Intelligent Data Mining using Artificial Neural Networks and Genetic Algorithms: Techniques and Applications

By

Jianhua Yang

A dissertation submitted in fulfilment of the

requirements for the degree of Doctor of Philosophy

University of Warwick, School of Engineering

02/05/2010

Contents

List of Figures **6**
List of Tables **9**
Acknowledgements **10**
Declaration **11**
List of Author's Publications **12**
Abstract **14**
Abbreviations **16**

Chapter 1 An Introduction to Intelligent Data Mining **18**
1.1 Data Mining 18
1.1.1 Procedures and Tasks 20
1.1.2 Challenges and Scope 22
1.2 Intelligent Data Mining 24
1.2.1 Artificial Neural Networks (ANNs) 25
1.2.2 Genetic Algorithms (GAs) 36
1.3 Research Objectives 44
1.4 Thesis Outline 45
References 46

Chapter 2 Hybrid Intelligent System Data Mining Techniques and the Genetic Neural Mathematical Method **58**
2.1 Introduction 58
2.2 Hybrid Intelligent System Data Mining Techniques 59
2.2.1 Adaptive Neuro-Fuzzy Inference System (ANFIS) 59
2.2.2 Evolving Fuzzy Neural Network (EFuNN) 60
2.2.3 Fuzzy ARTMAP 63
2.2.4 Cartesian Genetic Programming (CGP) 64
2.3 The Genetic Neural Mathematical Method (GNMM) 66
2.3.1 Step 1 – Genetic Algorithm for Input Optimization 69
2.3.2 Step 2 – Multi-Layer Perceptron Modelling 77
2.3.3 Step 3 – Rule Extraction using Mathematical Programming 88
2.4 Summary 90
References 92

Chapter 3 Prediction of Longitudinal Dispersion Coefficient in Rivers 99
3.1 Introduction 99
3.2 Background 100
3.3 Datasets and Pre-processing 105
3.3.1 Datasets 105
3.3.2 Data Pre-processing 105
3.3.3 Division into Training and Testing Data 107
3.4 GNMM Implementation 110
3.4.1 Variable Selection 110
3.4.2 MLP Training 117
3.4.3 Rule Extractions 120
3.5 Discussions 123
3.5.1 Principal Component Analysis 123
3.5.2 Self-Organizing Map 127
3.6 Summary 130
References 132

Chapter 4 Channel Selection and Classification of EEG Signals .. 135
4.1 Introduction 135
4.2 Background 136
4.3 Data III – Two-Class Motor Imagery 140
4.3.1 Experiment Setup 140
4.3.2 Pre-processing 141
4.3.3 Channel Selection 143
4.3.4 Classification Results 144
4.4 Data IV – Response Priming Paradigm 147
4.4.1 The Experiment 147
4.4.2 Pre-processing 149
4.4.3 Channel Selection and Pattern Classification 150
4.4.4 Rule Extraction 155
4.5 Summary 157
References 159

Chapter 5 Optimising the Number of Electronic Nose Sensors .. 162
5.1 Introduction 162
5.2 Background – Multisensor Data Fusion 163
5.3 Intelligent System Techniques Applied to MDF Problems 166
5.4 Data V – Eye Bacteria Species 170
5.5 GNMM Results and Discussions 171
5.6 Summary 177
References 178

Chapter 6 Classification of the Pima Indians Diabetes Database 182
6.1 Introduction 182
6.2 Dataset 183
6.3 GNMM Results 184
6.4 Other Hybrid IS DM Techniques 186
6.4.1 ANFIS 186
6.4.2 EFuNN 191
6.4.3 Fuzzy ARTMAP 194
6.4.4 CGP 195
6.5 GA Parameter 199
6.5.1 Interactions among GA Parameters 200
6.5.2 Determine the Parameter Set 201
6.5.3 Ranges and Step Sizes 203
6.5.4 Results 204
6.5.5 Discussions 208
6.6 Summary 209
References 211

Chapter 7 Conclusions and Future Work 217
7.1 Results Overview 217
7.1.1 GNMM Steps 217
7.1.2 Case Study Results 218
7.1.3 Advantages/Disadvantages 220
7.2 Future research directions 222
References 224

Appendix A Data I – UK Environmental Agency Data 225
Appendix B Data II – US Dispersion Data 232
Appendix C Matlab Programme for GNMM 234
Appendix D VBA Programme for GNMM 257
Appendix E RMSE and Winning Variables for Case 7 of Data I ... 261
Appendix F Appearance Percentage of Data III 262

List of Figures

Figure 1-1: The evolution of database system technology (adapted from Han and Kamber 2006) 19
Figure 1-2: Data mining as a step in the process of knowledge discovery 21
Figure 1-3: Data mining as a confluence of many disciplines (adapted from Tan, Steinbach et al. 2006) 23
Figure 1-4: Schematic diagram of a biological neuron 26
Figure 1-5: Illustration of an artificial neuron 26
Figure 1-6: Architecture of ANNs. (a) Multilayer Feedforward Neural Network; (b) Self-Organizing Map; (c) Multilayer Recurrent Neural Network; (d) Cellular Neural Network 28
Figure 1-7: Global optimization approaches (adapted from Coello Coello, Lamont et al. 2007) 37
Figure 1-8: Evolutionary Computation components 38
Figure 2-1: Adaptive Neuro-Fuzzy Inference System 61
Figure 2-2: Architecture of Evolving Fuzzy Neural Network (adapted from Kasabov 2007) 61
Figure 2-3: Architecture of Fuzzy ARTMAP (adapted from Xu, Xuan et al. 2009) 63
Figure 2-4: A possible CGP genotype and corresponding phenotype for a 2-bit parallel multiplier circuit (adapted from Walker and Miller 2008) 65
Figure 2-5: Interactions between GNMM components. (a) The GNMM algorithm; (b) A Simple Genetic Algorithm; (c) A three-layer MLP 68
Figure 2-6: Binary coding chromosome 70
Figure 2-7: Genetic operators 70
Figure 2-8: Adaptive mutation rate 74
Figure 2-9: Sample activation functions 82
Figure 2-10: The Back-Propagation training algorithm 84
Figure 2-11: GNMM extracts regression rules from trained MLPs 89
Figure 3-1: RMSE and winning variables for Data II training subset in Case 2 111
Figure 3-2: Appearance percentage for Case 2 of Data II training subset 113
Figure 3-3: Appearance percentage for Data I training subset 114
Figure 3-4: Appearance percentage for Data II training subset 114
Figure 3-5: Comparison of performance using all variables and selected variables for Data I training subset for a single run 116
Figure 3-6: Comparison of performance using all variables and selected variables for Data II training subset for a single run 116
Figure 3-7: Predicted and measured longitudinal dispersion coefficients for Data I 118

Figure 3-8**:** Predicted and measured longitudinal dispersion coefficients for Data II....118
Figure 3-9: Percentage of the first 7 principal components in Data I 124
Figure 3-10: Projections of Data I points and variables onto the first two principal components 124
Figure 3-11: Percentage of the first 5 principal components in Data II 126
Figure 3-12: Projections of Data II points and variables onto the first two principal components 126
Figure 3-13: SOM analysis of Data I....128
Figure 3-14: SOM analysis of Data II....130
Figure 4-1: Data III – two-class imaginary movements (Adapted from Lal, Hinterberger et al. 2005) 141
Figure 4-2: Least square approximation for a signal segment in Data III 142
Figure 4-3: Target and predicted values for Data III training/validation set 146
Figure 4-4: Schematic representation of stimulus material and trial structure in Data IV experiments 148
Figure 4-5: Position of EEG electrodes used in Data IV experiments arranged by: (a) position and (b) number 148
Figure 4-6: EEG signal of channel Cz for the first epoch of Data IV event No.1 and its LS approximations across different time windows 149
Figure 4-7: Appearance percentage distribution around the scalp for Data IV subsets....152
Figure 4-8: Classification accuracy for different subsets in Data IV 153
Figure 4-9: Target and predicted values for subset O1 154
Figure 4-10: Histogram of extracted regression rules from Data IV subset O1....156
Figure 5-1: Schematic architecture of an electronic nose 164
Figure 5-2: Statistics of the dataset....171
Figure 5-3: Appearance of each sensor in Data V for a single case ((a), (b) and (c)) and the mean appearance for all cases (d) 174
Figure 5-4: Comparisons of the RMSE for the last 50 generations for each case....175
Figure 6-1: Appearance percentage for each attribute in Data VI 185
Figure 6-2: Structure of ANFIS generated for Data VI 187
Figure 6-3: Target/predicted class values and ANFIS prediction error for Data VI....187
Figure 6-4: ANFIS rule viewer applied to Data VI 189
Figure 6-5: Rules extracted from the ANFIS system for Data VI 189
Figure 6-6: Membership functions for Attri5 190
Figure 6-7: Rules surface the ANFIS system for Data VI 190
Figure 6-8: Target and EFuNN prediction class values for Data VI....192
Figure 6-9: Rules extracted from the EFuNN system for Data VI....192
Figure 6-10: EFuNN MFs for the first two attributes of Data VI....193

Figure 6-11: Target/predicted class values and Fuzzy ARTMAP prediction error for Data VI 194
Figure 6-12: CGP settings and simulation results for Data VI 196
Figure 6-13: Arithmetic rules extracted from CGP for Data VI 196
Figure 6-14: Fitness values for 4 different population sizes 205
Figure 6-15: Fitness values for 4 different crossover probabilities 205
Figure 6-16: Fitness values for 5 different mutation probabilities.....206
Figure 6-17: Fitness value decreases as generation increases.....206
Figure 6-18: Appearance percentage calculated from GAs that were used to determine the optimal population number (a), crossover probability (b), mutation probability (c), and generation number (d).....208

List of Tables

Table 3-1: Variables in Data I and II 106
Table 3-2: Data I statistics 109
Table 3-3: Data II statistics 109
Table 3-4: GA parameters and CPU speeds/time 111
Table 3-5: Comparison of Data II_s (testing subset) results when using 4 different methods. For GNMM, the mean RMSE of 5 runs are given along with the standard deviations 119
Table 3-6: Rules fired for Data I 122
Table 3-7: Rules fired for Data II 122
Table 4-1: Configuration of GAs for Data III channels selection 143
Table 4-2: Classification results for Data III, which shows the results for training/validation and testing subset respectively 146
Table 4-3: EEG channels selected for each subset in Data IV 153
Table 5-1: GNMM configurations for the Data V 171
Table 6-1: Data VI statistics 183
Table 6-2: 10 most significant rules fired for Data VI 185
Table 6-3: Comparison of classification results for Data VI 198
Table 6-4: GA initial parameter range and step size 203
Table 6-5: Optimal GA parameters for Data VI 208
Table 6-6: Feature comparison of GNMM with other IS DM techniques 210
Table 7-1: A summary of case study data and results 219

Acknowledgements

I would like to thank my supervisors, Prof. Evor L. Hines and Dr. Daciana D. Iliescu, for their wonderful support, excellent supervision and endless guidance through the research and writing of this thesis. Their constant enthusiasm for my work was tireless and inspirational.

I would also like to thank my wife, Shan, for her support and tolerance of this work, as well as all night companionship during the final phase of this work. Special thanks go to all my family and friends for the help and support that they have given me in so many ways.

Thanks also go to Dr. Mark S Leeson, Prof. Ian Guymer, Prof. Nigel G. Stocks, Mr. Harsimrat Singh, Mr. Xu-Qin Li, Mr. John Erik Sloper, Ms Rachel Corke at the School of Engineering, and Dr. Friederike Schlaghecken at the Department of Psychology, University of Warwick, for all your support, guiding, discussions etc. in one way or another. I am also grateful for the encouragement I received over the years from Prof. Gregory P. King at the University of Lisbon.

Last but not least, I thank the financial support from the Warwick Postgraduate Research Fellowship (WPRF) and UK Overseas Research Students Awards Scheme (ORSAS).

Declaration

The work described in this thesis was conducted by the author, except where stated otherwise, in the School of Engineering, University of Warwick between the dates of October 2006 and September 2009. No part of this work has been previously submitted to the University of Warwick or any other academic institution for admission to a higher degree. All publications to date arising from this thesis are listed in the next section.

List of Author's Publications

Edited Book

E. L. Hines, M. S. Leeson, M. Martínez-Ramón, M. Pardo, E. Llobet, D. D. Iliescu and J. Yang (2008), 'Intelligent Systems: Techniques and Applications', Shaker Publishing, ISBN 978-90-423-0345-4

Book Chapters

X-Q Li, M. S. Leeson, E. L. Hines, D. S. Huang, J. Yang, D. D. Iliescu (2010), 'Neural Networks for Solving Linear and Quadratic Programming Problems with Modified Newton's and Levenberg-Marquardt Methods', in 'Advances in Mathematics Research', Volume 11, Editors: A. R. Baswell, Nova Publishers, ISBN 978-16-087-6970-4

J. Yang, E. L. Hines, I. Guymer, D. D. Iliescu, M. S. Leeson, G. P. King and X-Q Li (2008), 'A Genetic Algorithm-Artificial Neural Network Method for the Prediction of Longitudinal Dispersion Coefficient in Rivers', in 'Advancing Artificial Intelligence through Biological Process Applications', Editors: A. Porto, A. Pazos and W. Buño, Idea Group Inc., ISBN 978-15-990-4996-0

J. Yang, E. L. Hines, D. D. Iliescu, M. S. Leeson and P. Boilot (2008), 'Optimising the Number of Electronic Nose Sensors needed using Genetic Algorithms and Neural Networks', in 'Intelligent Systems: Techniques and Applications', Editors: E. L. Hines, M. S. Leeson, M. Martínez-Ramón, M. Pardo, E. Llobet, D. D. Iliescu and J. Yang, Shaker Publishing, ISBN 978-90-423-0345-4

J. Yang, E. L. Hines, I. Guymer, D. D. Iliescu and M. S. Leeson (2008), 'Multi-input Optimisation of River Flow Parameters and Rule Extraction Using Genetic-Neural Technique', in 'Intelligent Systems: Techniques and Applications', Editors: E. L. Hines, M. S. Leeson, M. Martínez-Ramón, M. Pardo, E. Llobet, D. D. Iliescu and J. Yang, Shaker Publishing, ISBN 978-90-423-0345-4

Journal Papers

J. Yang, E. L. Hines, J. E. Sloper, D. D. Iliescu, M. S. Leeson, (2010, under review). 'A Comparative Study of Hybrid Machine Learning Methods in the Classification of the Pima Indians Diabetes Database', Engineering Applications of Artificial Intelligence, ISSN 0952-1976

J. Yang, H. Singh, E. L. Hines, F. Schlaghecken, D. D. Iliescu, M. S. Leeson, and N. G. Stocks (2009, under review). 'Channel Selection and Classification of EEG

Signals: A Neural Network and Genetic Algorithm–based approach', Neural Computing and Applications, ISSN 0941-0643

G. P. King, J. Yang, J. Dias and N. Serra (2006), 'EOF Analysis of Seasonal and Interannual Variability of the Surface Circulation along the West Iberian Coast from 1995-2005', Geophysical Research Abstracts, Vol. 8, 03127, ISSN 1607-7962

Conference Papers

H. Singh, J. Yang, E. L. Hines, N. G. Stocks (2009), 'Channel selection for multi channel multi trial invasive BCI data' in International Conference on Electronic Design and Signal Processing (ICEDSP), Manipal Institute of Technology, Karnataka, India

J. Yang, E. L. Hines, I. Guymer, D. D. Iliescu and M. S. Leeson (2008), 'A Genetic Algorithm-Based Input Determination Method for Neural Networks' in IEEE Workshop and Summer School on Evolutionary Computing Lecture Series by Pioneers (WSSEC), Derry, Northern Ireland

J. Yang, E. L. Hines, I. Guymer, D. D. Iliescu, M. S. Leeson and G. P. King (2007), 'Prediction of Longitudinal Dispersion Coefficient in Rivers using Variables Identified by Genetic Algorithms', in the Fifth International Symposium on Environmental Hydraulics (ISEH V), Tempe, Arizona

J. Yang, E. L. Hines, D. D. Iliescu and M. S. Leeson (2007), 'GNMM and Accurate Longitudinal Dispersion Coefficient Prediction', in the Seventh UK Chinese Association of Resource and Environment (CARE) Annual Meeting, Greenwich, London, ISBN 978-09-551-9652-2

Other Publications

J. Yang, E. L. Hines, F. Schlaghecken, D. D. Iliescu, and M. S. Leeson (2009), 'Neural Network-based EEG Classification and Rule Extraction', CCNS (Centre for Cognitive & Neural Systems) Seminar Series, Warwick Digital Laboratory, University of Warwick, UK

E. L. Hines, J. Yang, P. Boilot, M. S. Leeson, D. D. Iliescu and J.W. Gardner (2007), 'Intelligent Systems for Gas Sensing', in the Joint Sensors, Instrumentation and Nanotechnology KTN Event, National Physical Laboratory, Teddington, UK

F. Schlaghecken, E. L. Hines, M. S. Leeson, D. D. Iliescu and J. Yang (2008), 'Genetic Algorithm-Artificial Neural Network Methods for the Identification and Prediction of Dynamic Functional Links in Human Cortical Activity Associated with Cognitive Control Processes', report to the Warwick Institute of Advanced Study (IAS)

Abstract

Data Mining (DM) refers to the analysis of observational datasets to find relationships and to summarize the data in ways that are both understandable and useful. Many DM techniques exist. Compared with other DM techniques, Intelligent Systems (ISs) based approaches, which include Artificial Neural Networks (ANNs), fuzzy set theory, approximate reasoning, and derivative-free optimization methods such as Genetic Algorithms (GAs), are tolerant of imprecision, uncertainty, partial truth, and approximation. They provide flexible information processing capability for handling real-life situations. This thesis is concerned with the ideas behind design, implementation, testing and application of a novel ISs based DM technique. The unique contribution of this thesis is in the implementation of a hybrid IS DM technique (Genetic Neural Mathematical Method, GNMM) for solving novel practical problems, the detailed description of this technique, and the illustrations of several applications solved by this novel technique.

GNMM consists of three steps: (1) GA-based input variable selection, (2) Multi-Layer Perceptron (MLP) modelling, and (3) mathematical programming based rule extraction. In the first step, GAs are used to evolve an optimal set of MLP inputs. An adaptive method based on the average fitness of successive generations is used to adjust the mutation rate, and hence the exploration/exploitation balance. In addition, GNMM uses the elite group and appearance percentage to minimize the randomness associated with GAs. In the second step, MLP modelling serves as the core DM engine in performing classification/prediction tasks. An Independent Component Analysis (ICA) based weight initialization algorithm is used to determine optimal weights before the commencement of training algorithms. The Levenberg-Marquardt (LM) algorithm is used to achieve a second-order speedup compared to conventional Back-Propagation (BP) training. In the third step, mathematical programming based rule extraction is not only used to identify the premises of multivariate polynomial rules, but also to explore features from the extracted rules based on data samples associated with each rule. Therefore, the methodology can provide regression rules and features not only in the polyhedrons with data instances, but also in the polyhedrons without data instances.

A total of six datasets from environmental and medical disciplines were used as case study applications. These datasets involve the prediction of longitudinal dispersion coefficient, classification of electrocorticography (ECoG)/Electroencephalogram (EEG) data, eye bacteria Multisensor Data Fusion (MDF), and diabetes classification (denoted by Data I through to Data

VI). GNMM was applied to all these six datasets to explore its effectiveness, but the emphasis is different for different datasets. For example, the emphasis of Data I and II was to give a detailed illustration of how GNMM works; Data III and IV aimed to show how to deal with difficult classification problems; the aim of Data V was to illustrate the averaging effect of GNMM; and finally Data VI was concerned with the GA parameter selection and benchmarking GNMM with other IS DM techniques such as Adaptive Neuro-Fuzzy Inference System (ANFIS), Evolving Fuzzy Neural Network (EFuNN), Fuzzy ARTMAP, and Cartesian Genetic Programming (CGP). In addition, datasets obtained from published works (i.e. Data II & III) or public domains (i.e. Data VI) where previous results were present in the literature were also used to benchmark GNMM's effectiveness.

As a closely integrated system GNMM has the merit that it needs little human interaction. With some predefined parameters, such as GA's crossover probability and the shape of ANNs' activation functions, GNMM is able to process raw data until some human-interpretable rules being extracted. This is an important feature in terms of practice as quite often users of a DM system have little or no need to fully understand the internal components of such a system. Through case study applications, it has been shown that the GA-based variable selection stage is capable of: filtering out irrelevant and noisy variables, improving the accuracy of the model; making the ANN structure less complex and easier to understand; and reducing the computational complexity and memory requirements. Furthermore, rule extraction ensures that the MLP training results are easily understandable and transferrable.

Abbreviations

AI	Artificial Intelligence
AN	artificial neuron
ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
ART	Adaptive Resonance Theory
BCI	Brain-Computer Interface
BN	biological neuron
BNS	biological neural system
BP	Back-Propagation
CGP	Cartesian Genetic Programming
CNN	Cellular Neural Network
CSP	common spatial patterns
DDR	Data Dimensionality Reduction
DM	data mining
EC	Evolutionary Computation
ECoG	electrocorticography
EDA	Exploratory Data Analysis
EEG	Electroencephalogram
EFuNN	Evolving Fuzzy Neural Network
EN	electronic nose
ERS/ERD	event-related synchronization/desynchronization
FCM	fuzzy c-means
FIS	fuzzy inference system
FNN	Feedforward Neural Network
FS	Feature Selection
GA	Genetic Algorithm
GNMM	Genetic Neural Mathematical Method
GP	Genetic Programming
ICA	Independent Component Analysis
IS	Intelligent System

KDD	Knowledge Discovery from Data
LM	Levenberg-Marquardt
LS	least square
MDF	Multisensor Data Fusion
MF	membership function
MLP	Multi-Layer Perceptron
MP	McCulloch-Pitts
MSE	Mean Square Error
PCA	Principal Components Analysis
PDF	Probability Density Function
PDGP	Parallel Distributed Genetic Programming
PE	processing element
PNN	Probabilistic Neural Network
PR	Pattern Recognition
RBF	Radial Basis Function
RFE	Recursive Feature Elimination
RNN	Recurrent Neural Network
RT	reaction time
SA	Simulated Annealing
SBS	Sequential Backward Selection
SC	Soft Computing
SFS	Sequential Forward Selection
SGA	Simple Genetic Algorithm
SOM	Self-Organizing Map
STD	standard deviation
SVD	Singular Value Decomposition
SVM	Support Vector Machine

Chapter 1 An Introduction to Intelligent Data Mining

1.1 Data Mining

Data mining (DM) refers to the analysis of observational datasets to find relationships and to summarize the data in ways that are both understandable and useful to the data owner (2001). The first book on DM appeared in 1991 (Piatetsky-Shapiro and Frawley 1991). However, the idea is not totally new – people have been seeking patterns in data since human life began: Hunters looked for patterns in animal migration behaviour, farmers looked for patterns in crop growth, politicians seek patterns in voter opinion, and lovers seek patterns in their partners' responses (Chakrabarti 2009).

In recent years DM has attracted great attention in the information industry and in society as a whole. This is because, on the one hand, modern computers and other piece of equipment are able to produce and store virtually unlimited datasets characterizing a complex system. In fact, database and information technology has been evolving systematically from primitive file processing

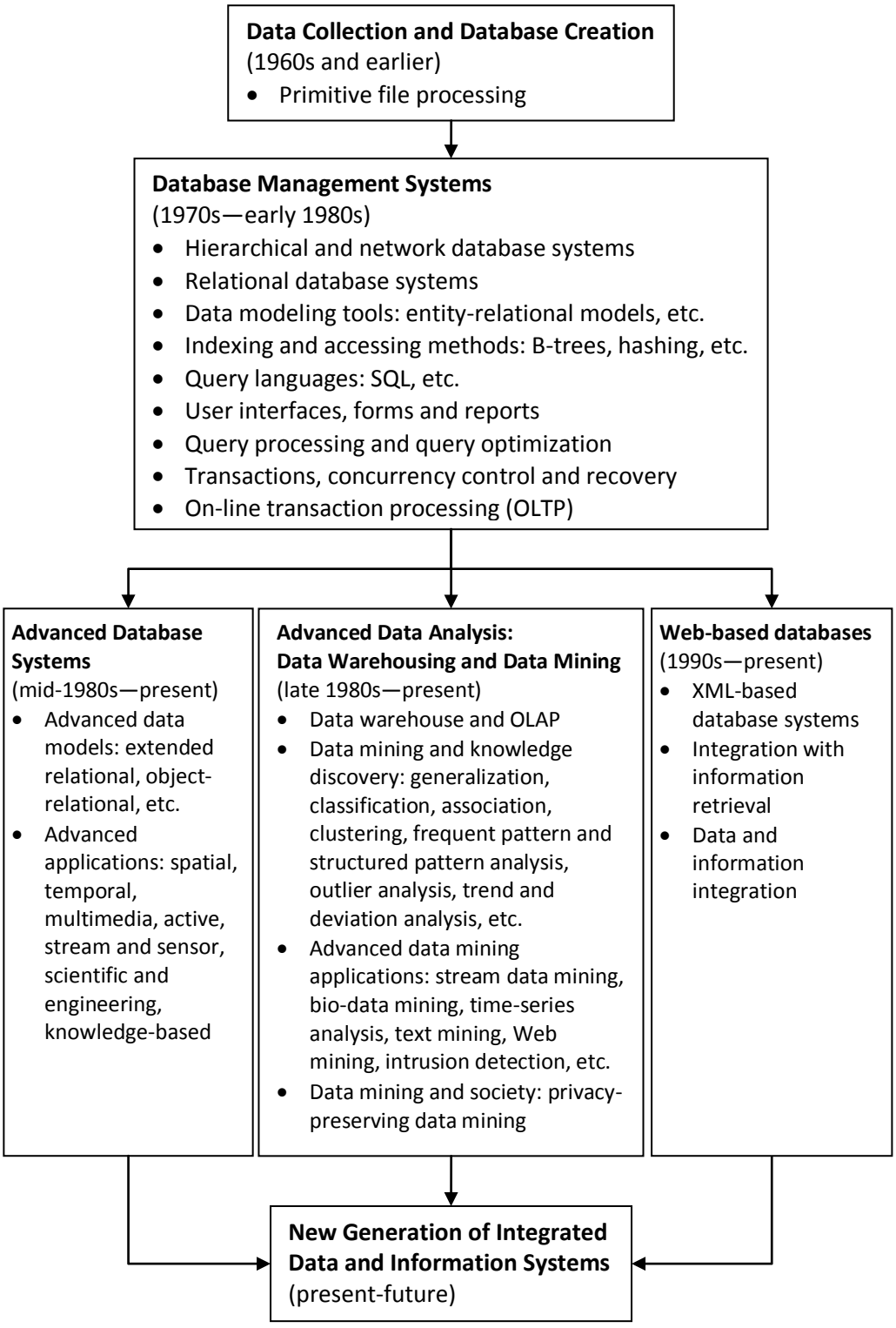


Figure 1-1: The evolution of database system technology (adapted from Han and Kamber 2006)

systems to sophisticated and powerful database systems as shown in Figure 1-1. On the other hand, however, there is no concise set of parameters that can fully describe the state of real-world complex systems studied nowadays by engineers, psychologists, economists, etc. (Busygin, Prokopyev et al. 2008). These on the contrary inspire the development of advanced DM which may employ techniques such as Artificial Neural Networks (ANNs), Genetic Algorithms (GAs), Support Vector Machines (SVMs), and fuzzy logic etc.

1.1.1 Procedures and Tasks

The aim of DM may be defined in many ways depending on the applications. For example the extraction of implicit, possibly previously unknown and potentially useful patterns and models from data, to uncover knowledge within the data associated with different processes and models (Charaniya, Hu et al. 2008; Elleithy 2008). From this point of view, DM is often set in the broader context of Knowledge Discovery from Data (KDD) (Tan, Steinbach et al. 2006). The KDD process involves several stages from data integration to knowledge interpretation of DM results, as shown in Figure 1-2.

It may also be inferred from Figure 1-2 that although the DM algorithms are central to knowledge discovery, the pre-processing of the data and the interpretation (as opposed to the blind use) of the results are both of great importance (Bramer 2007). This is due to the fact that pre-processing steps have a direct impact on the quality of the data that go into the DM engine;

while the interpretation of DM results may limit the its application and implementation.

There are different ways of categorizing DM tasks. For example, some researchers divide them into two categories – predictive tasks and descriptive tasks (Tan, Steinbach et al. 2006); while some others argue that there are more groups other than the two mentioned previously, e.g. Exploratory Data Analysis (EDA), discovering patterns and rules etc. (Hand, Mannila et al. 2001). This thesis adapts the categorization that captures the processes of a DM activity, i.e., data pre-processing, DM modelling, and knowledge description, as follows (Wang and Fu 2005):

- Data Dimensionality Reduction (DDR)
- Classification and Clustering
- Rule Extraction

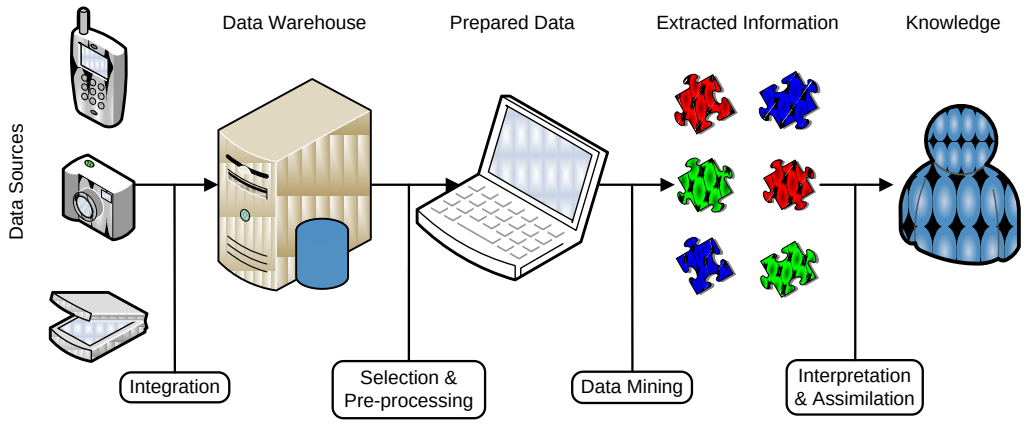


Figure 1-2: Data mining as a step in the process of knowledge discovery

DDR often involves feature extraction or feature selection, where new features are derived from the original data in order to reduce dimensionality and hence increase computational efficiency and classification accuracy. DDR utilizes techniques such as GAs, Principal Components Analysis (PCA), Sequential Forward Selection (SFS), and Sequential Backward Selection (SBS) etc. Classification and clustering is the process that connects DDR and rule extraction where various statistical and machine learning methods can be applied (e.g. linear regression, Radial Basis Function i.e. RBF). Rule extraction aims to present classification/clustering results in such a way that the data are easily understandable and knowledge gained from the data are transferable.

1.1.2 Challenges and Scope

Since its conception DM has achieved tremendous success. However, many new problems have emerged and there is still a lack of timely exchange of important topics in the community as a whole. In October 2005, Yang and Wu (2006) took the initiative to identify 10 challenging problems in data mining research, including the following:

- Developing a unifying theory of DM
- Scaling up for high dimensional data and high speed data streams
- DM for biological and environmental problems
- Mining complex knowledge from complex data

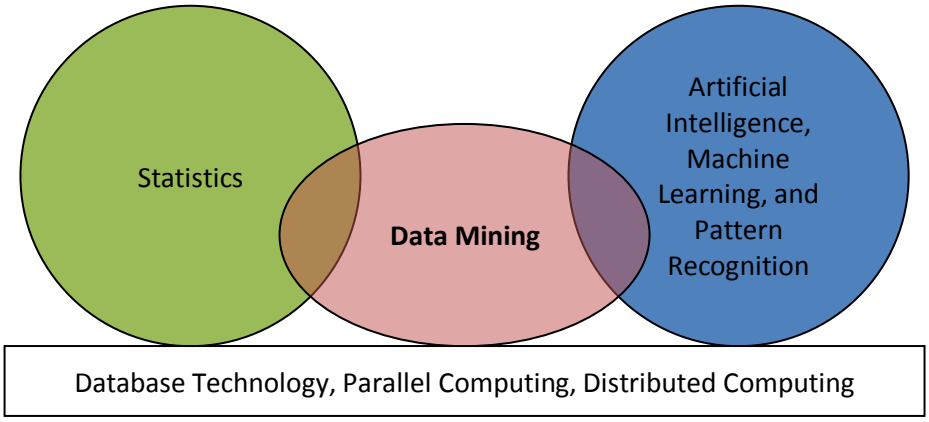


Figure 1-3: Data mining as a confluence of many disciplines (adapted from Tan, Steinbach et al. 2006)

Among the problems listed above, the first comes from the fact that DM is an inter-disciplinary field drawn upon disciplines such as statistics, machine learning, Pattern Recognition (PR), Artificial Intelligence (AI), database technology, and other areas as in Figure 1-3. Although Figure 1-3 shows a distinction between various techniques, in practice it is sometimes difficult to determine which discipline a specific technique belongs to. For example, decision tree is often regarded as a standard DM technique (Maimon 2007). However, Tan, Steinbach et al. (2006) treat it as a statistical classification method; Witten and Frank (2005) treat it as a kind of knowledge representation; Berthold and Hand (2003) use it in a so-called 'fuzzy decision tree', which makes it some sort of a hybrid between DM and AI techniques. This demonstrates the diversity of disciplines contributing to DM and that DM systems can be categorized according to various criteria such as the kinds of techniques utilized or the kinds of knowledge mined (Han and Kamber 2006).

1.2 Intelligent Data Mining

A common feature of all DM techniques is that they are all, to a certain extent, data analysis methods and can support/interact with each other. However, each discipline has its own distinct attributes that make it particularly useful for certain types of problems and situations. For example, the most fundamental difference between classical statistical applications and data mining may be suggested to be the size of the dataset. Statistical techniques alone may not be sufficient to address some of the more challenging issues in data mining, especially those arising from very large datasets (Hand, Mannila et al. 2001). On the other hand, an Intelligent System (IS) is all about learning rules and patterns from the data (Thuraisingham 1999). With the help of available computational power in IS tools, there is a great potential for significant advances in both theoretical and applied research in this DM area.

The term Intelligent Systems (ISs) is used interchangeably with Soft Computing (SC) in this thesis. It is a collection of methodologies that works synergistically and provides, in one form or another, flexible information processing capability for handling real-life situations. It differs from conventional data analysis (e.g. statistical methods) in that it is tolerant of imprecision, uncertainty, partial truth, and approximation (Venugopal 2009). It aims to exploit the tolerance for imprecision, uncertainty, approximate reasoning, and partial truth in order to achieve tractability, robustness, and low-cost solutions. The guiding principle is to devise methods of computation that lead to an

acceptable solution at low cost, by seeking for an approximate solution to an imprecisely or precisely formulated problem (Mitra and Acharya 2003).

1.2.1 Artificial Neural Networks (ANNs)

IS techniques consist of several computing paradigms, including ANNs, fuzzy set theory, approximate reasoning, and derivative-free optimization methods such as GAs and Simulated Annealing (SA) (Jang, Sun et al. 1997). It is well known that biological neural systems (BNSs) can perform extraordinarily complex computations without recourse to explicit quantitative operations, and are capable of learning over time. This property is thought to reflect the ability of large ensembles of neurons to learn through exposure to external stimuli and to generalize across related instances of the signal (Berthold and Hand 2007). Such properties of BNSs make them attractive as a model for IS methods. In fact, ANNs are distributed, adaptive, generally nonlinear means of learning comprised of different processing elements (PEs) called neurons (Bishop 1995). They are based on a computing model similar to the underlying structure of the human brain, the aim being to model the brain's ability to learn and/or adapt in response to external inputs.

1.2.1.1 Biological Roots

The basic building blocks of BNSs are nerve cells, referred to hereafter as biological neurons (BNs). A BN typically consists of a cell body, dendrites and an axon, as shown in Figure 1-4. From the cell body protrudes a number of

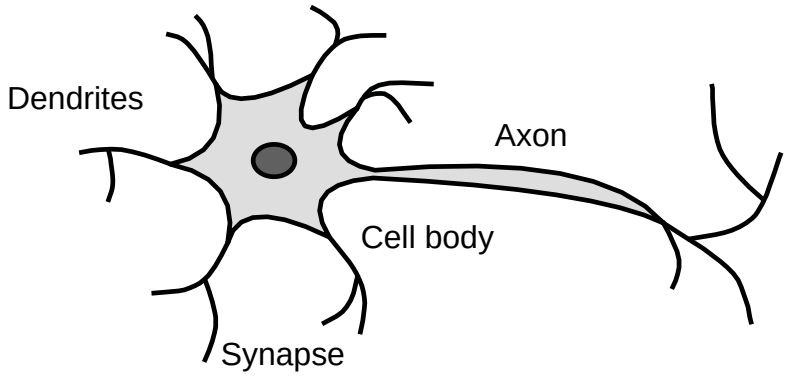


Figure 1-4: Schematic diagram of a biological neuron

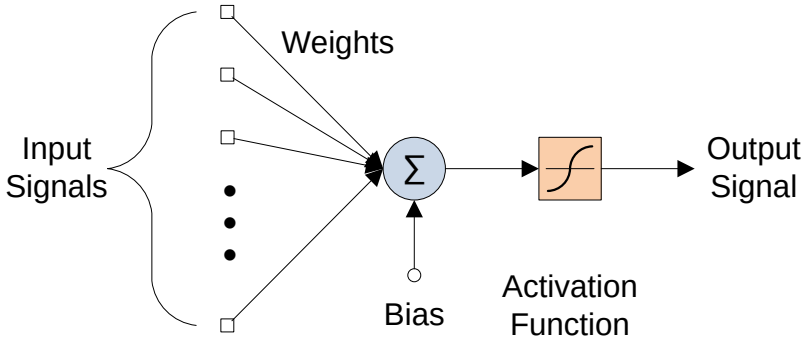


Figure 1-5: Illustration of an artificial neuron

branches called dendrites; the cell body and dendrites constitute the input to the neuron. There also extrudes from the cell body a long fibre called the axon (Arbib 2003). Neurons are massively interconnected, where an interconnection is between the axon of one neuron and one or more dendrites of one or more other neurons. This connectivity is referred to as a synapse. Signals propagate from the dendrites, through the cell body to the axon; from where the signals are propagated to all connected dendrites.

An artificial neuron (AN), also sometimes called PE, is a model of a BN. Although various types of ANs are being used in current research, the most

widely used is the McCulloch-Pitts (MP) model (Engelbrecht 2007). Figure 1-5 is a representation of an MP neuron. Each AN receives signals from the environment, or other ANs, gathers these signals and, when fired, transmits a signal to all connected ANs. Input signals are inhibitory or excitatory through negative and positive weights associated with each connection to the AN. The firing of an AN and the strength of the input signal are controlled via a function (i.e. activation function). Each neuron typically receives signals from outside, or from other neurons. When fired, these neurons compute a net input signal as a function of the respective weights. The net signal serves as input to the activation function using which the neuron then determines the output signal.

1.2.1.2 Fundamentals – Architectures and Training Algorithms

There are many different types of ANN models rather than a single type. Each form of ANN has different characteristics for a specific set of conditions, analogous to the functional specificity associated with different regions of the brain (Berthold and Hand 2007). However, all ANN models are specified in terms of three basic entities: models of the neurons themselves, models of synaptic interconnections and structures, and the training rules for updating the connecting weights (Lin and Lee 1996).

An ANN consists of a number of highly connected ANs such that each AN is connected to other ANs or to itself. According to the architecture, ANNs can be roughly classified into Feedforward Neural Networks (FNNs), Recurrent Neural

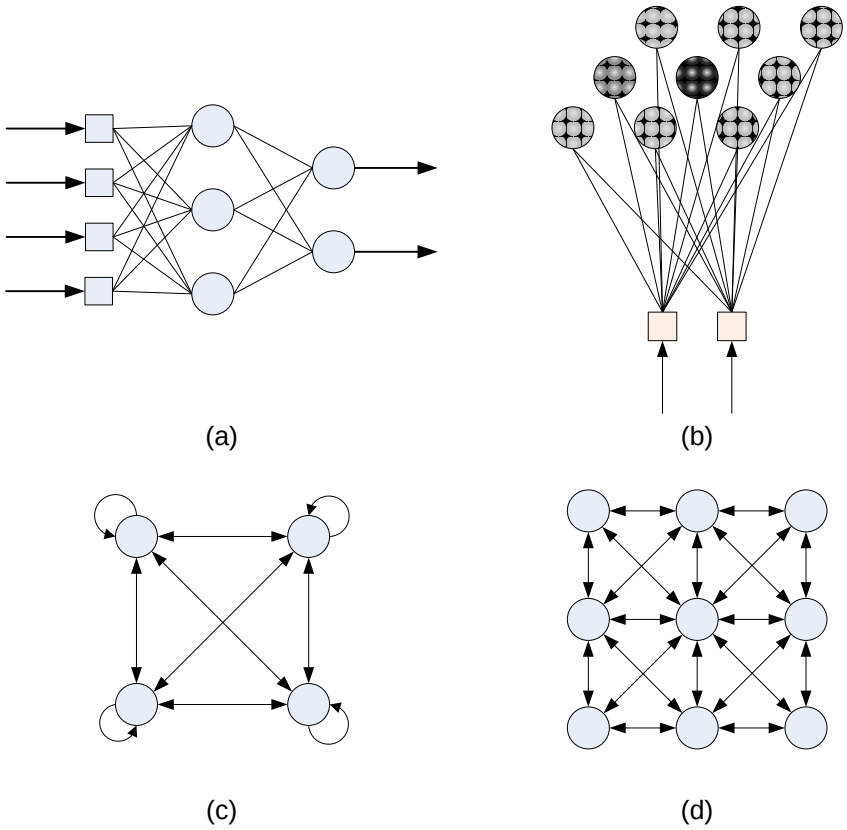


Figure 1-6: Architecture of ANNs. (a) Multilayer Feedforward Neural Network; (b) Self-Organizing Map; (c) Multilayer Recurrent Neural Network; (d) Cellular Neural Network

Networks (RNNs), and their combinations. Some popular network topologies including fully connected FNNs, RNNs, Self-Organizing Maps (SOMs), and Cellular Neural Networks (CNNs) are shown in Figure 1-6.

Figure 1-6 (a) shows a typical architecture of a FNN – ANs are arranged in layers, and each AN is connected to all ANs in adjacent layers. There is no connection between the neurons within each layer. The information flows in away whereby each AN takes inputs from all the nodes in the preceding layer and sends its single output value to all the nodes in the next layer. The leftmost layer (i.e. the input layer) is provided with input by the user, and the

output from the rightmost layer (i.e. the output layer) is the output which is finally used to do something useful (Millington 2006).

Popular FNNs include Multi-Layer Perceptrons (MLPs) and RBF networks, which are both fully connected layered FNNs. The MLP is the most popular arrangement of ANs (Haykin 1994; Hagan, Demuth et al. 1996; Jang, Sun et al. 1997). It has been shown (Cybenko 1989) that MLPs can approximate virtually any function with any desired accuracy, provided that there are enough hidden neurons in the network and that a sufficient amount of data is available. An MLP usually consists of three layers – an input layer, a hidden layer and an output layer. The number of input neurons is typically determined to correspond to the dimension of the input vector. The number of neurons in the hidden layer is determined experimentally and the dimension of the output vector to be modelled or the number of classes to be classified generally determines the number of output neurons. Each neuron has a number of inputs (from outside the neural network or the previous layer) and a number of outputs (leading to the subsequent layer or out of the neural network). A neuron computes its output response based on the weighted sum of all its inputs according to an activation function. Data flows in one direction through this kind of neural network starting from external inputs into the first layer, which are transmitted through the hidden layer(s), and then passes on to the output layer from which the external outputs are obtained.

RBF networks are supervised learning paradigms very similar to MLP except that they use radial basis transfer functions for the hidden layer rather than linear or sigmoidal ones. The RBF's operational principle is that it paves the input space with overlapping receptive fields, as they classify data using hyper-spheres rather than hyper-planes (Lin and Lee 1996).

The SOM is a feed forward unsupervised learning network (Kohonen 2001). It typically contains a two-dimensional single layer of neurons in addition to an input layer of branched nodes, as illustrated in Figure 1-6 (b). SOM neurons have two different types of connections. There are forward connections from the neurons in the input layer to the neurons in the output layer, and also lateral connections between neurons in the output layer. The lateral connections are used to create a competition between neurons.

FNNs can have loops: connections that lead from a later layer back to earlier layer(s). This architecture is known as a recurrent network. The architecture of Figure 1-6 (c) shows a typical RNN – the neurons are arranged in a grid, and connections are made between themselves and neighbouring points in the grid. FNNs such as the Hopfield network represent an auto-associative type of memory. However, they can have very complex and unstable behaviour and are typically much more difficult to control (Millington 2006).

A CNN consists of regularly spaced neurons that communicate only with the neurons in its immediate neighbourhood. Adjacent ANs are connected by mutual interconnections. Each AN is excited by its own signals and by signals flowing from its adjacent cells (Du and Swamy 2006). The architecture of a CNN is shown in Figure 1-6 (d).

In general, ANN training algorithms can be classified into two broad categories: parameter learning and structure learning. The former is concerned with the updating of the connecting weights in an ANN; and the latter deals with the network topology and their inter-connections (Lin and Lee 1996). Although there are numerous training algorithms depending on the type of AN and ANN architecture, the Back-Propagation (BP) algorithm (Bryson and Ho 1975) is currently the most popular for performing supervised learning tasks. It is not only used to train FNNs such as MLPs, but has also been adapted to RNNs (Du and Swamy 2006).

In BP, for a given input-output pair, the algorithm performs two phases of data flow. First, the input pattern is propagated from the input layer to the output layer and, as a result of this forward flow of data, it produces an actual output. Then the error signals resulting from any difference between the expected and actual outputs are back-propagated from the output layer to the previous layers for them to update their weights until the input layer is reached.

Training algorithms for unsupervised ANNs are different as there is no desired output. For example, SOM training is based on a competitive learning strategy: measured based on the Euclidean distance, the best neuron learns by shifting its weights from inactive connections to active ones. In other words, the neuron with the largest activation level among all neurons in the output layer becomes the winner (the winner-takes-all neuron). This neuron is the only neuron that produces an output signal. The activity of all other neurons is suppressed in the competition. Neurons close to the winner are also updated according to the neighbourhood relationships. In this way, SOMs effectively cluster the input vectors through a competitive learning process, while maintaining the topological structure of the input space.

1.2.1.3 Advantages and Challenges

The idea of DM – extracting information from data – has existed for decades. However, what makes DM tasks extremely challenging nowadays is the development of computer processing and storage/database technologies, which allow for example gigabytes of data to remain offline or even online for further analysis. ANNs are a computing methodology whose fundamental purpose is to recognize patterns in data (Bigus 1996). Due to its biological roots, the advantages of ANNs have made them one of the key methodologies used for modern DM. ANNs have also been used for many applications such as pattern classification, time series analysis, prediction, and clustering (Ye 2003). In terms of DM, several important and yet distinct features of ANNs are:

- ANNs do not require a priori knowledge about the data, which is often the opposite to traditional statistical model-based methods. ANNs are highly adaptive and the network is largely determined by the characteristics or patterns the network learned from the data (Maimon 2007). This feature makes ANN a data-driven approach which is ideal for real world problems where the data size is large but the meaningful patterns or underlying structure are yet to be discovered and may not be possible to determine in advance.
- ANNs have robustness and fault-tolerant capability. ANNs can handle incomplete or noisy data. Since the whole network consists of many parallel ANs, it is a distributed information system and information is stored in a distributed manner by the network structure. Thus, the overall performance does not degrade significantly when the information at some node is lost or some connections in the network are damaged (Du and Swamy 2006). On the other hand, ANNs are capable of improving their performance by updating the connection weights using the learning rules.
- ANNs can perform nonlinear modelling. Depending on the activation functions being used, a single AN can be linear or nonlinear. Thus, networks that connect these ANs can process nonlinear data. Moreover, the nonlinearity is of a special kind in the sense that it is distributed throughout the network (Haykin 1999). This capability is extremely useful

in case, for example, the underlying physical mechanism responsible for generation of the data is inherently nonlinear.

- ANNs are typically structured as parallel-processing structures. ANNs are usually made up of a number of ANs, each of which performs simple addition, multiplication, division, and threshold operations. This parallel structure has the advantage of, for example, relatively higher calculation speed and also allows for highly parallel software and hardware implementations. This feature makes ANNs well suited for implementation using very-large-scale-integrated (VLSI) technology. One particular beneficial virtue of VLSI is that it provides a means of capturing truly complex behaviour in a highly hierarchical fashion.

ANNs have a so-called 'black-box' nature – even though they are successfully trained, no information is available from them in symbolic form, suitable for verification or interpretation by humans (Mitra, Pal et al. 2002). By design, ANNs learn according to their training algorithms. After successful training, depending on its specific type, some networks are fixed while others are allowed to adapt during operation (Taylor and Darrah 2005). Thus, it is a challenge to understand how the network will handle unknown input.

There has been quite a lot of work aimed at extracting knowledge from trained networks in the form of symbolic rules (Du and Swamy 2006; Mantas, Puche et al. 2006; Chow and Cho 2007; Kahramanli and Allahverdi 2009). In general,

algorithms for rule extraction from ANNs can be grouped into three categories (Kahramanli and Allahverdi 2009; Ozbakir, Baykasoglu et al. 2009):

- Decompositional approaches involve rule extraction at the level of hidden and output units. This involves the extraction of rules from a network in a neuron-by-neuron series of steps. The advantage of this approach is that they can generate a complete set of rules for the trained ANNs. However, the process can be tedious and result in large and complex descriptions.
- Pedagogical approaches try to map inputs directly into outputs and views ANNs as black-boxes. The aim is to extract symbolic rules which map the input-output relationship as closely as possible. The number of these rules and their form do not directly correspond to the number of weights or the architecture of the ANN.
- Eclectic approaches incorporate elements of both decompositional and pedagogical techniques.

As more and more databases become available DM techniques such as rule extraction from ANNs has become a popular research topic. However, another equally important issue (e.g. in terms of their impact on the implementation of ANNs) – ANN input selection – has not invoked much of an interest. The fact that little attention was given to the matter of whether or not the inputs used to train the ANN are the most appropriate ANN inputs is basically due to the fact that not all of the available variables are necessarily equally informative

(since some may be correlated, noisy or have no significant relationship with the output variable(s) of interest) (Maier and Dandy 2000; Bowden, Dandy et al. 2005). According to Alexandridis, Patrinos et al. (2005) and Bowden, Dandy et al. (2005), the lack of input determination for ANNs may result in the following consequences:

- Irrelevant variables may add extra noise which has consequential impact on the accuracy of the model.
- Understanding complex models may be more difficult than understanding simple models that give comparable results.
- As input dimensionality increases, the computational complexity and memory requirements of the model increase.

1.2.2 Genetic Algorithms (GAs)

The second IS technique introduced in the current chapter is the GA. Techniques that are concerned with the determination of ANN inputs may be described differently in the literature (Yao 1999; Maier and Dandy 2000; Ramadan, Song et al. 2001; Alexandridis, Patrinos et al. 2005; Grivas and Chaloulakou 2006; Gualdron, Llobet et al. 2006). From the point of view of optimization, selecting appropriate inputs for ANNs can be treated as an optimization problem. That is, optimizing ANN inputs such that it achieves better performance. In a broader sense, a GA is a stochastic optimization methodology that belongs to the Evolutionary Computation (EC) family. Thus,

considerations of GAs are firstly given in the context of stochastic optimization.

1.2.2.1 Stochastic Optimization and Evolutionary Computation

Generally speaking, optimization techniques are classified into three categories (see Figure 1-7): enumerative, deterministic and stochastic (Coello Coello, Lamont et al. 2007). An enumerative search is deterministic but it does not employ any heuristics. This technique is inefficient as it tests each possible solution. Deterministic algorithms such as greedy and hill-climbing algorithms incorporate problem domain knowledge. However, they are often ineffective when applied to NP-Complete or other high-dimensional problems. On the other hand, stochastic optimization seeks to search the space more thoroughly without being trapped in a local optimum (Chang 2007). These techniques are useful when the search space is too large and has too complicated a structure to be best tackled with a method from the gradient descent family.

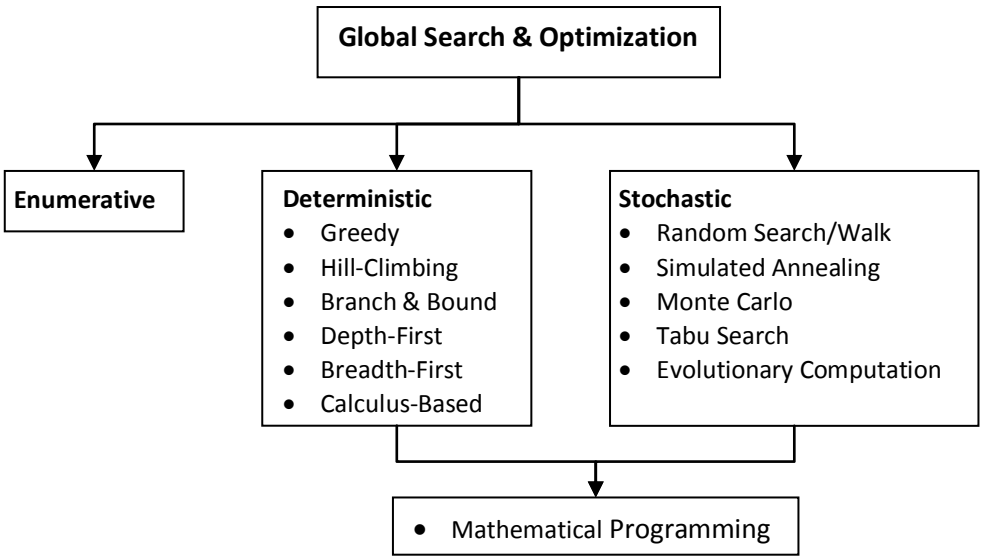


Figure 1-7: Global optimization approaches (adapted from Coello Coello, Lamont et al. 2007)

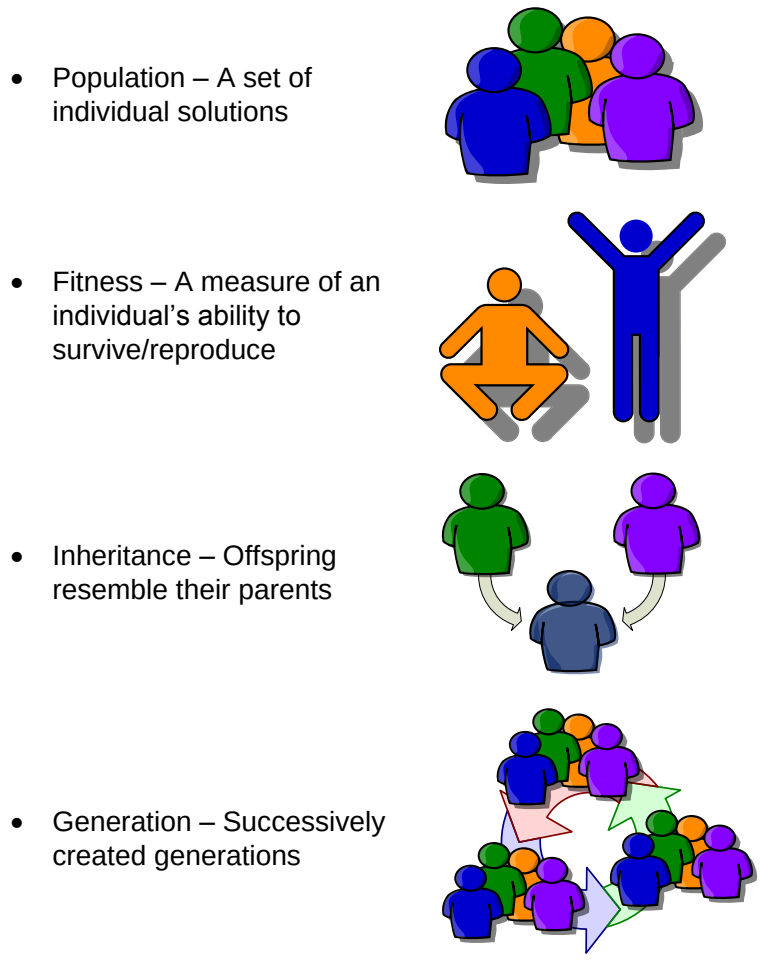


Figure 1-8: Evolutionary Computation components

EC is the most widely used stochastic technique. It is based on Darwinian evolutionary systems and includes, for example, GAs and Genetic Programming (GP). In general, EC systems will incorporate (as in Figure 1-8): one or more populations of individuals competing for limited resources; the notion of dynamically changing populations due to the birth and death of individuals; a concept of fitness which reflects the ability of an individual to survive and reproduce; and a concept of variational inheritance: offspring closely resemble their parents, but are not identical (De Jong 2006). Compared to other stochastic methods ECs have the advantage that they can be

parallelized with little effort (Rojas 1996). Since the calculations of the fitness function for each chromosome of a population are independent from each other, they can be carried out using several processors. Thus ECs are inherently parallel. ECs can be particularly effective in finding solutions where the individual pieces of the solution are important in combination, or where a sequence is important.

1.2.2.2 Genetic Algorithms (GAs)

The basic idea of a GA is to maintain a population of chromosomes, representing candidate solutions to the problem being solved. The possible solutions are generally coded as binary strings and these strings are equivalent to biological chromosomes. Other non-binary codings have proven to be useful in some applications (Damousis, Bakirtzis et al. 2004; Pendharkar and Rodger 2004; Gardner, Boilot et al. 2005; Srinivasa, Venugopal et al. 2007). Each bit of the binary string (chromosome) is referred to as a gene. A GA starts off with a population of randomly generated chromosomes and advances towards better chromosomes by applying genetic operators that are based on genetic processes occurring in nature (i.e. selection, crossover and mutation) (Mitchell 1996; Haupt and Haupt 2004).

The search is initialized with a random population of chromosomes, each representing a possible solution. Next, each chromosome in the population is decoded into a solution and its fitness is evaluated using an objective function.

During successive iterations, or *generations*, the adaptation or associated fitness of chromosomes in the population is quantified by means of fitness functions. Chromosomes for the new population are selected with a probability proportional to their fitness, related to the purpose of the application. Once the chromosomes have been selected, a crossover procedure is used to partially exchange genetic information between two parent chromosomes. Chromosomes from the parent pool are randomly paired up and are tested to determine if an exchange will take place based on a crossover probability. If an exchange is to take place, a crossover site is selected at random for the two chromosomes and the genetic material (genes) after the crossover site is then exchanged between the two parent strings. In so doing, two child chromosomes are produced, which form the members of a new population. If an exchange is not to take place (i.e. the crossover probability is less than the crossover probability parameter), then the two parents enter the new population unchanged. Mutation has the purpose of keeping the population diverse and preventing the GA from prematurely converging onto a local minimum. Each chromosome is tested on a probability basis to determine if it will be mutated. In the most commonly used form of mutation, the probability that each bit in the chromosome will be mutated is determined by the mutation probability parameter. If a bit is to be mutated, then this occurs by flipping its value (i.e. a '0' will become a '1' and vice versa). The application of the mutation operator marks the end of one GA cycle. The GA is usually allowed to run for a specified number of generations, or until

some stopping criterion is met; such as convergence of the population to a single solution.

A GA differs from many other optimisation methods by virtue of the fact that a population, or collection of possible solutions, is used rather than a single solution. It does not need knowledge of the problem domain, but it requires a fitness function to evaluate the fitness of a solution. A comprehensive description of GAs can be found in Goldberg (1989) and Holland (1992).

1.2.2.3 GAs for Feature Selection

Feature Selection (FS)is a concept used in PR that implies reduction of the input dimensionality while at the same time retaining as much as possible of their class discriminatory information. , these techniques can be classified into three broad categories: SBS, SFS, and stochastic selection (Theodoridis and Koutroumbas 2003).

SBS and SFS have been the subject of variable selection for many years (Gualdron, Llobet et al. 2006). Although the most frequently applied variable selection techniques so far are SBS and SFS, these two techniques are seldom used alone. This is because, according to Gardner, Boilot et al. (2005) and Scott, James et al. (2006), they will only explore a small fraction of the whole set of configurations and can become trapped in local minima.

Stochastic approaches, such as GAs and SAs, have been shown to be superior compared with SBS/SFS and have many successful applications (Guo and Uhrig 1992; Weller, Summers et al. 1995; Alexandridis, Patrinos et al. 2005; Gardner, Boilot et al. 2005). For example, Gardner, Boilot et al. (2005) applied a modified GA to find a good subset of sensors within an array of 32 carbon-black polymer resistors to be used in Probabilistic Neural Network (PNN) classifiers. The methods were shown to be accurate and fast at determining the sensors that should be used to discriminate bacteria. Alexandridis, Patrinos et al. (2005) presented a two-stage input selection method for RBF using a multi-objective optimization approach: in the first stage, a specially designed GA minimizes the prediction error with the aid of a monitoring dataset, while in the second stage a SA technique is used to try to reduce the number of explanatory variables. The efficiency of their method was also illustrated through its application to a number of benchmark problems.

GAs have also been used to evolve the architecture of ANNs (Guo and Uhrig 1992; Weller, Summers et al. 1995; Yao 1999; Kasabov 2001; Rivero, Dorado et al. 2009). For example, earlier work done by Guo and Uhrig (1992) has used GAs to select proper input variables for neural networks from hundreds of possible variables for nuclear power plants fault diagnosis. Work by Weller, Summers et al. (1995) used a GA to evolve the optimum set of inputs for ANNs in the prediction of nuclear reactor parameters under fault conditions. Recent work done by Kasabov (2001) employed the principle behind GAs to

dynamically adjust fuzzy neural networks' internal connections (e.g. weights). However, in these studies domain knowledge was often combined into the fitness function, and hence the method is not generic and has not been widely tested. In addition, some authors introduced ANN input deduction ratio into the fitness function. The problem with this is that higher inputs may produce less training error, and the balancing between the input deduction and ANN training error is always problem-specific.

A FS technique similar to GA is SA – although SA is not biologically based, SAs and GAs share very similar theoretical roots (Davis 1987), and it also has many successful applications (Gualdron, Llobet et al. 2006; Jansen and Wegener 2007; Llobet, Gualdron et al. 2007). However, compared to SA, GAs are population-based approaches, where there is the concept of competition (i.e. selection) between candidate solutions to a given problem. Furthermore, SAs generate new candidate points in the neighbourhood of the current point, while GAs allow the examination of points in the neighbourhood of two (or more) candidate solutions via the use of genetic operators such as crossover (De Castro 2006). Therefore, a GA tends to improve the solution consistently when given more time.

In summary, randomized FSs such as GAs are useful when the space of all possible feature subsets is prohibitively large and the choices of feature subsets are often difficult to evaluate (Liu and Motoda 2008). In addition,

these techniques necessarily depend on the ability to produce a sequence of random numbers and the sampling technique that is used (Sikora and Piramuthu 2007; Wang and Huang 2009).

1.3 Research Objectives

Each of IS techniques contributes a distinct methodology for addressing problems in its domain. This may be done in a cooperative, rather than a competitive, manner. The result is a more intelligent and robust system providing a human-interpretable, low-cost, approximate solution, as compared to traditional techniques (Mitra and Acharya 2003).

The unique contribution of this thesis is in the implementation of a hybrid IS DM technique for solving novel practical problems, the detailed description of this technique (Genetic Neural Mathematical Method, GNMM), and the illustrations of several applications solved by this novel technique.

The primary objective of this work is to design an IS system that can be applied effectively to some DM tasks such as those listed in Section 1.1.1 Procedures and Tasks:

- The system performs DDR so that computational burden is reduced;
- It also achieves high prediction/classification accuracy;
- The system is able to extract rules.

In the literature various approaches have been proposed to solve the above DM tasks. However, most of them treat these tasks separately i.e. they either just solve one task or they solve all tasks but there is no inter-connections between them. The current thesis addresses the approach that aims to accomplish all these tasks using a systematic approach, which simplifies the process in terms of applications.

The thesis also aims to explore the possibilities of applying this hybrid IS DM technique to environmental and biological applications. These two fields have attracted a lot of attention recently, which is not only because of the complexity of the problem, but also because of the massive quantities of the data that are available and increasing. However, from an environmental manager/biological scientist point of view, making sense of/from large datasets without knowing much about DM techniques remains a problem. This thesis will explore the solution of such problems using newly-proposed systematic approach.

1.4 Thesis Outline

The current chapter is a brief introduction to intelligent DM concepts and an outline of the overall structure of the thesis. In Chapter 2, we introduce some hybrid IS DM techniques, and give a detailed description of the Genetic Neural Mathematical Method (GNMM); Chapter 3 is concerned with the application of GNMM in the prediction of longitudinal dispersion coefficient; Chapter 4 is

concerned with the application to Brain-Computer Interface (BCI) data; Chapter 5 is concerned with the application to Electronic Nose (EN) data; and Chapter 6 is the application of GNMM to diabetes classification problem. Some well-studied datasets from published works/resources were used in these application chapters. In this way, the effectiveness of our DM technique can be compared with established techniques. Chapter 6 also presents benchmarking between GNMM and various hybrid IS DM techniques. And finally in Chapter 7 we present our conclusions and suggestions for future works.

References

Alexandridis, A., P. Patrinos, et al. (2005). "A two-stage evolutionary algorithm for variable selection in the development of RBF neural network models." Chemometrics and Intelligent Laboratory Systems **75**(2): 149-162.

Arbib, M. A. (2003). The handbook of brain theory and neural networks. Cambridge, Mass., MIT Press.

Berthold, M. and D. J. Hand (2003). Intelligent data analysis: an introduction. Berlin ; New York, Springer.

Berthold, M. and D. J. Hand (2007). Intelligent data analysis: an introduction. Berlin; New York, Springer.

Bigus, J. P. (1996). Data mining with neural networks: solving business problems--from application development to decision support. New York, McGraw-Hill.

Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford, Oxford University Press.

Bowden, G. J., G. C. Dandy, et al. (2005). "Input determination for neural network models in water resources applications. Part 1--background and methodology." Journal of Hydrology **301**(1-4): 75-92.

Bramer, M. A. (2007). Principles of data mining. London, Springer.

Bryson, A. E. and Y. C. Ho (1975). Applied optimal control, John Wiley & Sons.

Busque, M. and M. Parizeau (1997). "A Comparison of Fuzzy ARTMAP and Multilayer Perceptron for Handwritten Digit Recognition." Universite Laval, listopad.

Busygin, S., O. Prokopyev, et al. (2008). "Biclustering in data mining." Computers and Operations Research **35**(9): 2964-2987.

Cannas, B., A. Fanni, et al. (2006). "Data preprocessing for river flow forecasting using neural networks: Wavelet transforms and data partitioning." Time Series Analysis in Hydrology **31**(18): 1164-1171.

Carpenter, G. A., S. Grossberg, et al. (1992). "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps." IEEE Transactions on Neural Networks **3**(5): 698-713.

Carpenter, G. A., S. Grossberg, et al. (1991). "Fuzzy ART. Fast stable learning and categorization of analog patterns by an adaptive resonance system." Neural Networks **4**(6): 759-771.

Chakrabarti, S. (2009). Data mining: know it all. Burlington, MA, Elsevier/Morgan Kaufmann Publishers.

Chang, C.-I. (2007). Hyperspectral data exploitation: theory and applications. Hoboken, N.J., Wiley-Interscience.

Charaniya, S., W.-S. Hu, et al. (2008). "Mining bioprocess data: opportunities and challenges." Trends in Biotechnology **26**(12): 690-699.

Chow, T. W. S. and S.-Y. Cho (2007). Neural networks and computing: learning algorithms and applications. London; Hackensack, NJ, Imperial College Press; Distributed by World Scientific.

Coello Coello, C. A., G. B. Lamont, et al. (2007). Evolutionary algorithms for solving multi-objective problems. New York ; London, Springer.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function." Mathematics of Control, Signals, and Systems **2**: 303-314.

Damousis, I., A. Bakirtzis, et al. (2004). "A solution to the unit-commitment problem using integer-coded genetic algorithm." IEEE Transactions on Power Systems **19**(2): 1165-1172.

Davis, L. (1987). Genetic algorithms and simulated annealing. London; Los Altos, Calif., Pitman ; Morgan Kaufmann Publishers.

De Castro, L. N. (2006). Fundamentals of natural computing basic concepts, algorithms, and applications. Boca Raton, Chapman & Hall/CRC.

De Jong, K. A. (2006). Evolutionary computation: a unified approach. Cambridge, Mass., MIT Press.

del-Hoyo, R., B. Martín-del-Brío, et al. (2009). "Computational intelligence tools for next generation quality of service management." Neurocomputing **In Press, Corrected Proof**.

Du, K. L. and M. N. S. Swamy (2006). Neural networks in a softcomputing framework. London, Springer.

Elleithy, K. (2008). Innovations and advanced techniques in systems, computing sciences and software engineering. Dordrecht, Springer.

Engelbrecht, A. P. (2007). Computational intelligence: an introduction. Chichester, England; Hoboken, NJ, John Wiley & Sons.

Gardner, J. W., P. Boilot, et al. (2005). "Enhancing electronic nose performance by sensor selection using a new integer-based genetic algorithm approach." ISOEN 2003 - Selected Papers from the 10th International Symposium on Olfaction and Electronic Noses. **106**(1): 114-121.

Georgiopoulos, M., J. Huang, et al. (1994). "Properties of learning in ARTMAP." Neural Networks **7**(3): 495-506.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc.

Grivas, G. and A. Chaloulakou (2006). "Artificial neural network models for prediction of PM10 hourly concentrations, in the Greater Area of Athens, Greece." Atmospheric Environment **40**(7): 1216-1229.

Gualdron, O., E. Llobet, et al. (2006). "Coupling fast variable selection methods to neural network-based classifiers: Application to multisensor systems." Sensors and Actuators B: Chemical **114**(1): 522-529.

Guo, Z. and R. E. Uhrig (1992). Using genetic algorithms to select inputs for neural networks. 92TH0435-8), Baltimore, MD, USA, IEEE Comput. Soc. Press.

Hagan, M. T., H. B. Demuth, et al. (1996). Neural network design. Boston, PWS Pub.

Han, J. and M. Kamber (2006). Data mining: concepts and techniques. Amsterdam; London, Elsevier.

Hand, D. J., H. Mannila, et al. (2001). Principles of data mining. Cambridge, Mass., MIT Press.

Harding, S. (2008). Evolution of image filters on graphics processor units using cartesian genetic programming, Hong Kong, China, Inst. of Elec. and Elec. Eng. Computer Society.

Haupt, R. L. and S. E. Haupt (2004). Practical genetic algorithms. Hoboken, N.J., John Wiley.

Haykin, S. S. (1994). Neural networks : a comprehensive foundation. New York, Maxwell Macmillan International.

Haykin, S. S. (1999). Neural networks: a comprehensive foundation. Upper Saddle River, N.J., Prentice Hall.

Holland, J. H. (1992). Adaptation in natural and artificial systems. MA, USA, MIT Press Cambridge.

Jang, J.-S. R. (1993). "ANFIS: adaptive-network-based fuzzy inference system." IEEE Transactions on Systems, Man and Cybernetics **23**(3): 665-685.

Jang, J.-S. R., C.-T. Sun, et al. (1997). Neuro-fuzzy and soft computing : a computational approach to learning and machine intelligence. Upper Saddle River, NJ, Prentice Hall.

Jang, J.-S. R., C.-T. Sun, et al. (1997). Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence. Upper Saddle River, NJ, Prentice Hall.

Jansen, T. and I. Wegener (2007). "A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-boolean functions of unitation." Theoretical Computer Science **386**(1-2): 73-93.

Kahramanli, H. and N. Allahverdi (2009). "Rule extraction from trained adaptive neural networks using artificial immune systems." Expert Systems with Applications **36**(2, Part 1): 1513-1522.

Karray, F. O. and C. W. De Silva (2004). Soft computing and intelligent systems design : theory, tools, and applications. Harlow, England ; New York, Pearson/Addison Wesley.

Kasabov, N. (1998). "Evolving Fuzzy Neural Networks-Algorithms, Applications and Biological Motivation." Methodologies for the Conception, Design and Application of Soft Computing, World Scientific: 271-274.

Kasabov, N. (2001). "Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning." IEEE

Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **31**(6): 902-918.

Kasabov, N. (2008). "Evolving Intelligence in Humans and Machines: Integrative Evolving Connectionist Systems Approach." Computational Intelligence Magazine, IEEE **3**(3): 23-37.

Kasabov, N. K. (2007). Evolving connectionist systems: the knowledge engineering approach. London, Springer.

Kohonen, T. (2001). Self-organizing maps. Berlin ; New York, Springer.

Lin, C. T. and C. S. G. Lee (1996). Neural fuzzy systems: a neuro-fuzzy synergism to intelligent systems, London : Prentice-Hall International.

Liu, H. and H. Motoda (2008). Computational methods of feature selection. Boca Raton, Chapman & Hall/CRC.

Liu, P. and H.-X. Li (2004). Fuzzy neural network theory and application. River Edge, NJ, World Scientific.

Llobet, E., O. Gualdron, et al. (2007). "Efficient feature selection for mass spectrometry based electronic nose applications." Chemometrics and Intelligent Laboratory Systems **85**(2): 253-261.

Maier, H. R. and G. C. Dandy (2000). "Neural networks for the prediction and forecasting of water resources variables: A review of modelling issues and applications." Environmental Modelling and Software **15**(1): 101-124.

Maimon, O. (2007). Soft computing for knowledge discovery and data mining. New York, Springer.

Mantas, C. J., J. M. Puche, et al. (2006). "Extraction of similarity based fuzzy rules from artificial neural networks." International Journal of Approximate Reasoning **43**(2): 202-221.

Michalewicz, Z. (1996). Genetic algorithms + data structures = evolution programs. Berlin, Springer-Verlag.

Miller, J. F. and P. Thomson (2000). "Cartesian Genetic Programming." Lecture notes in computer science.(1802): 121-132.

Millington, I. (2006). Artificial intelligence for games. Amsterdam; Boston; Morgan Kaufmann, Elsevier.

Mitchell, M. (1996). An introduction to genetic algorithms. Cambridge, Mass., MIT Press.

Mitra, S. and T. Acharya (2003). Data mining: multimedia, soft computing, and bioinformatics. Hoboken, NJ, John Wiley.

Mitra, S., S. K. Pal, et al. (2002). "Data mining in soft computing framework: A survey." IEEE Transactions on Neural Networks **13**(1): 3-14.

Ozbakir, L., A. Baykasoglu, et al. (2009). "TACO-miner: An ant colony based algorithm for rule extraction from trained neural networks." Expert Systems with Applications **In Press, Corrected Proof**.

Pendharkar, P. and J. Rodger (2004). "An empirical study of impact of crossover operators on the performance of non-binary genetic algorithm based neural approaches for classification." Computers and Operations Research **31**(4): 481-498.

Piatetsky-Shapiro, G. and W. Frawley (1991). Knowledge discovery in databases. Menlo Park, Calif., AAAI Press: MIT Press.

Poli, R. (1997). Parallel distributed genetic programming applied to the evolution of natural language recognisers, Berlin, Germany, Springer-Verlag.

Ramadan, Z., X.-H. Song, et al. (2001). "Variable selection in classification of environmental soil samples for partial least square and neural network models." Analytica Chimica Acta **446**(1-2): 231-242.

Reeves, C. R. and J. E. Rowe (2003). Genetic algorithms : principles and perspectives : a guide to GA theory. Boston, Kluwer Academic Publishers.

Rivero, D., J. Dorado, et al. (2009). "Modifying genetic programming for artificial neural network development for data mining." Soft Computing **13**(3): 291-305.

Rojas, R. (1996). Neural networks: a systematic introduction. Berlin; New York, Springer-Verlag.

Rothlauf, F. (2006). Representations for genetic and evolutionary algorithms. Heidelberg, Springer.

Scott, S. M., D. James, et al. (2006). "Data analysis for electronic nose systems." Microchimica Acta **156**(3-4): 3-4.

Sikora, R. and S. Piramuthu (2007). "Framework for efficient feature selection in genetic algorithm based data mining." European Journal of Operational Research **180**(2): 723-737.

Sloper, J. E., G. L. Miotto, et al. (2008). "Dynamic Error Recovery in the ATLAS TDAQ System." Nuclear Science, IEEE Transactions on **55**(1): 405-410.

Soyguder, S. and H. Alli (2009). "An expert system for the humidity and temperature control in HVAC systems using ANFIS and optimization with Fuzzy Modeling Approach." Energy and Buildings **41**(8): 814-822.

Srinivasa, K. G., K. R. Venugopal, et al. (2007). "A self-adaptive migration model genetic algorithm for data mining applications." Information Sciences **177**(20): 4295-4313.

Tan, P.-N., M. Steinbach, et al. (2006). Introduction to data mining. Boston, Pearson Addison Wesley.

Tan, S. C., M. V. C. Rao, et al. (2008). "Fuzzy ARTMAP dynamic decay adjustment: An improved fuzzy ARTMAP model with a conflict resolving facility." Applied Soft Computing **8**(1): 543-554.

Taylor, B. J. and M. A. Darrah (2005). Rule extraction as a formal method for the verification and validation of neural networks, Montreal, QC, Canada, Institute of Electrical and Electronics Engineers Inc.

Theodoridis, S. and K. Koutroumbas (2003). Pattern recognition. Amsterdam ; Boston, Academic Press.

Thuraisingham, B. M. (1999). Data mining: technologies, techniques, tools, and trends. Boca Raton, CRC Press.

Venugopal, K. r. (2009). Soft computing for data mining applications. New York, Springer.

Walker, J. A. and J. F. Miller (2008). "The automatic acquisition, evolution and reuse of modules in Cartesian genetic programming." IEEE Transactions on Evolutionary Computation **12**(4): 397-417.

Wang, C.-M. and Y.-F. Huang (2009). "Evolutionary-based feature selection approaches with new criteria for data mining: A case study of credit approval data." Expert Systems with Applications **36**(3, Part 2): 5900-5908.

Wang, L. and X. Fu (2005). Data mining with computational intelligence. Berlin; New York, Springer.

Weller, P. R., R. Summers, et al. (1995). Using a genetic algorithm to evolve an optimum input set for a predictive neural network. Proceedings of 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, Sheffield, UK, IEE.

Wilson, G. and W. Banzhaf (2008). A comparison of cartesian genetic programming and linear genetic programming, Naples, Italy, Springer Verlag.

Witten, I. H. and E. Frank (2005). Data mining: practical machine learning tools and techniques. Amsterdam; Boston, MA, Morgan Kaufman.

Xu, Z., J. Xuan, et al. (2009). "Application of a modified fuzzy ARTMAP with feature-weight learning for the fault diagnosis of bearing." Expert Systems with Applications **36**(6): 9961-9968.

Yang, Q. and X. Wu (2006). "10 challenging problems in data mining resesarch." International Journal of Information Technology and Decision Making **5**(4): 597-604.

Yao, X. (1999). "Evolving artificial neural networks." Proceedings of the IEEE **87**(9): 1423-1447.

Ye, N. (2003). The handbook of data mining. Mahwah, N.J., Lawrence Erlbaum Associates, Publishers.

Chapter 2 Hybrid Intelligent System Data Mining Techniques and the Genetic Neural Mathematical Method

2.1 Introduction

Chapter 1 provides some theoretical background to IS DM techniques such as ANNs and GAs, and outlines the structure of the thesis. . This chapter introduces some hybrid IS DM techniques, which will be used for benchmarking studies in the thesis, and the Genetic Neural Mathematical Method (hereafter called GNMM). GNMM is a pattern classifier and analyser based on GAs and MLPs. It inherits the advantages (e.g. robustness and nonlinearity) of ANN. Furthermore, it also incorporates a GA and mathematical programming to achieve input selection and rule extraction. By utilizing the GA, GNMM is able to automatically optimise the number of inputs to the MLP, which serves as the core DM engine. Employing a mathematical programming method, GNMM is also capable of identifying regression rules extracted from the trained MLP.

2.2 Hybrid Intelligent System Data Mining Techniques

Each IS technique addresses problems in its domain. However, hybridizations of IS techniques typically enjoy the generic and application-specific merits of the individual SC tools that they integrate (Mitra and Acharya 2003). DM functions modelled by such hybrid systems include rule extraction, data compression, clustering, incorporation of domain knowledge, and partitioning. Although conventional ANNs are one of the key technologies used for DM, the influences of these hybrid systems on the ANN field have shown great potential. Let us now consider some in turn:

2.2.1 Adaptive Neuro-Fuzzy Inference System (ANFIS)

Since the invention of the Adaptive Neuro-Fuzzy Inference System (ANFIS) in 1993 (Jang), it has become a standard technique that has been widely used in many applications (Lin and Lee 1996; Jang, Sun et al. 1997). It uses a hybrid-learning algorithm to identify parameters for Sugeno-type fuzzy inference systems. It applies a combination of the least-squares method and the gradient descent method for training membership function (MF) parameters to emulate a given training dataset (Karray and De Silva 2004; Soyguder and Alli 2009).

ANFIS is a multilayer feed forward network where each node performs a particular function on incoming signals. It is normally represented by a six-layer feedforward neural network as shown in Figure 2-1. To perform a desired

input-output mapping, adaptive learning parameters are updated based on gradient learning rules (Jang ; Soyguder and Alli 2009). Both square and circle node symbols in Figure 2-1 are used to represents different properties of adaptive learning, among which the rule layer represents a set of fuzzy rules. The ANFIS model is one of the implementation of a first order Sugeno fuzzy inference system, and the rules are of the form:

- *IF* x_1 is $\mathbf{A}_1$ *AND* x_2 is $\mathbf{A}_2$, *THEN* $y = px_1 + qx_2 + r$

where x_1 and x_2 are inputs corresponding to the $\mathbf{A}_1$ and $\mathbf{A}_2$ term set, y is output, p, q, and r are constants.

2.2.2 Evolving Fuzzy Neural Network (EFuNN)

The Evolving Fuzzy Neural Network (EFuNN) proposed by Kasabov (1998; 2007; 2008) implements a strategy of dynamically growing and pruning the connectionist (i.e. ANN) architecture and parameter values. It consists of five layers (Figure 2-2): the input layer only represents the input variables; the second layer of nodes (fuzzy input neurons or fuzzy inputs) represents the fuzzyfication of each variable of the input space. These nodes can use Gaussian, triangular or other MFs; we have used triangular ones in order to reduce the computing complexity. The third layer is made up of rule nodes, evolving through time in a supervised way. The fourth layer represents the rule weights. And finally the last layer implements the output variable, providing the system

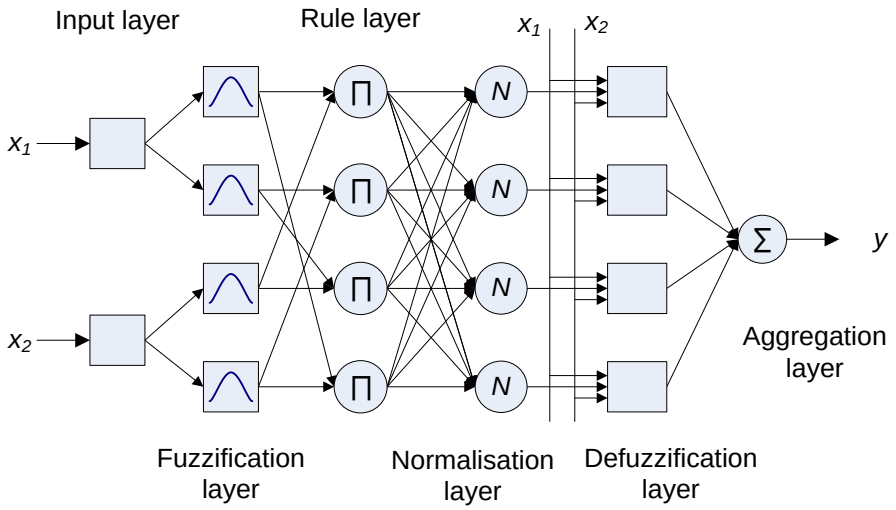


Figure 2-1: Adaptive Neuro-Fuzzy Inference System

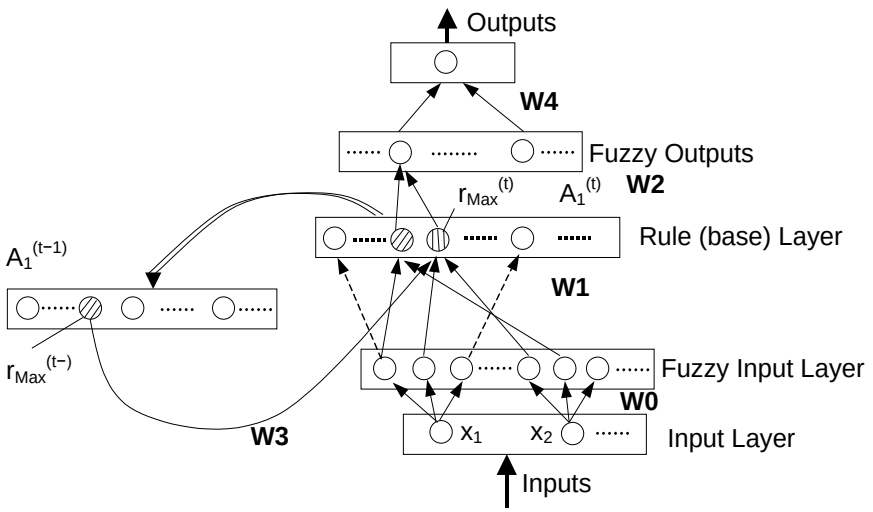


Figure 2-2: Architecture of Evolving Fuzzy Neural Network (adapted from Kasabov 2007)

output (Kasabov 1998; del-Hoyo, Martín-del-Brío et al. 2009). EFuNN is implemented in the NeuCom package[1] developed at Auckland University of Technology.

[1] The NeuCom Project, http://www.aut.ac.nz/research/research-institutes/kedri/research-centres/centre-for-data-mining-and-decision-support-systems/neucom-project-home-page

EFuNN learns by associating (learning) new data points (vectors) to a rule node r_j: the centres of this node's hyperspheres (i.e. $\mathbf{W1}(r_j)$ and $\mathbf{W2}(r_j)$) adjust in the fuzzy input space depending on the distance between the new input vector and the rule node through a learning rate l_j , and in the fuzzy output space depending on the output error through the Widrow-Hoff least mean square delta algorithm:

$$\mathrm{W1}\left(r_j^{(t+1)}\right) = \mathrm{W1}\left(r_j^{(t)}\right) + l_j.(x_f - \mathrm{W1}(r_j^{(t)})) \tag{2.1}$$

$$\mathrm{W2}\left(r_j^{(t+1)}\right) = \mathrm{W2}\left(r_j^{(t)}\right) + l_j.(y_f - \mathrm{A2}).\mathrm{A1}\left(r_j^{(t)}\right) \tag{2.2}$$

where x_f and y_f are fuzzy input and output vectors respectively; $\mathrm{A2} = f_2(\mathrm{W2.A1})$ is the activation vector of the fuzzy output neurons in the EFuNN structure when x is presented; $\mathrm{A1}\left(r_j^{(t)}\right) = f_2(D(\mathrm{W1}\left(r_j^{(t)}\right), x_f)$ is the activation of the rule node $r_j^{(t)}$. In other words, both weight vectors are iteratively adjusted – $\mathbf{W1}$ through unsupervised training based on a similarity measure and $\mathbf{W2}$ through supervised learning based on output error.

Furthermore, EFuNN allows for the construction of fuzzy rules from the network weights, and hence knowledge extraction. Similar to ANFIS, there is a rule layer in EFuNN to represent fuzzy rules. Thus, once the training is finished, fuzzy rules can be extracted from the system.

2.2.3 Fuzzy ARTMAP

Fuzzy ARTMAP is a competitive learning model based on the Adaptive Resonance Theory (ART). It is an extension of ART1 (for binary inputs) and ART2 (for continuous inputs) for fuzzy inputs (Carpenter, Grossberg et al. 1991; Carpenter, Grossberg et al. 1992; Georgiopoulos, Huang et al. 1994). Fuzzy ARTMAP consists of two ART modules, i.e. ART_a and ART_b, and an inter-ART map field F^{ab}, as in Figure 2-3. Both ART_a and ART_b are fuzzy ARTs (i.e. accepting fuzzy inputs), each of which is comprised of three layers: normalization layer F_0, input layer F_1 and recognition layer F_2. The main purpose of the map field F^{ab} is to classify a fuzzy pattern into the given class, or re-start the matching procedure (Liu and Li 2004).

Fuzzy ARTMAP implements supervised learning and processes fuzzy information and transforms it in terms of hyper-rectangles. Learning in Fuzzy ARTMAP encompasses the recruitment of new hyper-rectangular prototypes and expansion of the boundary of existing prototypes in the feature space.

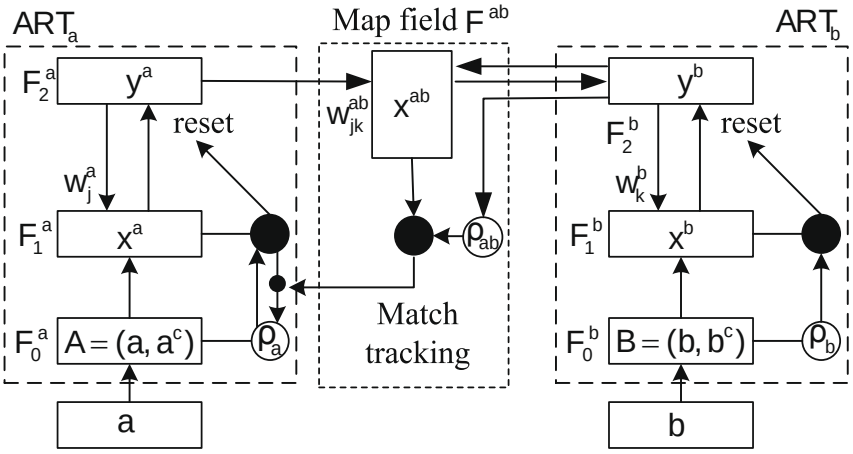


Figure 2-3: Architecture of Fuzzy ARTMAP (adapted from Xu, Xuan et al. 2009)

Like other incremental ANNs, the Fuzzy ARTMAP growth criterion is subject to a similarity measure between the input pattern and the prototypes stored in the network (Busque and Parizeau 1997; Tan, Rao et al. 2008). The Matlab package implementation of Fuzzy ARTMAP is available from the lab led by Carpenter[2] (1992).

2.2.4 Cartesian Genetic Programming (CGP)

Cartesian Genetic Programming (CGP) was originally developed by Miller and Thomson (2000) for the purpose of evolving digital circuits. CGP represents a program using a directed indexed graph as opposed to the tree representation normally used in conventional GP. The genotype is a fixed length representation consisting of a list of integers which encode the function and connections of each node in the directed graph. However, CGP uses a genotype-phenotype mapping that does not require all of the nodes to be connected to each other. As a result, the phenotype is bounded but has variable length. This allows areas of the genotype to be inactive and have no influence on the phenotype, leading to a neutral effect on genotype fitness called neutrality. An example of a CGP genotype and the corresponding phenotype that arose in the evolution of a 2-bit parallel multiplier is shown in Figure 2-4.

[2] CNS Tech Lab, Boston University, http://techlab.bu.edu/resources/software

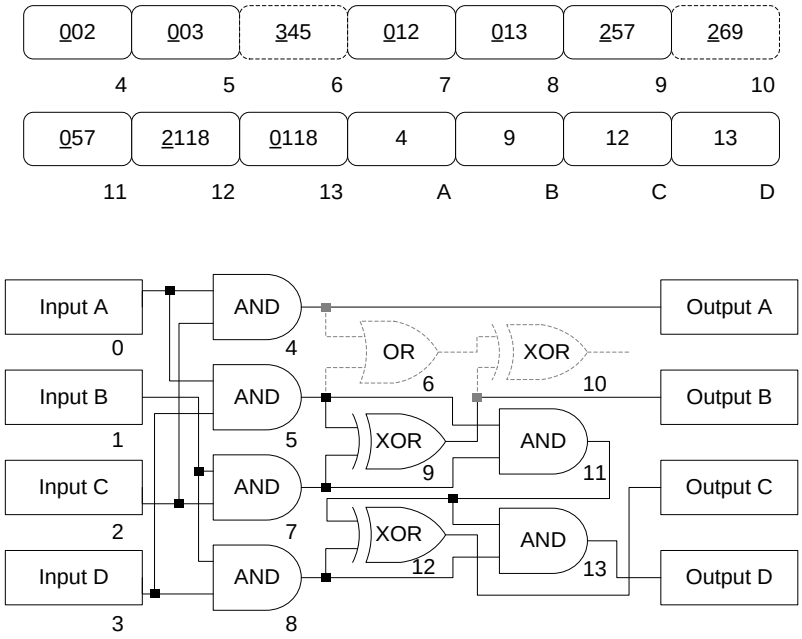


Figure 2-4: A possible CGP genotype and corresponding phenotype for a 2-bit parallel multiplier circuit (adapted from Walker and Miller 2008)

A benefit of CGP is that it allows the implicit reuse of nodes, as a node can be connected to the output of any previous node in the graph, thereby allowing the repeated reuse of sub-graphs. This is an advantage over tree-based GP representations where identical sub-trees have to be constructed independently (Walker and Miller 2008). The CGP technique has some similarities with Parallel Distributed Genetic Programming (PDGP) (Poli 1997). PDGP directly represents the graphs using a two-dimensional grid topology, in which each row of the grid is executed in parallel in the direction of data flow, with the program output being taken from the final row of the grid. This allows the formation of efficient programs by reusing partial results. Originally, CGP also used a program topology defined by a rectangular grid of nodes with a user defined number of rows and columns. However, later work on CGP showed that it was more effective when the number of rows is chosen to be

one (Harding 2008; Wilson and Banzhaf 2008). This one-dimensional topology is used throughout the thesis.

Due to its GP nature, rule extraction in CGP is straight forward – as the whole program is evolving arithmetic operators, the set of operators minimizing the training error can thus be used to present arithmetic rules. The CGP used here was implemented using the package developed by Sloper, Miotto et al. (2008). Compared with other hybrid IS DM techniques described above, CGP is solely based on EC. However, due to its built-in capability for rule extraction, it is also included in the current section and will also be used for benchmarking purposes.

2.3 The Genetic Neural Mathematical Method (GNMM)

The hybrid IS DM techniques described above have all been successfully applied to various problems. Although these techniques are used for benchmarking in the current context, it is not within the scope of the thesis to discuss advantages/disadvantages of a particular method over another. Due to their merits of ANNs and GAs as shown in Chapter 1, a novel way of combining these two techniques and at the same time overcoming their disadvantages is proposed (i.e. the GNMM method). Overall, GNMM is implemented in three steps:

(1) In the first step, GAs are used to evolve an optimal set of MLP inputs. The SGA technique is used in GNMM with binary chromosome codings, which represent the presence of an input as '1' and absence as '0'. Within SGA operators, the roulette wheel selection, single point crossover and mutation are used. GA parameters have to be selected carefully as these potentially have a great impact on performance and results. GNMM also utilizes an adaptive method to adjust the mutation rate based on the average fitness of successive generations. In addition, GNMM uses the elite group and appearance percentage to minimize the randomness, which is a problem associated with all stochastic optimizations.

(2) MLPs are used in GNMM both as the fitness function and the core DM engine – input variables found by GAs in the previous step are redirected into an MLP to perform the final modelling. The MLPs' pre-processing includes projecting the input data onto a small range so that training is more efficient. K-fold cross-validation is also used to avoid over fitting. In order to accelerate the training process, an ICA based weight initialization algorithm is used to determine optimal weights before the commencement of any training algorithms. The LM algorithm is used to achieve a second-order speedup compared to conventional BP training.

(3) In the third step, GNMM utilizes a mathematical programming based method to extract regression rules from trained MLPs. The method is not only used to identify the premises of multivariate polynomial rules, but also to explore features from the extracted rules based on data samples

associated with each rule. Therefore, the methodology can provide regression rules and features not only in the polyhedrons with data instances, but also in the polyhedrons without data instances.

The GNMM algorithm and interactions between GNMM components are illustrated in Figure 2-5. In the following sections, detailed descriptions of

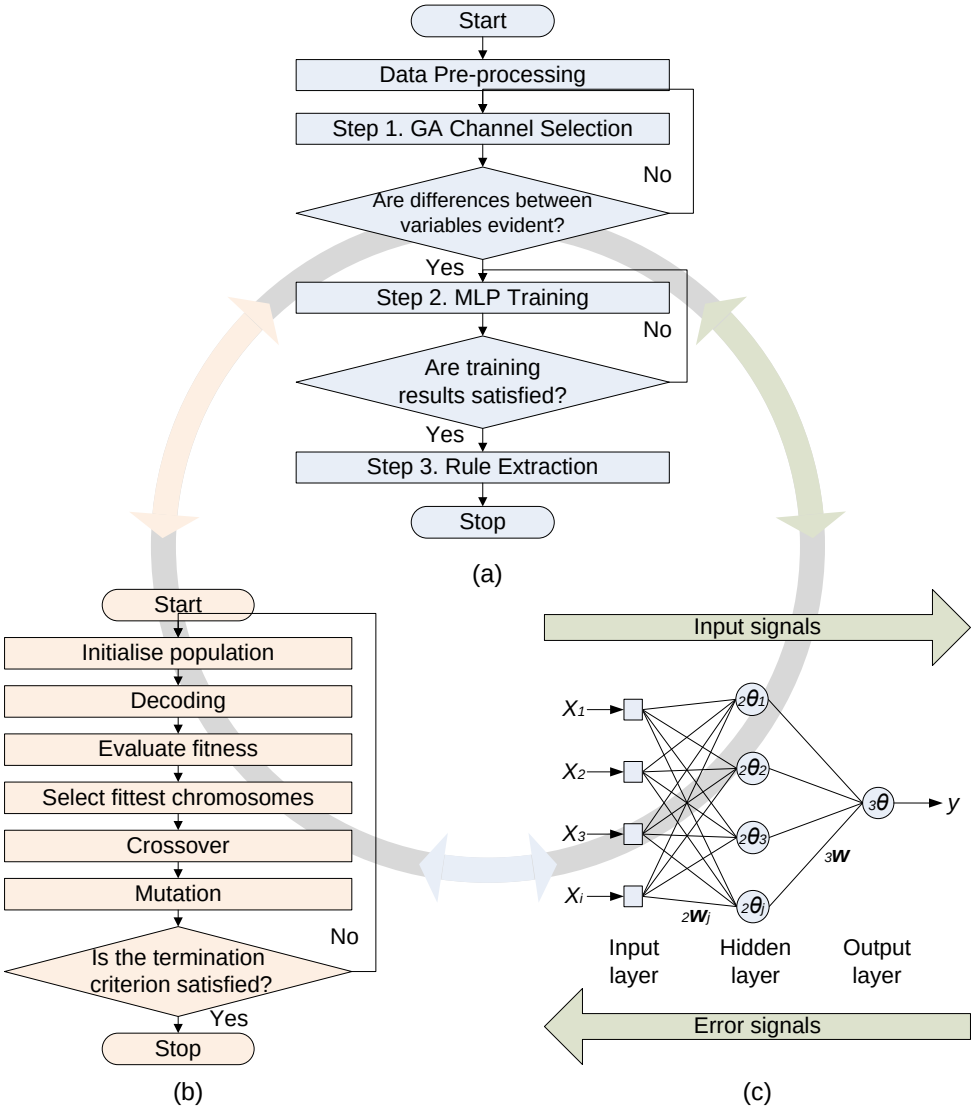


Figure 2-5: Interactions between GNMM components. (a) The GNMM algorithm; (b) A Simple Genetic Algorithm; (c) A three-layer MLP

these three steps will be presented, which is followed by a summary of the whole chapter.

2.3.1 Step 1 – Genetic Algorithm for Input Optimization

The GA technique used in GNMM is often referred to as the Simple Genetic Algorithm (SGA) (Holland 1975; Vose 1999; Reeves and Rowe 2003). After first introduced and investigated by Holland (1975), SGA has numerous variants (Bäck, Fogel et al. 1997; Chambers 2001). In general, a SGA exhibits the following features: finite population, bit representation, one-point crossover, bit-flip mutation and roulette wheel selection. However, alternative genetic operators have been introduced in the literature to alter the behaviour of GA, such as tournament selection, uniform crossover, and inorder mutation.Specifically designed GAs can be obtained by using different combinations of genetia operators, and are suitable for a particular range of problems. For the purpose of the current thesis, SGA is used due to its suitability to a wide range of problems and its solid theoretical basis.

2.3.1.1 Procedures

The general processes of SGA are shown in Figure 2-5 (a). SGA begins by generating initial population, and it ends, like any other optimization algorithm, by testing for stopping criteria, e.g. convergence. In between, SGA will decode each chromosome (i.e. solution) and evaluate their fitness for the problem under investigation. Based on these fitness values, these chromosomes will be

processed by genetic operators (i.e. selection, crossover and mutation) in order to produce the next generation.

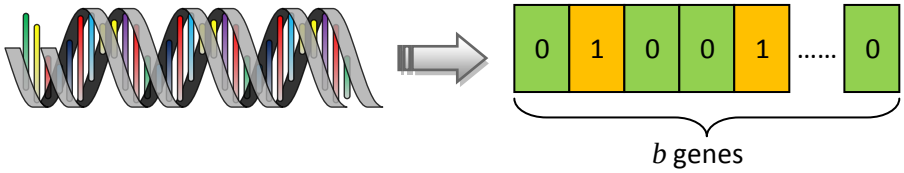
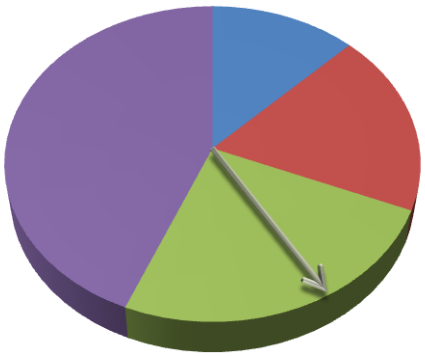
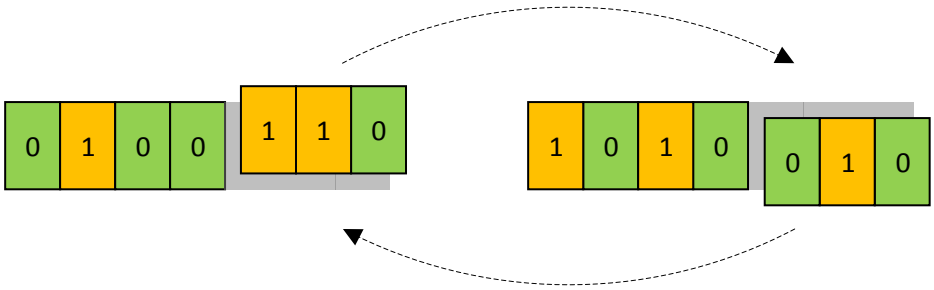


Figure 2-6: Binary coding chromosome



(a) Selection



(b) Crossover

(c) Mutation

Figure 2-7: Genetic operators

Let us assume there are two datasets $\mathbf{X}$ = {$x_{(1,1)}$,..., $x_{(a,b)}$} and $\mathbf{Y}$ = {y_1 ,..., y_a}, where $\mathbf{X}$ is the input data, $\mathbf{Y}$ is the dataset to be modelled/predicted (assuming $\mathbf{Y}$ is a one-dimension dataset for simplicity), a is the number of data samples and b denotes possible ANN inputs (i.e. available variables). GNMM starts off by randomly generating an initial population of chromosomes of size N_p. In nature, chromosomes are structures of compact intertwined molecules of DNA (Figure 2-6) (Engelbrecht 2007). In the context of GNMM, each chromosome represents a candidate solution to the input selection problem. A chromosome consists of *b* genes, each representing an input variable. The encoding of a gene is binary, meaning that a particular variable is considered as an input variable (represented by '1') or not (represented by '0'), as shown in Figure 2-6.

The GA fitness function is always problem-specific. For GNMM, the assessment of the fitness of a chromosome is the MSE when a three-layer MLP is being trained with the input variable subset $\mathbf{X}_i$ and output target $\mathbf{Y}$ for a certain number of epochs N_e. Provided that there are enough hidden neurons in the network and that a sufficient amount of data is available, MLPs can approximate virtually any function with any desired accuracy (Cybenko 1989). However, in the current stage the number of neurons in the hidden layer is set to a small fixed number. The reason for doing this is that the purpose of the current stage is only to explore the effectiveness of different input parameter combinations; such settings simplify the GA implementation and reduce the

computational burden.

Based on their fitness values, chromosomes will be processed by genetic operators such as selection, crossover, and mutation, as shown in Figure 2-7, to form the next generation; until a stopping criterion is met. Selection is one of the main operators in GAs, and relates directly to the Darwinian concept of survival of the fittest. The selection operator used in GNMM is roulette wheel selection; also sometimes referred to as proportional selection (Figure 2-7 (a)). In roulette wheel selection, a probability distribution proportional to the fitness is created, and the higher the fitness value, the more chance a chromosome has to be selected. Figure 2-7 (b) and (c) depict the crossover and mutation operators, in which two chromosomes exchange part of their genes or a random gene flips to its other possible value.

2.3.1.2 Parameters

Stopping criteria in GA include, like other optimization techniques, e.g. convergence and a set of pre-defined parameters such as N_p, N_e, generation size N_g, crossover probability p_c and mutation probability p_m. In fact, selecting GA parameters is very difficult due to the many possible combinations in the algorithm. In addition, a GA relies on random number generators for creating the population, selection, crossover and mutation. A different random number seed produces different results. As such, selecting GA parameters is always problem-specific.

Generally speaking, large population sizes are used to allow thorough exploration of complicated fitness surfaces. Crossover is then the operator of choice to exploit promising regions of fitness space by combining information from promising solutions. Mutation in the less critical genes may result in further exploitation of the current region.

Schaffer, Caruana et al. (1989) have reported results on optimum parameter settings for SGA. Their approach used the five cost functions in the De Jong's test function suite (De Jong 1975; Haupt and Haupt 2004). They used discrete sets of parameter values N_p = 10, 20, 30, 50, 100, 200; p_m = 0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.10; p_c = 0.05 to 0.95 in increments of 0.10; and 1 or 2 crossover points, which means that there were a total of 8400 possible combinations. Each combination was averaged over 10 independent runs. These authors found the best performance resulted for the following parameter settings: N_p = 20 to 30, p_c = 0.75 to 0.95, p_m = 0.005 to 0.01. Generally, parameter settings for GNMM will follow this range except that under some circumstances, for instance, when possible ANN inputs b is very large, we will have to increase the population size N_p accordingly. The issue of optimal parameters will be addressed in Section 6.5 GA Parameter.

Based on the above initial value ranges, GNMM also incorporates an adaptive mutation rate (Reeves and Rowe 2003; Yuen and Chow 2009). The algorithm for updating the mutation rate is depicted in Figure 2-8. In summary, when the

```
1   //Initial mutation rate
    p_m = 0.005
2   //Compute average fitness of first generation
    f̂(1)
3   //Iterate through the rest of generations
    FOR t = 2:N_p
        // Compute average fitness of the nth generation
        f̂(t)
        // Switch depending on whether average fitness increases
        IF f̂(t) / f̂(t−1) ≤ 0.1
            p_m = p_m × 0.1
        ELSE
            p_m = p_m × (log_10(f̂(t) / f̂(t−1)) + 1)
        END IF
    END FOR
```

Figure 2-8: Adaptive mutation rate

population has higher fitness (i.e. lower MSE), the mutation rate reduces to encourage exploitation of what has been found. Conversely, when we have a lower fitness, we increase the mutation rate to try to force further exploration. In this way, the GA optimisation process is realised by altering the mutation rate.

It should be noted that these parameters interact with each other so as to affect the behaviour of GNMM in complex, nonlinear ways. This means that no one particular choice for these parameter values is likely to be universally optimal. For a detailed discussion of GA parameters, please refer to Reeves and Rowe (2003) and De Jong (2006).

2.3.1.3 The Elite Group and Appearance Percentage

As a stochastic algorithm, randomness plays an essential role in GAs – all genetic operators need random procedures. As a result, two runs (i.e. an entire set of generations) with different random−number seeds will generally produce different detailed behaviour. Furthermore, in GNMM the fitness function is the training error of an MLP trained with the selected input variables and target outputs. The MLP parameters (e.g. weights and thresholds) are initialized randomly as well, which adds another level of uncertainty to the optimization problem. In GNMM, the randomness problem can be addressed by applying two techniques: one is introducing an elite group into GAs (Haupt and Haupt 2004); the other is what we call the *appearance percentage* (Yang, Hines et al. 2008).

The elite group is a collection of chromosomes that performed best and were made exempt from crossover and mutation and are retained in the next generation. Introducing the elite group into GAs strengthens the ability to search, which can be explained as exploitational with respect to high yielding regions and explorative with respect to other regions.

Another characteristic of GNMM that introduces randomness is that it uses MLPs' training error as the fitness function. As a result, even if the same winning chromosomes are found in successive generations, they may not yield the same performance within a certain number of epochs. In this case the

fitness value is not the only criteria for the evaluation of a schema. Thus, GNMM introduces the concept of *appearance percentage* (Yang, Hines et al. 2008).

GA researchers often report statistics, such as the best fitness found in a run and the generation at which the individual with that best fitness was discovered, averaged over many different runs of the GA on the same problem (Mitchell 1996). In GNMM, the averaging is extended to not only calculate different runs, but also different generations within the same run. A gene's (i.e. possible ANN inputs) *appearance percentage* is defined as a gene's accumulated appearance in the winning chromosome of each generation divided by the total number of generations. For example, if the 2nd gene appeared twice in the winning chromosome in a total of 20 generations, then the *appearance percentage* for this gene is 10%. In this way, although the GA's search evolves successively towards better generations, each generation is treated separately. Hence, the uncertainty associated with the randomness of the fitness function is minimized. Due to the fact that in GNMM the coding is binary, correspondence between genes and input variables can be easily found.

As a result of the input selection procedure, the input variables which occur most frequently throughout all the populations can therefore be identified. The final subset formed by these variables (denoted by $\mathbf{X}_f$) is the subset that produces the minimal error within a given number of epochs.

2.3.2 Step 2 – Multi-Layer Perceptron Modelling

In GNMM, MLPs serve both as the fitness function in the input optimization process and as the core DM engine. However, in both roles GNMM utilizes some different techniques compared to conventional MLPs such as the Independent Component Analysis (ICA)-based weight initialization algorithm and the Levenberg-Marquardt (LM) algorithm.

2.3.2.1 Pre-processing

Pre-processing includes scaling $\mathbf{X}_f$ and $\mathbf{Y}$ into the range [−1, 1] before passing them into the ANN to make the MLP training more efficient. For example, consider x_n to be an element of the n-th column vector ($\mathbf{x}_n$) in $\mathbf{X}_f$, the mapping is carried out as follows:

$$x_n^{'} = \frac{2 \times (x_n - \mathbf{x}_{min})}{(\mathbf{x}_{max} - \mathbf{x}_{min})} - 1 \tag{2.3}$$

where x_n′ denotes the mapped value, $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ are the minimum and maximum values in $\mathbf{x}_n$. After the ANN has been trained, the settings from Eq.(2.3) are used to transform any future inputs that are applied to the network. Thus, $\mathbf{x}_{min}$ and $\mathbf{x}_{max}$ effectively become a part of the network, just like the network weights and biases.

GNMM also utilizes a K-fold cross-validation technique to define the training and validation data. Each time a small randomly selected portion of $\mathbf{X}$ and $\mathbf{Y}$

(e.g. 10%×a) is set aside for validation before any training in order to avoid over-fitting (Lin and Lee 1996), and the rest are used for the training. As a consequence of cross-validation, the MLP does not necessarily reach its final epoch N_e.

2.3.2.2 Weight Initialization

The weight initialization of ANN plays a significant role in the convergence of a training method. It is common practice to initialize MLP weights and thresholds with small random values. However, this method is ineffective because of the lack of prior information on the mapping function between the input and output data samples (Du and Swamy 2006). There are several approaches (Yam and Chow 2000; Yam, Leung et al. 2002; Chow and Cho 2007) to estimate optimal values for the initial weights so that the number of training iterations is reduced. GNMM utilizes the ICA-based weight initialization algorithm proposed by Yam, Leung et al. (2002). The algorithm is able to initialize the hidden layer weights that extract the salient feature components from the input data. The initial output layer weights are evaluated in such a way that the output neurons are kept inside the active region.

ICA is a statistical and computational technique for revealing hidden factors that underlie sets of random variables, measurements, or signals (Hyvarinen, Karhunen et al. 2001). Suppose $\mathbf{M}$ is a zero-mean random variable that can be

observed (i.e. mixed signals), and $\mathbf{S}$ is its linear transform (i.e. source signals). Then the ICA problem is to determine a constant matrix $\mathbf{A}$

$$\boldsymbol{S} = \boldsymbol{AM} \tag{2.4}$$

so that components of the linearly transformed signals $\mathbf{S}$ are statistically independent from each other. The statistically independence is defined such that the joint probability density of $\mathbf{S}$ equals the product of the marginal densities of the individual components.

Yam, Leung et al.'s weight initialization approach utilizes the FastICA algorithm[3] (Hyvarinen 1999) to perform the actual calculations, which is summarized below:

- First, the mixture signals $\mathbf{M}$ are whitened

$$\boldsymbol{U} = \boldsymbol{V}^T\boldsymbol{M} \tag{2.5}$$

where $\mathbf{V}^{\mathrm{T}}$ is a whitening matrix, $\mathbf{U}$ denotes whitened signals and $E(\mathbf{U}\mathbf{U}^{\mathrm{T}})=\mathbf{I}$, which means that components of the whitened signals are uncorrelated and their variances equal to unity. E is the expectation operator and $\mathbf{I}$ is the identity matrix. The whitening matrix is computed

[3] Laboratory of Computer and Information Science, the Helsinki University of Technology, http://www.cis.hut.fi/projects/ica/fastica/.

using the Singular Value Decomposition (SVD) of the covariance matrix $E(\mathbf{X}\mathbf{X}^{\mathrm{T}})$. Thus, the separating matrix $\mathbf{A}$ is factorized by

$$\boldsymbol{A} = \boldsymbol{B}^{T}\boldsymbol{V}^{T} \tag{2.6}$$

where $\mathbf{B}$ is the orthogonal separating matrix.

- Next, matrix $\mathbf{B}$ is initialized randomly, and the whitened signal $\mathbf{U}$ will be used to iterate through the following steps to reach for convergence, which is defined as when the old and new values of $\mathbf{B}$ point in the same direction

(1) Let $\mathbf{B}^{+} = E\{\mathbf{U}\mathrm{g}(\mathbf{B}^{\mathrm{T}}\mathbf{U})\} - E\{\mathrm{g}'(\mathbf{B}^{\mathrm{T}}\mathbf{U})\}\mathbf{B}$;

(2) $\mathbf{B} = \mathbf{B}^{+}/\|\mathbf{B}^{+}\|$;

(3) Convergence test.

where g and g' are respectively a nonlinear function and its derivative.

To put it simply, Yam, Leung et al.'s weight initialization method computes the seperating matrix $\mathbf{A}$ from the ANN's input data using the above FastICA algorithm. And the optimal initial weights $\mathbf{W}ini$ and thresholds $\boldsymbol{\theta}ini$ are determined as

$$\boldsymbol{W}_{ini} = \delta\mathbf{A} \tag{2.7}$$

$$\boldsymbol{\theta_{ini}} = -\delta \boldsymbol{A} < X > \quad (2.8)$$

where δ is the scaling factor to keep the output of hidden neurons in the active region, and <**X**> is the mean vector of the input data. GNMM uses the hyperbolic tangent function in the hidden neurons and linear function in the output neurons. The active region is assumed to be the region in which the derivative of the hyperbolic tangent function is greater than 50% of its maximum derivative (i.e. maximum inputs to hidden neurons no greater than 0.8814).

It has been shown that Yam, Leung et al.'s weight initialization method is capable of speeding up the MLP learning process effectively (Yam, Leung et al. 2002). However, it should be noted that in GNMM the method is only applied to input-hidden connections as the output neurons use the linear activation function and random weights are used.

2.3.2.3 Choice of Activation Function

Generally speaking, the activation or squashing function is usually a nonlinear function that suppresses the range of the output of the neuron to a range of values (Zhang 2009). The purpose of the activation function is to introduce nonlinearity into the network and limit the output value of each neuron so that the behaviour of ANNs is not affected by extreme values produced by divergent neurons (Wang 2003). Transfer functions are applied to process the

weighted and biased inputs, among which five basic and widely adapted activation functions are illustrated in Figure 2-9.

The most commonly used activation function is the sigmoid function or logistic function (see Figure 2-9) (Bourg and Seemann 2004). It transforms the input,

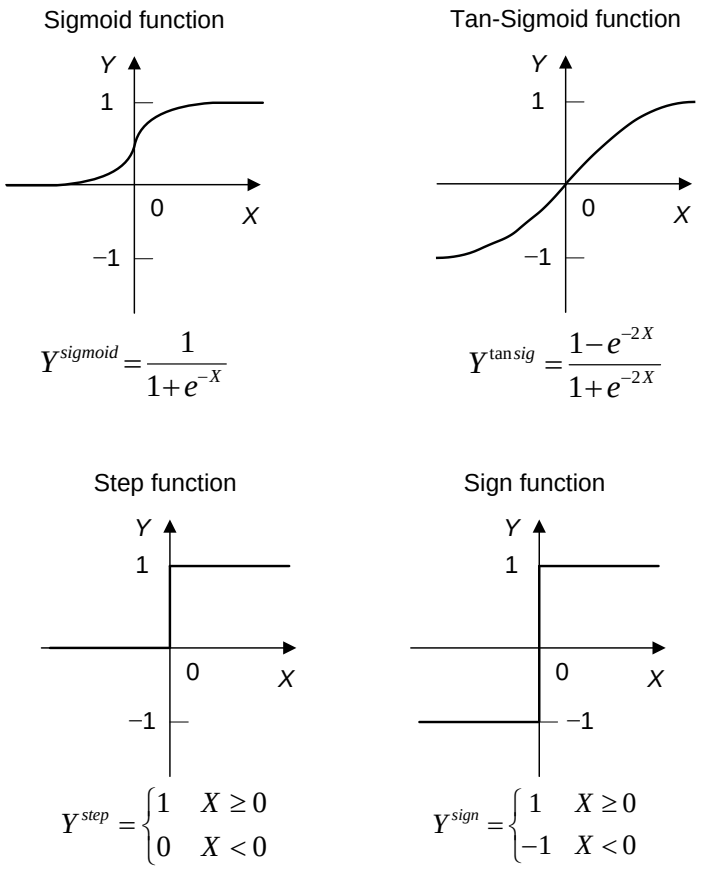


Figure 2-9: Sample activation functions

which can have any value between plus and minus infinity, into a reasonable value in the range between 0 and 1. It is also important to note that no matter how large (positive or negative) the input gets, the sigmoid function will never actually reach 0 or 1; it asymptotes to these values. The hyperbolic tangent function (i.e. tan-sigmoid) behaves in a similar way apart from the fact that it speeds up training (Bourg and Seemann 2004; Yu 2007).

Taking a closer look at the sigmoid and hyperbolic tangent transfer functions, it can be seen that when the weighted sum of all the inputs is near 0, then these functions are a close approximation of a linear function. As the magnitude of the weighted sum gets larger, these transfer functions gradually saturate. This behaviour corresponds to a gradual movement from a linear model of the input to a nonlinear model. In short, they have the ability to do a good job of modelling on three types of problems: linear problems, near-linear problems, and nonlinear problems (Berry and Linoff 2004). Due to these properties, the hyperbolic tangent function

$$Y^{tansig} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{2}{1 + e^{-2x}} - 1 \tag{2.9}$$

is chosen to be the activation function in the MLP's hidden layer.

The step and sign activation functions, also called hard limit functions, are often used in decision-making neurons for classification and PR tasks

(Negnevitsky 2005). The linear activation function means that the output of a neuron is simply the net input, which implies that input-output rule extraction is not necessary. Employing such a linear output neuron is useful when the output does not need to be confined to an interval between 0 and 1. Thus, it is used as the activation function for output neurons in GNMM.

2.3.2.4 Training Algorithm

In the standard BP, learning iterations (i.e. epochs) consists of two phases: in the feedforward pass, the actual output values of the network for each training pattern are calculated; in the backward propagation, any error signal is propagated back from the output layer toward the input layer. Weights are then adjusted as functions of the error signal.

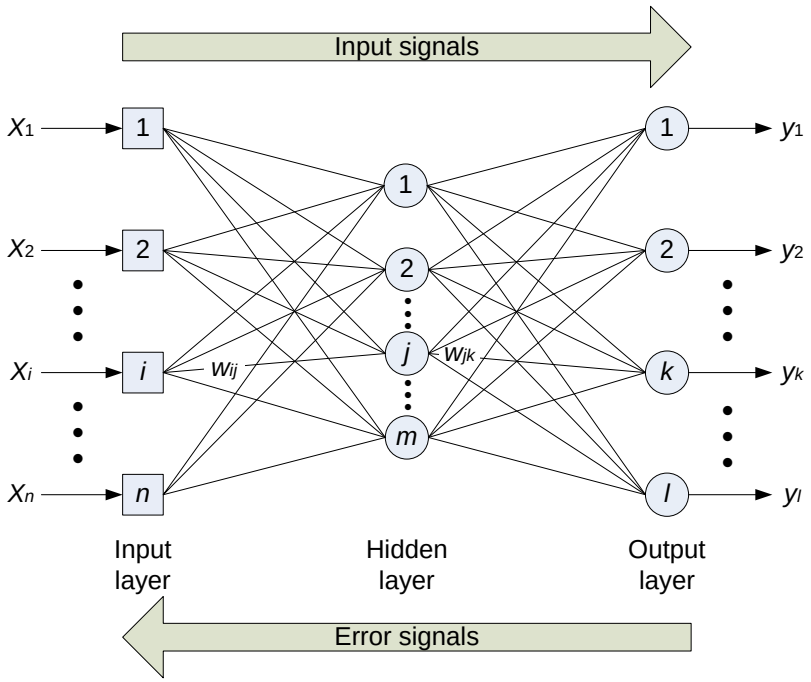


Figure 2-10: The Back-Propagation training algorithm

The MLP illustrated in Figure 2-10 consists of three layers: input layer, hidden layer and output layer. Let $\mathbf{x} = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$ and $\mathbf{y} = \{y_1, y_2, \ldots, y_k, \ldots, y_l\}$ be input and output signals respectively. Let indices j and m denote neurons in the hidden layer, w_{ij} and w_{jk} denote the weight for the connection between neurons i, j and j, k. Thus, the actual output of the kth neuron in the output layer can be formulated as:

$$y_k = f_O(\sum_{j=1}^{m} y_j w_{jk} + \theta_k) \tag{2.10}$$

where

$$y_j = f_H(\sum_{i=1}^{n} x_i w_{ij} + \theta_j) \tag{2.11}$$

where f_O and f_H are the activation function for the output and hidden layer respectively, y_j is the output of the jth neuron in the hidden layer, and θ denotes bias. Let $\mathbf{y^d} = \{y_1^d, y_2^d, \ldots, y_k^d, \ldots, y_l^d\}$ be the desired output from the output layer, and the objective function for optimization is defined as the Mean Square Error (MSE) between the actual output $\mathbf{y}$ and the desired output $\mathbf{y^d}$. Thus, for a given training dataset $\mathbf{x}$ and $\mathbf{y^d}$, the average error $E(\mathbf{w})$ is defined as:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{l}(y_k^d - y_k)^2 \tag{2.12}$$

where the vector $\mathbf{w}$ is a set of weights (treating the bias θ as a weight too) that describes the neurons in this network, and the aim of the training algorithm is to minimize $E(\mathbf{w})$. According to the gradient-descent method, the weights in the hidden-to-output connections are updated by

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}} \tag{2.13}$$

where η is the learning rate or step size, provided that it is a sufficiently small positive number.

Applying the chain rule, the derivative in Eq.(2.13) can be expressed as

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial X_k}\frac{\partial X_k}{\partial w_{jk}} = \eta(y_k^d - y_k)[f_O{}'(X_k)]y_j = \eta\delta_{Ok}y_j \tag{2.14}$$

where $X_k = \sum_{j=1}^{m} y_j w_{jk} + \theta_k$ is the net weighted input to the kth neuron, and δ_{Ok} is called the error gradient and defined as $\delta_k = (y_k^d - y_k)[f_O{}'(X_k)]$. For the weight update on the input-to-hidden connections, it can be obtained in a similar manner:

$$\Delta w_{ij} = \eta \delta_{Hj} x_i \tag{2.15}$$

where $\delta_{Hj} = [f_H{}'(X_j)] \sum_{k=1}^{l} \delta_{Ok} w_{jk}$.

The above description is the BP algorithm in its simplest form. It is a supervised gradient-descent technique, wherein the MSE between the actual output of the network and the desired output is minimized. For a detailed description of the BP method, there are many excellent resources available (Lin and Lee 1996; Haykin 1999; Negnevitsky 2005; Huang, Hung et al. 2006).

The main potential drawbacks of the standard BP algorithm are that they quite often suffer from becoming stuck in a local minimum and they may require long learning periods in order to encode the training patterns (Hoya 2005). Methods to overcome these include training with different initial random weights, allowing extra hidden neurons, and lowering the gain term etc (Du and Swamy 2006). GNMM uses the LM algorithm (Du and Swamy 2006; Huang, Hung et al. 2006) to train MLPs. The LM method is a variation of the standard BP based on the computation of the Hessian matrix. Compared with conventional BP algorithm, it achieves a second-order speedup.

Note that MLPs are used in GNMM both as the fitness function and the core DM engine. However, the training settings are different for the two stages. For example, when serving as the fitness function, the learning rate α has to be

sufficiently small (e.g. 0.01) in order to avoid network oscillation; but when being used as the final modeller, α can have a relatively greater value (e.g. 0.04) to accelerate the learning process. In addition, the number of neurons in the hidden layer can be different as well. Generally speaking, an ANN performs better with an increased number of hidden neurons. However, when MLP is used as the fitness function, GNMM limits its hidden neurons in a way that it equals a fixed number N_h (i.e. N_h = number of hidden neurons) to make the input selection fairer. For example, consider two cases: one is three inputs with three hidden neurons, the other is the same three inputs with four hidden neurons. Given enough iterations, the second case will almost always produce higher accuracy. But does it mean the second input combination is better than the first? The exception is that when the input variables are too few (e.g. number of input variables < N_h), in that case the number of hidden neurons are set to be the same as the number of inputs.

2.3.3 Step 3 – Rule Extraction using Mathematical Programming

GNMM utilizes a mathematical programming methodology proposed by Tsaih and Chih-Chung (2004) for identifying and examining regression rules extracted from MLPs. Let $\mathbf{x}_m = \{x_{(m,1)}, \ldots, x_{(m,i)}\}$ denote the m-th row vector in $\mathbf{X}_f$, where $0 < m < a$, $0 < i < b$, ${}_2\mathbf{w}_j = \{{}_2w_{1j}, \ldots, {}_2w_{ij}\}$ stand for the weights between the j-th hidden neuron and the input layer, ${}_2\theta_j$ stands for the threshold of the j-th hidden neuron (see Figure 2-11). Returning to Eq.(2.9), the output of the j-th hidden neuron for $\mathbf{x}_m$ can be written as

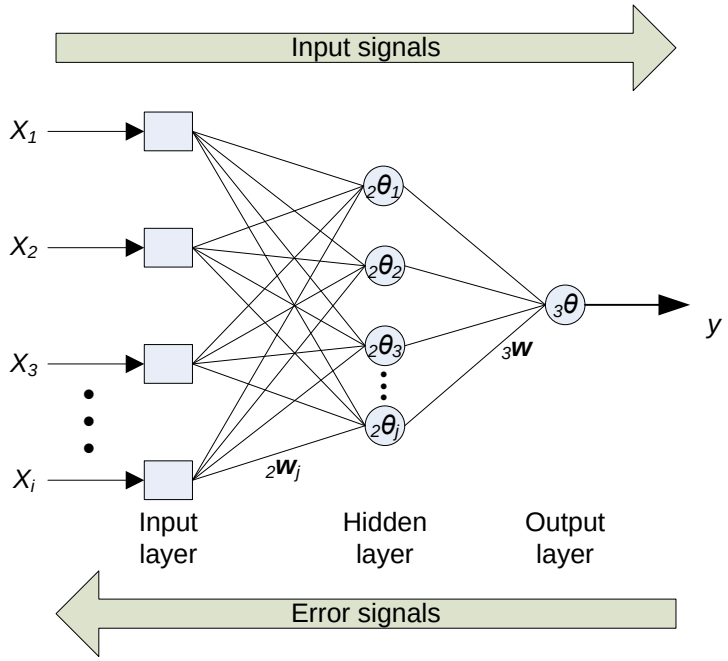


Figure 2-11: GNMM extracts regression rules from trained MLPs

$$ {}_{m}h_j \ = f(\mathbf{x}_m \mathbf{w}_j^{'} + \ {}_{2}\theta_j) \tag{2.16} $$

It has been shown (Tsaih and Chih-Chung 2004) that the following function $g(t)$ can be used to approximate tanh (Eq.(2.9))

$$ g(t) = \begin{cases} 1 & t \geq \kappa \\ \beta_1 t + \beta_2 t^2 & 0 \leq t \leq \kappa \\ \beta_1 t - \beta_2 t^2 & -\kappa \leq t \leq 0 \\ -1 & t \leq -\kappa \end{cases} \tag{2.17} $$

in which β_1 = 1.0020101308531, β_2 = −0.251006075157012, κ = 1.99607103795966. Letting $t_j = \mathbf{x}_m \mathbf{w}_j'$ and substituting $t = t_j + {}_{2}\theta_j$ into Eq.(2.17), we get the following

$$g(t_j + {}_2\theta_j) = \begin{cases} 1 & t_j \geq \kappa - {}_2\theta_j \\ (\beta_1 {}_2\theta_j + \beta_2 {}_2\theta_j^2) & \\ \quad +(\beta_1 + 2\beta_2 {}_2\theta_j)t_j & -{}_2\theta_j \leq t_j \leq \kappa - {}_2\theta_j \\ \quad +\beta_2 t_j^2 & \\ (\beta_1 {}_2\theta_j - \beta_2 {}_2\theta_j^2) & \\ \quad +(\beta_1 - 2\beta_2 {}_2\theta_j)t_j & -\kappa - {}_2\theta_j \leq t \leq -{}_2\theta_j \\ \quad -\beta_2 t_j^2 & \\ -1 & t_j \leq -\kappa - {}_2\theta_j \end{cases} \quad (2.18)$$

Thus for the j-th hidden neuron, the output value is approximated with a polynomial form of single variable t_j in each of four separate polyhedrons in the $\mathbf{x}_m$ space. For example, if $\mathbf{x}_m \in \{\mathbf{x}_m: -{}_2\theta_j \leq \mathbf{x}_m\mathbf{w}_j' \leq \kappa - {}_2\theta_j\}$w, then tanh ($\mathbf{x}_m\mathbf{w}_j'$ + ${}_2\theta_j$) is approximated with $\beta_{12}\theta_j + \beta_{22}\theta_j^2 + (\beta_1 + 2\beta_{22}\theta_j)t_j + \beta_2 t_j^2$. Because the activation function for the output layer is a linear function, a comprehensible regression rule associated with a trained ANN with i hidden neurons is thus:

- *IF* $\mathbf{x}_m \in \{\mathbf{x}_m: -{}_2\theta_j \leq \mathbf{x}_m\mathbf{w}_j' \leq \kappa - {}_2\theta_j$ for all $j\}$
- *THEN* $y' = {}_3\theta + \sum_{j=1}^{i} {}_3w_j (\beta_1 {}_2\theta_j + \beta_2 {}_2\theta_j^2 + (\beta_1 + 2\beta_2 {}_2\theta_j)t_j + \beta_2 t_j^2)$

Thus, once the training is done the neural network simulated output for $\mathbf{x}$ can be easily obtained. In other words, regression rules associated with the trained MLP can be derived.

2.4 Summary

The current chapter has briefly reviewed a selection of hybrid IS DM techniques including ANFIS, EFuNN, Fuzzy ARTMAP, as well as CGP. These

techniques have been successfully applied to a range of applications, and variations based on these techniques have been introduced in the literature to suit different needs. A simple fact is that, for example, using ANFIS as a keyword to search in fields 'Subject/Title/Abstract', database Compendex[4] returned a total of 1161 records after year 2000 (retrieved 30th July 2009). Due to this reason, this thesis will concentrate on these techniques in the most widely accepted forms, which are also the forms that have the greatest impact, rather than specific variations. Benchmarking studies using these techniques will be carried out in Chapter 6.

This chapter also gives a detailed description of GNMM, which is a general pattern classifier and modeller. It uses MLP as the core engine to perform data modelling/classification tasks, and hence inherits advantages that are built-in with the ANN techniques e.g. robustness and noise-tolerance. GNMM also employs the GA technique to perform MLP input optimization, which not only simplifies the MLP structure, accelerates the training process, but also makes the rule extraction more efficient and effective. Rule extraction eliminates MLP's 'black-box' nature, and makes knowledge extracted from GNMM more understandable.

In the following chapters (i.e. Chapter 3 to Chapter 5), we will show some applications of GNMM, in which we will demonstrate its implementation

[4] Compendex & Ei Backfile, http://www.ei.org/compendex.

details through case studies. And finally in Chapter 6, we will do a benchmarking study using GNMM against some hybrid IS DM methods, in which we will compare and summarize its relative advantages/disadvantages.

References

Bäck, T., D. B. Fogel, et al. (1997). Handbook of evolutionary computation. Bristol ; Philadelphia

New York, Institute of Physics Pub. ;

Oxford University Press.

Berry, M. J. A. and G. Linoff (2004). Data mining techniques: for marketing, sales, and customer relationship management. Indianapolis, Ind., Wiley Pub.

Bourg, D. M. and G. Seemann (2004). AI for game developers. Sebastopol, CA, O'Reilly.

Busque, M. and M. Parizeau (1997). "A Comparison of Fuzzy ARTMAP and Multilayer Perceptron for Handwritten Digit Recognition." Universite Laval, listopad.

Carpenter, G. A., S. Grossberg, et al. (1992). "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps." IEEE Transactions on Neural Networks 3(5): 698-713.

Carpenter, G. A., S. Grossberg, et al. (1991). "Fuzzy ART. Fast stable learning and categorization of analog patterns by an adaptive resonance system." Neural Networks 4(6): 759-771.

Chambers, L. (2001). The practical handbook of genetic algorithms : applications. Boca Raton, Fla., Chapman & Hall/CRC.

Chow, T. W. S. and S.-Y. Cho (2007). Neural networks and computing: learning algorithms and applications. London; Hackensack, NJ, Imperial College Press; Distributed by World Scientific.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function." Mathematics of Control, Signals, and Systems 2: 303-314.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems.

De Jong, K. A. (2006). Evolutionary computation: a unified approach. Cambridge, Mass., MIT Press.

del-Hoyo, R., B. Martín-del-Brío, et al. (2009). "Computational intelligence tools for next generation quality of service management." Neurocomputing In Press, Corrected Proof.

Du, K. L. and M. N. S. Swamy (2006). Neural networks in a softcomputing framework. London, Springer.

Engelbrecht, A. P. (2007). Computational intelligence: an introduction. Hoboken, N.J., John Wiley.

Georgiopoulos, M., J. Huang, et al. (1994). "Properties of learning in ARTMAP." Neural Networks 7(3): 495-506.

Harding, S. (2008). Evolution of image filters on graphics processor units using cartesian genetic programming, Hong Kong, China, Inst. of Elec. and Elec. Eng. Computer Society.

Haupt, R. L. and S. E. Haupt (2004). Practical genetic algorithms. Hoboken, N.J., John Wiley.

Haykin, S. S. (1999). Neural networks: a comprehensive foundation. Upper Saddle River, N.J., Prentice Hall.

Holland, J. H. (1975). Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence. Ann Arbor, University of Michigan Press.

Hoya, T. (2005). Artificial mind system : Kernel memory approach. Berlin New York, Spinger.

Huang, Y.-M., C.-M. Hung, et al. (2006). "Evaluation of neural networks and data mining methods on a credit assessment task for class imbalance problem." Nonlinear Analysis: Real World Applications 7(4): 720-747.

Hyvarinen, A. (1999). "Fast and robust fixed-point algorithms for independent component analysis." IEEE Transactions on Neural Networks 10(3): 626-634.

Hyvarinen, A., J. Karhunen, et al. (2001). Independent component analysis. New York, J. Wiley.

Jang, J.-S. R. (1993). "ANFIS: adaptive-network-based fuzzy inference system." IEEE Transactions on Systems, Man and Cybernetics 23(3): 665-685.

Jang, J.-S. R., C.-T. Sun, et al. (1997). Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence. Upper Saddle River, NJ, Prentice Hall.

Karray, F. O. and C. W. De Silva (2004). Soft computing and intelligent systems design : theory, tools, and applications. Harlow, England ; New York, Pearson/Addison Wesley.

Kasabov, N. (1998). "Evolving Fuzzy Neural Networks-Algorithms, Applications and Biological Motivation." Methodologies for the Conception, Design and Application of Soft Computing, World Scientific: 271-274.

Kasabov, N. (2008). "Evolving Intelligence in Humans and Machines: Integrative Evolving Connectionist Systems Approach." Computational Intelligence Magazine, IEEE 3(3): 23-37.

Kasabov, N. K. (2007). Evolving connectionist systems: the knowledge engineering approach. London, Springer.

Lin, C. T. and C. S. G. Lee (1996). Neural fuzzy systems: a neuro-fuzzy synergism to intelligent systems, London : Prentice-Hall International.

Lin, C. T. and C. S. G. Lee (1996). Neural fuzzy systems: a neuro-fuzzy synergism to intelligent systems. NJ, USA, Prentice-Hall, Inc. Upper Saddle River.

Liu, P. and H.-X. Li (2004). Fuzzy neural network theory and application. River Edge, NJ, World Scientific.

Miller, J. F. and P. Thomson (2000). "Cartesian Genetic Programming." Lecture notes in computer science.(1802): 121-132.

Mitchell, M. (1996). An introduction to genetic algorithms. Cambridge, Mass., MIT Press.

Mitra, S. and T. Acharya (2003). Data mining: multimedia, soft computing, and bioinformatics. Hoboken, NJ, John Wiley.

Negnevitsky, M. (2005). Artificial intelligence: a guide to intelligent systems. Harlow, England; New York, Addison-Wesley.

Poli, R. (1997). Parallel distributed genetic programming applied to the evolution of natural language recognisers, Berlin, Germany, Springer-Verlag.

Reeves, C. R. and J. E. Rowe (2003). Genetic algorithms: principles and perspectives: a guide to GA theory. Boston, Kluwer Academic Publishers.

Schaffer, J. D., R. A. Caruana, et al. (1989). "A study of control parameters affecting online performance of genetic algorithms for function optimization." Proceedings of the third international conference on Genetic algorithms: 51-60.

Sloper, J. E., G. L. Miotto, et al. (2008). "Dynamic Error Recovery in the ATLAS TDAQ System." Nuclear Science, IEEE Transactions on 55(1): 405-410.

Soyguder, S. and H. Alli (2009). "An expert system for the humidity and temperature control in HVAC systems using ANFIS and optimization with Fuzzy Modeling Approach." Energy and Buildings 41(8): 814-822.

Tan, S. C., M. V. C. Rao, et al. (2008). "Fuzzy ARTMAP dynamic decay adjustment: An improved fuzzy ARTMAP model with a conflict resolving facility." Applied Soft Computing 8(1): 543-554.

Tsaih, R. and L. Chih-Chung (2004). The layered feed-forward neural networks and its rule extraction. Proceedings, International Symposium on Neural Networks, Dalian, China, Springer-Verlag.

Vose, M. D. (1999). The simple genetic algorithm : foundations and theory. Cambridge, Mass., MIT Press.

Walker, J. A. and J. F. Miller (2008). "The automatic acquisition, evolution and reuse of modules in Cartesian genetic programming." IEEE Transactions on Evolutionary Computation 12(4): 397-417.

Wang, S.-C. (2003). Interdisciplinary computing in Java programming. Boston, Kluwer Academic.

Wilson, G. and W. Banzhaf (2008). A comparison of cartesian genetic programming and linear genetic programming, Naples, Italy, Springer Verlag.

Xu, Z., J. Xuan, et al. (2009). "Application of a modified fuzzy ARTMAP with feature-weight learning for the fault diagnosis of bearing." Expert Systems with Applications 36(6): 9961-9968.

Yam, J. Y. F. and T. W. S. Chow (2000). "A weight initialization method for improving training speed in feedforward neural network." Neurocomputing 30(1-4): 219-232.

Yam, Y.-F., C.-T. Leung, et al. (2002). "An independent component analysis based weight initialization method for multilayer perceptrons." Neurocomputing 48(1-4): 807-818.

Yang, J., E. L. Hines, et al. (2008). A Genetic Algorithm-Artificial Neural Network Method for the Prediction of Longitudinal Dispersion Coefficient in Rivers. Advancing Artificial Intelligence through Biological Process Applications. A. Porto, A. Pazos and W. Buño, Idea Group Inc.: 358-374.

Yu, L. (2007). Foreign-exchange-rate forecasting with artificial neural networks. New York, Springer.

Yuen, S. Y. and C. K. Chow (2009). "A genetic algorithm that adaptively mutates and never revisits." IEEE Transactions on Evolutionary Computation 13(2): 454-472.

Zhang, M. (2009). Artificial higher order neural networks for economics and business. Hershey, Information Science Reference.

Chapter 3 Prediction of Longitudinal Dispersion Coefficient in Rivers

3.1 Introduction

In previous chapters we reviewed DM concepts, theoretical backgrounds of IS DM techniques (Chapter 1), and proposed the GNMM method (Chapter 2). This is the first chapter in terms of GNMM's applications/case studies. It is followed by two other application chapters (Chapter 4 and Chapter 5) and one benchmarking chapter (Chapter 6) before conclusions are drawn. This chapter involves the application of GNMM in the field of Civil Engineering, specifically in the prediction of longitudinal dispersion coefficient in rivers. The aim of the current chapter is to provide an insightful analysis of GNMM's implementation in the context of longitudinal dispersion coefficient prediction. This is achieved by a detailed comparative study of GNMM's input determination and rule extraction process based on very well-studied classic and representative sets of data (Yang, Hines et al. 2007; Yang, Hines et al. 2008). Furthermore, PCA and SOM analysis are performed to cross-validate the GA input variable selection results.

3.2 Background

An important application of environmental hydraulics is the prediction of the fate and transport of pollutants that are released into watercourses, either as a result of accidents or as regulated discharges. Such predictions are primarily dependent on the water velocity, longitudinal mixing, and chemical/physical reactions etc, of which longitudinal dispersion coefficient is a key variable for the description of the longitudinal spreading in a river. After being first introduced in Taylor (1954), extensive studies have been made based on experimental and field data for predicting the dispersion coefficient (Jobson 1997; Seo and Cheong 1998; Deng, Singh et al. 2001; Wallis and Manson 2004; Boxall and Guymer 2007). The majority of such work has used the Advection-Dispersion Equation approach because strong physical basis makes it more amenable to predicting conditions in rivers and streams for which no model has previously been calibrated (Wallis and Manson 2004).

The concept of longitudinal dispersion coefficient was first introduced in Taylor (1954). Based on this work, the following integral expression was developed (Fischer, List et al. 1979; Seo and Cheong 1998) and generally accepted:

$$K = -\frac{1}{A}\int_0^B hu' \int_0^y \frac{1}{\varepsilon_t h}\int_0^y hu' dydydy \tag{3.1}$$

where K = longitudinal dispersion coefficient; A = cross-sectional area; B = channel width; h = local flow depth; u' = deviation of local depth mean flow

velocity from cross-sectional mean; y = coordinate in the lateral direction; and ε_t = local (depth averaged) transverse mixing coefficient. An alternative approach utilises field tracer measurements and applies the method of moments. It is also well documented in the literature (Rutherford 1994; Guymer 1999; Rowinski, Piotrowski et al. 2005) and defines K as

$$K = \frac{U_c^2}{2}\left[\frac{\sigma_t^2(x_2) - \sigma_t^2(x_1)}{\bar{t}_2 - \bar{t}_1}\right] \tag{3.2}$$

where U_c = mean velocity, x_1 and x_2 denotes upstream and downstream measurement sites, $\bar{t}$ = centroid travel time, $\sigma_t^2(x)$ = temporal variance.

However, owing to the requirement for detailed transverse profiles of both velocity and cross-sectional geometry, Eq.(3.1) is rather difficult to use. Furthermore, Eq.(3.2), called the method of moments (Wallis and Manson 2004), requires measurements of concentration distributions and can be subject to serious errors due to the difficulty of evaluating the variances of the distributions caused by elongated and/or poorly defined tails. As a result, extensive studies have been made based on experimental and field data for predicting the dispersion coefficient (Jobson 1997; Seo and Cheong 1998; Deng, Singh et al. 2001; Wallis and Manson 2004).

For example, employing 59 hydraulic and geometric datasets measured in 26 rivers in the United States, Seo and Cheong (1998) used dimensional analysis

and applied the one-step Huber method, a nonlinear multi-regression method, to derive the following equation:

$$K = 5.915(Hu^*)\left(\frac{B}{H}\right)^{0.62}\left(\frac{U}{u^*}\right)^{1.428} \tag{3.3}$$

in which u^* = shear velocity. This technique uses the easily measureable hydraulic variables B, H and U, together with a frequently used parameter, extremely difficult to accurately quantify in field applications, u^*, to estimate the dimensionless dispersion coefficient K from Eq.(3.3). Another empirical equation developed by Deng et al. (2001) is a more theoretically based approximation of Eq.(3.1), which not only includes the conventional parameters of (B/H) and (U/u^*) but also the effects of the transverse mixing ε_{t0}, as follows:

$$K = 0.15\left(\frac{Hu^*}{8\varepsilon_{t0}}\right)\left(\frac{B}{H}\right)^{5/3}\left(\frac{U}{u^*}\right)^{2} \tag{3.4}$$

where

$$\varepsilon_{t0} = 0.145 + \left(\frac{1}{3520.0}\right)\left(\frac{B}{H}\right)^{1.38}\left(\frac{U}{u^*}\right) \tag{3.5}$$

These equations are easy to use, assuming measurements or estimates of the bulk flow parameters are available. However, they may be unable to capture

the complexity of the interactions of the fundamental transport and mixing mechanisms, particularly those created by non-uniformities across the wide range of channels encountered in nature. In addition, the advantage of one expression over another is often just a matter of the selection of data and the manner of their presentation. Regardless of the expression applied, one may easily find an outlier in the data, which definitely does not support the applicability of a particular formula. An expectation that, in spite of the complexity of the river reach, the dispersion coefficient may be represented by one of the empirical formulae seems exaggerated (Rowinski, Piotrowski et al. 2005).

Furthermore, most of the studies have been carried out based on specific assumptions and channel conditions and therefore the performance of the equations varies widely for the same stream and flow conditions. For instance, Seo and Cheong (1998) used 35 of the 59 measured datasets to establish Eq.(3.3) and the remaining 24 for verifying their model. While the model of Deng et al. (2001) (Eq.(3.4) and Eq.(3.5)) is limited to straight and uniform rivers. They also assume that the river has a width-to-depth ratio greater than 10. Therefore, a model that has greater general applicability is desirable.

Recently ANN modelling approaches have been embraced enthusiastically by practitioners in water resources, as they are perceived to overcome some of the difficulties associated with traditional statistical approaches, e.g. making

assumptions with regard to stream geometry or flow dynamics (Maier and Dandy 1998). They offer an effective approach for handling large amounts of dynamic, non-linear and noisy data, especially when the underlying physical relationships are not fully understood (Haykin 1994; Hagan, Demuth et al. 1996; Cannas, Fanni et al. 2006).

In specific terms, several authors (Kashefipour, Falconer et al. 2002; Rowinski, Piotrowski et al. 2005; Tayfur and Singh 2005; Piotrowski, Rowinski et al. 2006; Tayfur 2006) have reported successful applications of ANNs to the prediction of dispersion coefficient. For example, in the case of Tayfur and Singh (2005) the ANN was trained and tested using 71 data samples of hydraulic and geometric variables and dispersion coefficients measured on 29 streams and rivers in the United States, with the result that 90% of the dispersion coefficient was explained. Rowinski, Piotrowski et al. (2005) applied an MLP with the LM Algorithm to three different datasets which have been explored in the literature. The lowest percentage of training data mean error was found to be 7.02%. However, there is a lack of a suitable input determination methodology for ANN models in these applications. Moreover, without further interpretation of the trained network, their results are not easily transferable.

3.3 Datasets and Pre-processing

3.3.1 Datasets

In the last decade, regions within the UK Environment Agency (EA) have completed a number of dye tracing studies and more than one hundred different tracing studies were analysed to obtain estimates of the mean travel velocity and longitudinal dispersion. A database of travel times and dispersion was developed (Guymer 1999) comprising the tracing works cited. The database (denoted by Data I) contains 196 data samples from 27 different rivers and includes information relevant to the traces; including geographical and physical attributes of the river reaches as well as optimized Advection-Dispersion Model (ADE) and Aggregated Dead Zone Model (ADZ) travel times, velocities, ADE longitudinal dispersion coefficients and ADZ dispersive fractions.

The second dataset, Data II, contains 71 sets of measurements from 29 rivers in the United States. This dataset has previously been very well studied in the literature (Seo and Cheong 1998; Deng, Singh et al. 2001; Rowinski, Piotrowski et al. 2005; Tayfur and Singh 2005).

3.3.2 Data Pre-processing

Usually data pre-processing of GNMM includes scaling the inputs and targets so that they fall within a specified range. However, since there are a total of 49 variables available in Data I, the prior objective of data pre-processing is to reduce the dimensionality of the original set of inputs by eliminating

Table 3-1: Variables in Data I and II

Data I (16 variables)		Data II (8 variables)	
Start location:	C_s (km^2), D_s (km) M_s (m^3/s), Q_s (m^3/s)	Independent:	B (m), H (m) U (m/s), u^* (m/s), α
End location:	C_e (km^2), D_e (km) M_e (m^3/s), Q_e (m^3/s)		
Reach:	S, L (m), D_r (m)		
Gauging station:	C_g (km^2), A (m^3/s) M_g (m^3/s), Q_g (m^3/s) I (m^3/s)	Dependent:	B/H, U/u^*, β

redundant and/or dependant variables. This will result in a set of independent inputs that are not necessarily related to the dispersion coefficient. This subset of inputs can then be used in GNMM to determine which of these inputs are most appropriate for mapping to the output.

Among the 49 available variables in Data I, non-numerical variables such as river name, flow excedence/category, and start location grid reference are removed first of all. These variables are valuable in terms of dye tracing studies but do not provide useful information in the current context. Secondly, dependant variables are discarded. For example, start and end position of the river location elevation are discarded, while reach slope is kept; reach sinuosity is removed while reach length and straight distance are kept. After being processed, Data I contains 16 variables which belong to 4 categories in the original dataset, e.g. start/end location, reach, and gauging station. These

variables are: for the start/end location (subscripts s/e), catchment area (C), distance from injection point (D), theoretical mean flow (M), and theoretical Q95 flow (Q); for the reach in question, slope (S), reach length (L), and straight distance (D_r); for the gauging station, catchment area (C_g), average daily flow (A), daily mean flow (M_g), theoretical Q95 flow (Q_g), instant flow (I). All these variables are listed in Table 3-1.

Data II contains 8 variables apart from the longitudinal dispersion coefficient. There are five independent variables: channel width (B), flow depth (H), velocity (U), shear velocity (u^*) and river sinuosity (α = channel length/valley length). Dependant variables are width-to-depth ratio (B/H), relative shear velocity (U/u^*) and channel shape variable ($\beta = ln(B/H)$). Data II variables are listed in Table 3-1 too. It is worth noting that dependent variables exist in Data II (B/H, U/u^*), which indicates that eliminating redundant and/or dependant variables is not always necessary in GNMM. Since the aim of this step is to reduce the dimensionality of the data by eliminating redundant and/or dependant variables, obviously in Data II the dimensionality is not a main issue. On the other hand, as we will show later, Data II can be treated as an example of how GNMM handles dependent variables.

3.3.3 Division into Training and Testing Data

Before the start of GA variable selection, it is necessary to divide the dataset into training and testing subsets. This is to avoid over fitting when

chromosomes are being evaluated in an MLP (Lin and Lee 1996). The division is achieved by selecting representative sets for both of the training and testing data:

(1) Among the 196 data samples contained in Data I, some contain a high percentage of missing values, or indications that the data recorded is inaccurate. In order to obtain reliable results, these data are removed. As a result, the final dataset contains 127 data samples (see Appendix A). After division, the training subset, denoted by Data I_t, contains 102 samples; while the testing subset, denoted by Data I_s, consists of the remaining 25.

(2) Similarly, Data II (see Appendix B) is divided into two subsets, Data II_t and Data II_s for training and testing respectively. Data II_t contains 49 datasets out of 71, while Data II_s consists of the remaining 22.

Normally when we have a large quantity of data we would typically use more data for training and less for testing. With small datasets we may repeat the process several times by randomly generating training and testing data to ensure that our results are reliable for the dataset. Table 3-2 and Table 3-3 show statistics of these subsets respectively. Note that in these tables, Avg, Avg_t and Avg_s mean the average for the whole dataset, training and testing subset average correspondingly.

Table 3-2: Data I statistics

	Start Location				End Location			
	$C_s\,(km^2)$	$D_s\,(km)$	$M_s\,(m^3/s)$	$Q_s\,(m^3/s)$	$C_e\,(km^2)$	$D_e\,(km)$	$M_e\,(m^3/s)$	$Q_e\,(m^3/s)$
Max	3314.75	41.50	49.55	9.47	3315.25	46.50	49.55	9.47
Min	16.00	1.00	0.18	0.02	9.25	3.40	0.39	0.03
Avg	714.51	9.82	13.04	2.10	858.97	16.43	15.15	2.45
Avg_t	643.05	8.49	13.57	1.97	838.79	16.27	17.10	2.61
Avg_s	732.15	10.15	12.90	2.13	863.91	16.47	14.63	2.41
	Reach			Gauging Station				
	S	$L\,(m)$	$D_r\,(m)$	$C_g\,(km^2)$	$A\,(m^3/s)$	$M_g\,(m^3/s)$	$Q_g\,(m^3/s)$	$I\,(m^3/s)$
Max	0.0244	14697.0	12133.50	3314.80	47.14	75.00	6.60	75.00
Min	0.0000	1058.00	915.57	20.00	0.44	0.48	0.06	0.50
Avg	0.0023	6037.06	4342.88	792.39	12.38	10.08	1.93	10.20
Avg_t	0.0030	6775.73	4834.09	736.99	13.48	11.33	1.83	11.29
Avg_s	0.0022	5856.02	4222.49	805.96	12.11	9.78	1.95	9.93

Table 3-3: Data II statistics

	$B\,(m)$	$H\,(m)$	$U\,(m/s)$	$u^*\,(m/s)$	B/H	U/u^*	β	α	$K\,(m^2/s)$
Max	711.2	19.94	1.74	0.553	156.5	19.63	5.05	2.54	892.0
Min	11.9	0.22	0.03	0.002	13.8	1.29	2.62	1.08	1.9
Avg	83.0	1.70	0.54	0.088	51.7	7.62	3.79	1.39	107.7
Avg_t	62.9	1.31	0.49	0.084	51.4	7.13	3.79	1.37	98.4
Avg_s	127.6	2.55	0.66	0.097	52.4	8.72	3.77	1.42	128.4

When forming these two subsets, the present work follows that of Tayfur and Singh (2005), in order to compare results. However, as mentioned in Tayfur and Singh (2005): 'In choosing the datasets for training and testing, special

attention was paid to ensure that we have representative sets so as to avoid bias in model prediction' (p. 993).

For example, in Data II the range for the dispersion coefficient (K) varies from 1.9 to 892 m^2/s, and K is greater than 100 m^2/s in 21 cases, which counts for about 30% of all available measured coefficient values. The range for the width-to-depth ratio (B/H) of the datasets varies from 13.8 to 156.5 and (B/H) is greater than 50 in 26 cases (37%). After division the percentages of K > 100 m^2/s and B/H > 50 are also comparable for both Data II_t and Data II_s. For example, in Data II_s 25% of K is greater than 100 m/s^2 (this ratio is 31% in Data II_t), also, in Data II_s 40% of B/H is greater than 50 (this ratio is 31% in Data II_t).

3.4 GNMM Implementation

3.4.1 Variable Selection

GNMM is mainly implemented in MATLAB (v7.2)[5] (see Appendix C), using the Genetic Algorithm and Direct Search Toolbox, as well as the Neural Network Toolbox (GNMM also includes a VBA script to visually select outstanding variables, see Appendix D). GA parameters are set as follows: p_c = 0.8, p_m = 0.01, the learning rate α = 0.01, the elite group size N_e = 2. Other settings for each GA run are as shown in Table 3-4, along with CPU speed and CPU time. It should be noted that N_e in Table 3-4 stands for 'number of epochs per chromosome'.

[5] The MathWorks, http://www.mathworks.com/.

Table 3-4: GA parameters and CPU speeds/time

	Case	N_p	N_g	N_e (/chrom.)	CPU Speed	CPU Time (s)
Data I	1	200	100	100	3.2 GHz (Pentium 4)	22760.52
	2	200	100	50	2.66 GHz (Celeron D)	34992.37
	3	400	100	100	900 MHz (UltraSPARC III)	139491.91
	4	400	200	20	3.2 GHz (Pentium 4)	34608.08
	5	400	400	20	900 MHz (UltraSPARC III)	216925.95
	6	200	200	20	2.66 GHz (Celeron D)	40111.13
	7	400	100	300	3.2 GHz (Pentium 4)	127634.52
Data II	1	200	100	100	3.20 GHz (Pentium 4)	17830.33
	2	200	100	50	2.66 GHz (Celeron D)	31772.62
	3	400	100	100	3.20 GHz (Pentium 4)	33600.17
	4	200	200	20	2.66 GHz (Celeron D)	45690.86
	5	200	200	100	3.20 GHz (Pentium 4)	39280.16

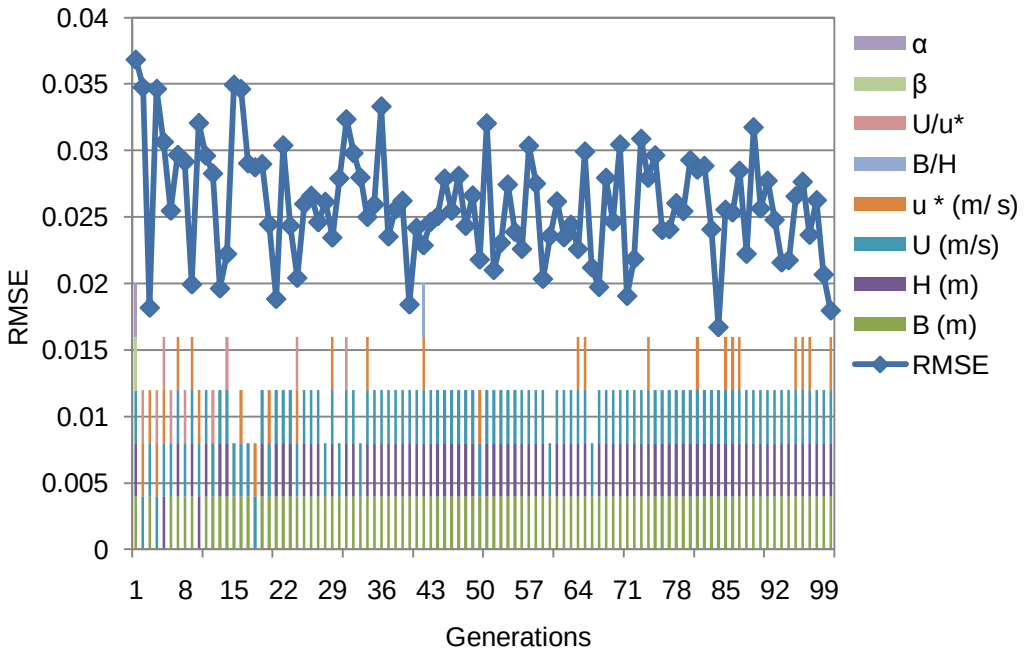


Figure 3-1: RMSE and winning variables for Data II training subset in Case 2. RMSE errors are show as dotted-lines, and the corresponding component variables of the winning chromosome are shown below the line

As described previously in Section 2.3.1.3 The Elite Group and Appearance Percentage, in order to minimize the randomness in the MLP initialization, we introduced the elite group and *appearance percentage* into GNMM. We will now consider these two techniques through the following example. For the purpose of easier visual presentation, Data II is chosen because it contains fewer variables than Data I (see Appendix E RMSE and Winning Variables for Case 7 of Data I). However, similar results can be found in all cases listed in Table 3-4.

Figure 3-1 illustrates the RMSE of the winning chromosome and its component variables for each generation of Data II training subset in Case 2. It can be seen that although the overall trend of the RMSE is decreasing, it is not necessarily the case that later generations produce lower RMSE than earlier ones. This is a distinct feature of GNMM. For ordinary GA optimization problems, there is no randomness associated with the fitness function. Thus, for a certain chromosome, it will always evaluate to a fixed fitness value. However, this is not the case for GNMM as the fitness function in GNMM is the training error of an MLP. Although we apply the ICA based weight initialization method to minimize the randomness, the ICA coefficient is still based on random numbers. Therefore, in GNMM it is possible that the same chromosome may still produce a different fitness value; this is the reason why we introduced the concept of *appearance percentage*.

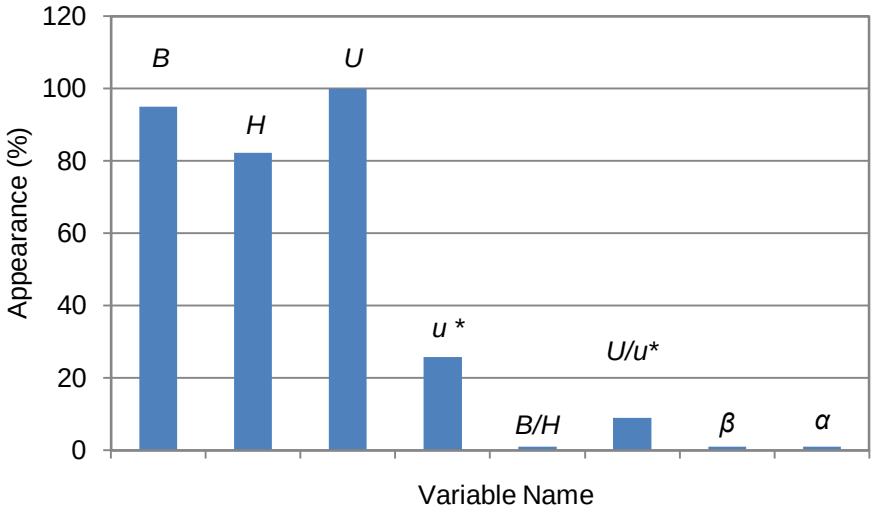


Figure 3-2: Appearance percentage for Case 2 of Data II training subset

It is also evident from Figure 3-1 that the changing range of the RMSE is narrowing too. This implies that the GA has identified a high yielding region and was searching exploitationally, since the GA has already found several variable combinations (e.g. U, H, and B) that produce a smaller error and was further exploring different combinations of these variables. Several successive generations around the 50th yield the same winning chromosome. This is the effect of the elite group. As mentioned before, chromosomes which performed best are protected so that they can compete in the next generation. Apparently, these chromosomes were kept as survivors in the next generation.

Compared to Figure 3-1, which depicts component variables in each winning chromosome, Figure 3-2 shows some statistical information for all the variables by means of *appearance percentage*. It can be seen that the most frequently appearing variables are U (100%), B (95%) and H (82%), followed by u^* (26%), U/u^* (9%), whereas β, α and B/H are all less than 1%. These results

seem promising as there is a clear distinction between different input variables; however, GNMM does not rely on a single run. This is because GA operations are based on random number generators. Once the random number seed changes, these results change accordingly. This also explains why there are 7 cases for Data I and 5 cases for Data II – more runs are needed to find a clear distinction. The results are shown graphically in Figure 3-3 and Figure 3-4.

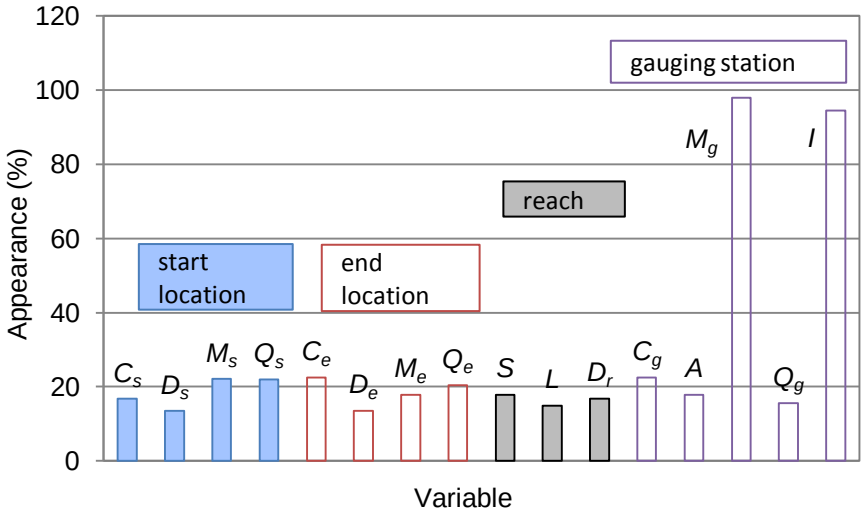


Figure 3-3: Appearance percentage for Data I training subset

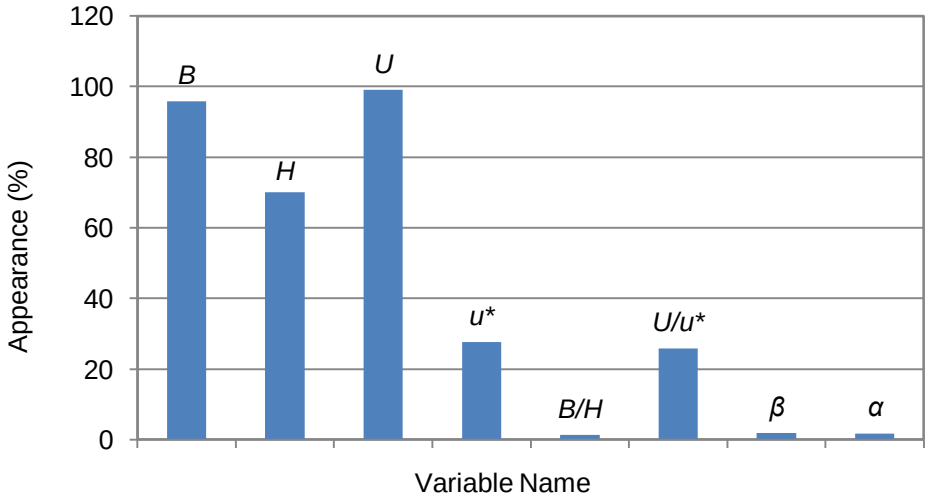


Figure 3-4: Appearance percentage for Data II training subset

After running the GA seven times for Data I and five times for Data II, a clear distinction was evident between variables. The *appearance percentage* of the input variables of these two datasets are shown in Figure 3-3 and Figure 3-4 respectively. Taking Figure 3-3 for example, it can be seen that the most frequently appearing variables are gauging station daily mean flow (M_g, 98%), and instant flow (I, 94%). The differences between the remaining variables are not very significant. The variables with more than 20% appearance are gauging station catchment area (C_g, 23%), end location catchment area (C_e, 22%), start location theoretical mean flow (M_s, 22%), start location theoretical Q95 flow (Q_s, 21%), and end location theoretical Q95 flow (Q_g, 20%). Variables with less than 15% appearances are reach length and start/end location distance from injection point (L, D_s and D_e, 14%, 13% and 13% respectively). As such, M_g and I for Data I are kept during the next MLP training stage in GNMM, while the rest of the variables are all removed. Similarly, it can be seen that the most frequently appearing variables for Data II are U (99%), B (96%) and H (70%), followed by u^* (28%), U/u^* (26%), whereas β, α and B/H are all less than 2%. Thus U, B and H for Data II are kept and the rest are removed.

It is interesting to note that Figure 3-4 has a very similar distribution to Figure 3-2. The only difference is that the high *appearance percentage* shown in Figure 3-2 is slightly reduced in Figure 3-4; and vice versa for the low *appearance percentage*. This is the averaging effect of the *appearance*

percentage technique. It should also be noted that dependant variables B/H, U/u^* and β are automatically filtered out.

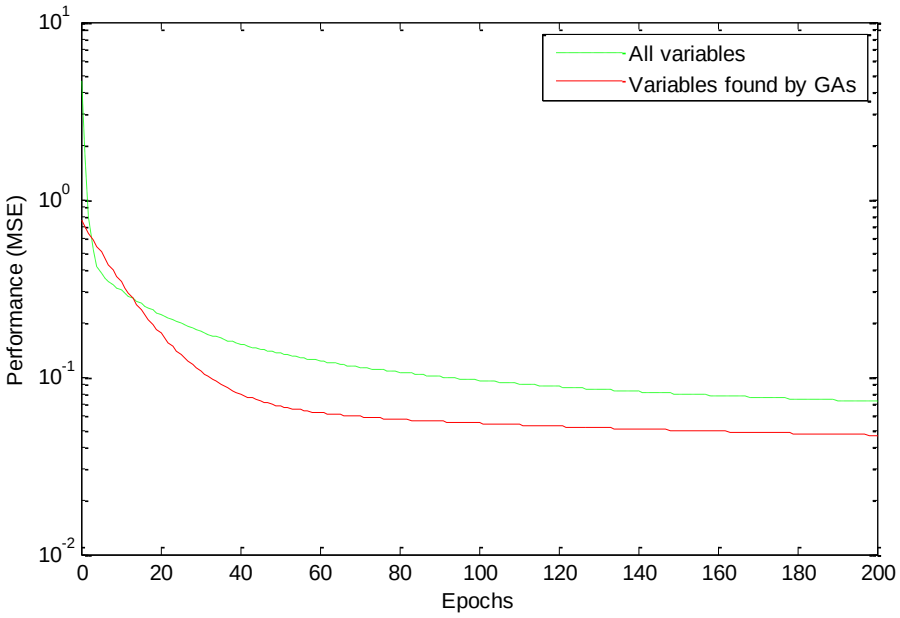


Figure 3-5: Comparison of performance using all variables and selected variables for Data I training subset for a single run

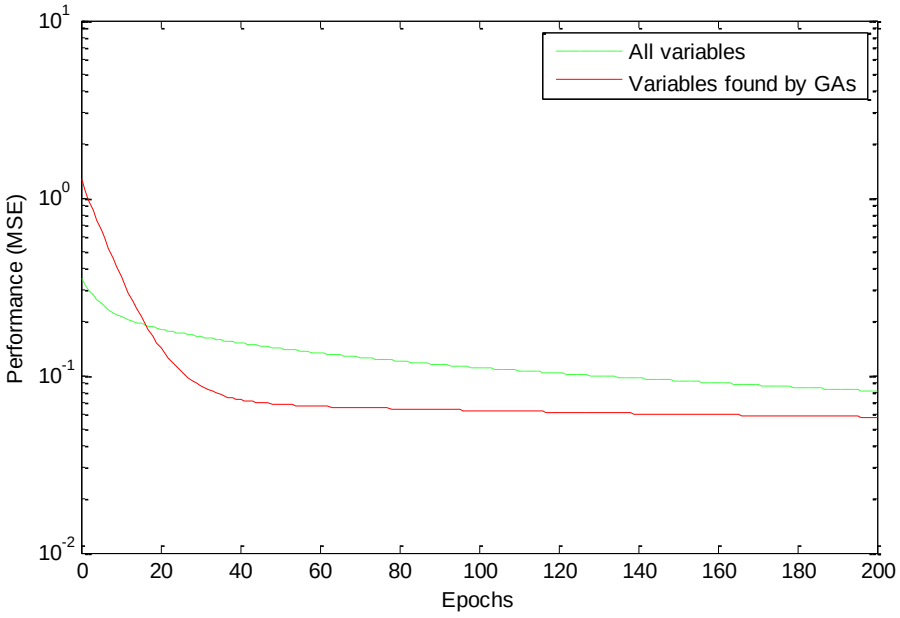


Figure 3-6**:** Comparison of performance using all variables and selected variables for Data II training subset for a single run

The effect of such variable selection can clearly be seen in Figure 3-5 and Figure 3-6. Within 200 epochs, the 'selected variable combination' of Data I achieved a much smaller MSE than using them all (0.043 vs. 0.162). The same is true for Data II (0.067 vs. 0.115).This further demonstrates that not all of the potential input variables are equally informative since some may be noisy, correlated or have no significant relationship with the longitudinal dispersion coefficient.

3.4.2 MLP Training

As a result of the input variables selection process, M_g and I for Data I and U, B and H for Data II are identified to be the ones most frequently occurring throughout all the populations. Thus the subset formed by these variables, which is the subset that produces the minimal error within a given number of epochs, is utilised in the final training process.

By setting learning rate to α = 0.04, number of neurons in the hidden layer = 5 and 3 for Data I and Data II respectively, and running the MLP five times, the minimum RMSE for Data I_t is 5.92, and the coefficient of determination (R^2) is 0.83 at iteration N_e = 2217513. The corresponding number for Data II_t is RMSE = 34.85, R^2 = 0.96 at N_e = 19887. In both cases, these results imply that the MLP model is satisfactorily trained. Figure 3-7 and Figure 3-8 show the measured and predicted longitudinal dispersion coefficients for Data I and Data II respectively. It is evident that the data is evenly distributed around the

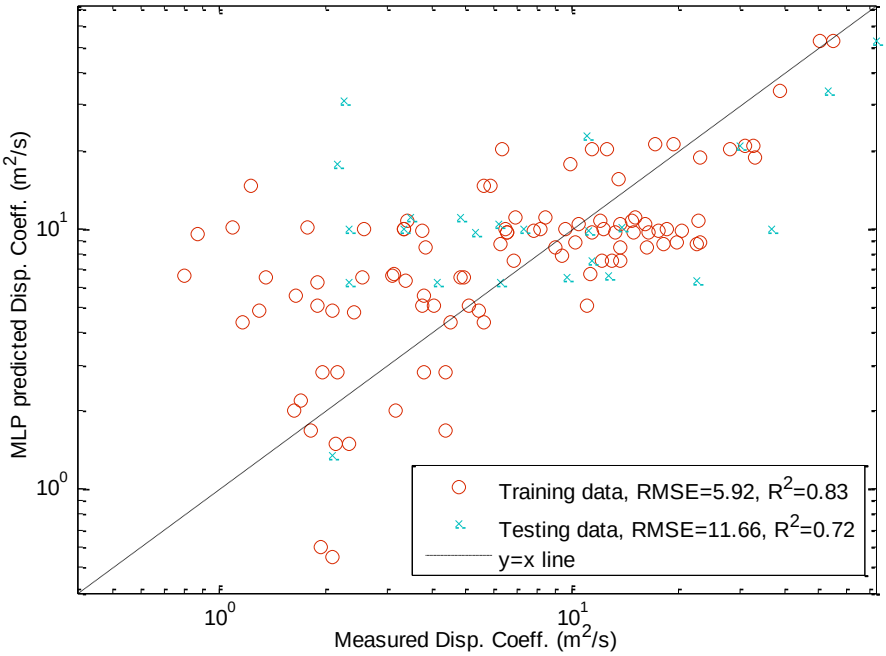


Figure 3-7: Predicted and measured longitudinal dispersion coefficients for Data I

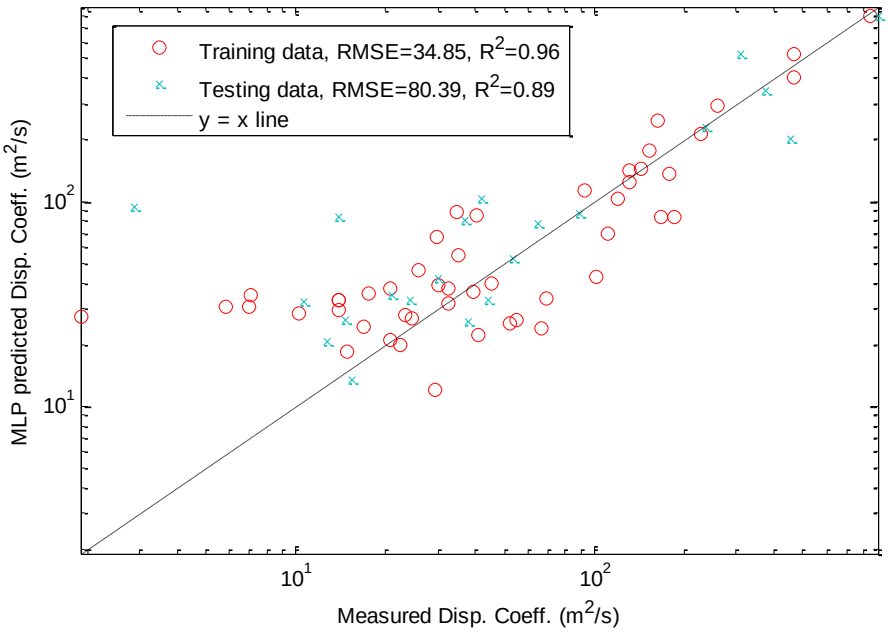


Figure 3-8: Predicted and measured longitudinal dispersion coefficients for Data II

Table 3-5: Comparison of Data II_s (testing subset) results when using 4 different methods. For GNMM, the mean RMSE of 5 runs are given along with the standard deviations

Model (Reference)	R^2	RMSE(m^2/s)
GNMM		128.5±23.9
MLP (Tayfur and Singh 2005)	0.70	193.0
Eq.(3.4) (Deng, Singh et al. 2001)	0.55	610.0
Eq.(3.3) (Seo and Cheong 1998)	0.50	812.0

'y=x' line. For Data I_s, the trained MLP produced RMSE = 11.66 and R^2 = 0.72. For Data II_s, these numbers are R^2 = 0.89 and RMSE = 80.39. This means that MLPs have been successfully trained.

Since Data II is a well studied dataset, a comparison is made between Data II results obtained using GNMM and other methods in the literature. In this case, over these five runs the mean RMSE and standard deviations are 128.5 and 23.9 for Data II_s. Comparing the above results to those in Tayfur and Singh (2005) (as in Table 3-5), GNMM performs better. Although MLPs are being adopted in both applications, the difference lies in the fact that only a portion of available variables are used in GNMM instead of using them all as in Tayfur and Singh (2005). For the test data, a comparison has also been made with some other models, as in Table 3-5, which also shows that GNMM produces the best results, and ANN models (GNMM and MLP in Tayfur and Singh (2005)) generally perform better.

3.4.3 Rule Extractions

An important feature of GNMM is that by eliminating redundant input data, understanding complex models has been made simple. This can be illustrated by applying mathematical-programming based rule extraction to trained MLPs.

For simplicity (less hidden neurons), we take Data II as an example. The final weights and biases of the MLP (see Figure 3-8 (c)) that minimizes MSE are as follows: ${}_3\theta$ = −0.6031, ${}_2\theta_1$ = 1.4022, ${}_2\theta_2$ = −0.0143, ${}_2\theta_3$ = −4.1393, ${}_3\mathbf{w}$ = (−1.7705, 0.8517, −1.2564), ${}_2\mathbf{w}_1$ = (4.1222, 0.9600, −1.5078), ${}_2\mathbf{w}_2$ = (5.7385, −4.3290, 1.1943), ${}_2\mathbf{w}_3$ = (−0.7147, −6.7842, 0.3987). Applying Eq.(2.18), we have

$$t_1 = 4.1222B' + 0.9600H' - 1.5078U' \tag{3.6}$$

$$t_2 = 5.7385B' + 4.3290H' + 1.1943U' \tag{3.7}$$

$$t_3 = -0.7147B' - 6.7842H' + 0.3987U' \tag{3.8}$$

where B', H' and U' are scaled variables according to Eq.(2.3) and Table 3-3. Also, we have

$$g(t_1) = \begin{cases} 1 & t_1 \geq 0.5939 \\ 0.9115 + 0.2981t_1 - 0.2510t_1^2 & -1.4022 \leq t_1 \leq 0.5939 \\ 1.8985 + 1.7059t_1 + 0.2510t_1^2 & -3.3983 \leq t_1 \leq -1.4022 \\ -1 & t_1 \leq -3.3983 \end{cases} \tag{3.9}$$

$$g(t_1) = \begin{cases} 1 & t_2 \geq 2.0104 \\ -0.1444 + 1.0092t_2 - 0.2510t_2^2 & 0.0143 \leq t_2 \leq 2.0104 \\ -0.0143 + 0.9948t_2 + 0.2510t_2^2 & -1.9818 \leq t_2 \leq 0.0143 \\ -1 & t_2 \leq -1.9818 \end{cases} \quad (3.10)$$

$$g(t_1) = \begin{cases} 1 & t_3 \geq 6.1354 \\ -8.4483 + 3.0800t_3 - 0.2510t_3^2 & 4.1396 \leq t_3 \leq 6.1354 \\ 0.1531 - 1.0760t_3 + 0.2510t_3^2 & 2.1432 \leq t_3 \leq 4.1396 \\ -1 & t_3 \leq 2.1432 \end{cases} \quad (3.11)$$

Since the activation function in the output neuron is a linear function, we also get

$$y' = -1.7705g_1 + 0.8517g_2 - 1.2564g_3 - 0.6031 \quad (3.12)$$

Thus, regression rules are extracted from the trained MLP. Among these 64 (4^3) potential rules, some are null and will never execute. Null rules can be identified using the Simplex algorithm, see Tsaih and Chih-Chung (2004) for details.

Rules fired for Data I and Data II are shown in Table 3-6 and Table 3-7 respectively, where the number of training and test data samples associated with each rule is also listed. Recall that in Eq.(2.17) the input domain of hidden neurons is divided into four sub-domains, this corresponds to the actual value of the digits in each rule ranging from 1 to 4. On the other hand, the length of each rule corresponds to the number of neurons in the hidden layer. Since 5 and 3 neurons were used in the hidden layer for Data I and Data II respectively,

Table 3-6: Rules fired for Data I

No.	Rule	Data I_t	Data I_s
1	24121	2	1
2	41342	2	
3	41442	55	13
4	42341	18	5
5	42342	8	
6	43241	7	2
7	43341		1
8	44131	3	
9	44141	3	1
10	44241	1	
11	44242	3	2

Table 3-7: Rules fired for Data II

No.	Rule	Data II_t	Data II_s
1	112	1	2
2	113	24	6
3	123	3	1
4	124	6	6
5	133		2
6	213	3	
7	214	2	1
8	223	5	
9	224	1	1
10	233	4	
11	431		1
12	441		1
13	442		1

hence the corresponding length of each rule set is 5 and 3.

The regression rules summarised above give us an idea of the importance of each rule and the distribution of the data. These rules can also be written in the antecedent/consequent format. For example, Rule 2 in Table 3-7, which is executed most of the time for both the training and test data in Data II, can be rewritten as

- *IF* $t_1 \geq 0.5939$
- *AND* $t_2 \geq 2.0104$

- *AND* $2.1432 \leq t_3 \leq 4.1393$
- *THEN* $y' = -1.7143 + 1.3519t_3 - 0.3154\ t_3^2$

However, the above derived y' needs to be mapped back to the normal range using the reverse function of Eq.(2.3) to obtain the GNMM simulated dispersion coefficient y:

$$y = \frac{(y' + 1) \times (K_{max} - K_{min})}{2} + K_{min} = 445.05 \times (y' + 1) + 1.90 \quad (3.13)$$

These regression rules could provide environmental managers or field response teams with a rapid assessment tool for identifying values of the longitudinal dispersion coefficients required for the prediction of contaminant spread and concentrations immediately following an accidental spill.

3.5 Discussions

In the following sections, PCA and SOMs are applied to Data I and II to cross-validate the input variables identified by GNMM.

3.5.1 Principal Component Analysis

PCA is a statistical technique used to transform a data space into a smaller space of the most relevant features (Hand, Mannila et al. 2001; Engelbrecht 2002). The aim is to project the original data space onto a linear subspace such

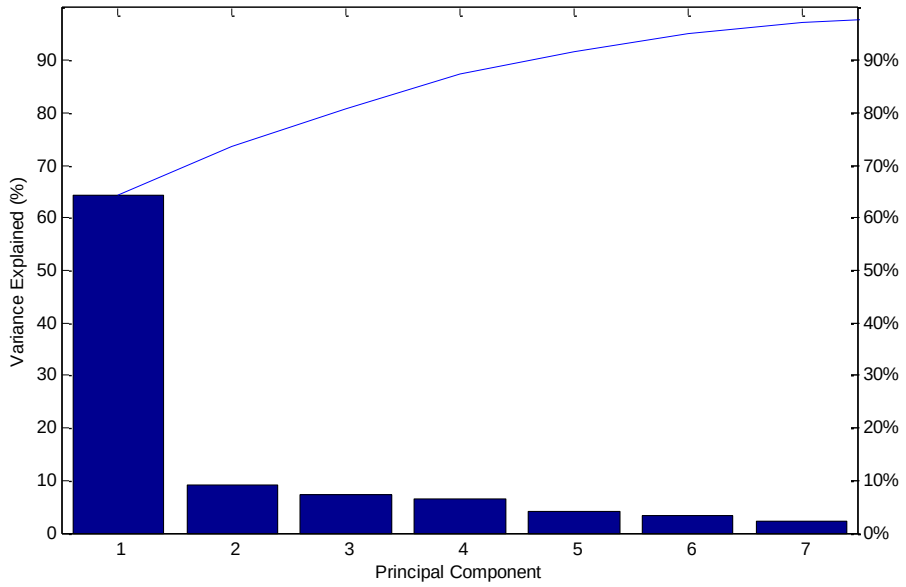


Figure 3-9: Percentage of the first 7 principal components in Data I

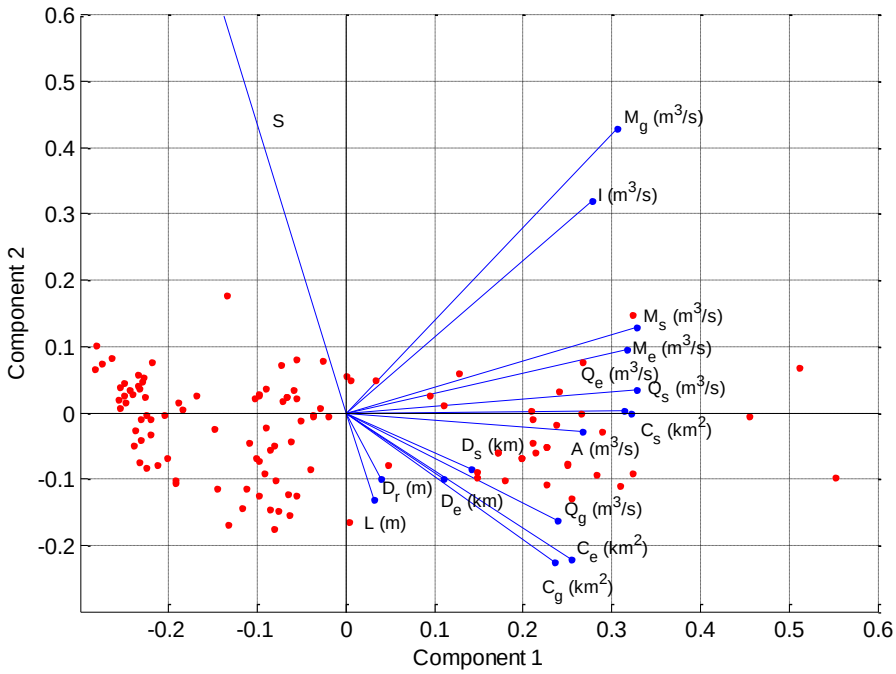


Figure 3-10: Projections of Data I points and variables onto the first two principal components

that the variance in the data is maximally explained within the smaller subspace. Features (or inputs) that have little variance are thereby removed.

The principal components of a dataset are found by calculating the covariance (or correlation) matrix of the data patterns, and by selecting the minimal set of orthogonal vectors (the eigenvectors) that span the space of the covariance matrix. Given the set of orthogonal vectors, any vector in the space can be constructed with a linear combination of the eigenvectors.

Figure 3-9 shows the percentage of the first 7 principal components in Data I. It can be seen that the only clear break in the amount of variance accounted for by each component is between the first and second components. However, the first component by itself explains more than 60% of the variance; with the second components, the variance explained is more than 70%. Therefore, it is reasonable to assert that the first two components can be regarded as being representative of Data I.

The projections of data samples and variables in Data I onto the first two principal components are depicted in Figure 3-10. It is interesting to note that the two variables selected by the GNMM (M_g and I) are clustered together although they are not outstanding in terms of contributions to the first principal component. Thus it may be appropriate to ask as to whether or not GNMM was working properly. It should be noticed that the dimensionality reduction achieved by PCA is realised by preserving as much of the relevant information from the original data as possible. From Figure 3-3, it can been seen that none of the variables apart from the two selected by GNMM has a

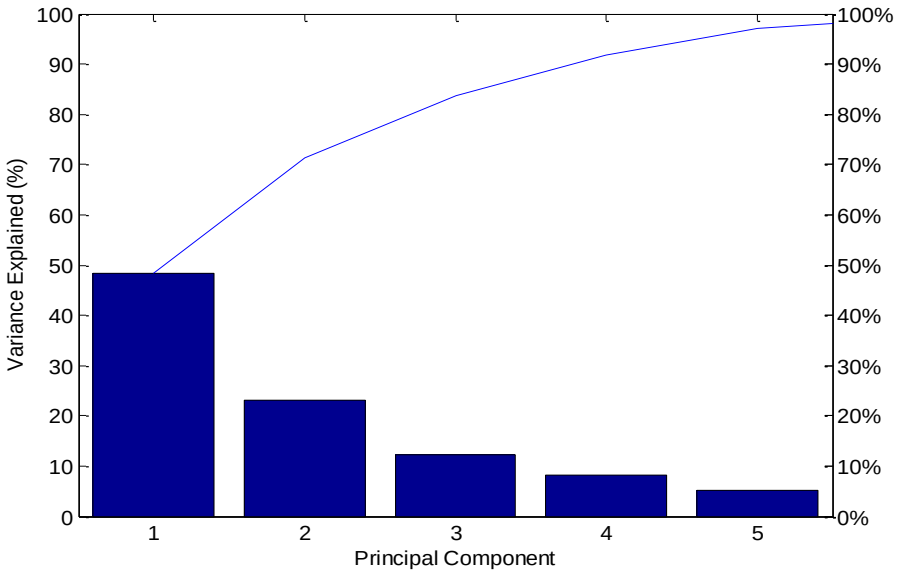


Figure 3-11: Percentage of the first 5 principal components in Data II

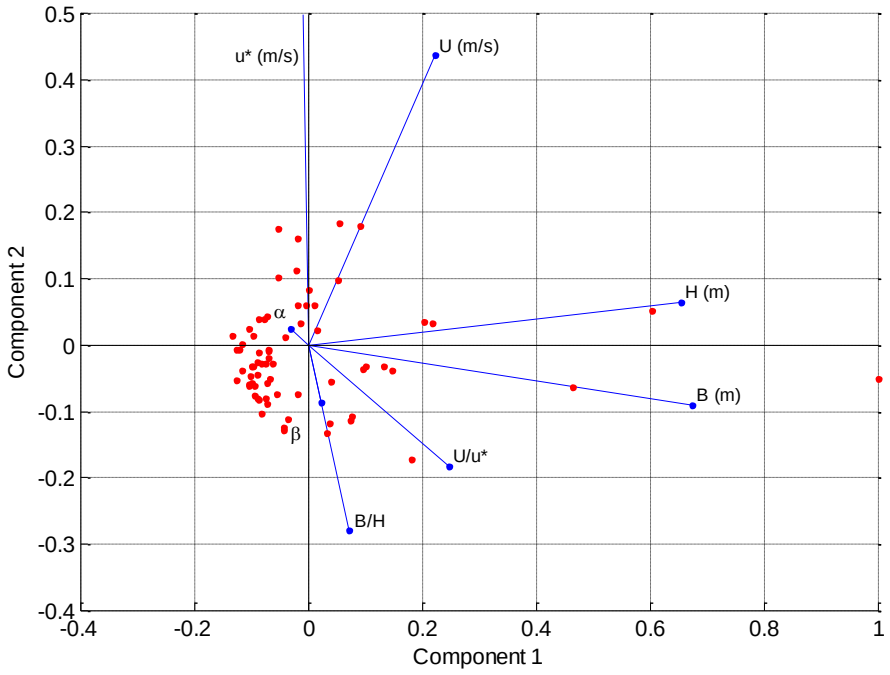


Figure 3-12: Projections of Data II points and variables onto the first two principal components

high *appearance percentage*. Therefore, the judgement is that although the first principal component preserves most information in Data I, it contains

little information about the longitudinal dispersion coefficient. Moreover, the fact that M_g and I are grouped together is the most important finding in Figure 3-10, since it is not necessary for any of the principal components to preserve the information about the longitudinal dispersion.

Figure 3-11 shows the percentage of the first 5 principal components in Data II; while Figure 3-12 illustrates the projections of data samples and variables in Data II onto the first two principal components. It is evident that B and H are grouped together, and they are the most important variables in the first principal component, which counts for around 50% of the total variance. Following a similar analysis to Data I, it can be concluded that the first principal component represents most of the longitudinal dispersion information in Data II.

3.5.2 Self-Organizing Map

The SOM is a multidimensional scaling method to project an input space onto a discrete output space, effectively performing a compression of input space onto a set of vectors. The output space is usually a two-dimensional grid. The SOM uses the grid to approximate the probability density function of the input space, while still maintaining the topological structure of the input space. That is, if two vectors are close to one another in input space, so is the case for the map representation. For a detailed description of SOM, please refer to (Haykin 1994; Engelbrecht 2002). In the present study, SOM analysis is performed using the MATLAB SOM Toolbox 2.0 developed at the Helsinki University of

Technology[6].

The effect of the SOM training process is to cluster together similar patterns, while preserving the topology of the input space. Training results in a set of trained weights with no explicit cluster boundaries. An additional step is required to find these cluster boundaries. One way to determine and visualize the cluster boundaries is to calculate the unified distance matrix (U-matrix), which contains a geometrical approximation of the vector distribution in the map. The U-matrix expresses the distance to the neighbouring vectors for each neuron. Large values within the U-matrix indicate the position of cluster boundaries.

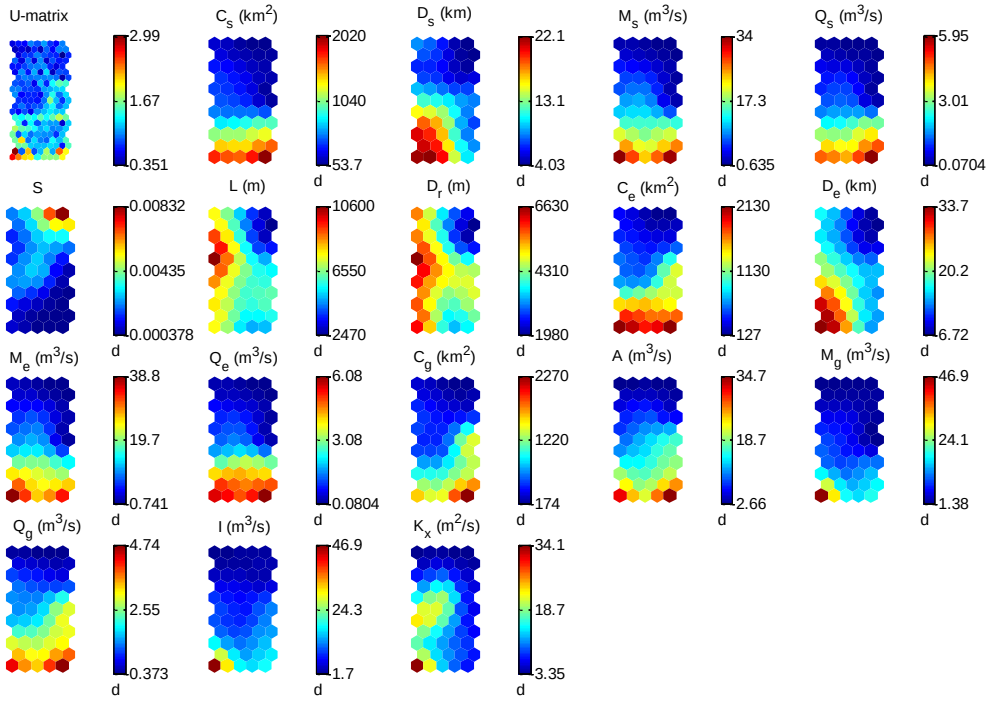


Figure 3-13: SOM analysis of Data I

[6] Laboratory of Computer and Information Science, the Helsinki University of Technology, http://www.cis.hut.fi/projects/somtoolbox/

Figure 3-13 shows the results of SOM analysis of Data I. In the U-matrix, a neuron's colour represents the distance to its neuron neighbours – a low value indicates a small distance between neurons. The 'component planes' show what variable values the neurons have taken. This can be used to look for relationships between individual variables. These plots will have the same number of cells as there were neurons in the map. As each plot is a 'slice' of the output if two plots appear to have a similar distribution of values then this shows the variables to be related. Note that apart from the U-matrix and individual variables, Figure 3-13 also indicates the longitudinal dispersion coefficient (K_x), which is the training target. From Figure 3-13 it may be seen that the pattern presented by the whole dataset (U-matrix) is quite different from K_x, which corresponds to our analysis in Section 3.5.1 Principal Component Analysis. This also indicates that too much irrelevant information is contained in Data I. On the other hand, variables found by GNMM (M_g and I) have similar representations to the training target (K_x). This means that the patterns identified by SOM for these variables (i.e. K_x, M_g and I) share some common feature, which validates our results for input determination for Data I.

Following similar steps as to the above, Figure 3-14 presents the results of SOM analysis for Data II. Unlike Figure 3-13, in Figure 3-14 the whole dataset and the training target have similar distributions. This is illustrated by comparing patterns in the U-matrix and K_x. Furthermore, GNMM's choices of input variables (B, H and U) all present these similar patterns. This indicates

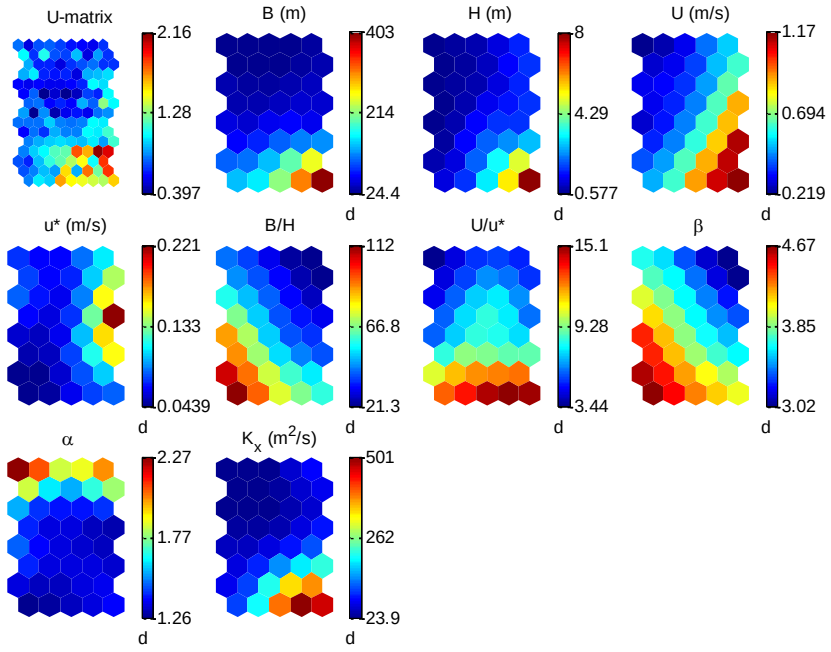


Figure 3-14: SOM analysis of Data II

that the patterns presented by the whole dataset are very similar to those of the training target, which is also closely related to the variables selected by GNMM. This from another point of view proves our variable selection technique.

3.6 Summary

In the current chapter, the GNMM method has been applied to two datasets. The first dataset contains 196 data samples from 27 different rivers measured by the UK Environment Agency (EA). Using variables identified by GNMM (2 out of a total of 49), we achieved an accuracy of longitudinal dispersion coefficient prediction of 0.72 for the coefficient of determination (R^2) and 11.66 for the Root Mean Square Error (RMSE). The second dataset contains 71 sets of measurements from 29 rivers in the United States. GNMM selected 3

variables out of 8. As a result, an R^2 of 0.96 and RMSE of 34.85 were obtained. Rules extracted from trained MLPs were also presented, which demonstrate not only the importance of each rule and but also the distribution of the data.

Through a benchmarking case study, the effectiveness of GNMM has been demonstrated by comparing the results generated by GNMM to those presented in the literature. Compared with conventional methods that provide longitudinal dispersion prediction (e.g. Eq.(3.3) and Eq.(3.4)), GNMM as a data driven approach needs no *a priori* knowledge. Although *a priori* knowledge is widely used in many ANN applications, they are dependent on expert knowledge and hence very subjective and case dependent. This is particularly true for complex problems, where the underlying physical mechanism is not fully understood. Furthermore, GNMM is adaptive. This means that when new data samples are applied to the system, the system is capable of self-learning and thus adjusting its results and improving prediction accuracy. Another advantage of GNMM over conventional methods is that, due to its ANN nature, it can approximate virtually any function with any desired accuracy without making assumptions with regard to stream geometry or flow dynamics.

In order to validate the effectiveness of GNMM's input determination method, we also provide an insightful analysis of the technique that uses GAs as an ANN input variable optimization tool in the context of longitudinal dispersion coefficient prediction. This is achieved by a detailed comparative study of the

GNMM's input determination process. Moreover, PCA and SOM analysis are performed to cross-validate the results of GA input variable selection.

References

Boxall, J. B. and I. Guymer (2007). "Longitudinal mixing in meandering channels: New experimental data set and verification of a predictive technique." Water Research 41(2): 341-354.

Cannas, B., A. Fanni, et al. (2006). "Data preprocessing for river flow forecasting using neural networks: Wavelet transforms and data partitioning." Time Series Analysis in Hydrology 31(18): 1164-1171.

Deng, Z. Q., V. P. Singh, et al. (2001). "Longitudinal dispersion coefficient in straight rivers." Journal of Hydraulic Engineering 127(1): 919-927.

Engelbrecht, A. P. (2002). Computational intelligence : an introduction. Chichester, England ; Hoboken, N.J., J. Wiley & Sons.

Fischer, H. B., E. J. List, et al. (1979). Mixing in inland and coastal waters. New York, Academic Press.

Guymer, I. (1999). A National Database of Travel Time, Dispersion and Methodologies for the Protection of River Abstractions. Environment Agency R&D Technical Report P346.

Hagan, M. T., H. B. Demuth, et al. (1996). Neural network design. Boston, PWS Pub.

Hand, D. J., H. Mannila, et al. (2001). Principles of data mining. Cambridge, Mass., MIT Press.

Haykin, S. S. (1994). Neural networks : a comprehensive foundation. New York, Maxwell Macmillan International.

Jobson, H. E. (1997). "Predicting travel time and dispersion in rivers and streams." Journal of Hydraulic Engineering 123(11): 971-978.

Kashefipour, S. M., R. A. Falconer, et al. (2002). "Modeling longitudinal dispersion in natural channel flows using ANNs." River Flow 2002: 111-116.

Lin, C. T. and C. S. G. Lee (1996). Neural fuzzy systems: a neuro-fuzzy synergism to intelligent systems. NJ, USA, Prentice-Hall, Inc. Upper Saddle River.

Maier, H. R. and G. C. Dandy (1998). "Understanding the behaviour and optimising the performance of back-propagation neural networks: an empirical study." Environmental Modelling and Software 13(2): 179-191.

Piotrowski, A., P. M. Rowinski, et al. (2006). Assessment of longitudinal dispersion coefficient by means of different neural networks. 7th International Conference on Hydroinformatics, Nice, FRANCE.

Rowinski, P. M., A. Piotrowski, et al. (2005). "Are artificial neural network techniques relevant for the estimation of longitudinal dispersion coefficient in rivers?" Hydrological Sciences Journal 50(1): 175-187.

Rutherford, J. C. (1994). River mixing. Chichester, Wiley.

Seo, I. W. and T. S. Cheong (1998). "Predicting longitudinal dispersion coefficient in natural streams." Journal of Hydraulic Engineering 124(1): 25-32.

Tayfur, G. (2006). "Fuzzy, ANN, and regression models to predict longitudinal dispersion coefficient in natural streams." Nordic Hydrology 37(2): 143-164.

Tayfur, G. and V. P. Singh (2005). "Predicting longitudinal dispersion coefficient in natural streams by artificial neural network." Journal of Hydraulic Engineering 131(11): 991-1000.

Taylor, G. (1954). "The Dispersion of Matter in Turbulent Flow through a Pipe." Proceedings of the Royal Society of London.Series A, Mathematical and Physical Sciences 223(1155): 446-468.

Tsaih, R. and L. Chih-Chung (2004). The layered feed-forward neural networks and its rule extraction. Advances in Neural Networks - ISNN 2004. International Symposium on Neural Networks, Dalian, China, Springer-Verlag.

Wallis, S. G. and J. R. Manson (2004). "Methods for predicting dispersion coefficients in rivers." Water Management 157(3): 131-141.

Yang, J., E. L. Hines, et al. (2007). Prediction of Longitudinal Dispersion Coefficient in Rivers using Variables Identified by Genetic Algorithms. The Fifth International Symposium on Environmental Hydraulics (ISEH V). Tempe, Arizona.

Yang, J., E. L. Hines, et al. (2008). A Genetic Algorithm-Artificial Neural Network Method for the Prediction of Longitudinal Dispersion Coefficient in Rivers. Advancing Artificial Intelligence through Biological Process Applications. A. Porto, A. Pazos and W. Buño, Idea Group Inc.: 358-374.

Chapter 4 Channel Selection and Classification of EEG Signals

4.1 Introduction

In Chapter 3 we applied GNMM to civil engineering datasets, where we demonstrated GNMM's implementation details and also cross-validated GNMM's input selection results using PCA and SOM. In this chapter, we will apply GNMM to two Electroencephalogram (EEG) classification problems. Compared to other naturally occurring dynamic patterns, EEG activity is not only at least as complex, it has the added advantage that an almost unlimited number of highly controlled variants can be created in an easy, cost-effective, and straight-forward manner, by simply setting different task parameters and giving different task instructions to the human participant. Therefore, EEG measures are an ideal testing ground for developing novel DM techniques.

There are two datasets used in the current chapter, and the body of the chapter splits into two parts accordingly. Firstly we will make use of a well studied dataset (denoted by Data III) – 64-channel electrocorticography (ECoG)

data for a two-class motor imagery, which have previously been used to perform channel selection and pattern classification tasks (Lal, Hinterberger et al. 2005); the other data (i.e. Data IV) are from a speeded 2-alternative forced-choice manual response task, collected using a 32-channel EEG system (Schlaghecken, Blagrove et al. 2008; Schlaghecken, Klapp et al. 2009).

4.2 Background

An EEG based BCI provides a possible means to implement a communication channel between the human brain and a computer. Patients who suffer from severe motor impairments (e.g., late stage of Amyothrophic Lateral Sclerosis (ALS), severe cerebral palsy, head trauma and spinal injuries) may use such a BCI system as an alternative form of communication through mental activity (Guger, Schlogl et al. 1999). Most human BCIs are based on extracranial EEG. Compared with invasive EEG (e.g., ECoG), this presents a great advantage in that it does not expose the patient to the risks of brain surgery. On the other hand, however, invasive EEG signals contain less noise.

According to Besserve et al. (2007), depending on the spatial extent of the physiological phenomenon under investigation, the ongoing EEG signals can be divided into two families: local or long range. Local measurements generally provide a measure of task-related activity picked up at a single sensor or electrode. By contrast, measurements of long range interactions quantify the coupling between signals detected at two distinct sensors, possibly revealing

information transfer between two distant neural ensembles. One of the fundamental technical difficulties with using EEG measures to classify neural activity results from spatiotemporal filtering, which limits the signal/noise ratio of the time series and blurs the localization of the relevant neural generators (Sanchez and Principe 2007).

Conventional neuroimaging analysis correlates external regressors such as task condition with activity in specific areas of the brain. PR may be viewed as an inversion of this methodology and instead predicts the external stimulus based on neuroimaging data. Unlike conventional analyses, these pattern-based analyses take into account the full spatial pattern of brain activity rather than concentrating on specific regions (Wandell 2008), and represent a new way of looking at neuroimaging data. A recent review by Haynes and Rees (2006) discusses several studies that have successfully used statistical PR to decode a person's current thoughts from their brain activity alone. They concluded that it was possible to correctly identify which object a subject is currently viewing, even when several alternative categories are presented. Lotte et al. (2007), presenting an exhaustive review of the algorithms already used for EEG-based BCI, conclude that ANNs are the classifiers which are most frequently used in BCI research.

ANNs as a PR technique are well established in BCI research and also have numerous successful applications (Shuter, Hines et al. 1994; Robert, Gaudy et

al. 2002; Robert, Karasinski et al. 2002; Singh, Li et al. 2007). For example, Shuter et al. (1994) proposed a ANN-based system to process EEG data for the monitoring of the depth of awareness under anaesthesia. They analyzed the awareness states of patients undergoing clinical anaesthesia based on the variations in their EEG signals using a three-layer BP network. The network accurately mapped the frequency spectrum into the corresponding awareness states for different patients and different amounts of anaesthetics. In a recently published paper, Singh et al. (2007) investigated EEG data using a combination of common spatial patterns (CSP) and MLPs to achieve feature extraction and classification. Event-related synchronization/desynchronization (ERS/ERD) maps were also used to investigate the spectral properties of the data. As a result, they achieved an accuracy of 97 % for the training data and 86 % for the test data. Robert et al. (2002) have reviewed more than 100 ANN applications concerned with EEG signal processing and classified these BCI-related applications into two categories: prediction and classification. The prediction class is usually concerned with estimating the possibility of predicting the side of hand movements (left or right) using EEG records prior to voluntary right or left hand movements. In some studies classification rates were not very high (from 51 to 83%). However, classification accuracies as high as 85-90% were achieved in others. In the classification category, neural network-based systems were trained to classify movement intention of left and right index finger or the foot using EEG autoregressive model parameters. A correct recognition rate of 80% was achieved in some applications. Overall,

the future for neural network-based BCI systems is very promising.

However depending on the application one of the drawbacks of conventional ANNs is that there is no explicit input optimization mechanism. As such, all available signals or features are typically fed into the network to accomplish the PR task(s). This input optimization problem also exists when the NN input data are signals collected by EEG electrodes. In terms of EEG classification, signals can be very noisy and contaminated by various motion artifacts produced at certain electrodes. The data acquisition task will also be made much more efficient if the electrodes are only a minimum subset of all available positions. In addition, algorithms for channel selection can identify suitable recording sites for individual subjects even in the absence of prior knowledge about the mental task. In fact, researchers have investigated various methods to optimize EEG channels. For example, Tian, Erdogmus et al. (2005) proposed a filter-based approach for EEG channel selection using mutual information (MI) maximization. Lal, Hinterberger et al. (2005) recently introduced a support vector feature selection method based on Recursive Feature Elimination (RFE) for the special case of EEG data.

Unlike conventional ANNs which utilize all available EEG channels and let the ANN adjust its internal connections, GNMM only concentrates on a subset of available channels. This subset (i.e. selected EEG channels) is optimally identified for dimensionality reduction using GAs. In this way, we combine the

merits of both conventional neuroimaging analysis and PRs.

4.3 Data III – Two-Class Motor Imagery

The intracranial ECoG recording is explicitly selected to validate the technique developed as it is expected to contain higher quality brain signals with low values of impedances. The dataset, which was used in the BCI competition III (Blankertz, Muller et al. 2006), comprises of a large number of labelled trials which makes it advantageous for evaluation of performance measures for the technique.

4.3.1 Experiment Setup

The experiments were performed in the department of epileptology of the University of Bonn (Lal, Hinterberger et al. 2005). During the experiment, a subject had to perform imagined movements of either the left small finger or the tongue (Figure 4-1). The time series of the electrical brain activity was picked up during these trials using an 8×8 ECoG platinum electrode grid which was placed on the contralateral (right) motor cortex. The grid was assumed to cover the right motor cortex completely, but due to its size (approx. 8×8cm^2) it also partly covered surrounding cortex areas. All recordings were performed with a sampling rate of 1000Hz. After amplification, the recorded potentials were stored as microvolt values. Every trial consisted of either an imagined tongue or an imagined finger movement and was recorded for 3 seconds duration. To avoid visually evoked potentials being reflected by the data, the

recording intervals started 0.5 seconds after the visual cue had ended. The whole dataset consists of 278 trials for training/validation and 100 trials for testing respectively. Within each trial, there are 3000 data points per channel (i.e. electrode) and a total of 64 channels available. The whole dataset is available in Matlab format from the BCI competition web site[7].

4.3.2 Pre-processing

A major difficulty in the processing of EEG data comes from the usually very large size of the dataset due to the relatively high sampling frequency. To reduce the data size we apply a least square (LS) approximation on a single trial basis. In fact, partial least square (PLS) has been used as a regression method to extract spatiotemporal patterns from EEG signals (Martínez-Montes, Valdés-Sosa et al. 2004; Kovacevic and McIntosh 2007). The LS technique used in the current work is the linear LS approximation of the EEG signal over a

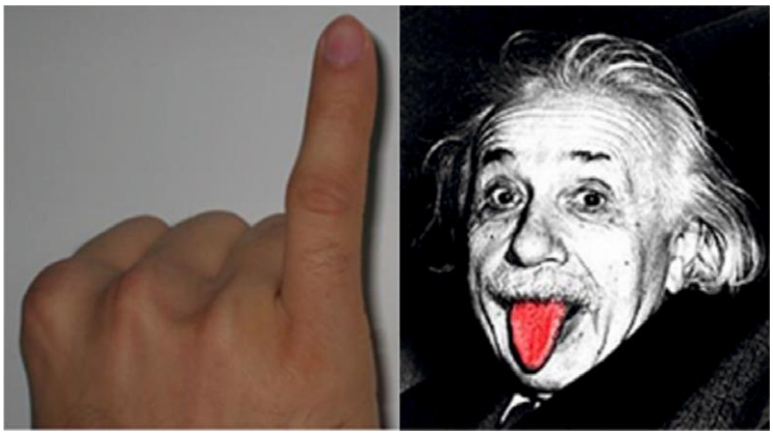


Figure 4-1: Data III – two-class imaginary movements (Adapted from Lal, Hinterberger et al. 2005)

[7] BCI Competition III, Intelligent Data Analysis Group, Fraunhofer FIRST, http://ida.first.fraunhofer.de/projects/bci/competition_iii/.

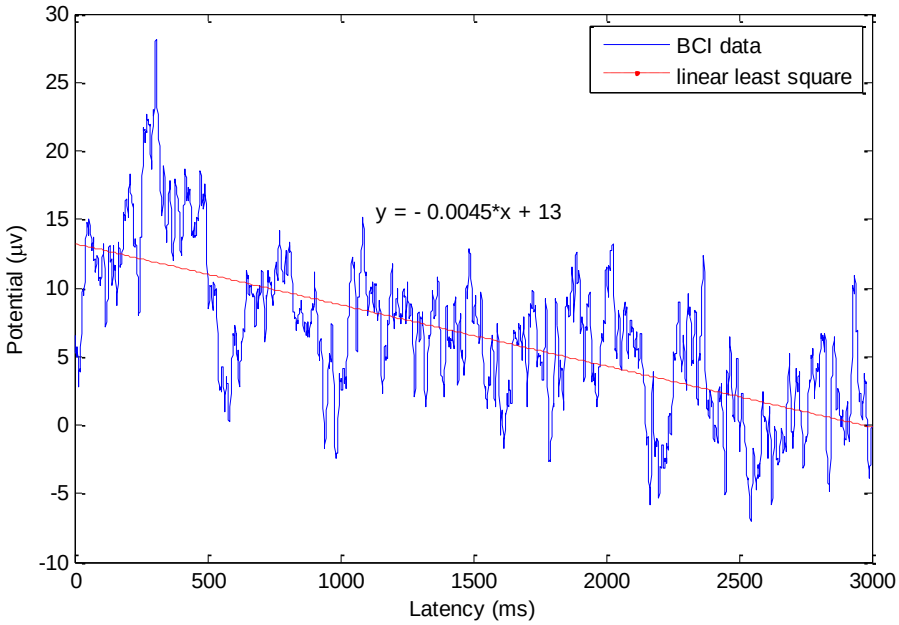


Figure 4-2: Least square approximation for a signal segment in Data III

specific time period. Let $x_{(t,\ b)}$ be the EEG signal measurements on channel b at time t. Thus, a linear LS approximation for EEG signals on this particular channel for a single trail may be formed thus:

$$x = pt + q \tag{4.1}$$

Also, the derivative of Eq. (4.1) gives:

$$dx / dt = p \tag{4.2}$$

which is the slope of the linear LS approximation. This value p is indicative of the changes in the signal for each channel during a specific time slot.

A linear LS approximation was performed on Data III on a single trial basis.

Figure 4-2 shows the original data and its LS approximation for the signal obtained from one electrode over an imaginary movement. It is clear that by doing LS approximation, the data size is greatly reduced while significant information (i.e. signal changing rate and direction over a specific time window) still remains. As a result of pre-processing, the dimension of Data III was reduced to 278×64 and 100×64 for the training/validation and testing sets respectively. Target values of 1 and −1 were used for imaginary finger and tongue movement.

4.3.3 Channel Selection

During channel selection, when a GA's chromosome is being evaluated, a total of 250 trials of the training set (~90%) were randomly selected for training and the remaining 28 kept for validation purposes. The MLPs used for evaluating purposes were configured so that the number of hidden neurons in the only

Table 4-1: Configuration of GAs for Data III channels selection

Case	N_p	N_g	N_e
1	400	400	20
2	200	200	50
3	400	100	50
4	400	200	100
5	600	100	50
6	200	100	200
7	400	100	150

hidden layer equals four when the channels being evaluated are greater than four and will otherwise be the same as the number of input neurons. An output layer of only one neuron was used throughout channel selection and the final pattern classification.

Seven iterations of the GA produced the best channel combinations to give the least error. The various GA configurations are shown in Table 4-1. It was observed that there existed 10 channels which appeared in more than 90% of all the generations. Hence these were specifically selected as the input data for the final classification. The channels being selected are [7 12 17 21 22 45 46 47 54 59], as in Appendix F. The other 54 less informative channels were thus removed from further analysis.

4.3.4 Classification Results

The subset of only 10 channels was fed into a three-layer MLP and trained using the LM algorithm to perform the final classification. The number of neurons in the hidden layer was increased to ten to maximize the classification rate. Furthermore, ten-fold cross validation was introduced to try to improve the generalization. As a result of five runs, the lowest RMSE value was calculated to be 0.4552, and the mean is 0.6382 and standard deviation is 0.165. Defining the coefficient of determination (R^2) as

$$R^2 = 1 - \frac{\sum_{i=1}^{a} (y_i - v_i)^2}{\sum_{i=1}^{a} y_i^{\,2}} \tag{4.3}$$

where y and v are the actual and predicted target class values, R^2 for the training set was found to be 79.28% using the best trained network (i.e. lowest RMSE). Target and predicted values for these 278 training/validation trials are shown in Figure 4-3. Note that in Figure 4-3 shadowed areas denote the training target value ±1±RMSE; data points represent the actual value produced by the MLP model with 10 input channels.

Treating the mid-point of the two target classes, in which case is '0', as the dividing point of those predicted values, the MLP model with only ten channels achieved an average classification rate of 83.39% with a standard deviation of 18.58. The above results compare favourably with those obtained by Lal, Hinterberger et al. (2005) on the same data, where they used RFE for channel selection and SVMs for pattern classification and achieved a minimum error of about 25.7% (i.e. an accuracy of 74.3%). Moreover, taking into account only those predicted values that fall into the range of the target ± RMSE (i.e. shadowed areas in Figure 4-3), our model achieved an average accuracy of 72.30%; with the positive class having a slightly higher rate.

Training an MLP with the same number of hidden neurons and configurations (e.g. learning rate, training algorithm etc.) but using all available 64 channels

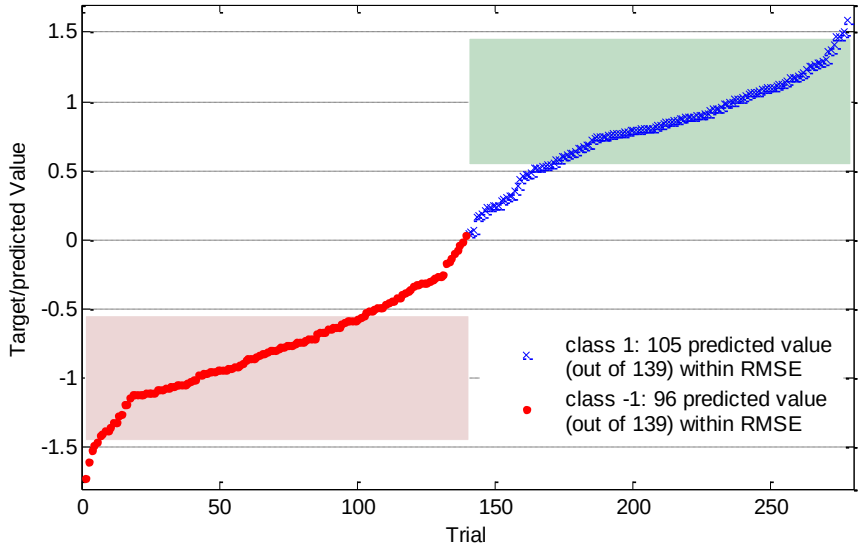


Figure 4-3: Target and predicted values for Data III training/validation set

Table 4-2: Classification results for Data III, which shows the results for training/validation and testing subset respectively. RMSE and $R^2$and are for the best trained network. Classification rate is calculated over five runs

		RMSE	R^2	Classification rate
Train/validation	All channels	0.2305	94.69%	96.12%±7.52
Data (278 trials)	10 selected channels	0.4552	79.28%	83.39%±18.58%
Testing data	All channels	1.7984	*	58.04%±9.36
(100 trials)	10 selected channels	1.3329		50.81%±4.74%

'*' means negative coefficient of determination, and hence were ignored.

for five times, we obtained a better classification rate: the lowest RMSE decreased to 0.2305 and R^2 increased to 94.69%, as seen in Table 4-2. The classification rate for five runs is 96.12%±7.52%. This is because one advantage of using MLPs is that the internal connection (i.e. weights) can adjust itself in a way that outperforming channels gain in weight while less-influential ones lose.

However, the trade-off is that MLPs trained using all channels have a lower generalization. This was ascertained through the fact that when classifying the 100 testing trials, the best trained MLP using the 10 selected channels achieved a lower RMSE (1.3329 vs. 1.7984). An interesting point is that, in terms of the testing set, the classification rate is higher for MLP trained using all channels than using selected channels only. This is because although some predicted values are on the 'correct' side of the axis, they scatter far from the RMSE area. This on the other hand, demonstrates the generalization ability of the MLP trained using selected channels.

4.4 Data IV – Response Priming Paradigm

Data IV were collected from a speeded 2-alternative forced-choice manual response task using a 32-channel EEG system (Schlaghecken, Blagrove et al. 2008; Schlaghecken, Klapp et al. 2009).

4.4.1 The Experiment

In a 2-alternative speeded choice reaction time (RT) task, participants had to execute a left-hand or right-hand button-press in response to briefly presented arrow stimuli pointing to the left or right. Each arrow target was preceded by an arrow prime, which could point either in the same or in the opposite direction as the target. These primes were visually 'masked' and therefore easy to ignore; see Schlaghecken and Eimer (2006) for a detailed description of the masked prime procedure. Furthermore, target arrows were flanked by

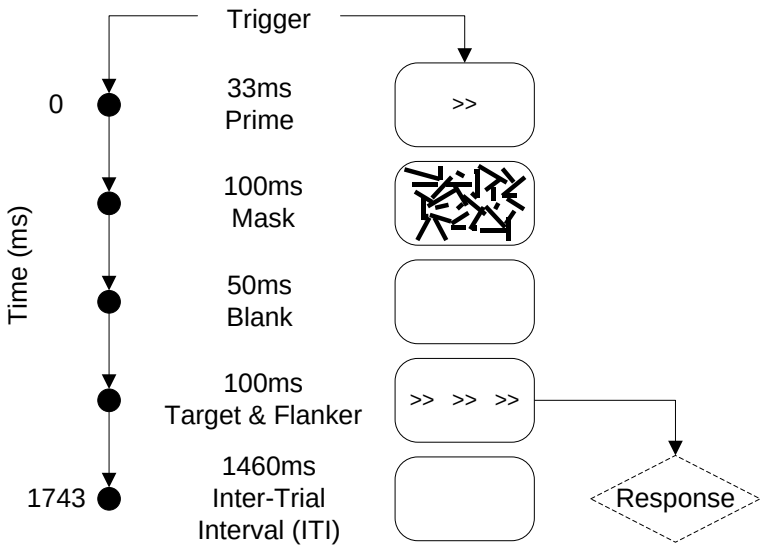


Figure 4-4: Schematic representation of stimulus material and trial structure in Data IV experiments

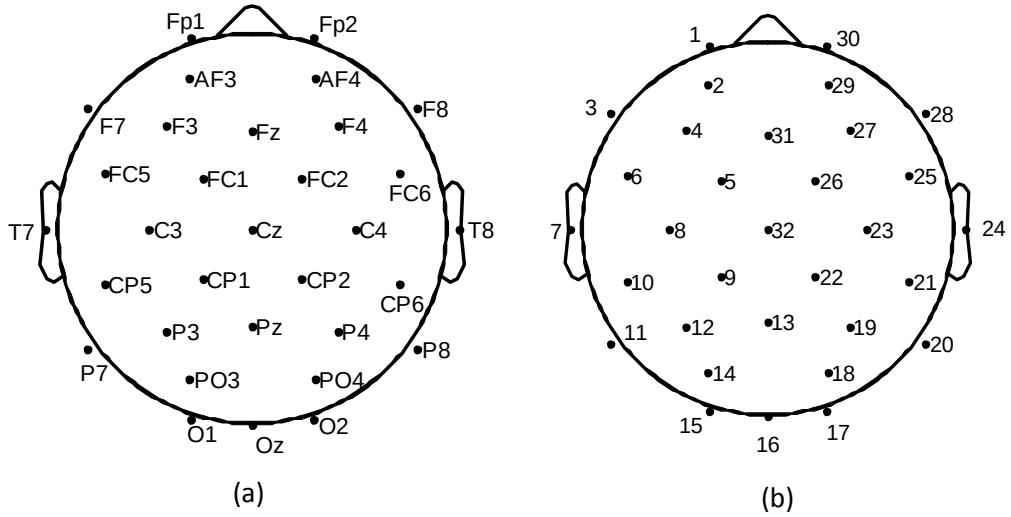


Figure 4-5: Position of EEG electrodes used in Data IV experiments arranged by: (a) position and (b) number

response irrelevant (to-be-ignored) distractor stimuli associated with either the same response as the target or the opposite response, which added a certain level of difficulty to response selection and execution (Eriksen flanker task (Eriksen and Eriksen 1974)). The interval from one prime onset to the next was fixed at 1743 ms. The experimental setup is illustrated in Figure 4-4.

The experiment consisted of 96 randomized trials per block, and 10 blocks per participant. EEG signals were measured using the BioSemi[8] ActiveTwo 32-channel EEG system. The electrode arrangement is shown in Figure 4-5. The EEG was sampled at a frequency of 256 Hz. The recording data for all runs was concatenated and converted into the BDF format (Schlogl 2003).

4.4.2 Pre-processing

The original data were triggered using the EEGLAB[9] Matlab toolbox. The pre-processing for Data IV involved a multi-time-windows LS approximation over a single trial. In order to trace the development of response-related EEG signals over time, the trial period was divided into 7 intervals spanning 250 ms each (denoted by INT1 through to INT7). Additionally, analysis was conducted on

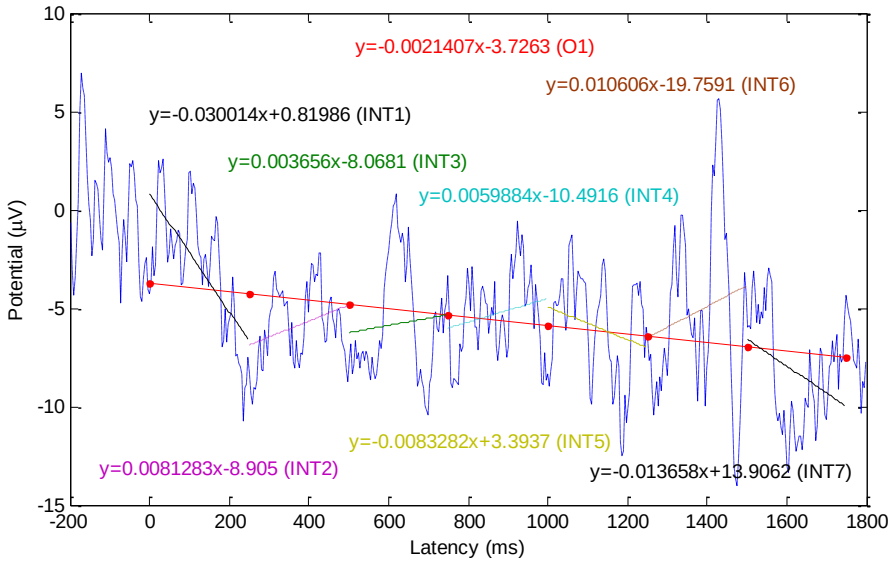


Figure 4-6: EEG signal of channel Cz for the first epoch of Data IV event No.1 and its LS approximations across different time windows

[8] BioSemi, http://www.biosemi.com/products.htm.
[9] Swartz Center for Computational Neuroscience, University of California San Diego, http://sccn.ucsd.edu/eeglab/.

one overarching time window spanning the whole length of a trial (denoted by O1). Consequently, 8 features/parameters are extracted from each EEG channel for each single trial. Figure 4-6 shows the EEG signal recorded on channel Cz for the first epoch of event No.1 and its LS approximations during different time slots. 8 LSs were calculated in total: 1 on the overall trial period and 7 others, each of which was calculated over an interval of 250 ms starting from latency = 0.

A specific difficulty that lies in Data IV is that the dataset can be divided into eight categories: 2 target directions × 2 prime directions × 2 flanker directions. Each of these can then be further divided into two sub-categories: correct or incorrect responses. For the current purposes, the data were split according to two criteria: (a) left or right hand response, and (b) correct or incorrect response, resulting in four classification targets: right hand incorrect (Class1), left hand incorrect (Class2), right hand correct (Class3), and left hand correct (Class4).

These pre-processed data were fed into GNMM, and the effectiveness of different time windows for channel selection and pattern classification was investigated.

4.4.3 Channel Selection and Pattern Classification

GAs are configured to run four times to explore different combinations of

input channels for each of those 8 sets until distinctions were evident between these EEG channels for each of the eight datasets. Investigating *appearance percentage* distribution for each channel yields not only the importance of each channel in the final pattern classification, but also the energy distribution around the scalp. The *appearance percentage* of each channel in the four GAs for each of those 8 sets is illustrated in Figure 4-7. Overall, the appearance distribution among channels is relatively smooth. In agreement with the to-be-classified phenomenon, manual motor response, the channels located near the hand-area of the left and right motor cortices (here, channels Cz, C3 and C4) are the most likely, whereas occipital (i.e., visual) and fronto-polar channels (Oz, O1/2, Fp1/2) are the least likely to be selected (see Figure 4-5 and Figure 4-7). Specifically, it seems that there are some connections between O1 (the overall signal changing rate) and INT6 (the changing rate just before the end of the trial). In these two cases, the area evenly distributed around the conceptual horizontal line linking the two ears is more active than the other areas. On the other hand, the distribution of INT3 is relatively sparse. Since we know that EEG signals recorded on adjacent scalp locations are not supposed to be very distinct, being sparse suggests that INT3 may not be an appropriate feature for the whole EEG signal.

In order to select the most frequently appearing channels for all 8 parameter subsets (INT1-7 and O1), the selection criterion was set to at least 80% appearance. However, another consideration is that the number of channels

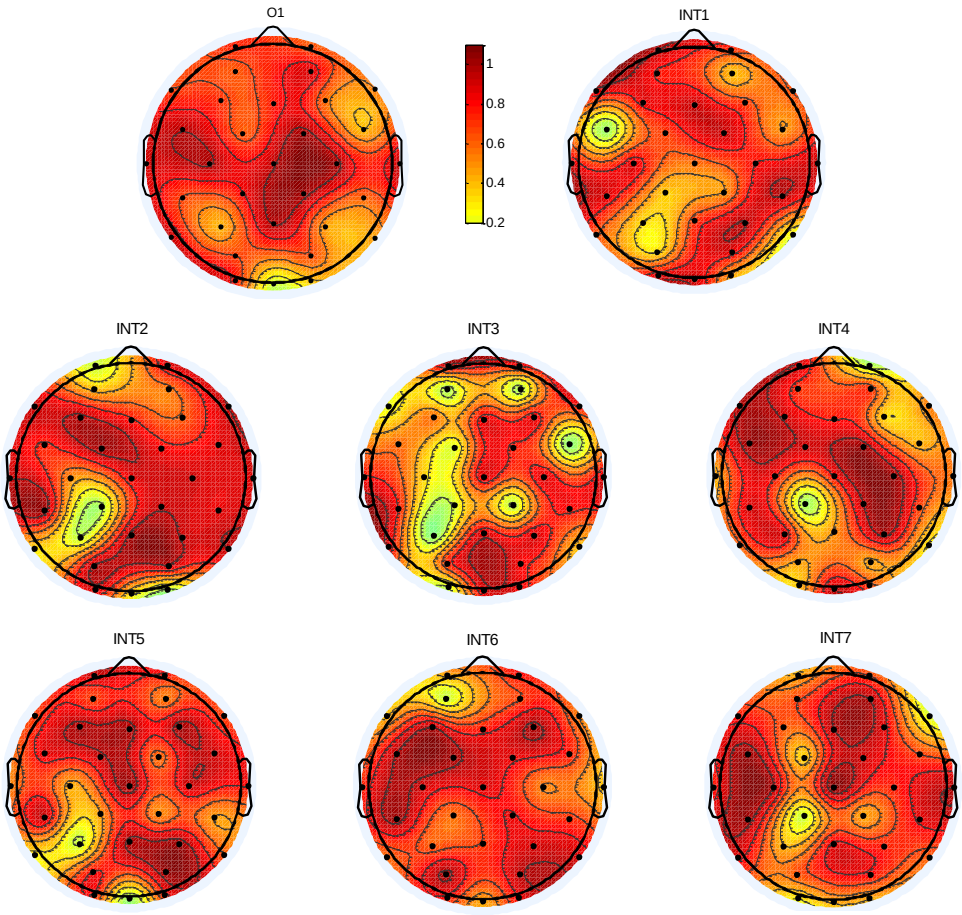


Figure 4-7: Appearance percentage distribution around the scalp for Data IV subsets. Colour indicates chances of a particular channel being selected by GAs for final classification – the darker the better

selected for each parameter subset should be the same or at least similar, in order to make comparisons possible. Therefore, the selection criterion was increased for individual parameter subsets until for each only 12 channels (13 in the case of INT1 as a result of the fact that two channels' *appearance percentage* appear to be exactly the same) were selected. Most of these appear more than 90% in the winning chromosomes (see Table 4-3).

Feeding the channels selected in Table 4-3 into MLPs and training with the LM algorithm, we are able to compare the classification accuracy between

different datasets, as shown in Figure 4-8. It is evident that correct responses (Class3 and 4) are more successfully classified. An interesting point here is that the overall classification rate mimics the trends of Class3 and 4, which share a similar pattern, and is inversely proportional to the rate of Class2. Although

Table 4-3: EEG channels selected for each subset in Data IV

Subset	Channels Selected	Criteria (appearance percentage)
O1	6 7 8 9 11 13 22 23 24 26 29 32	>=0.81
INT1	1 3 7 8 10 15 16 18 19 21 24 26 31	>=0.90
INT2	3 4 5 7 10 13 14 18 19 20 22 28	>=0.92
INT3	1 11 13 16 17 18 19 23 24 27 31 32	>=0.93
INT4	1 3 4 5 6 10 12 16 19 22 23 26	>=0.85
INT5	1 4 5 10 13 18 19 23 25 27 31 32	>=0.88
INT6	4 5 6 8 10 14 18 19 20 26 27 32	>=0.82
INT7	6 8 10 11 13 21 24 26 27 29 31 32	>=0.80

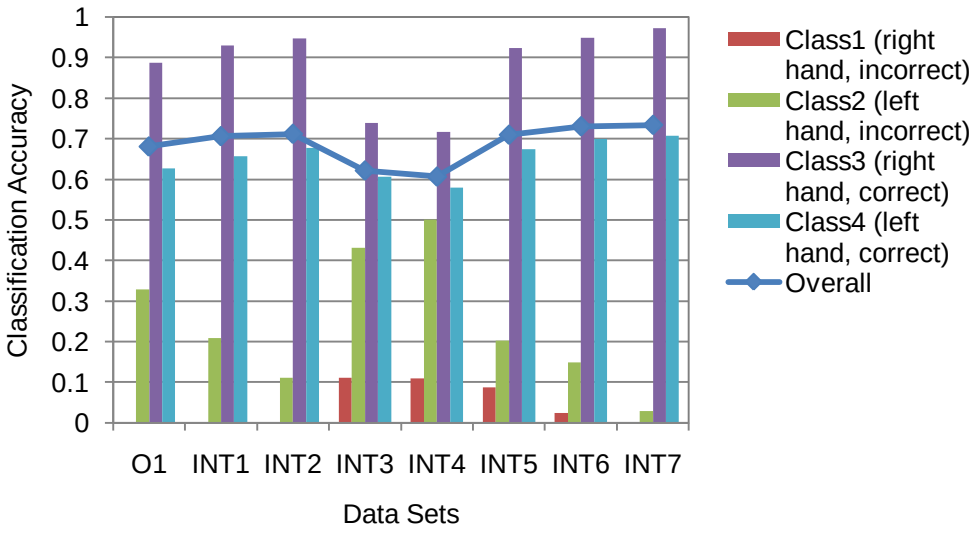


Figure 4-8: Classification accuracy for different subsets in Data IV

presenting sparse patterns, INT3-5 accounts for the most successful approximations for the classification of Class1. Another observation is that although INT2 and INT7 do not have any classification accuracy for Class1 and very low for Class2, their overall rates are among the highest. The discrepancy between high classification accuracy for correct responses and low accuracy for incorrect responses most likely is due to the fact that incorrect responses only constituted 13% of the overall dataset. As a consequence, insufficient information for the ANN to achieve reliable classification was present.

It should be noted that RTs (time from trial onset to the depression of a response button beyond a certain threshold) in this task was approximately 500-550 ms. Therefore, the high classification accuracy in INT 1 and 2 (about

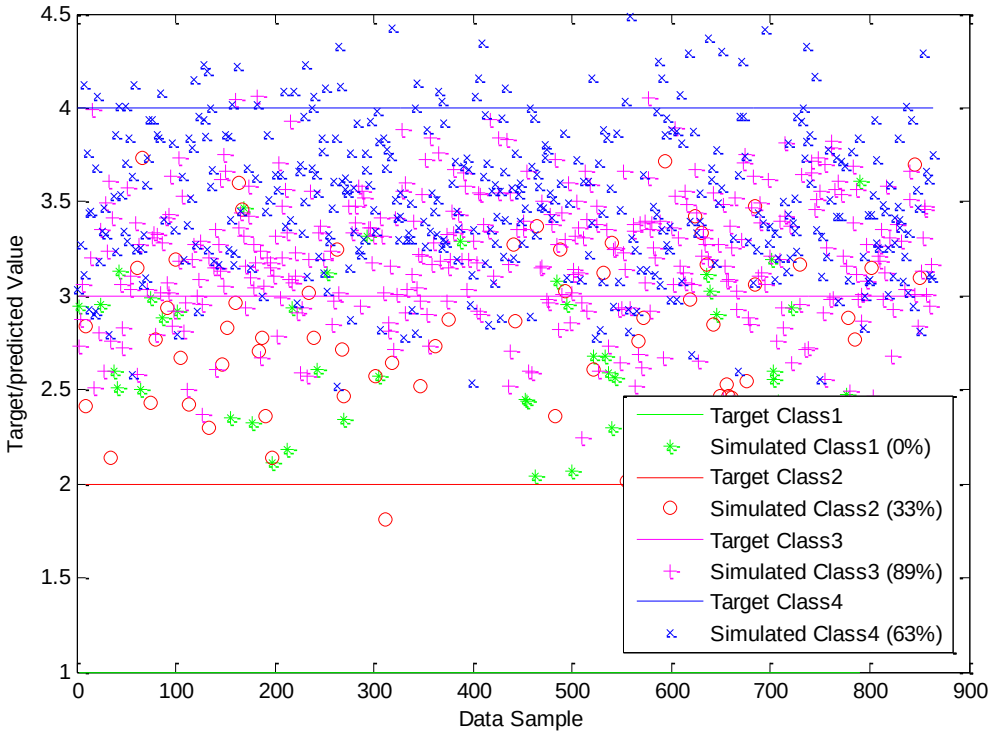


Figure 4-9: Target and predicted values for subset O1

80% for correct responses) reflects the rate of correct predictions of a yet-to-be-executed response. Furthermore, correct error classification was achieved with more than chance frequency (25%) only in the 500-1000 ms time-windows, that is, after an incorrect response had been executed. In line with recent neurophysiological studies (Vocat, Pourtois et al. 2008), this indicates that the most distinguishing feature of response errors lies in cognitive post-error processes, not in preceding 'erroneous' cognitive processes.

Figure 4-9 shows the target and predicted values for subset O1. It is evident that ANNs classify Class3 and 4 quite well. In spite of the fact that no Class1 instances were correctly identified, most of them were classified as Class3. Given that both Class1 and 3 denote right hand movement, this phenomenon suggests that the system can properly identify right hand movement regardless of outside stimuli. However, Class2, which present a low classification rate, were also mostly classified as Class3. Recall that datasets that have the highest rate for Class2 (INT3 and 4, see Figure 4-8) were very sparse in terms of channel appearance distribution (Figure 4-7), and the accuracy rate for Class4 is 63%, it can be concluded that the EEG signal for the left hand movement for this particular patient is more complicated.

4.4.4 Rule Extraction

Rule extraction was not discussed for Data III as in that case the data was obtained from a single subject with specific channel locations; while in the

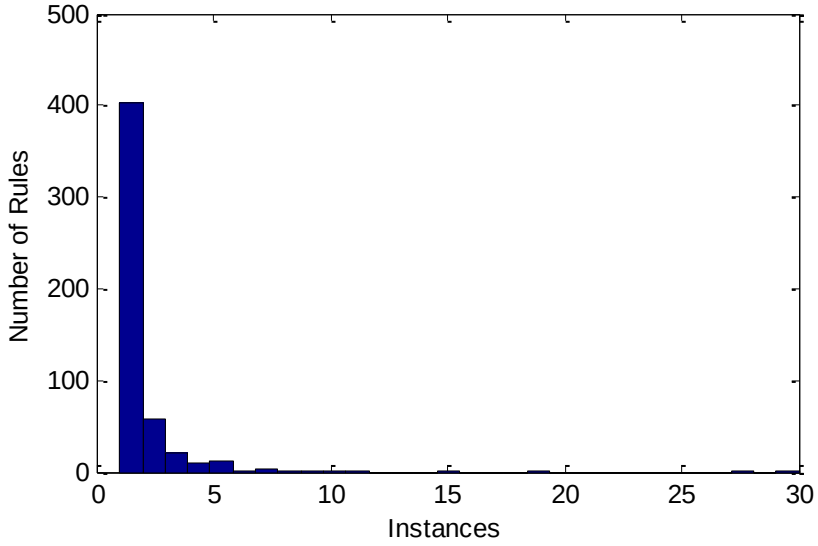


Figure 4-10: Histogram of extracted regression rules from Data IV subset O1

case of Data IV channel locations have been widely studied and rules can be tested and extended to a wider range of participants. Taking the MLP trained using O1 for instance, a total of 516 regression rules were extracted from subset O1. O1 is being used here instead of other datasets because the time interval for this set is much larger – it includes the whole trial period rather than its segments. As a result, it is feasible in practice and provides more error-tolerance. Histogram of rules extracted from O1 can be seen in Figure 4-10. Considering that there are 12 input and 8 hidden neurons, which in theory produces 65536 (4^8) possible rules, actual implemented rules are only a small proportion. From this point of view, the data have been narrowed down to some of the important rules rather than spread over the rule space.

4.5 Summary

In the current chapter, we applied the GNMM method to the EEG channel selection and classification problem. We have considered two datasets for our data driven technique. We demonstrated that GNMM is able to perform effective channel selections/reductions, which not only reduces the difficulty of data collection, but also greatly improves the generalization of the classifier. We have applied least square approximations to pre-process the data, and also discuss the effects of time window positions.

Some key conclusions can be drawn, as follows:

- By applying GA to optimize channel combinations, the significance of each channels relating to a specific task can be evaluated. Although the functionality of brain areas has long been studied, the difference between individuals can be vital in terms of EEG classification. This is especially true for those neurological patients who suffer from cerebral dysfunctions.
- Generally, using a selected subset improves the generalization ability of the model. This conclusion has also been reached by Lal, Hinterberger et al. (2005). More importantly, using selected channel subset(s) can result in a higher classification rate compared to using all available channels. This is mostly because channels containing irrelevant/noisy data have been removed.
- Another advantage of using a channel selection technique is that the

classifier is easy to understand. In particular, GNMM reduces its possible regression rules exponentially if the number of input neurons is reduced.

- In terms of LS pre-processing, it has greatly reduced the size of the dataset and improved the effectiveness of GNMM. From the present case studies, it seems that it is appropriate to use a combination of different time windows to achieve a high classification rate for correct and incorrect actual movement. However, establishing the precise number and temporal extent of these time windows for optimal results requires further investigation.
- In terms of both the topography of the selected channels and the time-course of classification accuracy, the results correspond to the neurophysiology of the processes under investigation, indicating that the present method might be usefully applied not only as a BCI-tool, but to basic neuroscientific research as well.

The selection of appropriate channels for EEG pattern classification has been one of the biggest problems for this kind of large datasets. By applying GNMM to two datasets, it is evident that GA based channel selection provides a potential solution to this problem. Furthermore, real-world applications based on a reduced number of EEG channels will be more feasible for patients that suffer from motor impairments.

References

Besserve, M., K. Jerbi, et al. (2007). "Classification methods for ongoing EEG and MEG signals." Biol Res 40(4): 415-437.

Blankertz, B., K.-R. Muller, et al. (2006). "The BCI competition III: Validating alternative approaches to actual BCI problems." IEEE Transactions on Neural Systems and Rehabilitation Engineering 14(2): 153-159.

Eriksen, B. A. and C. W. Eriksen (1974). "Effects of noise letters upon identification of a target letter in a nonsearch task." Perception and Psychophysics 16(1): 143-149.

Guger, C., A. Schlogl, et al. (1999). "Design of an EEG-based brain-computer interface (BCI) from standard components running in real-time under Windows." Biomed. Technik 44: 12-16.

Haynes, J. D. and G. Rees (2006). "Decoding mental states from brain activity in humans." Nature Reviews: Neuroscience 7(7): 523-534.

Kovacevic, N. and A. R. McIntosh (2007). "Groupwise independent component decomposition of EEG data and partial least square analysis." NeuroImage 35(3): 1103-1112.

Lal, T. N., T. Hinterberger, et al. (2005). "Methods Towards Invasive Human Brain Computer Interfaces." Advances in neural information processing systems.(17): 737-744.

Lotte, F., M. Congedo, et al. (2007). "A review of classification algorithms for EEG-based brain-computer interfaces." Journal of Neural Engineering 4(2): 1-13.

Martínez-Montes, E., P. A. Valdés-Sosa, et al. (2004). "Concurrent EEG/fMRI analysis by multiway Partial Least Squares." NeuroImage 22(3): 1023-1034.

Robert, C., J.-F. Gaudy, et al. (2002). "Electroencephalogram processing using neural networks." Clinical Neurophysiology 113(5): 694-701.

Robert, C., P. Karasinski, et al. (2002). "Monitoring anesthesia using neural networks: a survey." Journal of Clinical Monitoring and Computing 17(3-4): 259-67.

Sanchez, J. C. and J. C. Principe (2007). Optimal Signal Processing for Brain-machine Interfaces. Handbook of Neural Engineering. M. Akay. Hoboken, N.J., John Wiley: 3-28.

Schlaghecken, F., E. Blagrove, et al. (2008). "No difference between conscious and nonconscious visuomotor control: Evidence from perceptual learning in the masked prime task." Consciousness and cognition. 17(1): 84.

Schlaghecken, F. and M. Eimer (2006). "Active masks and active inhibition: A comment on Lleras and Enns (2004) and on Verleger, Jaskowski, Aydemir, van der Lubbe, and Groen (2004)." Journal of Experimental Psychology: General 135(3): 484-494.

Schlaghecken, F., S. T. Klapp, et al. (2009). "Either or neither, but not both: locating the effects of masked primes." Proceedings of the Royal Society of London, Series B: Biological Sciences 276(1656): 515-522.

Schlogl, A. (2003). "BIOSIG-an open source software library for biomedical

signal processing." Retrieved 03 July, 2008, from http://biosig.sourceforge.net.

Shuter, M. L., E. L. Hines, et al. (1994). Monitoring patient awareness states via neural network interpretation of EEG signals during anaesthesia trials. Proceedings of the International Conference on Neural Networks and Expert Systems in Medicine and Healthcare, Plymouth, UK, Univ. Plymouth.

Singh, H., X. Q. Li, et al. (2007). "Classification and feature extraction strategies for multi channel multi trial BCI data." International Journal of Bioelectromagnetism 9(4): 233-236.

Tian, L., D. Erdogmus, et al. (2005). Salient EEG Channel Selection in Brain Computer Interfaces by Mutual Information Maximization. Eng. Med. Biol. Soc., IEEE-EMBS.

Vocat, R., G. Pourtois, et al. (2008). "Unavoidable errors: A spatio-temporal analysis of time-course and neural sources of evoked potentials associated with error processing in a speeded task." Neuropsychologia 46(10): 2545-2555.

Wandell, B. A. (2008). "What's in your mind?" Nature Neuroscience 11(4): 384.

Chapter 5 Optimising the Number of Electronic Nose Sensors

5.1 Introduction

In previous chapters, we have provided an insightful analysis of GNMM's implementations (i.e. Chapter 3), and demonstrated its effectiveness through two complex EEG channel selection and classification problems (i.e. Chapter 4). The current chapter is concerned with an application of GNMM to the problem of optimal electronic nose (EN) sensor selection and pattern classification.

In terms of application, the use of EN and Multisensor Data Fusion (MDF) is widespread. Military applications include automated target recognition (e.g., for smart weapons), guidance for autonomous vehicles, remote sensing, battlefield surveillance, and automated threat recognition systems, such as identification-friend-foe-neutral (IFFN) systems. Nonmilitary applications include monitoring of manufacturing processes, condition-based maintenance of complex machinery, robotics, and medical applications.

5.2 Background – Multisensor Data Fusion

Odour classification systems used in machine olfaction, which are often called electronic noses (ENs), have been gaining favour in a wide range of industrial applications (Hines, Llobet et al. 1999; Llobet, Hines et al. 1999; Gardner, Shin et al. 2000; Dutta, Hines et al. 2003). An EN is a device that is designed to detect and discriminate among odours using a sensor array (Pearce, Schiffman et al. 2003). Typically, it comprises three main functional components: a sampling unit, a signal processing unit, and an odour classification unit (Phaisangittisagul and Nagle 2007). The sampling unit, which is analogous to biological olfactory receptors, typically consists of for example an array of gas sensors. The basic architecture of an electronic nose is shown in Figure 5-1 with the signals from an array of chemical sensors being processed and the 'smell fingerprint' being identified against those fingerprints already held in a knowledge base (i.e. a database for odours).

Usually the sensor element operates by measuring the physical property and outputting an analog signal which is amplified, filtered and then converted to a digital signal by the analog-to-digital (A/D) unit (Mitchell 2007). Unlike traditional analytical methods, EN sensor responses do not provide information on the nature of the compounds under investigation, but only give a 'digital fingerprint' of the odour, which can be subsequently investigated by means of data processing methods (Ulivieri, Distante et al. 2006). Thus, the composition of the sensor array of an electronic olfactory system is a

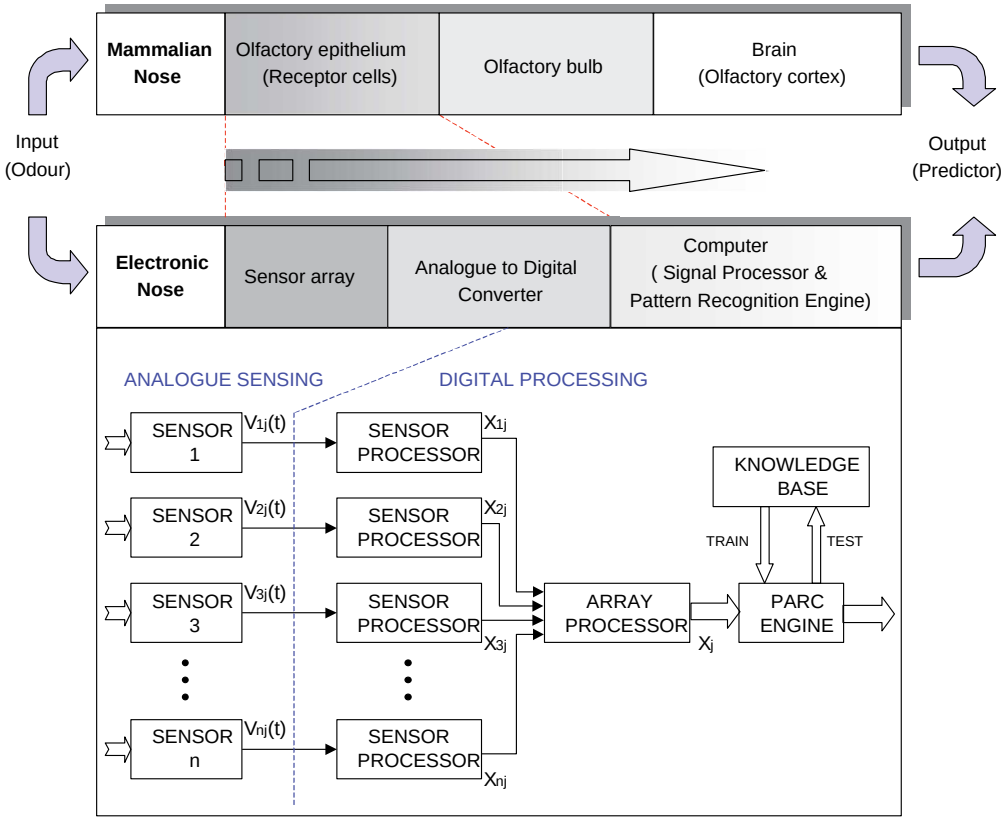


Figure 5-1: Schematic architecture of an electronic nose showing an array of chemical sensors, pre-processing, array processing and finally a supervised pattern recognition system (Adapted from Gardner and Yinon 2004)

fundamental choice which impacts significantly on the effectiveness of the overall system.

Sensors can be located in different ways (collocated, distributed, mobile) producing measurements of the same or of different types. Among these, the fusion of passive sensor data (e.g. electronic nose, EN), especially in the context of defence and security, is of particular importance (Koch 2007). Due to the emergence of new sensors, advanced processing techniques, and improved processing hardware, the MDF technology has undergone rapid growth since the late 1980s. In general, MDF is a technique by which data from

a number of sensors are combined through a centralized data processor to provide comprehensive and accurate information (Huang, Lan et al. 2007).

Another feature of MDF is that, due to recent advances in sensor developments, feature extraction, and data processing techniques, users are always provided with an increased amount of information using multi-sensor arrays (Gardner, Boilot et al. 2005). Taking the MLP as an example, a fully connected array of 10 sensors with 10 neurons in the hidden layer classifying 6 different odours would need 160 weights to be learnt. If the number of sensors in this MLP increases by 1, the number of weights would increase by 10. As the number of sensors in an EN array increase, the number of weights an MLP optimizes during training grows exponentially. On the other hand, increasing the dimensionality rapidly leads to the point where there may not be enough training data for the MLP to be trained optimally, in which case the MLP may provide a very poor representation of the input/output mapping. This is the phenomenon often referred to as *the curse of dimensionality* (Bishop 1995; Bishop 2006; Scott, James et al. 2006).

Generally speaking, even if each sensor is linked to specific classes of compounds, not all the sensors contribute to the characterisation of the odour which is being analysed (Ballabio, Cosio et al. 2006). Furthermore, not all of them are relevant to the particular PR classification task. Thus, the objective of any sensor selection algorithm should be to reduce the dimensionality and

also realise optimum PR accuracy, to eliminate redundant, noisy, or irrelevant sensors and thus find an optimal subset from an array of high dimensionality. By optimising the array size, the overall system performance can potentially be increased by maximising the information content and hence increasing the predictive accuracy.

5.3 Intelligent System Techniques Applied to MDF Problems

Hall and Llinas (2008) have identified three basic alternatives that can be used for multisensor data: (1) direct fusion of sensor data; (2) representation of sensor data via feature vectors, with subsequent fusion of the feature vectors; or (3) processing of each sensor to achieve high-level inferences or decisions, which are subsequently combined. However, due to the fact that sensor fusion models heavily depend on the application, there are no generally accepted models of sensor fusion – instead, there are numerous architectures and models for sensor fusion (Elmenreich 2007). Correspondingly, MDF techniques are drawn from, and bring together, a diverse set of more traditional disciplines, including digital signal processing, statistical estimation, control theory, and computer vision etc. Actually MDF itself is an interdisciplinary subject.

Compared with statistical methods (e.g. PCA), which are parametric and based on the assumption that the spread of the sensor data can be described by a Probability Density Function (PDF), IS-based PR techniques, for example MLP,

PNN, RBF, SOM, fuzzy inference systems (FISs), fuzzy c-means (FCM), fuzzy ARTMAP, EFuNN, and Gas, offer advantages such as learning capabilities, self-organization, generalization and noise tolerance (Hines, Boilot et al. 2003; Scott, James et al. 2006).

IS based PR techniques have been reported in the literature that determine an optimal subset of sensors for machine olfaction (Hines, Boilot et al. 2003; Gardner, Boilot et al. 2005; Ballabio, Cosio et al. 2006; Gualdron, Llobet et al. 2006; Scott, James et al. 2006; Llobet, Gualdron et al. 2007). For example, Gardner et al. (2005) introduced a modified GA called V-integer GA. In this V-integer GA, each chromosome was used with integer values from one to a pre-defined number of features/sensors representing the selected subset of features, and evaluated using PNN classifiers within the population. They also compared this V-integer GA with other search methods such as SFS or SBS and normal (binary) GAs. For the data-set used in their work, SFS achieved over 89% correct classification by selecting just three features, whereas SBS needed at least five features to reach the same level. With binary genes GAs, the dimensionality is reduced by 50–60% and the classification rates are on average 91%. Considering eight, six or four features, the optimal subsets returned by the V-integer genes GA selections have dimensionality reduced by over 80% and on average achieve around 90% correct classification. These results showed that the V-integer genes GA approach is an accurate search method when compared to some other feature selection techniques such as

SFS or SBS. However, in the V-integer GA, the number of sensors to be selected has to be defined in advance – in other words, there is potentially a lack of flexibility in some application scenarios.

On the other hand, Ballabio, Cosio et al. (2006) suggested a chemometric approach based on a partial ordering technique and the Hasse matrix. In this approach, the Hasse matrix can be obtained from each EN data sequence and the similarity between two sequences can then be evaluated with the definition of a distance between the corresponding Hasse matrices. Since all the signals which are temporarily selected are intrinsically ordered, the data provided by the EN can also be considered to be sequential data and can consequently be characterized as such. In this way, a similarity/diversity measure can thus be applied in order to characterize the class discrimination capability of each EN sensor. The distance based on the Hasse matrix is then used to link each EN time profile to a meaningful mathematical term (the Hasse matrix), which can be subsequently explored using multivariate analysis. However, in this model there is an absence of a proper classifier. The consequence of this is that the results of sensor selection are not comparable. In their case study, two sensors were selected out of a total of 15 to distinguish two features. This result was also confirmed by PCA. However, if the number of features increases, PCA may not be able to handle the problem and thus the whole method may fail to work efficiently.

Recently, a research group (Gualdron, Llobet et al. 2006; Llobet, Gualdron et al. 2007) have reported successful techniques for EN sensor selection. In the first case (Gualdron, Llobet et al. 2006), by evaluating different variable selection techniques (including deterministic and stochastic methods) coupled with neural network-based classifiers, they proposed a two-step strategy for sensor selection: a coarse selection based on a variance criterion followed by a SA process based on either fuzzy ARTMAP or the PNN. As a result, a success rate of 91.66% in simultaneous identification was obtained using only nine input variables (out of the 120 available) in their application. However, in this approach when computing the variance of each sensor, dependence (linear and/or nonlinear) between two sensors was not considered, and as such the selected subset may still contain redundant features and thus may not be the optimal subset. In the second case (Llobet, Gualdron et al. 2007), a three-step strategy for feature selection was presented: the first two steps were aimed at removing noisy, non-informative and highly collinear features; the third step makes use of a stochastic variable selection method (SA) to further reduce the number of variables. However, in this approach the threshold values for the discrimination ability and collinearity were both set heuristically. Therefore it is still possible that irrelevant sensors are not filtered out; and vice versa.

In the following sections, we will apply GNMM to the data that have been studied by Boilot, Hines et al. (2002) and Gardner, Boilot et al. (2005). On the one hand, we will demonstrate the effectiveness of GNMM by comparing the

results to those in the literature. Furthermore, GNMM's averaging effect during the variable selection stage will be studied.

5.4 Data V – Eye Bacteria Species

The EN dataset used (Data V) has previously been investigated by Boilot, Hines et al. (2002) and Gardner, Boilot et al. (2005). The data were collected using a Cyranose 320[10] EN to sample three dilutions of six eye bacteria species. The EN comprises an array of 32 sensors, and each dilution of these six bacteria was measured ten times. This gives a total of 180 samples belonging to six categories. For details about bacteria that cause eye infections and the experimental protocol/methodology, please refer to Boilot, Hines et al. (2002).

The statistics of the dataset are shown in Figure 5-2, in which the standard deviation (STD) is calculated according to $\sqrt{\sum(x-\bar{x})^2/(n-1)}$, where x is the data samples for each sensor and n is the total number of samples i.e. 180. It can be seen in Figure 5-2 that the maximum value for each sensor varies within a small range. This is because all the signals were produced by the same type of carbon black polymer composite resistors. However, the minimum values have a bigger variation, and so have the mean values, due to the fact that the EN sensors react differently to different odours. This feature helps in distinguishing odours using the EN data. It is noticeable that the STDs of sensors 8, 23, 24 and 32 are considerably larger than the others. These

[10] Smith Detection, www.smithsdetection.com.

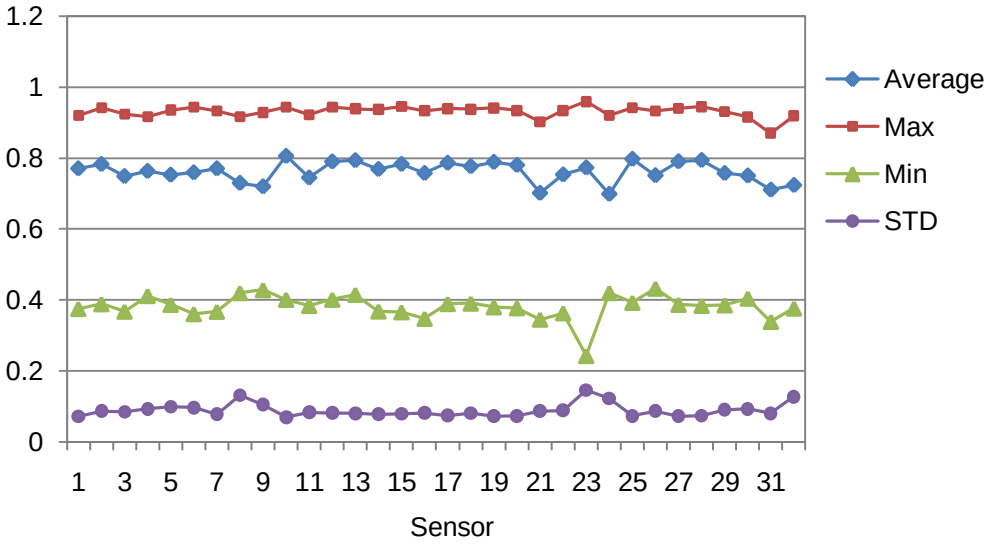


Figure 5-2: Statistics of the dataset

Table 5-1: GNMM configurations for the Data V

Case	N_p	N_g	N_e
1	30	100	20
2	30	100	50
3	30	150	20
4	30	150	50
5	30	200	20
6	30	200	50

findings may indicate sensors that would appear in the optimal subset of sensors.

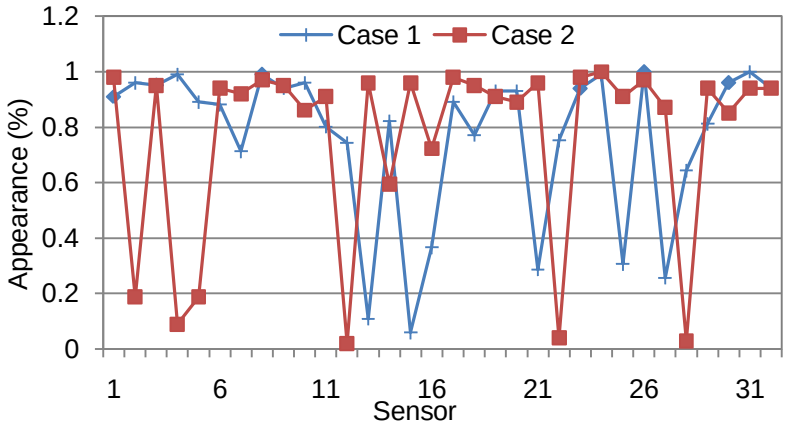
5.5 GNMM Results and Discussions

GNMM was implemented on a Sun workstation equipped with UltraSPARC III (900 MHz) CPUs. As suggested in the literature (Schaffer, Caruana et al. 1989), a relatively small population size and large mutation/crossover rate can

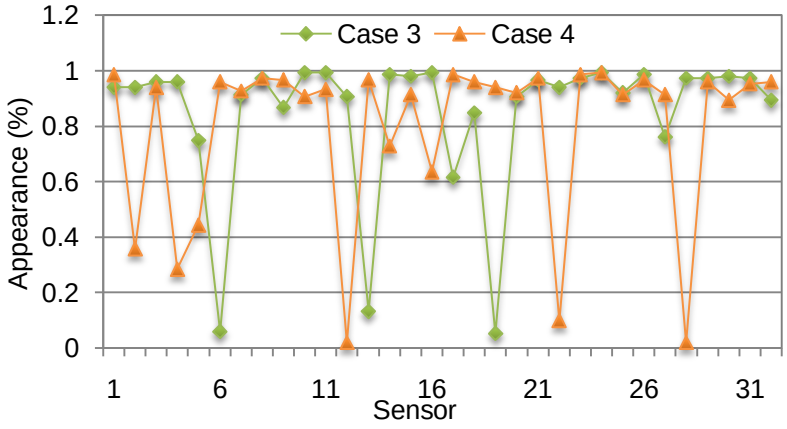
achieve thorough search in the search space. Thus the population size was kept small (30) for all GNMM runs and the mutation/crossover rate was set to be 0.8/0.01 respectively. GNMM was run six times for Data V, and the various configurations are shown in Table 5-1.

The *appearance percentage* of these 32 sensors in each of these six cases is shown in Figure 5-3 (a), (b) and (c), and the mean appearance percentage is shown in (d). Apart from illustrating each sensor's ability to be selected in the winning chromosome (i.e. the sensor subset performing the most accurate classification), Figure 5-3 also demonstrates the importance of repeating the GA's optimization processes. For example, sensor 6 performed quite well in case 1, 2, 4 and 6; however, this is not the case for case 3 and 5. By calculating an averaged 'appearance', we now know that the chance of sensor 6 being selected in the optimal sensor subset is quite low (~60%). On the other hand, sensor 19 approaches zero in case 3 and 5. But in Figure 5-3 (d) it can be seen that sensor 19 is not the worst one. To summarise Figure 5-3, by calculating the appearance percentage of each sensor, we smooth out the curve formed from a single GNMM implementation, and thus minimize the randomness associated with our GA and MLP.

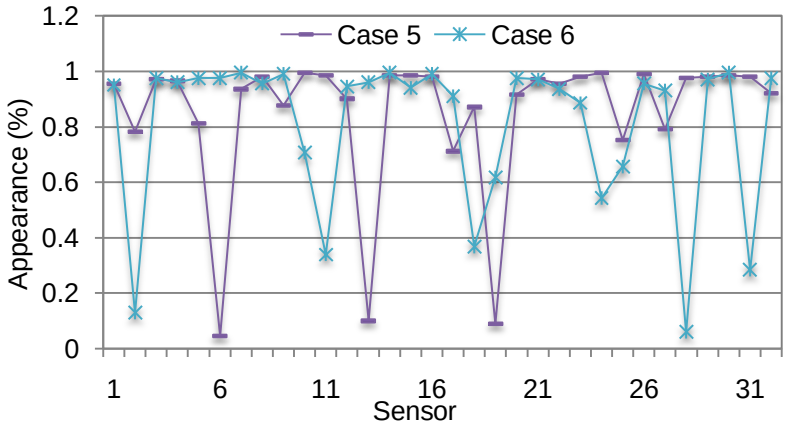
Figure 5-3 (d) also indicates that the best sensors are not quite distinguishable. However, a line can still be drawn to select the most important ones. By drawing a line at appearance percentage = 95%, we identified 6 sensors, which


Case 1
Case 2
Appearance (%)
1.2
1
0.8
0.6
0.4
0.2
0
1
6
11
16
21
26
31
Sensor

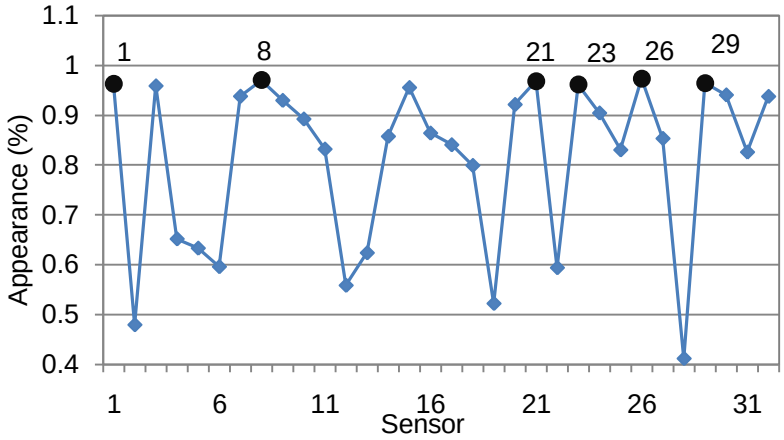
(a)


Case 3
Case 4
Appearance (%)
1.2
1
0.8
0.6
0.4
0.2
0
1
6
11
16
21
26
31
Sensor

(b)


Case 5
Case 6
Appearance (%)
1.2
1
0.8
0.6
0.4
0.2
0
1
6
11
16
21
26
31
Sensor

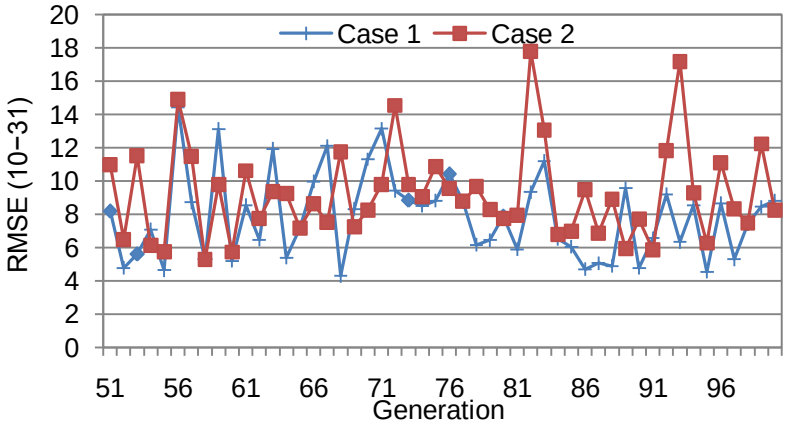
(c)



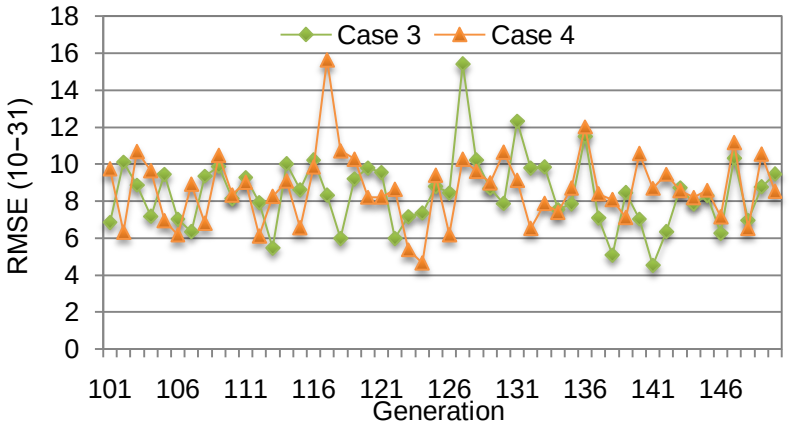
(d)

Figure 5-3: Appearance of each sensor in Data V for a single case ((a), (b) and (c)) and the mean appearance for all cases (d)

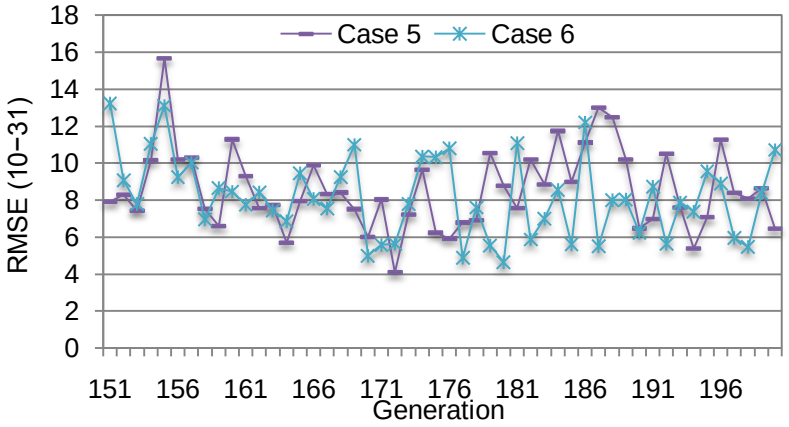
are illustrated by solid black dots in Figure 5-3 (d). These sensors and their mean appearance percentage are: 1 (96%), 8 (97%), 21 (97%), 23 (96%), 26 (97%) and 29 (96%). Reviewing the STDs of this sensor array, where sensor 8, 23, 24 and 32 are considerably higher than the others, it seems reasonable that some sensors (8 & 23) which contain the most diverse data were included in this optimal subset.



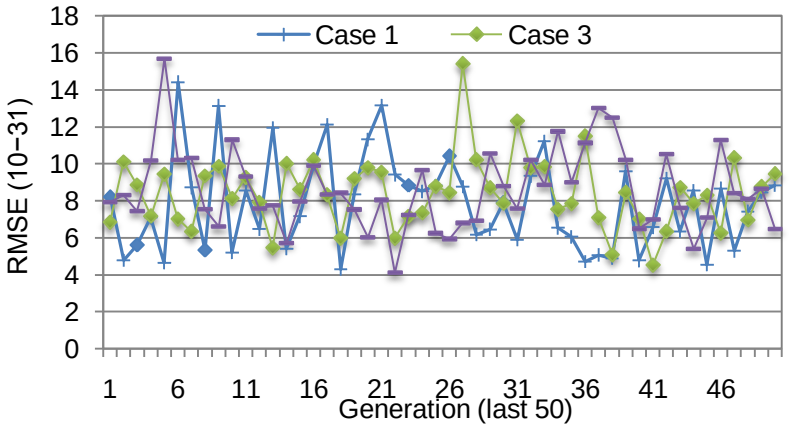
(a)



(b)



(c)



(d)

Figure 5-4: Comparisons of the RMSE for the last 50 generations for each case

Figure 5-4 provides comparisons of the RMSE for the last 50 generations for each case. These comparisons can help us understand GNMM's optimization process. Given that the other configurations are the same and only training epochs for the MLP classifier vary, which is the case illustrated by Figure 5-4 (a), (b) and (c), it is evident that larger number of epochs generally produce smaller values and variation of the optimization error (see Table 5-1). On the other hand, if the epochs are the same and the generations are different, larger generations normally yield lower error as in Figure 5-4 (d). However, the most outstanding feature in Figure 5-4 is the fact that later generations do not necessarily generate better performance. This finding, from another point of view, validates the importance of the mean *appearance percentage.*

In GNMM's MLP training stage, all of Data V were randomly divided into two subsets: one for training and one for validation. As a result, GNMM achieved 100% accuracy for both subsets. In order to test GNMM's training results, the data were again randomly divided into three equal subsets: one for training, one for testing, and one for validation. Once again, GNMM achieved 100% classification rate in recognizing the training and validation subsets. For the test set, an accuracy of 93% was achieved.

The optimal subset selected by GNMM (1, 8, 21, 23, 26, 29) is different compared with results obtained by (Boilot, Hines et al. 2002; Gardner, Boilot et al. 2005) (i.e. 8, 11, 15, 23, 31, 32), who applied the so-called V-integer GA

using PNN classification performance as the fitness function, as mentioned in Section 5.3 Intelligent System Techniques Applied to MDF Problems. However, in their work, there is an absence of a mechanism that minimizes the GA's randomness. As we already know from Figure 5-3 that a single run of a GA may not be representative of its overall performance, it is therefore necessary to run GA several times. Furthermore, a 100% and 93% classification rate compares favourably with the results from the previous work (90.6%) using the above six-sensor subset (i.e. 8, 11, 15, 23, 31, 32).

5.6 Summary

Recent advances in the field of ENs have led to new developments in sensor design, feature extraction (pre-processing), and data processing techniques. As a result, the user of EN systems is provided with an increased amount of information for the discrimination of odours using multi-sensor arrays. The dataset used in this chapter has previously been explored by other authors (Boilot, Hines et al. 2002; Gardner, Boilot et al. 2005). The number of sensors selected (i.e. 6) was deliberately made the same as those that have appeared in the literature. By comparing the results generated by GNMM to those presented in the literature, the effectiveness of GNMM is demonstrated.

GA researchers often report statistics, such as the best fitness found in a run and the generation at which the individual with that best fitness was discovered, averaged over many different runs of the GA on the same problem

(Mitchell 1996). The root cause of this is the random nature built-in with GA, which also holds true in the case of GNMM where averaging plays a vital role. The current chapter analysed the averaging effect of GNMM by looking at the GA implementation details.

It was found that the averaging performed in GNMM minimizes the randomness associated with a particular GA run and the evaluation of the fitness value. Furthermore, it also ensures that input variables are eventually evaluated in terms of possibility rather than, for example, a spectacular performance obtained in an extreme case.

References

Ballabio, D., M. S. Cosio, et al. (2006). "A chemometric approach based on a novel similarity/diversity measure for the characterisation and selection of electronic nose sensors." Analytica Chimica Acta 578(2): 170-177.

Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford, Oxford University Press.

Bishop, C. M. (2006). Pattern recognition and machine learning. New York, Springer.

Boilot, P., E. L. Hines, et al. (2002). "Classification of bacteria responsible for ENT and eye infections using the cyranose system." IEEE Sensors Journal 2(3): 247-252.

Dutta, R., E. L. Hines, et al. (2003). "Non-destructive egg freshness determination: An electronic nose based approach." Measurement Science and Technology 14(2): 190-198.

Elmenreich, W. (2007). A review on system architectures for sensor fusion applications. Software Technologies for Embedded and Ubiquitous Systems. 5th IFIP WG 10.2 International Workshop, SEUS 2007. Revised Papers. (Lecture Notes in Computer Science vol. 4761), Santorini Island, Greece, Springer Verlag.

Gardner, J. W., P. Boilot, et al. (2005). "Enhancing electronic nose performance by sensor selection using a new integer-based genetic algorithm approach." Sensors and Actuators B: Chemical 106(1): 114-121.

Gardner, J. W., H. W. Shin, et al. (2000). "An electronic nose system for monitoring the quality of potable water." Sensors and Actuators B: Chemical 69(3): 336-341.

Gardner, J. W. and J. Yinon (2004). Electronic noses & sensors for the detection of explosives. Dordrecht; Boston, Kluwer Academic Publishers: Published in cooperation with NATO Scientific Affairs Division.

Gualdron, O., E. Llobet, et al. (2006). "Coupling fast variable selection methods to neural network-based classifiers: Application to multisensor systems." Sensors and Actuators B: Chemical 114(1): 522-529.

Hall, D. L. and J. Llinas (2008). Multisensor Data Fusion. Handbook of multisensor data fusion: theory and practice. M. E. Liggins, D. L. Hall and J. Llinas. Boca Raton, FL, CRC Press: 1-14.

Hines, E. L., P. Boilot, et al. (2003). Pattern analysis for electronic noses. Handbook of machine olfaction: electronic nose technology. T. C. Pearce, S. S. Schiffman, H. T. Nagle and J. W. Gardner. Weinheim, Wiley-VCH: 133-160.

Hines, E. L., E. Llobet, et al. (1999). "Neural network based electronic nose for apple ripeness determination." Electronics Letters 35(10): 821-823.

Huang, Y.-b., Y.-b. Lan, et al. (2007). "Multisensor Data Fusion for High Quality Data Analysis and Processing in Measurement and Instrumentation." Journal of Bionic Engineering 4(1): 53-62.

Koch, W. (2007). Sensor Data Fusion: Methods, Applications, Examples Advances and challenges in multisensor data and information processing. E. Lefebvre. Amsterdam, IOS Press: 1-23.

Llobet, E., O. Gualdron, et al. (2007). "Efficient feature selection for mass spectrometry based electronic nose applications." Chemometrics and Intelligent Laboratory Systems 85(2): 253-261.

Llobet, E., E. L. Hines, et al. (1999). "Fuzzy ARTMAP based electronic nose data analysis." Sensors and Actuators, B: Chemical B61(1-3): 183-190.

Mitchell, H. B. (2007). Multi-sensor data fusion: an introduction. Berlin, Springer Verlag.

Mitchell, M. (1996). An introduction to genetic algorithms. Cambridge, Mass., MIT Press.

Pearce, T. C., S. S. Schiffman, et al. (2003). Handbook of machine olfaction : electronic nose technology. Weinheim [Germany], Wiley-VCH.

Phaisangittisagul, E. and H. T. Nagle (2007). "Enhancing multiple classifier system performance for machine olfaction using odor-type signatures." Sensors and Actuators B: Chemical 125(1): 246-253.

Schaffer, J. D., R. A. Caruana, et al. (1989). "A study of control parameters affecting online performance of genetic algorithms for function optimization." Proceedings of the third international conference on Genetic algorithms: 51-60.

Scott, S. M., D. James, et al. (2006). "Data analysis for electronic nose systems." Microchimica Acta 156(3-4): 3-4.

Ulivieri, N., C. Distante, et al. (2006). "IEEE1451.4: A way to standardize gas sensor." Sensors and Actuators B: Chemical 114(1): 141-151.

Chapter 6 Classification of the Pima Indians Diabetes Database

6.1 Introduction

In previous application chapters, we have benchmarked GNMM's effectiveness by comparing its prediction/classification results with those presented in the literature using the same dataset. For example, Chapter 3 utilizes prediction done by Tayfur and Singh (2005) based on Data II. Chapter 4 uses classification results obtained by Lal, Hinterberger et al. (2005) based on Data III. While in Chapter 5 GNMM results were compared against work done by Boilot, Hines et al. (2002) on Data V. In the current chapter, studies will be conducted to evaluate and compare the results obtained using GNMM with several widely used IS techniques including ANFIS, EFuNN, Fuzzy ARTMAP, and CGP, the aim being to further investigate GNMM's features before any conclusions are drawn in the final chapter (i.e. Chapter 7).

Furthermore, although GA parameter range was briefly discussed in Section 2.3.1.2 Parameters, it remains unclear as to whether different parameter

settings will result in different input variable selection results. The current chapter will try to address this question using a widely studied dataset.

6.2 Dataset

The Pima Indian Diabetes database, i.e. Data VI, obtained from UC machine learning repository[11] is owned by the National Institute of Diabetes and Digestive and Kidney Diseases (Smith, Everhart et al. 1988). It contains 768 instances, 8 input attributes and 1 target, which represents whether the data shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl found in any survey examination or during routine medical care (Lin and Soo 1997)). Attributes in Data VI are the number of times pregnant, plasma glucose concentration, diastolic blood pressure (mm Hg), triceps skin fold thickness (mm), 2-hour serum insulin (mu U/ml), body mass index (kg/m^2), diabetes pedigree function, and age denoted by Attr1 to Attr8 respectively. 268 instances of the data are positive, which is 34.9% of the database. There is no missing value instance. Some statistics of Data VI are shown in Table 6-1.

Table 6-1: Data VI statistics

	Attr1	Attr2	Attr3	Attr4	Attr5	Attr6	Attr7	Attr8
Max	17	199	122	99	846	67.1	2.42	81
Min	0	0	0	0	0	0	0.078	21
Avg	3.85	120.89	69.11	20.54	79.80	32.00	0.47	33.24

[11] Machine Learning Repository, UC Irvine, http://archive.ics.uci.edu/ml/index.html.

Data VI has been widely investigated previously in the literature (Smith, Everhart et al. 1988; Carpenter and Tan 1995; Lin and Soo 1997; Eggermont, Kok et al. 2004; Kahramanli and Allahverdi 2008), thus it is used to make comparisons in the current chapter. For example, Smith, Everhart et al. (1988) have applied ADAP, a feedforward neural network model, to this dataset using 576 training data and 192 testing data and achieved 76% accuracy. Eggermont, Kok et al. (Eggermont, Kok et al. 2004) achieved about 26% misclassification using GPs. In the recent work done by Kahramanli and Allahverdi (2008) a hybrid neural network that includes ANN and fuzzy neural network (FNN) was developed and they achieved an accuracy of 84.24%.

6.3 GNMM Results

First of all, an MLP was trained using Data VI with all available attributes using the LM algorithm. As a result, within 23 epochs it achieved an RMSE of 0.37 with an accuracy of 79.95%. Applying GNMM to Data VI, the four most significant attributes were found – Attr2, 6, 7, and 8 have the highest appearance percentage, as shown in Figure 6-1. Training the subset formed by these four attributes (denoted by Data VI_{gnmm}) and the classification target, with an MLP with four hidden neurons we achieved a classification RMSE of 0.38 with an accuracy of 79.30%. Hence by using 4 attributes out of 8 we achieved a similar accuracy. This implies that the model was successfully trained to achieve the PR tasks.

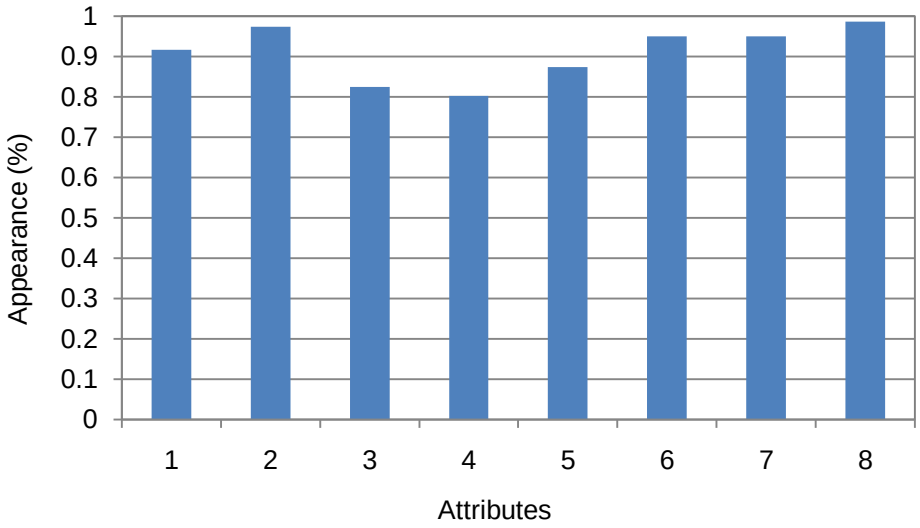


Figure 6-1: Appearance percentage for each attribute in Data VI

Table 6-2**:** 10 most significant rules fired for Data VI

No.	*Data* VI_t	*Data* VI_s
1441	114	17
4141	64	8
2441	44	6
3441	35	1
4441	33	0
1411	27	3
4241	25	2
1421	24	2
4341	22	2
1431	17	5

The 10 most significant rules (out of a total of 75) extracted from Data VI are shown in Table 6-2, in which Data VI_t stands for the training subset, and Data VI_s denotes the validation subset. From Table 6-2 it is evident that the validation set is representative – while the most significant rule for Data VI_t is

rule No.1441, the same is true for Data VI_t. This means that in general most data samples reside within the sub-spaces represented by this rule and the number gradually decreases as data sample move further away from it. In this way, GNMM not only determines the number of rules associated with each data sample, but also determines the importance of the sub-spaces and the distribution of data samples.

6.4 Other Hybrid IS DM Techniques

6.4.1 ANFIS

Applying ANFIS to Data VI (all 8 attributes used), with the default grid partitioning of the input space (Leondes 1999; Karray and De Silva 2004), the system soon ends up with the problem of the *curse of dimensionality*, as it produces as many as 256 (i.e. 2^8) rules when two MFs are used for each input, which is clearly an unacceptable number of rule permutations. However, if the ANFIS structure is generated using FCM clustering, which considerably reduces the number of rules (4 vs 256), ANFIS did achieve a good classification as will be shown in the next paragraph.

Figure 6-2 shows the ANFIS structure, in which case there are four membership functions for each input attribute and a total of four rule nodes. Within 100 epochs, the RMSE was reduced to 0.42 with a classification accuracy of 56.64%. The target/prediction class labels and prediction error for each sample are shown in Figure 6-3. From Figure 6-3 it is evident that the

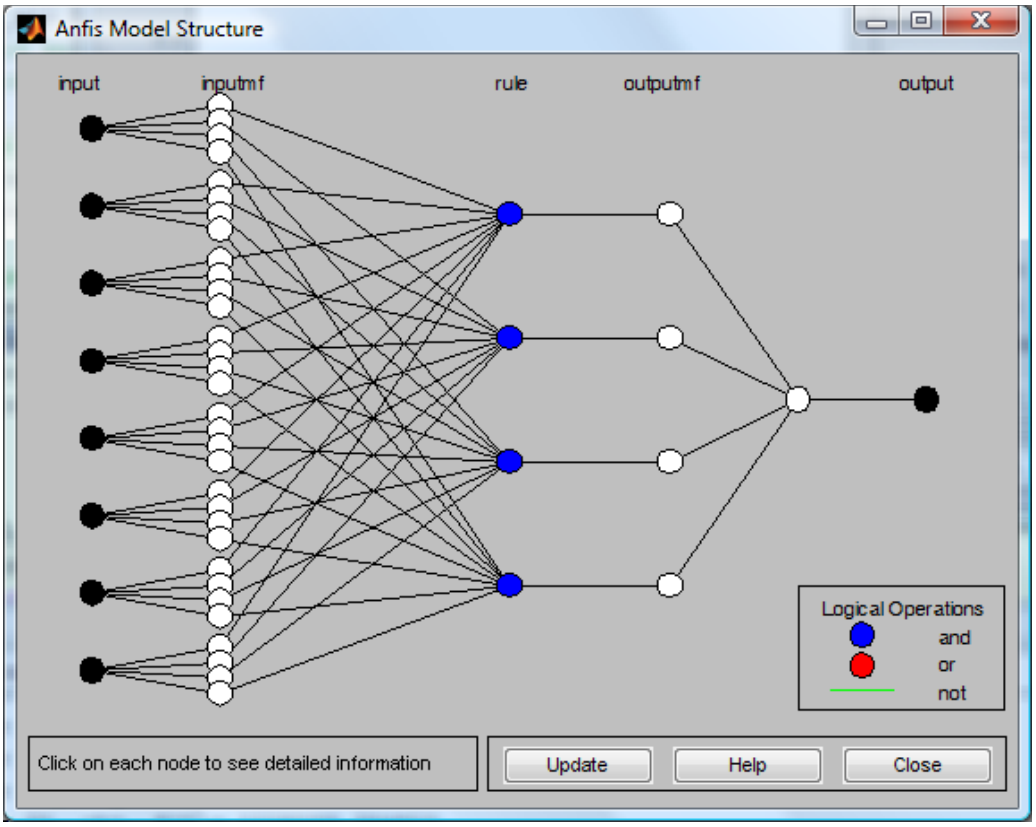


Figure 6-2: Structure of ANFIS generated for Data VI

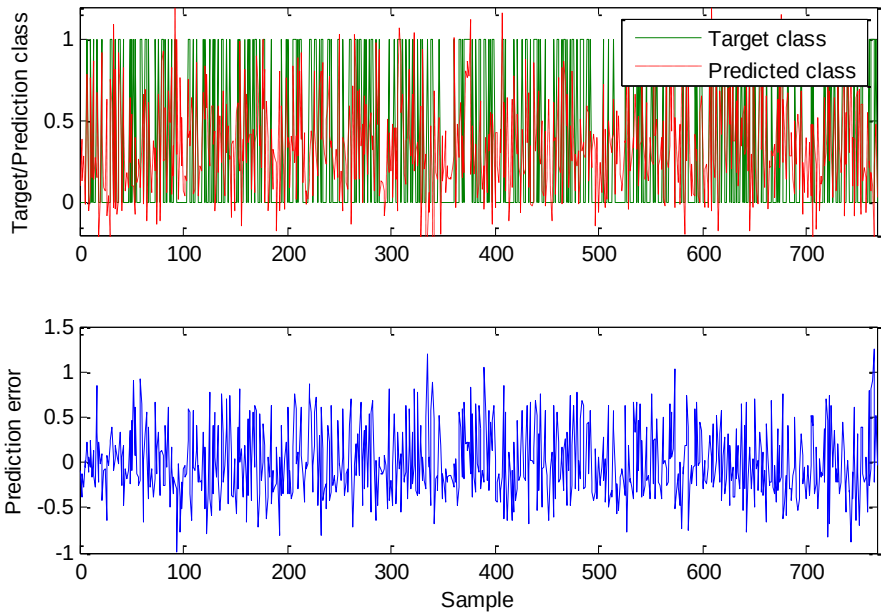


Figure 6-3: Target/predicted class values and ANFIS prediction error for Data VI

error distribution is quite random, which implies that a first-order Sugeno-type FIS may not be suitable for this problem. Compared with the GNMM results, this error is slightly higher (0.42 vs 0.38). However, ANFIS converges much faster, due to its hybrid learning and the ability to construct reasonably good input MFs (Ozkan 2006).

The rule viewer results from ANFIS are shown in Figure 6-4. A total of four rules are extracted from the system. However, each of these four rules has an antecedent consisting of 8 parts (i.e. 8 attributes). In terms of interpretability, this is not easily interpretable. Also note that the rule antecedent and consequent parts remain unchanged throughout training, as shown in Figure 6-5. It is also evident that no membership degrees are displayed. For the consequent part, each rule represents a single MF (i.e. the number of rules is equal to the number of output MFs) with the same unit weights, there is no rule sharing in the ANFIS system. Thus, the ANFIS training adjusts parameters such as MFs and network weights instead of manipulating rules and network structures as in some other systems such as EFuNN.

The MFs associated with Attr5 are shown in Figure 6-6, and the rule surface formed by the first two attributes is shown in Figure 6-7. From Figure 6-6 and Figure 6-7 it is clear that the concept representation learned by ANFIS is easier to understand (Boilot, Hines et al. 2000). This can be seen from the fact that inputs to the ANFIS rule space are attribute outputs; whereas inputs to GNMM

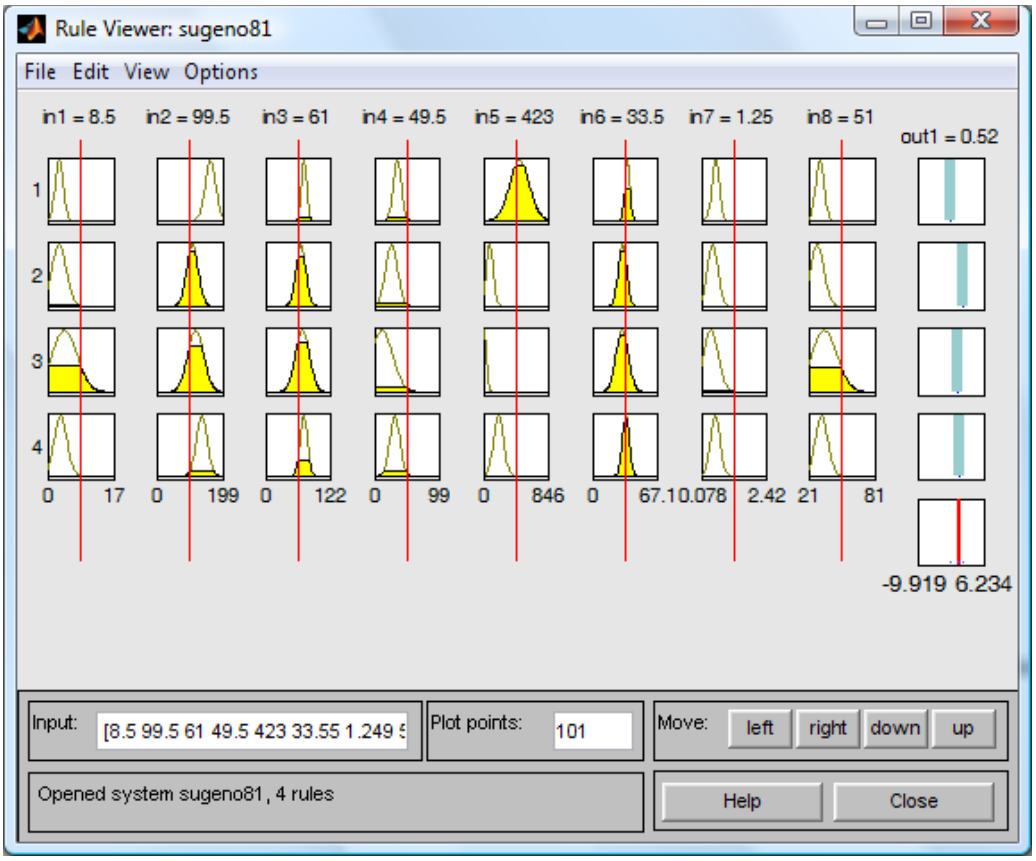


Figure 6-4: ANFIS rule viewer applied to Data VI

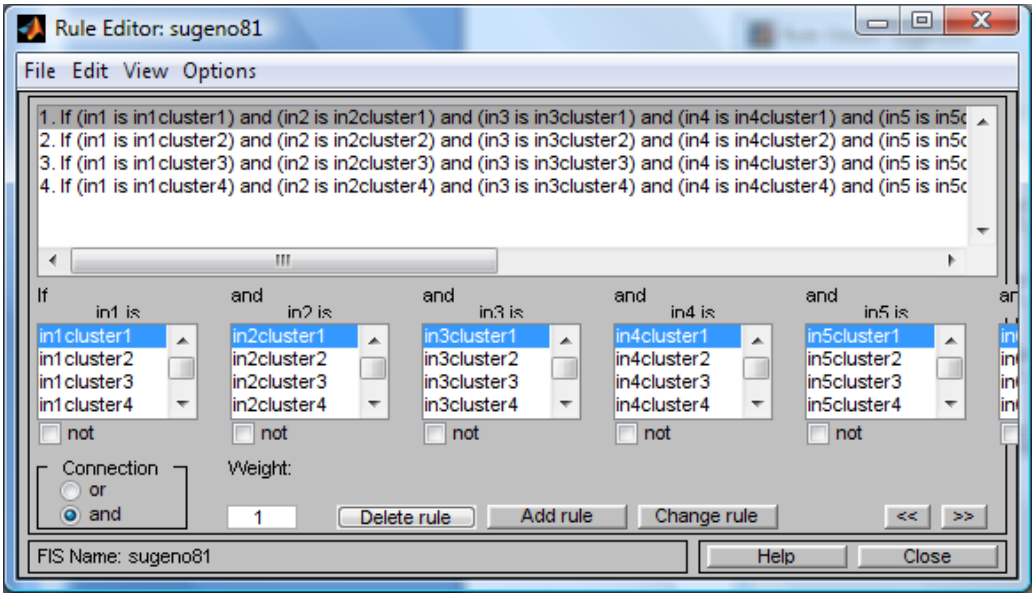


Figure 6-5: Rules extracted from the ANFIS system for Data VI

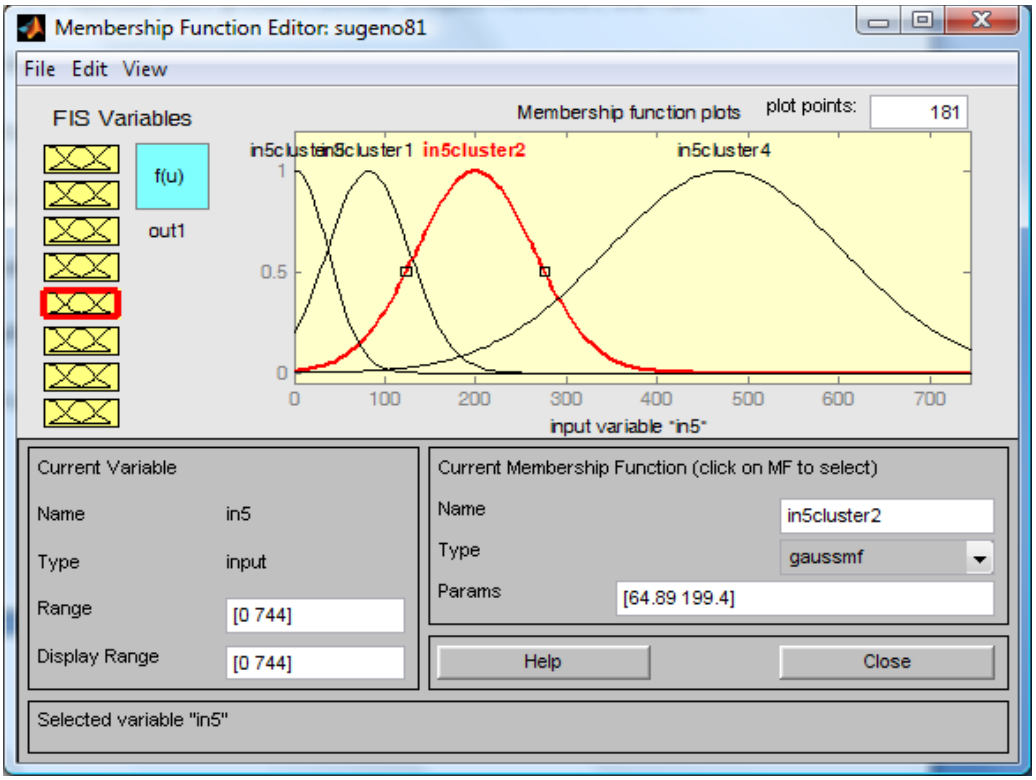


Figure 6-6: Membership functions for Attri5

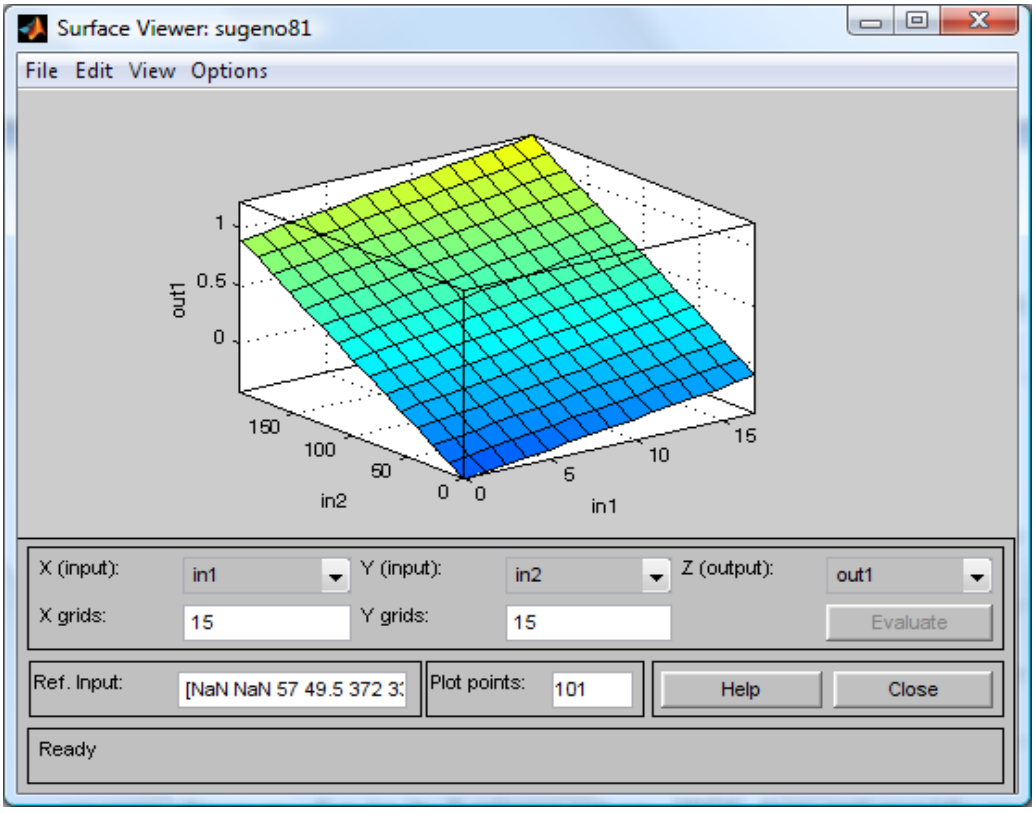


Figure 6-7: Rules surface the ANFIS system for Data VI

are values to first-layer neurons. However, due to its fuzzy nature, ANFIS does not provide an insight into the data distribution and rule importance, as each input belongs to different sets to different degrees. On the other hand, GNMM gives an idea of the importance of each rule by determining data samples that fall into the rule space.

Furthermore, ANFIS has a fixed structure that cannot adapt to the data in hand, therefore it has limited abilities for incremental, online learning (Kasabov 2007). Whereas in GNMM, the ANN structure can easily be adopted to take additional inputs.

6.4.2 EFuNN

Applying EFuNN to Data VI the system achieved an RMSE of 0.51, as depicted in Figure 6-8. In addition, the system produces 453 rules, each of which has 8 antecedent parts, as depicted in Figure 6-9. It is evident that too many rules affect the interpretability of the system.

The results show that the EFuNN rules are quite different from the ANFIS rules as shown in Figure 6-5. As opposed to the case in ANFIS where simple grid partitioning is applied and where training is performed mainly to adjust MF parameter, in EFuNN the aim of training is to find connection nodes that associate fuzzy inputs and outputs. Thus, EFuNN rules are given in the form of membership degrees that each input/output belongs to. Take rule No. 453,

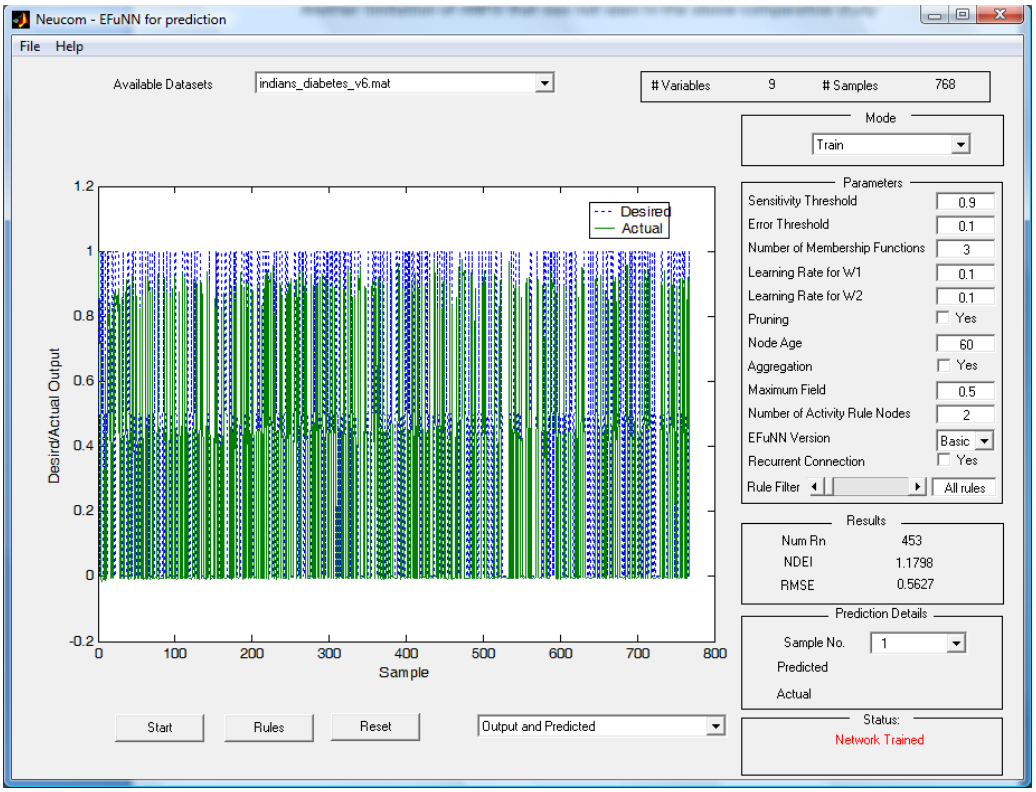


Figure 6-8: Target and EFuNN prediction class values for Data VI

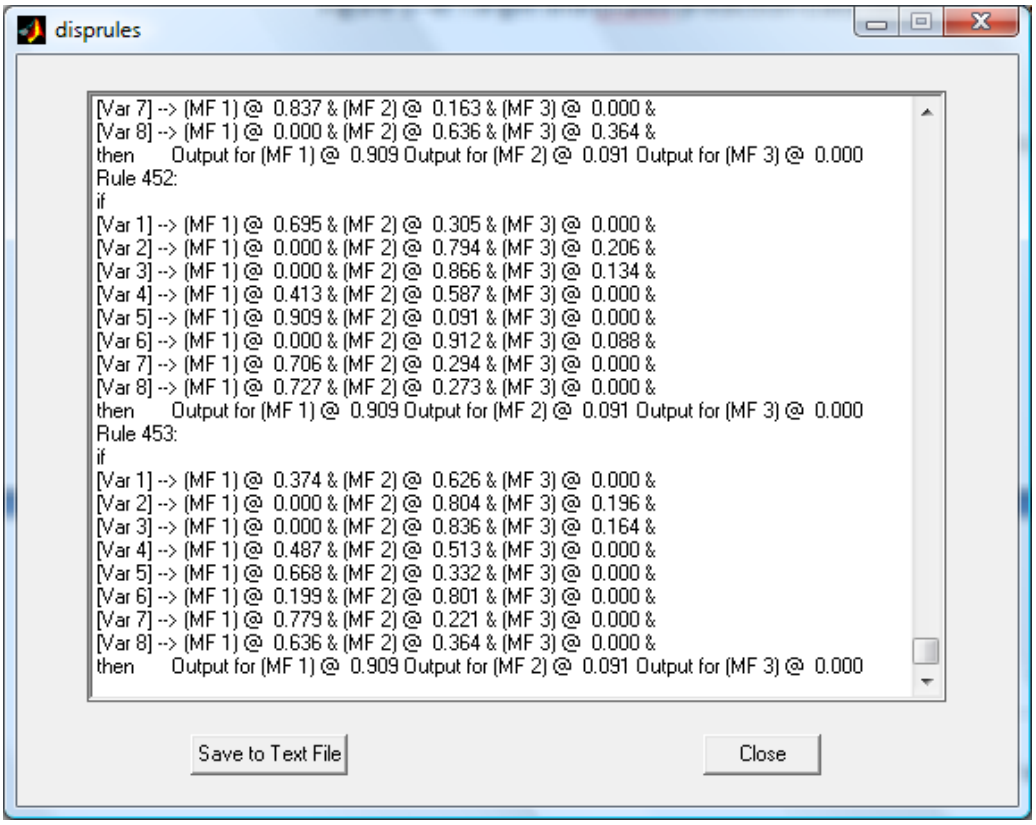


Figure 6-9: Rules extracted from the EFuNN system for Data VI

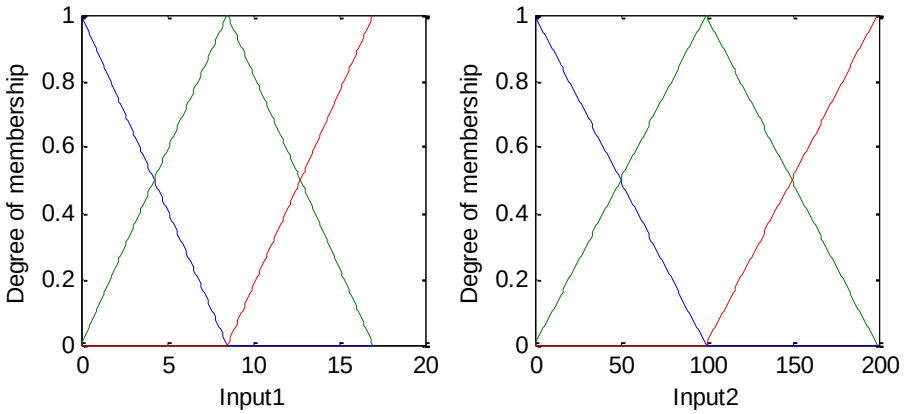


Figure 6-10: EFuNN MFs for the first two attributes of Data VI

which is highlighted in Figure 6-9, for an example. Basically it states that if input1 belongs to its $1^{st}/2^{nd}/3^{rd}$ MF to a degree of 0.374/0.626/0.000 respectively etc., then the fuzzy output is [0.909 0.091 0]. Based on these fuzzy output values, aggregations can be performed to produce predicted class values.

Figure 6-10 shows the MFs used for the first two inputs. Another popular choice for input MFs are Gaussian functions. As these MFs do not change as iterations proceed, choices have to be made before training starts. However, this highlights a potentially important disadvantage of EFuNN, i.e. the determination of the network parameters. There are many parameters in EFuNN such as number and type of MF for each input variable, sensitivity threshold, error threshold and the learning rates etc (see Figure 6-8). Even though a trial and error approach is practical, when the problem becomes more complicated (large numbers of input variables) determining the optimal parameters may be computationally expensive (Abraham and Nath 2001).

6.4.3 Fuzzy ARTMAP

Applying Fuzzy ARTMAP to Data VI, the system achieved an RMSE of 0.48 with an accuracy of 76.82% using 38 committed coding nodes. The target/prediction class values and the prediction error are depicted in Figure 6-11. From Figure 6-11 it is clear that unlike in previous cases where the predicted values can be non-integers, in the case of Fuzzy ARTMAP all predicted values are integers. Thus although for most data samples the system correctly performs the classification, a few incorrectly classified samples results in a relatively large RMSE error.

Compared with GNMM, the Fuzzy ARTMAP network has the advantage of being fast and requiring no fine tuning of parameters. It also retains all the information that it has been trained for and does not suffer from temporal

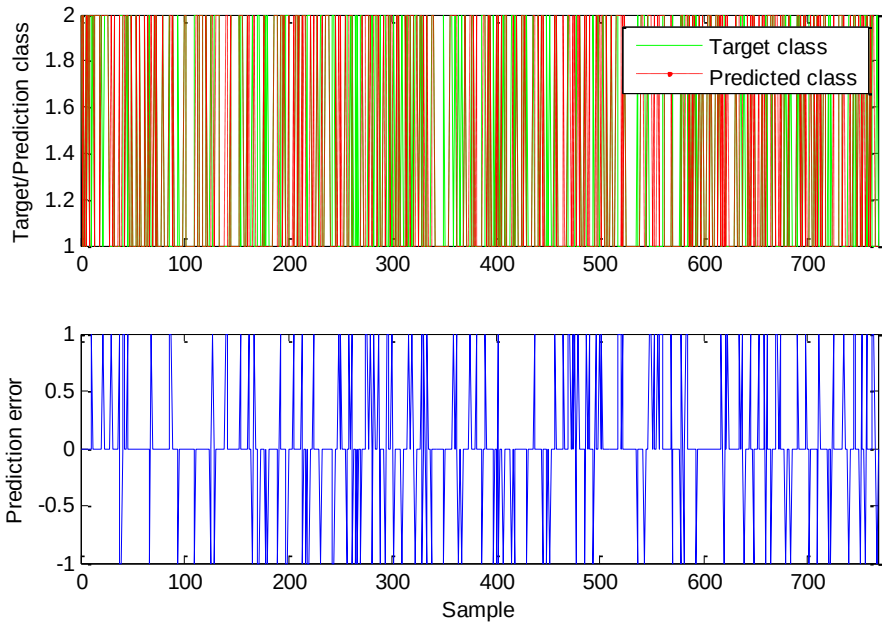


Figure 6-11: Target/predicted class values and Fuzzy ARTMAP prediction error for Data VI

instability during on-line training (Mahadevan and Raghavendra 1997). The drawback of Fuzzy ARTMAP is that it is only a predictor and not a generalizer (e.g. it does not provide non-integer values for Data VI); whereas GNMM can be a universal modeller. Another disadvantage of Fuzzy ARTMAP is that it is very sensitive to the order of presentation of the training data. It is also extremely sensitive to the selection of the vigilance parameter, which controls the size of the hyper-box, and finding the optimal value for the vigilance parameters can be quite challenging (Vilakazi and Marwala 2009).

In terms of interpretability, in the original work done by Carpenter, Grossberg et al. (1992) Fuzzy ARTMAP does not extract rules form the trained system. In successive research different authors have proposed methods to extract rules from trained Fuzzy ARTMAP (Carpenter and Tan 1995; Andres-Andres, Gomez-Sanchez et al. 2005; Tian, Liu et al. 2006), most of which rely on investigating clusters formed by committed nodes. However, the dilemma is that interpretability of Fuzzy ARTMAP increases with fewer committed nodes; whereas the system performance error tends to grow as the number of committed nodes decreases (Connolly, Granger et al. 2008; Granger, Connolly et al. 2008).

6.4.4 CGP

Applying CGP to Data VI, after running the programme five times the best results were obtained using setting depicted in Figure 6-12. The system

achieved a classification accuracy of 61.98%. However, the system produced as many as 396 'infinity' prediction values (out of 768), as a result of zero dividend, as opposed to the target value '1'. This makes the calculation of RMSE impossible.

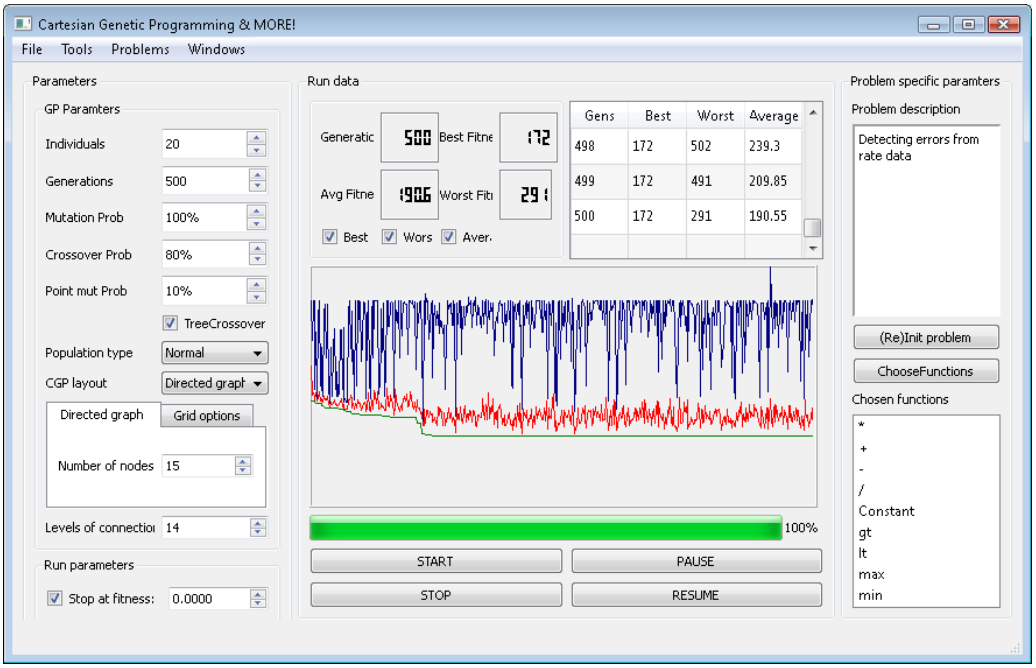


Figure 6-12: CGP settings and simulation results for Data VI

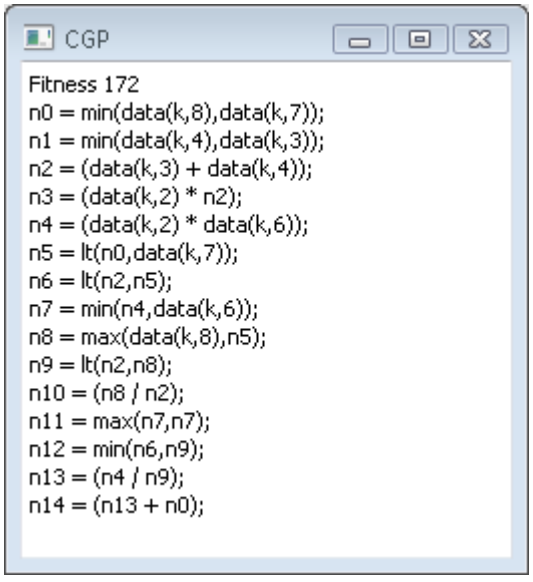


Figure 6-13: Arithmetic rules extracted from CGP for Data VI

Arithmetic rules extracted from CGP are shown in Figure 6-13. Since 15 nodes were used with the level of connections being 14 (in Figure 6-12), Figure 6-13 shows 15 arithmetic expressions. However, not all nodes were used to calculate the final results e.g. node 12. Furthermore, not all input attributes were used to calculate the final results – as shown in Figure 6-13, only attributes 2, 3, 4, 6, 7, and 8 were used.

Compared with GNMM, CGP achieved an automatic input variable deduction without an explicit input selection step. This was realized by evolving different input variables combined with various arithmetic operators. Furthermore, due to its GP nature, rule extraction from the CGP system is straight forward.

However, one of the main disadvantages of CGP, which also holds true for any EA based techniques, is that it can be very computer intensive, often requiring extensive computing power (Hughes and Ruprai 1999). Furthermore, functions that can be constructed by the algorithm need to be selected carefully (Schmutter 2002). One the one hand, the number of possible functions is immense; on the other hand, fewer functions will increase the efficiency of the algorithm.

A summary of classification results is given in Table 6-3, where comparisons are made based on the best result for each individual technique. Instead of computing average performance over several runs as in previous chapters, in the current chapter the best performance is used. This is because rules derived

Table 6-3: Comparison of classification results for Data VI. Results for GNMM, ANFIS, Fuzzy ARTMAP, and CGP are the best results out of 5 independent runs. FFuNN's results do not vary with the same settings, due to its way of random number generation

Method	Accuracy	RMSE	Number of rules
Various GPs in Eggermont, Kok et al. (2004)	74-72%	n/a	n/a
Hybrid ANN in Kahramanli and Allahverdi (2008)	84.24%	n/a	No rule extraction
GNMM	79.30%	0.38	75
ANFIS	56.64%	0.42	4
EFuNN	49.76%	0.51	453
Fuzzy ARTMAP	76.82%	0.48	n/a
CGP	61.98%	n/a	1

from the training are directly associated with training results. From this point of view, the number of rules and their antecedents/consequents are fixed once the training is done. From a practical viewpoint, only the best training results will be implemented in an engineering practice. Hence, the best results are a critical factor in determining the performance of different techniques.

Compared with classifiers such as Fuzzy ARTMAP, which produces discrete output class values, GNMM produces continuous outputs which results in a smaller RMSE. In contrast to fuzzy-space-mapping approaches such as EFuNN, GNMM has an input selection step which results in much fewer numbers of rules. Furthermore, GNMM's MLP basis ensures that it achieves higher classification accuracy than GP methods and first-order FISs. Overall, although

there exists methods that can achieve higher classification rate (e.g. methods in Kahramanli and Allahverdi (2008)), it is evident that GNMM achieved a balance between classification accuracy and reduction of number of rules generated – GA-based input deduction allows the elimination of input variables, and MLP modelling achieved a high classification rate.

6.5 GA Parameter

Section 2.3.1.2 Parameters has briefly discussed the GA parameter selection problem by introducing a good range of parameters proposed in the literature (Schaffer, Caruana et al. 1989; Haupt and Haupt 2004). In previous chapters of this thesis although different GA parameters have been used for different datasets, most falls into that range. In addition, these parameter settings follow certain rules, e.g. larger populations and generations for datasets with more input variables. However, one might still ask:

- Are these optimal parameter settings for the corresponding problem?
- Will the GNMM input selection results be different if another set of parameters were used?

The following sections explore the answers to these questions in the context of Data VI.

6.5.1 Interactions among GA Parameters

Over the years researchers have been trying to understand the mechanics of GA parameter interactions by using various techniques (Deb and Agrawal 1999). However, it still remains an open question as to whether there exists an optimal set of parameters for GA in general (De Jong 2007). The reason for this is two-fold: on the one hand conventional genetic operators can have various forms and control parameters and recent development in GA theory have also introduced many more parameters to be set (Fogel, Bäck et al. 2000; De Jong 2005); on the other hand achieving the exploration/exploitation balance involves adjusting these parameters simultaneously and is limited to the problem being dealt with (Maturana and Saubion 2008).

Techniques for assigning values to parameters can be classified according to the taxonomy proposed by Eiben, Michalewicz et al. (2007). In general, they are classified into two categories: one is parameter tuning, where parameters are fixed before the run; the other is parameter control, where parameters are modified during the run. Regardless of categories these techniques belong to, the interaction among GA parameters follow some general rules (Deb and Agrawal 1999; De Jong 2006; Lobo, Lima et al. 2007):

- GA parameters interact with each other so as to affect the behaviour of the system in complex, nonlinear ways.
- For a given problem the selected parameter values are not necessarily

optimal, even if the effort made to set them was significant.

- GAs with both crossover and mutation operators perform better than only crossover or mutation based GAs for simple problems.
- Large mutation steps can be good in the early generations, helping the exploration of the search space, and small mutation steps might be needed in the later generations to help fine-tune the suboptimal chromosomes.

GNMM incorporates some techniques that correspond to these rules, such as the adaptive mutation rate as detailed in Figure 2-8 and including both selection and mutation operators. For a detailed discussion about different GA parameter settings, please refer to De Jong (2006) and Lobo, Lima et al. (2007).

6.5.2 Determine the Parameter Set

The question of whether a particular set of GA parameter is optimal is largely dependent on the aim of GA optimization results. Depending on the nature of the problem being investigated, some researchers used the best fitness value as the criteria for evaluating GA parameters (e.g. in Costa, MacIel et al. (2005) and Cakir, Butun et al. (2006)); while some others also combined this with the time when the best solution was found (Vajda, Eiben et al. 2008). In GNMM, however, using the best fitness value is not ideal because for the winning chromosome it may have different fitness values due to MLP's randomness. For the same reason, the time when the best chromosome was found cannot

be used either. In GNMM the use of GA is to accumulate the appearance of each input variable in the winning chromosome so that the possibility of that variable appearing in the finial training can be determined. Statistical property that best describe GA's behaviour with this regard is its mean fitness value over the entire generation. Therefore, this is used as the criteria to evaluate the performance of different GA parameters.

When studying the effect of different GA parameters, one could try all different combinations systematically. However, this approach is practically impossible as GA parameters are not independent. A frequently used method is to adjust one variable while keeping all others constant (Schaffer, Caruana et al. 1989; Sun, Hines et al. 2005). Therefore, setting initial range for each parameter is vitally important. Setting step sizes for each parameter also needs careful considerations. On the one hand, large step size may result in selected parameters being very coarse; on the other hand, small step size may result in the test being very time-consuming. For example, Schaffer, Caruana et al. (1989) spent over a year of CPU time systematically testing a wide range of parameter combinations. The approach adopted here to determine GA parameter ranges and step sizes will follow those in Schaffer, Caruana et al. (1989) and Sun, Hines et al. (2005).

6.5.3 Ranges and Step Sizes

Sun, Hines et al. (2005) have studied initial parameter values suggested in Goldberg (1989), Michael Johnson and Rahmat-Samii (1997), and Man, Tang et al. (1999) and given a good range of initial values as in Table 6-4. The parameter set suggested by Schaffer, Caruana et al. (1989) (i.e. N_p = 20 to 30, p_c = 0.75 to 0.95, p_m = 0.005 to 0.01 in Section 2.3.1.2 Parameters), which is also the set being used in most case studies, falls into this range. Therefore, the following initial values will be used in our studies: population size 25, generations 100, crossover probability 0.6, and mutation probability 0.01, as in Table 6-4.

For the step sizes, Sun, Hines et al. (2005) used 25 for population size, 0.1 for crossover probability, these values seem reasonable and are thus adopted in the current study. For the mutation rate, GNMM uses adaptive mutation rate as detailed in Figure 2-8. However, in order to make fair comparisons, these

Table 6-4: GA initial parameter range and step size

	Population size	Number of generations	Crossover probability	Mutation probability
Suggested in Sun, Hines et al. (2005)	25 – 100	100 – 500	0.6 – 0.9	0.01 – 0.1
Initial value	25	100	0.6	0.01
Step size	25		0.1	0.02

values are kept constant in the current study with an incremental step of 0.02. Special attention is paid to the number of generations. Due to the fact that the performance criteria used in the current study is the mean fitness value over the entire generation, it is therefore expected that GA runs with larger generation tend to have smaller mean fitness value (lower MSE). Hence, during the first stage the number of generations is set to be 100, and the effect of the number of generations will be studied after optimal values for the rest parameters are found. The step sizes are also listed in Table 6-4.

6.5.4 Results

Figure 6-14 shows the fitness values over 100 generations for 4 different population sizes (i.e. 25, 50, 75, and 100). Note that in Figure 6-14 fitness means the average fitness value over the entire generation, as mentioned in Section 6.5.2 Determine the Parameter Set. It is clear that although the curve for different GA runs varies slightly, the overall trend is that the fitness value decrease dramatically during the first 10 generations and then oscillates around 0.62 as the GA evolves. Furthermore, the best fitness values achieved by different GA runs are very close – there is no significant difference between the curves after generation 10. This means that in the case of Data VI GAs have found similar best fitness values and hence achieved similar performance. It also implies that the population size does not affect the simulation results much as long as the GA runs over some generations. However, for population size 50, the curve is lower and more stable than the rest, hence 50 is selected to be the optimal population size.

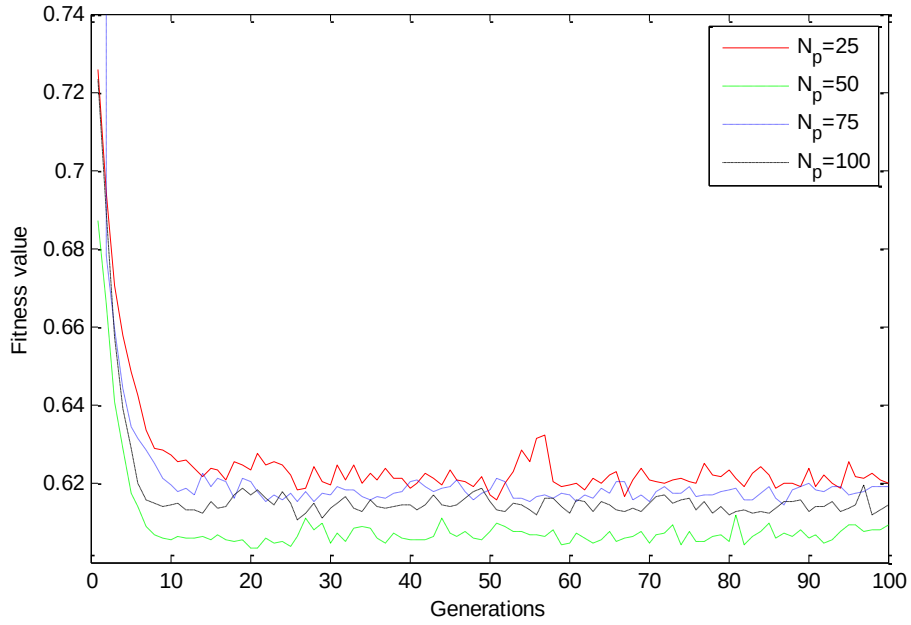


Figure 6-14: Fitness values for 4 different population sizes

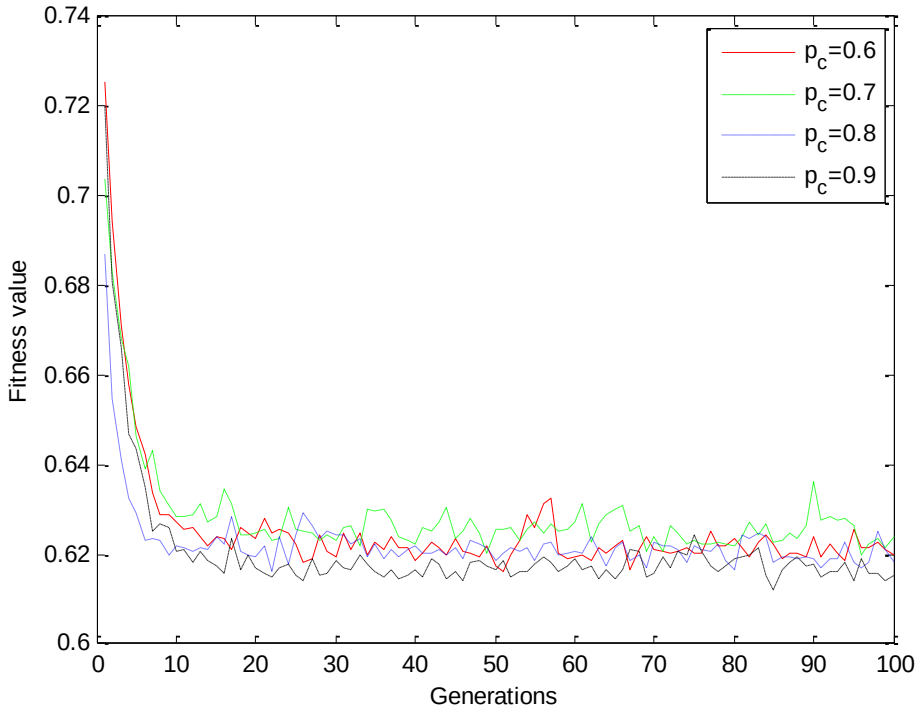


Figure 6-15: Fitness values for 4 different crossover probabilities

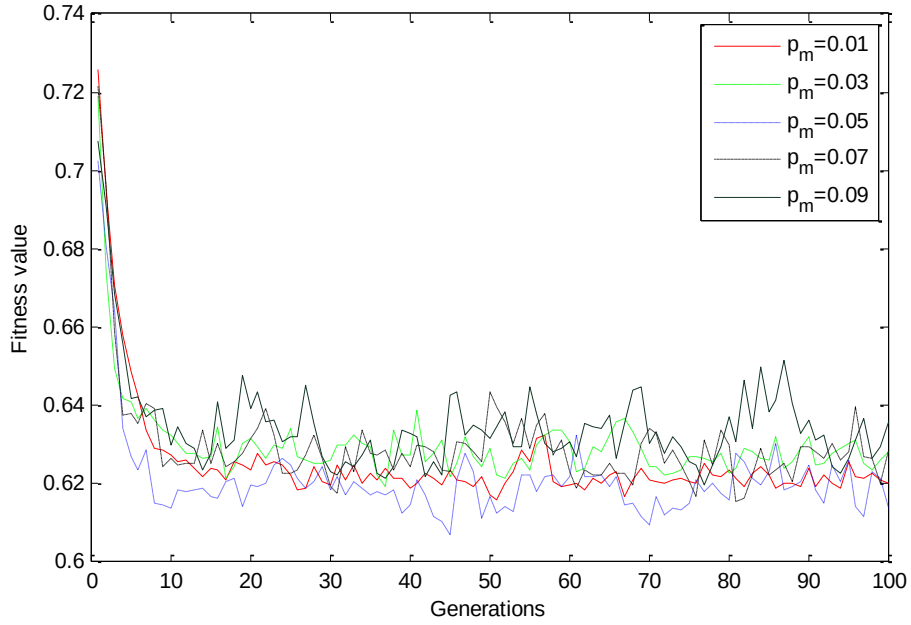


Figure 6-16: Fitness values for 5 different mutation probabilities

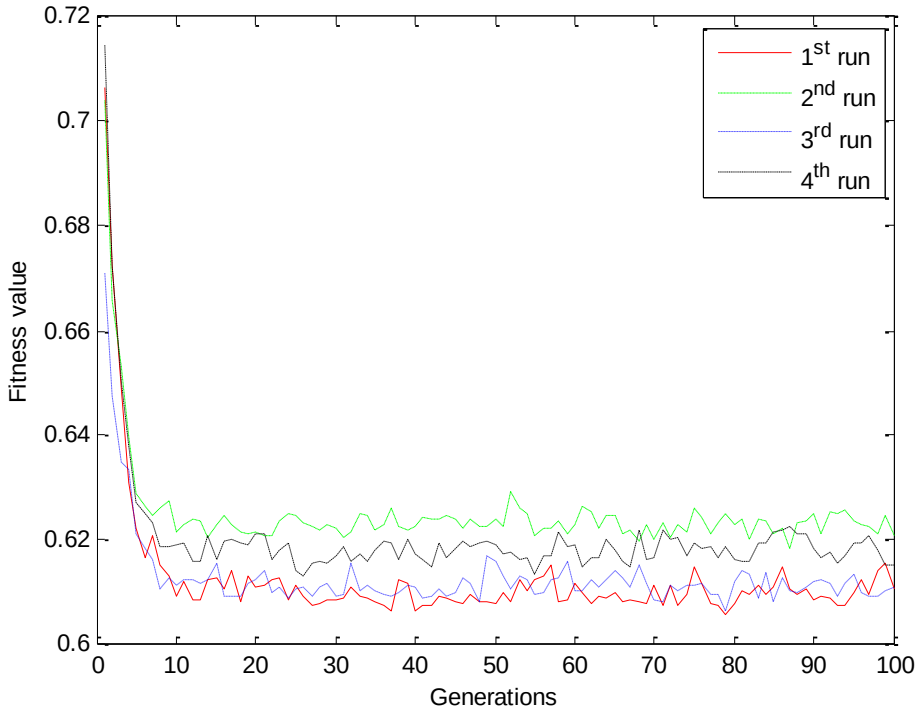


Figure 6-17: Fitness value decreases as generation increases

GA performance for different crossover/mutation probabilities are shown in Figure 6-15 and Figure 6-16 respectively. Similar to the case of population, it is evident that in both figures the fitness value decreases significantly in the first generations and then oscillates around 0.62 as the GA evolves. In Figure 6-15 crossover rate 0.9 produces the lowest fitness value and hence is used as the optimal value. For different mutation probabilities in Figure 6-16, the oscillation is more obvious. However, mutation rate 0.05 appears to produce the lowest RMSE and hence is used as the optimal value.

Generally speaking, increasing the generation number improves the GAs' performance. This can be seen in Figure 6-17, which depicts the decrease of fitness value as generation increases with other parameters being set. According to Figure 6-17, it can be seen that before the point around 30^{th} generation, the fitness value drops sharply; and after the 30^{th} generation, although the general trend is that the fitness value decreases gradually it oscillates too. Figure 6-17 illustrates that if the generation number exceeds 30, it will not have any substantial impact on the performance of the GA. Therefore, 30 is chosen as the optimal generation number.

Therefore, it is decided that the optimal parameter set for Data VI are 50 for population size, 30 for generation size, 0.9 for crossover probability, and 0.05 for mutation probability, which are also summarised in Table 6-5. Comparing with the initial range in Table 6-4, it is evident that the optimal set is mainly in

the middle of the original range.

6.5.5 Discussions

Once the optimal parameter set is determined, the next step is to investigate the averaging effect and whether different parameter sets leads to different input variable selection results. The appearance percentages calculated from GA runs that were used to determine the optimal parameter set in the previous section are shown in Figure 6-18. From these figures it is quite

Table 6-5: Optimal GA parameters for Data VI

Population Size	Number of Generations	Crossover Probability	Mutation Probability
50	30	0.9	0.05

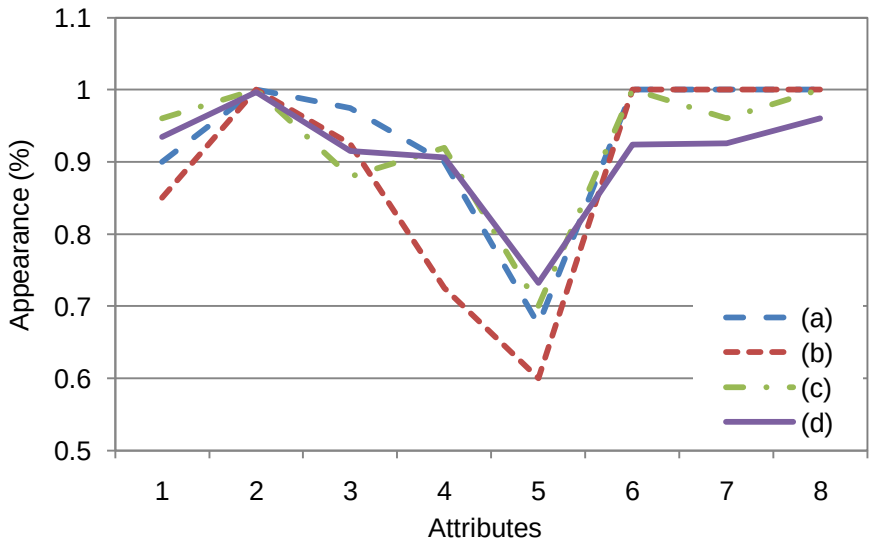


Figure 6-18: Appearance percentage calculated from GAs that were used to determine the optimal population number (a), crossover probability (b), mutation probability (c), and generation number (d)

obvious that although the exact appearance percentage number for each attribute varies for different cases, they all present a similar patter, i.e. Attr2, Attr6, Attr7, and Attr8 are the most often appearing attributes. These variable selection results also confirm our previous results presented in Section 6.3 GNMM Results.

To conclude the current section, the optimal set of GA operators for Data VI was determined experimentally. However, even if the optimal parameter set is being used, the variable selection results are the same as in cases where GA uses non-optimal parameter sets. It is worth noticing that in GNMM GAs are used to accumulate the possibility of a particular input variable to be in the variable combination that produces the minimum error. Therefore, the emphasis is to allow many possible variable combinations to evaluate and evolve and then a possibility can be formed. As long as different input variable combinations are evolving based on different initial random conditions (i.e. different GA runs and different initial MLP settings), the effect of GA parameters can be minimized in GNMM's input selection.

6.6 Summary

In the current chapter, the Pima Indian Diabetes database was used to compare features of GNMM with some other IS DM techniques. A summary of classification results is given in Table 6-3 and feature comparisons of GNMM and these techniques are shown in Table 6-6. The purposes of the current

Table 6-6: Feature comparison of GNMM with other IS DM techniques

	Core technique	Structure		Training		Rule extraction
		Input	Output	Method	Cross-validation	
GNMM	MLP	GA optimization needed	No limits	ICA weights initialization and LM	Yes	By dividing input space
ANFIS	Sugeno-type FIS	Fixed	Fixed to 1*	LS estimator and the gradient descent	Yes	Fuzzy rules, no rule sharing
EFuNN	Mamdani-type FIS	Can evolve over iterations	Can evolve over iterations	Hybrid unsupervised and supervised learning	No	Fuzzy rules, increase dramatically when more data presented
Fuzzy ARTMAP	ART	Fuzzy inputs	No limits	Incremental supervised	Yes	Rule extraction based on committed nodes
CGP	GP	Fixed	No limits	Darwinian evolution theory	No	Arithmetic operators from a pre-defined set

*later research have shown systems based on ANFIS that have multiple outputs, e.g. in Guney and Sarikaya (2008).

chapter are to summarise features of GNMM in the context of hybrid IS DM techniques. Although a comprehensive study would be required to benchmark the performance of GNMM against others, the current study will suffice to review its outstanding characteristics. From Table 6-6 it is evident that compared with FIS based systems such as ANFIS, which has a fixed number of inputs/outputs, GNMM's ANN nature make it fault-tolerant and can have variable or missing inputs/outputs. Compared to other ANN based approaches e.g. EFuNN and Fuzzy ARTMAP, GNMM presents the advantage of producing fewer rules. One obvious drawback of GNMM is that it is very computationally expensive, which also holds true for the other EC technique in comparison i.e. the CGP. However, the merit of GNMM compared with CGP is that CGP does not have a way to cross-validate the training process. Hence it may suffer from the problem of over-fitting.

In the current chapter, the influences of GA parameter settings in GNMM's variable selection stage were also studied. We have identified the optimal GA parameter set for Data VI. However, it has been shown that the influences of GA parameter can be minimized as long as different input variable combinations can be tested and evolve towards a better fitness value

References

Abraham, A. and B. Nath (2001). "A neuro-fuzzy approach for modelling electricity demand in Victoria." Applied Soft Computing 1(2): 127-138.

Andres-Andres, A., E. Gomez-Sanchez, et al. (2005). Incremental rule pruning for fuzzy ARTMAP neural network, Warsaw, Poland, Springer Verlag.

Boilot, P., E. L. Hines, et al. (2000). "Knowledge Extraction from Electronic Nose Data Sets Using Hybrid Neuro-fuzzy Systems." Sensors Update 8(1): 73-94.

Boilot, P., E. L. Hines, et al. (2002). "Classification of bacteria responsible for ENT and eye infections using the cyranose system." IEEE Sensors Journal 2(3): 247-252.

Cakir, M., E. Butun, et al. (2006). "Effects of genetic algorithm parameters on trajectory planning for 6-DOF industrial robots." Industrial Robot 33(3): 205-215.

Carpenter, G. A., S. Grossberg, et al. (1992). "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps." IEEE Transactions on Neural Networks 3(5): 698-713.

Carpenter, G. A. and A. H. Tan (1995). "Rule Extraction: From Neural Architecture to Symbolic Representation." Connection Science 7(1): 3-3.

Connolly, J.-F., E. Granger, et al. (2008). Supervised incremental learning with the fuzzy ARTMAP neural network, Paris, France, Springer Verlag.

Costa, C. B. B., M. R. W. MacIel, et al. (2005). "Factorial design technique applied to genetic algorithm parameters in a batch cooling crystallization optimisation." Computers and Chemical Engineering 29(10): 2229-2241.

De Jong, K. (2005). "Genetic algorithms: A 30 year perspective." Perspectives on Adaptation in Natural and Artificial Systems: 11.

De Jong, K. (2007). Parameter Setting in EAs: a 30 Year Perspective. Parameter Setting in Evolutionary Algorithms: 1-18.

De Jong, K. A. (2006). Evolutionary computation: a unified approach. Cambridge, Mass., MIT Press.

Deb, K. and S. Agrawal (1999). "Understanding interactions among genetic algorithm parameters." Foundations of Genetic Algorithms 5: 265-286.

Eggermont, J., J. N. Kok, et al. (2004). Genetic programming for data classification: Partitioning the search space, Nicosia, Cyprus, Association for Computing Machinery.

Eiben, A., Z. Michalewicz, et al. (2007). Parameter Control in Evolutionary Algorithms. Parameter Setting in Evolutionary Algorithms: 19-46.

Fogel, D. B., T. Bäck, et al. (2000). Evolutionary computation. Bristol; Philadelphia, Institute of Physics Publishing.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Reading, Mass., Addison-Wesley Pub. Co.

Granger, E., J. F. Connolly, et al. (2008). A comparison of fuzzy ARTMAP and Gaussian ARTMAP neural networks for incremental learning. Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on.

Guney, K. and N. Sarikaya (2008). "Multiple adaptive-network-based fuzzy inference system for the synthesis of rectangular microstrip antennas

with thin and thick substrates." International Journal of RF and Microwave Computer-Aided Engineering 18(4): 359-375.

Haupt, R. L. and S. E. Haupt (2004). Practical genetic algorithms. Hoboken, N.J., John Wiley.

Hughes, J. and B. Ruprai (1999). Distributed genetic programming, Google Patents.

Kahramanli, H. and N. Allahverdi (2008). "Design of a hybrid system for the diabetes and heart diseases." Expert Systems with Applications 35(1-2): 82-89.

Karray, F. O. and C. W. De Silva (2004). Soft computing and intelligent systems design : theory, tools, and applications. Harlow, England ; New York, Pearson/Addison Wesley.

Kasabov, N. K. (2007). Evolving connectionist systems: the knowledge engineering approach. London, Springer.

Lal, T. N., T. Hinterberger, et al. (2005). "Methods Towards Invasive Human Brain Computer Interfaces." Advances in neural information processing systems.(17): 737-744.

Leondes, C. T. (1999). Fuzzy theory systems : techniques and applications. San Diego, CA, Academic Press.

Lin, T.-H. and V.-W. Soo (1997). Pruning fuzzy ARTMAP using the minimum description length principle in learning from clinical databases, Newport Beach, CA, USA, IEEE.

Lobo, F. G., C. F. Lima, et al. (2007). Parameter setting in evolutionary algorithms. Berlin; New York, Springer.

Mahadevan, I. and C. Raghavendra (1997). Admission Control in ATM Networks using Fuzzy-ARTMAP, Lawrence Erlbaum.

Man, K. F., K. S. Tang, et al. (1999). Genetic algorithms: concepts and designs. London; New York, Springer.

Maturana, J. and F. Saubion (2008). A Compass to Guide Genetic Algorithms. Parallel Problem Solving from Nature – PPSN X: 256-265.

Michael Johnson, J. and Y. Rahmat-Samii (1997). "Genetic algorithms in engineering electromagnetics." IEEE Antennas and Propagation Magazine 39(4): 7-25.

Ozkan, C. (2006). Surface interpolation by adaptive neuro-fuzzy inference system based local ordinary kriging, Hyderabad, India, Springer Verlag.

Schaffer, J. D., R. A. Caruana, et al. (1989). "A study of control parameters affecting online performance of genetic algorithms for function optimization." Proceedings of the third international conference on Genetic algorithms: 51-60.

Schmutter, P. (2002). Object oriented ontogenetic programming: breeding computer programms that work like multicellular creatures. Dortmund, Univ., Systems Analysis Research Group.

Smith, J., J. Everhart, et al. (1988). Using the ADAP learning algorithm to forecast the onset ofdiabetes mellitus. Proceedings of the Symposium on Computer Applications and Medical Care, Computer Society Press.

Sun, L., E. L. Hines, et al. (2005). "Quarter-wave phase-compensating multidielectric lens design using genetic algorithms." Microwave and Optical Technology Letters 44(2): 165-169.

Tayfur, G. and V. P. Singh (2005). "Predicting longitudinal dispersion coefficient in natural streams by artificial neural network." Journal of Hydraulic Engineering 131(11): 991-1000.

Tian, D., Y. Liu, et al. (2006). SLNN: A neural network for fuzzy neural network's structure learning, Jinan, China, Inst. of Elec. and Elec. Eng. Computer Society.

Vajda, P., A. Eiben, et al. (2008). Parameter Control Methods for Selection Operators in Genetic Algorithms. Parallel Problem Solving from Nature – PPSN X: 620-630.

Vilakazi, C. B. and T. Marwala (2009). Computational Intelligence Approach to Condition Monitoring: Incremental Learning and Its Application. Intelligent Engineering Systems and Computational Cybernetics: 161-171.

Chapter 7 Conclusions and Future Work

This chapter summarises the main findings of this research and presents the conclusions that have been formed. It also includes suggestions for further work.

7.1 Results Overview

7.1.1 GNMM Steps

The GNMM method consists of three main steps: (1) A GA-based input variable selection; (2) MLP-based input/output mapping/classification; and (3) mathematical programming based regression rule extraction. The functionality of GNMM can be summarized as follows:

(1) Utilizing GAs to optimize input variables, this simplifies the MLP structure in GNMM, and makes the training process more efficient. The evaluation of the fitness for each input variable combination is determined via the training error (RMSE) when such an input combination is used in an MLP to perform the classification/prediction task. Since weights and thresholds for the MLP are randomly generated, GAs have to be run several times

until a clear distinction is evident between input variables as far as possible.

(2) The input variables found by the GA in conjunction with the associated targets are then used to develop an MLP. As in the previous step, the training has to be repeated several times in order to get satisfactory results due to its 'random' starting point. However, the learning rate can be set to a relatively large value in order to accelerate the training process.

(3) Extracting regression rules from the trained MLP neural network, which makes the training results much more transferable. Since the original data have been mapped into a specific range in pre-processing before the MLP is trained, rules extracted from the trained MLP have to reflect this feature (i.e. reversely map the rule results into normal ranges).

7.1.2 Case Study Results

A total of six datasets were used in the case study part of the thesis to illustrate the implementation and demonstrate the usefulness of GNMM. A summary of these case study data and results are shown in Table 7-1. These datasets belong to two categories i.e. environmental and medical, and are concerned with prediction and classification.

Data I & II are concerned with the prediction of longitudinal dispersion coefficients, which was dealt with in Chapter 3. Data III and IV are concerned

Table 7-1: A summary of case study data and results

		Dataset		GNMM results			Benchmarking literature	
		Nature	Dimension (attribute × sample)	Input selection	Best results (testing data)	Rule extraction	Methods	Results (classification rate)
Environmental data	I	Longitudinal dispersion coefficient prediction	49 × 196	2 out of 49	72% (R^2)	11 rules	N/A	N/A
	II	Longitudinal dispersion coefficient prediction	8 × 71	3 out of 8	89% (R^2)	13 rules	MLP (Tayfur and Singh 2005)	70% (R^2)
Medical data	III	ECoG classification	64 × 834000	10 out of 64	95.32%	431 rules	RFE and SVM (Lal, Hinterberger et al. 2005)	74.3%
	IV	EEG classification	32 × 428360	12 out of 32	80%	516 rules	N/A	N/A
	V	Eye bacteria species classification	32 × 180	6 out of 32	93%	87 rules	Integer based GA (Boilot, Hines et al. 2002)	90.6%
	VI	Diabetes classification	8 × 268	4 out of 8	79.30%	75 rules	ANFIS, EFuNN, Fuzzy ARTMAP, CGP	See Table 6-3

with EEG classification problems and was discussed in Chapter 4. Chapter 5 deals with EN sensed eye bacteria data. And finally Chapter 6 is concerned with a diabetes classification problem.

Although GNMM was applied to all six data sets, the emphasis is different for different chapters. For example, the emphasis of Chapter 3 was to give a detailed illustration of how GNMM works; Chapter 4 shows how to deal with difficult classification problems; the aim of Chapter 5 was to illustrate the averaging effect of GNMM; and finally Chapter 6 was concerned with comparing GNMM with other IS DM techniques. Datasets obtained from published works (i.e. Data II & III) or public domains (i.e. Data VI) where previous results are present in the literature were also used to summarise GNMM's features.

7.1.3 Advantages/Disadvantages

The idea of combining GAs with ANNs is not novel. What is novel about GNMM is that it also combines the mathematical programming based rule extraction as well as recent developments in the field such as ICA-based weight initialization. All these elements make GNMM an effective system that is capable of handling large amount of noisy data especially when the underlying relationships within the data are not fully understood.

GNMM is distinct from other solely ANN-based methods by also incorporating

variable selection and rule extraction. It benefits from GA's randomness – by setting different initial conditions the optimization starts from an arbitrary point in the search space. In this way each input variable accumulates its possibility to appear in the winning chromosome. The GA-based variable selection stage is capable of:

- Filtering out irrelevant and noisy variables, improving the accuracy of the model.
- Making the ANN structure less complex and easier to understand.
- Reducing the computational complexity and memory requirements.

Rule extraction is the attempt to overcome the 'black box' reputation that comes with ANNs. Such a process not only provides a facility that may help to explain the internal behaviour of an ANN, may help in understanding the underlying physical phenomena, but may also make the training results easily applicable/transferable.

As opposed to the above analysis which looks into GNMM's individual steps, as a closely integrated system GNMM has the merit that it needs little human interaction. With some predefined parameters, such as GA's crossover probability and the shape of ANNs' activation functions, GNMM is able to process raw data until some human-interpretable rules being extracted. This is an important feature in terms of practice as quite often users of a DM system

have little or no need to fully understand the internal components of such a system.

However, based on the analysis and case study applications throughout the thesis, it is the opinion of the author that GNMM as an IS DM technique has disadvantages depending, for example, on the problem being solved. An obvious problem is that determining the parameter values for GA is always data-dependent. Although the general guidelines exist, for example, small population is to be combined with large generation. However, to what extend a population number is sufficiently small is still arguable.

Furthermore, the GA optimization is based on iterations and hence very computationally expensive. The power of GAs (or stochastic optimization/randomized FS) will overtake that of non-random research only when the search space is large. However, as the name suggests, GAs have to be given enough time to 'evolve' their solutions to an optimal or sub-optimal. In case of small input space, GNMM may not be efficient in determining the optimal input subset of its MLP modelling.

7.2 Future research directions

In GNMM, rule extraction is based on the approximation of the hidden neurons' hyperbolic tangent activation function. Such an approximation is derived through the numerical analysis of Sequential Quadratic Programming.

As in any approximation, there are always associated errors. Thus, methods that extract regression rules from ANN with higher accuracy are desirable. Since neural networks are low-level computational structures that perform well when dealing with raw data, while fuzzy logic deals with reasoning on a higher level, using linguistic information acquired from domain experts, rule extraction from such a hybrid neuro-fuzzy system would be easier and more accurate. In particular, for example the EFuNN proposed by Kasabov (2001) implements a strategy of dynamically growing and pruning the connectionist (i.e. ANN) architecture. Therefore, a system that integrates GNMM and EFuNN would offer a promising approach to data modelling and rule extraction.

Moreover, GNMM as a data driven method relies heavily on the quality of the data. Typically, real-life data must not only be cleaned of errors and redundancy, but must also be organized in a fashion that makes sense in the context of the application. There exist problems in raw input data needed for knowledge acquisition, mainly due to uncertainty, vagueness, and incompleteness. While incompleteness arises due to missing or unknown data, uncertainty (or vagueness) can be caused by errors in physical measurements due to incorrect measuring devices or by a mixture of noisy and pure signals (Mitra and Acharya 2003). Thus, future works may also include applications of GNMM to some incomplete and highly noisy data.

References

Boilot, P., E. L. Hines, et al. (2002). "Classification of bacteria responsible for ENT and eye infections using the cyranose system." IEEE Sensors Journal **2**(3): 247-252.

Kasabov, N. (2001). "Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning." IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **31**(6): 902-918.

Lal, T. N., T. Hinterberger, et al. (2005). "Methods Towards Invasive Human Brain Computer Interfaces." Advances in neural information processing systems.(17): 737-744.

Mitra, S. and T. Acharya (2003). Data mining: multimedia, soft computing, and bioinformatics. Hoboken, NJ, John Wiley.

Tayfur, G. and V. P. Singh (2005). "Predicting longitudinal dispersion coefficient in natural streams by artificial neural network." Journal of Hydraulic Engineering **131**(11): 991-1000.

Appendix A Data I – UK Environmental Agency Data

A.1 PART I

ID	Cs	Ds	Ms	Qs	S	L	Dr
SW_12	0	1	0	0	1.27E-05	1058	915.57
SW_13	0	1.5	0	0	1.27E-05	1058	915.57
SW_12	0	1.5	0	0	2.15E-05	1058	915.57
SW_13	0	1.5	0	0	0.000118	1101.83	993
SW_10	0	1.5	0	0	0.000154	1472.13	1282
SW_11	0	2.2	0	0	0.000154	1472.13	1282
SW_14	0	2.2	0	0	0.000154	1977.5	1506
SW_02	0	2.2	0	0	0.000154	2127	1506
SW_11	0	2.5	0	0	0.000176	2127	1506
SW_14	0	3	0	0	0.000176	2186.24	1935.43
SW_02	0	3.4	0	0	0.000179	2208.5	2005
SW_14	0	3.4	0	0	0.000179	2208.5	2005
SW_06	0	3.5	0	0	0.000179	2208.5	2017.8
SW_18	0	3.5	0	0	0.000186	2412	2039.11
SW_07	0	3.5	0	0	0.000197	2412	2039.11
SW_21	0	3.5	0	0	0.000198	2969.12	2039.11
SW_21	0	4	0	0	0.000198	2969.12	2060
MID_04	0	4	0	0	0.000253	3016.57	2067
MID_05	0	4	0	0	0.000269	3016.57	2067
MID_06	0	4	0	0	0.000269	3230	2296.8
MID_12	0	4	0	0	0.000349	3253	2296.8
MID-01	0	4	0	0	0.000349	3287.56	2296.8
MID-01	0	4	0	0	0.00041	3287.56	2331
SW_22	0	4	0	0	0.00041	3343.2	2331
MID_04	0	4	0	0	0.000426	3530.1	2612
MID_05	0	4	0	0	0.000426	3538.7	2757
MID_04	16	4	0	0	0.000426	3538.7	2757
MID_06	34.5	4	0	0	0.000464	3538.7	2757
MID_04	78.75	4	0	0	0.000464	3728.37	2770
MID_05	92.6	4	0	0	0.000492	4034.28	2770
MID_06	106	4	0	0	0.000508	4083.59	2907
AN_13	106.25	4.4	0.183	0.0208	0.000512	4268.68	2907
NE_47	106.25	4.5	0.211	0.0255	0.000512	4293.44	2920
NE_59	106.25	4.5	0.296	0.0479	0.000512	4293.44	2946

NE_63	113.25	4.5	0.76	0.074	0.000535	4293.44	2956.5
NE_62	113.25	4.5	0.76	0.074	0.000535	4398	3015
NE_32	113.25	4.5	1.091	0.127	0.000544	4591.85	3015
NE_47	123.5	5	1.196	0.14	0.000544	4591.85	3477
NE_59	132.75	5	1.359	0.163	0.000567	4650.52	3555.6
NE_61	132.75	5	1.631	0.166	0.000606	4658	3555.6
NE_47	149.4	5	1.631	0.166	0.000606	4658	3600
NE_59	149.4	5.1	1.631	0.166	0.000618	4658	3602
NE_61	153	5.1	1.736	0.171	0.000618	4658.4	3602
NE_29	153	5.1	1.736	0.171	0.000618	4692	3617.23
NE_30	158.5	6	1.736	0.171	0.000618	4692	3686.43
NE_32	158.5	6	1.76	0.192	0.00068	4692	3728
AN_12	189	6	2.048	0.192	0.000706	4738.6	3728
AN_13	189	6	2.048	0.215	0.000706	4969	3817
NE_42	191.7	6	2.049	0.215	0.000706	5093.44	3817
NE_45	193.25	6	2.049	0.221	0.000719	5093.44	3817
NE_58	193.25	6	2.103	0.221	0.000719	5093.44	3880.2
NE_29	221	6.2	2.103	0.311	0.000761	5093.44	3930
NE_30	262	6.5	2.868	0.342	0.000898	5319	3942.6
NE_42	262	6.5	2.868	0.342	0.000904	5319	3942.6
NE_45	269.25	6.5	2.917	0.347	0.000904	5596.46	4007
NE_42	269.25	6.5	2.917	0.347	0.000904	5596.46	4007
NE_45	269.25	7	4.335	0.463	0.001006	5596.46	4007
NE_33	269.25	7	4.335	0.463	0.001094	5599	4007.4
NE_33	269.25	7.5	6.076	0.725	0.001116	5599	4007.4
NE_01	272	7.5	6.924	0.847	0.001116	5599	4007.4
NE_02	282.25	7.5	6.924	0.879	0.001116	5738.34	4022.29
NE_41	299.5	7.5	7.658	0.886	0.001125	5738.34	4196.75
NE_49	299.5	7.5	7.679	0.886	0.001239	5747.8	4196.75
NE_01	307.75	8.5	7.679	0.965	0.001319	5747.8	4196.75
NE_07	338.5	8.5	7.679	0.965	0.001342	5931.38	4196.75
NE_13	338.5	8.5	8.265	0.965	0.001342	5931.38	4271.67
NE_03	338.5	9	8.265	1.134	0.001402	6124.27	4271.67
NE_06	338.5	9.2	8.265	1.153	0.001481	6124.27	4271.67
NE_12	338.5	9.2	8.295	1.179	0.001481	6184.28	4271.67
NE_34	343	9.5	8.38	1.179	0.001505	6184.28	4271.67
NE_50	343	9.5	8.38	1.197	0.001595	6184.28	4418.29
NE_34	370	10	8.426	1.197	0.001692	6184.28	4465.7
NE_35	370	10	9.553	1.197	0.001766	6184.28	4465.7
NE_36	407.5	10	9.553	1.259	0.001766	6414.4	4465.7
NE_37	407.5	10	9.553	1.259	0.001856	6468.39	4476.32
NE_38	407.5	10	9.553	1.357	0.001856	6468.39	4476.32
NE_39	407.5	10	9.553	1.393	0.001856	6497.3	4486.5
NE_35	460	10	10.003	1.393	0.002264	6497.3	4504.4
NE_36	485	10	10.003	1.393	0.002264	6497.3	4516.7
NE_37	485	10	11.182	1.393	0.002264	6497.3	4688.2
NE_38	485	10.1	11.182	1.393	0.002522	6960.6	4688.2
NE_39	489.75	10.2	11.182	1.501	0.002534	6960.6	4688.2
NE_09	489.75	10.5	11.182	1.501	0.002534	6960.6	4688.2
NE_35	505.75	10.5	11.182	1.501	0.00261	6960.6	4755
NE_36	505.75	10.5	11.218	1.501	0.00261	7005.7	4755
NE_37	505.75	10.5	12.034	1.501	0.002749	7005.7	4910

NE_38	525.75	10.5	12.478	1.621	0.002749	7005.7	4910
NE_31	525.75	11	12.478	1.621	0.002749	7074.8	4910
NE_49	563.5	11	12.478	1.621	0.002749	7074.8	4980
NE_49	617	11.25	12.478	1.621	0.002857	7093.8	4980
NE_22a	658.5	11.25	13.754	1.686	0.002857	7313.7	5110.4
NE_22a	676.5	12	16.487	1.927	0.003165	7586.5	5110.4
NE_17	851.25	12	16.487	2.898	0.003396	7586.5	5110.4
NE_19	898.25	12.5	16.487	3.086	0.003396	7596.8	5110.4
NE_20	901.5	12.5	16.487	3.364	0.003396	7690.74	5118.4
NE_21	913.25	12.5	16.546	3.364	0.003396	7690.74	5118.4
NE_22a	913.25	13	16.546	3.364	0.003396	7690.74	5760
NE_17	913.25	14.5	16.546	3.364	0.003647	7690.74	5760
NE_20	925.75	14.7	18.284	3.409	0.003647	7690.74	5796.3
NE_21	925.75	15	18.284	3.409	0.00365	7700	6027.87
NE_17	1116	15	18.284	3.409	0.00365	7700	6027.87
NE_20	1116	15	18.671	3.498	0.00365	8037.8	6027.87
NE_21	1116	16	18.671	3.59	0.0038	8037.8	6027.87
NE_08	1116	16.5	18.671	3.59	0.004031	8090.7	6027.87
NE_40	1126.75	16.5	18.671	3.59	0.004031	8199.6	6205.4
NE_17	1126.75	16.5	19.485	3.615	0.004031	8199.6	6315
NE_19	1126.75	16.7	19.485	3.615	0.004031	8453.62	6376.4
NE_20	1164.5	17.5	19.652	3.615	0.004031	8560	6376.4
NE_21	1361.25	17.5	20.042	3.615	0.004146	9133.4	6454.6
NE_08	1361.25	17.5	20.042	3.717	0.004146	9243.43	6560
NE_08	1361.25	17.5	20.042	3.717	0.004209	9402.3	6635.11
NE_22b	1417	17.5	20.077	3.722	0.004721	9627.2	6762.6
NE_24	1417	17.5	20.11	3.787	0.004721	9627.2	6762.6
NE_26	1417	19	20.11	4.117	0.004726	10017.4	6780
NE_27	1417	19	20.37	4.337	0.004726	10255.6	6825.6
NE_23	1879.5	19.25	20.502	4.337	0.004726	10255.6	6839.19
NE_43	1879.5	19.5	20.663	4.337	0.00485	10255.6	6839.19
NE_23	1908.75	20	22.002	4.38	0.005036	10518.4	6987.57
NE_23	1944.25	20	24.165	4.38	0.006028	10525.16	7200
NE_24	1944.25	20	39.57813	4.575	0.006281	10705.7	7200
NE_26	1982.5	20	39.57813	4.575	0.00644	10967.35	7406.6
NE_14	2135.25	22.5	39.691	4.866	0.007049	10967.35	8668.6
NE_24	2179	28	44.159	4.977	0.007963	12516	8668.6
NE_06	2480.5	28.5	44.159	8.079468	0.012405	12516	8700.6
NE_12	2480.5	33	46.442	8.079468	0.013051	12886.68	8904.8
NE_16	2501	34.5	46.86	8.111	0.013051	13777.2	9334.18
NE_15	3314.75	41.5	49.547	9.473	0.024371	14697	12133.5

A.2 PART II

ID	Ce	De	Me	Qe	Cg	A	Mg	Qg	I

SW_12	9.25	3.4	0	0	20	0.444	0	0.06	0.5
SW_13	20.2	3.4	0	0	47.6	0.664	0	0.074	0.575
SW_12	20.2	3.4	0	0	74.9	0.664	0	0.1	0.575
SW_13	27.5	3.5	0	0	74.9	1.093	0	0.106	0.6
SW_10	34	4.5	0	0	112.7	1.21	0	0.106	0.636
SW_11	42.25	4.5	0	0	120	1.21	0	0.106	0.684
SW_14	43	4.5	0	0	120	1.423	0	0.15	0.753
SW_02	45.75	5.1	0	0	120	1.423	0	0.15	0.753
SW_11	113.25	5.1	0	0	149.4	1.423	0	0.21	0.753
SW_14	113.25	6	0	0	171.9	1.429	0	0.21	0.753
SW_02	113.25	6	0	0	171.9	1.7	0	0.21	1.28
SW_14	123.5	6.5	0	0	178.5	1.7	0	0.21	1.28
SW_06	132.75	6.5	0	0	178.5	2.091	0	0.21	1.3
SW_18	132.75	6.5	0	0	262	2.091	0	0.231	1.45
SW_07	132.75	7	0	0	262	2.091	0	0.3	1.75
SW_21	158.5	7.5	0	0	262	2.095	0	0.3	1.75
SW_21	158.5	7.5	0	0	262	2.095	0	0.311	1.806
MID_04	178	7.63	0	0	262	2.391	0	0.311	1.806
MID_05	178	8	0	0	262	3.16	0.477	0.48	1.806
MID_06	181.75	8.5	0	0	262	3.16	0.477	0.48	2.08
MID_12	191.7	8.5	0	0	262	3.16	0.527	0.498	2.082
MID-01	193.25	8.5	0	0	262	3.16	0.753	0.564	2.082
MID-01	193.25	9.25	0	0	262	3.16	0.753	0.564	2.082
SW_22	262	9.5	0	0	287.3	3.28	0.753	0.564	2.1
MID_04	262	9.5	0	0	287.3	3.28	0.753	0.564	2.1
MID_05	262	9.5	0	0	287.3	3.28	1.28	0.564	2.1
MID_04	283	9.5	0	0	287.3	3.65	1.28	0.564	2.2192
MID_06	283	10	0	0	287.3	3.65	1.3	0.564	2.2192
MID_04	283	10	0	0	299.5	3.674	1.423	0.564	2.242
MID_05	299.5	10	0	0	299.5	3.684	1.423	0.564	2.242
MID_06	299.5	10	0	0	299.5	3.684	1.423	0.564	2.242
AN_13	307.75	10	0	0	299.5	3.684	1.45	0.564	2.3
NE_47	329.75	10	0.393	0.025544	299.5	3.684	1.75	0.564	2.3
NE_59	329.75	10.1	0.461	0.025544	299.5	4.208	1.75	0.564	2.8
NE_63	338.5	10.5	1.359	0.0591	299.5	5.026	2.08	0.564	2.8
NE_62	338.5	10.5	1.736	0.0599	299.5	5.026	2.082	1.1	2.8
NE_32	338.5	10.5	1.736	0.163	299.5	5.026	2.082	1.1	2.807
NE_47	338.5	10.5	1.736	0.171	299.5	5.026	2.082	1.1	2.807

NE_59	338.5	10.5	1.76	0.171	299.5	5.026	2.091	1.1	3
NE_61	370	10.5	1.76	0.171	299.5	5.026	2.091	1.1	3
NE_47	370	11	1.968	0.192	299.5	5.026	2.091	1.1	3.681
NE_59	407.5	11	2.048	0.192	299.5	5.026	2.1	1.1	3.681
NE_61	407.5	11	2.048	0.192	370	5.026	2.1	1.1	3.681
NE_29	407.5	11.25	2.048	0.221	370	5.026	2.209	1.1	3.733
NE_30	407.5	11.25	2.103	0.221	378	5.026	2.209	1.1	3.733
NE_32	407.5	11.25	2.103	0.233	422.75	5.026	2.242	1.24	3.733
AN_12	409	11.5	2.590972	0.311	422.75	5.026	2.242	1.24	3.736
AN_13	425.5	12	2.590972	0.311	422.75	5.026	2.242	1.24	3.736
NE_42	431.25	12	2.917	0.347	422.75	7.415	2.3	1.25	3.736
NE_45	431.25	12.5	2.917	0.347	422.75	7.415	2.3	1.25	4.723
NE_58	431.25	13	3.664	0.413	422.75	7.415	2.8	1.25	4.951
NE_29	431.25	13	3.664	0.413	422.75	7.415	2.8	1.25	4.951
NE_30	460	14.5	3.664	0.413	422.75	7.415	2.8	1.317	5.297
NE_42	498	14.5	4.335	0.463	422.75	7.415	2.807	1.317	5.297
NE_45	499	14.5	4.335	0.463	422.75	7.415	2.807	1.317	5.297
NE_42	499	14.7	4.644	0.495	422.75	7.71	3.681	1.317	5.297
NE_45	525.75	15	4.644	0.495	422.75	7.71	3.681	1.317	5.3
NE_33	525.75	15	7.658	0.879	422.75	7.71	3.681	1.317	5.418
NE_33	525.75	15	8.222072	1.104	422.75	8.38	3.684	1.317	5.418
NE_01	563.5	15	8.222072	1.134	455	8.38	3.684	1.474	5.418
NE_02	563.5	15	8.38	1.134	455	8.38	3.684	1.572	5.46
NE_41	563.5	15.5	8.38	1.134	455	8.38	3.684	1.572	5.46
NE_49	591	15.5	8.38	1.153	499	11.17	3.736	1.572	5.487
NE_01	596.25	15.9	8.426	1.161	499	13.337	3.736	1.572	5.487
NE_07	611.6	15.9	8.426	1.19478	499	13.337	3.736	1.572	5.487
NE_13	611.6	16	8.426	1.19478	499	13.337	4.208	1.572	5.544
NE_03	617	16	8.699	1.259	499	13.647	4.723	1.572	5.544
NE_06	617	16	9.355	1.259	499	13.647	4.951	1.572	5.7
NE_12	621.1	16	11.182	1.259	499	13.647	4.951	1.572	6.138
NE_34	621.1	16.5	11.182	1.357	514.75	13.647	5.3	1.572	6.411
NE_50	621.1	16.5	11.182	1.357	514.75	13.647	5.418	1.572	7.5
NE_34	640.7	16.5	11.182	1.409	514.75	13.647	5.418	1.572	7.51

NE_35	660.5	16.5	11.182	1.501	569.8	13.647	5.418	1.572	8
NE_36	670.7	16.7	11.218	1.501	569.8	13.647	5.46	1.572	8
NE_37	676.5	17	11.218	1.501	569.8	13.647	5.46	1.63	10
NE_38	774.75	17	11.551	1.501	569.8	13.647	5.487	2.01	10
NE_39	852	17.5	12.034	1.501	591	13.647	5.487	2.3	10
NE_35	854.75	17.5	12.478	1.621	657.8	13.647	5.487	2.3	10.21
NE_36	901.5	17.5	12.478	1.621	851.25	13.647	5.544	2.39	10.33
NE_37	925.75	17.5	12.478	1.621	852	13.647	5.544	2.39	10.33
NE_38	925.75	17.5	12.478	1.621	852	14.453	5.7	2.39	10.33
NE_39	925.75	17.5	12.478	1.621	915	14.453	6.138	2.779	10.33
NE_09	979.75	17.5	12.499	1.629	915	14.453	6.411	2.779	10.467
NE_35	986	19	12.812	1.663	915	14.453	7.5	2.779	10.467
NE_36	986	19	12.812	1.663	915	17	10	2.779	10.467
NE_37	1017.25	19	12.812	1.663	915	17	10	2.779	10.467
NE_38	1126.75	19	12.812	1.663	915	17	10	2.779	11
NE_31	1126.75	19.4	14.316	1.809	1007.5	17	10.21	2.9	11
NE_49	1126.75	19.5	16.546	1.927	1007.5	17	10.33	2.9	11
NE_49	1126.75	19.5	16.546	2.224	1007.5	17	10.33	2.9	11
NE_22a	1229.5	19.6	16.546	3.086	1360	17	10.33	2.9	11
NE_22a	1283	20	16.546	3.353	1360	17	10.33	2.9	11.5
NE_17	1283	20	18.284	3.409	1360	17	10.467	2.9	11.5
NE_19	1361.25	20	18.284	3.409	1360	17	10.467	2.9	11.5
NE_20	1361.25	20.5	18.284	3.409	1360	17	10.467	2.9	12.036
NE_21	1361.25	20.5	18.671	3.409	1360	17.564	10.467	2.9	12.036
NE_22a	1417	21.5	18.671	3.509	1360	19.576	11.5	2.9	12.485
NE_17	1417	21.5	18.671	3.59	1360	19.576	11.5	2.9	12.485
NE_20	1417	22.5	18.911	3.59	1360	19.576	11.5	3.02	12.485
NE_21	1452	22.5	18.911	3.59	1360	19.576	12.036	3.02	12.5
NE_17	1452	22.5	18.911	3.615	1360	19.576	12.036	3.02	12.5
NE_20	1452	22.5	18.911	3.615	1360	19.576	12.485	3.02	13.927
NE_21	1462.75	22.5	19.652	3.615	1360	19.576	12.485	3.02	15
NE_08	1462.75	22.5	19.652	3.722	1360	19.576	12.485	3.02	15

NE_40	1462.75	22.5	20.077	3.722	1552	19.576	12.5	3.02	18
NE_17	1462.75	22.5	20.11	3.755	1552	19.576	12.5	3.02	18.595
NE_19	1612	23.3	20.11	3.755	1552	19.576	13.927	3.02	18.595
NE_20	1612	23.9	20.11	3.755	1552	19.576	15	3.02	18.595
NE_21	1617	25	20.179	3.755	1552	19.576	15	3.02	18.595
NE_08	1617	25.4	20.37	3.787	1552	19.576	18	3.02	19
NE_08	1671	26	20.663	3.82	1552	19.9	18.595	3.02	19
NE_22b	1908.75	26.5	21.55	4.114	1552	20.31	18.595	3.02	20
NE_24	1908.75	26.5	22.019	4.38	1552	20.31	18.595	3.565	20
NE_26	1982.5	26.5	22.159	4.38	1552	20.31	18.595	3.565	20
NE_27	2000.75	26.5	24.454	4.38	1552	21.969	19	3.565	20
NE_23	2135.25	27	39.57813	4.866	1586	21.969	19	3.576	20
NE_43	2135.25	28	39.57813	4.866	1586	21.969	20	3.576	20
NE_23	2179	28.5	39.691	4.977	1586	21.969	26.037	3.576	26.037
NE_23	2275	30.75	39.691	5.099	1586	21.969	26.037	3.576	26.037
NE_24	2480.5	31	40.61018	8.079468	1706	21.969	26.037	5.84	26.037
NE_26	2480.5	33	40.61018	8.079468	2175.6	44.38	35.69	5.84	35.69
NE_14	2501	38	40.61018	8.111	2175.6	44.38	37.552	5.84	37.552
NE_24	2501	38.5	46.442	8.111	2175.6	44.38	37.747	5.84	37.747
NE_06	2678.5	41.5	46.442	8.148739	2175.6	44.38	37.747	6.131	37.747
NE_12	2678.5	41.5	46.86	8.148739	3314.8	47.136	75	6.131	75
NE_16	2678.5	42	47.615	8.148739	3314.8	47.136	75	6.131	75
NE_15	3315.25	46.5	49.549	9.472	3314.8	47.136	75	6.6	75

Appendix B Data II – US Dispersion Data

Stream	*B* (m)	*H* (m)	*U* (m/s)	*u* * (m/ s)	*B/H*	*U/u**	β	α	K_x (m^2/s)
Antietam Creek, Md.	12.8	0.3	0.42	0.057	42.7	7.37	3.8	1.4	17.5
Antietam Creek, Md.	24.1	0.98	0.59	0.098	24.6	6.02	3.2	2.3	101.5
Antietam Creek, Md. *	11.9	0.66	0.43	0.085	18	5.06	2.9	2.3	20.9
Antietam Creek, Md.	21	0.48	0.62	0.069	43.8	8.99	3.8	1.3	25.9
Monocacy River, Md.*	48.7	0.55	0.26	0.052	88.5	5	4.5	1.3	37.8
Monocacy River, Md. *	93	0.71	0.16	0.046	131	3.48	4.9	1.3	41.4
Monocacy River, Md.	51.2	0.65	0.62	0.044	78.8	14.09	4.4	1.3	29.6
Monocacy River, Md.	97.5	.1.15	0.32	0.058	84.8	5.52	4.4	1.6	119.8
Monocacy River, Md.	40.5	0.41	0.23	0.04	98.8	5.75	4.6	1.6	66.5
Conococheague Creek, Md.	42.2	0.69	0.23	0.064	61.2	3.59	4.1	2.3	40.8
Conococheague Creek, Md.	49.7	0.41	0.15	0.081	121	1.85	4.8	2.3	29.3
Conococheague Creek, Md. *	43	1.13	0.63	0.081	38.1	7.78	3.6	1.3	53.3
Chattahoochee River, Ga. *	75.6	1.95	0.74	0.138	38.8	5.36	3.7	1.3	88.9
Chattahoochee River, Ga.	91.9	2.44	0.52	0.094	37.7	5.53	3.6	1.6	166.9
Salt Creek, Neb.	32	0.5	0.24	0.038	64	6.32	4.2	1.4	52.2
Diffcult Run, Va.	14.5	0.31	0.25	0.062	46.8	4.03	3.9	1.1	1.9
Bear Creek*, Colo.	13.7	0.85	1.29	0.553	16.1	2.33	2.8	1.1	2.9
Little Pincy Creek, Md.	15.9	0.22	0.39	0.053	72.3	7.36	4.3	1.1	7.1
Bayou Anacoco, La.	17.5	0.45	0.32	0.024	38.9	13.33	3.7	1.4	5.8
Bayou Anacoco, La.	25.9	0.94	0.34	0.067	27.6	5.07	3.3	1.4	32.5
Bayou Anacoco, La.	36.6	0.91	0.4	0.067	40.2	5.97	3.7	1.4	39.5
Comite River, La.	15.7	0.23	0.36	0.039	68.3	9.23	4.2	1.3	69
Bayou Bartho1omew, La.	33.4	1.4	0.2	0.031	23.9	6.45	3.2	2.5	54.7
Tickfau River, La.	15	0.59	0.27	0.08	25.4	3.38	3.2	1.8	10.3
Tangipahoa River, La.	31.4	0.81	0.48	0.072	38.8	6.67	3.7	1.5	45.1
Tangipahoa River, La. *	29.9	0.4	0.34	0.02	74.8	17	4.3	1.5	44
Red River, La.	253.6	1.62	0.61	0.032	157	19.06	5.1	1.2	143.8
Red River, La.	161.5	3.96	0.29	0.06	40.8	4.83	3.9	1.4	130.5
Red River, La.	152.4	3.66	0.45	0.057	41.6	7.89	3.7	1.4	227.6
Red River, La.	155.1	1.74	0.47	0.036	89.1	13.06	4.5	1.2	177.7
Sabina River, La.	116.4	1.65	0.58	0.054	70.5	10.74	4.3	1.2	131.3
Sabina River, La. *	160.3	2.32	1.06	0.054	69.1	19.63	4.2	1.2	308.9
Sabina River*, Tex.	14.2	0.5	0.13	0.037	28.4	3.51	3.4	2.5	12.8
Sabina River*, Tex.	12.2	0.51	0.23	0.03	23.9	7.67	3.2	2.1	14.7
Sabina River*, Tex.	21.3	0.93	0.36	0.035	22.9	10.29	3.1	1.5	24.2
Mississippi River, La:*	711.2	19.94	0.56	0.041	35.7	13.66	3.6	1.4	237.2

Mississippi River, Mo. *	533.4	4.94	1.05	0.069	108	15.22	4.7	1.4	457.7
Mississippi River, Mo.*	537.4	8.9	1.51	0.097	60.4	15.57	4.1	1.4	374.1
Wind/Big. River, Wyo.	44.2	1.37	0.99	0.142	32.3	6.97	3.5	1.6	184.6
Wind/Big. River, Wyo.	85.3	2.38	1.74	0.153	35.8	11.37	3.6	1.6	464.6
Wind/Big. River, Wyo.*	59.4	1.1	0.88	0.119	54	7.39	4	1.2	41.8
Wind/Big. River, Wyo.	68.6	2.16	1.55	0.168	31.8	9.23	3.5	1.2	162.6
Copper Creep, Va.	16.7	0.49	0.2	0.08	34.1	2.5	3.5	2.5	16.8
Clinch River, Va.	48.5	1.16	0.21	0.069	41.8	3.04	3.7	1.3	14.8
Clinch River, Va.*	28.7	0.61	0.35	0.069	47	5.07	3.9	1.1	10.7
Clinch River, Va.	57.9	2.45	0.75	0.104	23.6	7.21	3.2	1.1	40.5
Clinch River, Va.*	53.2	2.41	0.66	0.107	22.1	6.17	3.1	1.1	36.9
Copper Creek, Va.	18.3	0.38	0.15	0.116	48.2	1.29	3.9	2.5	20.7
Copper Creek, Va.	16.8	0.47	0.24	0.08	35.7	3	3.6	2.5	24.6
Powell River, Tenn. *	36.8	0.87	0.13	0.054	42.3	2.41	3.7	2.2	15.5
Copper River, Va.	19.6	0.84	0.49	0.101	23.3	4.85	3.2	1.3	20.8
Nooksack River, Wash.	64	0.76	0.67	0.268	84.2	2.5	4.4	1.3	34.8
John Day River, Ore.*	25	0.58	1.01	0.14	43.1	7.21	3.8	1.1	13.9
John Day River, Ore.*	34.1	2.47	0.82	0.18	13.8	4.56	2.6	1.9	65
Yadkin River, N.C.	70.1	2.35	0.43	0.101	29.8	4.26	3.4	2.2	111.5
Yadkin River, N.C.	71.6	3.84	0.76	0.128	18.6	5.94	2.9	2.2	260.1
Minnesota River	80	2.74	0.034	0.0024	29.2	14.17	3.4		22.3
Minnesota River	80	2.74	0.14	0.0097	29.2	14.43	3.4		34.9
Amita River	37	0.81	0.29	0.07	45.7	4.14	3.8		23.2
Amita River	42	0.8	0.42	0.069	52.5	6.09	4		30.2
White River*	67	0.59	0.35	0.044	114	7.95	4.7		30.2
Nooksack River	86	2.93	1.2	0.53	29.4	2.26	3.4	1.3	153
Susquehanna River	203	1.35	0.39	0.065	150	6	5	1.1	92.9
Bayou Anacoco	20	0.42	0.29	0.045	47.6	6.44	3.9	1.4	13.9
Muddy River	13	0.81	0.37	0.081	16	4.57	2.8		13.9
Muddy River	20	1.2	0.45	0.099	16.7	4.55	2.8		32.5
Comite River	13	0.26	0.31	0.044	50	7.05	3.9	1.3	7
Comite River	16	0.43	0.37	0.056	37.2	6.61	3.6	1.3	13.9
Missouri River	183	2.33	0.89	0.066	78.5	13.48	4.4	1.4	465
Missouri River	201	3.56	1.28	0.084	56.5	15.24	4	1.4	837
Missouri River*	197	3.11	1.53	0.078	63.3	19.62	4.2	1.4	892

Appendix C Matlab Programme for GNMM

C.1 gnmm_ga

Contents

- Function reference
- Defining variables and checking input arguments
- Loading data file
- Generating training and validation sets
- FastICA toolbox path
- Recording training configuration & progress
- GA's initial run
- GA's successive iterations
- Recording training progress and saving results
- Reporting programme termination
- Fitness function for GNMM's GA process
- ICA weight initialization


```
function [ ] = gnmm_ga( data_file, output_file, varargin )
```

Function reference

```
Using GAs to find variable combinations that produce the
minimum error when training input/target data in a three-layer
MLP.
```

Syntax

```
[ ] = gnmm_ga( data_file, output_file );
[ ] = gnmm_ga( data_file, output_file, [argID, value, ... ])
```

Description

```
[ ] = gnmm_ga( data_file, output_file ) takes two input
arguments, as below.
    data_file (string)
    The file name that contains training & target data. It has
    to be in the '.mat' format and contain the matrix called
    'train_data'. The matrix has to be arranged such that data
```

```
    samples are in rows and and variables in columns; training
    targets to the right of training inputs.
    output_file (string)
    The file name that will be used as GA's output and
    training records. It consists of two files: one is the
    'output_file.mat' contaiing a copy of the latest/final
    Matlab workspace; the other is the 'output_file.txt'
    recording function inputs and the programme
    starting/finishing time. The output_file name is
    preferably contructed as follows: 'ga_*_[123]', where '*'
    stands for the name of the data (e.g. k3b), while [123] is
    the order of the gnmm_ga's implementation.
[ ] = gnmm_ga( data_file, output_file, [argID, value, ... ])
takes several optional input arguments in the format of
[argID, value] pairs. 'argID' will always be the type of
'string', but 'value' may vary as detailed below. In the
absence of these optional arguments, gnmm_ga will use the
default values.
    'ANN_POCH' (string), value (int)
    Number of epochs for each chromosome during the MLP
    training. Default (20).
    'NEURON_H' (string), value (int)
    Number of neurons in the MLP's hidden layer. (8).
    'NEURON_O' (string), value (int)
    Number of neurons in the MLP's output layer. (1).
    'ANN_REPEAT' (string), value (int)
    Number of repeating times in the MLP training when
    evaluating a single chromosome. (10).
    'ICA_USED' (string), value (bool)
    Whether FastICA toolbox will be used. (true).
    'pop_size' (string), value (int)
    Population size. (30).
    'genrs' (string), value (int)
    Generation size. (100).
    'm_alter' (string), value (bool)
    Whether mutation rate will be altered. (true).
    'm_rate' (string), value (double)
    (Initial) mutation rate. (0.05).
    'training_per' (string), value (double)
    Training data percentage. (0.9).
    'valid_per' (string), value (double)
    Validation data percentage. (0.1).
```

Examples

```
[ ] = gnmm_ga( 'class4_train_k3b.mat',...
    'ga_k3b_2', 'ANN_POCH', 20 );
```

Notes

```
In order to use the FastICA toolbox, the toolbox has to
be placed in the parent folder, i.e. the toolbox folder and
the current folder share the same parent folder. The current
function is checked to be compatible with FastICA 2.5.
The current version of the function works with Matlab R2008a
(7.6).
```

See Also

```
gnmm_ga_write, gnmm_ann
```


Defining viriables and checking input arguments

```
error( nargchk( 1, Inf, nargin ));
if mod( nargin, 2 ) ~= 0
    error( 'Inputs got to be pairs' )
end

% global ANN_POCH NEURON_H NEURON_O ANN_REPEAT ICA_USED
ANN_POCH = 20;
NEURON_H = 4;
NEURON_O = 1;
ANN_REPEAT = 10;
ICA_USED = false;
pop_size = 30;
genrs = 100;
m_alter = true;
m_rate = 0.05;
training_per = 0.9;
valid_per = 0.1;

i=1;
while i <= length( varargin )
    if ischar( varargin{ i } )
        switch varargin{ i }
            case 'ANN_POCH'
                i = i + 1;
                ANN_POCH = varargin{ i };
            case 'NEURON_H'
                i = i + 1;
                NEURON_H = varargin{ i };
            case 'NEURON_O'
                i = i + 1;
                NEURON_O = varargin{ i };
            case 'ANN_REPEAT'
                i = i + 1;
                ANN_REPEAT = varargin{ i} ;
            case 'ICA_USED'
                i = i + 1;
                ICA_USED = varargin{ i };
            case 'pop_size'
                i = i + 1;
                pop_size = varargin{ i };
            case 'genrs'
                i = i + 1;
                genrs = varargin{ i };
            case 'm_alter'
                i = i + 1;
                m_alter = varargin{ i };
            case 'm_rate'
                i = i + 1;
                m_rate = varargin{ i };
```

```
            case 'training_per'
                i = i + 1;
                training_per = varargin{ i };
            case 'valid_per'
                i = i + 1;
                valid_per = varargin{ i };
            otherwise
                error( 'Wrong argID.' )
        end
    else
        error( 'Input argument pair has to start with strings' )
    end
    i = i + 1;
end
```

Loading data file

```
data_saved = load( data_file );
raw_data = data_saved.train_data;
raw_data = raw_data';
[ data_std, std_record ] = mapstd( raw_data );
```

Generating training and validation sets

```
[ total_para total_files ] = size( data_std );
files_train = ceil( total_files * training_per );
files_val = floor( total_files * valid_per );
random_position = randperm( total_files );
train_serial = random_position( 1 : files_train );
valid_serial = random_position( files_train + 1 :...
    files_train + files_val );

% global TRAIN_DATA TRAIN_TARGET VAL
TRAIN_DATA = data_std( 1 : total_para - NEURON_O, train_serial );
TRAIN_TARGET = data_std( total_para - NEURON_O + 1 :...
    total_para, train_serial );
VAL.P = data_std( 1 : total_para - NEURON_O, valid_serial );
VAL.T = data_std( total_para - NEURON_O + 1 :...
    total_para, valid_serial );
chr_length = total_para - NEURON_O;
```

FastICA toolbox path

```
if ICA_USED == true
    current_sys = computer;
    switch current_sys
        case { 'SOL2', 'GLNX86' }
            dir_sep = '/';
        case 'PCWIN'
            dir_sep = '\';
        otherwise
            disp( 'Unknown OS.' )
    end
    current_p = pwd;
    ica_path = strcat( current_p( 1 :...
        max( strfind( current_p, dir_sep ))), 'FastICA_25' );
    addpath( ica_path )
```

```
end
```

Recording training config & progress

```
t = cputime;
start_time = datestr( now );
fid1 = fopen( strcat( output_file, '.txt' ), 'w' );
fprintf( fid1, 'Programme starts    @ %10s\n\n', start_time );
fprintf( fid1, 'NO. of epoches for each chromosome  %6.0f\n',...
    ANN_POCH );
fprintf( fid1, 'NO. of hidden neurons               %6.0f\n',...
    NEURON_H );
fprintf( fid1, 'Neurons in the output layer         %6.0f\n',...
    NEURON_O );
fprintf( fid1, 'NO. of trainings each chromosome    %6.0f\n',...
    ANN_REPEAT );
fprintf( fid1, 'Is FastICA used?                    %6.0f\n',...
    ICA_USED );
fprintf( fid1, 'Population size                     %6.0f\n',...
    pop_size );
fprintf( fid1, 'Total generations                   %6.0f\n',...
    genrs );
fprintf( fid1, 'Is mutation rate altered            %6.0f\n',...
    m_alter );
fprintf( fid1, 'Mutation rate                       %6.2f\n',...
    m_rate );
fprintf( fid1, 'Training data percentage            %6.2f\n',...
    training_per );
fprintf( fid1, 'Validation data
percentage          %6.2f\n\n',...
    valid_per );
fprintf( fid1, 'Current generation:\n' );
```

GA's initial run

```
options1 = gaoptimset( 'Generations', 1, 'PopulationType',...
    'bitstring', 'MutationFcn', { @mutationuniform, m_rate },...
    'PopulationSize', pop_size );
[ x fval reason output population scores ] = ...
    ga( @ann_fitness, chr_length, [ ], [ ], [ ], [ ], [ ],...
    [ ], [ ], options1 );

fprintf(fid1,...
    'Generation: %3.0f      Best results (MSE): %6.5f;\n',...
    1, fval );
record_input( 1, : ) = x;
scores_mean = mean( scores );
switch m_alter
    case true
        record_output( 1, : ) = [ fval scores_mean m_rate ];
    case false
        record_output( 1, : ) = [ fval scores_mean ];
    otherwise
        disp( 'Wrong ''m_alter''.' )
end
```

GA's sussessive iterations

```
for n =2:genrs
    options2 = gaoptimset( 'Generations', 1, 'PopulationType',...
        'bitstring', 'MutationFcn', { @mutationuniform, ...
        m_rate }, 'PopulationSize', pop_size, 'InitialPop',...
        double(population));
    [ x fval reason output population scores ] = ...
        ga( @ann_fitness, chr_length, [ ], [ ], [ ], [ ], [ ],...
        [ ], [ ], options2 );

    record_input( n, : ) = x;
    scores_mean = mean( scores );
    switch m_alter
        case true
            record_output( n, : ) = [ fval scores_mean m_rate ];
            m_ratio = scores_mean / record_output( n - 1, 2 );
            if m_ratio <= 0.1
                m_rate = m_rate * 0.1;
            else
                m_rate = m_rate * ( log10( m_ratio ) + 1 );
                if m_rate > 1
                    m_rate = 1;
                end
            end
        case false
            record_output( n, : ) = [ fval scores_mean ];
    end
    fprintf(fid1,...
        'Generation: %3.0f      Best results (MSE): %6.5f;\n',...
        n, fval );
end
```

Recording training progress and saving results

```
finish_time = datestr( now );
cpu_time = cputime - t;
fprintf( fid1, '\n\n\n' );
fprintf( fid1, 'programme finishes  @ %10s\n', finish_time );
fprintf( fid1, 'total CPU time        %6.2f (s)', cpu_time );
fclose( fid1 );

%save( output_file, 'record_input', 'record_output' )
save( output_file )
```

Reporting programme termination

```
if strcmp( eval( 'computer' ), 'SOL2')
    %eval( [ '! echo "' output_file...
    %    '" | mail j.yang.3@warwick.ac.uk' ] )
    eval( [ '! echo "' output_file...
        ' done" | /usr/lib/sendmail esrebt@eagle' ] )
end
```

Fitness function for GNMM's GA process

Nested function. Evaluating each chromosome's fitness acccording to its training error in an MLP.

```
    function scores = ann_fitness( pop )
        % global ANN_POCH NEURON_H NEURON_O ANN_REPEAT
        % global ICA_USED TRAIN_DATA TRAIN_TARGET VAL

        train_row = find( pop );
        scores_accu = 0;
        if isempty( train_row )
            scores = 100;
        else
            for j = 1: ANN_REPEAT
                train_input = TRAIN_DATA( train_row, : );
                val.P = VAL.P( train_row, : );
                val.T = VAL.T;

                net = newff( minmax( train_input ),
[ NEURON_H,...
                    NEURON_O ], { 'tansig', 'purelin' },...
                    'trainlm' );
                net.trainParam.epochs = ANN_POCH;
                net.trainParam.show = NaN;
                net.trainParam.showWindow = false;
                net = init( net );
                if length( train_row ) >= NEURON_H && ICA_USED
                    net = ica_wi(net, train_input, TRAIN_TARGET);
                end

                [ net, tr ]=train( net, train_input, ...
                    TRAIN_TARGET, [ ], [ ], val );
                x = size( tr.perf, 2 );
                scores_accu = scores_accu + tr.perf( x );
            end
            scores = scores_accu / ANN_REPEAT;
        end
```

ICA weight initialization

Nested function. Weight initialization using FastICA 2.5.

```
        function net = ica_wi( net, train_input, TRAIN_TARGET )
            ica_inputs = train_input;
            inputs_mean_rec = mean( ica_inputs, 2 );
            inputs_mean = repmat( inputs_mean_rec,...
                [ 1 size( ica_inputs, 2 )]);
            inputs_mean_moved = ica_inputs - inputs_mean;

            [ inputs_source, A, W ] = ...
                fastica( inputs_mean_moved,...
                'verbose', 'off', 'numOfIC',...
                net.layers{ 1 }.size, 'displayMode', 'off',...
```

```
                'stabilization', 'on' );
            % 50% of maximum direvative for 'tansig'
            fifty_active = log( 3 + 2 * 2 ^.5 ) / 2;
            input_wt_co = fifty_active /...
                max( max( abs( inputs_source )));
            input_wt = input_wt_co * W;
            input_thr = -1 * input_wt_co * W* inputs_mean_rec;

            net.IW{ 1, 1 } = input_wt;
            net.b{ 1, 1 } = input_thr;

            hidden_out = tansig( input_wt_co * inputs_source );
            out_wt = lscov( hidden_out', TRAIN_TARGET' )';
            net.LW{ 2, 1 } = out_wt;
        end
    end
end
```

Published with MATLAB® 7.8

C.2 gnmm_ga_write

Contents

- Function reference
- Checking input arguments
- Loading training records and results
- Writing results onto an excel file
- Converting numbers into excel column characters.


```
function [ ] = gnmm_ga_write( results_file, excel_file, ...
    worksheet_name )
```

Function reference

```
Reading results obtained by 'gnmm_ga' (i.e. sample_file.txt
and sample_file.mat), and writing these results into an Excel
file.
```

Syntax

```
[ ] = gnmm_ga( results_file, [ ], [ ] );
[ ] = gnmm_ga( results_file, excel_file, [ ] );
[ ] = gnmm_ga( results_file, excel_file, worksheet_name );
```

Description

```
 [ ] = gnmm_ga( results_file, [ ], [ ] ) takes a single input
 arguments:
     results_file (string)
     The file comes in two parts. The '.mat' file contains the
     final/latest copy of the GA's evolutionary results, in
     which matrices 'record_input' and 'record_output' together
     contains the winning chromosome and corresponding training
     error, mean error over the whole population, and mutation
     rate ( in case of mutaion rate altering). The '.txt' part
     records 'gnmm_ga.m's funciton inputs and information of
     its implementation.
 [ ] = gnmm_ga( results_file, excel_file, [ ] ) takes an
 additional input arguments, excel_file (string), which holds
 the whole data.
 In gnmm_ga( results_file, excel_file, worksheet_name ), the
 worksheet_name (string) specifies the worksheet that holds the
 perticular gnmm results.
```

Examples

```
 [ ] = gnmm_ga_write( 'ga_k3b_1', [ ], [ ] );
 [ ] = gnmm_ga_write( 'ga_k3b_1', 'results', [ ] );
 [ ] = gnmm_ga_write( 'ga_k3b_1', 'results', '1st' );
```

Notes

```
 In order to use the FastICA toolbox, the toolbox has to
 be placed in the parent folder, i.e. the toolbox folder and
 the current folder share the same parent folder. The current
 function is checked to be compatible with FastICA 2.5.
 The current version of the function works with Matlab R2008a
 (7.6).
```

See Also

```
 gnmm_ga, gnmm_ann
```

Copyright (c) 2006-2008 J. Yang, ISEL, Warwick University, UK

Checking input arguments

```
error( nargchk( 1, 3, nargin ));
if isempty( results_file )
    error('The ''results_file'' input must not be empty')
end

results_txt = [ results_file '.txt' ];
results_mat = [ results_file '.mat' ];

imple_number = strfind( results_file, '_' );
if isempty( excel_file )
    excel_file = results_file( 1: imple_number( end ) - 1 );
    excel_file = [ 'result_' excel_file '.xls' ];
end
```

```
if isempty( worksheet_name )
    imple_case = results_file( imple_number( end ) + 1 : end );
    switch imple_case
        case '1'
            worksheet_name = '1st';
        case '2'
            worksheet_name = '2nd';
        case '3'
            worksheet_name = '3rd';
        otherwise
            worksheet_name = [ imple_case 'th' ];
    end
end

excel_pres = exist( excel_file, 'file' ); %either 0 or 2
if excel_pres
    [ typ, desc ] = xlsfinfo( excel_file );
end
```

Loading training records and results

```
eval(['load ' results_mat ' ''record_input''
''record_output''' ])
record_fields = { 'ANN_POCH',   'NEURON_H',  'NEURON_O',...
    'ANN_REPEAT', 'ICA_USED',  'pop_size',   'genrs',...
    'm_alter',   'training_per','valid_per', 'm_rate',...
    'start_time', 'finish_time',  'cpu_time'};
record_length = length( record_fields );
for i = 1 : record_length
    load ( results_mat, record_fields{ i } );
    if exist( record_fields{ i }, 'var' )
        record_value{ i } = eval( record_fields{ i } );
    else
        % older version of 'gnmm_ga.m'
        fid = fopen( results_txt, 'r' );
        record_texts = textscan( fid, '%s' );
        record_texts = record_texts{ 1 };
        % finish_location = strfind( record_texts, 'finishes' );
        % finish_location =...
        %    find( ~cellfun( @isempty, finish_location ));
        record_value{ i } = [ record_texts{ end - 6 }...
            ' ' record_texts{ end - 5 } ];
        record_value{ i + 1 } = str2double( record_texts{ end-
1 });
        fclose( fid );
        break
    end
end

if m_alter == true
    output_fields = { 'fval', 'scores_mean', 'm_rate' };
else
    output_fields = { 'fval', 'scores_mean' };
end

var_num = num2str( (1 : size( record_input, 2 ) )' );
var_name = repmat( 'Var', size( record_input, 2 ), 1 );
input_fields = cellstr([ var_name var_num ])';
```

```
results_sheet = [input_fields output_fields;...
    num2cell( [ record_input record_output ] ) ];
```

Writing results onto an excel file

```
xlswrite( excel_file, results_sheet, worksheet_name )

field_end = xls_num2col( record_length + 1 );
if ~excel_pres
    xlswrite( excel_file, { 'Sheet' }, 'config', 'A1' )
    title_range = [ 'B1:' field_end '1'];
    xlswrite( excel_file, record_fields, 'config', title_range )
    exist_sheets = '2';
else
    exist_sheets = num2str( length( desc ) + 1 );
end

xlswrite( excel_file, { worksheet_name },...
    'config', [ 'A' exist_sheets])
config_range = [ 'B' exist_sheets ':' field_end exist_sheets ];
xlswrite( excel_file, record_value, 'config', config_range )
function xls_col = xls_num2col( xls_num )
```

Converting numbers into excel column characters.

```
if xls_num < 27
    xls_col = char( xls_num + 64 );
    return;
end

first = floor( xls_num / 26 );
if first < 27
    xls_col = char(first + 64);
else
    error('Too many input variables');
end
second = rem( xls_num, 26 );
xls_col = [ xls_col char( second + 64 )];
```

Published with MATLAB® 7.8

C.3 gnmm_ann_cv

Contents

- Function reference
- Defining variables and checking input arguments

- Loading data file
- FastICA toolbox path
- Recording training configuration & progress
- MLP's iterations
- Recording training progress
- Reporting programme termination
- ICA weight initialization


```
function gnmm_ann_cv( data_file, col_to_use, varargin )
```

Function reference

```
Using ANNs to model input/output relationships between input
variables found by GAs and output targets.
```

Syntax

```
[ ] = gnmm_ann_cv( data_file, col_to_use );
[ ] = gnmm_ann_cv( data_file, col_to_use, output_file );
[ ] = gnmm_ann_cv( data_file, col_to_use, [argID, value, ...])
```

Description

```
[ ] = gnmm_ann_cv( data_file, col_to_use ) takes two input
arguments, as below.
    data_file (string) The file name that contains training &
    target data. It has to be in the '.mat' format and contain
    the matrix called 'train_data'. The matrix has to be
    arranged such that data samples are in rows and and
    variables in columns; training targets to the right of
    training inputs.
    col_to_use (vector) A row vector that contains the
    variable numbers that appear most when being trained by
    GNMM's GA process.
[ ] = gnmm_ann_cv( data_file, col_to_use, output_file ), takes
an optional argument 'output_file' (string), which specifies
the name of the 'mat' file that holds the ANN training
results.
[ ] = gnmm_ann_cv( data_file, col_to_use, [argID, value, ...
]) takes several optional input arguments in the format of
[argID, value] pairs. 'argID' will always be the type of
'string', but 'value' may vary as detailed below. In the
absence of these optional arguments, gnmm_ann_cv will use the
default values.
    'output_file' (string) The name of the 'mat' file that
    holds the ANN training results.
    'ANN_POCH' (string), value (int) Number of epochs for the
    MLP training. Default (30000). Increasing this number will
    increase the memory usage dramatically.
    'NEURON_H' (string), value (int) Number of neurons in the
    MLP's hidden layer. (12).
    'NEURON_O' (string), value (int) Number of neurons in the
    MLP's output layer. (4).
    'ICA_USED' (string), value (bool) Whether FastICA toolbox
```

```
      will be used. (false).
      'iterations' (string), value (int) Number of iterations
      for each MLP's implementation. (2000)
      'l_rate' (string), value (double) The learning rate.
      (0.04).
      'training_per' (string), value (double) Training data
      percentage. (0.9).
      'valid_per' (string), value (double) Validation data
      percentage. (0.1).
      'valid_num' (string), value (double) The number of k-fold
      cross validation. (10).
```

Examples

```
 [ ] = gnmm_ann_cv( 'data_l1b', [ 3 4 8 20 26 28 29 31 32 36 45
48
 52 56 60 ] );
```

Notes

```
 In order to use the FastICA toolbox, the toolbox has to be
 placed in the parent folder, i.e. the toolbox folder and the
 current folder share the same parent folder. The current
 function is checked to be compatible with FastICA 2.5.
 The current version of the function works with Matlab R2008a
 (7.6).
```

See Also

```
 gnmm_ga, gnmm_ga_write, gnmm_rules
```

Copyright (c) 2006-2008 J. Yang, ISEL, Warwick University, UK

Add k-fold cross validation

Defining viriables and checking input arguments

```
error( nargchk( 1, Inf, nargin ));

switch version( '-release' )

    case {'2008b', '2009a'}
        p = inputParser;   % Create an instance of the class.
        p.addRequired( 'data_f', @ischar );
        p.addRequired( 'col_use', ...
            @( x ) sum( x - double( int32( x ))) == 0 );
        p.addOptional( 'output_f', ...
            ['ann' data_file( strfind( data_file, '_' ): ...
            end )], @ischar );
        p.addParamValue( 'ANN_POCH', 3000, ...
            @( x ) x > 0 && mod( x, 1 ) == 0 );
        p.addParamValue( 'NEURON_H', 10, ...
            @( x ) x > 0 && mod( x, 1 ) == 0 );
        p.addParamValue( 'NEURON_O', 1, ...
            @( x ) x > 0 && mod( x, 1 ) == 0 );
```

```
        p.addParamValue( 'ICA_USED', false, @islogical );
        p.addParamValue( 'iterations', 100, ...
            @( x ) x > 0 && mod( x, 1 ) == 0 );
        p.addParamValue( 'training_per', 0.9, @( x ) x<= 1 );
        p.addParamValue( 'valid_per', 0.1, @( x ) x <= 1 );
        p.addParamValue( 'l_rate', 0.04, @( x ) x <= 1 );
        p.addParamValue( 'valid_num', 10, ...
            @( x ) x > 0 && mod( x, 1 ) == 0 );

        % Parse and validate all input arguments.
        p.parse( data_file, col_to_use, varargin{ : } );

    case '2006a'
        if mod( nargin, 2 ) ~= 0
            error( 'Inputs got to be pairs' )
        end

        p.Results.data_f = data_file;
        p.Results.col_use = col_to_use;
        p.Results.output_f = ...
            ['ann' data_file( strfind( data_file, '_' ): end )];
        p.Results.ANN_POCH = 3000;
        p.Results.NEURON_H = 4;
        p.Results.NEURON_O = 1;
        p.Results.ICA_USED = false;
        p.Results.iterations = 1000;
        p.Results.training_per = 0.9;
        p.Results.valid_per = 0.1;
        p.Results.l_rate = 0.04;
        p.Results.valid_num = 10;

        i=1;
        while i <= length( varargin )
            if ischar( varargin{ i } )
                switch varargin{ i }
                    case 'output_file'
                        i = i + 1;
                        p.Results.output_f = varargin{ i };
                    case 'ANN_POCH'
                        i = i + 1;
                        p.Results.ANN_POCH = varargin{ i };
                    case 'NEURON_H'
                        i = i + 1;
                        p.Results.NEURON_H = varargin{ i };
                    case 'NEURON_O'
                        i = i + 1;
                        p.Results.NEURON_O = varargin{ i };
                    case 'ICA_USED'
                        i = i + 1;
                        p.Results.ICA_USED = varargin{ i };
                    case 'iterations'
                        i = i + 1;
                        p.Results.iterations = varargin{ i };
                    case 'training_per'
                        i = i + 1;
                        p.Results.training_per = varargin{ i };
                    case 'valid_per'
                        i = i + 1;
                        p.Results.valid_per = varargin{ i };
```

```
                    case 'l_rate'
                        i = i + 1;
                        p.Results.l_rate = varargin{ i };
                    otherwise
                        error( 'Wrong argID.' )
                end
            else
                error( ['Input argument pair has ' ...
                    'to start with strings'] )
            end
            i = i + 1;
        end

    otherwise
        error( ['Check to see if this version of ' ...
            'Matlab support ''inputParser''.'] )
end
```

Loading data file

```
data_saved = load( p.Results.data_f );
raw_data = data_saved.train_data;
clear data_saved
raw_data = raw_data';
[ data_std, std_record ] = mapstd( raw_data );
clear raw_data
```

FastICA toolbox path

```
if p.Results.ICA_USED == true
    current_sys = computer;
    switch current_sys
        case { 'SOL2', 'GLNX86' }
            dir_sep = '/';
        case 'PCWIN'
            dir_sep = '\';
        otherwise
            disp( 'Unknown OS.' )
    end
    current_p = pwd;
    ica_path = strcat( current_p( 1 :...
        max( strfind( current_p, dir_sep ))), 'FastICA_25' );
    addpath( ica_path )
end
```

Recording training config & progress

```
t = cputime;
start_time = datestr( now );
fid1 = fopen( strcat( p.Results.output_f, '.txt' ), 'w' );
fprintf( fid1, 'GNMM''s ANN training process\n' );
fprintf( fid1, 'Programme starts    @ %10s\n\n', start_time );
fprintf( fid1, 'Total iterations                    %6.0f\n',...
    p.Results.iterations );
fprintf( fid1, 'NO. of epoches in each iteration    %6.0f\n',...
    p.Results.ANN_POCH );
fprintf( fid1, 'NO. of hidden neurons               %6.0f\n',...
```

```
    p.Results.NEURON_H );
fprintf( fid1, 'Neurons in the output layer          %6.0f\n',...
    p.Results.NEURON_O );
fprintf( fid1, 'Is FastICA used?                     %6.0f\n',...
    p.Results.ICA_USED );
fprintf( fid1, 'Training data percentage             %6.2f\n',...
    p.Results.training_per );
fprintf( fid1, 'Validation data percentage           %6.2f\n',...
    p.Results.valid_per );
fprintf( fid1, 'Learning rate                        %6.2f\n',...
    p.Results.l_rate);
fprintf( fid1, 'K-fold cross validation              %6.0f\n',...
    p.Results.valid_num);
fprintf( fid1, 'Variables slected                    %6.0f\n',...
    p.Results.col_use);
fprintf( fid1, 'Current iteration:\n\n' );
```

MLP's iterations

```
[ total_para total_files ] = size( data_std );
files_train = ceil( total_files * p.Results.training_per );
files_val = floor( total_files * p.Results.valid_per );
error_record = [];

for j = 1 : p.Results.iterations
    start_time1 = datestr(now);

    % Generating training and validation sets
    for k = 1 : p.Results.valid_num
        random_position = randperm( total_files );
        train_serial = random_position( 1 : files_train );
        valid_serial = random_position( files_train + 1 :...
            files_train + files_val );
        TRAIN_DATA = data_std( p.Results.col_use, train_serial );
        TRAIN_TARGET = data_std( total_para - ...
            p.Results.NEURON_O + 1 : total_para, train_serial );
        VAL.P = data_std( p.Results.col_use, valid_serial );
        VAL.T = data_std( total_para - p.Results.NEURON_O +
1 :...
            total_para, valid_serial );

        if k == 1
            net = newff( minmax( TRAIN_DATA ),
[ p.Results.NEURON_H, ...
                p.Results.NEURON_O ], {'tansig', 'purelin'},
'trainlm' );
            net.trainParam.epochs = p.Results.ANN_POCH;
            net.trainParam.show = NaN;
            net.trainParam.showWindow = false;
            net.trainParam.lr = p.Results.l_rate;
            net = init( net );
            if p.Results.ICA_USED
                net = ica_wi( net, TRAIN_DATA, TRAIN_TARGET );
            end
        end
        [ net, tr ]=train( net, TRAIN_DATA, TRAIN_TARGET, ...
            [ ], [ ], VAL );
        sim_error = tr.perf( size( tr.perf, 2 ) );
```

```
    end

    finish_time1 = datestr(now);
    if j == 1
        error_record = sim_error;
        fprintf( fid1, ['Initial training MSE
' ...
            '%6.4e\n'], error_record );
        fprintf( fid1, ['Iteration starts
' ...
            '%6s\n'], start_time1 );
        fprintf( fid1, ['Iteration finishes
' ...
            '%6s\n\n'], finish_time1 );
    end
    if sim_error < error_record
        error_record = sim_error;
        fprintf( fid1, ['training MSE  @%6.0fth iteration  ' ...
            '%12.4e\n'], j, error_record);
        fprintf( fid1, ['iteration starts
' ...
            '%6s\n'], start_time );
        fprintf( fid1, ['iteration finishes
' ...
            '%6s\n\n'], finish_time1 );

        save( p.Results.output_f, 'net', 'train_serial', ...
            'valid_serial', 'col_to_use', 'data_std')
    end
end
```

Recording training progress

```
finish_time = datestr( now );
cpu_time = cputime - t;
fprintf( fid1, '\n\n\n' );
fprintf( fid1, 'programme finishes  @ %10s\n', finish_time );
fprintf( fid1, 'total CPU time        %6.2f (s)', cpu_time );
fclose( fid1 );
```

Reporting programme termination

```
if strcmp( eval( 'computer' ), 'SOL2')
    %eval( [ '! echo "' p.Results.output_f...
    %    '" | mail j.yang.3@warwick.ac.uk' ] )
    eval( [ '! echo "' p.Results.output_f...
        ' done" | /usr/lib/sendmail esrebt@eagle' ] )
end
```

ICA weight initialization

Nested function. Weight initialization using FastICA 2.5.

```
    function net = ica_wi( net, TRAIN_DATA, TRAIN_TARGET )
        ica_inputs = TRAIN_DATA;
```

```
        inputs_mean_rec = mean( ica_inputs, 2 );
        inputs_mean = repmat( inputs_mean_rec,...
            [ 1 size( ica_inputs, 2 )]);
        inputs_mean_moved = ica_inputs - inputs_mean;

        [ inputs_source, A, W ] = fastica( inputs_mean_moved,...
            'verbose', 'off', 'numOfIC',...
            net.layers{ 1 }.size, 'displayMode', 'off',...
            'stabilization', 'on' );
        % 50% of maximum direvative for 'tansig'
        fifty_active = log( 3 + 2 * 2 ^.5 ) / 2;
        input_wt_co = fifty_active /...
            max( max( abs( inputs_source )));
        input_wt = input_wt_co * W;
        input_thr = -1 * input_wt_co * W* inputs_mean_rec;

        net.IW{ 1, 1 } = input_wt;
        net.b{ 1, 1 } = input_thr;

        hidden_out = tansig( input_wt_co * inputs_source );
        out_wt = lscov( hidden_out', TRAIN_TARGET' )';
        net.LW{ 2, 1 } = out_wt;
    end
end
```

Published with MATLAB® 7.8

C.4 gnmm_rules

Contents

- Function reference
- Define some constants and load previously saved variables
- Define the data matrix
- Nested function.


```
function [ rules_train, rules_val ] = gnmm_rules( results_file )
```

Function reference

```
 Extract regression rules from trained MLPs.
```

Syntax

```
 [ rules_train, rules_val ] = gnmm_rules( results_file );
```

Description

```
 [ rules_train, rules_val ] = gnmm_rules( results_file ) takes
 a singgle input argument 'results_file', which is the 'mat'
 file name that holds GNMM's ANN training results.
 'rules_train' and 'rules_val' each contains two columns, and
 as many rows as the number of rules fired for the training and
 validation sub data set. The first column is the actual rule
 being fired, the second column is the corresponding number of
 data samples
```

Examples

```
 [ rules_train, rules_val ] = gnmm_rules( 'ann_11b' );
```

Notes

```
 The current version of the function only works with MLPs whose
 output layer only contains a single neuron. See References for
 details.
```

See Also

```
 gnmm_ga, gnmm_ga_write, gnmm_ann
```


Define some constants and load previously saved variables

beta1=1.0020101308531; beta2=-0.251006075157012;

```
kupa=1.99607103795966;
col_to_use = [];
train_serial = [];
valid_serial = [];
data_std = [];
net = [];
eval([ 'load ' results_file ' ''net'' ''train_serial'''...
    ' ''valid_serial'' ''col_to_use'' ''data_std''' ])
% eval(['load ' results_file])
theta = net.b{ 1 }';
condi = [ ];
for i=1:length( theta )
    condi( i, 1 ) = - kupa - theta( i );
    condi( i, 2 ) = - theta( i );
    condi( i, 3 ) = kupa - theta( i );
end
```

Define the data matrix

```
data = data_std( col_to_use, : );
data_tran = ( net.IW{ 1, 1 } * data )';
data_t = data_tran( train_serial, : );
data_v = data_tran( valid_serial, : );
```

```
rules_train = rule_find( data_t, train_serial );
rules_val = rule_find( data_v, valid_serial );
```

Nested function.

Calculate the actual rule numbers and the each rule's execution.

```
    function rule_list = rule_find( data_sub, serial_sub )
        %temp = []; this means double, WRONG!!
        %base2dec('444', 5)=124
        rule_count = zeros( 1, base2dec( num2str( ones( 1, ...
            length( theta )) * 4, '%1.0f' ), 5 ));

        for k = 1 : length( serial_sub )
            data_ind = data_sub( k, : );
            for o = 1 : length( theta )
                if data_ind( o ) >= condi( o, 3 )
                    temp( o ) = '4';
                else if data_ind( o ) < condi( o, 3 ) && ...
                            data_ind( o ) >= condi( o, 2 )
                        temp( o ) = '3';
                    else if data_ind( o ) < condi( o, 2 ) && ...
                                data_ind( o ) >= condi( o, 1 )
                            temp( o ) = '2';
                        else temp( o ) = '1';
                        end
                    end
                end
            end
            inde = base2dec( temp, 5 );
            rule_count( inde ) = rule_count( inde ) + 1;
        end

        rules_real = find( rule_count ~= 0 );
        for p = 1 : length( rules_real )
            rule_list( p, 1 ) = str2double( dec2base( ...
                rules_real( p ), 5 ));
            rule_list( p, 2 ) = rule_count( rules_real( p ));
        end
    end
end
```

Published with MATLAB® 7.8

C.5 gnmm_TestData

Contents

- Function reference
- Defining variables and checking input arguments
- Loading data file and training results
- Compute the rmse/r^2 and display results


```
function gnmm_TestData( data_file, result_mat, varargin )
```

Function reference

```
Apply trained ANNs to training/test data to perform the
pattern recognition task. Also shows the 'coefficient of
determination' for the original and test data.
```

Syntax

```
[ ] = gnmm_TestData( data_file, result_mat );
[ ] = gnmm_TestData( data_file, result_mat, test_data )
```

Description

```
[ ] = gnmm_TestData( data_file, result_mat) takes
two input arguments, as below.
    data_file (string) The file name that contains training &
    target data. It has to be in the '.mat' format and contain
    the matrix called 'train_data'. The matrix has to be
    arranged such that data samples are in rows and and
    variables in columns; training targets to the right of
    training inputs.
    result_mat (string) The workspace saved as in '.mat'
    format from the previous ANN training stage using
    parameters selected by GAs.
[ ] = gnmm_TestData( data_file, result_mat, test_data ), takes
an optional argument 'test_data' (string), which specifies the
name that contains test data. It has to be in the '.mat'
format and contain the matrix called 'test_data'. Format
requirement is the same as in data_file.
```

Examples

```
gnmm_TestData( '2class_new', 'ann_new3', '2class_test');
```

See Also

```
gnmm_ga, gnmm_ga_write, gnmm_rules
```

Defining viriables and checking input arguments

```
error( nargchk( 1, Inf, nargin ));
% data_file = '2class_new';
% result_mat = 'ann_new3';
% test_data = '2class_test';
% varargin = {};

switch version( '-release' )

    case '2008b'
        p = inputParser;   % Create an instance of the class.
        p.addRequired( 'data_f', @ischar );
        p.addRequired( 'result_m', @ischar );
        p.addOptional( 'data_t', 'data_t', @ischar );

        % Parse and validate all input arguments.
        p.parse(data_file, result_mat, varargin{ : } );

    case '2006a'
        p.Results.data_f = data_file;
        p.Results.result_m = result_mat;
        if length( varargin ) == 0
            p.Results.data_t = 'data_t';
        else if length( varargin ) == 1
                p.Results.data_t = varargin{ 1 };
            else
                error( 'Too many inputs' );
            end
        end

        p.Results.data_t = test_data;

    otherwise
        error( ['Check to see if this version of ' ...
            'Matlab support ''inputParser''.'] )
end
```

Loading data file and training results

```
data_saved = load( p.Results.data_f );
raw_data = data_saved.train_data';
[ data_std, std_record ] = mapstd( raw_data );
data_saved = load( p.Results.result_m );
net = data_saved.net;
% train_serial = data_saved.train_serial; valid_serial =
% data_saved.valid_serial;
col_to_use = data_saved.col_to_use;
% data_std = data_saved.data_std;

presence_test = ~strcmp(p.Results.data_t , 'data_t');
if presence_test
    data_saved = load( p.Results.data_t );
    test_data = data_saved.test_data';
```

```
    test_std = mapstd('apply', test_data, std_record);

end
clear data_saved

% map the simulated value back to original range according y =
% (x-xmean)*(ystd/xstd) + ymean;;
target_std = std_record.xstd(end);
target_mean = std_record.xmean(end);
map_back = @(x) (target_mean + x*target_std);
r_square = @(tar, pred) (1  - sum((pred - tar).^2)/sum(tar.^2));
```

Compute the rmse/r^2 and display results

This is done by mapping trained data back to its original range for the training data

```
train_orig = raw_data(end, :);
train_simed = sim(net, data_std(col_to_use, :));
train_simed_orig = map_back(train_simed);
train_rmse = mse(train_simed_orig - train_orig)^.5;
train_r2 = r_square(train_orig, train_simed_orig);
disp('-------------------------------------------');
fprintf('training R-squared          %6.4f\n', train_r2);
fprintf('training RMSE               %6.4f\n', train_rmse);

% for the testing data
if presence_test
    test_orig = test_data(end, :);
    test_simed = sim(net, test_std(col_to_use, :));
    test_simed_orig = map_back(test_simed);
    test_rmse = mse(test_simed_orig - test_orig)^.5;
    test_r2 = r_square(test_orig, test_simed_orig);
    disp('-------------------------------------------');
    fprintf('testing R-squared           %6.4f\n', test_r2);
    fprintf('testing RMSE                %6.4f\n', test_rmse);
end
```

Published with MATLAB® 7.8

Appendix D VBA Programme for GNMM

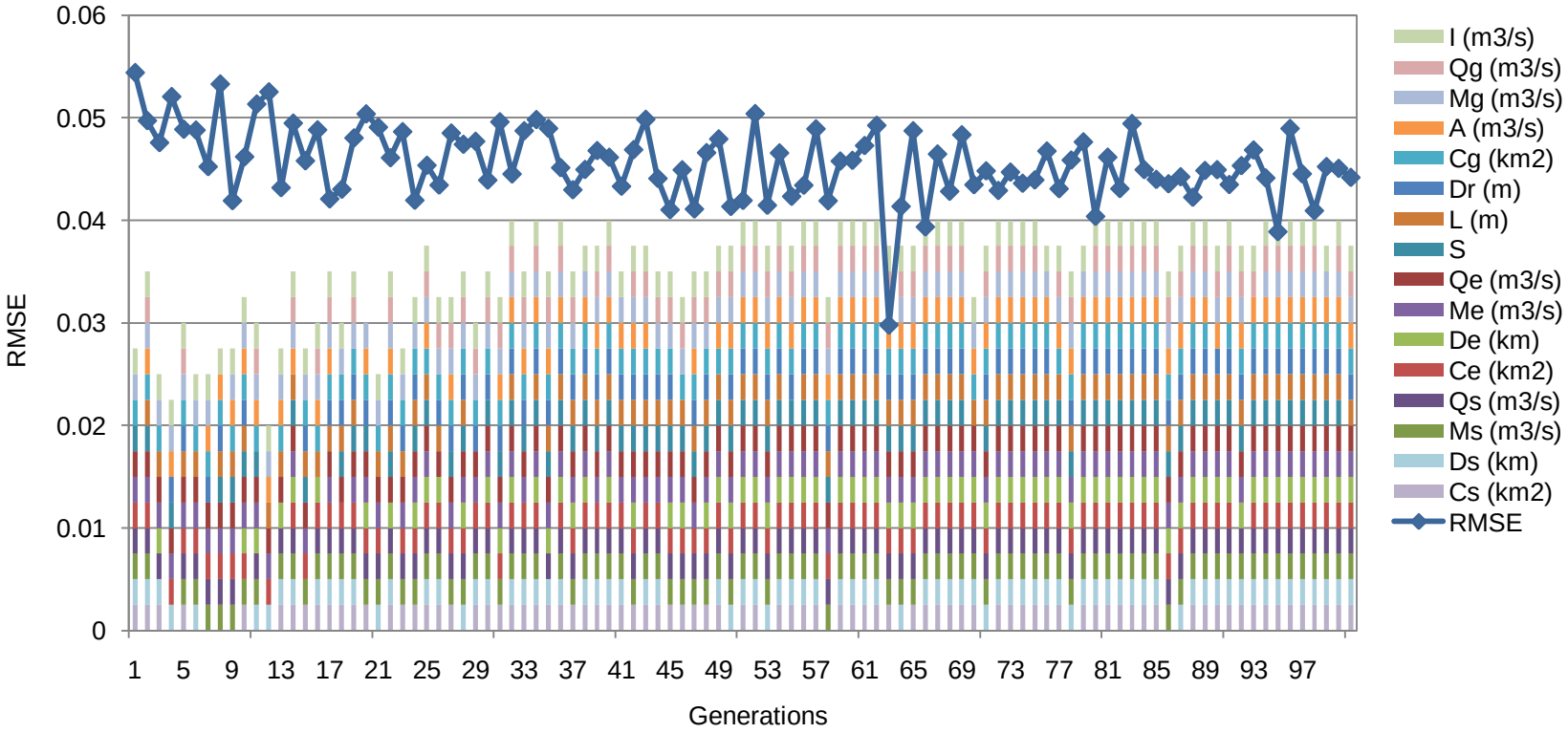
```
Attribute VB_Name = "Module1"
Option Explicit

Sub GNMM_Analyse()
    'Analyse GNMM's GA results file (Excel 2007 format, 'xlsx'),
    'pick up mostly appeared variables
    Dim Sheet_Number As Integer
    Dim Var_Number As Integer
    Dim Count As Integer
    Dim Genes_Number As Integer
    Dim Rows_Done As Integer
    Dim First_Sheet As Boolean
    Dim Summary_Sht As Worksheet
    Dim Input_Area As Range
    Dim DataRange As Range
    Dim Formula_Sum As String
    Dim App_Per As Chart

    Application.ScreenUpdating = False

    Sheet_Number = Worksheets.Count
    'MsgBox "Total number of worksheets in the current file: " & Sheet_Number
    Var_Number = InputBox("Key in the total number of variables", _
    "GNMM user input")
    'Var_Number = 60
    First_Sheet = True

    For Count = 1 To Sheet_Number
       If Worksheets(Count).Name = "summary" Then
          MsgBox "This macro may have already been implemented." _
          & " If not, rename worksheet 'summary'."
          Exit Sub
       End If
    Next Count

    For Count = 1 To Sheet_Number
       If Worksheets(Count).Name <> "config" Then
          If First_Sheet Then
             Set Summary_Sht = _
             Worksheets.Add(After:=Sheets(Sheet_Number))
             Summary_Sht.Name = "summary"
             With Worksheets(Count)
                .Range(.Range("A1"), _
                .Range("A1").Offset(0, Var_Number - 1)).Copy _
                Summary_Sht.Range("B1")
             End With
```

```
            Summary_Sht.Range("A1") = "Sheet"
            First_Sheet = False
        End If

        Worksheets(Count).Activate
        'The actual gene number is the number below - 1
        Genes_Number = Range("A1").CurrentRegion.Rows.Count

        Set Input_Area = _
        Range(Range("A1").Offset(Genes_Number, 0), _
        Range("A1").Offset(Genes_Number, Var_Number - 1))

        Formula_Sum = "= AVERAGE( R[" & _
        1 - Genes_Number & "]C : R[-1]C )"

        'For Each cell In Input_Area
        '   cell.FormulaR1C1 = Formula_Sum
        '   cell.Interior.Color = RGB(128, 60, 90)
        'Next cell
        Input_Area.FormulaR1C1 = Formula_Sum
        Input_Area.Interior.Color = RGB(128, 60, 90)
        Input_Area.Copy

        Summary_Sht.Activate
        Rows_Done = Range("A1").CurrentRegion.Rows.Count
        With Summary_Sht.Range("A1")
            .Offset(Rows_Done, 1).PasteSpecial _
            xlPasteValues, xlPasteSpecialOperationNone
            .Offset(Rows_Done, 0) = Worksheets(Count).Name
        End With
    End If
    'MsgBox "Worksheet " & Worksheets(Count).Name & " has been processed."
Next Count

'If the following contains "= AVERAGE (R[ " &..." (space inside)
'it produces an error
Set DataRange = Summary_Sht.Range("B1", _
Range("B1").Offset(0, Var_Number - 1)).Offset(Sheet_Number, 0)

DataRange.FormulaR1C1 = "= AVERAGE( R[" & _
1 - Sheet_Number & "]C : R[-1]C )"
Range("A1").End(xlDown).Offset(1, 0).Value = "Average"

Set App_Per = Charts.Add
App_Per.Name = "App Per"
App_Per.SetSourceData Source:=DataRange
ActiveChart.ChartType = xlColumnClustered

Summary_Sht.Activate
With Summary_Sht.Range("A1")
    .CurrentRegion.Copy
    .End(xlDown).Offset(2, 0).PasteSpecial Transpose:=True
    .End(xlDown).Offset(2, 0).CurrentRegion.Select
End With
```

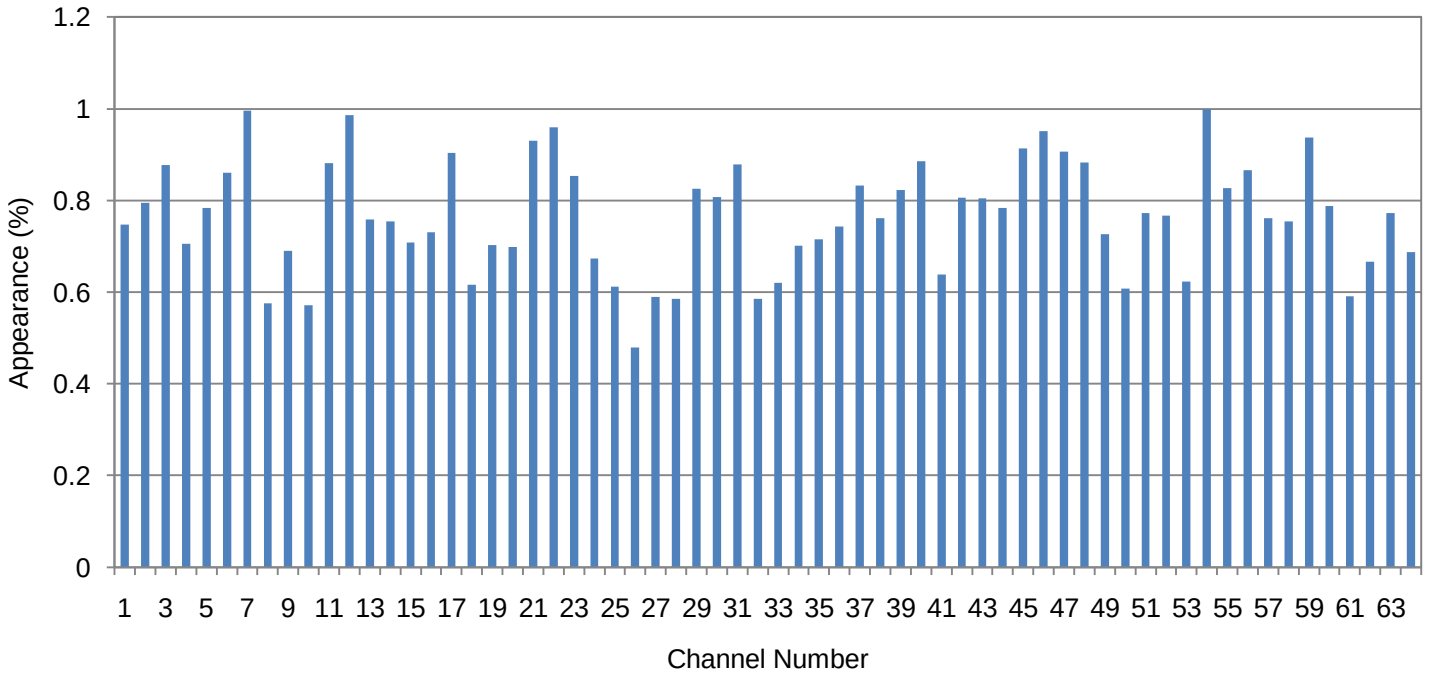
```
    ActiveSheet.ListObjects.Add(xlSrcRange, _
    Selection, , xlYes).Name = "Table1"
End Sub

Sub GNMM_Record()
    'Write the mostly appeared variable number into a cell
    Dim rngData As Range
    Dim rngRow As Range
    Dim Picked_C As Range
    Dim Picked_R As Range
    Dim Ind_Num As String
    Dim Total_Num As String
    Dim Count As Integer
    Dim Row_Ext As Integer

    'Locate datarows
    Set rngData = ActiveSheet.ListObjects("Table1").DataBodyRange

    Total_Num = "[ "
    Count = 0
    'Loop through all data rows
    For Each rngRow In rngData.Rows
        'Only process visible rows
        If rngRow.EntireRow.Hidden = False Then
            'Check calculation
            Ind_Num = rngRow.Cells(1).Value
            Total_Num = Total_Num & " " & Right(Ind_Num, 2)
            Count = Count + 1
        End If
    Next rngRow
    Total_Num = Total_Num & " ]"

    Set Picked_C = Range("A1").End(xlDown).Offset(2, 0). _
    End(xlToRight).Offset(0, 2)
    Set Picked_R = Picked_C.CurrentRegion

    Row_Ext = Picked_R.Rows.Count
    If Row_Ext = 1 Then
        Picked_C = "Criteria"
        Picked_C.Offset(0, 1) = "Count"
        Picked_C.Offset(0, 2) = "Lists"
    End If
    Picked_C.Offset(Row_Ext, 0) = ActiveSheet.ListObjects(1). _
    AutoFilter.Filters(rngData.Columns.Count).Criteria1
    Picked_C.Offset(Row_Ext, 1) = Count
    Picked_C.Offset(Row_Ext, 2) = Total_Num

    MsgBox Total_Num

    With ActiveSheet.ListObjects("Table1").Range
        .AutoFilter Field:=.Columns.Count
    End With

    'If ActiveSheet.ListObjects(1).ShowAutoFilter Then
```

```
    'If ActiveSheet.ListObjects(1).AutoFilter.Filters(4).On Then
    'MsgBox ActiveSheet.ListObjects(1).AutoFilter.Filters(4).Criteria1
    'ActiveSheet.ListObjects("Table1").ShowAutoFilter = True
    'End If
    'End If
End Sub
```

Appendix E RMSE and Winning Variables for Case 7 of Data I



Appendix F Appearance Percentage of Data III



Selected channels are [7 12 17 21 22 45 46 47 54 59].