Proceedings of the Sixth International Workshop on Machine Learning (26-28). Ithaca, NY: Morgan Kaufmann.
- Satarov G.A., Kamenskii V.S. (1977): General approach to the expert estimations by the nonmetrical multidimensional scaling methods. In: Statistical methods of the expert estimations analysis. Moscow, Nauka, p.251-266 (In Russian)
- Severwright J (1997): Organizational Benefits of Fuzzy Logic in Marketing and Finance. In: Fuzzy logic: applications and future directions. London, UNICOM Seminars Limited, 195-208.
- Scott, D., Suppes P., (1958), Foundation aspects of theories of measurement, *Journal of Symbolic Logic*, v.23, 113-128.
- Shavlik JW (1994): Combining symbolic and neural learning. *Machine Learning* 14:321-331.
- Shavlik, J., & Towell, G. (1989): Combining explanation-based learning and artificial neural networks. Proceedings of the Sixth International Workshop on Machine Learning, 90-93. Ithaca, NY: Morgan Kaufmann.
- Smirlock, M.I and Starks L. Day of the Week and Intraday Effects in Stock Returns, *Journal of Financial Economics* 17, 197-210, 1986.
- Smyth P, Heckerman D, Jordan MI (1997): Probabilistic independence networks for hidden Markov probability models. *Neural Computation* 9 (2):227-270.
- Spiegelhalter D, Dawid A, Lauritzen S, Cowell R (1993): Bayesian analysis in expert systems. *Statistical Science* 8:219-282.

- Spirtes P, Glymour C, Scheines R (1993): Causation, Prediction, and Search. Springer-Verlag, New York.
- Srivastava A, Weigend A (1997): Data Mining in Finance. *International Journal of Neural Systems, special issue on Noisy Time Series* 8 (4):367-372.
- Sullivan R, Timmermann A, White H (1998): Dangers of Data-Driven Inference: The Case of Calendar Effects in Stock Returns. University of California, San Diego, Dept. of Economics, <http://weber.ucsd.edu/Depts/Econ/Wpapers/Files/ucsd9816.pdf>
- Sun CT, Wu MD (1996): Fuzzy Multi-Staged Genetic Algorithm in Stock Investment Decision. In: Soft computing, International conference; 4th, IFSA, Fukuoka, Japan, World Scientific, 507-510.
- Suppes P., Zines J. (1963): Basic measurement theory. In: Luce, R., Bush, R., and Galanter (Eds). Handbook of mathematical psychology, v. 1, NY, Wiley, 1-76.
- Swanson N., White H. (1995): A model-selection Approach to assessing the information in the term structure using linear models and artificial neural networks, *Journal of Business & Economic Statistics*, July, Vol. 13, n.3, 265-275.
- Tan AH (1994): Rule learning and extraction with self-organizing neural networks. In: Proceedings of the 1993 Connectionist Models Summer School. Hillsdale, NJ. Lawrence Erlbaum Associates, 192-199.
- Terehina A.Y. (1973): Multimentional scaling methods and data visualization (Review). Automatic and Telemechanics, v.7, p.80-94. (In Russian)
- Thole U, Zimmerman HJ, Zysno P (1979): On the suitability of minimum and product operators for the intersection of fuzzy sets. *Fuzzy Sets and Systems* 4:37-51.
- Tilli TA (1993): Practical tools for simulation and optimization fuzzy systems with various operators and defuzzification method. In: 1st European Congress on Fuzzy and Intelligent Technologies, Aachen, Germany, v.1, 256-262.
- Trippi R, Turban E (1996): Neural Networks in Finance and Investing. Irwin, Chicago.
- Tukey JW (1962): The future of data analysis. *Ann. Statist* 33:1-67.
- Tyrin Y.N., Litvak B.G., Orlov A.I., Satarov G.A., Shmerling D.S. (1981): Analysis of nonnumeric information. - Moscow, 80p (In Russian)
- Tyrin Y.N., Vasilevich A.P., Andrukevich P.F. (1977): Statistical methods of ranking. In: Statistical methods of expert estimations analysis, Moscow, 30-58.
- Twain Mark (Samuel Clemens) (1894): *The Tragedy of Pudd'nhead Wilson*, reprint 1996, (New York, Oxford University Press), ch. 13.
- Ullman, J., (1988): Principles of database and knowledge base systems, v. 1, Rockville, Mass.: computer Science Press.
- Van Eyden R (1996): The Application of Neural Networks in the Forecasting of Share Prices. *Finance and technology Publishing*.
- Verma T, Pearl J (1990): Equivalence and synthesis of causal models. In: Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence. San Francisco, CA, Morgan Kaufmann, 220-227
- Vityaev E. (1976): Method for forecasting and discovering regularities, (Computing systems, #67), Institute of Mathematics, Novosibirsk, 54-68 (in Russian)..
- Vityaev E. (1983): Data Analysis in the languages of empirical systems. Ph.D. Diss, Institute of Mathematics SD RAS, Novosibirsk, p. 192 (In Russian)
- Vityaev E. (1992) Semantic approach to knowledge base development: Semantic probabilistic inference. (Comp. Syst. #146): 19-49, Novosibirsk, (in Russian).
- Vityaev E., Moskvitin A. (1993): Introduction to discovery theory: Discovery software system. (Comp. Syst., #148): 117-163, Novosibirsk (in Russian).

- Vityaev E., Logvinenko A. (1995): Method for testing a systems of axioms, Computational Systems, Theory of computation and languages of specification, (Comp. Syst., #152), Novosibirsk, p. 119-139 (in Russian).
- Von Altrock C. (1997): Fuzzy Logic and NeuroFuzzy Applications in Business and Finance. Prentice Hall PTR, NJ.
- Wai Man Leung, Yiu Ming Cheung, Lei Xu (1997): Abstractions of Mixtures of Experts Models to Nonlinear Financial Forecasting. In: Nonlinear Financial Forecasting (Proceedings of the First INFFC). *Finance & Technology Publishing*, 153-168.
- Wang, K., Li, Y., and Erickson, J. (1997): A New Look at the Monday Effect, *Journal of Finance* 52, 2171-2187.
- Weigend AS, Zimmermann HG, Neuneier R (1996): Clearning. In: Refenes P, Abu-Mostafa Y, Moody J, Weigend AS, Eds. *Neural Networks in Financial Engineering*. World Scientific, Singapore.
- Weigend A, Shi S. (1997): Taking time seriously: hidden Markov Experts applied to financial Engineering. In: Proc. of the IEEE/IAFE Conf. on Comp. Intelligence for Financial Engineering, NY, 1997. Piscataway, NJ, IEEE Service Center.
- Weigend A, Shi S. (1998): Predicting daily probability distribution of SP500 returns. NYU, Information Systems Dept., IS-98-23.
<http://www.stern.nyu.edu/~aweigend/Research/Papers/HiddenMarkov>
- Weiss S, Indurkha N (1993): Rule-based regression. In: Proc. of the International Joint Conference on AI. San Francisco, Morgan-Kaufmann.
- Widmer, G. (1990): Incremental knowledge-intensive learning: A case study based on an extension to Bergadano & Giordana's integrated learning strategy (Technical Report). Austrian Research Institute for Artificial Intelligence.
- Wong F (1994): Neurofuzzy Computing Technology. *Journal of Computational Intelligence in Finance*, Special Topic: Neural and Fuzzy Systems, 2(3):8-11.
- Worbois P (1975): Beyond regression: new tools for prediction and analysis in the behavioral sciences (Ph.D. Diss.). Harvard University.
- Wright G, Ayton O, Eds (1994): Subjective probability. Chichester, NY, Wiley.
- Yuize H et al. (1991): Decision Support System for Foreign Exchange Trading. In: International Fuzzy Engineering Symposium, p.53-64.
- Zadeh L (1977): The concept of linguistic variable and its application to approximate reasoning -1. *Information Sciences*, 8:199-249.
- Zadeh L (1988): On the treatment of uncertainty in AI. *The Knowledge Engineering Review* 3:81-85.
- Zagoruiko N (1972): Pattern recognition methods and their applications. Moscow, (in Russian).
- Zagoruiko N.G., Samokhvalov K.F., Sviridenko D.I. (1978): Logic of empirical research, Novosibirsk. (In Russian)
- Zagoruiko N.G., Elkina V.N. Eds. (1976): Machine Methods for Discovering Regularities. Proceedings of MOZ'76, Novosibirsk. (In Russian)
- Zhuravlev Yu, Nikiforov V (1971): Recognition algorithms based on computing estimates. *Kibernetika* 3, Kiev, (in Russian).
- Zighed A, Auray J, Duru G (1992): SIPINA: Méthode et logiciel. La cassagne.
- Zighed, DA (1996): SIPINA-W, <http://eric.univ-lyon2.fr/~ricco/sipina.html>, Université Lumière, Lyon.

Subject index

- Accuracy, 32, 83
- Activation function, 36, 37
- Activation value, 37
- Adaptive linear forecast, 219
- Analytical learning, 18
- Analytical learning system, 117
- AR - Autoregression, 22
- ARIMA, 21, 22, 28, 30, 29, 219, 225, 226
- ARIMA - AutoRegressive Integrated Moving Average, 22
- ARIMA model-building procedure
 - steps
 - diagnosis, 25, 27
 - estimation, 25, 27
 - identification, 25
- ARIMA seasonal, 25
- ARIMA XII
 - algorithm, 27
 - seasonal model, 28
- Arity, 137
- Atom, 137
- Atomic formulas, 155
- Attribute, 118, 144
- Attribute-based
 - examples, 130, 131
 - logical methods, 122
- Attribute-based methods, 128
- Attributes, 15
- Attribute-value, 118
 - learning system, 120
 - object presentation, 118
- Attribute-value language, 115, 118,, 123, 129, 130
- Attribute-value pairs, 146
- Autocorrelation function (ACF), 25
- Autoregression, 2, 22
 - first-order, 22
 - model, 22
 - order, 22
 - second order, 22
- Autoregression methods, 118
- Autoregressive, 29
- Autoregressive model, 23
- AVL, 115
- Background knowledge, 6, 115, 116, 117, 120, 124, 125, 131, 140, 141, 147, 150, 151, 165, 173, 175, 177, 243
 - partial, 143
- Bayes' rule, 106
- Best predictions, 157
- Boolean function, 174
 - lower unit, 66
 - maximal upper zero, 66
 - minimal lower unit, 66
 - upper zero, 66
- Boosting, 48
- Bootstrap aggregation, 48
- Border, 88
- Borderline
 - performance, 89
- Branch-and-Bound, 159
- BST regularity, 159
- BST rules, 156, 157, 159, 162
- BSUM - bounded sum, 244
- C4.5, 84
 - bagged, 85
 - decision trees, 86
- Case-based method, 34
- Case-based reasoning, 32, 35, 168
- Causal modeling, 102
- Chain, 69
- Chaotic deterministic systems, 10
- Classification, 124, 125
- Clause
 - head, 136
- Comparison of FOIL and MMDR, 166
- Complete Round Robin, 49

- Complexity, 32**
- Comprehensibility, 83, 84**
- Comprehensible, 101**
- Confidence, 72**
- Consistency, 72**
- Context**
 - complete, 261
 - dependence, 240
 - incomplete, 261
 - independence, 240
 - over complete, 261
- Context dependence, 240**
- Context independence, 240**
- Context space, 234, 238, 249, 252, 261, 262, 265**
 - exact complete, 239
- Credit card decisions, 88**
- Cross-validated committees, 48**
- Cyclical scales, 130**
- Data Base Management systems, 12**
- Data clustering, 107**
- Data marts, 13**
- Data Mining, 1, 2, 12, 13, 22, 58, 166, 167, 168, 232, 234**
 - conceptual challenges, 19
 - definitions, 16
 - dimensions, 5
 - finance tasks, 3
 - financial, 2, 21
 - hybrid approach, 44
 - learning paradigms, 17
 - learning principles, 14
 - learning process, 19
 - methodologies, 4
 - methods, 2
 - numeric methods, 186
 - packages, 16
 - relational, 2
 - systems, 169
 - technological challenges, 20
 - tool, 169
- Data Mining methodology, 20**
- Data Mining methods, 3, 4, 9, 32, 49, 231**
 - assumptions, 5
 - differences, 29
 - invariance, 176
 - reliability, 47
 - specifics, 30
- Data Mining models, 15**
- Data Mining products, 16**
 - components, 16
 - features, 16
- Data type, 128, 129, 133, 151, 158, 163, 164, 167, 168, 173, 177**
 - absolute, 174
 - cyclical, 130, 169
 - discovering, 177
 - equivalence, 171, 174
 - implicit, 129
 - interval ordering, 172
 - nominal, 174
 - partial ordering, 171
 - permissible transformations, 175
 - relational, 167
 - semi-ordering, 172
 - strong ordering, 172
 - testing, 177
 - tolerance, 171
 - tree-type, 173
 - weak ordering, 171
- Data types, 4, 20, 33, 128, 130, 133, 159, 166, 167, 168, 169, 170, 175, 176, 181**
 - mixed, 179
 - numerical, 174
- Data warehouses, 13**
- Database, 1, 2, 13, 20**
- Data-based methods, 231**
- Databases, 166**
- DBMS - Data Base Management systems, 12**
- Decision trees, 219**
- Debugging, 266, 267, 268**
 - algorithm, 267

- context-free, 258
- contextual, 258
- heuristic, 266, 267
- rules, 267
- Decision making**
 - expert-based, 235
- Decision tree, 2, 72, 78, 79, 80, 87, 88, 122, 167**
 - C4.5, 84
 - close world assumption, 75
 - comprehensibility, 83
 - ensemble, 85
 - evaluating, 83
 - extraction from a trained neural network Trepan algorithm, 96
 - methods comparison, 86
 - predictive accuracy, 83
 - Trepan, 88
- Decision trees, 115, 151**
 - forecasting performance, 227
- Decision trees ensembles, 84**
- Decision-making methods**
 - data-based, 231
 - expert-based, 231
 - model-based, 231
- Defuzzification, 245, 247, 248, 266, 267**
- Dempster-Shafer theory, 235**
- Deterministic finite state automata, 43**
- DFA - Deterministic Finite state Automata, 43**
- Differencing, 22, 23, 29**
 - degree, 22, 23
 - first difference, 24
 - second difference, 24
 - third difference, 24
- Dimensions, 6**
- Discovery software, 154, 155**
- Disjunctive normal form, 75**
 - discovering, 76
- Disturbance, 22, 23**
- DM - Data Mining, 1**
- DNF**
 - complexity, 94
 - learning sets of rules, 94
 - minimal rules, 94
- DNF - disjunctive normal form, 75, 118**
- Domain**
 - knowledge, 138
 - theory, 138
- Domain knowledge, 116**
 - incomplete, 128
 - incorrect, 128
- Domain theory, 116, 147**
- DR - Deductive Reasoning, 7**
- DT - Decision Trees, 7**
- Dynamic probabilistic network, 111**
- Dynamic systems, 3, 9**
- ECCS - exact complete context space, 239**
- Embeddability, 35**
 - statistical methods, 29
- Empirical axiomatic theories, 181**
- Empirical axiomatic theory, 169, 179, 180**
- Empirical axioms, 187**
- Empirical contents of data, 170**
- Empirical dependence, 180**
- Empirical dependencies, 187**
- Empirical interpretation, 169, 180**
- Empirical real-world system, 175**
- Empirical system, 180**
- Equivalence relation, 171**
- Exact complete context space, 259, 260, 263, 265, 267, 268, 270, 273**
- Example-based learning, 40**
- Existential variables, 135**
- Expectation-Maximization (EM), 110**
- Expert mining, 5, 21, 59, 232**
- Explainability, 32**
- Explanation-based learning system, 117**
- First-order logic methods, 226**
- Financial forecasting, 3**

- First Order Combined Learner (FOCL), 147**
- First order language, 186**
- First order logic, 135, 147, 151, 154**
- First order logic languages, 123**
- First-order**
 - monadic rules, 122
- First-order learning tasks, 132**
- First-order logic, 116, 144, 153**
 - rules, 122
- First-order logic methods, 229**
- First-order logic representation, 153**
- First-order logic rules, 151**
- First-order methods, 153**
- First-order predicate, 147**
- First-order representations, 122**
- Fisher test, 160**
- FL - Fuzzy Logic, 7, 232**
- Flexibility, 7, 29, 32, 35**
- FOCL, 147, 150, 151**
- FOIL, 147, 151, 165, 219**
 - forecasting performance, 227
- Forecast**
 - interval, 125, 128
 - numerical, 125
- Forecast performer, 17, 18**
 - frameworks, 17
- Forecasting performance, 227**
- Forecasting tasks**
 - classification, 126
 - interval, 126
 - numerical, 126
- Fundamental Analysis, 2, 11**
- Fuzzification, 242, 243**
- Fuzzy inference, 262, 267, 276, 277**
 - MIN-MAX, 249
- Fuzzy logic, 231, 232, 234, 235, 237, 238, 239, 240, 241, 243, 244, 246, 247, 263, 281**
 - approach, 234, 282
 - assumption, 244
 - max-min, 254, 259
 - trading, 281
- Fuzzy logic in finance, 241, 278, 279**
- Fuzzy operators, 245**
 - compensatory operators, 245
 - GAMMA, 245
 - MIN-AVG, 245
- Fuzzy output, 247**
- Fuzzy rule inference, 244, 245**
 - aggregation, 244
 - composition, 244
- Fuzzy rules, 243, 247**
 - debugging, 249
 - definition, 245
- Fuzzy set theory, 239**
- Fuzzy statistics, 262**
- Fuzzy-inference decisions, 239**
- Fuzzy-probabilistic approach, 255**
- General probabilistic inference**
 - approach, 114
- Generalization, 151, 154, 155**
- Generative stochastic modeling, 102**
- Genetic algorithms, 2, 17, 45, 46**
 - advantages, 46
 - genotype, 47
 - population, 47
 - transitional operators, 47
- GG - Gross Gain/loss, 222**
- Gross gain/loss, 222**
- Ground fact, 144**
- Hamming distance, 34**
- Hansel chains, 64, 66, 70**
 - construction, 69
- Hansel's lemma, 61, 67**
- Hidden Markov Model (HMM), 109**
- Homomorphism, 175**
- Horn clause, 136**
- Human logic, 235**
- Human-readable, 116, 117, 166**
- Human-readable rules, 154**
- Hybrid approach, 232**
- Hybrid methods, 2**
- Hybrid systems, 232**

- Hypotheses**, 125, 151, 153
 - probabilistic, 126
- Hypotheses space**, 87, 167
- Hypothesis**
 - language, 87
- Hypothesis language**, 133
- Hypothesis space**, 117, 140, 147, 159
- IBL - Instance-Based Learning**, 7
- ILP**, 115, 116, 117, 123, 127, 144
 - Application, 125
 - software, 146
 - typed language, 132
- ILP - Inductive Logic Programming**, 2, 7
- Individual attribute distribution**, 83
- Inductive learning system**, 117
- Inductive Logic Programming**, 2, 115, 116, 124
- Information gain metric**, 148
- Inheritance**, 187
- Initial rule**, 152
- Input functions**, 36
- Instance-based learning**, 21, 32
 - AVO method, 34
 - methods, 35
- Instance-based learning**, 17
- Inter-argument constraints**, 140
- Inter-argument constraints approach**, 140
- Interpolation methods**, 231
- Interpretability**, 179
- Interpretable**, 128, 154, 170, 173, 176, 177
- Interval forecast tasks**, 125
- Invariance**, 176
- k nearest neighbor**, 2, 35, 36, 87
- Knowledge discovery**
 - fuzzy logic, 252, 266
 - fuzzy logic based, 233
- Knowledge domain**, 12
- Knowledge extraction**, 59
- Knowledge representation**, 17, 18, 19, 20, 128
- Knowledge-base**, 260
 - stochastic approach, 103
 - stochastic models, 112
- k-tuple**, 137
- Language**
 - attribute-based with types, 133
 - decision tree, 92
 - empirical axiomatic theories, 169
 - first-order logic, 169
 - first-order logic with types, 133
 - m-of-n rules, 101
 - propositional logic, 72
 - relational database, 146
- Languages**
 - attribute-based, 133
 - strongly typed, 133
- Law-like rule**, 151, 154, 155
- Learning**
 - mechanism, 17, 18
 - paradigm, 18
 - steps, 19
- Learning algorithm**
 - unstable, 87
- Learning methods**, 102, 167
 - Data-based, 102
 - knowledge-based, 102
 - stable, 87
- Linear discriminant method**, 87
- Linear regression**, 87
- Linguistic rules**, 267, 269, 277, 283
- Linguistic variable**, 242
- Literals**, 136
 - positive, 136
- Logic of empirical theories**, 180
- Logic Programming**, 116
- Logical expressions**, 35
- Logical probabilistic rules**, 151
- Logical relations**, 128
- Low compactness**, 35
- MA - moving average**, 22

Machine Learning, 19, 153

- paradigms, 17
- relational systems, 146

Machine Method for Discovering

- Regularities (MMDR), 116, 151**

Majority-voting scheme, 93**Market Theory, 10****Markov chain, 43****Markov Chain Monte Carlo method (MCMC), 86****Markov Chains, 2****Markov probabilistic network, 109****Markov process, 108****Markovian law, 108****Mathematical statistics, 236****Matrices**

- attribute-based, 169
- closeness, 169
- multiple comparisons, 169
- ordered data, 169, 181
- pair comparisons, 169, 181
- proximity, 181

Matrix

- attribute-based, 181
- binary relation, 183
- of closeness, 184
- of orderings, 183

MAX-MIN inference, 244**Meaningful hypotheses, 128****Measurement procedure, 180****Measurement scale, 167****Measurement theory, 128, 152, 167, 168, 176, 182****Measurer, 181****Membership functions, 238, 239, 240, 241, 247, 248, 262, 264, 266, 268****MF - membership function, 238****Minimal DNF, 75****MMDR, 76, 116, 125, 147, 151, 152, 153, 154, 155, 162, 163, 165, 167, 170, 181, 219, 225**

- forecasting performance, 227

MMRD, 151, 158**Modal logic, 263, 264****Model, 15****Model-based approach, 231****m-of-n**

- expression, 82, 83
- rule, 83
- test, 100
- tree, 82

Monadic, 167**Monadic function, 120****Monadic predicates, 120****Monitoring excessive claims, 88****Monotone Boolean functions, 51, 59, 66, 67**

- algorithm RESTORE, 67
- oracle, 67
- restoration, 21

Monotonicity, 49, 50, 59, 61, 168, 246, 267, 268

- hypothesis, 50

Moving average, 22, 29

- first-order process, 23
- order, 22
- process, 23

Multivariate and pair comparisons, 182**Neural network, 36, 38, 87, 97**

- activation level, 37
- clearing method, 98
- example, 38
- extracting rules, 83, 95
- hidden layer, 37
- input layer, 37
- output layer, 37
- structure, 40
- symbolic representation, 95
- Trapan, 83, 95

Neural networks, 2, 7, 17, 21, 30, 36, 40, 44, 71, 115, 118, 219, 226, 232

- accuracy, 40

- backpropagation, 39, 45, 54, 57
- backpropagation in finance, 41
- dynamically modifying approach, 40
- feedforward modular, 39, 45
- forecasting performance, 227
- gated, 45
- generalizability, 40
- learning method steps, 38
- performance, 40
- properties, 42
- recurrent, 39, 42
- shortages, 7
- structure, 232
- trading models, 3
- Neuro-fuzzy**
 - approach, 232
 - hybrid systems, 232
- NN - Neural Networks, 7**
- Nominal, 179**
- Normalization, 33**
- Numeric representation, 172**
- Numerical data, 35, 127**
- Numerical system, 175**
- OLAM - Online Analytical Mining, 14**
- Online Analytic Processing, 12, 13**
- Online Analytical Mining, 14**
- On-Line Transaction Processing, 12**
- Operation**
 - context-free, 258
- Operationalization, 141**
- Overdue mortgages, 88**
- PACF - Partial autocorrelation**
 - function, 25
- Paradigm, 17**
 - analytic learning, 17
 - case-based reasoning, 17
 - genetic algorithms, 17
 - hybrid, 17
 - instance-based learning, 17
 - Neural Networks, 17
 - rule induction, 17
- Partial autocorrelation function**
 - estimated, 26
- Performance criterion, 81**
- Permissible transformations, 175**
- Piecewise linear interpolation, 277**
- PILP - Probabilistic ILP, 7**
- Predicate, 129, 132, 135, 137, 174**
 - arguments, 144
 - defined extensionally, 135
 - defined intentionally, 135, 136
 - extensional, 141, 144
 - extensionally defined, 138
 - initial extensional, 143
 - initial intensional, 143
 - intensional, 141
 - intensionally defined, 138
 - localized, 134
 - monadic, 137
 - n-ary, 144
 - operational, 138
 - unary, 137
- Predicate development, 128**
- Predicate discovery, 168**
- Predicates, 128, 170**
 - empirical, 180
 - extensional, 146
 - theoretical, 180
- Preprocessing, 41**
- Preprocessing mechanism, 168**
- Probabilistic first-order rules, 153**
- Probabilistic Inference, 2**
- Probabilistic laws, 114, 162**
- Probabilistic network, 103, 104**
- Probability, 235, 236, 237, 239, 240, 272**
- Probability space, 237, 238**
- Probability subjective, 237**
- Problem ID profile, 6**
- Programming language Escher, 130**
- Projection function, 131**
- Projection functions, 132, 145**
- Prolog, 130, 146**
- Propositional logic language, 120**

Propositional representation, 121

Propositional rule, 79

Propositional tasks, 132

Protocol of observations, 180, 187

Pure interpolation methods, 277

Quantifiers, 138

RDM, 116, 117, 173

RDM - Relational Data Mining, 2, 165

Readability, 72

Recurrent Neural Networks

rule extraction, 43

Refutability, 154

Refutable, 154, 155

Regression without models, 21

Regularities, 157, 159, 162, 166, 167, 168, 169

Regularity, 157, 158, 162, 163, 168

Regularity type, 158

Relation, 120, 123, 144, 182

interval-ordering, 172

n-ary, 144

partial ordering, 171

quasi-ordering, 171

semi-ordering, 172

strong ordering, 172

tree-type, 173

weak ordering, 171

Relational concept definitions, 146

Relational data mining 2, 5, 10, 20, 42, 116, 117, 125, 126, 128, 130, 166, 168, 179, 225, 230

hybrid probabilistic, 126, 127

methods, 103, 147

Relational data types, 170

Relational database, 117, 144

Relational method, 80

Relational structure, 174

Relations, 129, 173

Representative measurement theory, 2, 128, 151, 153, 166, 180

Response speed, 32

R-MINI, 76

RMT, 167, 168, 169

RMT - Representative Measurement Theory, 2

RNN - Recurrent neural networks, 39

Round Robin complete method, 47

Relational methods, 219

Rule, 15, 136

discovering, 76

disjunctive normal form, 93

error-free, 91

extensional, 141

extensional form, 141

first-order, 72

first-order probabilistic, 81

If-Then, 243

incorrect, 151

intensional form, 141

MFR, 82

partial, 151

propositional, 72, 78

propositional probabilistic, 104

relational, 79

RG, 76

RL, 77

tree form, 75

understandable, 2

Rule induction, 18

Rule matrix, 246

Rule type, 163, 165

Rule-based methodology, 114

Rule-extracting methods

classification, 95

Rules

extensional, 143

first-order logic, 151

intensional, 141, 143

law-like, 155

law-like logical probabilistic, 154

propositional disjunctive normal form, 81

understandable, 72

SAS, 22

Scalability, 7, 32, 35

Scale

- nominal, 171
- of algebraic differences, 186
- of equal finite intervals, 186
- positive differences, 185

Scientific laws, 153, 154, 155

Second Interpolation, 234, 252, 267, 277

Selector function, 133

Self-organizing map, 43

Semantic approach, 265

Semantic probabilistic inference, 157, 159

Semantic probabilistic inferences, 157

Semantic procedures, 263

- labeling procedure, 263
- reference label set, 263

Shannon function, 67

Share trading, 88

Sharpe ratio, 221, 222

Significance level, 162

Single relation, 121

Single universal table, 123

SOM - Self-Organizing Map, 43

Splitting criteria, 81

Splitting test, 81

- multiple-attribute, 81
- single-attribute, 81

SPSS, 28, 29

Simulated trading performance, 222

ST - Statistical Methods, 7

Stationarity, 23

Stationary time series, 25

Statistical Decision Support systems, 12

Statistical methods, 21, 29

Statistical significance, 29, 127, 151

Statistical systems

- SAS, 22
- SPSS, 22

Stochastic models

Dynamic Probabilistic Network (DPN), 107

Hidden Markov Model (HMM), 107

hierarchical mixture model, 108

Hierarchical Mixture of Experts (HME), 107

switching gated model, 107

Stopping criteria, 84

- global criterion, 84
- local criterion, 84
- Trepan, 84

Subjective probabilities, 272

Sub-rules, 155

Supervised learning, 14

TA - Technical Analysis, 11

Target value, 15

Technical Analysis, 2, 11

- difficulties, 11

Term, 137

Threshold, 37

Time series, 9

Tolerance, 32

Tolerance relation, 171

Trading performance, 31

Trading strategies, 220

- active, 219, 229
- Buy-and-Hold, 219
- passive, 219
- risk free, 219

Training examples, 15, 81, 117

Training sample, 15

Transformation group, 174

Tree

- complexity, 83

Trepan, 96

Tuning, 266, 267, 268

- algorithm, 267
- heuristic, 267

Type, 129, 132

Type system, 128

Typing, 134, 135, 140, 150

- knowledge, 150

- Unary function, 120**
- Unary predicates, 120, 146**
- Uncertainty, 237, 239**
 - lexical, 235, 237
 - linguistic, 235, 238
 - probabilistic linguistic, 237
 - stochastic, 235, 236, 237
- Understandable, 95, 117, 166**
- Understandable rules, 128**
- Understanding, 43**
- Units, 36**
 - hidden, 37
 - input, 37
 - output, 37
- Universal axiomatizable, 188**
- Universal formulas, 187**
- Universal relation, 123**
- Universe, 241**
- Unsupervised learning, 14, 107**
- Variablization, 148**
- Weighted links, 36, 37**