Novel Computationally Intelligent Machine Learning Algorithms for Data Mining and Knowledge Discovery

Iffat A. Gheyas
Department of Computing Science and Mathematics
University of Stirling,
Stirling FK9 4LA
Scotland, UK

This thesis has been submitted to the University of Stirling in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

November, 2009

Declaration

I hereby declare that this thesis has been composed by me, that the work and results have not been presented for any university degree prior to this and that the ideas that I do not attribute to others are my own.

Iffat Gheyas

2009

Abstract

This thesis addresses three major issues in data mining regarding feature subset selection in large dimensionality domains, plausible reconstruction of incomplete data in cross-sectional applications, and forecasting univariate time series. For the automated selection of an optimal subset of features in real time, we present an improved hybrid algorithm: SAGA. SAGA combines the ability to avoid being trapped in local minima of Simulated Annealing with the very high convergence rate of the crossover operator of Genetic Algorithms, the strong local search ability of greedy algorithms and the high computational efficiency of generalized regression neural networks (GRNN). For imputing missing values and forecasting univariate time series, we propose a homogeneous neural network ensemble. The proposed ensemble consists of a committee of Generalized Regression Neural Networks (GRNNs) trained on different subsets of features generated by SAGA and the predictions of base classifiers are combined by a fusion rule. This approach makes it possible to discover all important interrelations between the values of the target variable and the input features. The proposed ensemble scheme has two innovative features which make it stand out amongst ensemble learning algorithms: (1) the ensemble makeup is optimized automatically by SAGA; and (2) GRNN is used for both base classifiers and the top level combiner classifier. Because of GRNN, the proposed ensemble is a dynamic weighting scheme. This is in contrast to the existing ensemble approaches which belong to the simple voting and static weighting strategy. The basic idea of the dynamic weighting procedure is to give a higher reliability weight to those scenarios that are similar to the new ones. The simulation results demonstrate the validity of the proposed ensemble model.

Acknowledgement

People often say that getting a PhD is stressful. I can't agree there. I really enjoyed every single moment of it. I was in huge financial distress. I didn't think that I would ever see the day! My thesis is a miracle to me. I wish to acknowledge a few people who made this thesis work possible. I have never had the chance to thank them –until now.

I would like to thank my principal supervisor, Professor Leslie Smith, for his expert guidance, invaluable advice, great feedback and patient encouragement. Proud to call him my best friend!!! He pays attention; works hard and always supports my academic needs, whatever they might be. It is for him, in the whole university, only I have the SAS software installed on my computer. I used both the MATLAB and SAS software packages for performing the simulations and analyses; and it gave me a great flexibility to implement and experiment existing machine learning algorithms. In 2008, when it became that bad that I often could not work at times due to financial hardship, he did everything so that I could take my office computer home. I used to work late at night. I was very stressed out (due to financial reason) and we had ups and downs and battles. But he is always very patient and forgiving with me. I was privileged to work with him for the last 4 years.

Sincere thanks to my external examiner Professor Colin Fyfe of the University of the West of Scotland and my internal examiner Dr Bruce Graham for insightful comments on the final revisions of this thesis.

I would like to take this opportunity to thank my second supervisor Dr David Cairns. My tuition fee is over £30,000. I could not pay the last instalment of £2,600. I did not even have enough money for myself. I was asking everyone in the department to give me a part-time job. When heard, Dr Cairns became very sympathetic. This is some of what he wrote to me back in February 2008: "I am very sorry to hear of your difficult circumstances. I have asked Kevin Swingler if he has any work available at the moment and unfortunately neither he nor I are able to offer any work since we do not have any available funds. I think that it is important that you have a discussion with Prof. Smith about your funding concerns to see if it is possible to come to some agreement with the university which avoids or defers the need to pay the outstanding balance. ...I will discuss your situation with him to see what can be done."

I'm sure this is the sweetest email I've ever received. This story has a happy ending!

I wish to gratefully acknowledge the help of the following persons: When I was looking for part time jobs so that I could continue my research, Dr Simon Jones and Dr Savi Maharaj kindly agreed to act as my referees. I really want to thank, my MSc supervisor, Dr Amir Hussain for his initial help in getting me started. Thanks especially to Ms Kate Howie for helping me choose the right statistical tests for my project.

Virtually everybody around me was stressed out. I have been a prolific reader of academic papers in my studies for the last four years. I always send a lot of

interlibrary loan (ILL) requests (I averaged around 50 requests per day) which made the library staffs extremely stressed and tense. I deeply appreciate their help.

My deepest gratitude goes to my family for their unflagging love and support

To My Parents

&

Especially, To My Sister Dr Ferdous Gheyas

For

Their Love, Endless Support and Encouragement

Abbreviations

ACF	Autocorrelation Function
ACO	Ant Colony Optimization
AIC	Akaike Information Criterion
ANN	Artificial Neural Networks
ARIMA	Autoregressive Integrated Moving Average
BIC	Bayesian Information Criterion
BP	Blood Pressure
CI	Confidence Interval
d	Deseasonalized
DFT	Discrete Fourier Transform
DWT	Discrete Wavelet Transform
EM	Expectation Maximization
ERM	Expected Risk Maximization
ERNN	Elman's Recurrent Neural Networks
FW	Filter approach + Wrapper approach (a hybrid algorithm)
GA	Genetic Algorithm
GARCH	Generalized Auto-Regressive Conditional Heteroskedasticity
Gbest	Global best
GE	Generalized Regression Neural Network Ensemble
GEFTS	Generalized Regression Neural Network Ensemble for Forecasting Time Series (proposed time series forecasting algorithm)
GEMI	Generalized Regression Neural Network Ensemble for Multiple Imputation
GESI	Generalized Regression Neural Network Ensemble for Single Imputation
GRNN	Generalized Regression Neural Networks
GRNN MI	Generalized Regression Neural Networks with Multiple Imputation
GRNN SI	Single Imputation with Generalized Regression Neural Networks
HA	ARIMA-GARCH+ERNN (a hybrid algorithm)
HC	Hill Climbing
HD	Hot Deck Imputation
HD MI	Hot Deck Imputation method with Multiple Imputation
HD SI	Hot Deck Imputation method for Single Imputation
HES	Heterogeneous ensemble with simple averaging
HES MI	Heterogeneous ensemble with simple averaging for Multiple Imputation
HES SI	Heterogeneous ensemble with simple averaging for Single Imputation
HEW	Heterogeneous ensemble with weighted averaging
HEW MI	Heterogeneous ensemble with weighted averaging for Multiple Imputation
HEW SI	Heterogeneous ensemble with weighted averaging for Single Imputation
HOS	Homogeneous ensemble with simple averaging

HOS MI	Homogeneous ensemble with simple averaging for Multiple Imputation
HOS SI	Homogeneous ensemble with simple averaging for Single Imputation
HOW	Homogeneous ensemble with weighted averaging
HOW MI	Homogeneous ensemble with weighted averaging for Multiple Imputation
HOW SI	Homogeneous ensemble with weighted averaging for Single Imputation
HUX	Half Uniform Crossover
KNN	*K*-Nearest Neighbours Algorithm
KNN MI	K-Nearest Neighbours Algorithm for Multiple Imputation
KNN SI	K-Nearest Neighbours Algorithm for Single Imputation
logsig	logistic sigmoid function
MAR	Missing at Random
MCAR	Missing Completely at Random
MCMC	Markov Chain Monte Carlo
MI	Multiple Imputation
MLP	Multilayer Perceptrons
MLP MI	Multilayer Perceptrons for Multiple Imputation
MLP SI	Multilayer Perceptrons for Single Imputation
MNAR	Missing Not at Random
MS	Mean Substitution
MSE	Mean square error
nd	non-deseasonalized
NN	Neural Networks
PACF	Partial Autocorrelation Function
Pbest	Personal best
PC	Principal Component
PCA	Principal Component Analysis
PSO	Particle Swarm Optimization
RBFN	Radial Basis Function Networks
RBFN MI	Radial Basis Function Networks for Multiple Imputation
RBFN SI	Radial Basis Function Networks for Single Imputation
RNN	Recurrent Neural Networks
SA	Simulated Annealing
SAGA	SA (Simulated Annealing) +GA (Genetic Algorithm) (proposed feature subset selection algorithm)
SBS	Sequential Backward Selection
SFBS	Sequential Floating Backward Selection
SFFS	Sequential Floating Forward Selection
SFS	Sequential Forward Selection
SI	Single Imputation
SRM	Structural Risk Minimization
SU	Symmetric Uncertainty
SVM	Support Vector Machines
tanh	hyperbolic tangent sigmoid function

VC dimension	Vapnik-Chervonenkis dimension
WKNN	Weighted K-Nearest Neighbours Algorithm
WKNN MI	Weighted K-Nearest Neighbours Algorithm for Multiple Imputation
WKNN SI	Weighted K-Nearest Neighbours Algorithm Single Imputation
ZI	Zero Imputation

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data mining involves the use of sophisticated machine learning algorithms to uncover fundamental patterns in the data. With the unprecedented rate at which data is being collected today in almost all fields of human endeavour, there is an emerging economic and scientific need to extract useful information from it [1]. Data mining is the process of automatic or semi-automatic discovery of patterns from massive databases, and is a highly inter-disciplinary field representing the synthesis of multiple disciplines, including database systems, data warehousing, machine learning, statistics, algorithms, data visualization, and high performance computing. It is an emerging field with the potential for breakthroughs in the understanding of complex physical and biological systems. Data mining has taken many disciplines, from Astronomy to Medicine and Economics, by storm.

Computational Intelligence is a very young discipline [2]. Computational Intelligence includes the study of the design of intelligent agents. An intelligent agent is a system that perceives its environment, learns from experience, and makes appropriate choices given perceptual limitations and finite computation. Finding information hidden in data is as theoretically difficult as it is practically important. The ultimate goal of applying Computational Intelligence to data mining is to discover unknown patterns from data by automatic means without any a priori knowledge of what patterns might look like.

1.1. Research Problems and Motivations

We focus on three major areas of data mining: (1) Feature subset selection in large dimensionality domains, (2) reconstruction of incomplete datasets in cross-sectional studies, and (3) univariate time-series forecasting. In this section, we explain our research motivations, and the research questions that we address in this thesis. In preparation for forthcoming chapters, the reader is offered a clear and thorough discussion of the research problems.

1.1.1 Feature Subset Selection in Large Dimensionality Domains

The purpose of data mining is knowledge discovery: that is to generate new knowledge about events and phenomena from existing data sets whether for classification or for forecasting future events. Data sets usually consist of a number of vectors, each corresponding to some occurrence of an event: each vector consists of a large number of features (or explanatory variables). In general, which features matter for classification (or prediction) is not known. As a result, often all sorts of information about events of interest are gathered. Due to improvements in data acquisition capacity, falling costs of data storage, and development of database and data warehousing technology, more and more high dimensional datasets (with tens or hundreds of thousands of features) are emerging [3]. Many of these features are irrelevant or redundant. Unnecessary features increase the size of the search space and make generalization more difficult. This **curse of dimensionality,** (each feature constitutes a separate dimension to the problem) is a major obstacle in machine learning and data mining. Hence feature selection is an active area of research in pattern recognition [4], machine learning [5], data mining [6] and statistics [7]. In

particular, the prediction performance of a learning algorithm (e.g., a decision tree or a neural network learning algorithm) depends on how efficiently the algorithm learns patterns in the data. Irrelevant and redundant features increase the search space size, making patterns more difficult to detect, resulting in algorithms (whichever are used) finding it more difficult to capture rules that are necessary for forecasting or classification. This also holds true for people interpreting data with redundant or irrelevant features. In addition, the more features there are, the higher is the risk of over-fitting. This is because the probability that some features will coincidentally fit the data increases, unless the sample size grows exponentially with the number of features. Furthermore, in most practical applications, we want to know the collection of core variables that are most critical in explaining an event. For example, to reduce the risk of cancer we have to avoid risk factors. If we fail to select the core risk factors, there may simply be too many risk factors to consider. If people's lifestyles have too many risk factors—especially if they are irrelevant and misleading—people will lose their willingness to change their lifestyle to reduce the risk of cancer. To take another example, in medical diagnosis, a disease is diagnosed by various tests. Different diagnostic tests might have different costs as well as risks associated with them. For instance, an invasive exploratory surgery can be much more expensive and risky than say, a blood test. Feature subset selection can help reduce the number of costly and risky diagnostic procedures. It also helps speed up the diagnosis process by identifying best diagnostic tests.

Feature subset selection entails choosing the subset of the features from the original feature space that maximizes the prediction or classification accuracy. In

principle, the feature subset selection approach is based on the **principle of parsimony (**also known as Occam's razor) [8]. This says that we prefer the model with the smallest possible number of parameters that adequately represents the data. Albert Einstein is quoted in Parzen (1982, p. 68) as remarking that "everything should be made as simple as possible, but not simpler" [9]. However, the principle of parsimony is difficult to apply to feature selection problems. Selecting the best feature subset is proven to be an NP-complete combinatorial problem [10]. There are a number of reasons why this task is so challenging. First, features which do not appear relevant when taken singly may become highly relevant when taken with others. There can be two-way, three-way or complex multi-way interactions among features. As a result a feature that is weakly associated with the particular prediction or classification can improve prediction accuracy if it is complementary to other features. Second, relevant features may be redundant so that the omission of some of them will remove unnecessary complexity and some noise from the forecasting problem. There can be many levels of multi-way redundancy in the feature space. Third, high feature correlation does not imply absence of feature complementarity. Fourth, high levels of multicollinearity increase the probability that a good predictor of the output signal will be found to be non-significant and rejected from the model.

An exhaustive search of all possible subsets of features will guarantee that the best subset of features is found. Unfortunately this is computationally impractical for even a medium sized database since for *n* features, the number of all possible feature subsets is 2^n which is too large to be evaluated even for modest *n.* A major thrust of current research work is focused on the determination of an optimal subset of

features. The choice is a trade-off between the computational time and the quality of the generated feature subset solutions. Finding a highly accurate and fast search algorithm for the selection of optimal feature subsets is a major open problem that we address in chapter 4.

1.1.2 Reconstruction of Incomplete Datasets in Cross-Sectional Studies

The second problem that we addressed in this thesis is reliable reconstruction of the missing data. Missing values occur when values are not recorded for all attributes. Missing values can occur for example due to recording problems, instrument limitations, and unfavourable observing conditions. The study may be overly complex and/or long, or subject may be tired and/or not paying attention and miss the question. These types of missing values are called random missing values. Alternatively, missing values may be non-random in nature. Non-random missing values can happen simply because there are some attributes that are not applicable for some instances (e.g. certain medical question may only be meaningful for female patients or patients over a certain age). Additionally, some subjects may choose not to answer some questions. In these cases, instances with missing values differ systematically from instances with no missing values.

Incomplete data is an unavoidable problem when dealing with real world datasets. The problem is that these missing values result in less efficient estimates because of the sample bias, and the reduced sample size. Further, most data mining algorithms cannot work directly with incomplete datasets. To make the matter worse, many real world problems suffer from high dimensionality. If missing data are

randomly distributed across cases, we could even end up with no valid cases in the dataset, because each of them will have at least one missing data element. Hence, missing value imputation is widely used, by necessity. Imputation refers to the replacement of missing data with statistically plausible values. However, a naïve or unprincipled imputation may create more problems than it solves. A poor imputation strategy may result in distorted created samples that can mislead classification, prediction and clustering techniques. The algorithm used to generate imputed values must be "correct", that is, it must accommodate the necessary predictor variables and their associations. Rubin contends that good imputation methods use all information related to missing cases [11].

Missing data imputation is challenging because possible biases exist since the subjects with missing values are often systematically different from the subjects without missing values [11]. These biases are difficult to eliminate since the precise reasons for missing data are usually not known. Hence, imputations should reflect the full uncertainty about missing values. The determination of uncertainty is not straightforward. If the uncertainty is underestimated, the classifier trained with the imputed dataset will overfit the training data and produce erroneous outputs. To fully account for all sources of variation, it is essential to allow for sampling variation and imputation variation in the imputation. Sampling variation occurs when we sample a population and estimate a parameter from the sample rather than from the population of interest as a whole. If we have taken a different sample from the population, we might have obtained different parameter estimates. Imputation variation is similar. Missing values are replaced by the best surrogate values. However, an imputed value

is just a guess at the actual value; it is not an actual (observed) value. Imputation variation arises from the fact that there is uncertainty regarding the actual value of the missing data. Imputation can also lead to an overestimation of the uncertainty of the estimate. If the uncertainty is overestimated, the classifier trained with the imputed dataset will underfit the training data and exhibit poor prediction capability. Procedures for imputation that incorporate appropriate variability among imputations within a model are called “proper” [11].

A major focus of research is to develop an imputation algorithm that preserves the multivariate joint distribution of input and output variables. Much of the information in these joint distributions can be described in terms of means, variances and covariances. If the joint distributions of the variables are multivariate normal, then the first and second moments completely determine the distributions. On average, the imputation should give reasonable predictions for the missing data, and variability among them should reflect an appropriate degree of uncertainty. An imputation model must preserve all important associations among variables in the dataset, including interactions. The design of a robust imputation algorithm is an important open issue of research for the mid-term or even long future that we pursue in chapter 5.

1.1.3 Univariate Time Series Forecasting

Interest in explaining the evolution of variables is centuries old, originating in astronomy and meteorology. The father of modern astronomy, Johannes Kepler

(1571-1630) is credited for identifying the importance of time series forecasting [12]. The modelling of univariate time series is a subject of great importance in a variety of fields, in astronomy, meteorology, business, economics, and beyond. Any random phenomenon (variable) that can be measured over time is a time series. It is worth emphasizing that to speak of a time series implies that there is some form of randomness. We usually do not refer the fully predictable phenomenon that changes over time (such as the position of a pendulum regularly swinging back and forth) as a time series. A time series is a stochastic process that describes the evolution of a random variable. Time series data follows one subject's changes over the course of time. The units of time will vary with the application; they could be years, quarters, months, days, or even microseconds, depending on the situation to be modelled. The unit of time is not important. What's important is that the observations are equally spaced in time. In time series studies, we are interested in time delay or time lag (or time step), not in actual time. If the observations are not equally spaced, everything gets much more complicated. One approach is to use a smoothing technique to interpolate points at equally spaced intervals, and then use the interpolated values for training instead of original data.

Time series data differs considerably from cross-sectional data. Cross-sectional data (also known as spatial data) refers to data collected by observing many subjects (such as individuals, firms, or countries/regions) at the same point of time, or without regard to differences in time. Analysis of cross-sectional data usually consists of comparing the differences among the subjects. In contrast, a univariate time series is a sequence of observations of the same random variable at different times. Lagged

variables are one of the primary differences between cross-sectional data and time series data. Like cross-sectional data, univariate time series data have predictor and target variables. In time series analysis, lag values of the target are used as predictor variables. In other words, the predictor variables are the values of the given variable at selected lags and the output variables are the forecasts for different horizons. The concept of lagged variable does not apply to spatial data analysis because the observations do not represent a chronological sequence.

The goal of time series (temporal or longitudinal) analysis is to predict future values of a given variable based on its past behaviour. In other words, the goal of time series analysis is to build a model that represents the time series and then use the model to forecast the future values. It is very important that time series forecasts present not only the best guess about the future outcome (i.e. conditional mean), but also a careful assessment of the degree of uncertainty (i.e. conditional volatility) associated with the outcome. For instance, the average temperature (20^{o}C) alone is not indicative of the temperature of a place, if it varies from 0^{o}C to 40^{o}C. In the context of time series, the estimation of both the expected future outcome and the future uncertainty of the outcome are of paramount importance for future planning.

1.1.3.1. Applications of Time Series Analysis

Time series forecasting is a problem encountered in many fields of applications.

Astronomy: Astronomers make use of time series analysis techniques for a variety of purposes, including studies of asteroid rotation rates, active galactic nuclei, and

detection of extrasolar planets. Time series forecasting methods are used to predict many significant sights in the sky.

Business and Economics: Many economic time series are regularly recorded by government agencies, economic organizations, and others. Time series analysis helps directly in business planning. A firm can know the long term trend in the sale of its products. Knowledge of time series analysis is indispensible to Economics. It provides a guide to movements in the stock exchange, share prices, foreign exchange rates and interest rates in the financial system. Today, time series analysis is an indispensable element in monitoring and forecasting economic trends (inflation /deflation/stagflation/reflation) and labour market trends.

Demography: Time series are a useful source of information on the status and trends of demographic changes including infant mortality, life expectancy, population growth, and international migration.

Ecology: Time series are important for understanding the mechanisms that regulate populations of animals and plants. Extinctions and originations are mechanisms used by the ecosystem to maintain the dynamical equilibrium that is favourable to life itself. The goal of time series analysis is to increase our understanding of the ecological dynamics.

Geology: Time series analysis is used to forecast climate change and predict extreme weather conditions. Much data are collected regularly in meteorology, climatology, hydrology, oceanography, seismology, volcanology and environmental research, including the temperature at noon on successive days at recording station, monthly rainfalls, daily precipitation, weekly hours of sunshine, wind patterns, amounts of

pollutants in the environment, trends of river discharges, sea temperature, ozone depletion, global warming, glacial erosion and seismic signals in volcanoes.

Physiology: Time series analysis methods are used to quantify physiological data for classification and identification of different pathological conditions. Variability of electroencephalogram (EEG)—a continuous trace measuring brain activity, electrocardiogram (ECG)—measures the electrical activity of the heart, human gait dynamics and blood pressure over time are some of the physiological signals that are analyzed using time series analysis techniques. Other examples include the weight of a particular person measured daily, and the weight of a newborn baby measured at successive checkups.

Social Work Studies: Time series analysis helps researchers bring clarity to the bewildering complexity of social situations, including suicide rates in young people, the trend of alcohol, tobacco and other drug use, rates of divorce among couples who marry at earlier ages, general trend in teen age pregnancy rates, crime rates, alarming trend in the obesity epidemic among children and adolescents and popularity ratings of politicians.

Others: Many time series arise in areas other than above areas. Examples include: the number of tourists visiting the United Kingdom in successive years, the number of swine flu cases over time, the number of deaths from road accidents in a particular state in successive months and changes in traffic congestion over time.

1.1.3.2. Time Series Forecasting: Challenges

The time series have been proven notoriously resistant to prediction by conventional machine learning algorithms used in spatial data analysis and modelling since time

series violates at least two fundamental assumptions of spatial statistics: assumptions of independence and identity of distribution (i.i.d.). Independence implies that the training cases are statistically independent. The most common violation of the independence assumption occurs when cases are observed in a certain order relating to time. The basic idea of stationarity (identity) is that the probability laws (i.e. the multivariate joint distribution) governing the process do not change with time or space—that is, the process is in statistical equilibrium.

Many techniques have been developed to forecast time series. The key limitation of existing time series forecasting algorithms is that these algorithms are extremely complicated. A simple and robust time series forecasting algorithm is an interesting open question. In chapter 6, we present an improved algorithm for time series forecasting.

1.2. Unique Contributions of this Thesis

Feature selection is a very important step in pattern recognition. We found that among existing algorithms there is no single algorithm that satisfies all the desired properties. There is a group of conventional search algorithms that easily get stuck in local minima but tend to be fast to find a good solution in reasonable time. These algorithms fail to determine the optimal solution due to premature convergence to some local optimum, even though sufficient time is available for the search. There is another group of conventional search algorithms, which are fairly resistant to local minima trapping, but are too slow to find reasonable solutions. We propose a novel

approach SAGA (=SA+GA) for the problem of optimal feature subset search which has both the benefits of elimination of local minima and fast convergence. SAGA is a hybrid algorithm that combines the strengths of the simulated annealing algorithm (SA), the genetic algorithm (GA), the hill-climbing algorithm (HC), and the generalized regression neural network (GRNN), all in one convenient package. The properties and computational aspects of the proposed algorithm are discussed in detail in chapter 4. We published a journal paper from this work. The bibliographic details of the article:

- Gheyas, I. A., & Smith, L.S. (2010). Feature subset selection in large dimensionality domains. *Pattern Recognition,* 43(1), 5-13.

The next two challenges were (1) multiple imputation of missing values, and (2) univariate time series forecasting. While working on these projects, we began to realize that the feature selection algorithm is important but this is only a partial solution to the curse of dimensionality which is the root cause of most of the data mining problems. Feature selection algorithms reduce dimensionality via the selection of the smallest feature subset that maximizes prediction performance. The obvious problem arises when there are not enough training examples to include all the important predictor variables in the model. When learning sample size is much below the effective dimensionality (the size of smallest critical subset) of the problem, feature selection algorithms are required to balance the trade-off between the dimensionality of the resulting feature subset solution and the degrees of freedom lost, leading to a suboptimal feature subset solution. The problem is, we can never know "in advance" how many features are required to model the problem, in order to

obtain a maximum level of predictability. Furthermore, it is often difficult and perhaps even dangerous to make generalizations about optimal sample size for a given dimension. The best sample size is always case specific because it depends on the noise level, population size, complexity and heterogeneity of the target population. To overcome these problems, along with the proposed feature subset selection algorithm SAGA, we suggest the use of a homogeneous ensemble of simple generalized regression neural networks (base GRNN learners), each concentrating on a sub-problem of the problem domain. We recommend the use of this method regardless of the sample size. In the proposed ensemble method, each base GRNN learner is trained to predict the target variable based on a different combination of features selected by SAGA. Thus, each base learner concentrates on learning a distinct subset of prediction rules, instead of all the rules (infeasible for small training sets). Together, base learners can identify most of the good classification rules.

The development of an ensemble of learners entails two issues: the selection of the base learners that constitute the ensemble and the integration of their outputs to construct the ensemble output. In the proposed ensemble method, SAGA is used to find an optimal subset of base learners. The base GRNN learners are then combined using a single combiner GRNN that is used in the final prediction. Examining the variations of outputs of base learners, the combiner GRNN predicts the target values of unseen cases. One of the biggest advantages of our ensemble method as compared to conventional ensemble methods is that it is a local approximation algorithm. Each base GRNN learner (implicitly) assigns a weight to each feature to reflect the relative

importance of that feature in a dynamic fashion with respect to the instance to be predicted. Similarly, the combiner GRNN (implicitly) assigns a reliability weight to each individual base learner's output based on its performance on similar cases. Consequently, our ensemble approach is capable of understanding and predicting heterogeneous patterns. Simulated data and real data show that the method is effective and robust against the curse of dimensionality. In addition, our ensemble approach has relatively few parameters that need to be set by experimentation and analysis, which enhances the appeal of our approach. The proposed ensemble method reduces human intervention to a minimum and thereby reduces human errors, yielding more accurate results. Furthermore, the ensemble technique we develop is completely non-parametric, therefore no distributional assumptions are required. This characteristic enlarges the applicability of our scheme. Following the proposed ensemble approach, we present two customized algorithms in this thesis: one for missing data imputation and one for univariate time series forecasting—the conceptual and practical advantages of these algorithms are discussed in chapters 5 and 6, respectively. Our initial results are extremely encouraging. We produced the following papers based on the materials presented in chapter 5 and chapter 6.

a. <u>Book Chapter (resulted from chapter 5)</u>

Gheyas, I.A, and Smith, L.S. (2009). 'Reconstruction of Cross-sectional Missing Data Using Neural Networks'. In Dominic Palmer-Brown et al. (Eds.), *Engineering Applications of Neural Networks,* Communications in Computer and Information Science (CCIS) , vol. 43, Springer Berlin Heidelberg, pp. 28-34.

b. Refereed Conference Paper (resulted from chapter 5)

Gheyas, I. A., & Smith, L.S. (2009). A novel nonparametric Multiple Imputation Algorithm for Estimating Missing Data. *Paper presented and published in the Proceedings of World Congress on Engineering 2009* Vol. II, WCE 2009, July 1-3, London, UK.

c. Journal Publication (resulted from chapter 5)

Gheyas, I. A., & Smith, L.S., A novel nonparametric repeated imputation algorithm for estimating missing data. *Neural Networks.* (Current status: Revisions being processed).

d. Refereed Conference Paper (resulted from chapter 6)

Gheyas, I. A., & Smith, L.S. (2009). A neural network approach to time series forecasting, *Paper presented and published in Proceedings of the World Congress on Engineering 2009*, vol. II, WCE 2009, July 1-3, London, UK.

1.3. Layout

Feature subset selection, missing data handling and univariate time series forecasting are challenging problems for which many algorithms have been developed. **Chapter 2** reviews the merits and demerits of some of the most popular data mining algorithms in action.

In **Chapter 3**, we discuss the implementation of benchmark algorithms. We also describe the datasets, the statistical tests and the different model selection criteria that were used in this study for comparing algorithms.

Chapter 4 presents our proposed algorithm for selecting an optimal subset of features from a large set of possible features and describes the experimental results and comparisons between the proposed algorithm and the existing algorithms.

Chapter 5 describes the design of the proposed missing data imputation algorithms, explains the comparison methodology employed and provides the experimental results obtained with the proposed and conventional imputation algorithms.

Chapter 6 introduces our proposed algorithm for time series forecasting and gives performance comparison between the proposed and the conventional algorithms.

The thesis ends with **chapter 7**, in which the present study is summarized and conclusions are drawn. Strategies for improving the proposed algorithms are also discussed.

Chapter 2

Review of Previous Work

In this chapter we present the previous work in the areas selected for research: (a) Feature Subset Selection, (b) Handling of cross-sectional missing data, and (c) Time Series Forecasting. Accomplishing all these tasks requires a machine learning algorithm. Hence before we begin our review of these issues, we give an overview of well-known machine learning algorithms.

2.1 Machine Learning Algorithms

The main task of machine learning algorithms is to extract information out of a set of data. Given input variables, machine learning algorithms predict the output variable.

Regression-based techniques are perhaps the most popular modelling methods that use least squares or maximum-likelihood estimation approach to obtain the estimated function [13]. However, in regression-based techniques, users need to specify the form of the function. In reality, it is very difficult to specify an appropriate model. Also, these methods have two major inherent drawbacks: over-fitting and local minima.

Other popular machine learning algorithms include naïve Bayes learning [14], decision trees [15] and *K*-nearest neighbours (KNN) [16]. The naïve Bayes learning algorithm, based on Bayes Theorem, is one of the oldest learning algorithms [17]. This algorithm assumes that explanatory variables are mutually independent.

However, the assumption of independence is often violated in reality which leads to poor generalization performance [18].

A decision tree (and a regression tree) generating algorithm represents the learned function in the form of a decision tree or a set of 'if…then' rules. Although decision trees are easy to understand and interpret, they do have many disadvantages [15]. Generating a decision tree is computationally expensive. Decision tree generating algorithms are unstable. Slight variations in the training data can result in different feature selections at each node within the tree. The effect can be significant since attribute choices affect all descendent sub trees. Also, deduced rules can be very complex. Furthermore, the approach to constructing decision trees usually involves using greedy heuristics (such as entropy reduction) that over-fit the training data and lead to poor accuracy in future predictions.

The *K*-nearest neighbour (KNN) algorithm is a popular choice for many real life applications. In the KNN algorithm, the Euclidean distance is computed between the new feature vector and each feature vector from the training set. *K*-closest neighbours (*K* being the number of neighbours) are then found by analyzing the distance matrix. Euclidean distance metric can be used to compute nearness. If the output variable is a categorical variable, then the KNN algorithm takes a vote among the *K*-nearest neighbours and chooses the class voted for by the majority of the neighbours. If the output variable is a continuous variable, the output value is the average of the *K*-nearest patterns. One disadvantage of this algorithm is that it treats all *K* neighbours in similar way without consideration of the distance differences

between the query instance and its neighbours. To overcome this problem, weighted KNN was proposed by Dudani (1976) [19]. In WKNN, the votes of each of the *K* nearest neighbours are weighted by the respective proximity to the new example. The closer neighbours are weighted far more heavily than those further away. However, the primary disadvantage of WKNN and KNN methods is that the value of parameter *K* must be provided by the user. Selecting an appropriate value for the parameter *K* is not trivial. The performance of these algorithms (WKNN and KNN) degrades when the neighbourhood size is too small or too large [20].

From a very large number of relatively simple processing units, the brain manages to perform extremely complex tasks. Artificial neural networks (ANN) are a programming paradigm that seek to emulate the microstructure of the brain, and are extensively used in artificial intelligence problems [21]. A neural network is a network of many simple processors (units). The advantages of ANN lie in their ability to learn arbitrary function mappings with little or no prior knowledge about the function itself. They are distribution free. Many neural networks have been developed, but the most popular neural networks are Multilayer Perceptrons (MLP), Radial Basis Function Networks (RBFN), and Generalized Regression Neural Networks (GRNN). In GRNN (and traditional regression based techniques), the number of predictor variables determines the number of model parameters and, therefore, the complexity of the model. In MLP and RBFN, the model complexity is governed by the number of hidden nodes and hidden layers in addition to the number of predictor variables.

There is an eternal tension in model building between model complexity (resulting in high accuracy on the training set) and generalization ability to the test and validation sets. Increasing the complexity of the model, in order to increase the accuracy on the training set, eventually and inevitably leads to degradation in the generalization ability to the test and validation sets [22]. In order to relieve the overfitting problem for ANNs (and regression based techniques), Vapnik and his collaborators introduced Support Vector Machines (SVM) [23]. Conventional ANNs (and regression based techniques) are based on Empirical Risk Minimization (ERM) where the best network structure is determined by minimizing the mean square error (MSE) on training data. In contrast, SVM utilizes the Structural Risk Minimization (SRM) principle that minimizes the upper bound of the generalization error to determine the structure of the network. This induction principle is based on the fact that the generalization error is bounded by the sum of the training error and a confidence interval term that depends on the Vapnik-Chervonenkis (VC) dimension. The reader is directed to Vapnik's reference book [22-24] and an introductory reference book Cristianini/Shawe-Taylor [25] for further study. However, Tay and Cao [26] showed that the free parameters of the loss function play an important role in the performance of SVMs and there is little general guidance to determine the parameters of SVM. Improper selection of these parameters is a further root cause of overfitting and underfitting.

During recent years, neural network ensemble is becoming a hot spot in machine learning and data mining. Neural network ensemble is a machine learning paradigm where multiple neural networks (base learners) are trained to solve the

same problem. One of the motivations behind neural network ensembles is the divide-and-conquer strategy, where a complex problem is decomposed into different components each of which is tackled by an individual neural network. A number of recent studies have shown that neural network ensemble forecasting model is an effective approach to the development of a high performance forecasting system [27-28]. Dietterich (1997) gave three motivating reasons for using neural network ensembles [29]. The first reason is that, the training data might not provide sufficient information for choosing a single best neural network. For example, there may be many neural networks performing equally well on the training set. Thus, combining these neural networks may be a better choice. The second reason is that, the search processes of the learning algorithms might be imperfect. For example, even if there exists a unique best model, it might be difficult to achieve since running the algorithms results in sub-optimal models. Thus, ensembles can compensate for such imperfect search processes. The third reason is that, the hypothesis space being searched might not contain the true target function, while ensembles can give some good approximations.

The ensemble network can be either homogeneous or heterogeneous. The heterogeneous ensemble network obtains the overall output from different independent network structures, whereas the homogeneous ensemble network obtains the overall output from similar independent network structures. Methods of homogeneous ensemble model generation basically rely on varying the parameters related to the design and the training of neural network. Typically, an ensemble (homogeneous and heterogeneous ensembles) is constructed in two steps. In the first

step, a large set of candidate base classifiers (neural networks) is generated and then an optimal subset of base classifiers, from the base classifiers pool, is selected to form the ensemble of classifiers. To ensure diversity in the homogeneous ensemble set, the random sub-spacing strategy is used [30]. In the random subspace method, many different features subsets are randomly chosen for producing candidate component classifiers and then base classifiers are iteratively refined using sequential hill-climbing or stochastic techniques. On the other hand, for heterogeneous model generation, neural network ensemble members are created by using different neural network types. After the base classifiers are constructed and refined, an optimal subset of ensemble members are selected from the pool of candidate classifiers. An effective (homogeneous and heterogeneous) ensemble should consist of high-accuracy classifiers that disagree on their predictions. Popular methods for selecting an optimal subset of ensemble members include PCA (Principal Component Analysis) [31], correlation analysis [32], GA (Genetic Algorithm) [33], PSO (Particle Swarm Optimization) [34], and Hill-Climbing (HC) [35]. However, these techniques are not ideal techniques for constructing base classifiers and selecting a subset of ensemble members, since hill-climbing and stochastic algorithms are plagued by local minima and premature convergence problems, respectively and filter methods like PCA and correlation analysis are not robust against complex interactions among the outputs from base classifiers. At the second step of ensemble construction, the predictions of the learned models are integrated. The most popular combination schemes are majority and weighted voting [36]. The majority voting approach involves averaging the predictions of the individual networks. Majority voting rule constitutes a very appealing method due to its conceptual and implementational

simplicity. The major drawback of majority voting is that this approach treats each member equally, i.e., it does not stress ensemble members that can make more contribution to the final generalization. The weighted voting approach is an improvement over the majority voting approach. In the weighted voting approach, the total weight is one and each member of the ensemble is entitled to a portion of this total weight according to their performance. The ensemble decision on a new instance is obtained by a weighted vote of all ensemble members. Weights are adjusted using iterative prediction-error minimization method. The popular weight optimization methods include gradient descent algorithm [37], GA [38] and PSO [39]. A major limitation of conventional weighted voting approach is that it is a static approach. The weights for ensemble members do not depend on the instance to be classified. To overcome this problem, Jacobs and Jordan (1991) proposed a system of experts (base level classifiers) and gating networks where the gating network makes a stochastic decision about which single expert to use on each occasion [40]. Unfortunately, their dynamic selection scheme does not support the fuzzy concept which is fundamental to the ensemble approach.

2.2. Feature Subset Selection

A number of approaches towards optimal feature subset selection have been proposed in the literature. They fall into three broad categories: (1) filter, (2) wrapper, and (3) hybrids of filter and wrapper approaches.

2.2.1 Filter Approach

Filter methods remove unnecessary features without using a learning algorithm. These algorithms select features on the basis of the intrinsic properties of the data. In filter approaches, features are first scored and ranked based on certain statistical criteria. Then, the features with highest ranking values are selected. Frequently used filter methods include t-test [41], Mann-Whitney U-test [42], chi-square test [43], RELIEF algorithm [44], Pearson's correlation coefficient [45], mutual information [46], symmetric uncertainty (SU) [47], and Principal Component Analysis (PCA) [48]. The main advantage of filter methods is that they are extremely fast. However, past studies suggest that filter methods are not very accurate for high-dimensional data [49]. These methods are not robust against interactions among features and feature redundancy. In addition, it is not clear how to determine the cut-off point for rankings in order to select only truly important features and to exclude noise. Furthermore, filter methods are very diverse and motivated by various theoretical arguments. Hence, one method that performs well for some datasets may perform badly on other datasets. Thus it is hard to decide which filter method is best fit for a particular dataset.

2.2.2 Wrapper Approach

In the wrapper approach, feature selection is "wrapped" in the learning algorithm. The learning algorithm is applied to subsets of features and tested on a hold-out set. The wrapper approach uses the prediction accuracy of the learning algorithm to determine how good a given set of features is. Generally, wrapper methods are more accurate than filter methods [50]. However, wrappers are often criticized as a "brute

force" method since they require massive amounts of computation [50]. Wrapper methods are computationally more demanding than filter methods because they evaluate the candidate feature subsets using a learning algorithm, and most of the existing learning paradigms are iterative in the sense that they repeat the same procedure until some termination criterion is fulfilled. To accelerate the wrapper approach in feature subset search, it is vital to employ a fast learning algorithm.

Since exhaustive search is not computationally feasible, wrapper methods must employ a search algorithm to search for an optimal subset of features. Wrapper methods can broadly be classified into two categories based on search strategy: (i) greedy and (ii) randomized/stochastic.

2.2.2.1 Greedy Wrapper Approach

Empirical studies suggest that in practice, the greedy wrapper approach is computationally much more advantageous than the stochastic wrapper approach [51]. Sequential Backward Selection (SBS) (also known as Backward Stepwise Elimination (BSE)) and Sequential Forward Selection (SFS) (also known as Forward Stepwise Selection (FSS)) are two most commonly used wrapper methods that use a greedy hill climbing search strategy [52-53]. SBS starts with the set of all features and progressively eliminates the least promising ones. SBS stops if the performance of the learning algorithm drops below a given threshold by removal of any of the remaining features. SBS relies heavily on the monotonicity assumption [54]. The monotonicity assumption states that the prediction accuracy of a learning algorithm never decreases as the number of features increases. This assumption is dubious

because of the difficulties associated with search space dimensionality and overfitting. In reality, the predictive ability of a learning algorithm decreases as the dimensionality of the feature subspace increases after the maximum point due to a decreasing number of samples for each combination of features. When faced with high-dimensional data, SBS often finds it difficult to identify the separate effect of each of the explanatory variables on the target variable. Because of this, good predictors can be removed early on in the algorithm. In SBS, once a feature is removed, it is removed permanently. By contrast, SFS starts with an empty set of features and iteratively selects one feature at a time—starting with the most promising feature—until no improvement in classification accuracy can be achieved.

SFS is robust to multicollinearity problems but sensitive to feature interaction. On the other hand, SBS is robust to interaction problems but sensitive to multicollinearity problems. Hence, both SBS and SFS can easily be trapped into local minima. The problem with SFS and SBS is their single-track search. Hence, Pudil et al (1994) suggest floating search methods (SFFS, SFBS) that perform greedy search with provision for backtracking [55]. Starting from an empty feature set, the SFFS procedure consists of applying after each forward (feature adding) step a number of backward (feature removing) steps as long as the resulting subsets are better than previously evaluated ones at that level. SFBS works analogously, but with the full feature set. Consequently, there are no backward steps at all if the performance cannot be improved. After each backward step, SFBS performs forward steps as long as the resulting subsets are better than previously evaluated ones at that level. However, recent empirical studies demonstrate that SFFS (Sequential Floating

Forward Selection) is not superior to SFS [56] and SFBS (Sequential Floating Backward Selection) is not feasible for feature sets of more than about 100 features [57]. Greedy algorithms (SBS, SFBS, SFFS, and SFS) sequentially add or remove features. The obvious problem with this approach is that the utility of an individual feature is often not apparent on its own, but only in combinations including just the right other features.

2.2.2.2 Randomized/Stochastic Wrapper Approach

Stochastic algorithms developed for solving large scale combinatorial problems such as Ant Colony Optimization (ACO) [58], Genetic Algorithm (GA) [55], Particle Swarm Optimization (PSO) [59] and Simulated Annealing (SA) [60] are at the forefront of research in feature subset selection. These algorithms can efficiently capture feature redundancy and interaction and do not require the restrictive monotonicity assumption. However, these algorithms are computationally very expensive (though far less expensive than exhaustive search).

2.2.2.3 Hybrids of Filter and Wrapper Approaches

Recently, several authors proposed hybrid approaches mixing both filter and wrapper methods. Examples of hybrid algorithms include Pearson's correlation and a GA [61], $t-$statistics and a GA [62], RELIEF algorithm and a GA [63], principal component analysis (PCA) and an ACO[64], chi-square approach and a multi-objective optimization algorithm [65]; mutual information concept and a genetic algorithm (GA) [66], and Symmetrical Uncertainty (SU) and a GA [67]. The idea behind the hybrid method is that filter methods are first applied to select a feature

pool and then the wrapper method is applied to find the optimal subset of features from the selected feature pool. This makes feature selection faster since the filter method rapidly reduces the effective number of features under consideration. Advocates of hybrid methods argue that the risk of eliminating good predictors by filter methods is minimized if the filter cut-off point for a ranked list of features is low. However, hybrids of filter and wrapper methods may suffer in terms of accuracy because a relevant feature in isolation may appear no more discriminating than an irrelevant one in the presence of feature interactions.

2.3 Handling of Cross-Sectional Missing Data

Little and Rubin [68] and Schafer [69] classify cross-sectional missing data into three categories: (i) Missing Completely at Random (MCAR), Missing at Random (MAR), and Missing Not at Random (MNAR). MCAR occurs when the data which is missing does not depend on the values of any variable in the dataset. Possible reasons for MCAR include manual data entry procedure, incorrect measurements, equipment error, changes in experimental design, accidental skipping of a question or questions etc. MAR occurs when the probability of missing data on a particular variable x_1 (say, smoking status) depends on other observed variables $(x_2, x_3, \cdots, x_n)$ (say, age), but not on x_1 itself. For example, teenagers are less likely to answer a question on smoking habits. MNAR occurs when the probability of missing data on particular variable x_1 depends on the variable x_1 itself. For example, a question regarding smoking habits is less likely to be answered when the respondent is a smoker.

MCAR and MAR data are recoverable, whereas MNAR is not [70]. Hence, MCAR and MAR are also called ‘ignorable missing data mechanism’, whereas

MNAR is called 'Non-ignorable missing data mechanism'. If missing data are MNAR, critical information is lost from the data and, hence there is no universal method for predicting this missing data. Therefore, there is increasing need for estimates of missing data to represent adequate implicit uncertainty about actual values. Various methods are available for handling missing data [Figure 2.1].

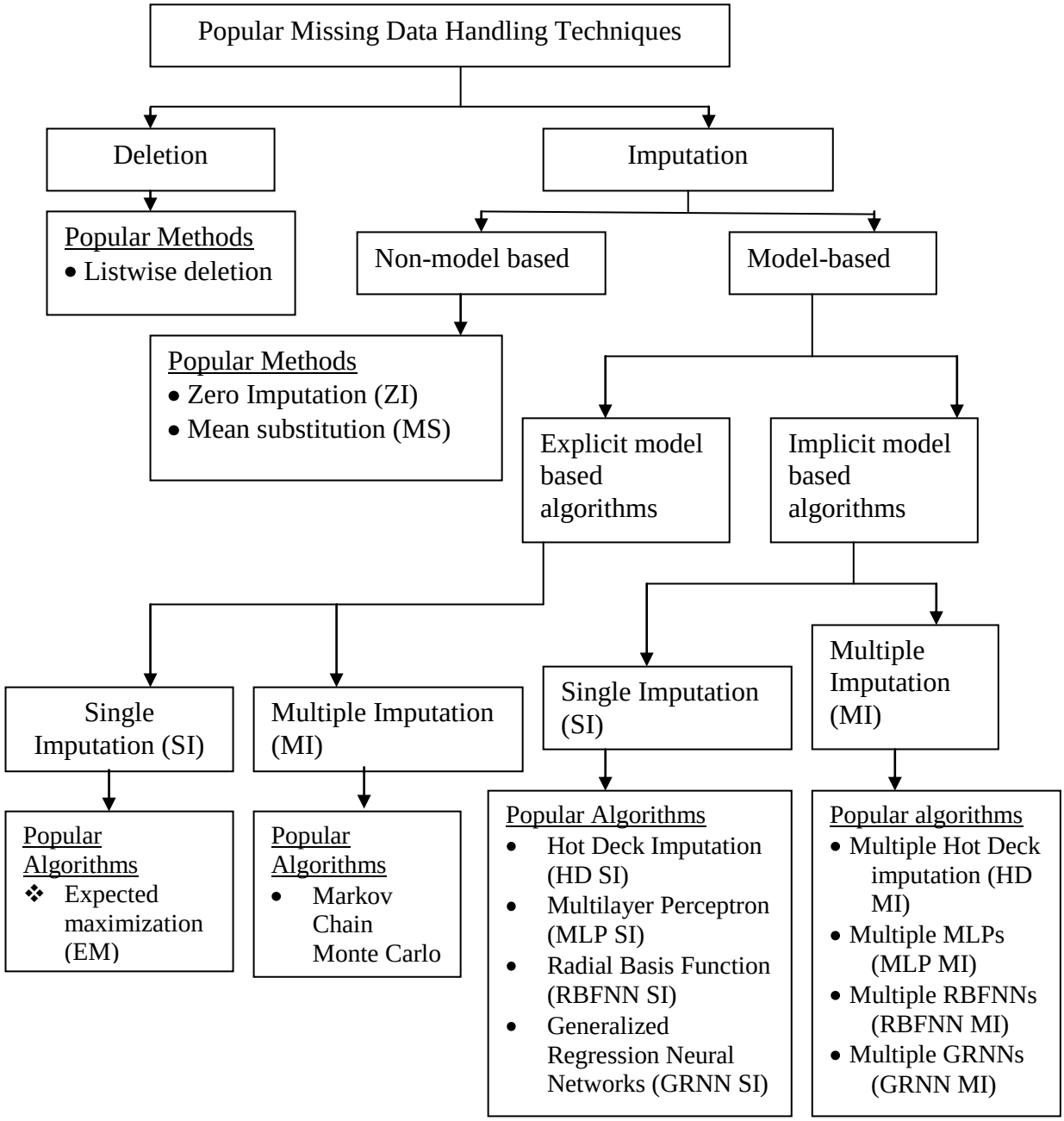

Figure 2.1: Popular methods of handling Missing data

Missing data handling methods can be broadly classified into two categories: deletion and imputation. The most popular method of 'deletion' category is listwise deletion (also known as complete case analysis) [71]. Listwise deletion is a common default method for treating missing data in many widely used commercial statistical software packages such as SAS and SPSS. This method drops an entire case (subject) if a value of any explanatory variable is missing. This is a straightforward process, but cases with missing values should not simply be ignored because it will reduce sample size and may induce systematic selection bias.

Imputation methods replace missing values with values estimated from the available data. The major advantage of imputation is that it uses 'expensive to collect' data, that would otherwise be discarded. Imputation techniques can be split into procedures based on non-model based and model-based approaches. The end products of non-model based approach are surrogate values to replace missing data. In contrast, the end products of model based methods are the estimated values of model parameters, which in turn are used to impute missing values.

The most commonly used **non-model based procedures** are zero imputation [71] and mean substitution [72]. In zero imputation, missing values are always replaced by a zero value. This method is simple and provides all cases with complete data. However, this method does not utilize any information about the data. The integrity and usefulness of the data can be jeopardized as a result. Mean substitution is an improvement over zero imputation. This is a popular imputation method. It

replaces missing values of a given variable with the mean value (unconditional mean) for that variable. Its main advantage is that it produces "internally consistent" sets of results. However, this method does not take into account the relationship among variables and can distort the multivariate relationships. Mean substitution artificially reduces variance. Severe biases can result when the missing data characteristic is MAR but not MCAR.

The most sophisticated techniques for the treatment of missing values are **model-based**. A key advantage of these methods is that they consider interrelations among variables. The model-based techniques can be classified into two categories: explicit model based algorithms and implicit model based algorithms.

Explicit models create a parametric representation of the dataset. These explicit model based methods are based on a number of assumptions. A statistical model provides accurate estimates only when model assumptions are satisfied. If the assumptions are violated, the validity of imputation values derived from applying these techniques may be in question. Commonly used explicit model based methods include Least Squares imputation techniques [73], Expectation Maximization algorithms (EM) [74], and Data Augmentation (DA) algorithm or Markov Chain Monte Carlo (MCMC) [75].

Implicit model based algorithms often have semi-parametric or non-parametric flavours. These methods make few or no distributional assumptions about the underlying phenomenon that produced the data. Deck imputation methods

employing best match (donor) values are the most popular implicit model based algorithm. Deck imputation algorithms are divided into two categories: cold deck imputation and hot deck imputation.

Cold deck imputation methods [76] replace a missing value by a constant value from an external source, for example, from a similar research study found in the literature or from expert opinion or judgement. However this method is rarely used because it cannot be applied when such an external source is unavailable. Besides, these methods are also open to experimenter bias.

Hot deck (HD) imputation is a frequently used imputation method. Hot deck imputation procedures replace missing values on incomplete records using values from similar, but complete records of the same dataset. Past studies suggest that this approach is promising [72]. There are two popular variants of hot-deck imputation algorithm: *K*-nearest neighbour (KNN) imputation algorithm [77] and weighted *K*-nearest neighbour imputation algorithm (WKNN) [78]. A major limitation of these methods is the difficulty in defining what is ‘similar’.

Recently a number of studies applied Multilayer Perceptrons (MLP) [79], Radial Basis Function Networks (RBFNs) [80], and Generalized Regression Neural Networks (GRNNs) [81] to impute missing values. However, these techniques (except for GRNNs) are quite complicated, with many free parameters that can affect the quality of the imputation. Estimating all parameters simultaneously induces errors. In contrast to other neural networks, GRNN has only one free parameter, but

GRNN exhibits a terrible curse of dimensionality problem [82]. The curse of dimensionality is a serious issue, especially when the missing value percentage is high. A few studies have explored ensemble learning for missing data imputation [83]. The common disadvantage of existing ensemble neural networks is the static weighting in classifier fusion.

Both explicit and implicit model based methods are further divided into single and multiple imputation methods. In any single imputation model, one would be offered a single parameter estimate with no sense of how this parameter estimate might vary across equally plausible sets of missing data imputations. Standard single imputation produces a single filled-in dataset, where each missing value is replaced with a single value. The replaced values are then treated as if they were actual values. In general, single imputation offers the advantage of allowing complete data analysis methods to be used, and it requires less work to impute each missing value only once. Expectation maximization (EM SI), multilayer perceptrons (MLP SI), radial basis function networks (RBFN SI) and hot deck imputation algorithms (HD SI) are examples of single imputation (SI) algorithms. A major problem with single imputation is that this approach cannot reflect sampling and imputation variability [84]. Therefore, the estimated variances of the parameters are biased toward zero, leading to statistically invalid inferences. Rubin (1978) proposed multiple imputation (MI) to solve this problem [85]. A detailed summary of MI is given in Rubin [11], Rubin and Schenker [86], Schafer [69], and Schafer and Olsen [87]. In MI, the focus is on getting the joint probability density function of model parameters that represent sampling and imputation variability. In this approach, model parameters are

randomly drawn from the distribution and each missing datum is replaced by $m > 1$ possible values (usually between 2 to 10 possible values, but commonly 5) to accurately reflect uncertainty and to preserve data relationships and aspects of the data distribution. Multiple hot deck imputation algorithms (HD MI), MCMC, and multiple neural networks (NN MI) are examples of MI algorithms. It requires that the analyst specifies an imputation model, imputes several data sets, analyze them separately, and then combines the results. MI builds on the advantage of single imputation. It allows the use of complete-data analysis methods for the data analysis. In addition, it incorporates random error since it requires random variation in the imputation process. MI produces improved estimates of standard errors when compared with single imputation methods because repeated estimations are used. It can accommodate any model and any data and does not require specialized software. MI also increases efficiency of parameter estimates because it minimizes standard errors and simulates proper inferences from the data. The three disadvantages of MI when compared with other imputation methods are: (a) more effort to create the multiple imputations, (b) more time to run the analyses, and (c) more computer storage space for MI-created datasets [11]. These are hardly issues with current development in computer technology. However, several authors have raised questions with regard to the validity of MI approach [88-89]. They report that MI procedure tends to yield longer confidence intervals. In other words, this approach may overestimate standard error or variance (the standard error squared) of missing value estimates.

2.4. Time Series Forecasting

Various modelling techniques are available for time series forecasting. Among them, ARIMA (Autoregressive Integrated Moving Average) modelling methodology is without a doubt the most widely used method for forecasting the future value of time series and GARCH (Generalized Autoregressive Conditional Heteroskedasticity) is a standard method of forecasting volatility (uncertainty) of the future expected value of the output [90]. ARIMA and GARCH are powerful methods. However they have many drawbacks. These regression-based techniques use correlograms for selecting lagged input variables. The major disadvantage of cross-correlation is that it is only able to detect linear dependence between two variables. Since cross-correlation is unable to capture nonlinear dependence between variables, it can lead to omission of important inputs that are nonlinearly related to the output. Like correlation filter, ARIMA model assumes linear dependence between the time series data. As real world data or relationships may be inherently nonlinear, the ARIMA methodology may suffer from significant biases in data mining. Like ARIMA, GARCH is a parametric approach. GARCH performs best when its assumptions are satisfied. Although GARCH is explicitly designed to model time-varying conditional variances, GARCH models often fail to capture highly irregular phenomena [91]. In addition, ARIMA and GARCH are extremely complex techniques, which require a great deal of experience to be used effectively [92]. These techniques have many parameters that need to be adjusted and there is no simple method for the adjustment of parameters. Also, modelling time series with ARIMA and GARCH methodologies requires a combined detrending and deseasonalization. Detrending refers to a process by which a long term trend is removed from time series data. Deseasonlization refers

to removing cyclic variation from the data. In order to detrend and deseasonalize the series, seasonal differencing and non-seasonal differencing are applied to the data. Determining the appropriate order of differencing in seasonal time series processes is particularly difficult. Overdifferencing can lead to inefficient estimates.

Neural networks (NNs) are now the biggest challengers to conventional time series forecasting techniques [93]. An advantage of neural network techniques is their good ability to map non-linear dependencies between input and output data [94]. There is an ongoing controversy regarding whether deseasonalizing data prior model estimation improves the performance of neural network time series forecasting models [95-106]. This is an open question that requires further research.

There are no specific rules for selecting lag variables of ANN for forecasting time series. Feature subset selection approaches explored in the past include PCA [107], mutual information [108], RELIEF algorithm [110], SFS (sequential forward selection) [110], SBS (sequential backward selection) [111], stochastic algorithms [112-114], DFT (Discrete Fourier Transform) [115], and DWT (Discrete Wavelet Transform) [115]. In particular, wavelet analysis is gaining popularity as a method for selecting features from time series due to its local properties. However, these conventional feature selection methods are not powerful enough. One of the most critical issues to be solved in the application of the wavelet analysis (and other filter methods like PCA) is to determine the correct threshold value that allows the removal of noise-dependent high frequencies, while conserving the signal bearing high frequency terms of the signal. However, there is currently no rule for choosing

the right threshold value and this threshold is determined arbitrarily. Moreover, filter methods (PCA, mutual information, relief algorithm, DFT, and DWT) and SFS cannot capture complex interactions among the lagged input variables in a time series. The results of SBS are particularly questionable since SBS is not robust against multicollinearity and the lagged inputs are statistically dependent. Stochastic algorithms suffer from slow and premature convergence.

A variety of NNs are available. However, multilayer perceptrons (MLPs) with backproapagation learning are the most employed NNs in time series studies [116]. Two different architectures of MLPs are applied for time series forecasting purposes: (i) feed-forward [117] and (ii) feedback (or recurrent) [118].

The feedforward MLPs are those type of models in which the outputs can be sent only to the immediate next layers. Strict feedforward architecture does not maintain a short-term memory. Recurrent neural networks (RNN) provide this facility through a number of feedback loops. A generalized RNN can send input in either direction from and to all layers. Thus the output not only depends on the external input it receives but also on the state of the network in the previous time step. The structure is inherently dynamic, which gives the network powerful representation capabilities [119]. Currently, there are three types of recurrent neural networks. These are (in increasing order of complexity and capability):

(1) Tapped delay line models: The network has past inputs explicitly available to determine its response at a given point in time.

(2) Context models or partial recurrent models: These models retain the past output of nodes instead of retaining the past raw inputs.

(3) Fully recurrent models: These models employ full feedback and interconnections between all nodes.

The most popular recurrent neural network underlying connectionist models of time series forecasting is the Elman Recurrent Neural Network (ERNN), which is a partial recurrent neural network. Empirical studies report that the partially recurrent neural networks show better generalization capability compared to complex fully recurrent architectures and tapped delay line models [119]. Fully recurrent neural networks suffer from a large number of weights and convergence difficulties. In fully recurrent neural networks, every unit is connected to every other unit and the disadvantage is that components need to have weights associated with them. The corresponding extra weights might cause overfitting. Similarly, using tapped delay line increases the number of inputs of the network many fold. A consequence of networks with many inputs will be over-fitting.

Previously reported comparative studies between ARIMA-GARCH and MLP (feedforward and feedback) time series models reveal contradictory or inconclusive findings and fail to establish which technique is superior [120-122]. However, the key drawback of MLPs limiting their practical application lies in their complex structures and multiple parameters. In fact, the design of MLPs is even more complicated than the design of ARIMA and GARCH models. The results suggest

that there is plenty of scope for going badly wrong with MLP model [122]. Great care is needed when fitting a MLP model and using it to produce forecasts.

Several studies investigated Radial Basis Function Networks (RBFN) [123] and Generalized regression neural networks (GRNN) [124] in time series forecasting. When creating a RBFN, users must find appropriate numbers of units in the input and hidden layers and the optimal kernel parameters. It is difficult to give general guidelines on how to optimize each parameter in RBFN [125]. GRNN is a relatively simple algorithm with only one free parameter, to forecast time series. These studies show that GRNN always generates physically plausible forecasts [126]. However, GRNN is particularly prone to the curse of dimensionality. When handling univariate time series, the curse of dimensionality is a serious problem.

There are a number of studies on time series forecasting that argue that forecasting performance improves in hybrid models [127]. In such hybrids, whilst the neural network model deals with non-linearity, the ARIMA model deals with the non-stationary linear component. Such models are generally constructed in a sequential manner, with the ARIMA model first applied to the original time series. Then neural networks are applied to its residuals. In this manner, the hybrid model is assumed to exploit the strengths of ARIMA and ANN models in capturing different patterns. However, recent studies indicate that hybrid forecasting techniques do not necessarily outperform individual algorithms [128]. They demonstrate that the hybrid forecasting techniques can underperform significantly compared to its constituents' performances. The authors of these studies emphasize that there may not be any

additive association between the linear and non-linear elements. In addition, one cannot guarantee the residuals of the linear component may comprise valid non-linear patterns.

A few recent studies propose the neural network ensemble for time series forecasting [129-130]. However, constructing ensemble neural networks is obviously a formidable task. The basic goal when designing an ensemble is the same as when establishing a committee of people: each member of the committee should be as competent as possible, but the members should be complementary to one another.

Chapter 3

Methods and Materials

The goal of this chapter is to brief the reader on some background information of our studies. In forthcoming chapters, we will concentrate solely on the unique contributions we made in this field. In this thesis, we address three key challenges in the field of data mining— i) optimal feature subset selection, ii) missing data imputation, and iii) univariate time series forecasting—by presenting novel algorithms for solving these problems. Our research methodology involves two steps. In the first step, we compare the proposed algorithms with several well-known algorithms on synthetic and real world datasets. In the second step, we use statistical tests to determine which algorithm is the most accurate and to know if the differences between two algorithms are found to be statistically significant. This chapter gives an overview of the following topics. Implementation strategies of benchmark algorithms are discussed in section 3.1. Section 3.2 deals with benchmark datasets, while statistical tests used in this thesis are discussed in section 3.3. Section 3.4 is designed to illustrate which model selection criteria (such as SRM, *k*-fold cross validation etc.) were applied for determining the architecture of a model for different applications.

3.1 Implementation Strategies of Benchmark Algorithms

Figure 3.1 displays the conventional algorithms that we used in this study—this section describes the implementation of all these algorithms.

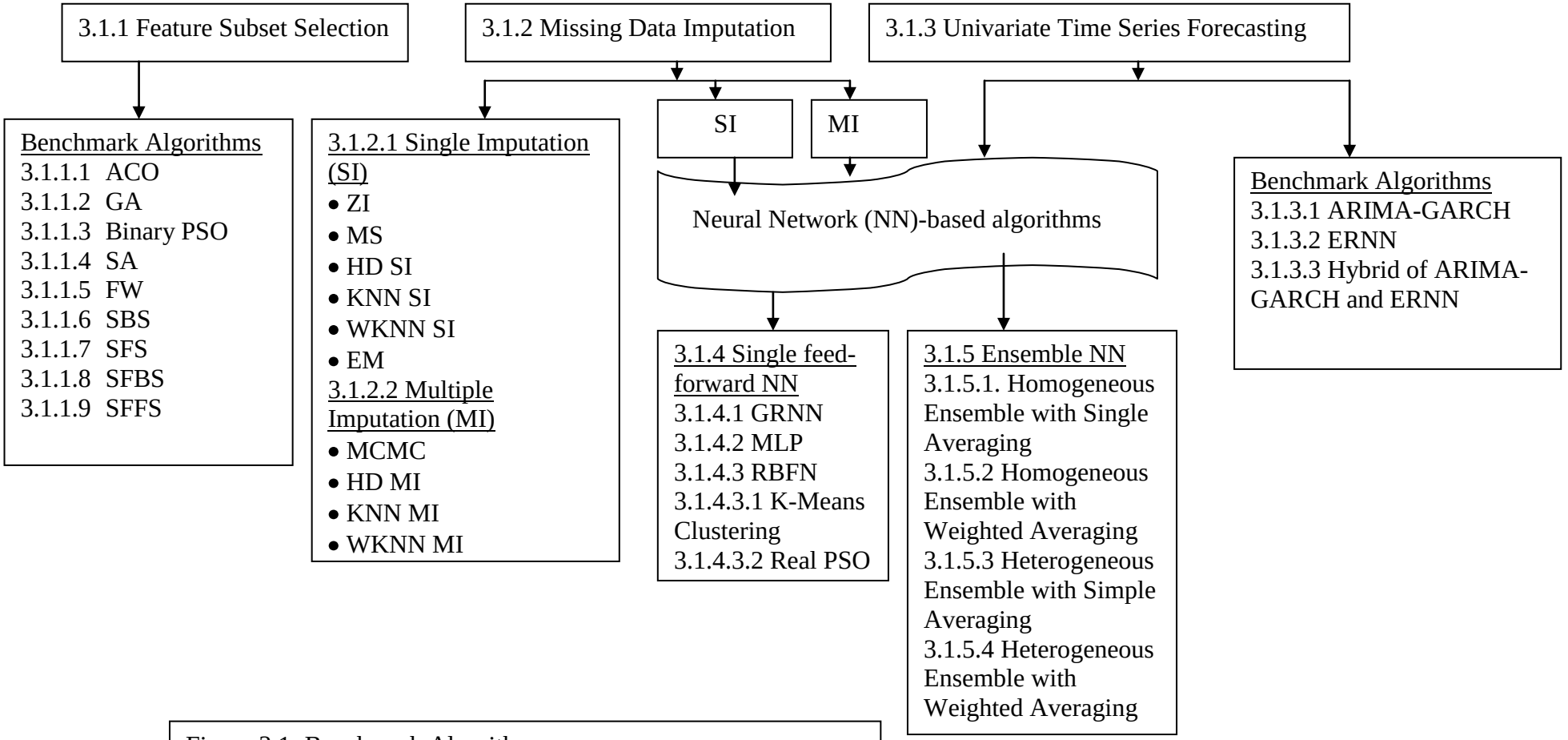

Figure 3.1: Benchmark Algorithms

3.1.1. Implementations of Benchmark Feature Selection Algorithms

We compare our proposed feature subset selection algorithm with four popular stochastic search algorithms (Ant colony Optimization, Genetic Algorithm, Particle Swarm Optimization, and Simulated Annealing), a hybrid algorithm of filter and wrapper methods (FW), and four greedy search algorithms (Sequential Backward Selection, Sequential Forward Selection, Sequential Floating Backward Selection, Sequential Floating Forward Selection). For each search algorithm, we need a machine learning algorithm that evaluates candidate feature subset solutions. We used GRNN (Generalized Regression Neural Networks) to evaluate candidate solutions. A fitness score is assigned to each feature subset solution based on the accuracy (as a fraction, not a percentage) of the GRNN classifier trained by that feature subset solution.

The fitness score is always between 0 and 1, with 1 the best score possible. The GRNN is described in section 3.1.4.1.

Stochastic Search Algorithms

Population size and stopping criteria are common free parameters for stochastic algorithms. There are no hard and fast rules concerning the choice of these parameters. Past studies demonstrate that an initial population size of 100 or more has to be chosen in order to get satisfactory results [131]. Practically, a population size of around 100 is quite common. The size of the initial population affects the algorithm's convergence performance. Small population size may result in premature convergence, while large size will increase computational efforts and may make convergence slow. For this study, we arbitrarily choose the initial population size as 100.

We used the CPU time as the stopping criterion for fairer comparison. The time required to evaluate a candidate feature subset solution depends on the dimension of feature space and the sizes of the training and test sets. With a stopwatch, we observe and record the time required to evaluate solutions of every possible dimension. After evaluating each solution, the computer retrieves the time expected to be used for the evaluation of a feature subset with the same dimension size and subtracts it from the total time available for use.

3.1.1.1 Ant Colony Optimization Algorithm (ACO)

ACO is inspired by the foraging behaviour of real ant colonies. The worker ants of almost all ant species forage away from the colony for food. Once ants have found a food source, they leave behind a scent (pheromone) for other ants to follow. Only ants that are carrying food leave pheromone trails. In a short time, there is strong chemical trail from marching columns of ants transporting food from the food source to the nest. When the source is exhausted, no new trails are marked by returning ants and the scent slowly dissipates. ACO was first proposed in 1992 by Marco Dorigo in order to solve a difficult combinatorial optimization problems [132]. Figure 3.2 presents the block diagram of the ACO.

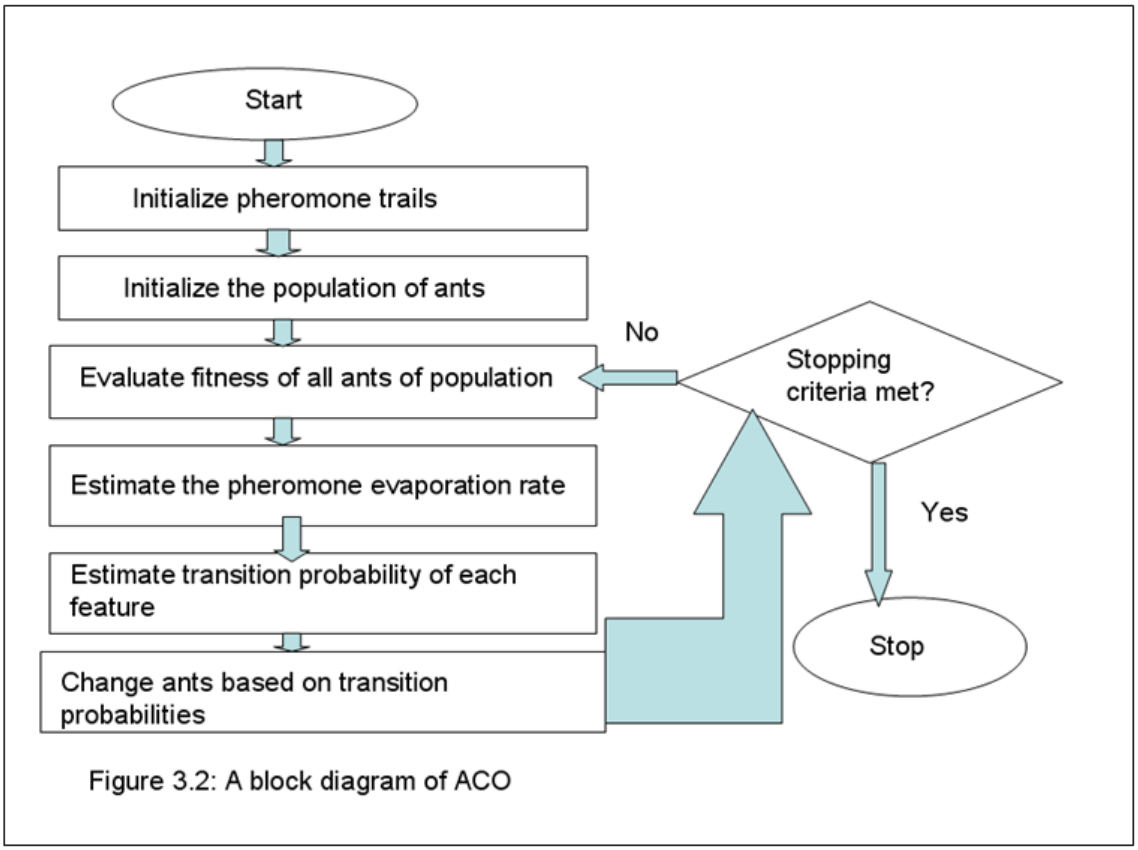

Figure 3.2: A block diagram of ACO

Pseudo Code of ACO

Step 1: Each particular feature represents a particular region. Initialize the pheromone level of each region (feature) to 1.

Example: Suppose we have a feature space given by the map $\bar{A,B,C,D}$ with a dimensionality of 4. The initial pheromone level of each feature (region) is 1 (Example Table 3.1.1.1.1).

Example Table 3.1.1.1.1: Initial Pheromone Distribution

Region	Pheromone level
A	1
B	1
C	1
D	1

Step 2: Set the maximum search time ($T_{\max}$) in seconds that the algorithm can take.

Example: Let, $T_{\max} = 100$

Let us also assume that we have the following information about the given dataset in the memory cell:

Example Table 3.1.1.1.2: Information stored in the memory

Dimensionality of the solution	*Time required to evaluate the solution (in seconds)*
1	2
2	4
3	7
4	12

Step 3: The initial population consists of 100 ants. An ant (candidate feature subset solution) is a binary string with the length of *N*, where *N* is the number of features. Bit value '1' stands for the presence of a feature and '0' stands for its absence so that each ant represents a feature subset solution. Initially, each ant is given randomly chosen subset of features.

Example: Let the initial population size of ants be 3. The subsets selected by the ants:

Ant 1: [1,0,0,1]; Ant 2: [1,1,1,1]; and Ant 3: [0,0,0,1]

Step 4: Evaluate the fitness of each ant using GRNN and update the best-to-date solution.

Example Table 3.1.1.1.3: Fitness values of Ants:

Ant	Solution	Accuracy	Fitness
1	[1,0,0,1]	80%	0.80
2	[1,1,1,1]	59%	0.59
3	[0,0,0,1]	40%	0.40

Step 5: Retrieve the time spent in fitness evaluation (please see example table 3.1.1.1.4) and then estimate the pheromone evaporation rate (ρ) using the equation (3.1).

$$\rho = \frac{T_{spent}}{T_{\max}} \qquad (3.1)$$

Where, T_{spent} = Total time spent so far.

Example Table 3.1.1.1.4: Retrieved information about the time spent in fitness evaluation:

Ant	Solution	Dimensionality	Evaluation time
Ant 1	[1,0,0,1]	2	4
Ant 2	[1,1,1,1]	4	12
Ant 3	[0,0,0, 1]	1	2
Overall time Spent			4+12+2= 18

Therefore, $\rho = \frac{18}{100} = 0.18$

Step 6: Update pheromone level in each region as in equation (3.2) (example table 3.1.1.1.5 provides a numerical example):

$\tau_i(t+1) = \tau_i(t) - \rho \times \tau_i(t) + \Delta\tau_i(t)$ (3.2) where, $\Delta\tau_i(t) = \sum(\Delta\tau_{ik}(t))k$

$\tau_i(t+1)$ and $\tau_i(t)$ represent the total amount of pheromone existing at feature 'i' at the beginning of the periods *(t+1)* and *t* respectively. ρ is pheromone evaporation rate. $\sum \Delta\tau_{ik}(t)$ is the sum of fitness scores of the ants that have selected the feature 'i' at time *t* and *k* represents the number of ants that have selected the feature. $\Delta\tau_i(t)$ indicates the increase in the total pheromone at region i at the period t.

Example Table 3.1.1.1.5: The updated pheromone level at each feature (after 1 iteration):

Feature	$\tau_i(t)$	$\rho \times \tau_i(t)$ (*p=0.18)*	k	$\Delta\tau_i(t) = \sum(\Delta\tau_{ik}(t))/k$	$\tau_i(t+1) = \tau_i(t) - \rho \times \tau_i(t) + \Delta\tau_i(t)$
A	1	1×0.18=0.18	2	(0.8+0.59)/2 =0.70	[1-0.18]+0.70=1.52
B	1	1×0.18=0.18	1	0.59/1 =0.59	[1-0.18]+0.59=1.41
C	1	1×0.18=0.18	1	0.59/1=0.59	[1-0.18]+0.59=1.41
D	1	1×0.18=0.18	3	(0.8+0.59+0.4)/3=0.60	[1-0.18]+0.60=1.42

Step 7: Estimate transition probability of each region or feature at $(t+1)$ as in equation (3.4): $P_i(t+1) = \tau_i(t+1) / \sum_{j=1}^{g} \tau_j(t+1)$ (3.3)

Where, $P_i(t+1)$ is the transition probability of feature 'i' at time $(t+1)$. $\tau_i(t+1)$ is the total pheromone at feature 'i' at $(t+1)$. g is the number of features in the feature space. $\sum_{j=1}^{g} \tau_j(t+1)$ stands for the amount of pheromone present in all features at $(t+1)$.

Example table 3.1.1.1.6 provides a numerical example of transition probability estimation.

Example Table 3.1.1.1.6: Transition probability of each feature at iteration $(t+1)$:

Feature	$\tau_i(t+1)$	$P_i(t+1) = \tau_i(t+1) / \sum_{j=1}^{g} \tau_j(t+1)$
A	1.52	1.52/5.76= 0.27=0.3
B	1.41	1.41/5.76=0.24
C	1.41	1.41/5.76=0.24
D	1.42	1.42/5.76=0.25
$\sum_{j=1}^{g} \tau_j(t+1)$	1.52+1.41+1.41+1.42=5.76	

Step 8: Change the feature subset solutions (ants) as follows:

- Generate a random number between 0 and 1 for each element of the feature vector.
- If the random number is smaller than or equal to the transition probability of the feature, set the value of the bit to 1. Otherwise set the value to 0.

Step 9: Check if the stopping criterion is satisfied; if not, go back to step 4.

3.1.1.2 Genetic Algorithm (GA)

Genetic Algorithm (GA) was first proposed by Holland in 1975 (Holland, 1975). GA is inspired by Darwin's theory of evolution [133]. Figure 3.3 presents the block diagram of the GA. GA was implemented as follows:

Step 1: Encode each candidate subset solutions into chromosome-like strings, in order that the genetic operators can be applied to them. Each gene in a chromosome is a bit, which takes a value of either 1 or 0. Each gene location in a chromosome corresponds to a particular feature. 1 stands for presence of a feature and 0 for absence.

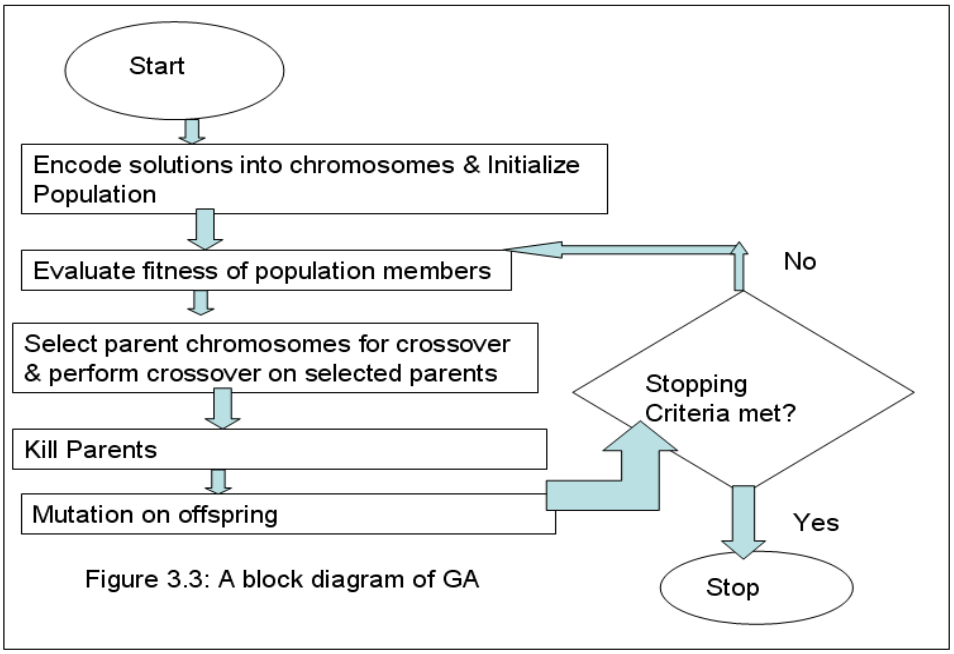

Figure 3.3: A block diagram of GA

Step 2: Randomly select a pool of chromosomes from the entire population to form a new initial population. The size of the population is one of the most important choices faced by any user of genetic algorithms. There are two significant problems associated with a lack of robustness in a GA. When the population size is too small or the genetic diversity is too small, a GA can converge too quickly on a local optimum. On the other hand, if the population size is too large or the genetic diversity is too large an optimal solution may not emerge in a reasonable time. In this study, the initial population size was set to 100 chromosomes.

Step 3: Evaluate the fitness value (prediction accuracy) of each chromosome in the pool and update the time available for searching.

Step 4: According to the fitness value, probabilistically pick 50 pairs of couples from the available pool. There are two competing factors that need to be balanced in the selection process: *selective pressure* and *genetic diversity*. Selective pressure, the tendency to select only the best members of the current generation to propagate to the next, is required to direct the GA to an optimum. Genetic diversity, the maintenance of a diverse solution population, is also required to ensure that the solution space is adequately searched. Too much selective pressure can lower the genetic diversity and the GA converges prematurely. Yet, with too little selective pressure, the GA might not converge to an optimum in a reasonable time. A proper balance between the selective pressure and genetic diversity must be maintained for the GA to converge in a reasonable time to a global optimum.

Various types of selection strategies have been proposed. The most popular selection strategies are elitist selection, roulette-wheel selection, tournament selection, and ranking selection [134]. In elitist selection, only the best chromosomes are allowed to survive to the

next cycle of reproduction. But selecting only the best chromosomes has one major disadvantage; all chromosomes in a population will start to look the same very quickly. In roulette wheel selection, each chromosome in the population is assigned a roulette wheel slot (i.e. a probability of being selected) sized in proportion to its fitness. In effect, the probability of a chromosome being selected is given by the ratio of its fitness value (its roulette wheel slot size) over the total sum of fitness values of all population chromosomes (the entire size of the roulette wheel). The main problem of this approach is that if one chromosome has a fitness value much better than others, the chromosome will tend to be selected much more often than the others. If the super-performer chromosome represents a local optimum (rather than a global optimum) in the solution space, there will be a premature convergence for that suboptimal solution. In tournament selection, a number of chromosomes are chosen randomly from the population and the best chromosome from this group is selected as parent. This process is repeated until the required number of chromosomes is obtained. One potential problem with tournament selection is that it does not guarantee that the best solution in the current generation is passed on to the next. Ranking selection consists of two steps. First all chromosomes are ranked according to their fitness values. Then a selection procedure conceptually similar to roulette-wheel selection is applied based on the rankings, rather than on the original fitness values. The better the ranking of a chromosome, the higher its probability of being selected. We emphasize that in this method chromosomes are selected based only on their performance relative to other chromosomes. Information about the magnitude of fitness differences between chromosomes is intentionally discarded. In this study, we used rank selection strategy for picking a chromosome for crossover. We performed the task in the following manner:

Calculate cumulative selection probabilities in the following sequence. First, arrange chromosomes in descending order of their fitness scores (i.e. prediction accuracies). Second, rank the chromosomes on the basis of their fitness. Assign rank 1 to the lowest fitness score, 2 to the next and so on. Third, estimate the probability of selection for each chromosome using the equation (3.4) and then calculate the cumulative selection probability of each chromosome.

$$p_i = \frac{c_i}{n} \qquad (3.4)$$

Where p_i =selection probability of the *i*-th chromosome.

c_i = rank point of the *i-th* chromosome,

n = sum of the rank points assigned to the chromosomes in the pool

Generate a random number 'r' between 0 and 1 and pick the chromosome whose cumulative selection probability is equal to or just smaller than 'r'. Repeat this step until the desired number of chromosomes is selected.

Numerical example: The following is an example of cumulative selection probability estimation based on ranking selection strategy.

Example Table 3.1.1.2.1: cumulative selection probability

Chromosome	Fitness	Rank	Selection Probability	Cumulative Selection Probability
Chromosome 4	0.91	4	0.4	0.4
Chromosome 1	0.84	3	0.3	0.7
Chromosome 3	0.58	2	0.2	0.9
Chromosome 2	0.35	1	0.1	1.0
Total		10	1.0	

Example Table 3.1.1.2.2: Spin the Roulette Wheel to select the potential mates:

	Random number	Selected chromosome
First spin of the wheel	0.2	Chromosome 4
Second spin of the wheel	0.6	Chromosome 1
Third spin of the wheel	0.5	Chromosome 1
Fourth spin of the wheel	0.9	Chromosome 3

Step 5: Each couple creates two offspring by crossover and then the parents die. In this study, we used the half uniform crossover scheme (HUX). In HUX, exactly half of the non-matching parents' genes are swapped. The crossover is performed in the following way. Let us assume that two parents are:

Parent 1: Chromosome 4: [1, 0, 1, 1, 0]

Parent 2: Chromosome 1: [1, 1, 1, 0, 1]

Find the number of genes (or bits) that are different in chromosomes (solutions) of two parents (Chromosome 4 and Chromosome 1): Different genes = [2^{nd} bit, 4^{th} bit, 5^{th} bit]. Therefore, the total number of genes that are different, n=3. Check if the number n is even or odd, If n is even, then both parents will contribute equal number ($n/2$) of corresponding bits that are different in two parents. Otherwise, if n is odd, then the offspring will carry $\lfloor (n+1)/2 \rfloor$ genes from the unique genes of the better parent and $\lfloor (n-1)/2 \rfloor$ genes from the unique genes of the other parent. In our example, n is 3—an odd number. Hence seed solutions will carry 2 genes (that are different in parents) of chromosome 4 (whose fitness score is 0.91) and 1 gene of chromosome 1(whose fitness score is 0.84).

Therefore, two offspring of chromosome 4= [1, 0, 1, 1, 0] and

chromosome 1=[1,1,1,0,1] are x=[1,0,1,1,1] and y=[1,1,1,1,0].

Step 6: Perform mutation on the offspring. A GA is sensitive to the mutation rate parameter value. The mutation rate is usually set to a low value such as 0.001 to avoid losing good solutions [135]. If the mutation rate is very high, the GA descends into a random search. Conversely, too low a mutation rate implies too little exploration in the search space. We set the mutation rate to 0.05 in order to reduce the risk of premature convergence. Generate a random number between 0 and 1 for each gene of a child chromosome. If the random number is greater than 0.95, mutate the gene.

Step 7: Repeat steps 3 to 7 until the stopping criterion is satisfied.

3.1.1.3 Binary Particle Swarm Optimization (PSO)

PSO is a population based stochastic optimization algorithm developed by Eberhart and Kennedy in 1995 [136]. PSO simulates the behaviour of bird flocking. The foraging strategy of birds is to follow the bird which is nearest to the food. In PSO, each single solution, called a particle, is considered as bird in the search space. A block diagram of PSO is presented in section 3.4.

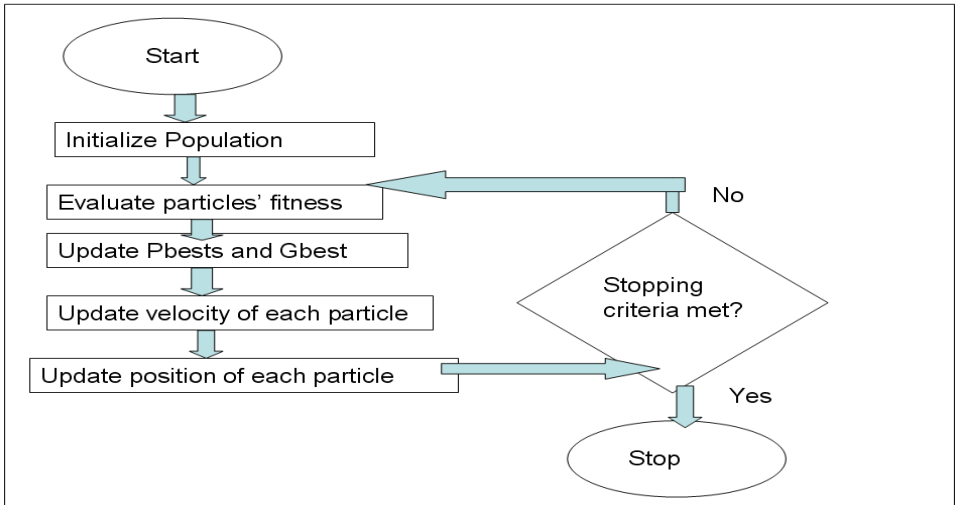

Figure 3.4: A block diagram of PSO

Pseudo code of binary PSO: The outline of the algorithm is as follows:

Step1: We fix the number of particles to be 100. In this study, we represent the particle's position as binary bit strings of length *N*, where *N* is the total number of features. Every bit represents a feature, the value '1' means the corresponding feature is selected while '0' not selected.

Step 2: Initialize the position and velocity of each particle randomly. The position (location) of a particle implies a subset of features. The velocity of a particle implies how many of the particle's bits or features should be changed, at a particular moment in time or in a certain iteration

Step 3: Evaluate the positions of the particles (i.e. prediction accuracy of the subset of features) using a GRNN. Update Pbest (particle's personal best) and Gbest (global best-to-date), if needed. Update total CPU time consumed. On the first iteration, set Pbest fitness value equal to the current fitness value and the Pbest location equal to the current location. The best fitness value among the Pbest fitness values of particles is assigned to the Gbest value and its location is assigned to the Gbest location.

Step 4: After finding the two best values, the algorithm updates the velocity of each particle with following equation (3.5). The velocities of particles have an upper velocity limit and a lower velocity limit.

If velocity<lower velocity limit, then velocity = minimum velocity.

If velocity>upper velocity limit, then velocity = maximum velocity.

$$V_{t+1} = wV_t + c_1 r_1 (Pbest[]\text{-}Present[]) + c_2 r_2 (Gbest[]\text{-}Present[]) \quad (3.5)$$

$$where, \quad w = 1 - \frac{T_{spent}}{T_{\max}}$$

Here, $V_{(t+1)}$ and V_t are the velocity of the particle at time *(t+1)* and *t* respectively. c_1 and c_2 are the acceleration constants, where c_1 moderates the maximum step size toward the personal best of that particle, while c_2 moderates the maximum step size toward the global best particle in just one iteration.

The values of these coefficients range from 0 to 4. Usually, $c_1 = c_2 = 2.$ The balance between global and local search throughout the course of a run is critical to the success of PSO. r_1 and r_2 are two separate random numbers between (0, 1). w is the linearly decreasing inertia weight. $T_{\max}$ represents the maximum time that the algorithm can take to perform a search operation and T_{spent} represents the total amount of time consumed so far. The inertia weight (w) is employed to control the effect of the previous velocity on the current velocity. It influences the trade-off between the global and local extrapolation abilities of the particles. A larger inertia weight facilitates a global search (searching new areas) whereas a smaller inertia weight facilitates a local search to fine-tune the current search area. The inertia weight w is a decreasing function of time. At the beginning of the process, a larger inertia weight is used for global extrapolation. The inertia weight becomes smaller and smaller for local search, as the search progresses. *Pbest[]* is the personal best position of the particle. *Present[]* is the current position of the particle. *Gbest[]* is the global best position.

Weng et al (2007) suggest setting the maximum velocity at one-third of the total number of features in the feature space, because this prevents an overly-large velocity [137]. A particle can be near to an optimal solution, but a high velocity may make it move

far away. By limiting the maximum velocity, particles cannot fly too far away from the optimal solution.

A simple example of estimating the velocity of a particle

Let, Maximum velocity =4 and Minimum velocity= 1.

w = inertia weight at time *(t+1)* =0.8; V_t=4; $c_1 = c_2 = 2$; $r_1 = 0.02$; $r_2 = 0.82$;

Gbest[] = [1,0,1,0,0]; *Pbest*[] = [1,0,0,0,1]; *Present* = [0,0,0,0,1]

Pbest[]-Present=[1,0,0,0,1]-[0,0,0,0,1]=1 [since 1 bit is different]

Gbest[]-Present=[1,0,1,0,0]-[0,0,0,0,1]=3 [since 3 bits are different]

$$V_{(t+1)} = wV_t + c_1 r_1 (Pbest[]\text{-}Present[]) + c_2 r_2 (Gbest[]\text{-}Present[])$$

$$V_{(t+1)} = (0.8 \times 4) + (2 \times 0.02 \times 1) + (2 \times 0.82 \times 3) = 8.16$$

Therefore, $V_{(t+1)}$=4 [since maximum velocity=4]

Step 5: Update positions of every particle by its new velocity based on the following position update strategies:

- If the particle's velocity $V_{(t+1)}$ is less than, or equal to, the position difference between the particle and Gbest (i.e. *Gbest[]-Present[]*) then randomly pick $V_{(t+1)}$ bits of the particle—that are different from that of *Gbest*—and change the bits.
- If the particle's velocity $V_{(t+1)}$ is greater than the position difference between the particle and Gbest, then in addition to changing all the different bits of the particle, randomly change $V_{(t+1)} -$ (*Gbest[]-Present[])* bits outside the different bits between the particle and Gbest.

Step 6: If stopping criteria are met, then stop. Otherwise go back to step 3.

3.1.1.4 Simulated Annealing (SA): SA, as a mathematical technique, was first formally introduced by S. Kirkpatrick, et al in 1983 [138]. SA is inspired by physical annealing of solids. Annealing is a heat treatment in which metals, glass or other materials are exposed to an elevated temperature for an extended time and then slowly cooled in an attempt to make them less brittle and more workable. A block diagram of SA is presented in figure 3.5.

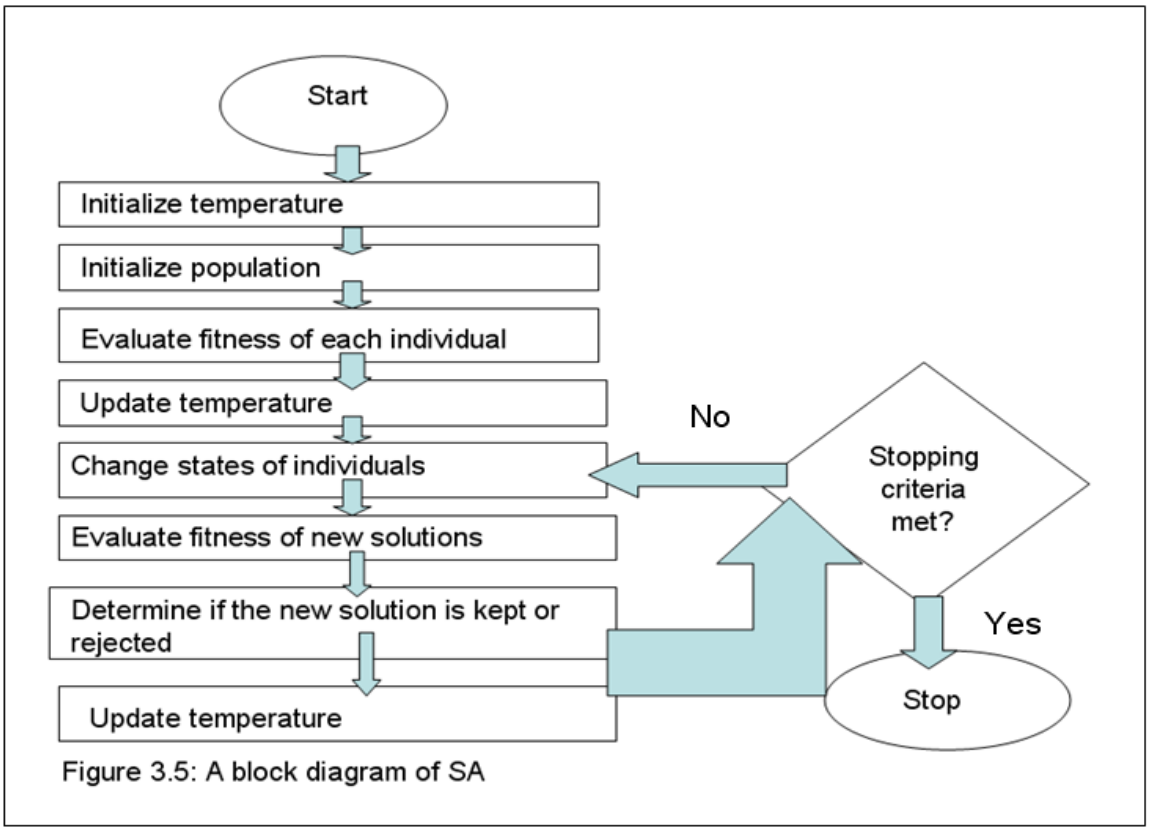

Figure 3.5: A block diagram of SA

Pseudo Code of SA

Step 1: Set the initial temperature **(**T_i**)**: T_i = Total run time for SA;

Step 2: Set the current temperature **(**T_c**):** $T_c = T_i$

Step 3: Initialize population: randomly select 100 individuals $I \in I(:100)$ from the pool of individuals for initial population.

Step 4: Evaluate the fitness of each individual: based on each individual I, extract a new dataset D_{new} from the (normalized) original dataset D with the features that are present in the solution of the individual. Evaluate the fitness scores $E_o \ (= E_o(1:100))$ of feature subsets using GRNN and store the information (feature subset solutions with fitness scores).

Step 5: Update the effective temperature **(**T**):** Based on the dimensionality of each individual evaluated in the previous step, retrieve the time elapsed in evaluating the individual. Calculate the total time spent T_{spent} on evaluating individuals of population by adding the time spent for each individual. Finally update the effective temperature: $T_c = T_c - T_{spent}$ (3.6)

Step 6: For all current feature subset vectors $I \in I(:100)$ change the bits of vectors with probability $p_{mutation}$: $p_{mutation} = 1 - E_o$ (3.7)

Step 7: Evaluate the fitness $E_n \ (= E_n(1:100))$ of the new candidate solutions if not already evaluated.

Step 8: Determine if this new solution is kept or rejected and update the database:

- If $E_n \geq E_o$, the new solution is accepted. The new solution replaces the old solution and E_o is set to E_n.
- If $E_n < E_o$, calculate the Boltzmann acceptance probability P_{accept}. If the acceptance probability is greater than or equal to a random number between 0 and 1, the new solution is accepted and it replaces the old one and E_o. On the other hand, if the

acceptance probability is less than the random number, the new solution is rejected and the old solution stays the same : $P_{accept} = \exp(-(E_o - E_n)/T_c)$ (3.8)

Step 9: Update the effective temperature.

Step 10: If the effective temperature is greater than or equal to zero, return to step 6. Otherwise, the run is finished.

Hybrid Search Algorithm

3.1.1.5 FW (Filter + Wrapper):

This hybrid algorithm consists of a number of popular filter methods and a stochastic algorithm. FW is a three stage algorithm. Figure 3.6 shows a schematic block diagram of the FW.

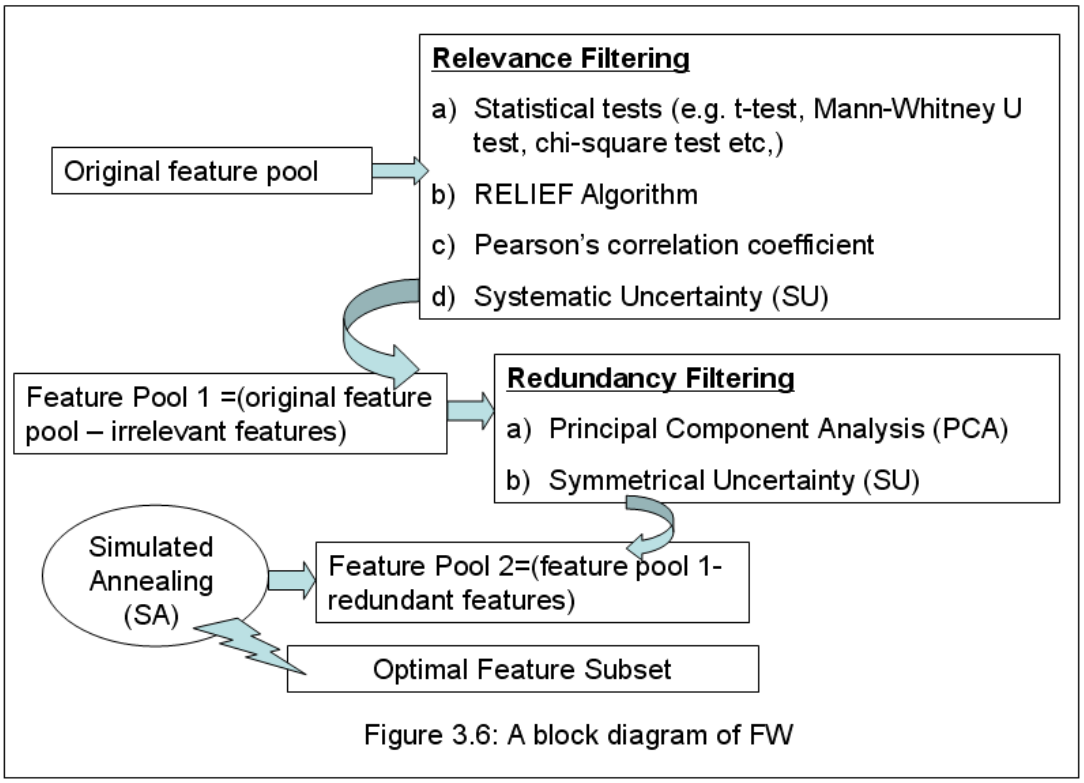

Figure 3.6: A block diagram of FW

Stage 1: Relevance Filtering: Since it is hard to decide which filter method is best for a dataset because the performance of a filter method varies with different datasets [139], we use a number of popular filter methods to filter out irrelevant features. FW eliminates a feature when all of these filter methods—a) Statistical hypothesis testing, b) Pearson's correlation coefficients, c) Relief algorithm, and d) symmetric uncertainty— dismiss the feature as irrelevant at the 0.05 level.

a) Statistical Hypothesis Testing: The significance test is a classical feature selection approach.

Pseudo code of hypothesis testing:

1. Establish hypotheses:

   Null Hypothesis, H_o : There is no association between *X* and *Y*.

   Alternative Hypothesis, H_a : There is an association between *X* and *Y*.

2. Calculate test statistic
3. Assess significance level: We used the significance level of 5% (a commonly used level) in all our statistical tests.
4. Finally, decide whether to accept or reject the null hypothesis.

The following tests are frequently used to measure how strongly a particular independent variable *X* explains variations in the dependent variable *Y*.

Student's *t*-test and *z*-test: The student's *t*-test is perhaps the most widely used parametric test. This can be used when the following assumptions are met:

- Data points are independent from each other.
- T-test should be used when *n* (the size of each group) is smaller than 30.

- The distributions should be normal for the equal and unequal variance t-test.
- The variances of the samples should be the same for the equal variance t-test.
- All individuals must be selected at random from the population.
- All individuals must have equal chance of being selected.
- Sample sizes should be as equal as possible but some differences are allowed.

The observed values of the feature are separated into two groups based on the target variable and then homogeneity of variance is tested for using the *F*-test. The formula of confidence interval (CI) for comparing means of two groups with unequal variances is:

$$CI = \bar{x}_A - \bar{x}_B \pm t_{\alpha/2,df} \sqrt{\left(\frac{s_A^2}{n_A} + \frac{S_B^2}{n_B} \right)} \qquad (3.9)$$

The formula of confidence interval (CI) for comparing means of two groups with equal variances is:

$$CI = \bar{X}_A - \bar{X}_B \pm t_{\alpha/2,df} \sqrt{s_p^2 \left(\frac{1}{n_A} + \frac{1}{n_B} \right)} \qquad (3.10)$$

Where, $S_p^2 = \frac{(n_A - 1)S_A^2 + (n_B - 1)S_B^2}{n_A + n_B - 2}$

Where, $\bar{x}_A$ and $\bar{x}_B$ are the sample means of the independent variable *X*. s_A^2 and S_B^2 are the sample variances. n_A and n_B are the sizes of the two samples. The *t*-critical value ($t_{\alpha/2,df}$) must be found from a table or it will be given by statistical software. To find it, only alpha level (α) and the degrees of freedom (df) are needed. Customarily the alpha level (cut-off

point) is set at 0.05. The df is simply the number of data points in both datasets minus 2. If the confidence interval contains 0, then the feature *X* is assumed to be irrelevant to the dependent variable of interest. If the confidence interval does not include zero, it is concluded that the feature *X* is relevant.

z-test is preferable when *n* is greater than 30. For the *z*-test degrees of freedom are not required since z-scores of 1.96 and 2.58 are used for 5% and 1% respectively.

$$95\% \quad CI = \bar{X}_A - \bar{X}_B \pm 1.96\sqrt{s_p^2\left(\frac{1}{n_A}+\frac{1}{n_B}\right)} \qquad (3.11)$$

$$99\% \quad CI = \bar{X}_A - \bar{X}_B \pm 2.58\sqrt{s_p^2\left(\frac{1}{n_A}+\frac{1}{n_B}\right)} \qquad (3.12)$$

where, $S_p^2 = \frac{(n_A - 1)\hat{S}_A^2 + (n_B - 1)\hat{S}_B^2}{n_A + n_B - 2}$

If the confidence interval includes zero, then the null hypothesis (of no association) cannot be rejected at the stated level of significance.

The Mann-Whitney U Test: is also called the Mann-Whitney-Wilcoxon or Wilcoxon rank-sum test or Wilcoxon-Mann-Whitney test. The *t*-test is a parametric statistical test. It is sensitive to departures from normality, and to the difference between the two variances, especially when the sample sizes are different. Hence, whenever there exists some doubts about the validity of these assumptions, the nonparametric Mann-Whitney test is an excellent alternative. However, the Mann-Whitney test is less powerful than the *t* test when

its assumptions are met, because the Mann-Whitney test first converts the values of the observations into ranks, and some information is lost in the process.

The observations are divided into two groups according to the response variable *Y*. To perform the Mann-Whitney test, we first rank all the values of a feature *X* from low to high, paying no attention to which group each value belongs. These rankings are then re-sorted into the two separate samples. We then use the following formula to calculate the value of Mann-Whitney U-test:

$$U = N_1 N_2 + \frac{N_2(N_2+1)}{2} - R_2 \qquad (3.13)$$

Where U= Mann-Whitney U test statistic; N_1 =Sample Size 1; N_2= Sample Size 2, with N_2 representing the larger of the two samples, if they are different; R_2=the observed sum of ranks for sample 2; and $N_1 N_2 + \frac{N_2(N_2+1)}{2}$ = the maximum possible value of R_2

The decision rule is to reject the null hypothesis if the *p*-value of the test statistic for two-tailed test at α=0.05 is less than or equal to the significance level. For more details of the test, the reader is referred to Siegel (2d Ed.) [140].

Chi-Square (χ^2) Test of Independence: The χ^2-test is a tool to look at the relationship between two categorical variables. To perform chi-square test of independence, first we have to calculate expected frequencies for all combinations of categories. A contingency table (similar to that displayed in example table 3.1.1.5.1) is created from frequency data. Each cell of the contingency table contains observed frequency counts. The first step, then,

in calculating the χ^2 statistic is generating the expected frequency (presented in parenthesis in example table 3.1.1.5.1) for each cell of the table as in equation (3.14).

$$\text{Expected Frequency} = \frac{(\text{Row total} \times \text{Column total})}{\text{total n for table}} \qquad (3.14)$$

Example Table 3.1.1.5.1: A simple 2×2 Contingency Table

	Y=1	Y=0	Total
X=1	A(a)	B (b)	A+B
X=0	C (c)	D (d)	C+D
Total *n*	A+C	B+D	A+B+C+D

Here, *A, B, C, D* represent actual frequency and *a, b, c, d* represent expected frequency.

The χ^2 statistic is calculated as in equation (3.15).

$$\chi^2 = \sum \frac{(\text{observed frequency} - \text{expected frequency})^2}{(\text{expected frequency})} \qquad (3.15)$$

$$\chi^2 = \frac{(A-a)^2}{a} + \frac{(B-b)^2}{b} + \frac{(C-c)^2}{c} + \frac{(D-d)^2}{d}$$

Degrees of freedom: $\text{df} = (\text{numberof columns-1})(\text{numberof rows-1})$

The level of significance (*p* value) is assessed using the χ^2 and the degrees of freedom. If the $p \leq 0.05$, we reject the null hypothesis.

b) RELIEF Algorithm: The RELIEF algorithm, originally proposed by Kira and Rendell, is a popular filter feature selection [141]. A block diagram of RELIEF algorithm is presented in Figure 3.7.

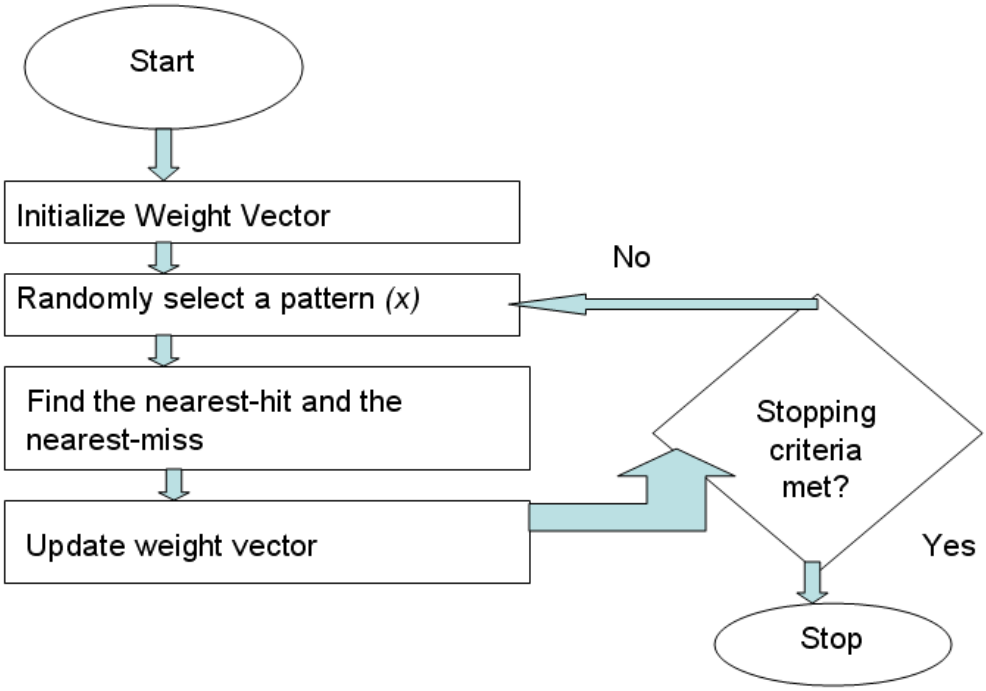

Figure 3.7: A block diagram of RELIEF algorithm

The RELIEF algorithm is used to remove irrelevant features. It assigns relevance values to features. The basic idea of the RELIEF algorithm is to iteratively estimate the feature weights according to their ability to discriminate between neighbouring patterns. The algorithm holds a weight vector over all features and updates this vector according to the input patterns presented. The initial weights of all features are set to be zero. In each iteration, a pattern x is randomly chosen and then two nearest neighbours of x are found, one from the same class (termed the *nearest hit* or NH) and the other from different class (termed the nearest miss or NM). The weight of the *i-th* feature is then updated as in equation (3.16):

$$w_i = w_i + \left| x^{(i)} - NM^{(i)} \right| - \left| x^{(i)} - NH^{(i)} \right| \qquad (3.16)$$

c) Pearson's Correlation Coefficient: measures the strength of the relation between a feature (x) and a dependent variable (y). The Pearson correlation coefficient R is calculated using the following formula:

$$R = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \qquad (3.17)$$

The value of correlation coefficient ranges between -1 and +1. The greater the absolute value of a correlation coefficient, the stronger the relationship. The Pearson's correlation coefficient is based on the assumptions of normality, linearity and homoscedasticity. If $|R| \leq 0.05$, we remove the feature from the feature pool.

d) Relevance Analysis using Symmetric Uncertainty (SU): In information theory, *entropy* is a measure of the amount of uncertainty about a random variable *Y*. Equations (3.18) and (3.20) define the entropy of Y before and after observing values of another variable X, respectively. Equation (3.19) defines the entropy of X.

$$H(Y) = -\sum_i p(y_i) \log_2 p(y_i) \qquad (3.18)$$

$$H(X) = -\sum_j p(x_j) \log_2 p(x_j) \qquad (3.19)$$

$$H(Y/X) = -\sum_j p(x_j) \sum_i p(y_i/x_j) \log_2 p(y_i/x_j) \qquad (3.20)$$

Where, $p(y_i)$ and $p(x_j)$ are the prior probabilities for all values of random variable *Y* and *X*, respectively. $p(y_i/x_j)$ is the conditional probability of y_i given x_j.

The amount by which the entropy of *Y* decreases reflects additional information about *Y* provided by *X* and is called *Information Gain* (or *mutual information*) as shown in equation (3.21). Mutual Information, as a nonlinear measure of correlation, is frequently used in feature subset selection. It measures how well a given variable *X* distinguishes instances into classes of variable *Y*.

$$I(Y;X) = H(Y) - H(Y/X) \qquad (3.21)$$

Mutual Information is nonnegative (i.e., $I(Y;X) \geq 0$) and symmetric (i.e. $I(Y;X) = I(X;Y)$). $I(Y;X) = 0$, if and only if *X* and *Y* are independent random variables. The higher the information gain, the more informative the feature and, thus, the more predictive power it has. A drawback of *Information Gain* is that it favours features with many different values.

The symmetric uncertainty (SU) solves the drawback of information gain by dividing it by the sum of the entropies of class labels *Y* and features *X* ; and can, therefore, be used as a criterion for feature selection [180].

$$SU(Y;X) = 2\left[\frac{I(Y;X)}{H(X)+H(Y)}\right] \qquad (3.22)$$

SU is applied to measure the correlation between features and target variable. It ranges between 0 and 1. If a feature has a low symmetric uncertainty to the target variable, it implies that the feature has poor prediction ability to the target variable. On the other hand,

the feature has strong prediction ability to the target variable if the SU is high. We remove a feature if the symmetrical uncertainty between the feature and the target variable is less than 0.05.

Stage 2: Redundancy Filtering: FW uses PCA and SU to filter out redundant features from the relevant features of the output variable. FW removes only those features that are regarded as redundant by both methods (i.e. PCA and SU).

a) Principal Component Analysis (PCA): is often applied to high dimensional data with redundancy. PCA transforms a set of *p* correlated variables $X^T = [X_1, X_2, X_3]$ into a set of *p* uncorrelated variables called principal components (PCs). The new variables (PCs) are linear combinations of the original variables.

PCA involves decomposition of the covariance matrix (COV matrix) of the original dataset so that the original coordinate system of correlated variables is transformed to a new set of uncorrelated variables (PCs). The new rescaled variables (PCs) have unit variance and are independent of one another.

The PCA is represented by

$$COV \quad u_j = \lambda_j u_j \qquad (3.23)$$

u_j =eigenvector and λ_j =eigenvalue. There are as many eigenvectors as there are input variables.

The actual transformation is done through eigenvectors also called loadings or weights. Each eigenvector contains loadings for each of the original variables that transform them to

the new variables (PCs). The values of the new variables are called scores (or PC scores) that are obtained by projecting the original inputs onto the eigenvectors (i.e. multiplying original inputs with corresponding loadings). To obtain the transformed variables (PCs), the original variables are multiplied by the weights (or loadings).

Thus, the equation for the value of the first principal component can be written as:

$$PC_1 = u_{11}x_1 + u_{12}x_2 + u_{13}x_3 \qquad (3.24)$$

Where, $\{x_1, x_2, x_3\}$ represent normalized original input variables. The first row of the eigenvector matrix defines the weights for the first PC (PC_1). The first eigenvector is given by $\{u_{11}, u_{12}, u_{13}\}$.

Eigenvalues denotes the variance of the new variables (PCs) and they are in descending order. Therefore, the first PC represented by the first eigenvector captures the largest amount of variation in the original data; each subsequent PC captures the largest amount of remaining variance, and so on. The amount of variation captured by each PC is given by their corresponding eigenvalues. Theoretically, there are as many PCs as there are input variables, but because the first few PCs capture most of the variance, a threshold naximum variance can be defined as suitable cutoff point (90 percent, 95 percent, etc) taking into account the noise and and variance in the data to select an adequate number of PCs that sufficiently represent the original data while discarding the rest. The selected PCs can then be used as inputs to a machine learning algorithm as an alternative to using original variables.

The algorithm FW retains a PC if it accounts for more than 1% of variance. The selected PCs are then used as features as an alternative to using the original variables.

b) Redundancy Analysis using Symmetrical Uncertainty (SU): The symmetrical uncertainty (SU) between each pair of features is computed to measure the feature redundancy. If the SU between two features is greater than 0.95, we remove the feature that has the lower SU with the output variable.

Stage 3: Search for an Optimal Feature Subset Solution: FW uses SA to find an optimal feature subset solution from the selected PCs since our empirical results suggests that SA is better than other stochastic algorithms. The pseudo code of SA is given in section 3.1.1.4.

Greedy Search Algorithms

3.1.1.6 Sequential Backward Selection (SBS)

A block diagram of SBS is presented in figure 3.8.

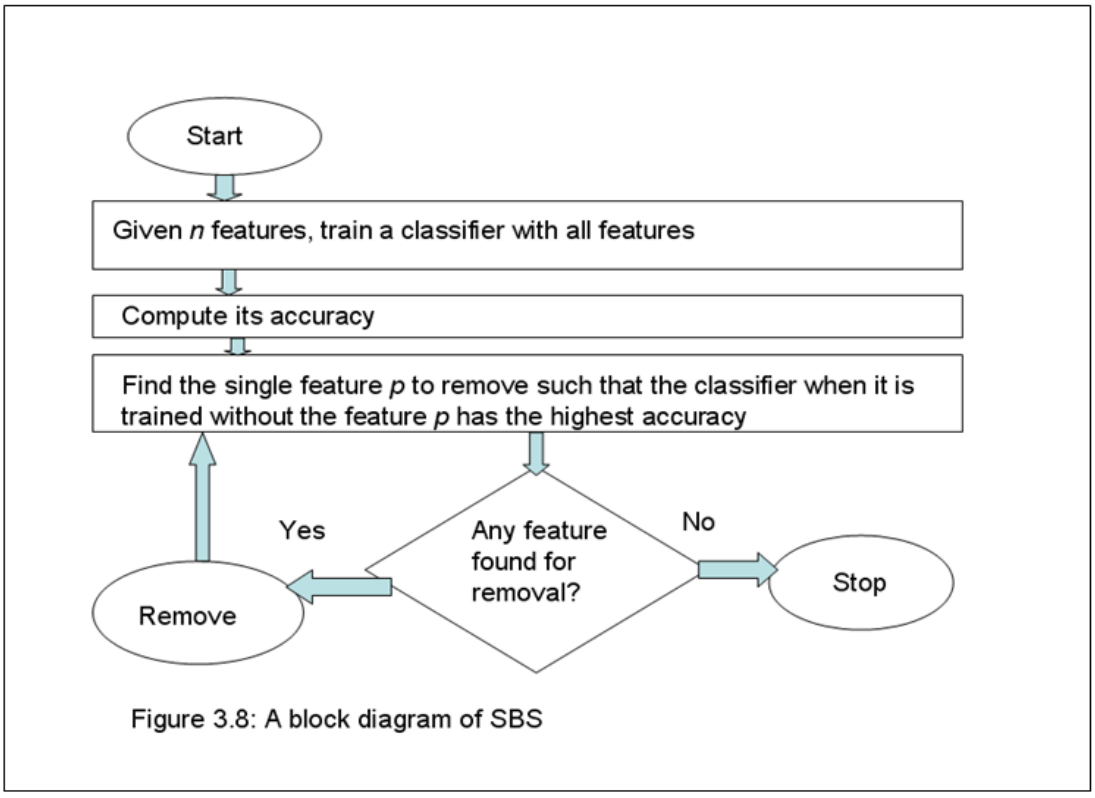

Figure 3.8: A block diagram of SBS

Step 1: *Initialize feature set:* Let's assume that Y is a feature set that contains all the features: $Y = \{x_1, x_2, \cdots, x_n\}$

Step 2: *Evaluate the feature set* Y*:* Train a GRNN with the feature set Y. Evaluate the network's performance on the test set.

Step 3: *Prune the least important feature from the set* Y*:* Construct new feature sets by excluding one feature of root set (Y) each. Evaluate each new feature set in order to estimate the marginal contribution of each feature used in a trained GRNN. Delete the least important feature from the set (Y) if the accuracy of the GRNN does not drop and recursively repeat this step for the remaining features.

3.1.1.7 Sequential Forward Selection (SFS)

A block diagram of SFS is presented in figure 3.9.

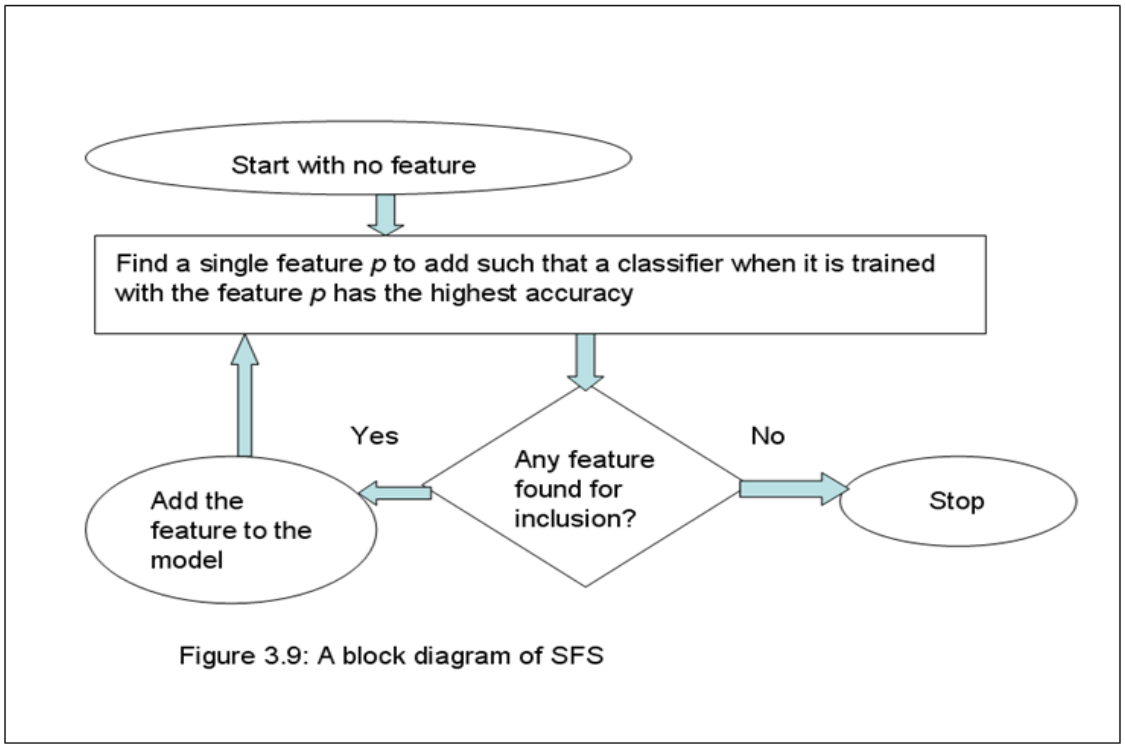

Figure 3.9: A block diagram of SFS

Step 1: *Initialize feature set:* Let's assume that Y is an empty feature set: $Y = \{\}$

Step 2: *Find the best feature and update* Y*:* A GRNN was constructed with one feature at a time. The trained networks were evaluated on the test set to identify the best feature. Add the best feature into Y.

Step 3: *Find the next best feature and update* Y*:* Add one new feature to the GRNN at a time. Evaluate trained networks and identify the next best feature. Add the next best feature into Y. Thus, sequentially add the best feature that improves results more than the others until there is no more improvement.

3.1.1.8 Sequential Floating Backward Selection (SFBS)

A block diagram of SFBS is presented in figure 3.10.

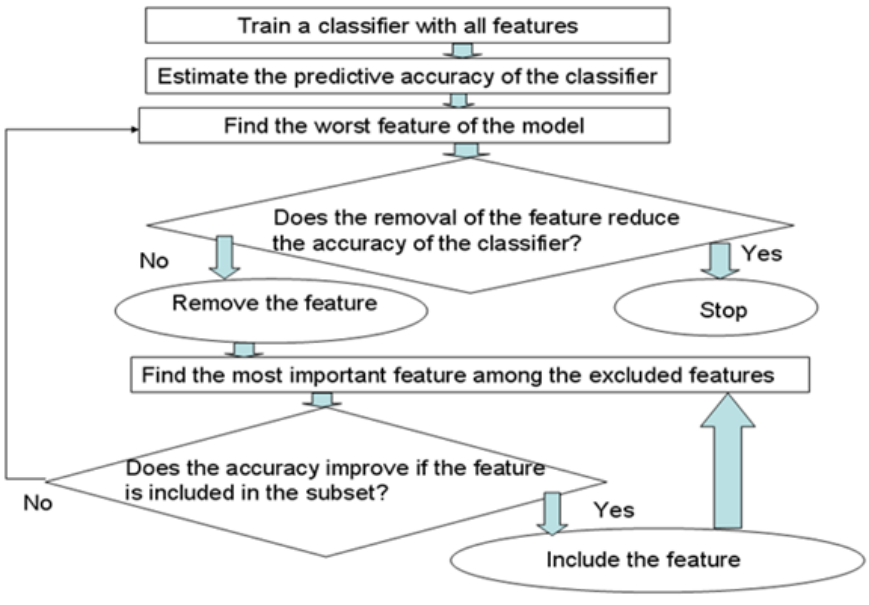

Figure 3.10: A block diagram of SFBS

Step 1: *Initialize feature set:* Let's assume that Y is a feature set that contains all the features: $Y = \{x_1, x_2, \cdots, x_n\}$

Step 2: *Evaluate the feature set* Y*:* Train a GRNN with the feature set Y and measure the GRNN's performance on the test set.

Step 3: *Exclusion:* Find the worst feature using the basic sequential backward selection (SBS) and remove the feature from the current feature set Y if the accuracy of GRNN does not decrease**.**

Step 4: *Inclusion:* Use the basic SFS method to find the most important feature among the excluded features. Add the most significant feature to the current subset Y if the accuracy of GRNN improves. Repeat this step until the performance of GRNN does not improve on inclusion of additional features.

Step 5: Return to step 3. SFBS stops when the last iteration through all events has not led to any improvements.

3.1.1.9 Sequential Floating Forward Selection (SFFS)

A block diagram of SFFS is presented in figure 3.11.

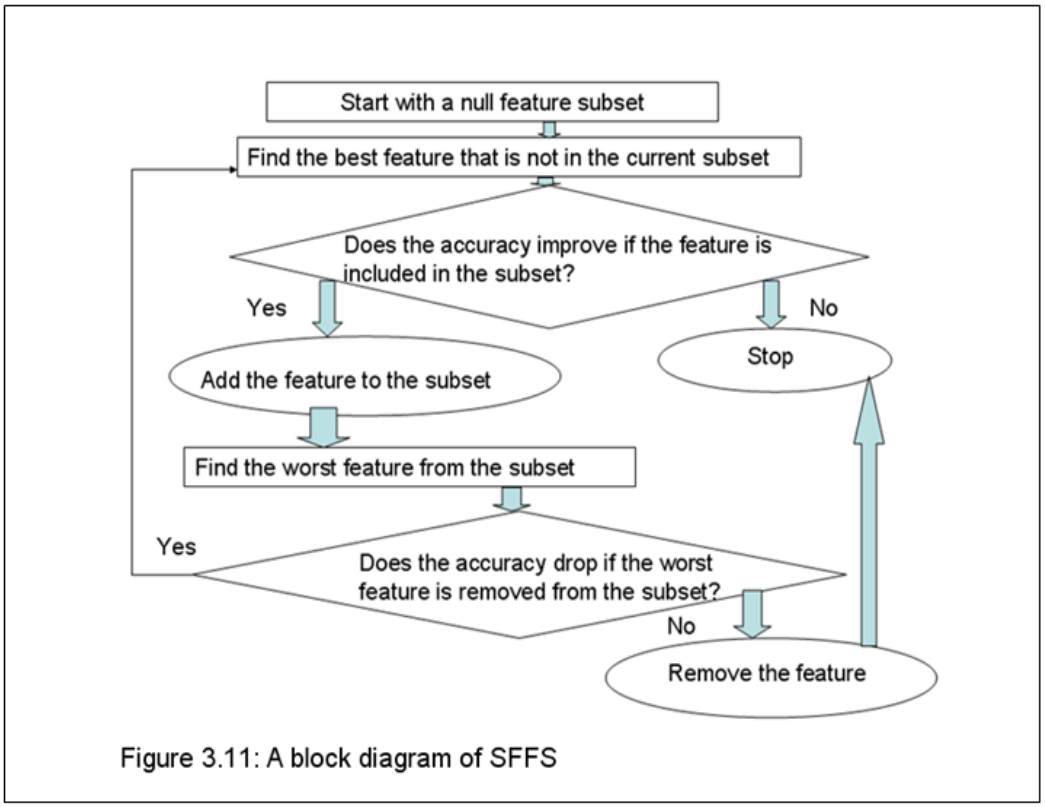

Figure 3.11: A block diagram of SFFS

Step 1: *Initialize feature set:* Let's assume that Y is an empty feature set: $Y = \{\}$

Step 2: *Inclusion:* Using the basic SFS method, select the best feature and update Y.

Step 3: *Exclusion:* Using the basic SBS method, select the worst feature and remove it from Y. Repeat this step until the performance of GRNN does not deteriorate.

Step 4: Return to step 2. SFFS stops when additional features cannot be added or removed from Y.

3.1.2 Implementation of Conventional Missing Data Imputation Algorithms

We compared the proposed missing data imputation algorithm with a number of single imputation algorithms as well as several multiple imputation algorithms.

We treat the variable with missing value as target, the remaining variables as predictors. We explain how we implemented these algorithms in this section.

3.1.2.1 Single Imputation (SI) Algorithms:

3.1.2.1.1. ***Zero Imputation (ZI)*** replaces the missing values with zero (Figure 3.12).

Original Dataset

	Features		
Instance	***x1***	***x2***	***x3***
1	5	3	2
2	?	1	?
3	7	5	?
4	2	?	4

Imputed Dataset

	Features		
Instance	***x1***	***x2***	***x3***
1	5	3	2
2	0	1	0
3	7	5	0
4	2	0	4

Figure 3.12: Missing data imputation using Zero Imputation

3.1.2.1.2. ***Mean Substitution (MS)*** fills in the missing values by their variable means (Figure 3.13).

Original Dataset

	Features		
Instance	***x1***	***x2***	***x3***
1	5	3	2
2	?	1	?
3	7	5	?
4	2	?	4
Mean	5	3	3

Imputed Dataset

	Features		
Instance	x1	x2	x3
1	5	3	2
2	5	1	3
3	7	5	3
4	2	3	4
Mean	5	3	3

Figure 3.13: Missing data imputation using Mean Substitution

3.1.2.1.3. ***Hot Deck Imputation*** **(HD)** works in the following two stages as shown in Figure 3.14.

Step 1: Find the closest donors for the missing value from complete records using the Euclidean distance matching function.

Step 2: Substitute the most similar case's value for the missing value.

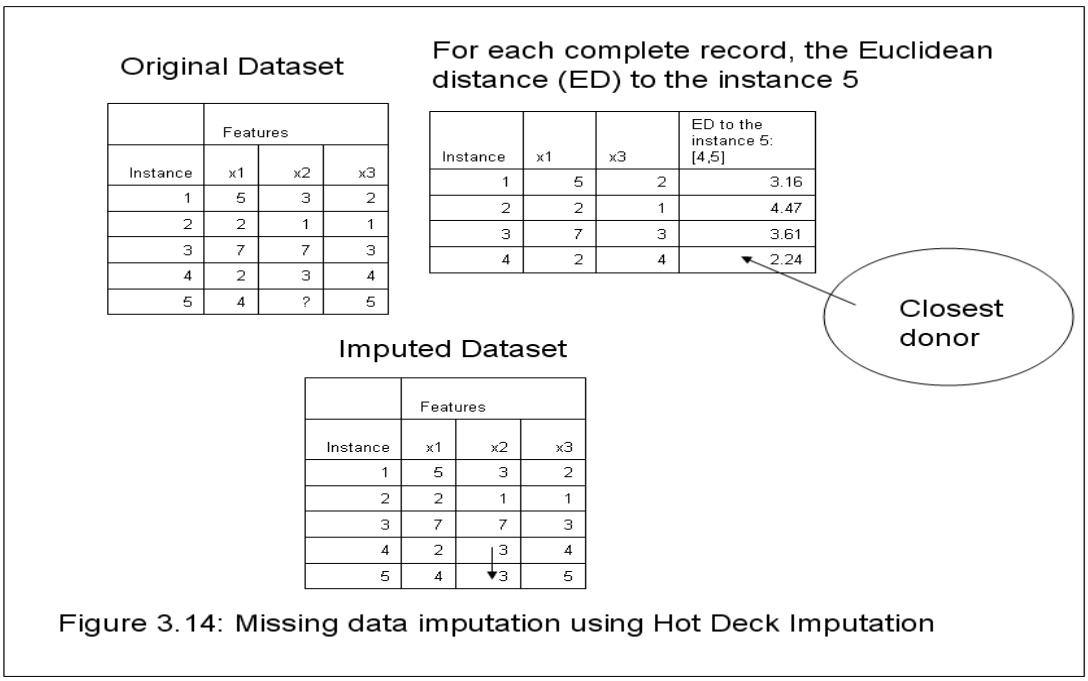

Figure 3.14: Missing data imputation using Hot Deck Imputation

3.1.2.1.4. ***K-Nearest Neighbours Algorithm (KNN)*** imputes missing values by the average value of the *K* nearest patterns, as presented in equation (3.25) and Figure 3.15.

$$x_{ij} = \frac{\sum_{k=1}^{K} x_{kj}}{k} \qquad (3.25)$$

Where x_{ij} represents a missing value in the *j*-th variable of the *i*-th instance. *K* is the number of nearest neighbours and x_{kj} is the value of the *j*-th variable of the *k*-th nearest neighbour.

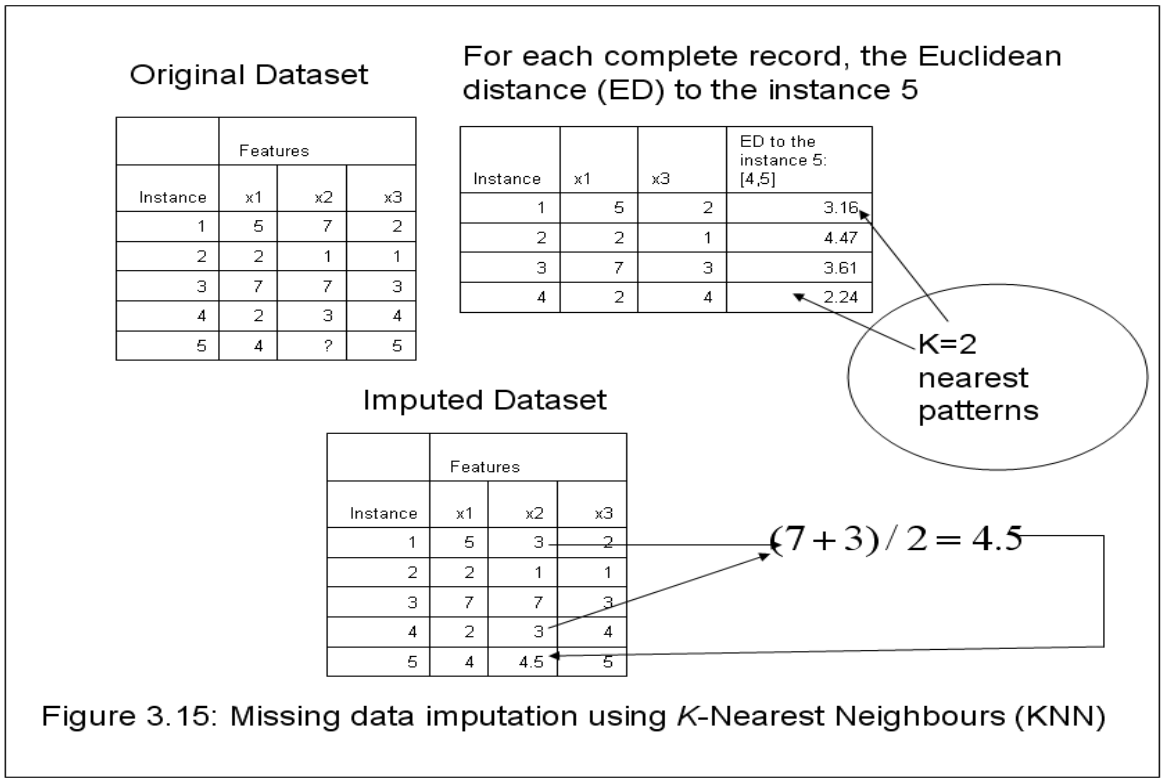

Figure 3.15: Missing data imputation using *K*-Nearest Neighbours (KNN)

3.1.2.1.5. ***Weighted K-Nearest Neighbours (WKNN) algorithm*** replaces missing values with a weighted average of the *K*-nearest neighbours, as presented in Figure 3.16 and Equation (3.26). Let's assume that the value of the *j*-th variable of the *i*-th instance (x_{ij}) is missing.

Using WKNN algorithm, x_{ij} is replaced by $$x_{ij} = \frac{\sum_{k=1}^{K} w_k x_{kj}}{\sum_{k=1}^{K} w_k} \qquad (3.26)$$

where $w_k = \frac{1}{d_{ik}}$

Here, w_k is the weight associated to the *k*-th nearest neighbour. x_{kj} is the value of the *j*-th variable of the *k*-th nearest neighbour. d_{ik} is the Euclidean distance between the *i*-th pattern

(the instance with the missing value) and the *k*-th nearest neighbour. *K* is the number of nearest neighbours. In other words, the weight w_k of the *k*-the nearest donor is equal to the reciprocal of its Euclidean distance to the instance with missing values.

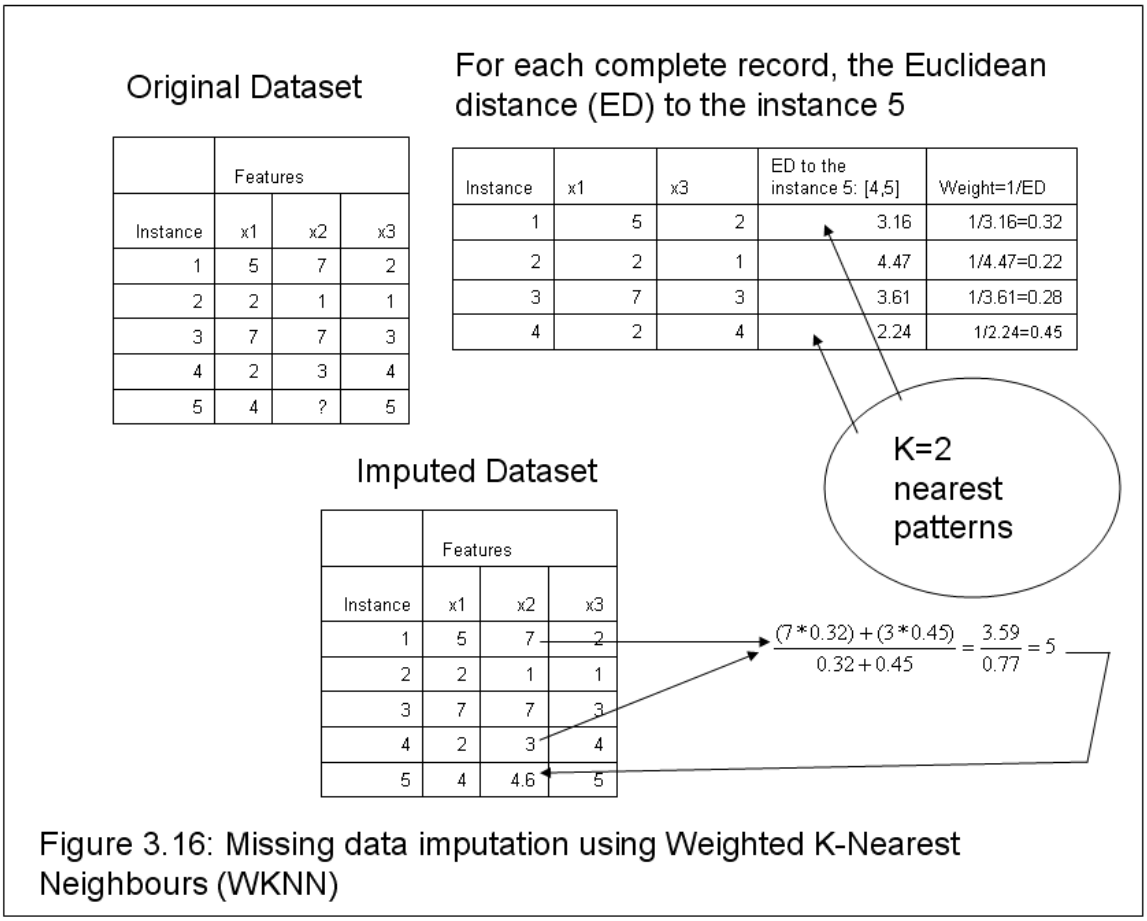

Figure 3.16: Missing data imputation using Weighted K-Nearest Neighbours (WKNN)

3.1.2.1.6. ***Expectation Maximization (EM)*** is a procedure for parameter estimation in the presence of missing data. It is an iterative two-step (E-step and M-step) process.

The **E (Expectation) step** fills in the missing values using estimated values for the model parameters (initially random values are assigned to these parameters).

Then, the **M (Maximization) step** re-estimates the parameters by using the observed and imputed values to maximize the log-likelihood of the model.

The algorithm iterates from E to M steps until the log-likelihood converges to a stationary point. The implementation of EM assumes all attributes to be independent and normally distributed.

3.1.2.1.7. Single Imputation with Neural Network-based Algorithms

Three neural network models (MLP, RBFN, and GRNN) were designed for the imputation of missing data. Neural networks were trained as described in section 3.1.4. We also tested four neural network ensemble models: HES SI (Heterogeneous ensemble with simple averaging for Single Imputation), HEW SI (Heterogeneous ensemble with weighted averaging for Single Imputation), HOS SI (Homogeneous ensemble with simple averaging for Single Imputation), and HOW SI (Homogeneous ensemble with weighted averaging for Single Imputation). The members in heterogeneous ensembles (HES SI and HEW SI) are a MLP, a RBFN and a GRNN, whereas the members in homogeneous ensembles (HOS SI and HOW SI) are GRNNs. The homogeneous and heterogeneous ensembles are discussed in section 3.1.5.

Like EM, neural networks (MLP, RBFN, and GRNN) and neural network ensembles (HES SI, HEW SI, HOS SI, and HOW SI) impute missing values by two iterative steps: (1) imputing missing values based on initial estimates of model parameter values. (2) Updating model parameters.

3.1.2.2 Multiple Imputation

We implemented the following multiple imputation algorithms: MCMC (Markov Chain Monte Carlo—a multiple-imputation version of the EM algorithm), MLP MI (Multilayer

Perceptron with multiple imputation), RBFN MI (Radial Basis Function Networks with Multiple Imputation), GRNN MI (Generalized Regression Networks with Multiple Imputation), HES MI (a multiple-imputation version of the HES SI), HEW MI (a multiple-imputation version of the HEW SI), HOS MI (a multiple-imputation version of the HOS SI), HOW MI (a multiple-imputation version of the HOW SI), HD MI (Hot Deck Multiple Imputation), KNN MI (K-Nearest Neighbours Algorithm with Multiple Imputation) and WKNN MI (Weighted K-Nearest Neighbours Algorithm with Multiple Imputation). A commonly practiced multiple imputation (MI) analysis consists of three steps: Imputation, Analysis and Pooling. The figure 3.17 illustrates these steps:

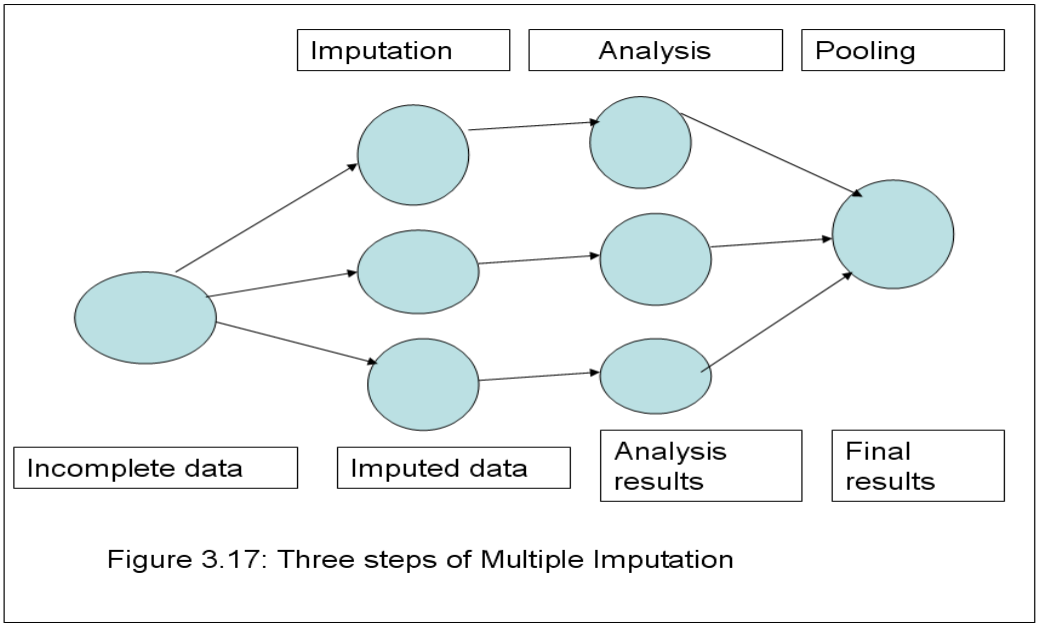

Figure 3.17: Three steps of Multiple Imputation

Imputation: In this step, each missing value is imputed for several (*M*) times, which yields *M* (*M*=3 in the Figure 3.17) complete datasets. In MCMC, MLP MI, and RBFN MI, multiple imputations are generated by randomly selecting *M* sets of initial parameter estimates. Starting at different initial parameter guesses will generally lead to different local

optimal solutions. In GRNN MI, HOS MI, HOW MI, HES MI, HEW MI, KNN MI and WKNN MI, *M* new training sets are randomly extracted from the original training set, each comprising 70% training examples. Different training sets will lead to different imputation models. *M* different models will lead to different complete datasets. In HD MI, *M* donors are randomly chosen for each missing value from the pool of potential donors that will lead to *M* different complete datasets. Like EM, each of the following multiple imputation algorithms (MCMC, MLP MI, RBFN MI, GRNN MI, HES MI, HEW MI, HOS MI, and HOW MI) is an iterative process that alternatively fills in missing values and makes inferences about the unknown parameters. However, these algorithms do this in a stochastic or random fashion.

The **Imputation *I*-step***:* Given an estimated mean vector and covariance matrix of parameters, the *I*-step simulates the missing values for each data point independently by randomly drawing parameters from their conditional distribution.

The **posterior *P*-step**: Given a complete dataset obtained in the previous step, the mean vector and covariance matrix are recomputed. The new mean vector and covariance matrix are then used in the next *I*-step.

The above two steps are iterated until the mean vector and covariance matrix stabilize.

Analysis: Each of the *M* completed datasets are analyzed. This step results in the *M* analysis results. For example, with *M* imputations, *M* different sets of mean and variance for a parameter Q can be computed. Let us suppose that $\hat{Q}_i$ and $\hat{U}_i$ are the mean and variance estimates from the *i*'th imputed dataset, $i = 1,2,\cdots,M$.

Pooling: The *M* analysis results are integrated into a final result (i.e. the vector of means and the variance-covariance matrix of model parameters). Simple rules exist for combining the *M* analysis results. For example, the combined mean estimate for Q from multiple imputation is the average of the *M* complete data estimates:

$$\overline{Q} = \frac{1}{M}\sum_{i=1}^{M}\hat{Q}_i \qquad (3.27)$$

Let's suppose that $\overline{U}$ is the within-imputation variance, which is the average of the M complete data-estimates:

$$\overline{U} = \frac{1}{M}\sum_{i=1}^{M}\hat{U}_i \qquad (3.28)$$

And B is the between-imputation variance

$$B = \frac{1}{(M-1)}\sum_{i=1}^{M}(\hat{Q}_i - \overline{Q})^2 \qquad (3.29)$$

Then the variance estimate associated with $\overline{Q}$ is the total variance (Rubin, 1987) [11]

$$T = \overline{U} + \left(1 + \frac{1}{M}\right)B \qquad (3.30)$$

3.1.3 Implementation of Conventional Univariate Time Series Forecasting Algorithms

Time series models use past values of the variable to forecast the future values of the considered variable. Generally conventional time series forecasting techniques determine the number of steps into the past needed to forecast the current value pursuing a trial and

error approach. The primary difference between univariate time series models and other types of models is that lag variables of the target variable are used as predictor variables. Let's assume $Z_1, Z_2, ..., Z_n$ is a series of observations of a random variable Z_t. We write Z_t for the observation made at time *t*. We do assume that in a time series, observations are equally spaced in time. In order to forecast the future value of a time series, we assume that for each time point *t*, Z_t is a random variable (example table 3.1.3.1). Therefore, the feature space of a univariate time series includes a large number of time-lagged input variables.

Example Table 3.1.3.1: An example of Lag Variables (autoregressive terms)

	Target Variable	***Lagged Input Variables***		
Instances	Variable Z_0 (observations at lag 0)	Variable Z_1 (observations at lag 1)	Variable Z_2 (observations at lag 2)	Variable Z_3 (observations at lag 3)
1	$\boldsymbol{Z_n}$	$\boldsymbol{Z_{n-1}}$	$\boldsymbol{Z_{n-2}}$	$\boldsymbol{Z_{n-3}}$
2	$\boldsymbol{Z_{n-1}}$	$\boldsymbol{Z_{n-2}}$	$\boldsymbol{Z_{n-3}}$	$\boldsymbol{Z_{n-4}}$
3	$\boldsymbol{Z_{n-2}}$	$\boldsymbol{Z_{n-3}}$	$\boldsymbol{Z_{n-4}}$	$\boldsymbol{Z_{n-5}}$
4	$\boldsymbol{Z_{n-3}}$	$\boldsymbol{Z_{n-4}}$	$\boldsymbol{Z_{n-5}}$	$\boldsymbol{Z_{n-6}}$

We built two models using the time series forecasting algorithms: *Net-P* for forecasting of the stationary conditional mean and *Net-Q* for forecasting of the stationary conditional volatility. We translate the estimated stationary values into non-stationary states. Firstly, the *Net-P* is trained on the stationarized observed time series. Then, the *Net-Q* is trained on the squared residuals of the *Net-P*. In the *t*-step ahead out-of-sample forecasting, we convert the stationary conditional variance to a non-stationary conditional variance by multiplying the stationary variance by t .

Time series forecasts are generated in an incremental manner. In this case, the initial forecast is carried out only one time step into the future. Then, this one-step ahead forecast

value is treated as an actual value (rather than as a forecast value); it becomes the reference point for a new one-step-ahead forecast, and so on.

When modelling time series, the following steps were followed:

i. Pre-processing of time series: (a) trend and periodicity removal, and (b) Target transformation; and

ii. Model-building

These two steps are discussed in detail below. The process stages are also presented in Figure 3.18.

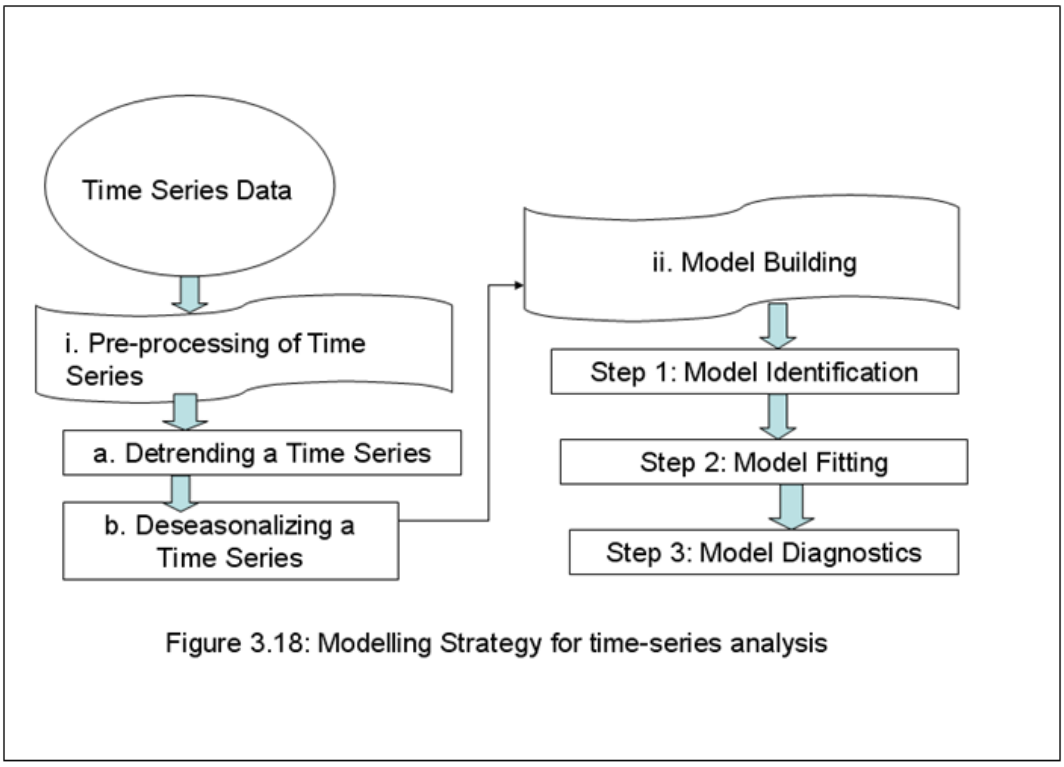

Figure 3.18: Modelling Strategy for time-series analysis

i. Pre-processing of Time Series

It is very difficult for any technique to forecast non-stationary time series values satisfactorily. Hence, the strategy is to somehow transform a non-stationary series into a

stationary one. We train the learning algorithms on the stationary data, instead of the original data. The predicted values are then transformed back into the original units.

The first, and most important, step in any time series is to plot the observations against time. The plot is vital. In general, we should be able to transform a non-stationary time series into a stationary one if the series exhibits certain patterns. Detrending and deseasonalizing a time series will make it stationary.

a. Detrending a time series

In trend stationary time series, the mean and variance functions are constant over time. Plotting the time series enables the analyst to determine whether the mean and variance are changing over time. If the graph moves upward/downward throughout, then the mean is changing over time, due to long term trends (Figure 3.19). The occurrence of oscillations with increasing amplitudes indicates unstable variance (Figure 3.20). This happens when the periodicity interacts with the trend. If the time series is not trend stationary, we can often transform it to trend stationarity with one of the following ways depending on the type of nonstationarity:

- *Nonstationary data with an unstable mean and stable variance over time:* Such processes (Figure 3.19) can be reduced to trend stationarity by applying non-seasonal differencing of order d .

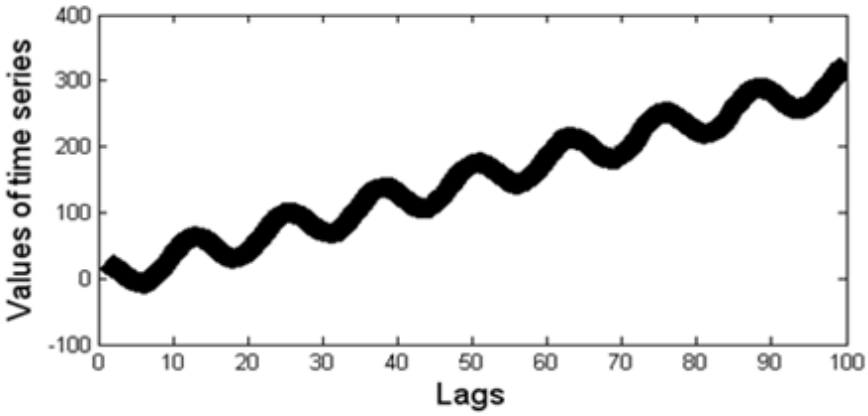

Figure 3.19: Time series with a changing mean and a constant variance

• *Unstable mean and unstable variance:* If the time series has unstable mean and variance with time (Figure 3.20), before differencing, the logarithmic transformation was applied.

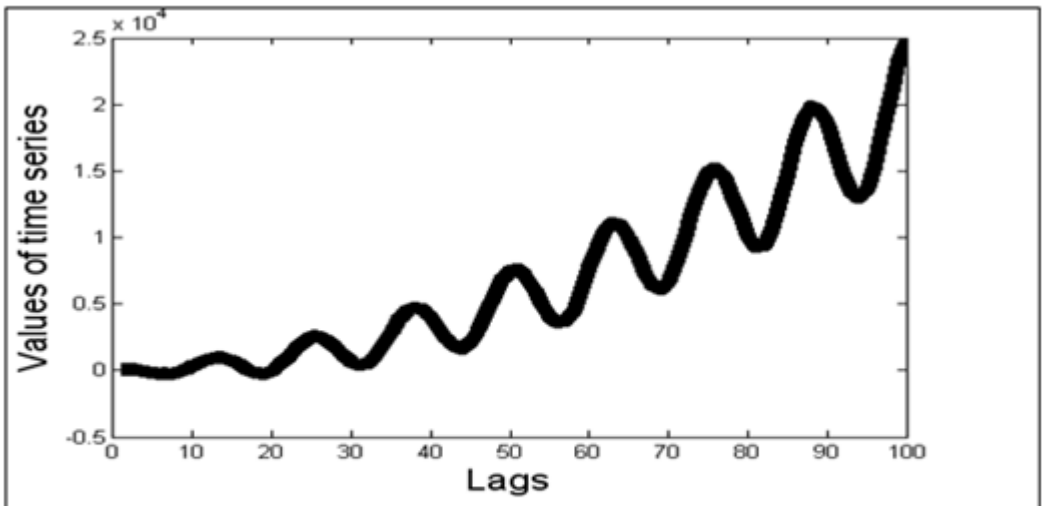

Figure 3.20: Time series with a changing mean and a changing variance

The procedures of carrying out non-seasonal and seasonal differencing on a time series are briefly described below.

Non-seasonal Differencing is a popular and effective way of removing trend from a time series. The strategy is to apply successive differencing until the differenced series become stationary. If the series has positive autocorrelations out to a high number of lags (figure 3.21), then it probably needs a higher order of differencing. The correlation of a variable with itself over successive time intervals is called autocorrelation (ACF) or serial correlation.

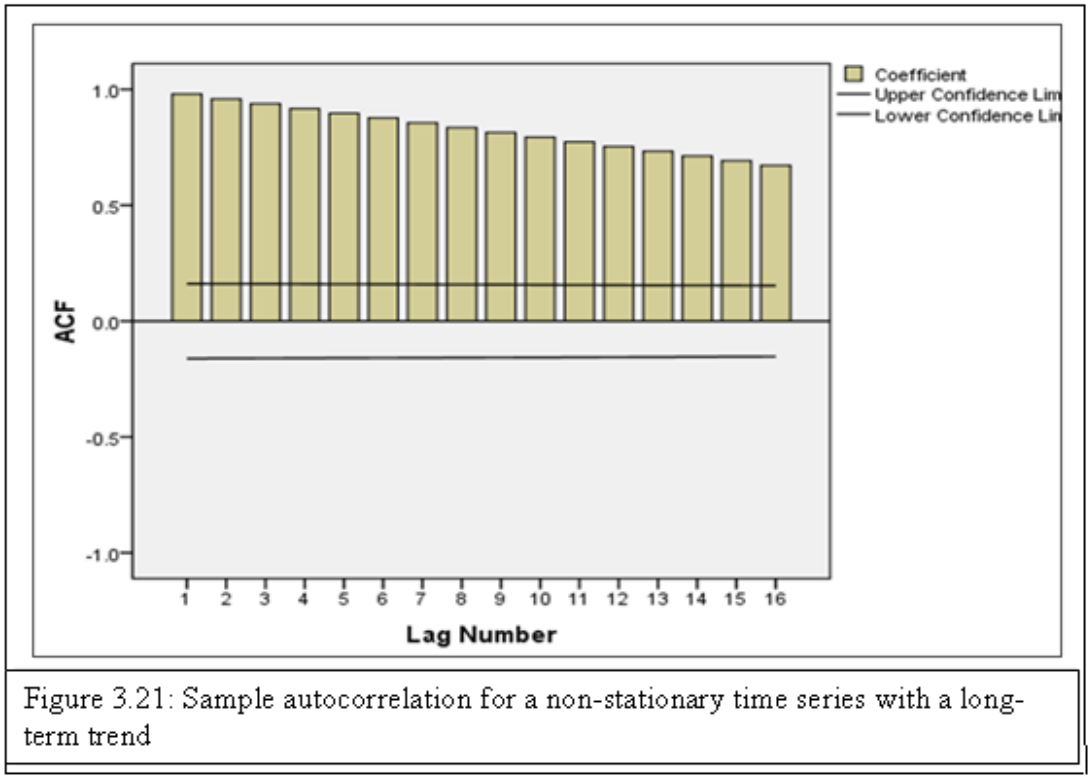

Figure 3.21: Sample autocorrelation for a non-stationary time series with a long-term trend

First order differencing is usually sufficient to obtain trend stationarity. However, higher-order differencing was applied when necessary. The new series $X_1, X_2, \cdots, X_{N-1}$ is formed from the original series $Z_1, Z_2, \cdots, Z_N$ by first-order differencing:

$$Z_t = Z_{t+1} - Z_t = \nabla X_{t+1} \qquad (3.31)$$

Occasionally second-order differencing is required using the operator ∇^2, where

$$\nabla^2 X_{t+2} = \nabla X_{t+2} - \nabla X_{t+1} = X_{t+2} - 2X_{t+1} + X_t \qquad (3.32)$$

Differencing tends to introduce negative correlation. If the series initially shows strong positive autocorrelation, then a non-seasonal difference will reduce the autocorrelation. A detrended time series looks flat (Figure 3.22).

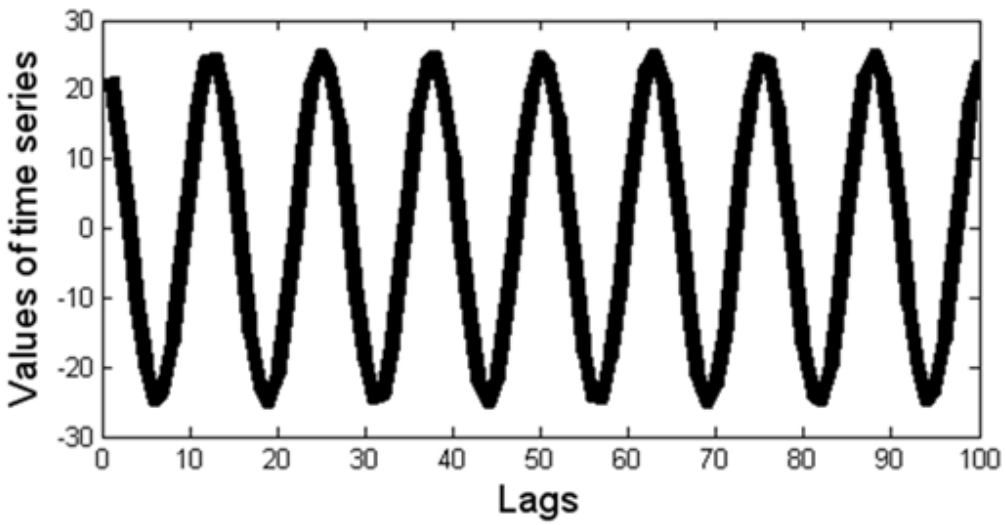

Figure 3.22: De-trended time series

b. <u>Deseasonalizing a time series</u>

If there are recurring patterns in the trend stationary time series (Figure 3.22), some form of ***seasonal differencing*** was considered to make the data stationary. The autocorrelation of a periodic function is itself, periodic with the very same period. We plot the detrended data (flat time series) and examine the autocorrelation function of the data to see if there exist significant autocorrelations at the seasonal lags s where, $s > 1$ (for example, $s = 12,24,36, and\ 48$).

If there appear fairly significant autocorrelations at the seasonal lags, we need to assume that seasonality is playing a significant role in determining the variation in this data (Figure 3.23). In general, when we have "flat" time series data we can simply plot the sample ACF of the data and see if there are "spikes" in it and possibly around the seasonal lags of $s, 2s, 3s, \cdots 4s$ etc. If there are, then more likely the data has seasonality in it and some form of seasonal differencing was considered to make the data stationary. We perform the first order seasonal span differencing operation as follows:

$$\nabla_s X_t = X_t - X_{t-s} \qquad (3.33)$$

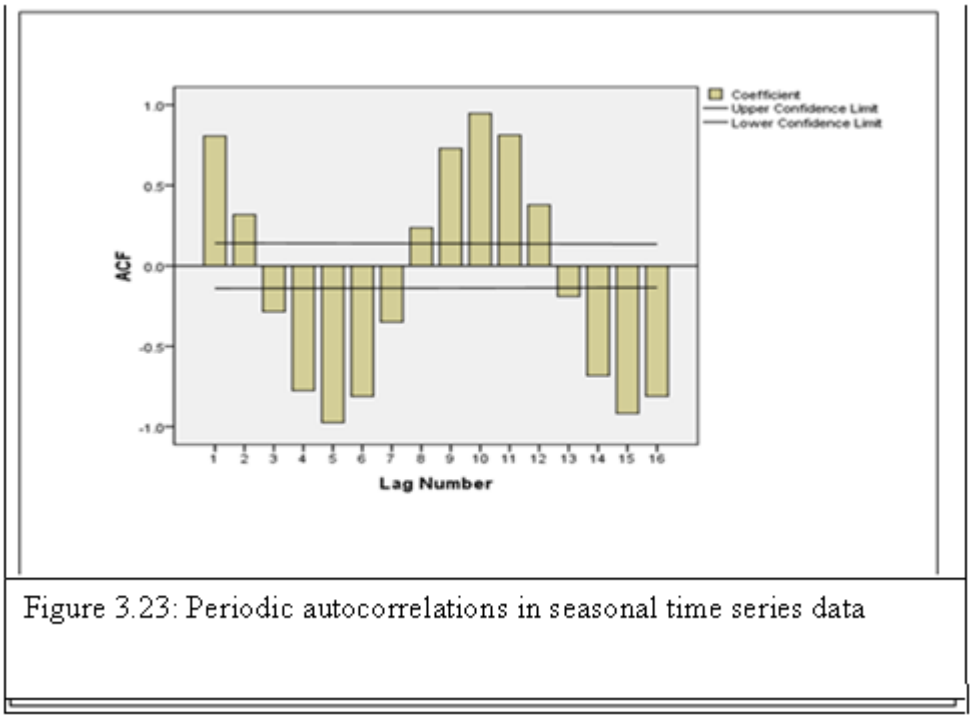

Figure 3.23: Periodic autocorrelations in seasonal time series data

If the lag-1 autocorrelation is zero or even negative, or the autocorrelations are all small and pattern less (Figure 3.24) then the series does not need further differencing.

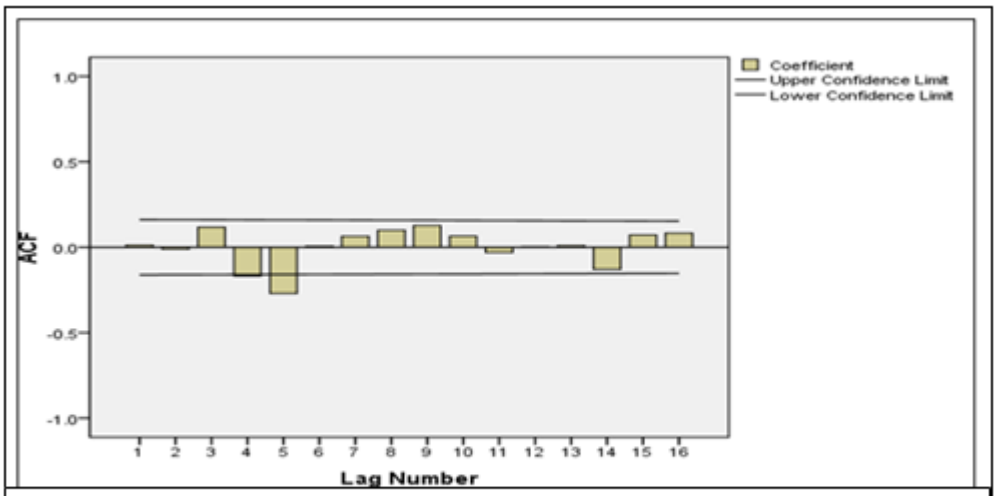

Figure 3.24: Sample autocorrelations of a stationary time series

The common wisdom is that "overdifferencing" should be avoided since overdifferencing can introduce patterns into the original observations which were not actually in the data before the differencing. If the lag-1 autocorrelation is more negative than -0.5 (and theoretically negative lag-1 autocorrelation should never be greater than 0.5 in magnitude) this may mean the series has been overdifferenced (Figure 3.25). Another symptom of possible overdifferencing is an increase in the standard deviation, rather than a reduction, when the order of differencing is increased. The optimal order of differencing is often the order of differencing at which the standard deviation is lowest. The time series plot of an over-differenced series may look quite random at first glance, but if we look closer we will see a pattern of excessive changes in sign.

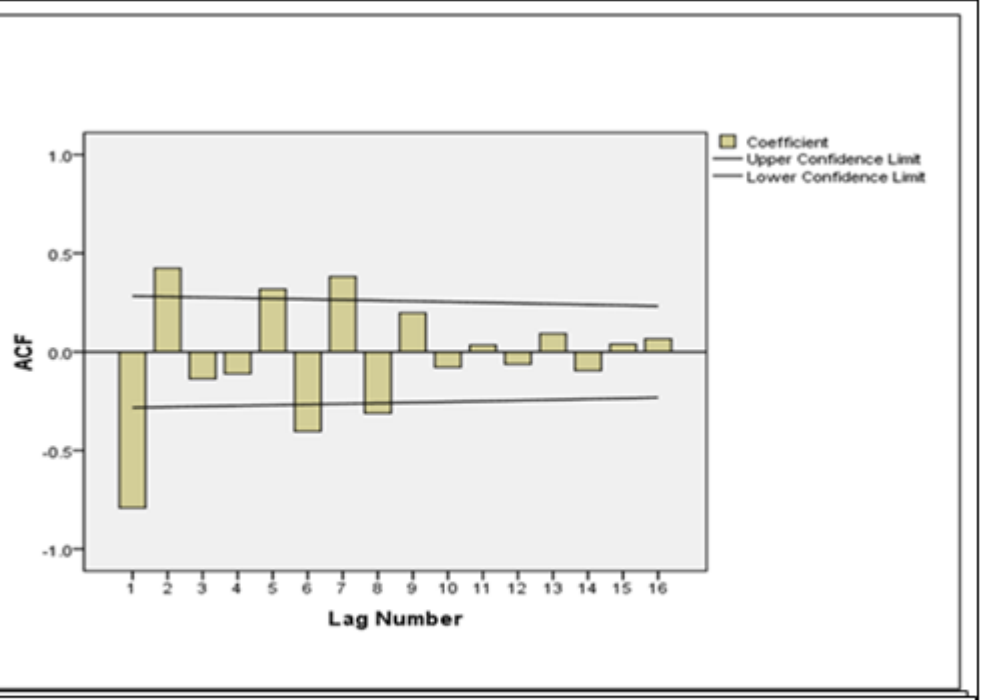

Figure 3.25: Autocorrelations of an over-differenced series

ii. Model Building Method

It is useful to normalize the time series values to the interval [0, 1]. This helps speed up the algorithm convergence. There are three main steps in the model-building procedure.

Step 1: Model Identification: For the identification of ARIMA and GARCH models, we look at the time plot of the series, compute many different statistics (e.g. ACF and PACF) from the data, and also apply knowledge from the subject area in which the data arise (such economics, physics, chemistry or biology) and choose a tentative model. For the ARIMA and GARCH models, we considered both autoregressive (lagged values of time series) and moving average terms (lagged error terms of time-series) as potential input variables, whereas for neural networks, only autoregressive terms were considered as potential input

variables. Neural Networks (all ensemble and neural network classifiers) learn two models, one for the future values of the time series and one for future volatilities. First, a model is fitted to the actual time series data to forecast the future value, and the in-sample squared residuals of the model are obtained. Next, a second model is fitted to the squared residuals to forecast future volatilities on time series data. For neural network based algorithms, we applied our proposed feature subset selection algorithm SAGA (presented in chapter 4 of this thesis) for selecting an optimal subset of predictors (i.e. input nodes) from all candidate predictors (with at least 10 observations). All the other model parameters (e.g. number of nodes in the hidden layer) were initially assigned to the lowest possible values. We then gradually increase the values of these parameters until the performance starts to deteriorate (please see section 3.1.4).

Step 2: Model Fitting: Model fitting consists of finding the best possible estimates of parameter coefficients for tentative ARIMA and GARCH models. On the other hand, the training process of neural networks involves adjusting the connection weights of the given model.

Step 3: Model Diagnostics: Model diagnostics is concerned with analyzing the quality of the model that we have specified and estimated. If no inadequacies are found, the modelling is assumed to be complete, and the model is used to forecast future time series values. Otherwise, we return to model identification in order to choose another model in light of inadequacies found. In this way, we cycle through three steps until, ideally, an acceptable model is found.

In this study we used the following benchmark time series forecasting algorithms:

- ARIMA-GARCH methodology: discussed in section 3.1.3.1
- Elman's Recurrent Neural Networks: discussed in section 3.1.3.2
- A hybrid algorithm of ARIMA-GARCH and ERNN: discussed in section 3.1.3.3.
- Feedforward Neural Networks (MLP, RBFN, and GRNN): discussed in section 3.1.4.
- Neural Network Ensembles: We evaluated the following four popular neural network ensembles: (1) HES (Heterogeneous ensemble with simple averaging), (2) HEW (Heterogeneous ensemble with weighted averaging), (3) HOS (Homogeneous ensemble with simple averaging), (4) HOW (Homogeneous ensemble with weighted averaging). Among these algorithms, the heterogeneous ensembles (HES and HEW) consist of four members (a GRNN, an ERNN, a MLP and a RBF), whereas the homogeneous ensembles (HOS, and HOW) consist of several GRNNs. The homogeneous and heterogeneous ensembles are discussed in section 3.1.5.

3.1.3.1 ARIMA –GARCH Methodology:

The ARIMA models are used to the purposes of forecast for time series data. GARCH models provide a varying estimate for the volatility of the series for every point.

Box-Jenkins ARIMA model: The ARIMA methodology has been collectively developed by many researchers. ARIMA models are commonly called Box-Jenkins models after the US mathematicians George Box and Gwilym Jenkins who popularized them in the mid 1970s [142].

ARIMA is an acronym for autoregressive integrated moving average. Autoregressive and moving average refer to two of the components of the model.

A non-stationary time series Z_t is said to follow an ARIMA (p, d, q) model if the *d*'th difference is a stationary process. In order to use the ARIMA model, we have to identify the values of *p* (the number of the autoregressive terms), *d* (the number of differences) and *q* (the number of the lagged forecast errors or moving average terms in the prediction equation). A simple ARIMA (p, 1, q) model is defined by:

$$W_t = \phi_1 W_{t-1} + \phi_2 W_{t-2} + \cdots + \phi_p W_{t-p} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} \qquad (3.34)$$

Or, in terms of the observed series

$$Z_t - Z_{t-1} = \phi_1 (Z_{t-1} - Z_{t-2}) + \phi_2 (Z_{t-2} - Z_{t-3}) + \ldots + \phi_p (Z_{t-p} - Z_{t-p-1}) + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} \qquad (3.35)$$

Which we may rewrite as

$$Z_t = (+\phi_1) Z_{t-1} + (\phi_2 - \phi_1) Z_{t-2} + (\phi_3 - \phi_2) Z_{t-3} + \cdots + (\phi_p - \phi_{p-1}) Z_{t-p} - \phi_p Z_{t-p-1} + a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - \cdots - \theta_q a_{t-q} \qquad (3.36)$$

Therefore, the output is a regression on a number of the most recent output values and several immediately previous error terms.

Z_t = observation at time *t*

a_t = shock, innovation or error at time *t*.

a_t, $\theta_1 a_{t-1}$, $\theta_2 a_{t-2}, \ldots, \theta_q a_{t-q}$ are the moving average terms.

$W_{t-p},...,W_{t-1},W_t$ are the first-order difference series.

$(+\phi_1 Z_{t-1}$, $(\phi_2 - \phi_1 Z_{t-2}$, $\phi_2(Z_{t-2} - Z_{t-3})$, …, $(\phi_p - \phi_{p-1} Z_{t-p}$, $\phi_p Z_{t-p-1}$ are the autoregressive terms.

θ = moving average coefficients

ϕ = autoregressive coefficients

If $d = 0$, the ARIMA model is an ARMA*(p,q)*. Further derivations can also take into account periodicity by considering autoregressive or moving average trends that occur at certain point in time. In case of periodicity the ARIMA model is expressed as ARIMA *(p,d,q)(P,Q)* where P is the number of seasonal autoregressive lags and Q is the number of seasonal moving average lags.

GARCH (Generalized Auto-Regressive Conditional Heteroskedasticity) Models: Bollerslev (1986) first proposed GARCH models to generate volatility forecasts [143]. GARCH is a variance model. GARCH is used to isolate the predictable components of uncertainty (or volatility). GARCH models specify current conditional variance as a function of past conditional variances and past squared residuals. GARCH models are used to study and forecast the conditional variances.

The usual approach to GARCH (p, q) models is to model an error term ε_t in terms of a standard white noise $e_t \sim N(0,1)$ as $\varepsilon_t = \sqrt{h_t} e_t$ (3.37)

$$h_t = \omega + \sum_{i=1}^{q} \alpha_i (\varepsilon_{t-i}^2) + \sum_{j=1}^{p} \beta_j (h_{t-j}) \quad (3.38)$$

Here, p represents the number of lagged squared residuals (autoregressive lags) and q denotes the number of considered lagged variance values (moving average lags).

Maximum likelihood method (MLE) is used to estimate the parameters of the model.

3.1.3.2 Elman's Recurrent Neural Network (ERNN)

The Elman's recurrent neural network (ERNN) is first proposed by Jeff Elman in 1990 [144]. This is a variation on the multilayer perceptron. The difference is that an ERNN contains recurrent connections from the hidden nodes to a layer of context units consisting of unit-time delays. These context units store the outputs of the hidden nodes for one time-step. The training of the recurrent networks requires the update of recurrent weights (i.e. the connection weights between hidden nodes and their corresponding context units). Except for this portion, training is similar to MLP networks (described in section 3.1.4.2).

Architecture of Elman Networks

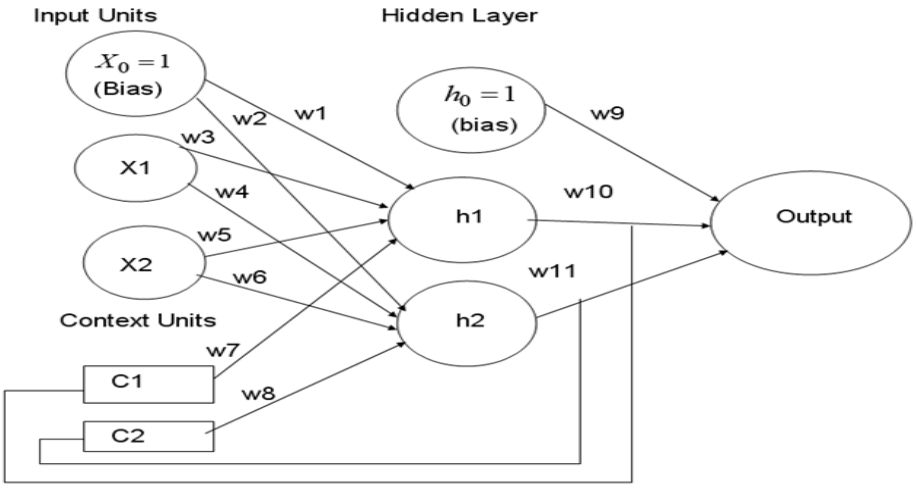

Figure 3.26: Elman Recurrent Neural Networks

The ERNN uses only autoregressive terms without moving average terms. It consists of an input layer, one or more hidden layers and an output layer. The network is fully connected. Each node in a given layer has a weight connecting it to every node in the next layer. The first layer is the input layer. The input layer consists of one or more input nodes that distribute the input to the hidden layer nodes. The intermediate (hidden) layer(s) consists of a number of hidden nodes each of which computes a non-linear transformation. In our study, in order to reduce the risk of overfitting, we fixed the number of hidden layers to 1. The third (output) layer contains as many nodes as output variables. Each output node contains a linear or nonlinear transfer function. In our study, the logistic sigmoid (logsig) was used as activation for the hidden nodes of the network, whereas the hyperbolic tangent sigmoid function (tanh) was used for the output node of the network. Both input and hidden layers have an additional bias node, with 1 as their output value. The bias nodes are also connected to all of the nodes in the subsequent layer. During training, the connection weights are iteratively adjusted so as to minimize the error function using the back-propagation learning algorithm. There is one context unit for each hidden unit, so an Elman network includes as many context units as hidden nodes. Each context unit stores the previous output value of the corresponding hidden node. This recurrent connection gives the network an exponential "memory" of past events. This "memory" makes the Elman network very effective in learning temporal patterns. Each hidden unit receives an input signal from the input units and the corresponding context unit.

Training of the Elman network: consists of the following steps:

Step 1: Initialize connections with random weights

Step 2: Present the first instance of the training set to the input layer.

Step 3: Calculate the network output

Step 4: Compare the predicted output with the actual output.

Step 5: Back-propagate the error by adjusting weights of the hidden and output layer.

Step 6: Copy the hidden nodes to the context nodes.

Step 7: Repeat steps 2-6, this time by presenting the next instance in the sequence until the end of the sequence is reached.

Step 7: Repeat steps 2-7 until the training error is sufficiently small.

Steps of the Forecasting Process using Elman networks:

Step 1: Put inputs for time t to the input units.

Step 2: Compute hidden unit outputs at time step t using net input from input units and from context units.

Step 3: Compute the final output of the network for time t as usual.

Step 4: Copy the outputs in the hidden units at time step t into the context units.

3.1.4.3.1 Hybrid Algorithm of ARIMA-GARCH & ERNN (HA)

This hybrid algorithm assumes that the residuals of the linear model (such as an ARIMA model) will contain non-linear patterns, which a non-linear net (model), such as a neural network, should be able to model. First we model the linear part of time series by fitting an ARIMA model to the time series. Then the residuals of ARIMA model are modelled using ERNN. Thereafter, we model the linear part of the future volatility by fitting a GARCH model to the squared residual series from the ERNN model. Finally, the residual series from the GARCH model were modelled using ERNN in order to model the nonlinear part of future volatility.

Therefore, the combined forecasts are:

The predicted future value $\hat{Z}(t)$ of time-series at time t :

$$\hat{Z}(t) = \hat{L}(t) + \hat{N}(t) \qquad (3.39)$$

And

The predicted future volatility of time series at time t:

$$\hat{V}(t) = \hat{V}_G(t) + \hat{V}_N(t) \qquad (3.40)$$

Where, $\hat{Z}(t)$= the predicted time series value at time t forecasted by the hybrid algorithm HA; $\hat{L}(t)$= the predicted time series value at time t forecasted by the ARIMA; $\hat{N}(t)$= the predicted time series value at time t forecasted by the ERNN; $\hat{V}(t)$ =the predicted future volatility at time t forecasted by the HA, $\hat{V}_G(t)$=the predicted future volatility at time t forecasted by the GARCH; $\hat{V}_N(t)$=the predicted future volatility at time t forecasted by the ERNN. ARIMA and GARCH models include both autoregressive and moving average terms, whereas the ERNN contains only autoregressive parameters. Figure 3.27 shows a block diagram of the hybrid model.

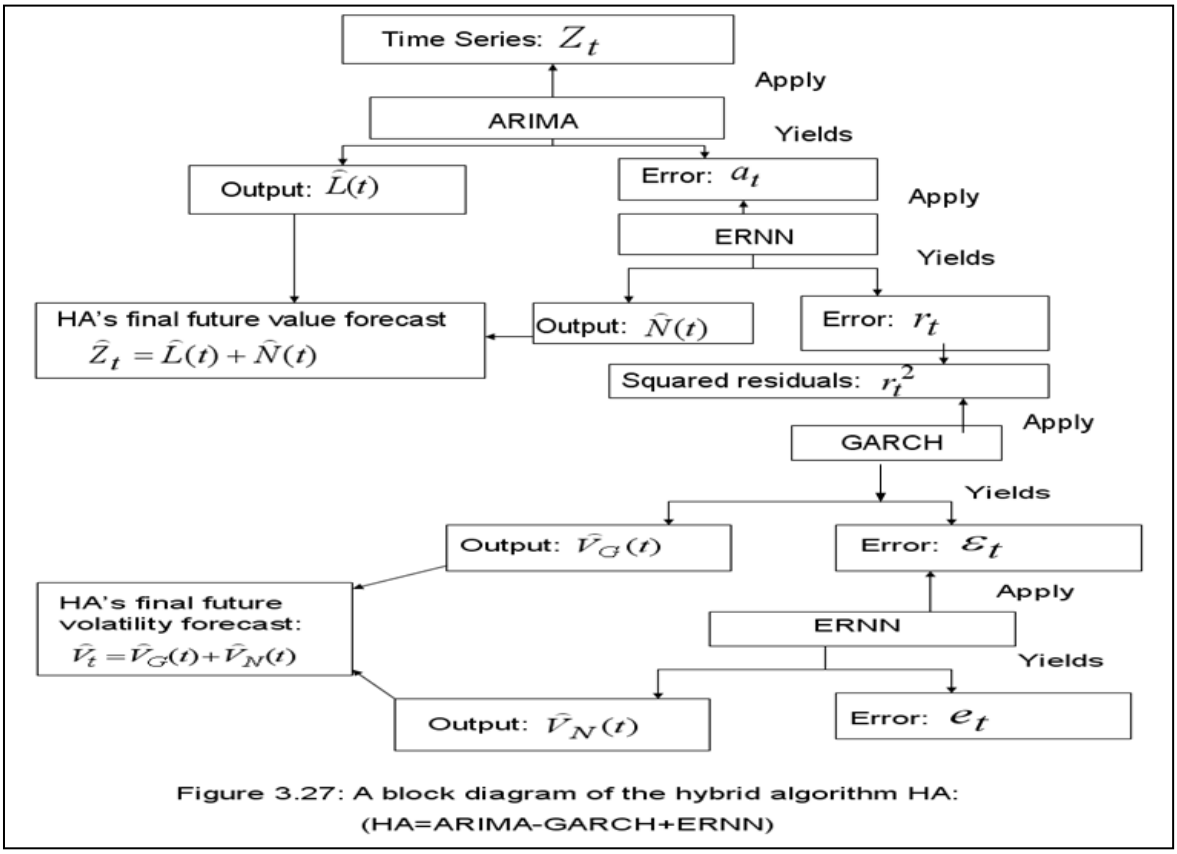

Figure 3.27: A block diagram of the hybrid algorithm HA: (HA=ARIMA-GARCH+ERNN)

3.1.4 Single Feed-forward Neural Networks

The variables are normalized to be within the range [0, 1]. In this section, we describe the four popular feed-forward neural networks.

3.1.4.1. Generalized Regression Neural Networks (GRNN)

GRNN is a simple, yet very powerful instance-based learning algorithm. In GRNN (Specht, 1991) each observation in the training set forms its own cluster [145].

GRNN is an instance-based algorithm. In GRNN each observation in the training set forms its own cluster. When a new input pattern $x = (x_1, \cdots, x_n)$ is presented to the GRNN for the prediction of the output value, each training pattern (prototype pattern)

$y_i = (y_{i1}, \cdots, y_{in})$ assigns a membership value h_i to x based on the Euclidean distance d, where

$$d = d(x, y_i) = \sqrt{\sum_{j=1}^{n} (x_j - y_{ij})^2} \qquad (3.41)$$

and

$$h_i = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{d^2}{2\sigma^2} \right) \qquad (3.42)$$

n is the total number of features in the study. x_j is the value of the *j*-th feature of the presented pattern (features can be multivalued or not). y_{ij} is the value of the *j*-th feature of the *i*-th prototype pattern and σ is the smoothing function parameter. We found that the performance of GRNN is not very sensitive to the exact setting of the parameter (σ). We arbitrarily set each centre's width to 2 times the average distance to 10 nearest neighbours.

Finally, GRNN calculates the output value *z* of the pattern *x* as in equation (3.43). The predicted output of the GRNN for the pattern x is the weighted average of the outputs of all prototype patterns. GRNN can handle continuous output variables and categorical output variables with two categories: event of interest (coded as '1') or not (coded as '0'):

$$z = \frac{\sum_i (h_i \times \textbf{output of } \mathbf{y}_i)}{\sum_i h_i} \qquad (3.43)$$

If the output variable is binary, then GRNN calculates the probability of event of interest. If the output variable is continuous, then it estimates the value of the variable.

3.1.4.2. Multilayer Perceptrons (MLP)

The MLP was developed by David Rumelhart, Geoffrey Hinton and Ronald Williams in 1986 [146]. The MLP is perhaps the most widely used neural network model.

Architecture of MLP networks

A typical MLP consists of three or more layers: one input layer, one output layer and one or several hidden layers (Figure 3.28).

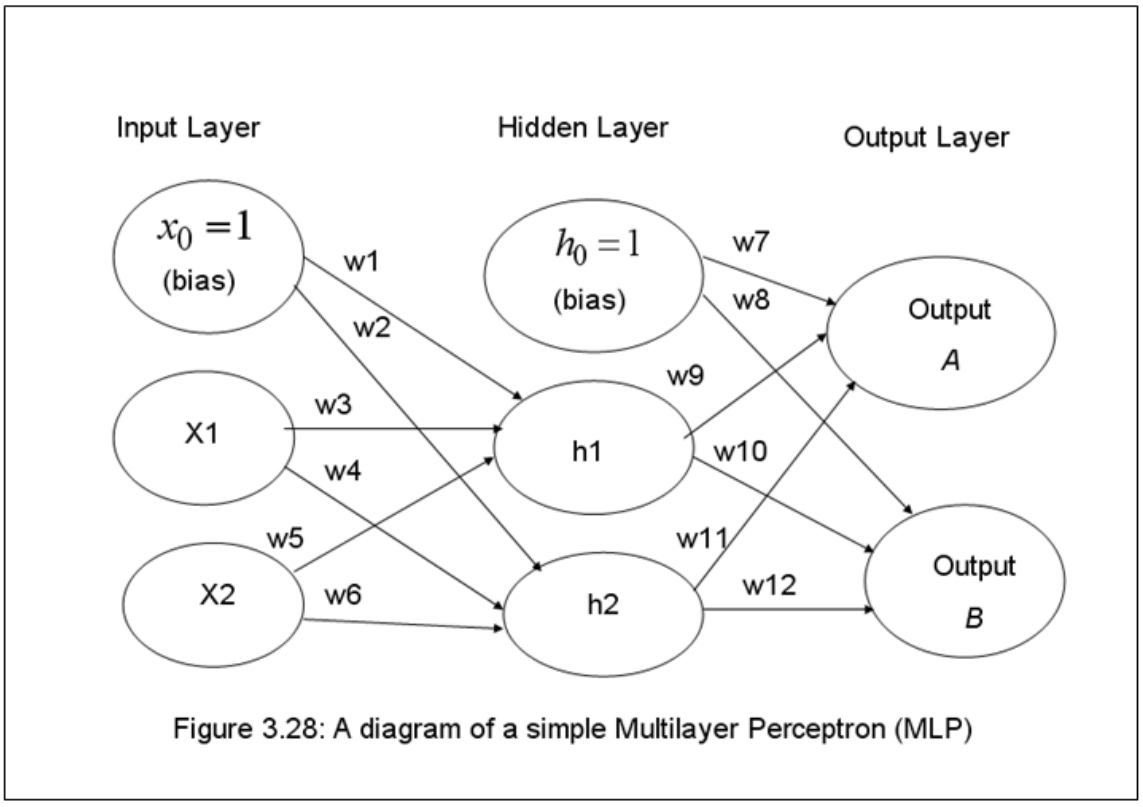

Figure 3.28: A diagram of a simple Multilayer Perceptron (MLP)

Input layer: The input layer includes a set of nodes (units); one for each input variable. There is an extra node called a bias node. The output from this node is always 1. The input

layer nodes are to only pass the inputs and perform no computation. Each input node (including the bias node) is connected to each hidden node by a connection weight.

Hidden layer: A hidden layer includes *N* hidden nodes plus a bias node. The bias node supplies a constant input 1 to the output node. No computation is done by the bias node. Each hidden node (including the bias node) is connected to the output node by a connection weight *w*. Every hidden node (except the bias node) contains a nonlinear transfer function which computes the node's output *y* based on its input *x*. The sigmoidal functions such as logistic sigmoid (logsig) and hyperbolic tangent sigmoid function (tanh) are the most common choices.

logsig function: $$y = \frac{1}{(1 + e^{-z})} \quad (3.44) \quad where, \quad z = \sum_{j=0}^{N} (w_j x_j)$$

tanh function: $$y = \frac{(e^{2z} + 1)}{(e^{2z} - 1)} \quad (3.45) \quad where, \quad z = \sum_{j=0}^{N} (w_j x_j)$$

$x_j = jth\ input\ of\ the\ node;\ and\ w_j = connection\ weight\ between\ these\ two\ nodes$

Output layer: The number of nodes in the output layer is equal to the number of output variables. Each node contains a transfer function to determine the output. For the output units, a transfer function suited to the distribution of target values should be chosen:

- For binary targets, the logistic function is an excellent choice.
- For continuous-valued targets with a bounded range, the logsig and tanh functions can be used. If it is desirable to constrain the outputs of a network to the range from 0 to 1 then the

output layer should use logsig function. On the other hand, if the output variable ought to be bounded within [-1, +1], the output layer should use the tanh function.

- For continuous-valued targets with no known bound, the output layer should use the linear transfer function.

Training of the MLP

The backpropagation learning algorithm (gradient descent algorithm) is applied on MLP to find the appropriate connection weights *(w)*. Training MLP with backpropagation learning algorithm is described below.

Step 1: Specify model parameters.

- *Number of input nodes (input variables)*: The number of nodes in the input layer is obtained by feature selection algorithms.
- *Number of hidden layers*: Empirical studies suggest that generally for most applications one hidden layer is sufficient [120]. The new findings also suggest that the risk of overfitting increases with the number of hidden layers. After considering the pros and cons, we chose one hidden layer for our MLP models [122].
- *Number of hidden nodes*: The hidden layer contains several hidden nodes (i.e. processing units) and a bias node. Too few or too many hidden nodes can cause the MLP to underfit or overfit during training. In most situations, there is no way to determine the best number of hidden nodes except through trial, error and observation. We determine the number of hidden nodes with the trial and error procedure.
- *Types of Transfer functions:* A transfer function ensures that the values in a network remain within a reasonable range. The transfer functions in different layers may or may not be identical, but the transfer functions for all nodes in the same layer should be identical, so

that all input values for a hidden or output node are within the same range. We scaled data in the range of 0 to 1. Hence, the forecasted values are assumed to be bounded in the range of 0 to 1. To satisfy this requirement, the logistic-sigmoid function was chosen as the activation function for the output node. The main purpose of the hidden node transfer functions is to introduce nonlinearity into the network. Common hidden node transfer functions are logistic function, tangent hyperbolic function (tanh), and Gaussian function. We tried each of these three functions individually during our tests and in all cases the performance of the network was pretty much the same. In this study, tanh is selected as the activation function of hidden nodes.

Step 2: Initially, we construct one hundred candidate MLP nets with one hidden node. The optimization of parameters (i.e. the connection weights) for MLP is in general a very difficult problem since the parameter space may have many local minima and other stationary points. In order to skip the local minima, one hundred candidate neural networks were built. For each net, the connection weights were randomly initialized in the interval of [-1, 1].

Step 3: We train all these hundred candidate networks simultaneously.

Training patterns are passed one at a time through a network. When an input signal is fed into a network, the back propagation learning algorithm does training in two passes (i.e. forward pass and reverse pass).

Part 1: Forward Pass

- In the forward pass, each hidden node *j* receives an input from every input node plus bias. The hidden nodes compute outputs as in equation (3.46):

$$h_j = \frac{(2y+1)}{(2y-1)} \qquad (3.46) \qquad where \quad y = \sum_{i=0}^{N} (w_{ji} x_i)$$

Similarly, the output node computes the network output as in equation (3.47)

$$out = \frac{1}{(1 + e^{-z})} \qquad (3.47) \qquad where \quad z = \sum_{j=0}^{N} (W_j h_j)$$

Where h_j is the output of hidden node *j*. Each node *j* receives input from every input node *i*. A weight w_{ji} is associated with each input node x_i. x_0 and w_{jo} are called the bias term ($x_0 = 1.0$) and the bias weights respectively. W_j represents the weight between the hidden node *j* and the output node. h_0 and W_0 are the bias term (h_0=1) and the bias weights respectively.

- Estimate the prediction error using the equation (3.48).

out_error=out(1-out)(Target-out) (3.48)

The "*out(1-out)*" term is necessary in the equation because of the sigmoid function –if we were only using a linear transfer function it would just be *(Target-out).*

Part 2: Reverse Pass

- Adjust the connection weights between the hidden nodes and the output node using the equation (3.49).

Let Wn_j be the new (trained) weight and W_j be the old weight of the connection between the hidden node *j* and the output node. h_j represents the output of the node *j*.

$$Wn_j = W_j + \eta (out\_error \times h_j) \qquad (3.49)$$

The constant η (called the learning rate, and nominally equal to one) is put in to speed up or slow down the learning if required. The new weight replaces the old weight and W_j is set to Wn_j. We update all the weights between the hidden layer and the output layer in this way. We fixed the learning rate at 0.001 for all experiments.

- Calculate the error for each hidden node as in equation (3.50).

$$\delta_j = h_j(1-h_j)(out\_error \times W_j) \qquad (3.50)$$

Where, δ_j denotes the error for the hidden node *j* and W_j represents the connection weight between the hidden node *j* and the output node. h_j is the output of the *j*-th hidden node. Again, the factor "$h_j(1-h_j)$" is present because of the sigmoid squashing function.

- Adjusts the connection weight between the input node *i* and the hidden node *j* using equation (3.51).

$$wn_{ji} = w_{ji} + \eta \delta_j x_i \qquad (3.51)$$

Where η is the learning rate and we set the learning rate $= 0.001$. wn_{ji} and w_{ji} are new and old connection weights between the input node *i* and the hidden node *j* respectively. δ_j denotes the error for the hidden node *j*. x_i represents the input supplied by the input node *i*. We update all the connection weights between the input layer and the hidden layer. Replace the old weight by the new found weight.

$$w_{ji} = wn_{ji} \qquad (3.52)$$

Each pass through all of the training examples is called one epoch or iteration of training. The performance of a network is evaluated on the validation set after each iteration. As long as the network continues to improve on the validation set, training is continued. The network training is stopped when the error on the validation data set did not decrease for 10 consecutive epochs. Using the validation data set for the optimization of weights minimizes the risk of overfitting.

Step 4: Record the best performance on the test set.

Step 5: Expand the nets by adding an extra hidden node. For each net, the initial weights were randomly set within the range [-1, 1]. Keep repeating the whole process (from step 2 to the end) until the best performance is achieved. Stop the process when the error rate on the test set starts to deteriorate.

Prediction Process using MLP

The calculation of the final output values proceeds layer by layer. First, the input signals are applied to the first layer, and each neuron of the first layer calculates its output value. Next, these values are propagated to the next layer; and so forth, until the final layer is reached where the values produced are actually the output values of the net.

3.1.4.3. Radial Basis Function Neural Networks (RBFN)

RBFN was introduced into the neural network literature by Broomhead and Lowe in 1988 [147]. After the MLP networks, the RBFN is one of the most used neural network. Figure 3.29 illustrates a RBFN with inputs $x_1, \cdots, x_n$ and output $\hat{y}$. The RBFN consists of a three layered neural network with one input layer, one hidden layer with Gaussian Radial Basis Function units and one output layer. An RBFN is a fully connected network in which each node in one layer is connected in the forward direction to every node in the next layer

Input layer: The input layer is the first layer. The first layer is composed of input nodes whose number is equal to the dimension of the input vector. Each node represents an input variable. The input layer does nothing but pass on inputs to the next layer. There are no connection weights between the input layer and the hidden layer.

Hidden layer: is the middle layer. This layer consists of *N* hidden nodes and a bias node with constant output equal to 1.0. Each hidden node (except the bias node) represents a prototype vector. The optimum number of hidden nodes is based on a trial-and-error approach. When a new input vector is presented to the network, each RBFN node in the hidden layer assigns a membership value to the new input pattern as in equation (3.42).

Each node (including the bias node) in the hidden layer is connected to the output node by a weight. In the training step, weights are learned from the training data.

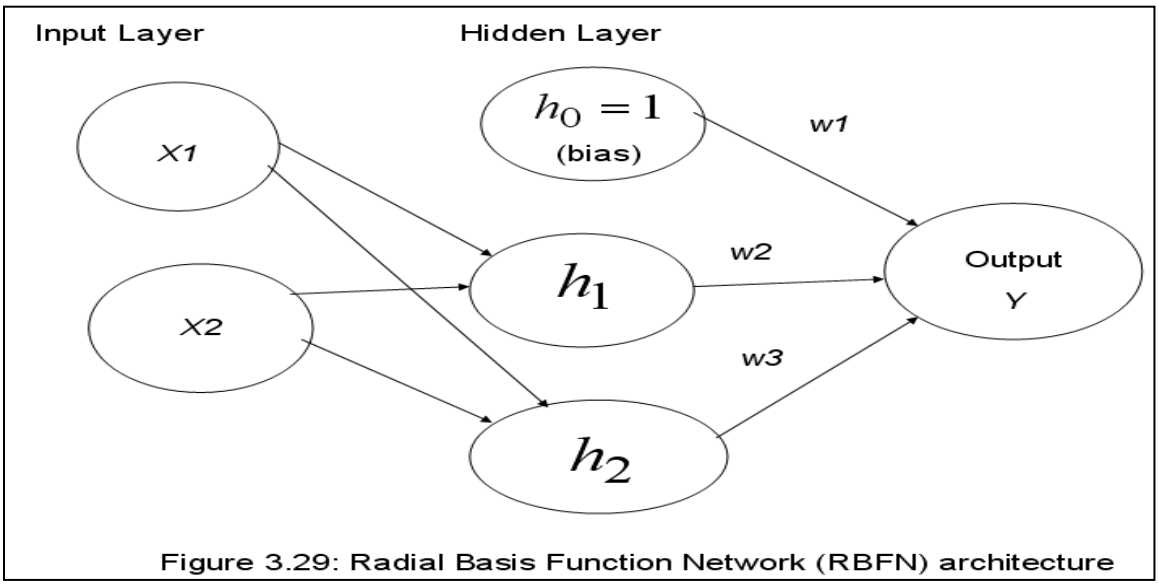

Figure 3.29: Radial Basis Function Network (RBFN) architecture

Output layer: The output layer consists of only one node, called the output node. The output node has a linear transfer function. The value of the output node is calculated by the following equation:

$$output = \sum_{j=0}^{N} w_j h_j \qquad (3.53)$$

Where w_j is the connection weight between the *j*-th hidden node and the output node. h_j is the output of the *j*-th hidden node. h_0 and w_0 are called the bias term $h_0 = 1$ and the bias weights respectively.

Training of the RBFN

Specify parameters for a RBFN: The optimum number of hidden nodes: are determined using a trial-and-error approach. Each hidden node corresponds to a prototype pattern. These prototype patterns are selected using *K*-means clustering (described in section 3.1.4.3.1). The widths of the radial basis functions are optimized using the real-valued Particle Swarm Optimization (PSO) algorithm (described in section 3.1.4.3.2), since the PSO was originally developed for real-valued spaces.

We adjust connection weights of the RBFN following the procedure below:

Step 1: Initially, we construct one hundred candidate RBFN nets by initializing the network weights with small random values. The initial weights range from -1 to +1.

Step 2: Present input patterns from the training set one at a time to the network and update the connection weights. Upon presenting an input pattern to the network, the hidden nodes compute their outputs as in equation (3.42). The output from each hidden node is passed to the output node.

Step 3: The output node determines the output of the whole networks using the equation (3.53).

Step 3: Estimate the prediction error using the equation (3.54).

$$\mathbf{error = Actual\ Output - PredictedOutput} \qquad \mathbf{(3.54)}$$

Step 4: Adjust the connection nodes between the hidden node *j* and the output node as in equation (3.55).

$$\textbf{Adjusted Weight} = \textbf{Old Weight} + \eta(\textbf{error} \times \textbf{h}_\textbf{j}) \qquad (3.55)$$

Where h_j denotes the output of the hidden node *j*. η denotes the learning rate and is constant. We set the learning rate $= 0.001$.

The weights are updated on a pattern-by-pattern basis until the entire training data set is completed which is called one "iteration" (or epoch). After each epoch, the networks are tested using the validation dataset. If the validation error remains the same for more than 10 successive epochs, it is assumed that the network has converged and the training is terminated. When validation error increases—indicating that the network is over fitting the training data – the training is stopped. Terminate the algorithm if the stopping criteria are satisfied; otherwise continue the iteration. After the training, all 100 trained networks are evaluated on test data and the best network is selected.

3.1.4.3.1 *K*-Means Clustering Algorithm

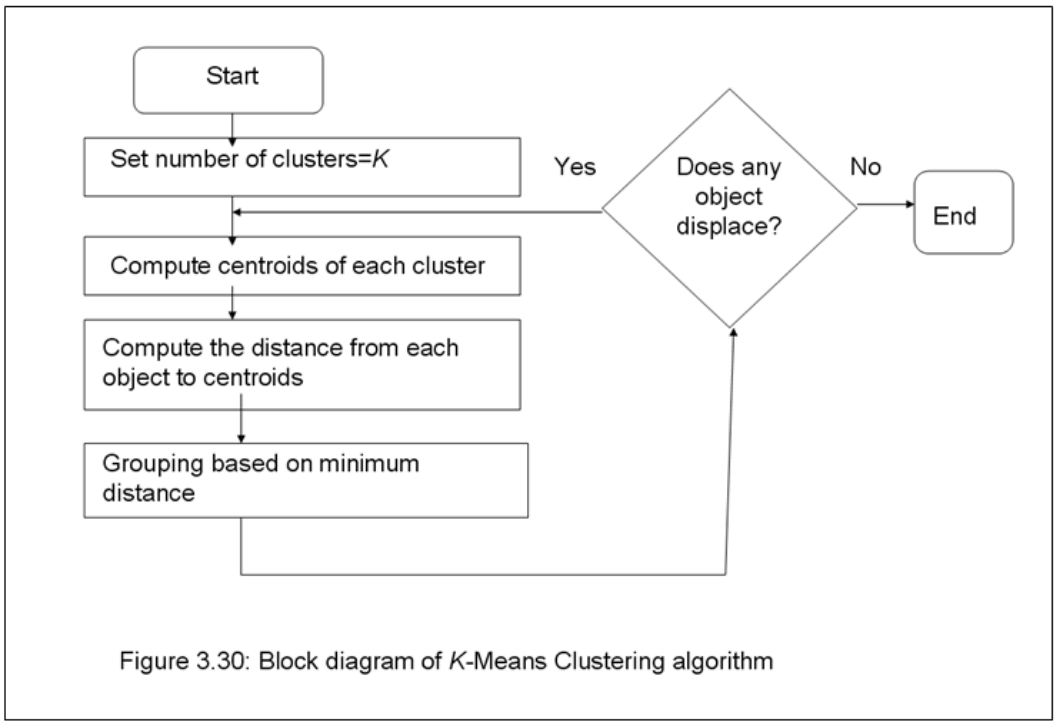

Figure 3.30: Block diagram of *K*-Means Clustering algorithm

Step 1: Begin with a decision on the value of *K*= number of clusters.

Step 2: Take the first *K* training input patterns as cluster centres (centroids). Assign each of the remaining (N-K) training input patterns to cluster with the nearest centroid. After each assignment, re-compute the centroid of the gaining cluster. The centroid is the input pattern with the minimum average Euclidean distance to all members in the cluster.

Step 3: Take each sample in sequence and compute its (Euclidean) distance from the centroid of each of the clusters. If a sample is not currently in the cluster with the closest centroid, switch this input pattern to that cluster and update the centroid of the cluster gaining the new input pattern and the cluster losing the input vector.

Step 4: Repeat step 3 until convergence is achieved, that is until a pass through all training patterns causes no new assignments.

3.1.4.3.2 Real-Valued Particle Swarm Optimization

A real-valued Particle Swarm Optimization (PSO) was applied to optimize the width (σ) of radial basis function around centre when using RBFN. Here we will give a short description of the real-valued PSO proposed by Kennedy and Eberhart [136].

Let us have a dataset of 4-dimensional input vectors (example table 3.1.4.3.2.1).

Example Table: 3.1.4.3.2.1: A hypothetical dataset

	Input Variables				Output variable
Instance	X1	X2	X3	X4	Y
1	5	4	2	0.1	20
2	10	13	4	0.7	15
3	8	17	7	1	16

First of all, we normalize each variable to [0, 1] (example table 3.1.4.3.2.2).

Example Table 3.1.4.3.2.2: Normalized dataset

	Input Variables				Output variable
Instance	X1	X2	X3	X4	Y
1	0.5	0.2	0.3	0.1	1.0
2	1.0	0.8	0.6	0.7	0.8
3	0.8	1.0	1.0	1.0	0.8

Pseudo code of real-valued PSO

Step 1: Specify parameters.

- Range of width factor (σ): Find the maximum possible distance (d_{max}) between two input patterns. The range of width (σ) of radial basis function: 0-d_{max}

Example

$$d_{\max} = \sqrt{(2-0)^2 + (2-0)^2 + (2-0)^2 + (2-0)^2} = \sqrt{4} = 2$$

Therefore, range of width (σ) of radial basis function: 0-2

• Population size: We fix the number of particles in a swarm to be 10.

• Position of each particle: Let the RBFN consists of *N* hidden nodes (radial basis functions). Therefore, our search space is *N*-dimensional, and the *i*'th particle of the swarm at time *t* can be represented by a *N*-dimensional position vector $X_i(t) = [X_{i1}(t), X_{i2}(t), \cdots, X_{iN}(t)]$ where each bit represents the width of a radial basis function and each bit is a real value in the interval [0, $d_{\max}$]. We then randomly initialize the position of each particle.

Example: Let, number of particle=4, number of hidden nodes =3, and $d_{\max}$=2.

Particle 1: [0.3, 0.2, 1.2], Particle 2: [0.5, 0.7, 0.5], Particle 3: [1.4, 0.1, 1.8], and

Particle 4: [2.0, 1.1, 0.4]

• Maximum and Minimum Velocities:

We set maximum velocity ($V_{\max}$) of any particle to be $(d_{\max})/3$ and minimum velocity ($V_{\min}$) to be 0.05.

• Velocity of each particle: The velocity of the *i*'th particle at time *t* is denoted by $V_i(t) = [V_{i1}(t), V_{i2}(t), \cdots, V_{id}(t)]$ where each bit ranges from 0.05 to $(d_{\max})/3$. Particle velocities are initially randomly determined.

Example: maximum velocity=2/3=0.7

Velocity of Particle 1 at time *t:* $V_1(t) = [0.2,0.4,0.1]$, Velocity of Particle 2 at time *t*: $V_2 = [0.7,0.2,0.5]$, Velocity of Particle 3 at time *t*: $V_3 = [0.3,0.6,0.4]$, Velocity of Particle 4 at time *t*: $V_4 = [0.5,0.5,0.5]$.

Step 2: Evaluate the fitness (prediction performance) *F* of each particle by training a RBFN using the particle's current position $X_i(t)$.

Step 3: Compare the performance of each individual to its best performance so far:

If $F(X_i) > F(P_{i,best})$

$F(P_{i,best}) = F(X_i)$, and $P_{i,best} = X_i$

Step 4: Compare the performance of each particle to the global best particle: if $F(X_i) > F(P_{gbest})$.

$F(P_{gbest}) = F(X_i)$, and $P_{gbest} = X_i$

Step 5: Change the velocity of each bit of the particle according to equation (3.56).

$$V_{ij}(t+1) = w.V_{ij}(t) + c_1 r_1 (P_{ij.best} - X_{ij}) + c_2 r_2 (P_{gbest} - X_{ij}) \quad (3.56)$$

where $w = 1 - \frac{T_{spent}}{T_{\max}}$

If $V_{ij}(t+1) > V_{\max}$, then $V_i(t+1) = V_{\max}$

If $V_{ij}(t+1) < V_{\min}$, then $V_i(t+1) = V_{\min}$

Where, $V_{ij}(t+1)$ and $V_{ij}(t)$ denote velocities of *j*'th bit for the *i*'th particle at time *(t+1)* and *t*, respectively. *w* is the inertia weight which shows the effect of previous velocity vector on the new vector. $T_{\max}$ denotes the total number of iterations and T_{spent} denotes the

number of iterations performed so far. r_1 and r_2 are two random numbers between (0, 1). c_1 and c_2 are two positive constants: $c_1 = c_2 = 2$.

Example: Let, Particle 1: $X_1 = [0.3, 0.2, 1.2]$; $X_{11} = 0.3$; $X_{12} = 0.2$; $X_{13} = 1.2$;

$F(X_1) = 0.55$; $P_{1,best} = [0.5, 0.3, 0.6]$; $P_{11,best} = 0.5$; $P_{12,best} = 0.3$; $P_{13,best} = 0.6$;

$F(P_{1,best}) = 0.63$; $P_{gbest} = [0.9, 0.1, 1.5]$; $P_{1,gbest} = 0.9$; $P_{2,gbest} = 0.1$; $P_{3,gbest} = 1.5$;

$F(P_{gbest}) = 0.74$; $V_1 = [0.4, 0.3, 0.1]$; $V_{11} = 0.4$; $V_{12} = 0.3$; $V_{13} = 0.1$; $T_{\max} = 100$,

$T_{spent} = 40$; $V_{\max} = 0.7$; $V_{\min} = 0.1$; $c_1 = c_2 = 2$; $w = 1 - \frac{T_{spent}}{T_{\max}} = 1 - \frac{40}{100} = 0.6$

$$V_{11}(t+1) = w * V_{11}(t) + c_1 r_1 (P_{11,best} - X_{11}(t)) + c_2 r_2 (P_{1,gbest} - X_{11})$$
$$= (0.6 * 0.4) + 2 * 0.4 * (0.5 - 0.3) + 2 * 0.9 * (0.9 - 0.3) = 0.24 + 0.16 + 1.08 = 1.48$$

Therefore, $V_{11}(t+1) = 0.7$ [since $V_{11}(t+1) > V_{\max}$]

$$V_{12}(t+1) = w * V_{12}(t) + c_1 r_1 (P_{12,best} - X_{12}(t)) + c_2 r_2 (P_{2,gbest} - X_{12})$$
$$= (0.6 * 0.3) + 2 * 0.2 * (0.3 - 0.2) + 2 * 0.6 * (0.1 - 0.2) = 0.18 + 0.04 - 0.12 = 0.1$$

$$V_{13}(t+1) = w * V_{13}(t) + c_1 r_1 (P_{13,best} - X_{13}(t)) + c_2 r_2 (P_{3,gbest} - X_{13})$$
$$= (0.6 * 0.1) + 2 * 0.5 * (0.6 - 1.2) + 2 * 0.9 * (1.5 - 1.2) = 0.06 - 0.6 + 0.54 = 0$$

Therefore, $V_{13}(t+1) = 0.1$ [since $V_{13}(t+1) < V_{\min}$]

Therefore, the velocity of particle 2 at time (t+1):

$V_1(t+1) = [V_{11}(t+1), V_{12}(t+1), V_{13}(t+1)] = [0.7, 0.1, 0.1]$

Step 6: Move each particle to the new position using equation (3.57)

$$X_{ij}(t+1) = X_{ij}(t) + V_{ij}(t+1) \qquad (3.57)$$

If $X_{ij}(t+1) > d_{\max}$; then $X_{ij} = d_{\max}$ [Here, $d_{\max}$ denotes the maximum possible width.]

If $X_{ij}(t+1) < 0$, then $X_{ij}(t+1) = 0$ [since width cannot be less than 0]

Example: Let, Particle 1: $X_1(t) = [X_{11}(t) = 0.3,\ X_{12}(t) = 0.2,\ X_{13}(t) = 1.2]$.

$V_1(t+1) = [V_{11}(t+1) = 0.7, V_{12}(t+1) = 0.1, V_{13}(t+1) = 0.1]$

$X_{11}(t+1) = X_{11}(t) + V_1(t+1) = 0.3 + 0.1 = 0.4$

$X_{12}(t+1) = X_{12}(t) + V_{12}(t+1) = 0.2 + 0.1 = 0.3$

$X_{13}(t+1) = X_{13}(t) + V_{13}(t+1) = 1.2 + 0.1 = 1.3$

Therefore, the final position of the particle 1 at time *(t+1)*:

$X_1(t+1) = [X_{11}(t+1), X_{12}(t+1), X_{13}(t+1)] = [0.4, 0.3, 1.3]$

Step 7: Go to step 2, and repeat until stopping criterion is met.

3.1.5 Ensemble Neural Networks

ANN ensemble consists of several individually trained ANN classifiers (base classifiers) that are jointly used to solve a problem. In our experiments, two types of ensembles—homogeneous and heterogeneous ensembles are constructed. A heterogeneous ensemble is a collection of different neural networks (ERNN, GRNN, MLP and RBFN) that together "vote" on a given example. All the individual networks in the heterogeneous ensemble are trained on the same training data, with the same predictors. In contrast, a homogeneous ensemble is a collection of the same kind of neural networks trained using different feature subsets as opposed to only one feature subset used in a heterogeneous ensemble model. For generating homogeneous ensemble models, we used multiple GRNNs each trained using different feature subsets. Each ensemble member tries to predict the response variable. Homogeneous and heterogeneous ensembles of neural networks are presented in Figures 3.31-3.33. The base classifiers of heterogeneous ensemble models constructed for the

substitution of missing values include: GRNN, MLP, and RBFN (Figure 3.32). On the other hand, the base classifiers of heterogeneous ensemble models constructed for time series forecasting include: ERNN, GRNN, MLP, and RBFN (Figure 3.33).

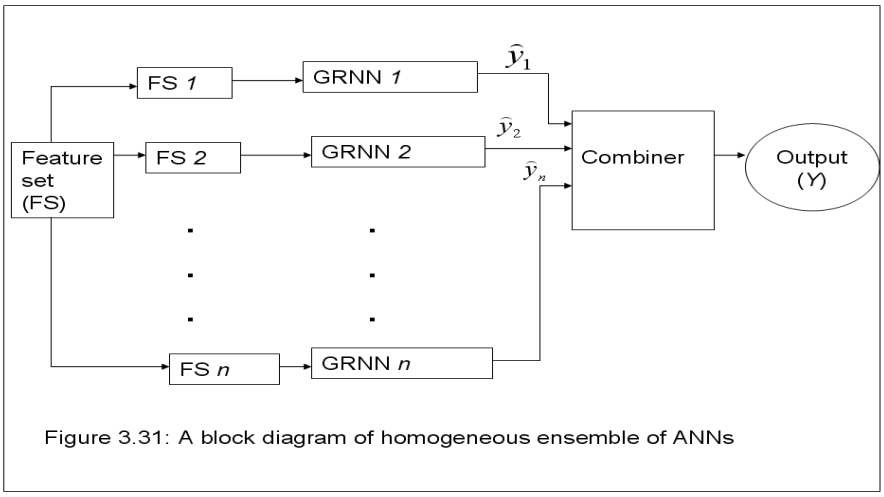

Figure 3.31: A block diagram of homogeneous ensemble of ANNs

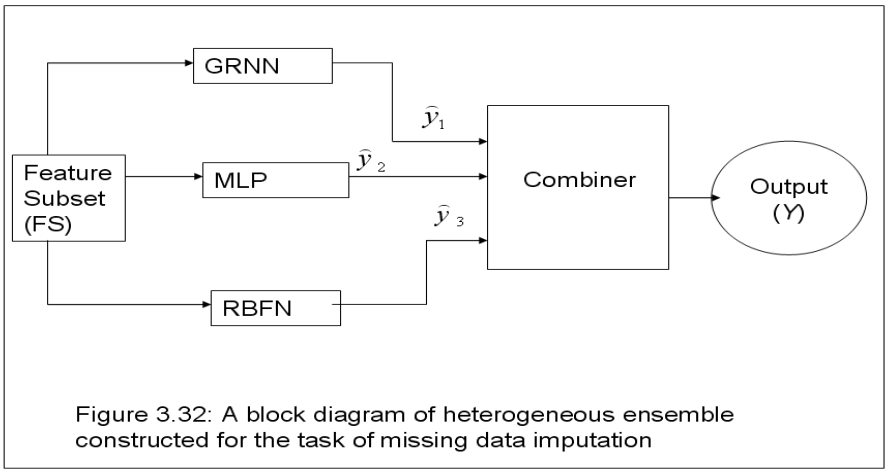

Figure 3.32: A block diagram of heterogeneous ensemble constructed for the task of missing data imputation

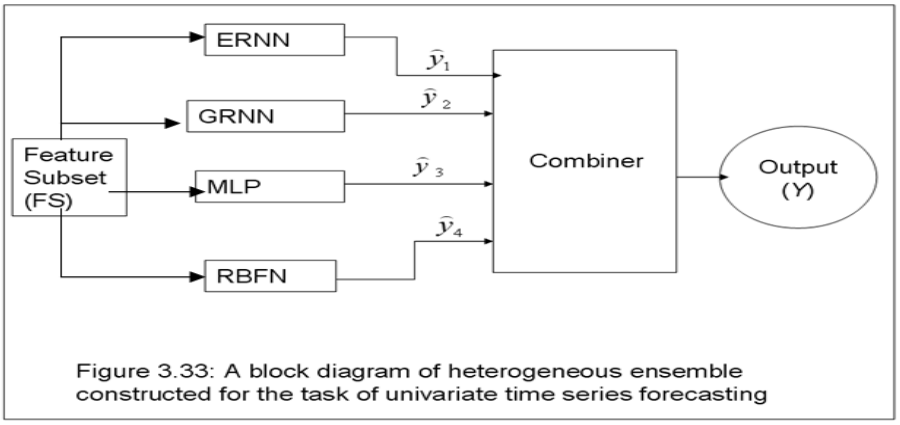

Figure 3.33: A block diagram of heterogeneous ensemble constructed for the task of univariate time series forecasting

The outputs of ensemble members are fused together to get the final decision. Majority and weighted majority voting are common methods for combining the outputs of ensemble members. In our study, we evaluated four types (two homogeneous and two heterogeneous) of neural network ensembles: (1) Heterogeneous ensemble with majority voting, (2) Heterogeneous ensemble with weighted majority voting, (3) Homogeneous ensemble with majority voting, and (4) Homogeneous ensemble with weighted majority voting.

In majority voting, the final prediction of the ensemble on each test data point is an average of the predictions of all ensemble members, as shown in equation (3.58).

$$Y_i = \frac{\sum_{j=1}^{n} \hat{y}_{ij}}{n} \qquad (3.58)$$

Where, Y_i = Actual output of the i th pattern, $\hat{y}_{ij}$ = output of the i th pattern predicted by the j th member, and n =total number of base classifiers.

In weighted majority voting, each ensemble member votes with its confidence. The final output of the ensemble was calculated using the following equation:

$$Y_i = \frac{\sum_{j=1}^{n} (w_j \hat{y}_{ij})}{\sum_{j=1}^{n} w_j} \qquad (3.59)$$

Where, w_j =the weight with which the j'th ensemble member participates in the final output, and $\hat{y}_{ij}$ = output of the i th pattern predicted by the j'th member.

The vote weights of the base classifiers were optimized by a real-valued PSO. The vote weights are assigned to base classifiers based on the global best-fitted combination. The optimization process is described in detail here.

We normalize each variable to the range [0, 1].

Pseudo code of real-valued PSO for optimizing vote weights of ensemble members

Step 1: Specify parameters

- Range of weights: Constrain the reliability weights of ensemble members to be within 0 and 1.
- Population Size: The swarms were set to 10 particles.
- Position of each particle: Let the ensemble consist of 4 base classifiers. Therefore, the search space is 4-dimensional, and the i'th particle of the swarm at time t can be represented by a 4-dimensional position vector. $X_i(t) = [X_{i1}(t), X_{i2}, X_{i3}, X_{i4}]$, where each bit represents the reliability weight of an ensemble member and each bit is a real value in the interval [0, 1]. We randomly initialize the position of each particle.

- Maximum and Minimum Velocities: We set maximum velocity ($V_{\max}$) of any particle to be $1/3$ or 0.33 and minimum velocity ($V_{\min}$) to be 0.05.
- Velocity of each particle: The velocity of the i'th particle at time t is denoted by $V_i(t) = [V_{i1}(t), V_{i2}(t), V_{i3}(t), V_{i4}(t)]$ where each bit represents the velocity of the reliability weight of an ensemble member and each bit is a real value in the interval [0.05 to 0.33]. Particle velocities are initially randomly determined.

Step 2: Evaluate the fitness (prediction accuracy) of each particle by training an ensemble using the particle's current position $X_i(t)$. The prediction accuracy was estimated by 10-fold cross validation.

Let $F(X_i)$ be the fitness score of the *i*'th particle.

$F(X_i)$ = Prediction accuracy of the *i*'th particle as a fraction (3.60)

Where, $0 \le F(X_i) \le 1$

Step 3: The swarm was initialized randomly with each particle's personal best position ($P_{i,best}$) being the same as its current position. Compare the fitness of each particle $F(X_i)$ to its best fitness so far $F(P_{i,best})$; and update each particle's best position. If $F(X_i) > F(P_{i,best})$, then $F(P_{i,best}) = F(X_i)$, and $P_{i,best} = X_i$

Step 4: Update the global best particle (P_{gbest}) and its fitness ($F(P_{gbest})$):

If $F(X_i) > F(P_{gbest})$, then $F(P_{gbest}) = F(X_i)$ and $P_{gbest} = X_i$.

Step 5: Change the velocity of each bit of the particle according to the equation (3.61):

$$V_{ij}(+1) = w.V_{ij}() + c_1 r_1 (P_{ij.best} - X_{ij}) + c_2 r_2 (P_{gbest} - X_{ij}) \quad (3.61)$$

where $w = 1 - \frac{T_{spent}}{T_{\max}}$

If $V_{ij}(t+1) > V_{\max}$, then $V_i(t+1) = V_{\max}$

If $V_{ij}(t+1) < V_{\min}$, then $V_i(t+1) = V_{\min}$

Where, $V_{ij}(t+1)$ and $V_{ij}(t)$ denote velocities of j'th bit for the i'th particle at time *(t+1)* and *t*, respectively. w is the inertia weight which shows the effect of previous velocity vector on the new vector. $T_{\max}$ denotes the total number of iterations and T_{spent} denotes the number of iterations performed so far. r_1 and r_2 are two random numbers between (0, 1). c_1 and c_2 are two positive constants: $c_1 = c_2 = 2$.

Step 6: Move each particle to the new position using equation (3.62)

$$X_{ij}(t+1) = X_{ij}(t) + V_{ij}(t+1) \qquad (3.62)$$

If $X_{ij}(t+1) > 1$; then $X_{ij} = 1$ [since the maximum weight cannot be greater than 1]

If $X_{ij}(t+1) < 0$, then $X_{ij}(t+1) = 0$ [since the minimum cannot be less than 0].

Step 7: Go to step 2, and repeat until stopping criterion is met.

3.2 Description of Datasets used for Model Evaluation

In this section, we describe the datasets that we used for the experiments in our three studies—(i) feature subset selection in large dimensionality domains, (ii) reconstruction of incomplete datasets in cross-sectional studies, and (iii) univariate time series forecasting. In each of these studies, three types of datasets were used: (i) synthetic datasets, (ii) easily available real world public use datasets from repositories, and (iii) new real-life datasets. Synthetic datasets were used to find the best default parameter setting, except in the first

study “Feature subset selection in large dimensionality domains” in chapter 4 where synthetic datasets were used to compare feature subset selection algorithms. An overview of this section is shown in figure 3.34.

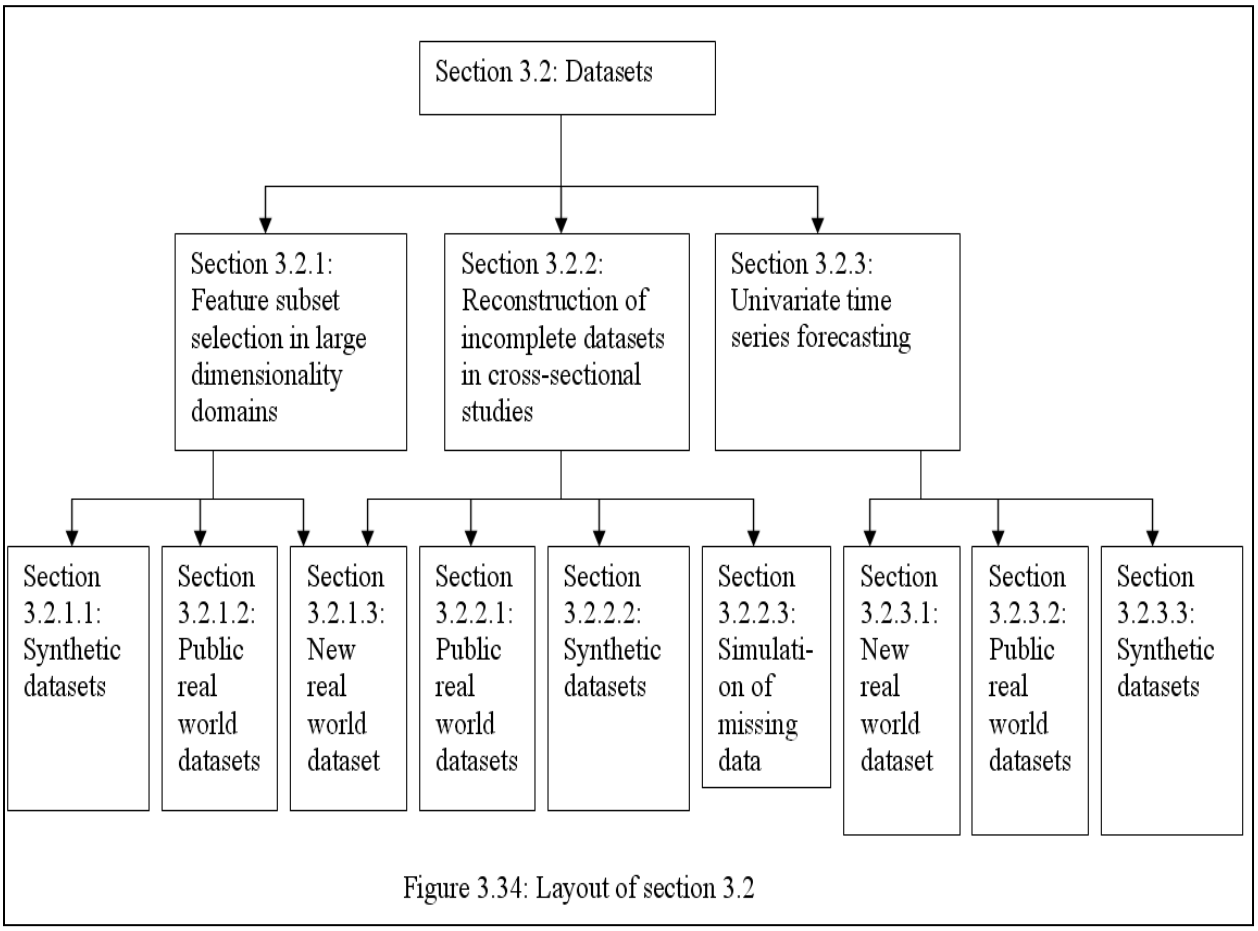

Figure 3.34: Layout of section 3.2

3.2.1 Datasets used in the Study of Feature Subset Selection

We use 11 synthetic datasets, 18 real-world benchmark datasets and one new real world dataset to perform experiments. All of these datasets are high dimensional.

3.2.1.1. ***Synthetic Datasets*****:** Feature interactions and feature redundancy are two major problems often encountered when reducing the dimensionality of feature space. The

principal motivation behind generating synthetic datasets was to recreate these problems on a large scale and perform experiments on controlled datasets.

Each dataset includes 10,000 instances each of 10,000 features. Approximately one third of these features were completely irrelevant. Among these 10,000 features only ten informative features were included in the model. One third of them were actually the exact copy of the set of these ten relevant features. The remaining features are correlated to varying degrees with the relevant features. All features are continuous-valued. They are highly correlated and they interact with one another. The response variable is a binary variable. The following steps were taken to generate these datasets.

Step 1: Specify different mean vectors and different covariance matrices for all the features for the eleven different datasets. Since mean vectors and covariance matrices of no two datasets are the same, the joint distribution of features is different in each dataset.

Step 2: Generate ten thousand combinations of feature values for each dataset from its unique mean vector and covariance matrix.

Step 3: The probability of the event of interest for each instance was estimated by the following model (we specified different sets of model parameters for different datasets). Only ten features among ten thousand features were included in the model. To simulate interactions between features, we included three interaction terms. Interaction terms are formed by the multiplication of two or more explanatory variables. We included one two-way interaction term ($-\beta_5 X_{66} X_{5789}$), one three-way interaction term ($-\beta_9 X_{420} X_{1103} X_{8652}$) and one multi-way interaction term ($+\beta_6 X_{420} X_{6166} X_{6999} X_{7200}$).

$$P = 1/(1+\exp(Z)) \qquad (3.63)$$

$$Z = \beta_0 + \beta_1 X_{66} + \beta_2 X_{1103} + \beta_3 X_{4447} + \beta_4 X_{5789} - \beta_5 X_{66} X_{5789} + \beta_6 X_{420} X_{6166} X_{6999} X_{7200} + \beta_7 X_{8652} + \beta_8 X_{9995} - \beta_9 X_{420} X_{1103} X_{8652}$$

Where, $P(Y)$ = Probability of the event of interest; $(X_1, X_2, \cdots, X_{10000})$ represent different features; $(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8, \beta_9)$ are the model parameters.

We used equation (3.63) to generate all instances of the synthetic datasets. All the features in the model were arranged in the random order in all datasets. The differences between the datasets are mainly due to different combinations of feature values and different values of model parameters.

Step 4: Generate a uniformly distributed random number in the range (0, 1) for each observation. If the random number is greater than the probability of the event of interest, the value of the response variable is 1, otherwise 0.

3.2.1.2. ***Benchmark datasets (modified):*** In addition to 11 synthetic datasets, we tested feature subset selection algorithms on 18 benchmark datasets. Benchmark datasets were taken from UCI machine learning repository. The benchmark datasets are real-world datasets. The benchmark datasets on which the algorithms were tested are: (1) Adult dataset, (2) Annealing dataset, (3) Breast Cancer Wisconsin (Diagnostic) dataset, (4) Breast Cancer Wisconsin (Prognostic) dataset, (5) Chess—King-Rook vs. King-Pawn, (6) Congressional Voting Records dataset, (7) Dermatology-Psoriasis, (8) Dermatology-Seboreic Dermatitis, (9) Dermatology—Lichen Planus, (10) Dermatology—Pityriasis Rosea, (11) Dermatology—Cronic Dermatitis, (12) Dermatitis—Pityriasis Rubra, (13) Hepatitis, (14) Mushroom, (15) Spambase, (16) Wine, (17) Yeast, and (18) Zoo. The descriptions of the original benchmark datasets are available in [148]. These datasets

contain varying number of features and instances, but have fewer than 10,000 features. Hence, we add a series of randomly generated features to each dataset to make a total of 10,000 features. We added completely irrelevant features because we did not want to destroy the original properties of the benchmark datasets. We did not change the number of observations of the benchmark datasets.

3.2.1.1. ***New real world dataset (Smoking dataset):*** We received a three stage cross sectional survey data on the smoking habits of teenagers from the centre for tobacco control research at the University of Stirling and Open University. The data were collected from Scotland, England, Northern Ireland and Wales in three survey stages: stage1 in 1999, stage 2 in 2002 and stage 3 in 2004. The response variable is a binary variable (1=smoker, 0=non-smoker). Explanatory variables include socio-demographic characteristics of respondents, their knowledge and attitudes towards tobacco promotion of all sorts and their smoking knowledge, attitudes and behaviour. This smoking dataset contains 285 features, 3,321 instances but has a large number of missing values. This dataset contains about 37% missing values. Among the respondents, an overall proportion of 11 percent (355 respondents) are smokers. We applied our proposed missing data imputation algorithm (GEMI) to replace missing values (details are available in chapter 5). We did not add artificial features to this dataset.

3.2.2 Datasets used in the Study of Missing Data Imputation

The two hundred synthetic datasets were used to determine the best default parameter setting for our novel algorithms. However, these synthetic datasets were not used for comparing imputation algorithms. We tested missing data imputation algorithms on 50

public real datasets and one new real world dataset (‘smoking dataset’). The smoking dataset is described above (in section 3.2.1.3). The rest of the datasets are described in sections 3.2.2.1 and 3.2.2.2. For controlled experiments, a proportion of known values were artificially removed from these dataset. However, we did not artificially remove values from ‘Smoking’ dataset which originally contains a very high percentage (about 37%) of missing values. In section 3.2.2.3, we describe how we simulated missing data according to three mechanisms: MCAR, MAR and MNAR.

3.2.2.1 Public real-world datasets

The public real-world datasets are obtained from UCI Machine Learning Repository [148]. The UCI datasets on which we tested the algorithms are: (1) Abalone, (2) Adult, (3) Annealing, (4) Arcene, (5) Arrhythmia, (6) Automobile, (7) Balance Scale, (8) Blood Transfusion Service Center, (9) Breast Cancer Wisconsin (Diagnostic), (10) Breast Cancer Wisconsin (Prognostic), (11) Car Evaluation, (12) Census-Income (KDD), (13) Chess (King-Rook vs. King), (14) Chess (King-Rook vs. King-Pawn), (15) Congressional Voting Records, (16) Contraceptive Method Choice, (17) Credit Approval, (18) Cylinder Bands, (19) Dermatology, (20) Dorothea, (21) Echocardiogram, (22) Ecoli, (23) Glass Identification, (24) Haberman’s Survival, (25) Hayes-Roth, (26) Heart Disease, (25) Hepatitise, (27) Horse Colic, (28) Housing, (29) Internet Advertisements, (30) Japanese Credit Screening, (31) Ionosphere, (32) Iris, (33) Letter Recognition, (34)Low Resolution Spectrometer, (35) Lung Cancer, (36) Magic Gamma Telescope, (37) MONK’s Problems, (38) Mushroom, (39)Nursery, (40) Parkinsons, (41) Pima Indians Diabetes, (42) Pittsburgh Bridges, (43) Poker Hand, (44) Post-Operative Patient, (45)Soybean (large), (46) Spambase, (47) SPECHT Heart, (48) Thyroid disease, (49) Wine, and (50) Yeast.

3.2.2.2 Synthetic Datasets

We generated 200 synthetic datasets to search for the best default parameter configuration in the proposed missing value imputation algorithm. Each dataset includes 10,000 instances each of 100 features. All features are continuous-valued. The response variable is a binary variable. The following steps were taken to generate these datasets.

Step 1: Specify different mean vectors and different covariance matrices for 90 features $(x_1, x_2, \cdots, x_{90})$ for the 200 different datasets. Since mean vectors and covariance matrices of no two datasets are the same, the joint distribution of features is different in each dataset.

Step 2: Generate ten thousand combinations of feature values for each dataset from its unique mean vector and covariance matrix.

Step 3: Create 10 new features $(x_{91}, \cdots, x_{100})$ from the first 90 features, using the following equation:

$$x_{ij} = \alpha_0 + \beta_1 x_{i(j-15)} + \alpha_2 x_{i(j-22)} x_{i(j-27)} x_{i(j-50)} - \alpha_3 x_{i(j-66)} x_{i(j-74)} + \sigma_j R_{ij} \quad (3.64)$$

Where, $j = 91,92,\cdots,99,100$; and $i = 1,\cdots,10000$; and $\sigma > 0$

x_{ij} represents the value of jth feature for the ith instance. $(\alpha_1, \alpha_2, \alpha_3)$ are model parameters.

To simulate interactions between features, we included two interaction terms. Interaction terms are formed by the multiplication of two or more explanatory variables. We included one two-way interaction term ($x_{i(j-66)} x_{i(j-74)}$), and one three-way interaction

term ($x_{i(-22)}, x_{i(-27)}, x_{i(-50)}$). R_{ij} is a normally distributed random number with mean 0 and standard deviation 1, σ_j denotes the standard error of the feature x_j .

Step 4: The probability of the event of interest for each instance was estimated by the following model. Only 6 features among 100 features were included in the model.

$$P(Y) = 1/(1+\exp(-Z)) \qquad (3.65)$$
$$Z = \beta_0 + \beta_1 x_{91} + \beta_2 x_{94} + \beta_3 x_{95} + \beta_4 x_{97} - \beta_5 x_{99} x_{100}$$

Where, $P(Y)$ = Probability of the event of interest; $(x_1, x_2, \cdots, x_{100})$ represent different features; $(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5)$ are the model parameters.

All the features in the model were arranged in the random order in all datasets. The differences between the datasets are mainly due to different combinations of feature values and different values of model parameters. We specified different sets of model parameters for different datasets

Step 4: Generate a uniformly distributed random number in the range (0, 1) for each observation. If the random number is greater than the probability of the event of interest, the value of the response variable is 1, otherwise 0.

We then artificially deleted values from these datasets in order to create missing values in complete cases.

3.2.2.3 Simulation of Missing Data

We deleted values from the complete training data to simulate ignorable and non-ignorable missing observations in a dataset.

MCAR missing values: were generated in following steps.

Step 1: Generate uniformly distributed random number in the interval (0, 1) for each observation.

Step 2: Specify a range of values within the interval (0, 1) depending on the percentage of data to be removed.

Step 3: Remove the observation if the corresponding random number lies within the range.

Non-random (MAR & MNAR) missing values: For non-random missing data, we have to remove data in such a way so that removed values of variable x_k depends on the variables x_m and x_n.

Step 1: To simulate non-random missing data, we used a model for the non-responsiveness. The model estimates the probability of removal values of a variable x_k. We generate MAR missing data using equation (3.66) and MNAR missing data using equation (3.67).

$$p(x_{ki}) = \frac{1}{1+\exp(-(\beta_0 + \beta_m x_{mi} + \beta_n x_n \ldots))} \quad (3.66)$$

$$p(x_{ki}) = \frac{1}{1+\exp(-(\beta_0 + \beta_1 x_{ki}))} \quad (3.67)$$

Where, $p(x_{ki})$ = Probability of removal of x_{ki} in the $i-th$ observation, x_{mi} = Value of variable x_m in the *i*-th observation, x_{ni} = Value of variable x_n in the *i*-th observation

$\beta_0, \beta_m, \beta_n \ldots$ are model parameters.

Step 2: Generate a uniformly distributed random number (R_i) in the interval (0, 1) for each observation of the variable x_k.

3.2.3 Datasets used in the Study of Univariate Time-Series Forecasting: We find the best default parameter values for our proposed time series forecasting algorithm through experiments with 200 synthetic datasets. We compared the time series forecasting algorithms on one new real world dataset ('Pulse Pressure dataset') and 35 publicly available real life datasets. These datasets are described below.

3.2.3.1 New real-world dataset ('Pulse Pressure'): The kidneys play a vital role in controlling blood pressure. Dialysis patients are often hypertensive and their blood pressure rises progressively over time. Hypertension (high blood pressure) is thought to be one of the major causes of death in people with chronic kidney disease. There are three measures of blood pressure (BP): systolic BP, diastolic BP, and pulse pressure. Pulse pressure (the difference between the systolic and diastolic readings) is a more accurate predictor of cardiovascular events than systolic and diastolic blood pressure alone. We collected daily pulse pressure readings of one particularly long surviving dialysis patient during an 18 year period (from 1989 to 2006) from the electronic patient records at Glasgow Royal Infirmary with the patient's permission. The time plot of pulse pressure measurements is shown in figure 3.35. Autocorrelations for the time series are shown in figures 3.36.

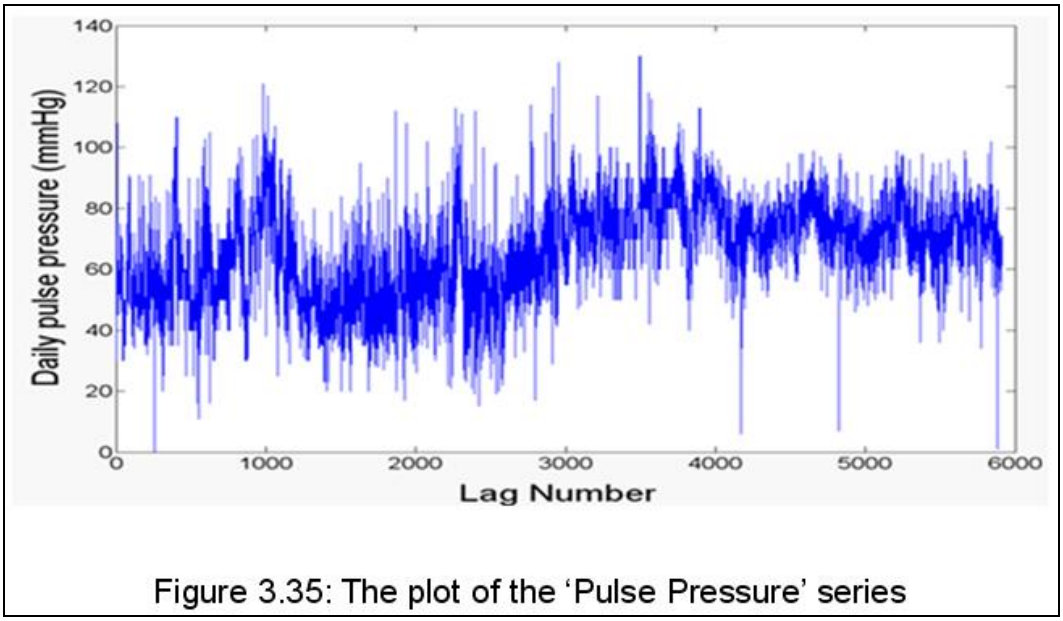

Figure 3.35: The plot of the 'Pulse Pressure' series

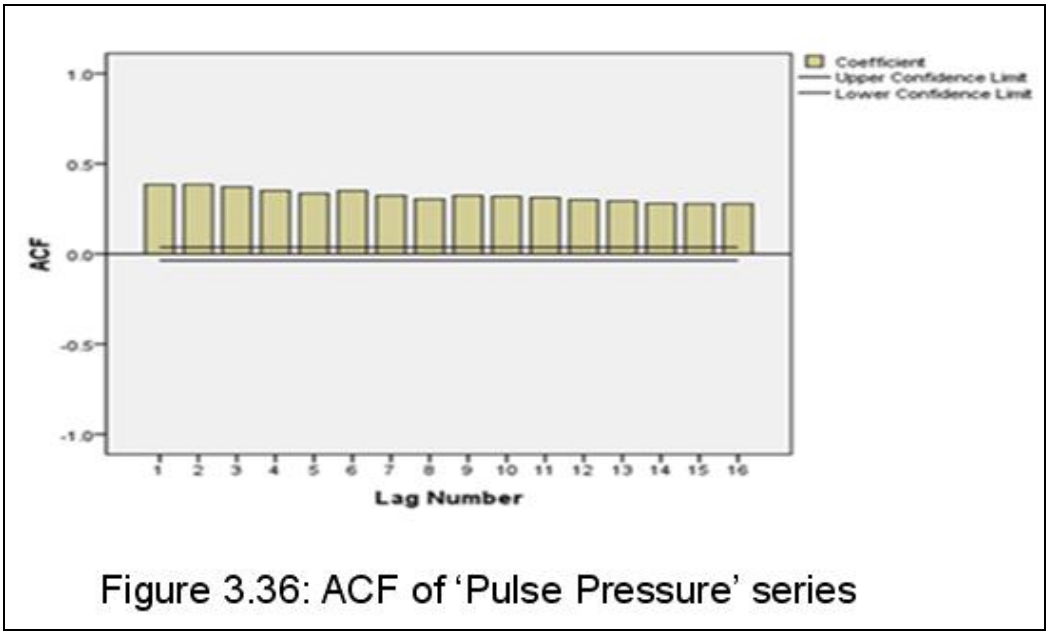

Figure 3.36: ACF of 'Pulse Pressure' series

A correlogram like that in Figure 3.36, where autocorrelation values do not come down to zero reasonably quickly, indicates non-stationarity and so the series needs to be differenced. The pattern on the correlogram (Figure 3.36) indicates that there is a long term trend in the data. In this case, the non-seasonal differencing should be applied to induce stationarity.

We split the pulse pressure series into two equal halves of 18 years. The means and standard deviations of these two halves are presented in table 3.1.

Table 3.1: Summary statistics of the pulse pressure series

Groups	Mean	Standard deviation
First half	57	16
Second half	75	11

The *F*-test shows that there is no statistically significant difference in the variances of first and second halves. However, the independent two sample *t* test reveals that the difference between the means is significant. Therefore, the mean of the series is changing with time, but its variance is stable. These test results confirm the conclusions drawn from the correlograms (Figure 3.36). Thus, it appears that we should start by taking a single nonseasonal difference.

Figure 3.37 shows a time series plot of first-order difference series.

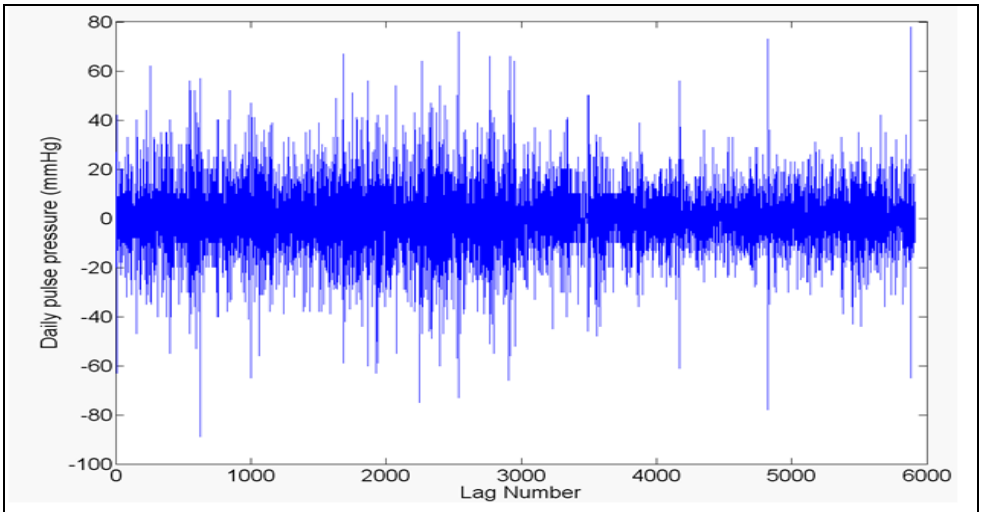

Figure 3.37: First-order difference pulse pressure series

Figure 3.38 shows the ACF of First-order difference pulse pressure series.

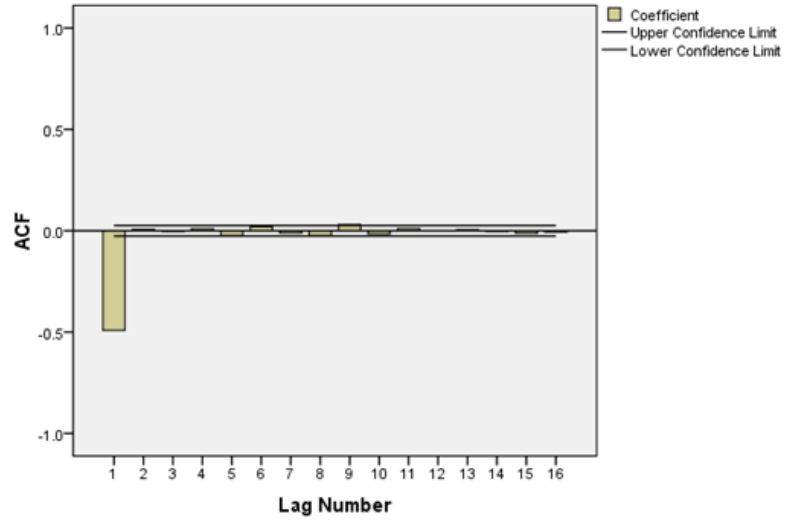

Figure 3.38: The ACF of first-order difference pulse pressure series

Further differencing is not required, since the lag-1 autocorrelation is negative (-0.5). However, this series may be over-differenced. To verify, the *p*-value measuring significance of difference in variances is calculated using a two-tailed *F*-statistics, since another symptom of possible over-differencing is an increase in the standard deviation, rather than a reduction, when the order of differencing is increased. In other words, the optimal order of differencing is often the order of differencing at which the standard deviation is lowest. *F*-test confirms that the difference in variance before and after the first-order differencing is not significant. Furthermore, *T*- and *F*-tests reveal that means and standard deviations of two halves of the first-order difference pulse pressure series are not significantly different at the 0.05 level of probability. Both the two checks failed to find that it is over-differenced.

Summary statistics of the first and second halves of the first-order difference pulse pressure series are presented in table 3.2.

Table 3.2: Summary statistics of the first-order difference pulse pressure series

Groups	Mean	Standard deviation
First half	0.0135	17
Second half	-0.0024	12

3.2.3.2 Public real datasets

In our study, we used following datasets: (1) Airline Data, (2) Bankruptcy Data, (3) Car Data, (4) Electricity Data (5) Gas Data, (6) Hog Data, (7) Hongkong Data, (8) Income Data, (9) Nile Data, (10) Population 1 Data, (11) Population 2 Data, (12) Public Expenditures Data, (13) Share Data, (14) Star Data, (15) Sunspot Data, (16) Temperatures Data, (17) Unemployed Feamales Data, (18) Unemployed 1 Data, (19) Unemployed 2 Data, (20) US Interest Rates Data, (21) Zurich Data, (22) Consumer Markets, (23) Demographics, (24) Flow of Funds, (25) GDP, (26) Government, (27) Income & Earnings, (28) Industry, (29) International Trade, (30) Labor Markets, (31) Money, Credit & Interest Rates, (32) Prices, (33) Real Estate, (34) Society, (35) Stock Markets & Foreign Currency. The first 21 datasets are freely accessible through [149] and all remaining public datasets are hosted on [150].

3.2.3.3 Synthetic datasets

We generated 200 different datasets for experiments. Synthetic datasets were generated using the following steps. We create synthetic time series each of length 1000. Synthetic datasets were generated using the following steps.

Step 1: *Estimate the components of time series***:** The following components of time series are estimated using mathematical models

(i) Base Value: arbitrarily specify the base value (ν) of the time series data.

(ii) Random noise: Estimate the random noise (using Gaussian distribution) for each time point: $\xi_t = \sigma * R$ where *R*=Normal Random Number in (0, 1). We assign the parameter value σ.

(iii) Non-random irregular fluctuation: We define five different sinusoidal models to generate non-random noise (I_t). First we split the time series of size N into fifty equal segments of the length T. Then we randomly select a sinusoidal model for each segment to determine non-random sequences.

(iv) Regular cyclical change (Seasonality/Periodicity): This refers to something that happens periodically, i.e. on regular basis. We estimate the periodicity (s_t) at time lag t using a sinusoidal model:

$$s_t = a_1 \sin\left(\frac{2\pi t}{T_1} + \theta_1\right) + \cdots + a_n \sin\left(\frac{2\pi t}{T_n} + \theta_n\right) \qquad (3.68)$$

(v) Where, we set the parameter values ($a_1, a_2, \cdots, a_n$), ($T_1, T_2, \cdots, T_n$) and $\theta_1, \theta_2 \cdots, \theta_n$.

(vi) Long term trend: We used linear ($m_t = \beta_0 + \beta_1 t$), exponential ($m_t = \beta_0 + \beta_1 \exp$), logarithmic ($m_t = \beta_0 + \beta_1 \log$), or polynomial functions

($m_t = \beta_n t^n + \beta_{n-1} t^{n-1} + \cdots + \beta_2 t^2 + \beta_1 t + \beta_0$) to estimate the trend m_t at time point t.

($\beta_0, \beta_1, \cdots, \beta_n$) are model parameters.

Step 2: *Combine the components of time series* into one based on the additive ($Y_t = \nu + m_t + S_t + I_t + \xi_t$), multiplicative ($Y_t = \nu + m_t S_t I_t \xi_t$), or additive-multiplicative model ($Y_t = \nu + m_t S_t I_t + \xi_t$). **Figures 3.39-3.43** exhibit how synthetic time series looks at different stages of development.

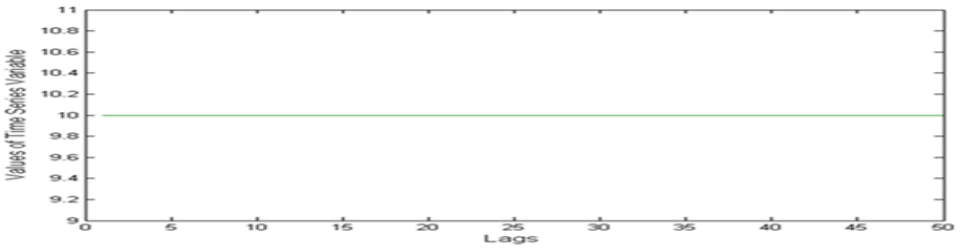

Figure 3.39: Synthetic time series with base value only

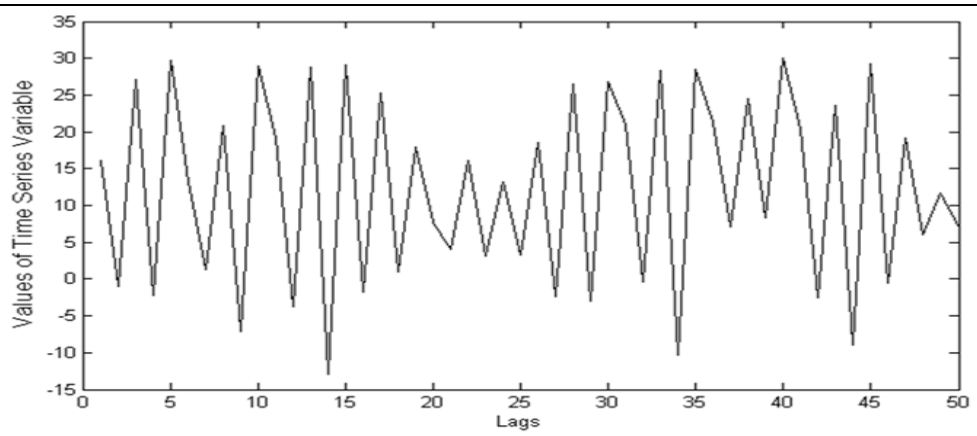

Figure 3.40: Synthetic time series: Z (t)= base value + random noise

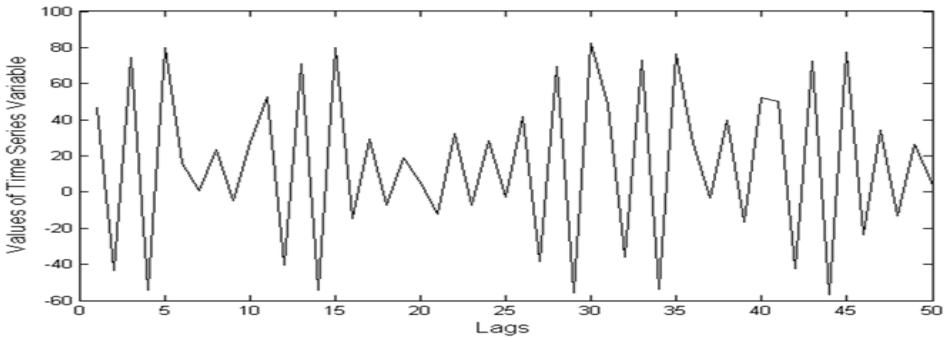

Figure 3. 41: Synthetic time series:
Z (t)= base value + random noise + non-random irregular fluctuations

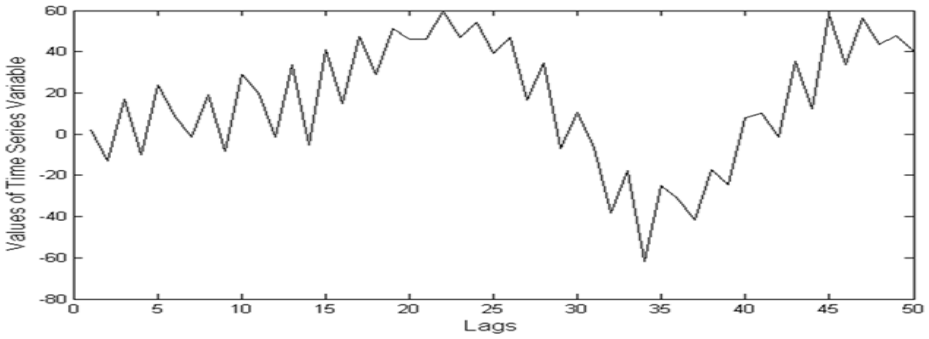

Figure 3.42: Synthetic time series: Z (t)= base value + random noise + non-random irregular fluctuations + periodicity

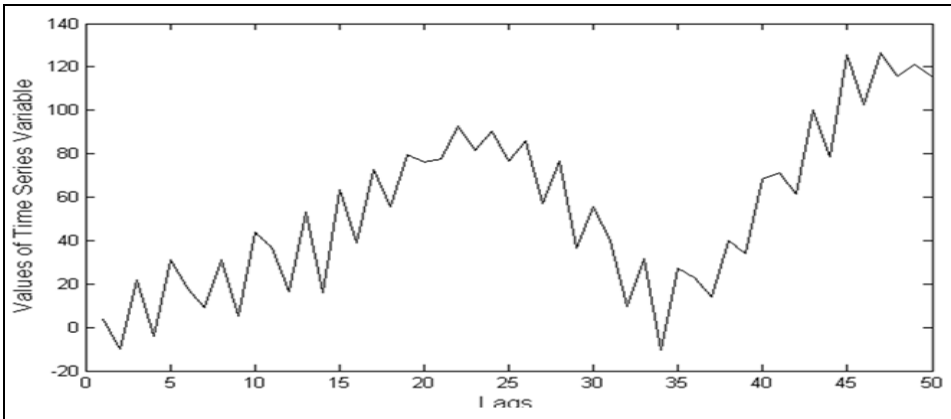

Figure 3.43: Synthetic time series: Z (t)= base value + random noise + non-random irregular fluctuations + periodicity + long-term trend

3.3. Statistical Tests

We applied the following statistical tests to see if any differences among algorithms were of statistical significance. For more details, readers should consult Siegel S, Castellan NJ (1988) [140].

3.3.1 The Friedman Two-Way Analysis of Variance by Ranks

Null Hypothesis*:* The performance of k different algorithms have the same rank totals.

Alternative Hypothesis: The performances of k different algorithms have significantly different rank totals.

The Friedman test determines whether the rank totals for each algorithm differ significantly from the values which would be expected by chance. To do this test, we compute the value of the statistic which we shall denote as F_r.

$$F_r = \left[\frac{12}{Nk(k+1)} \sum_{j=1}^{k} R_j^2 \right] - 3N(k+1) \qquad (3.69)$$

Where, N = Number of Datasets

k = number of search algorithms

R_j = sum of ranks of the j 'th algorithm.

$\sum_{j=1}^{k} R_j^2$ = sum of the squares of the sums of ranks over all algorithms

Appendix table *M* in [401] gives the probabilities associated with values of F_r as large or larger than the tabled values for various values of N and k. If the observed value of F_r is

larger than the tabled value of F_r at the chosen significance level, then null hypothesis may be rejected in favour of alternative hypothesis.

When the obtained value of F_r is significant, it indicates that at least one of the algorithms differ from at least one other algorithm. It does not tell the researcher which one is different, nor does it tell the researcher how many of the algorithms are different from each other. For these answers, we performed the statistical test "Comparisons of Groups or Conditions with a Control" for each pair of algorithms. We discuss this test in the following section (section 3.3.2).

3.3.2 Comparisons of Groups or Conditions with a Control

Null Hypothesis: The performances of two algorithms (algorithm 1 and algorithm 2) are the same.

Alternative Hypothesis: The performances of two algorithms (algorithm 1 and algorithm 2) are not the same.

We can test the significance of differences between two algorithms by using the following inequality:

$$|R_1 - R_2| \geq q(\alpha, \pm c)\sqrt{\frac{Nk(k+1)}{6}} \qquad (3.70)$$

Where, R_1 =rank total of the algorithm 1; R_2 = rank total of the algorithm 2; N = total number of datasets, k = total number of search algorithms ranked, $c = k - 1$; $\alpha = 0.05$. α

represents the level of significance in statistical tests. Values of $q(\alpha,\pm c)$ are given in Appendix Table A_{III} in [140].

If the value of $|R_1 - R_2|$ exceeds the value of $q(\alpha,\pm c)\sqrt{\frac{Nk(k+1)}{6}}$, there is a statistically significant difference between the two algorithms.

3.4. Model Selection Procedures

Model selection criteria are mostly used to decide which model is more appropriate for explaining a specific dataset. A central problem to model selection is over-fitting. In order to overcome the observed over-fitting in model selection, we applied 10-fold cross validation in the experiments described in chapters 4 ("Feature Subset Selection in Large Dimensionality Domains") and 5 ("Reconstruction of Incomplete Datasets in Cross Sectional Studies") of this thesis. In 10-fold cross validation, we divide the data into 10 subsets of approximately equal size. For each model complexity, the learner trains 10 times, each time using one of the subsets as the validation set, one of the subsets as the test set, and the remaining subsets as the training set. In the training set, we fix the tuning parameters and learn the model which is tested using the validation set. We pick the set of tuning parameters that gives the best performance in the validation set. Then the algorithm learns the model with the chosen tuning parameters, this time using training and validation sets. The performance of this final model is assessed using the testing set.

Real world time series datasets frequently contain relatively few observations (less than 100 is common). Hence, it is not practicable to split time series datasets into three parts (a training set, a validation set and a testing set) for experiments described in chapter 6

("Univariate Time Series Forecasting") of this thesis. Alternatives to conventional *k*-fold cross validation based model selection approach include: ERM (Expected Risk Maximization), AIC (Akaike Information Criterion), BIC (Bayesian Information Criterion), and SRM (Structural Risk Minimization). All four (ERM, AIC, BIC, SRM) only require us to compute the training error and not full cross validation error. Out of the candidate models, one with the least mean square error (MSE) is selected in the ERM approach. The ERM approach should not be used for model selection since this model selection mechanism is exposed to a very high risk of over-fitting the data. AIC, BIC, and SRM have a smaller risk of over-fitting than ERM.

The SRM principle was first set out in a 1974 paper by Vapnik and Chervonenkis [151]. SRM estimates how well an algorithm will perform on future data solely based on its training set error, and a property (VC dimension) of the learning algorithm. The VC dimension (Vapnik-Chervonenkis dimension) characterizes the capacity of a trained model. The VC dimension is the maximum number of vectors that can be shattered by the trained net or model. In SRM, the upper bound on the test error is given by

$$test\_error \le Training\_error + \sqrt{\frac{\left[h\left\{\log\left(\frac{2N}{h}\right)+1\right\}-\log\left(\frac{\eta}{4}\right)\right]}{N}} \quad \text{with probability } (-\eta),$$

where h is the VC dimension of the model, and N is the size of the training set. The SRM provides a trade-off between the complexity of approximating functions (the VC dimension of approximating functions) and the quality of fitting the training data (empirical error). SRM selects the model (from a large number of candidate models) whose sum of empirical

risk and VC confidence is minimal. Nevertheless, VC dimension is difficult, if not NP-hard, to compute [152].

Alternatives to VC-dimension-based model selection (SRM or SRMVC) include: (1) AIC (Akaike Information Criterion) and (2) BIC (Bayesian Information Criterion). Like SRM, AIC and BIC have the advantage that we only need the training error.

$$AIC = n\left[\ln\left(\frac{RSS}{n}\right)\right] + 2k \qquad (3.71)$$

$$BIC = n\left[\ln\left(\frac{RSS}{n}\right)\right] + \ln(n)k \qquad (3.72)$$

Where, RSS = residual sum-of-squares on the training data from the estimated model, n = Number of data points in training set, and k = number of free parameters

Given any two estimated models, the models with the lower value of AIC or BIC is the one to be preferred. These approaches resolve over-fitting problem by introducing a penalty term for the number of parameters in the model. BIC is more conservative than AIC since the penalty of BIC is stronger than that of AIC. SRM is wildly conservative. SRM is consistent but difficult to implement in practice. We applied BIC to compare time series forecasting models.

CHAPTER 4

Feature Subset Selection in Large Dimensionality Domains

4.1. Introduction

Searching for an optimal feature subset from a high dimensional feature space is an NP-complete problem, as discussed in section 1.1.1. Furthermore, as was pointed out in section 2.2, existing algorithms usually fail to find an optimal feature subset solution. The process of stagnation, excessively slow convergence and premature convergence all inhibit the ability of conventional feature selection algorithms to perform successfully. We present and test a novel hybrid algorithm for selection of optimal feature subsets. The proposed algorithm consistently generates better feature subsets compared to existing search algorithms within a predefined time limit and keeps improving the quality of selected subsets as the algorithm runs. The rest of the chapter is organized as follows: the new algorithm in section 4.2, comparative performance measurement in section 4.3, results and discussion in section 4.4, summary and conclusions in section 4.5, and future work in section 4.6.

4.2. Proposed algorithm

A good search algorithm should provide: (1) good global search capability that allows for the exploration of new regions of the solution space without getting stuck in local minima, (2) rapid convergence to a near optimal solution, (3) good local search ability, and (4) high computational efficiency.

We present a hybrid algorithm (SAGA), named after two major underlying search algorithms (SA and GA), for selecting optimal feature subsets efficiently. This algorithm is based on SA (simulated annealing), GA (genetic algorithm), GRNN (Generalized Regression Neural Networks) and a greedy search algorithm. SAGA combines the ability to avoid being trapped in a local minimum of SA with a very high rate of convergence of the crossover operator of GA, the strong local search ability of the greedy algorithm and high computational efficiency of GRNN. Our hybrid approach solves the feature selection problems without including filter steps. Hence, unlike existing hybrid algorithms, SAGA does not compromise accuracy for speed.

The SA algorithm here is a mutation-based search approach. Mutation represents a long jump in the search space. The strength of SA is good global search ability. The major disadvantage of SA is its slow convergence speed. On the other hand, GA implements both crossover and mutation operations. The strength of GA is its rapid convergence, but the combination of crossover and a low fixed mutation rate often traps the search in a local minimum. In addition, the local search capability of SA and GA is weak. By contrast, greedy algorithms have good local search ability, but lack global search ability.

SAGA organizes a search in three stages. We encode possible feature subset solutions in ordered, fixed-length binary strings where ‘1’ indicates the presence of the feature and ‘0’ its absence.

Stage 1: SAGA employs SA to guide the global search in a solution space. As long as the temperature is very high, SA accepts every new solution, thus yielding a near random

search through the search space. On the other hand, as the temperature becomes close to zero, only improvements are accepted. The SA is run for approximately 50% of the total time available. The pseudo code of SA is provided below. How the CPU time was recorded is discussed in the last paragraph of this section.

//Pseudo code of SA

Step 1 Initialize Parameters:

- Set the initial temperature **(**T_i**)**: T_i = Total run time for SA.
- Set the current temperature **(**T_c**):** $T_c = T_i$
- Randomly select 100 feature subset solution $I \in I(:100)$ from the pool of possible solutions for initial population.

Step 2 Evaluate the fitness of each solution: Measure the fitness $E_o \left(= E_o(1:100)\right)$ of solutions in the population using GRNN and store the information (feature subset solutions with fitness scores) where, E_o is the prediction accuracy as a fraction (not a percentage) and $0 \le E_o \le 1$.

Step 3 Update the effective temperature (T_c**):** $T_c = T_c - T_{spent}$ (4.1) where T_{spent} = total time spent so far.

Step 4: For all current feature subset vectors $I \in I(:100)$ change the bits of vectors with probability $p_{mu} \in p_{mu}(:100)$: $p_{mu} = 1 - E_o$ (4.2)

Step 5: Evaluate the fitness $E_n \left(= E_n(1:100)\right)$ of the new candidate solutions if not already evaluated.

Step 6 Determine if this new solution is kept or rejected and update the database:

If $E_n \ge E_o$, the new solution is accepted. The new solution replaces the old solution and E_o is set to E_n: $E_o = E_n$. Else we accept the new solution with a certain probability $p_{ac} \in p_{ac}(:100)$: $p_{ac} = \exp\left(-\frac{E_o - E_n}{T_c}\right)$ (4.3).

Step 7: Update the effective temperature T_c. If the effective temperature is greater than zero, return to step 4. Otherwise, the run is finished.

Stage 2: SAGA applies the GA on the 100 best-to-date solutions found by the SA. A total of 50 pairs are picked from the chromosome pool using linear ranking selection. Selection is done "with replacement", meaning that the same chromosome can be selected more than once to become a parent. Rank selection first ranks the population and then every chromosome receives fitness from this ranking. All chromosomes have a chance to be selected. However, the better the chromosomes are, the more chances to be selected they have. Each pair creates two offspring using the half uniform crossover scheme (HUX) and then the parents die. In HUX, exactly half of the non-matching parents' genes are swapped. The main purpose of crossover in GA is to exchange information between pairs of good solutions to form new (and hopefully better) solutions. The mutation operator in GA introduces new genes into the population and retains genetic diversity. GA's initial population already has the good genes found in stage 1. Hence we set a very low mutation rate in order to ensure a quick convergence to an optimal solution. If the value of this parameter is set higher than this level, many good genes are likely to be lost due to the continual emergence of randomness in the genomes. Due to the selection of the fittest chromosomes, the crossover and a very low mutation rate (0.0001), GA converges quickly to a near optimal solution. The GA runs for about 30% of total time spent by SAGA to find the optimal feature subset solution. The pseudo code of the GA is presented below.

//The pseudo code of GA

Step 1: Construct a chromosome pool of size 100 with the 100 fittest chromosomes from the list of feature subset solutions evaluated so far by the SA.

Step 2: Select 50 pairs of chromosomes with replacement using rank-based selection strategy.

Step 3: Perform crossover between the chromosomes using the half uniform crossover scheme (HUX). In HUX, half of the non-matching parents' genes are swapped.

Step 4: Kill the parent solutions.

Step 5: Mutate offspring with probability 0.0001.

Step 6: Evaluate the fitness of the offspring provided if it has not already been evaluated and if sufficient time is available. Update the database and estimate the time left.

Step 7: Go back to step 2 if the time is not up.

Stage 3: SAGA applies a hill-climbing feature selection algorithm. The greedy algorithm performs a local search on the *k*-best solutions (elite) given by two global optimization algorithms (SA and GA) and selects the best neighbours (in our context neighbours are defined in terms of the Euclidean distance between a pair of feature subsets). The hill climbing algorithm is run in the remaining execution time. The pseudo code of hill climbing algorithm is given below.

// Pseudo code of Hill-climbing algorithm

Step 1: Select the best-to-date solution.

Step 2: Create 10000 new candidate solutions from the selected solution by changing only one bit (feature) at a time.

Step 3: Evaluate the new solutions if they are not evaluated before and update the database. Replace the previous solution by the new solution(s) if they are better than the previous solution.

Step 4: Go back to step 2 and perform the hill climbing on each of the accepted new solutions. Repeatedly apply the process from steps 2 to 3 on selected solutions as long as the process is successful in finding improved solutions in every repetition
and as long as the time is available.

Step 5: Update the database and update the time available.

Step 6: Select the next best-to-date solution from the database and go back to step 2 if time is still available.

Computational efficiency is essential for exploring a huge search space. To enhance it, the following measures were taken. First, SAGA employs a robust and fast learning algorithm (GRNN) for assessing candidate solutions. GRNN, based on fuzzy means clustering, is a 'one-pass' algorithm. GRNN has just one parameter (smoothing factor: σ) that needs to be chosen. However, we have discovered the best default value for this parameter through experiments with 200 synthetic datasets (the process of generating synthetic datasets is described in section 3.3.1 of the chapter 3. It is worth noting that these 200 datasets were

not used in the feature subset selection study. Appendix table 4A shows performance impact of different σ values. We set the default value for each centre's width (σ) to 2 times the average distance to 20 nearest neighbours. The default value for the smoothing parameter σ will almost always produce, a good, if not the best, result. In our feature subset selection experiments, we specified the default setting. Hence in GRNN we need not to develop and validate many predictive models. Another major reason why we choose GRNN is that it suffers relatively more from the curse of dimensionality than other algorithms [153]. This is an advantage when trying to eliminate unnecessary variables because GRNN does not have the luxury of producing good results when there are irrelevant and redundant features.

In order to alleviate overfitting, the predictive power (i.e. accuracy or MSE) of the candidate feature subsets was assessed on the holdout test set not used for training. We perform 10-fold cross validation to estimate the testing accuracy of the GRNN classifier. The higher the accuracy, the fitter the solution. If the accuracies of two solutions are the same, then the solution using the smaller number of features wins. Before the evaluation of feature subsets, each feature was normalized by scaling it between 0 and 1.

Second, Cooper and Hinde (2003) report that evolutionary algorithms spend approximately a third of the time testing already tested candidate solutions [154]. SAGA stores information about the candidate solutions evaluated so far in a database and never evaluates a possible solution more than once.

As a result, SAGA has all the four qualities mentioned above.

One of the objectives was realizing exactly how time consuming the feature selection task can be. Hence, a predefined time limit instead of the maximum number of total iterations was chosen as the stopping criterion which has inevitably made our algorithm rather complicated (In any practical application, one should therefore use a standard stochastic algorithm with imposing a maximum number of iterations as stopping criterion). Our empirical observations suggest that a search algorithm spends almost 99% of its running time evaluating the fitness of solutions and the time period required to perform other tasks (such as such as solution generation) is almost negligible. The computation time required to evaluate a feature subset depends on the number of features present in the subset and the number of instances in the dataset. Hence, we empirically find out the time t $\left(t : 10000\right)$ required to estimate the fitness scores of feature subsets with various dimensionalities from 1 to 10,000 using GRNN and store the information (dimensionalities of subsets and time required to assess their fitness) in a database. After evaluating each solution, the amount of CPU time consumed is retrieved from the server and the amount of time available for the search is updated.

4.3. Comparative Performance Analysis

We compare our algorithms with the following benchmark algorithms: four commonly used greedy search algorithms (SBS (sequential Backward Selection, SFS (Sequential Forward Selection), SFFS (Sequential Floating Forward Selection), and SFBS (Sequential Floating Backward Selection)) and four popular stochastic search algorithms (ACO (Ant Colony Optimization), GA (Genetic Algorithm), PSO (Particle Swarm Optimization) and SA (Simulated Annealing). We also compare our algorithm against a hybrid of filter and wrapper approaches—FW (filter-wrapper). Many hybrid algorithms have been proposed

for feature subset selection with encouraging results. It was not possible to implement all the methods and empirically assess them. Instead, based on the experience of other authors, we develop a representative hybrid algorithm FW. This consists of a number of popular filter methods (standard statistical tests – such as unpaired *t*-test, Mann-Whitney U test, and Chi-Square test; RELIEF algorithm; Pearson's Correlation Coefficient; Symmetric Uncertainty; and Principal Component Analysis) and a stochastic algorithm (simulated annealing). The benchmark algorithms are described in section 3.1.1. The proposed and benchmark algorithms were tested on 30 high dimensional datasets (descriptions of datasets are provided in section 3.2.1).

There are a number of strategies employed to ensure fair comparison of search algorithms.

- All algorithms were run on a 3.40 GHz Intel® Pentium® D CPU with 2 GB RAM.
- The values of each feature were normalized in a 0 to 1 range before the experiment.
- All algorithms use GRNN classifiers to evaluate each of the resulting subsets using 10-fold cross validation.
- No algorithm evaluates the same solution more than once.
- Each algorithm was allocated exactly the same amount of search time. Each stochastic search algorithm (ACO, FW, GA, PSO, SA and SAGA) was run 10 times on each dataset, each time with different initial populations of 100 individuals. The final performance of each algorithm was calculated by averaging over all 10 simulations.
- Algorithms were ranked based on their performance. Their performance is measured in terms of classification accuracy with the best solution found during the entire run. Two different solutions having the same accuracy level are assessed in terms of the number of

features present in the feature subset solutions. We assign rank 1 to the best algorithm and rank m $(m \leq 10)$ to the worst algorithm. The Friedman test is used to test the null hypothesis that the performance is the same for all algorithms. After applying the Friedman test and noting that it is significant, a pairwise comparison test (Siegel S, Castellan NJ (1988)), comparison of groups or conditions with a control, was used in order to test the (null) hypothesis that there is no significant difference between any pair of the ten algorithms [140]. Section 3.3 briefly describes these statistical tests.

4.4. Results and Discussion

We compare our proposed algorithms (SAGA) with the conventional search algorithms (ACO, FW, GA, PSO, SA, SBS, SFBS, SFFS and SFS) on 30 high-dimensional datasets. The best-to-date feature subset solutions for real words datasets are presented in appendix table 4B. The algorithms were evaluated based on the fitness of the best feature subset solutions generated by the algorithms within the allowed time limits. The Friedman test reveals significant differences ($p < 0.05$) in the performance of the ten search algorithms at all time limits. Table 4.1 shows statistical test results for pairwise comparisons of algorithms. We assign the rank 1 to the best algorithm, the rank 2 to the next best algorithm and so on.

Table 4.1: Pairwise Comparisons between Search Algorithms

After 1 hour

Rank	Algorithm(s)	Significantly outperformed algorithms
1	SAGA, SFS	(1) FW, (2) GA, (3) SFFS, (4)ACO, (5) SBS, (6) SFBS, (7)PSO, (8) SA
2	FW	(1) GA, (2) SFFS, (3) ACO, (4) SBS, (5) SFBS, (6)PSO, (7)SA
3	GA	(1) SBS, (2) SFBS, (3) PSO, (4) SA
4	SFFS, ACO, SBS, SFBS	(1) PSO, (2) SA
5	PSO, SA	-

After 8 hours

Rank	Algorithm(s)	Significantly outperformed algorithms
1	SAGA, SFS	(1)SFFS, (2)FW, (3)PSO, (4)GA, (5)SBS, (6)SA, (7)ACO, (8)SFBS
2	SFFS, FW	(1)GA, (2)SBS, (3)SA, (4)ACO, (5)SFBS
3	PSO	(1)SA, (2)ACO, (3)SFBS
4	GA, SBS	(1)ACO, (2)SFBS
5	SA, ACO	(1)SFBS
6	SFBS	-

After 16 hours

Rank	Algorithm(s)	Significantly outperformed algorithms
1	SAGA	(1)SFBS, (2)SFFS, (3)FW, (4)SFS, (5)SA, (6)PSO, (7)ACO, (8)GA, (9)SBS
2	SFBS, SFFS	(1)SFS, (2)SA, (3)PSO, (4)ACO, (5)GA, (6)SBS
3	FW	(1)SA, (2)PSO, (3)ACO, (4)GA, (5)SBS
4	SFS, SA	(1)PSO, (2) ACO, (3)GA, (4)SBS
5	PSO	(1)GA, (2)SBS
6	ACO, GA	(1)SBS
7	SBS	

After 24 hours

Rank	Algorithm(s)	Significantly outperformed algorithms
1	SAGA	(1)FW, (2)SFBS, (3)SFFS, (4)SA, (5)SFS, (6)GA, (7)PSO, (8)ACO, (9)SBS
2	FW	(1)SFFS, (2)SA, (3)SFS, (4)GA, (5)PSO, (6)ACO, (7)SBS
3	SFBS, SFFS	(1)SFS,(2)GA, (3)PSO, (4)ACO, (5)SBS
4	SA, SFS	(1)GA, (2)PSO, (3) ACO, (4)SBS
5	GA, PSO, ACO	(1)SBS
6	SBS	-

After 72 hours

Rank	Algorithm(s)	Significantly outperformed algorithms
1	SAGA	(1)SA, (2)FW, (3)SFBS,(4)SFFS, (5)SFS, (6)PSO, (7)GA, (8)ACO, (9)SBS
2	SA, FW	(1)SFBS, (2)SFFS, (3)SFS, (4)PSO, (5)GA, (6)ACO, (7)SBS
3	SFBS, SFFS	(1)SFS, (2)PSO, (3)GA, (4)ACO, (5)SBS
4	SFS, PSO	(1)ACO, (2)SBS
5	GA, ACO	(1)SBS
6	SBS	-

Table 4.1: Pairwise Comparisons among Search Algorithms (cont.)

		After 168 hours
Rank	Algorithm(s)	Significantly outperformed algorithms
1	SAGA	(1) SA, (2)FW, (3)SFBS, (4)SFFS, (5)PSO, (6)SFS, (7)GA, (8)ACO, (9)SBS
2	SA	(1)FW, (2)SFBS, (3)SFFS, (4)PSO, (5)SFS, (6)GA, (7)ACO, (8)SBS
3	FW	(1)SFBS, (2)SFFS, (3)PSO, (4)SFS, (5)GA, (6)ACO, (7)SBS
4	SFBS, SFFS	(1)SFS, (2)GA, (3)ACO, (4)SBS
5	PSO	(1)GA, (2)ACO, (3)SBS
6	SFS, GA	(1) ACO, (2)SBS
7	ACO	(1) SBS
8	SBS	-
		After 240 hours
Rank	Algorithm(s)	Significantly outperformed algorithms
1	SAGA	(1)SA, (2)FW, (3)PSO, (4)SFBS, (5)SFFS, (6)SFS, (7)GA, (8)ACO, (9)SBS
2	SA	(1)FW, (2)PSO, (3)SFBS, (4)SFFS, (5)SFS, (6)GA, (7)ACO, (8)SBS,
3	FW	(1)PSO, (2)SFBS, (3)SFFS, (4)SFS, (5)GA, ((6)ACO, (7)SBS
4	PSO, SFBS, SFFS	(1)SFS, (2)GA, (3)ACO, (4)SBS
5	SFS, GA	(1) ACO, (2)SBS
6	ACO	(1)SBS
7	SBS	-

Figure 4.1 displays the results for the mean accuracy of the reduced datasets at run times of 1, 8, 16, 24, 72, 168 and 240 hours: SAGA outperforms the others at all run times, though not always significantly.

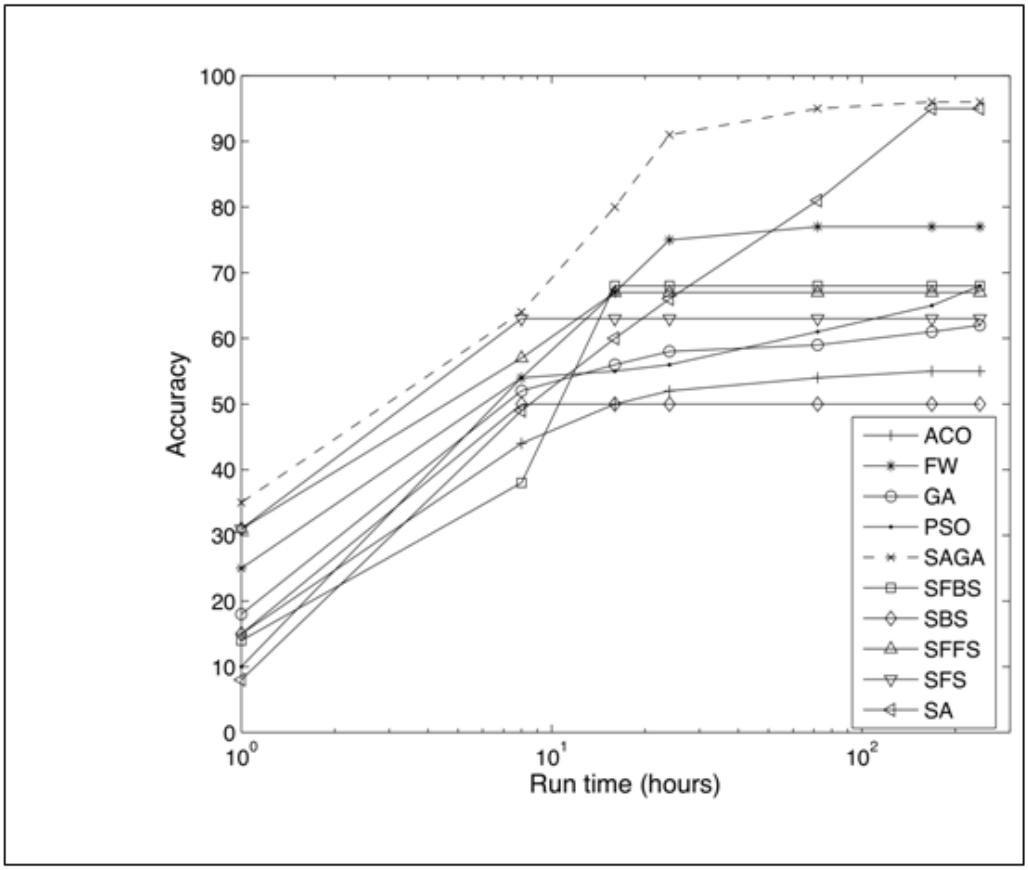

Figure 4.1: Performance of the different algorithms (accuracy only).

Table 4.2 summarizes the results in more detail, providing the mean accuracy, average number of features selected over all 30 datasets both with standard deviations. The average accuracy represents the overall performance. However, if two solutions are equally accurate (as is almost the case with SAGA and SA at 168 and 240 hours), then the one with fewer features is fitter. We note that SAGA achieves the same accuracy with far fewer features than SA at 168 and 240 hours. Table 4.2 also reports the relative performance of search algorithms in terms of the number of algorithms that are significantly worse than the control (based on pairwise comparison tests).

Table 4.2: Summary performance report: A comparison of feature subset selection algorithms (The values in parenthesis represents the standard deviation. The lower the standard deviation, the more consistent the algorithm.)

Search Method	Time (hrs)	No of significantly out-performed algorithms	Accuracy (%)	Features	Search Method	Time (hrs)	Number of significantly out-performed algorithms	Accuracy (%)	Features
ACO	1	2	15 (8)	4241(1231)	SAGA	1	8	35 (182)	71 (26)
	8	1	44 (17)	2375 (1296)		8	8	64 (12)	27 (24)
	16	2	50 (19)	1595 (4388)		16	9	80 (10)	22 (14)
	24	1	52 (19)	836 (546)		24	9	91 (6)	10 (6)
	72	1	54 (18)	599 (808)		72	9	95 (4)	17 (9)
	168	1	55 (18)	524 (1792)		168	9	96 (3)	17 (7)
	240	1	55 (18)	442 (617)		240	9	96 (3)	12 (8)
FW	1	7	25 (129)	5797(2487)	SBS	1	2	15 (79)	8750 (87)
	8	5	54 (14)	436 (283)		8	2	50 (5)	35 (38)
	16	5	67 (11)	218 (129)		16	0	(converged)	(converged)
	24	7	75 (12)	169 (134)		24	0		
	72	7	77 (10)	256 (249)		72	0		
	168	7	77 (10)	226 (107)		168	0		
	240	7	77 (10)	200 (91)		240	0		
GA	1	4	18 (93)	6984 (18)	SFBS	1	2	14 (75)	9040 (96)
	8	2	52 (15)	553 (409)		8	0	38 (8)	57 (29)
	16	1	56 (14)	446 (379)		16	6	68 (6)	45 (27)
	24	1	58 (14)	553 (2119)		24	5	(converged)	(converged)
	72	1	59 (13)	513 (204)		72	5		
	168	2	61 (11)	388 (320)		168	4		
	240	2	62 (10)	482 (206)		240	4		
PSO	1	0	10 (53)	5598 (3046)	SFFS	1	2	31 (161)	1 (0.4)
	8	3	54 (11)	852 (1230)		8	5	57 (9)	41 (10)
	16	2	55 (11)	242 (166)		16	6	67(5)	34 (11)
	24	1	56 (11)	222 (305)		24	5	(converged)	(converged)
	72	2	61 (7)	240 (87)		72	5		
	168	3	65 (5)	307 (100)		168	4		
	240	4	68 (4)	181 (76)		240	4		
SA	1	0	8 (42)	6647 (3451)	SFS	1	8	31 (162)	3 (2)
	8	1	49 (12)	708 (745)		8	8	63 (11)	37 (42)
	16	4	60 (10)	290 (137)		16	5	(converged)	(converged)
	24	4	66 (11)	142 (208)		24	4		
	72	7	81 (6)	229 (153)		72	2		
	168	8	95 (3)	323 (187)		168	2		
	240	8	95 (3)	157 (69)		240	2		

The key findings of this work are:

- SAGA holds the first position at all seven durations of running where the best performance is significantly better than the next best at the significance level of 0.05.
- The rate of improvement in search algorithms decreases as the time passes. However, as the running time increases, the improvement in the performance of SAGA declines slowly, relative to the others, with the exception of SA. Extensive experiments illustrate that after eight hours of searching; SAGA had a 64% mean accuracy with a 12% standard deviation. The mean accuracy rate of SAGA improved by 32% (from 64% to 96%) over the last 232 hours of running for which the standard deviation was reduced by about 9% (from 12% to 3%), while the overall accuracy rate of SA increased from 49% to 95% (46%) and the standard deviation dropped from 12% to 3% (a 9% drop). The other algorithms had significantly lower rates of increase in accuracy with increasing running time. The accuracy improvements of other algorithms over the last 232 hours of running are as follows: using the ACO the mean accuracy increased by about 10% and the standard deviation increased by about 1%, using the FW the accuracy increased by about 23% while the standard deviation dropped by about 4%, using the GA the accuracy increased by about 10% while the standard deviation dropped by about 5%, and using the PSO the accuracy increased by about 14% while the standard deviation dropped by about 7%.
- Throughout the running period, SAGA selected a much smaller number of features than other algorithms. For this reason, the performance of SAGA remained significantly better than SA even when the accuracy rates of both algorithms were almost the same (at 168 and 240 hours).

In addition, we note that if the search space is too large while the time available to conduct a search through the search space is very brief, SFS is a good choice among other conventional search algorithms. 1 hour and 8 hours later, SFS finished joint first with SAGA, outperforming eight algorithms.

Further, if sufficient time is available, then SA should be considered among the conventional search algorithms. After the first 1 hour of running, the performance of SA was the worst among all search algorithms. After 8 hours, SA outperformed two algorithms. After 24 hours, it outperformed four algorithms. After 72 hours of searching, it became the second best algorithm, outperforming seven algorithms. After 168 hours, it outperformed eight algorithms.

We also noted that when a greedy algorithm reaches the local minima it cannot climb 'out'. Both SBS and SFS rapidly converged within approximately 8 hours. SFFS and SFBS offer only slightly more resistance to local minima. They converged within 16 hours. We found an interesting pattern in the relative performance of the algorithms (in terms of the number of significantly outperformed algorithms) over time. The relative performance of ACO, GA, SBS and SFS deteriorates, while the relative performance of PSO and SA improves as the time elapses. The relative performance of SFBS and SFFS roughly follows the Gaussian distribution over time. In addition, FW offers the most consistency (after SAGA) over time in terms of relative performance. Although FW applies a SA, it did not generate a dramatic performance improvement over time, like SA did. We suspect that a number of key features were already removed from the feature pool by filter methods, before SA began its search.

4.5 Summary and conclusions

We have presented a hybrid algorithm SAGA that combines the strengths of a number of existing algorithms to select the optimal feature subsets from a large feature space. SAGA is a hybrid of a number of wrapper methods—a SA, a GA, a GRNN, and a greedy search algorithm. We compare our proposed algorithm against the following benchmark algorithms: ACO, FW, GA, PSO, SA, SBS, SFBS, SFFS, and SFS on both synthetic and real world-datasets. Among these datasets, one dataset ('Smoking dataset') has 285 features and the remaining 29 datasets have 10,000 features each. We study the performance of these algorithms at different time intervals: after 1, 8, 16, 24, 72, 168, and 240 hours of running. The performance of our algorithm is highly encouraging. SAGA shows the best performance over every interval. We conclude that no existing algorithm is entirely satisfactory in isolation, but that a carefully designed combination can overcome the weaknesses of each.

4.6 Future Work

Feature subset selection is important both in supervised and unsupervised data analysis. The applicability of the proposed algorithm SAGA is limited to the supervised learning domain. In the future, we plan to extend the SAGA in order that it can be applied to feature selection in unsupervised learning (data clustering). In an unsupervised scenario, in particular the assessment of the quality of a feature subset solution becomes more intricate.

Appendix 4A: The performance of different smoothing factor parameter values

Rank (based on statistical significance)	Each centre's width	Classification Accuracy (%)
4	(Euclidean distance to the nearest member)*0.5	79 (12)
3	Euclidean distance to the nearest member	85 (8)
3	(Euclidean distance to the nearest member)*1.5	87 (7)
3	(Euclidean distance to the nearest member)*2	88 (7)
3	(Euclidean distance to the nearest member)*2.5	84 (8)
4	(Euclidean distance to the nearest member)*3	81 (12)
1	(Average Euclidean distance to 5 nearest members)*2	95 (4)
1	(Average Euclidean distance to 10 nearest members)*2	96 (4)
1	(Average Euclidean distance to 20 nearest members)*2	97 (3)
1	(Average Euclidean distance to 30 nearest members)*2	96 (5)
1	(Average Euclidean distance to 40 nearest members)*2	95 (5)
2	(Average Euclidean distance to 50 nearest members)*2	92 (7)
3	(Average Euclidean distance to 60 nearest members)*2	88 (10)
4	(Average Euclidean distance to 70 nearest members)*2	84 (11)
5	(Average Euclidean distance to 80 nearest members)*2	75 (12)
5	(Average Euclidean distance to 90 nearest members)*2	72 (14)
5	(Average Euclidean distance to 100 nearest members)*2	72 (17)

Appendix 4B: Best solution found for each real-world dataset (underlined features were selected for inclusion in the optimal set)

Modified Benchmark Dataset	**Original Features**
Adult (total observations = 48842 and total features=500 Original features =13 Artificial features=487) Accuracy: 86% Features selected: 8	(1)age, (2)workclass, (3) fnlwgt, (4) education, (5) marital-status, (6)occupation, (7) relationship, (8)race, (9)sex, (10)capital-gain, (11) Capital-loss, (12) hours-per-week, (13) native country
Annealing (total observations =798 and total features=500 Original features =38 Artificial features=462) Accuracy: 96% Features selected: 13	(1) family, (2) product-type, (3) steel, (4)carbon, (5)hardness, (6)temper-rolling, (7) condition, (8) formability, (9) strength, (10) non-ageing, (11)surface-finish, (12) surface-quality, (13)enamelability,(14) bc, (15)bf, (16)bt, (17)bw/me, (18)bl, (19)m, (20)chrom, (21)phos, (22)cbond, (23)marvi, (24)exptl, (25)ferro, (26)corr, (27)blue/bright/vam/clean, (28)lustre, (29)jurofm, (30)s, (31)p, (32)shape, (33)thick, (34)idth, (35)len, (36)oil, (37)bore, (38) packing
Breast Cancer Wisconsin (diagnostic) (total observations=569 and total features=500 Original features =32 Artificial features=462) Accuracy: 94% Features selected: 6	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32
Breast Cancer Wisconsin (Prognostic) (total observations =198 and total features=500 Original features =34 Artificial features=466) Accuracy: 82% Features selected: 3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34

Chess (King-Rook vs. King Pawn) (total observation=6 and total features=500 Original features =36 Artificial features=464) Accuracy: 99% Features selected: 20	(1) bkblk, (2) bknwy, (3) bkon8, (4) bkona, (5) bkspr, (6) bksbq, (7)bkxcr, (8)bkswp, (9)blxwp, (10)bxqsq, (11)cntxt, (12)dsopp, (13)dwipd, (14)hdchk, (15)katri, (16)mulch, (17)qxmsq, (18)r2ar8, (19)reskd, (20)reskr, (21)rimmx, (22)rkxwp, (23)rxmsq, (24)simpl, (25)skach, (26)skewr, (27)skrxp, (28)spcop, (29)stlmt, (30)thrsk, (31)wkcti, (32)wkna8, (33)wknck, (34)wkovl, (35)wkpos, (36)wtoeg
Congressional Voting (total observations =435 and total features=500 Original features =16 Artificial features=484) Accuracy: 99% Features selected: 7	(1)handicapped-infants, (2)water-project-cost-sharing, (3)adoption-of-the-budget-resolution, (4)physician-fee-freeze, (5)el-salvador-aid, (6)religious-groups-in-schools, (7)anti-satelite-test-ban, (8)aid-to-nicaraguan-contras, (9)mx-missile, (10)immigration, (11)synfuels-corporation-cutback, (12)education-spending, (13)superfund-right-to-sue, (!4)crime, (15)duty-free-exports, (16) export-administration-act-south-africa
Dermatology: Psoriasis (total observations=366 and total features=500 Original features =33 Artificial features=467) Accuracy: 100% Features selected: 3	(1)Age, (2)erythema, (3)scaling, (4)definite borders, (5)itching, (6)koebner phenomenon, (7)polygonal papules, (8)follicular papules, (9)oral mucosal involvement, (10)knee and elbow involvement, (11)scalp involvement, (12)family history, (13)melanin incontinence, (14)eosinophils in the infiltrate, (15) PNL infiltrate, (16)fibrosis of papillary dermis, (17)excytosis, (18)acanthosis, (19)hyperkeratosis, (20)parakeratosis, (21)clubbing of the rete ridges, (22)elongation of the rete redges, (23) thinning of the suprapapillary epidermis, (24) spongiform pustule, (25)munro microabcess, (26)focal hypergranulosis, (27)disappearance of the granular layer, (28)vacuolisation and damage of basal layer, spongiosis, (29)saw-tooth appearance of retes, (30)follicular horn plug, (31)perifollicular parakeratosis, (32)inflammatory monoluclear infiltrate, (33) band-like infiltrate

Dermatology: Sebreic Dermatitis(total observations=366 and total features=500 Original features =33 Artificial features=467) Accuracy: 99% Features selected: 7	(1)Age, (2)erythema,(3)scaling, (4)definite borders, (5)itching, (6)koebner phenomenon, (7)polygonal papules, (8)follicular papules, (9)oral mucosal involvement, (10)knee and elbow involvement, (11)scalp involvement, (12)family history, (13)melanin incontinence, (14)eosinophils in the infiltrate, (15) PNL infiltrate, (16) fibrosis of papillary dermis, (17) excytosis, acanthosis, (18)hyperkeratosis, (19)parakeratosis, (20)clubbing of the rete ridges, (21)elongation of the rete redges, (22)thinning of the suprapapillary epidermis, (23)spongiform pustule, (24)munro microabcess, (25)focal hypergranulosis, (26)disappearance of the granular layer, (27)vacuolisation and damage of basal layer, ((28)spongiosis, (29)saw-tooth appearance of retes,(30) follicular horn plug, (31)perifollicular parakeratosis, (32)inflammatory monoluclear infiltrate, (33)band-like infiltrate
Dermatology: Lichen Planus (total observations=366 and total features=500 Original features =33 Artificial features=467) Accuracy: 99% Features selected: 2	(1)Age, (2)erythema, (3)scaling, (4)definite borders, (5)itching, (6)koebner phenomenon,(7) polygonal papules, (8)follicular papules, (9)oral mucosal involvement, (10)knee and elbow involvement, (11)scalp involvement, (12)family history, (13)melanin incontinence, (14)eosinophils in the infiltrate, (15)PNL infiltrate, (16)fibrosis of papillary dermis, (17)excytosis, (18)acanthosis, (19)hyperkeratosis, (20)parakeratosis, (21)clubbing of the rete ridges, (22)elongation of the rete redges, (23)thinning of the suprapapillary epidermis, (24)spongiform pustule, (25)munro microabcess, (26)focal hypergranulosis, (27)disappearance of the granular layer, (28)vacuolisation and damage of basal layer, (29)spongiosis, saw-tooth appearance of retes, (30)follicular horn plug, (31)perifollicular parakeratosis, (32)inflammatory monoluclear infiltrate, (33)band-like infiltrate
Dermatology: Pityriasis Rosea (total observations=366 and total features=500 Original features =33 Artificial features=467) Accuracy: 100% Features selected: 12	(1)Age, (2)erythema, (3)scaling, (4)definite borders, (5)itching, (6)koebner phenomenon, (7)polygonal papules, (8)follicular papules, (9)oral mucosal involvement, (10)knee and elbow involvement, (11)scalp involvement, (12)family history, (13)melanin incontinence, (14)eosinophils in the infiltrate, (15)PNL infiltrate, (16)fibrosis of papillary dermis, (17)excytosis, (18)acanthosis, (19)hyperkeratosis, (20)parakeratosis, (21)clubbing of the rete ridges, (22)elongation of the rete redges, (23)thinning of the suprapapillary epidermis, (24)spongiform pustule, (25)munro microabcess, focal hypergranulosis, (26)disappearance of the granular layer, (27)vacuolisation and damage of basal layer, (28)spongiosis, (29)saw-tooth appearance of retes, (30)follicular horn plug, (31)perifollicular parakeratosis, (32)inflammatory monoluclear infiltrate, (33)band-like infiltrate

Dermatology: Cronic Dermatitis (total observations=366 and total features=500 Original features =33 Artificial features=467) Accuracy: 100% Features selected: 6	(1)Age, (2)erythema, (3)scaling, (4)definite borders,(5) itching, (6)koebner phenomenon, (7)polygonal papules, (8)follicular papules, (9)oral mucosal involvement, (10)knee and elbow involvement, (11)scalp involvement,(12) family history, (13)melanin incontinence, (14)eosinophils in the infiltrate, (15)PNL infiltrate, (16)fibrosis of papillary dermis, (17)excytosis, acanthosis, (18)hyperkeratosis, (19)parakeratosis, (20)clubbing of the rete ridges, (21)elongation of the rete redges, (22)thinning of the suprapapillary epidermis, (23)spongiform pustule, (24)munro microabcess, (25)focal hypergranulosis, (26)disappearance of the granular layer, (27)vacuolisation and damage of basal layer, (28)spongiosis, (29)saw-tooth appearance of retes, (30)follicular horn plug, (31)perifollicular parakeratosis, (32)inflammatory monoluclear infiltrate, (33)band-like infiltrate
Dermatology: Pityriasis Rubra Pilaris (total observations=366 and total features=500 Original features =33 Artificial features=467) Accuracy: 100% Features selected: 1	(1)Age,(2) erythema, (3)scaling, (4)definite borders, (5)itching, (6)koebner phenomenon, (7)polygonal papules, (8)follicular papules, (9)oral mucosal involvement, (10)knee and elbow involvement, (11)scalp involvement, (12)family history, (13)melanin incontinence, (14)eosinophils in the infiltrate, (15)PNL infiltrate, (16)fibrosis of papillary dermis, (17)excytosis, acanthosis, (18)hyperkeratosis, (19)parakeratosis, (20)clubbing of the rete ridges, (21)elongation of the rete redges, (22)thinning of the suprapapillary epidermis, (23)spongiform pustule, (24)munro microabcess, (25)focal hypergranulosis, (26)disappearance of the granular layer, (27)vacuolisation and damage of basal layer, (28)spongiosis, (29)saw-tooth appearance of retes, (30)follicular horn plug, (31)perifollicular parakeratosis, (32)inflammatory monoluclear infiltrate, (33)band-like infiltrate
Hepatitis (total observations=155 and total features=500 Original features =18 Artificial features=482) Accuracy: 90% Features selected: 4	(1)Age, (2)sex, (3)steroid, (4)antivirals, (5)fatigue, (6)malaise, (7)anorexia, liver big, (8)liver firm, (9)spleen palpable, (10)spiders, (11)ascites, (12)varices, (13)bilirubin, (14)alk phosphate, (15)sgot, (16)albumin, (17)protime, (18)histology

Hepatitis (total observations=155 and total features=500 Original features =18 Artificial features=482) Accuracy: 90% Features selected: 4	(1)Age, (2)sex, (3)steroid, (4)antivirals, (5)fatigue, (6)malaise, (7)anorexia, liver big, (8)liver firm, (9)spleen palpable, (10)spiders, (11)ascites, (12)varices, (13)bilirubin, (14)alk phosphate, (15)sgot, (16)albumin, (17)protime, (18)histology
Mushroom (total observations=8124 and total features=500 Original features =22 Artificial features=478) Accuracy: 100% Features selected: 3	(1)Cap-shape, (2)cap surface, (3)cap-color, (4)bruises, (5)odor, (6)gill-attachment, (7)gill-spacing, (8)gill-size, (9)gill-color, (10)stalk-shape, (11)stalk-root, (12)stalk-surface-above-ring, (13)stalk-surface-below-ring, (14)stalk-color-above-ring, (15)stalk-color-below-ring, (16)veil-type, (17)veil-color, (18)ring number, (19)ring-type, (20)spore-print-color, (21)population, (22) habitat
Spambase (total observations=4601 and total features=500 Original features =57 Artificial features=443) Accuracy: 92% Features selected: 31	(1) word_freq_make, (2) word_freq_address, (3) word_freq_all, (4) word_freq_3d, (5) word_freq_our, (6) word_freq_over, (7) word_freq_remove, (8) word_freq_internet, (9) word_freq_order, (10) word_freq_mail, (11) word_freq_receive, (12) word_freq_will, (13) word_freq_people, (14) word_freq_report, (15) word_freq_addresses, (16) word_freq_free, (17) word_freq_business, (18) word_freq_email, (19) word_freq_you, (20)word_freq_credit, (21) word_freq_your, (22) word_freq_font, (23) word_freq_000, (24) word_freq_money, (25) word_freq_hp, (26) word_freq_hp1, (27) word_freq_george, (28) word_freq_650, (29) word_freq_lab, (30) word_freq_labs, (31) word_freq_telnet, (32) word_freq_857, (33) word_freq_data, (34)word_freq_415, (35) word_freq_85, (36)word_freq_technology, (37) word_freq_1999, (38) word_freq_parts, (39)word_freq_pm, (40) word_freq_direct, (41) word_freq_cs, (42) word_freq_meeting, (43) word_freq_original, (44) word_freq_project, (45) word_freq_re, (46) word_freq_edu, (47) word_freq_table, (48) word_freq_conference, (49) char_freq_;, (50) char_freq_(:, (51) char_freq_[:, (52)char_freq_, (53) char_freq_, (54) char_freq_, (55) capital_run_length_average, (56) capital_run_length_longest, (57)Capital_run_length_total

Wine (total observations=178 and total features=500 Original features =13 Artificial features=487) Accuracy: 94% Features selected: 4	(1) alcohol, (2)metalic acid, (3)ash, (4) alcalinity of ash, (5) magnesium, (6) total phenols, (7) flavanoids, (8) nonflavanoid phenols, (9) proanthocyanins, (10) color intensity, (11) hue, (12) OD280/OD315 of diluted wines, (13) Proline
Yeast (total observations=1484 and total features=500 Original features =8 Artificial features=492) Accuracy: 69% Features selected: 1	**Solution 1:** (1)mcg, (2)gvh, (3)alm, (4)mit, (5)erl, (6)pox,(7)vac,(8)nuc **Solution 2:** (1)mcg, (2)gvh, (3)alm, (4)mit, (5)erl, (6)pox,(7)vac,(8)nuc **Solution 3**: (1)mcg, (2)gvh, (3)alm, (4)mit, (5)erl, (6)pox,(7)vac,(8)nuc **Solution 4:** (1)mcg, (2)gvh, (3)alm, (4)mit, (5)erl, (6)pox,(7)vac,(8)nuc **Solution 5:** (1)mcg, (2)gvh, (3)alm, (4)mit, (5)erl, (6)pox,(7)vac,(8)nuc
Zoo (total observations=101 and total features=500 Original features =16 Artificial features=484) Accuracy: 100% Features selected: 1	(1) hair, (2) feathers, (3) eggs, (4) milk, (5) airborne, (6) aquatic, (7) predator, (8) toothed, (9) backbone, (10) breathes, (11)venoumous, (12) fins, (13) legs, (14) tail, (15) domestic, (16) catsize
Smoking (total observations=3321 and total features=285 Original features =285 Artificial features=0) Accuracy: 100% Features selected: 12	***Best to date feature subset includes the following features***: (1) Prompted awareness of brands-Regal, (2) Brand identification (masked packs)—B & H, (3) Brand identification (masked packs):--Berkeley, (4) Brand identification (masked packs)—Lambert & Butler, (5) Brand identification (masked packs)—Marlboro, (6) Brand identification (masked packs)—Regal, (7) Correct identification of Richmond, (8) Number of masked brands identified, (9) Number of Superkings marketing types seen (inc sponsorship), (10) any sibling smoke, (11) parental smoking status—both, mum, dad, neither, (12) Any close friends smoke

Chapter 5

Reconstruction of Incomplete Datasets in Cross-Sectional Studies

5.1. Introduction

Missing data is a problem that is ubiquitous to all studies and a source of multiple problems from an analytic point of view (reduced statistical power, increased type I error, bias), which we discussed in detail section 1.1.2. As we discuss in section 2.3, conventional algorithms suffer from many problems including lack of local approximation properties, curse of dimensionality, parametric degradation, and having too many free parameters. The treatment of incomplete data is an important, if not the most important step in the pre-processing of data. We propose a novel nonparametric multiple imputation (MI) algorithm GEMI. GEMI is a Generalized Regression Neural Network Ensemble (GE) comprising a number of Generalized Regression Neural Networks (GRNNs). We also developed a single imputation (SI) version of this approach—GESI. We compare our algorithms with well-known imputation algorithms on 51 real-world datasets (50 public datasets with artificially introduced missing values and a new real-life dataset with 37% missing data) for various percentage of missing values. The effectiveness of the algorithms is evaluated in terms of (i) the accuracy of output classification, (ii) interval estimation accuracy of missing value estimates and (iii) point estimation accuracy of missing value estimates. In terms of the first two criteria listed above, GEMI outperformed GESI and all the conventional imputation algorithms. The rest of the chapter is organized as follows: novelties of the proposed

algorithms in section 5.2, details of novel algorithms in section 5.3, comparative performance measurement in section 5.4, results and discussion in section 5.5, summary and conclusions in section 5.6, and future work in section 5.7.

5.2. Novelties of Proposed Algorithms: GEMI & GESI

We developed two novel missing value imputation algorithms—GEMI (GRNN-Ensemble for Multiple-Imputation) and GESI (GRNN-Ensemble for Single Imputation). Both algorithms construct an ensemble model of Generalized Regression Neural Networks (GRNN)—a well respected computational intelligence based algorithm proposed by Donald Specht in [145]. An ensemble of GRNNs is a set of different GRNNs which together solve a problem. GESI is a single imputation version of GEMI. The only difference between GEMI and GESI lies in the number of imputations that are made for any one missing value. In GESI, one replacement value is created and imputed for each missing observation. In GEMI, several missing values are independently imputed, creating multiple (m) datasets and multiple estimates, one for each replication of the imputation process. The multiple estimates are averaged over the m imputations to create a single MI estimate of the statistic of interest. To account for the uncertainty of the imputation process, MI variance estimators that incorporate both within- and between-imputation components are used to account for imputation-related variance in the overall variance of the estimate of interest. The pseudo codes of GEMI and GESI are presented in section 5.3. In this section, we discuss the advantages of these novel approaches.

First, a good imputation algorithm should use as much information as possible. The problem is that a variable X may be related both to the missing data mechanisms and to the actual outcomes of the covariates. To produce high-quality imputations for a particular variable X, the imputation model should include variables that are (a) potentially related to X, and (b) potentially related to the missing mechanism of X. A general guideline is that the imputer should use a model that is general enough to preserve any associations among variables (two-, three-, or even higher-way associations) that may be the target of subsequent analysis. Hence, our algorithms attempt to use the maximum possible information content available to yield optimal performance. Our novel algorithms (GEMI and GESI) that are used to conjointly model the data and missing data mechanisms exploit information, not only from observations, but also from presence and absence of values in other variables. This is a difficult task to accomplish. For each variable X_i, we define a corresponding indicator variable I_i, to indicate whether the value of the variable X_i is missing or not. However, adding all those indicator features comes at a price: it increases the dimensionality of the patterns. Consequently, the curse of dimensionality problem appears and the prediction performance degrades severely.

In order to ensure the inclusion of all the important predictors and interactions in the model, our algorithms (GEMI and GESI) follow a novel homogeneous ensemble framework that can cope with the curse of dimensionality. GEMI and GESI are Generalized Regression Neural Network (GRNN) ensembles where a collection of GRNNs is trained for the same task. Ensemble members are trained on different subsets of features. These separately trained GRNNs are then combined to form one unified

prediction model. There are numerous ensemble approaches available in the literature. Compared with existing ensemble learning algorithms, our proposed ensemble approach is unique in the following two aspects.

(i) **Selection of ensemble members**: The selection of an optimal feature subset and the selection of an optimal set of ensemble members are similar types of combinatorial problems. Hence, the same techniques are used for both of these problems. We reviewed these techniques and their advantages and disadvantages in sections 2.1 and 2.2 of chapter 2. Our novel algorithms (GEMI and GESI) apply our proposed feature subset selection algorithm (SAGA) to select several parsimonious feature sets with excellent prediction performances. These subsets are used to train separate GRNNs. GEMI and GESI then identify a parsimonious set of trained GRNNs (ensemble members) for the ensemble model using the SAGA. We demonstrated in chapter 4 that SAGA offers the most parsimonious and best-fitting models.

(ii) **Method of combining the outputs of ensemble members**: Weighted Voting is the most widely used and accepted strategy available for combining the outputs of several ensemble members trained independently to perform a prediction task. A weighted voting system is one in which the preferences of some voters carry more weight than the preferences of other voters. The major disadvantage of traditional static weighted voting system is that the weights for each ensemble member's vote do not depend on patterns to be learned. Recent studies show that when the performance of the ensemble members is not uniform for all patterns, the efficiency of this type of voting is affected negatively [155]. To overcome this problem, Jacobs and Jordan proposed a

dynamically selected ensemble neural network in which the combiner or the gating network (an MLP network) selects a single base classifier rather than combining the predictions of the classifiers [40]. However, it does not serve our purpose because it does not reduce the curse of dimensionality. We discovered an effective means of combining such ensemble members. In our proposed ensemble approach, the neural network ensemble consists of two layers of GRNNs. Each GRNN in the first layer (called base learner) is trained to predict the variable of interest using a distinct feature subset identified by the SAGA. The number of GRNNs in the first layer is also determined by the SAGA. There is only one GRNN in the second layer (called combiner). The second layer GRNN uses the outputs of the first layer as inputs and produces the final prediction for the unobserved items. Figure 5.1 illustrates the basic framework for the proposed GRNN ensemble. GRNN is a local approximation algorithm. To embed this feature into our own algorithms (GEMI & GESI), we used GRNN as base learners as well as the ensemble combiner. Consequently, GEMI and GESI are dynamic weighted voting methods. These methods adapt weights of base learners in a pattern-by-pattern fashion. For each instance to be classified, they assign weights to base learners proportional to their accuracy on similar cases.

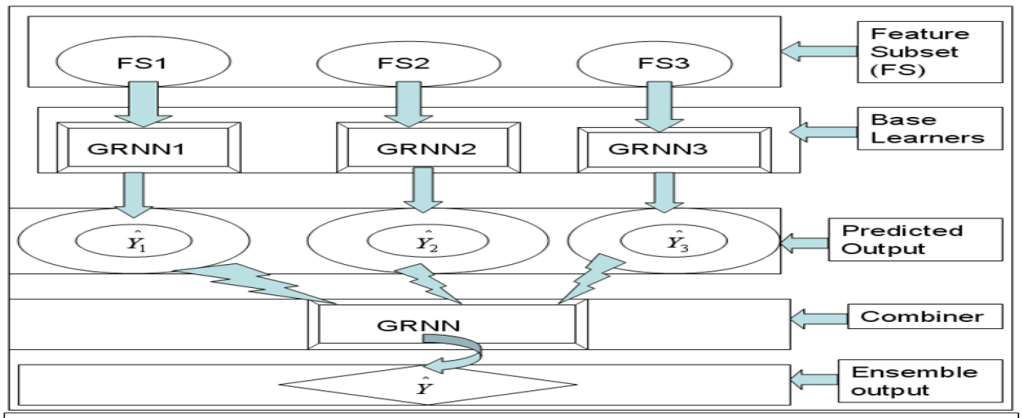

Figure 5.1: The framework of the proposed ensemble classifier

In addition to the high number of predictor variables, there is another major pitfall to be aware of when imputing missing data. Missing values are determined using complete case analysis. The sample size decreases as the percentage of missing values increases. A small sample size tends to result in a poor model fit. In order to reduce this problem, our algorithms (GEMI and GESI) imitate the iterative computation of the MCMC and EM algorithms, although these new algorithms do not use the maximum likelihood approach. GEMI and GESI repeatedly alternate between two steps, the *M* (Maximization)-step and the *E* (Expectation)-step. In the first *M*-step, GEMI and GESI fit the imputation model based on the complete cases only. Then, in the first *E*-step, missing values are imputed by the imputation model estimated at the previous *M*-step. Given a complete sample, the next-*M* step updates the imputation model. This new imputation model is then used in the next *E*-step for re-estimating missing values. This process continues until the algorithm finds a stationary repetitive state or the desired state.

Second, our proposed algorithm GEMI is that it is a multiple imputation (MI) algorithm. MI accounts for data by restoring not only the natural variability in the missing data, but also by incorporating the uncertainty caused by estimating missing data. GEMI uses Rubin's multiple imputation framework [11] that allows us to create proper multiple imputations in complex multivariate settings. To perform multiple imputations using GEMI, we create 30 training sets since a larger number of samples will give a better estimate of the random error, every time randomly selecting 70% of the available data. Each dataset is analyzed. With 30 training sets, 30 separate sets of parameter estimates (i.e. conditional mean and conditional variance) are obtained.

Individual parameter estimates are combined to get global parameters of the conditional target distribution for the missing data. We then create 30 replications of the original datasets, resulting in 30 datasets. For imputing missing values, GEMI simulates draws from the distribution of interest. The replicas of a record will differ in imputed values but not in observed values. Variation in estimates across these multiple datasets permits estimation of overall variation, including both sampling and imputation variance.

Third, both novel algorithms (GEMI and GESI) are based on a fuzzy clustering scheme. This is a non-parametric algorithm and minimizes distributional assumptions. This algorithm can easily handle data from different distributions. The datasets imputed by these algorithms will have the same distributional shape as the observed data. Another advantage of our methods (GEMI & GESI) is that they are local approximators whereas many conventional imputation algorithms such as MCMC, EM, and MLPs are global approximators. Global approximators formulate one predictive formula for the entire search space. In contrast, local approximators formulate many effective formulas in order to address local variations. In addition, since GEMI and GESI are similarity based algorithms, they are less sensitive to potential outliers. GEMI and GESI have inherited these qualities from their underlying algorithm GRNN. A brief overview of the GRNN was given in section 3.1.4.1.

Fourth, there are only a few parameters in GEMI and GESI that need to be selected. These parameters include the parameters of GRNN and the parameters of SAGA. Apart from the above mentioned parameters, there is one extra parameter (number of imputations: m) in GEMI.

GRNN has only one adjustable parameter (the smoothing parameter σ). Appendix 4*A1* shows performance impact of different σ values. The choice of SAGA parameter values and the total number of imputations (m) depends on the computational cost one is willing to incur. Hence, the task of specifying m and SAGA parameters is relatively straightforward. We determine default settings through experiments with 200 synthetic datasets (these 200 datasets were not used in the missing data imputation study). Details about how synthetic datasets were generated are described in section 3.2.2.2. Users can accept default parameter values where resource constraints permit, since default parameter values will almost always produce good results. They can also set much higher values for these parameters (parameter m and parameters of SAGA). Usually, the higher parameter values provide better results. In our missing data imputation experiments, we used default settings. A reasonable number of different parameter values were tried in order to find those values for which the code seemed to perform best overall. The relative performance of different parameter values were compared using statistical tests (Friedman Two-Way Analysis of Variance by Ranks and Comparisons of Groups or Conditions with a Control [140]). The list of default values of parameters, stored in the configuration file in our novel algorithms, are given in table 5.1.

Based on the resource constraint, we set the value of m to be 30. Ideally, the selection of the parameter value m should depend on the resource (e.g. memory and computational cost) availability, which we mentioned earlier. Our research demonstrates that precision improves by increasing the number of imputations (*m*). Our aim is to clarify this implication, so that the analyst may make an informed choice of

the parameter value. Hence, we reported the performance impacts of different m values in Appendix *5A*. It appears that the best value for parameter m is 100.

Table 5.1: Default Parameter Settings for GEMI & GESI

Parameter	Default Setting
Number of imputations *m* (a parameter of GEMI)	30
Smoothing Factor Parameter σ (For GEMI & GESI)	(Each centre's width is set to average Euclidean distance to 20 nearest members) *2
Parameters of SAGA	
Population size of SA, GA, and hill-climbing (HC) algorithm while selecting 100 optimal feature subsets	100
Population size of SA, and GA while selecting an optimal subset of base learners	≤ 100[a]
Population size of HC while selecting an optimal subset of base learners	10
Stopping criterion of the SA and GA	Stop the algorithm if the best solution does not improve in the last 200 runs.

[a] Utilize 100 (or less, if the number of parsimonious diverse feature subset solutions generated by SAGA does not reach 100).

5.3. Details of Novel Algorithms: GEMI & GESI

Computational details and the pseudo codes of GEMI and GESI are given in sections 5.3.1. Both GEMI and GESI apply SAGA (presented in chapter 4 of this thesis) to select optimal subsets of features for the imputation of missing values. SAGA sequentially employs three different search algorithms (SA, GA and Hill-Climbing algorithm) to find the optimal solution. A proper implementation of SA is tricky, since

we need to adjust temperature based on the stopping criteria. The pseudo code of SA is presented in section 5.3.2.

5.3.1 The Pseudo code of GEMI & GESI

In the implementation of GESI, we execute only the first two steps of the full procedure outlined in this section. GESI fits a single ensemble model (model-*P*) as shown in Figure 5.2. Model-*P* is a set of models $(p_1, p_2,..., p_n)$ in which each element $(p_1, p_2,..., p_n)$ is a single "ensemble" model that estimates the conditional mean of a missing value, where n is the total number of missing values.

For the implementation of GEMI, the full process has to be adhered to. GEMI fits m sets of (identical in the sense that they consist of the same feature ensembles, but non-identical in the sense that they are trained on different sets of training examples) replicas of a pair of prototype ensemble models—model-*P*: $(P_1, P_2,..., P_m)$ and model-*Q*: $(Q_1, Q_2,..., Q_m)$—as shown in Figure 5.3. Both prototype models (*P* and *Q*) consist of *n* sub models (one model for each of the *n* missing values). $(p_1, p_2,..., p_n)$ and $(q_1, q_2,..., q_n)$ represent the sub models of the prototype models *P* and *Q*, respectively. The model-*P:* $P = (p_1, p_2,..., p_n)$ predicts the conditional means of missing values, whereas the model-*Q*: $Q = (q_1, q_2,..., q_n)$ predicts the conditional variance of each missing value.

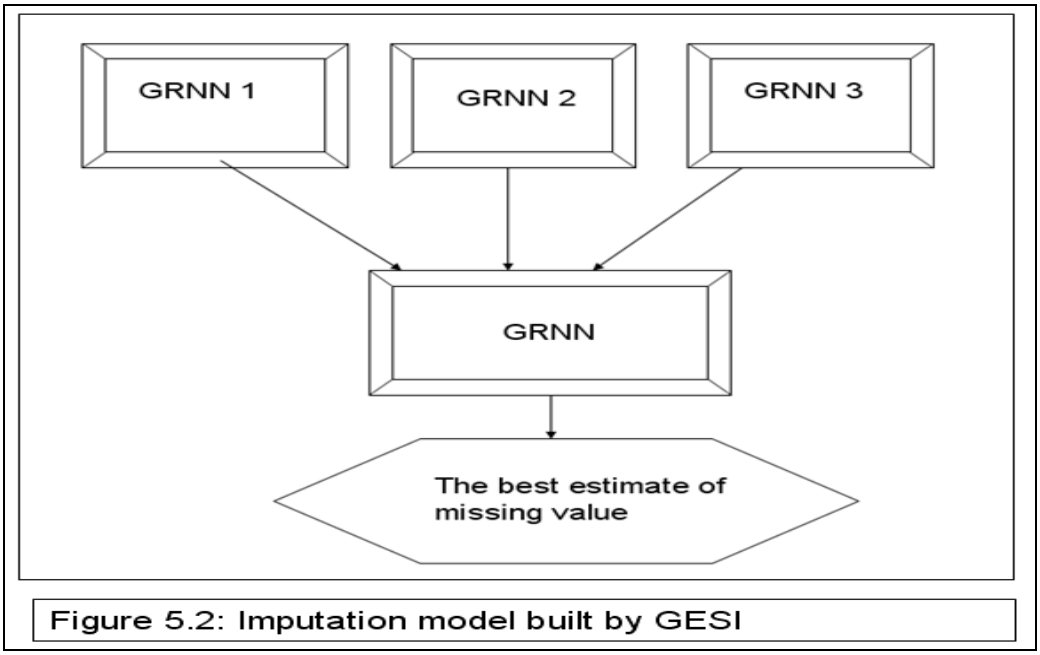

Figure 5.2: Imputation model built by GESI

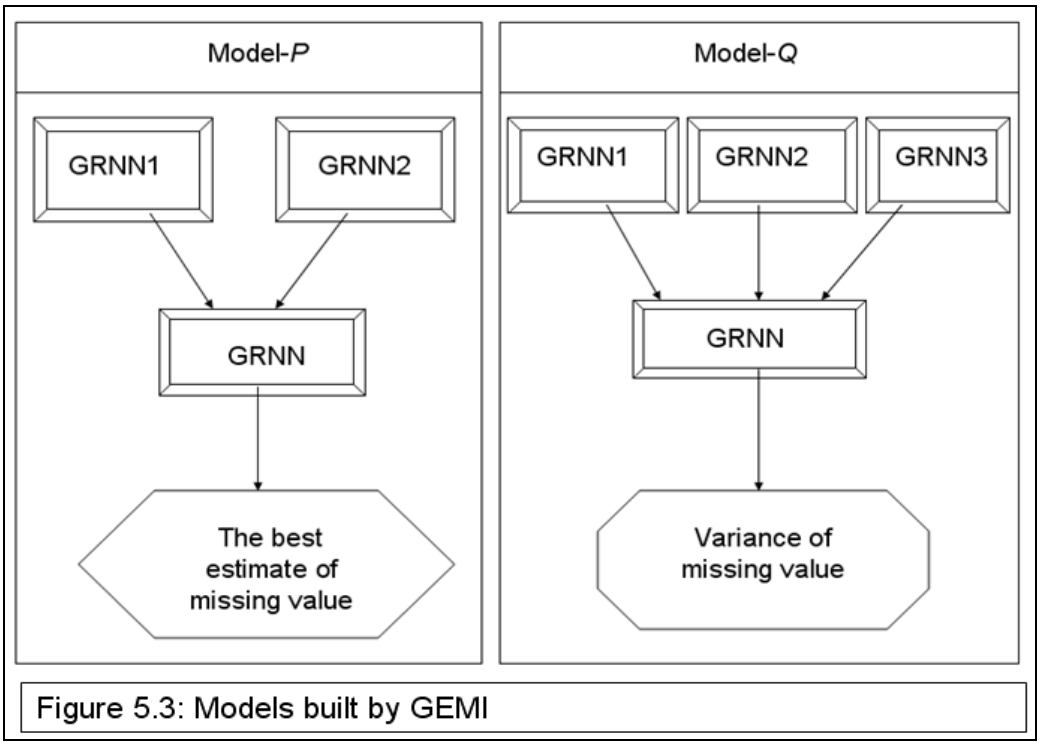

Figure 5.3: Models built by GEMI

Step 1: Normalize each variable to the range [0, 1].

Step 2: Design and construction of prototype model-*P*: As mentioned before, in Model-*P*, there is a set of models ($p_1, p_2, ..., p_n$) in which each element $(p_1, p_2, ..., p_n)$ is a single "ensemble" model (SEM_m) that estimates the conditional mean of a missing value, where n is the total number of missing values. In other words, in Model-*P*, there is a SEM_m for each missing value in the dataset, wherein the variable with the missing value is the target variable. We define an indicator variable I for each main variable X with I_{ij}=1 if X_{ij} is missing and 0 otherwise. Here, X_{ij} denotes the value of the *j*'th main variable in the i'th pattern. I_{ij} denotes the value of the *j*'th indicator variable in the i'th pattern. The main ($X = (X_1, X_2 ...)$) and indicator ($I = (I_1, I_2, ...)$) variables are considered as candidate input variables.

Step 2.1: The maximization *M* -step:

Create a loop to build a SEM_m model for each missing value:

$k = 1$; $n =$ total number of missing values

While $k \leq n$

***Step 2.1.1*:** Use SAGA to identify 100 (or less, when the total number of diverse solutions found so far is less than 100) of the best feature subsets (FS) for the target variable from the pool of candidate input variables (both the main and indicator variables) using 10 fold cross validation. The parameters of SAGA are fixed at the values shown in table 5.1.

***Step 2.1.2*:** Train a separate GRNN with each of these 100 feature subsets. This will give us 100 trained GRNNs. These GRNNs are trained to predict the output variable d_{*j}

***Step 2.1.3*:** Use SAGA to select an optimal subset of base GRNN classifiers from the pool of 100 trained GRNNs to form the ensemble model (p_k) using 10-fold cross validation. The parameters of SAGA are fixed at the values shown in table 5.1.

***Step 2.1.4*:** Complete the construction of the ensemble model (p_k) by training the combiner GRNN to predict the conditional mean of the missing value d_{ij} using the outputs of the base learners as inputs.

$k = k + 1;$

End (**While** $k \leq n$ Loop)

Given a complete sample, the next *M*-step updates the model $P = (p_1, p_2, ..., p_n)$ and then the next *E*-step re-estimates the missing values based on the updated models. The two steps are iterated until the conditional means of missing values become stable. The process of developing model *P* is further illustrated in Figure 5.4.

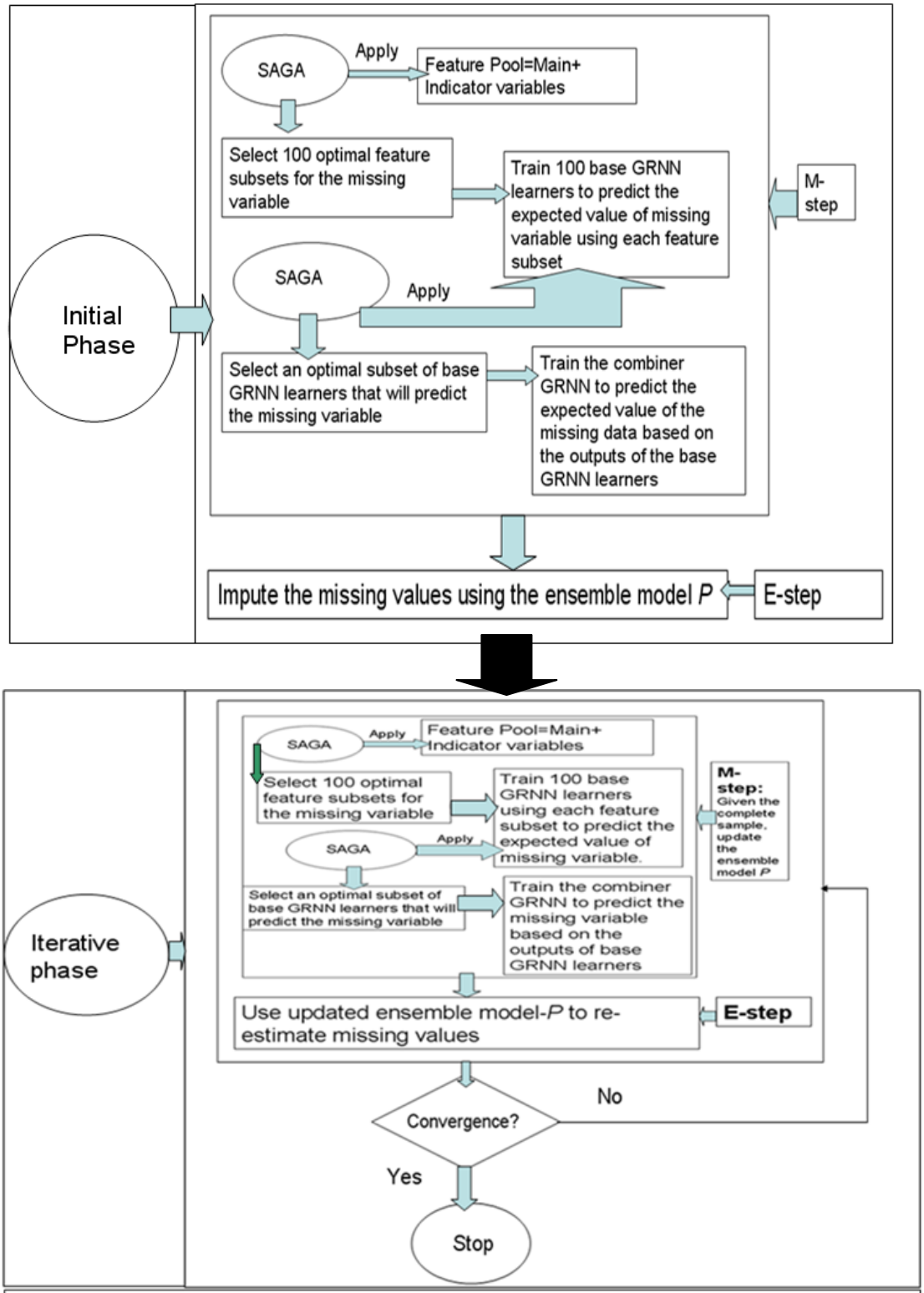

Figure 5. 4: A block diagram of the construction of ensemble model-*P* for the conditional mean of missing data

Step 3: Design and construction of prototype model-*Q*: Model-*Q* is a set of models $Q = \{q_1, q_2, ..., q_n\}$ in which each element $\{q_1, q_2 ..., q_n\}$ is a single "ensemble" model (SEM_v) that estimates the conditional variance of a missing value. In Models-*Q*, there is a SEM_v for each missing value in the dataset. In other words, there is a SEM_v model associated to each SEM_m model. To construct a SEM_v model, the following steps were executed.

K=1;

$n =$ total number of missing values

While $K <= n$

// build the model q_k

Step 3.1: Define input and output variables for the model (q_k): The target variable of the model q_k is defined as the squared residuals of the model p_k on the training set. All variables, except the variable with missing values (i.e. the target variable of model p_k), and the corresponding indicator variables are treated as candidate input variable for the model q_k.

Step 3.2: Apply SAGA to select the ensemble of base learners:

***Step 3.2.1*:** Identify 100 near-optimal but diverse feature subset solutions using 10-fold cross validation. The default parameter setting used in the experiment is listed in table 5.1.

Step 3.2.2: Train a base GRNN learner using each of the 100 feature subset solutions.

Step 3.2.3: Apply SAGA to select an optimal subset of GRNNs from the pool of 100 trained GRNNs, which will be used to form the model q_k. Table 5.1 summarizes the SAGA parameter setting for the experiments.

Step: 3.2.4: To complete the construction of the ensemble model q_k, train the combiner GRNN to predict the expected squared error of the ensemble model p_k based on the outputs of the base GRNN learners.

K=K+1;

End (While *K*<=*n*) loop

The model-*Q* construction process is illustrated in Figure 5.5.

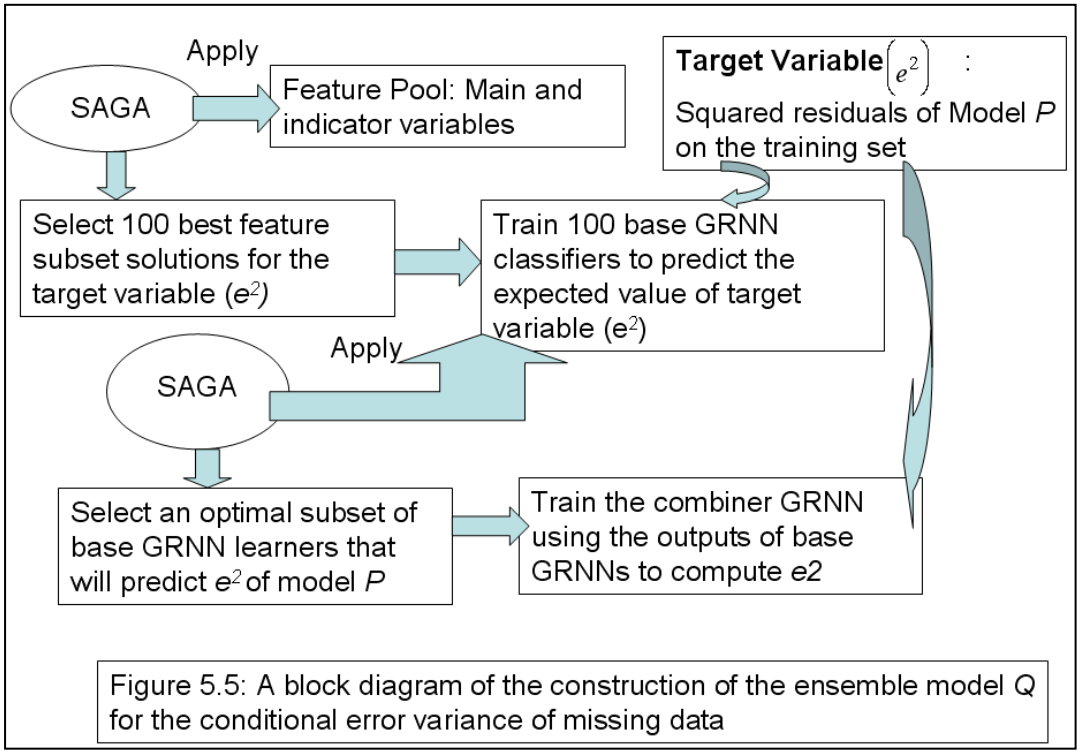

Figure 5.5: A block diagram of the construction of the ensemble model Q for the conditional error variance of missing data

Step 4: Construct multiple models for estimating parameters of missing observations: Generate *m* exact replicas $((P_1,Q_1),(P_2,Q_2),...,(P_m,Q_m))$ of the prototype models *P* and *Q*. Train the replicas using 70% training data rather than 100%. The training set of each replica was selected randomly. While training the different replicas, we did not apply SAGA for selecting ensemble members since these ensemble

models are constructed from the prototype configuration. The replicas were trained using the following steps:

//Use a loop to train replica models

K =1;

While K<=m

Step 4.1: Train each member GRNN of the replica ensemble model P_k on the training set using input (main and indicator variables) and output (variable with the missing value) data.

Step 4.2: Present the training patterns to P_k for the prediction of target variable and compute squared residuals.

Step 4.3: Train each member GRNN of the replica model Q_k using inputs (main and indicator predictor variables) and output (squared residuals of P_k) data.

K=K+1;

End Loop /* Replica models $((P_1,Q_1),(P_2,Q_2),...,(P_m,Q_m))$ are now ready for use.*/

The process of developing a pair of replicas of models P and Q is summarized in Figure 5.6.

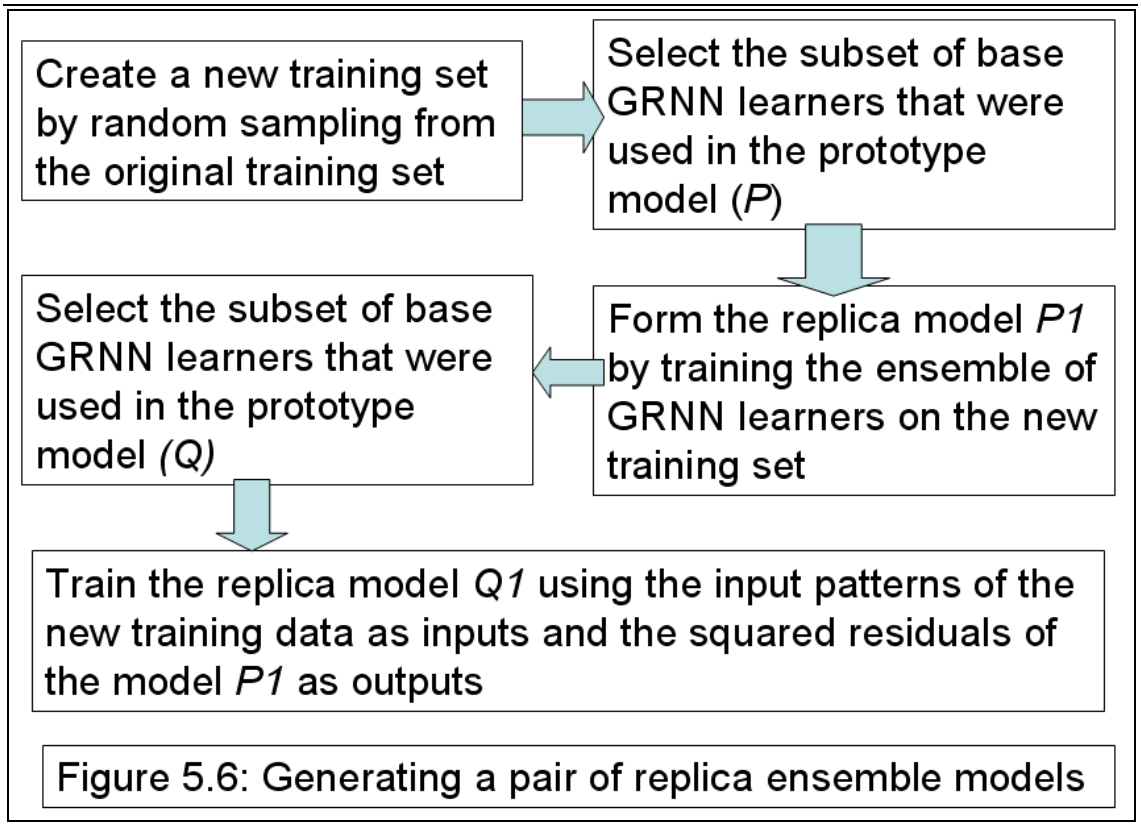

Figure 5.6: Generating a pair of replica ensemble models

Step 5: Estimate the parameters of missing data

L=1;

n=number of missing values

m= number of replica models

While $L<=n$

k =1;

While $k \leq m$

- Present the fully trained model P_k with the missing value d_L for predicting the conditional mean ($\hat{Y}_{Lk}$) of missing value (d_L).
- Present the missing value (d_L) to the trained model Q_k for predicting the conditional variance ($\hat{U}_{Lk}$) of missing value (d_L).

❖ End While loop $k \le m$

- The conditional mean of the missing value d_L is the average of the single estimates:

$$\widehat{Y}_L = \frac{1}{m}\sum_{k=1}^{m}\widehat{Y}_{Lk} \qquad (5.1)$$

- Estimate the within-imputation variance $\overline{U}_L = \frac{1}{m}\sum_{k=1}^{m}\widehat{U}_{Lk}$ (5.2)

- Estimate the between-imputation variance

$$B_L = \frac{1}{m-1}\sum_{k=1}^{m}\left(\widehat{Y}_{Lk} - \overline{Y}_L\right)^2 \qquad (5.3)$$

- The total variance T of the missing value d_L is

$$T_L = \overline{U}_L + \left(1 + \frac{1}{m}\right)B_L \qquad (5.4)$$

//We now know the mean and variance of the missing value d_L.

L=L+1;

End (of While Loop *L*<=*n*)

//We now know the conditional means and variances of all missing values.

Step 6: Replace each missing value by *m* plausible values: Replicate each record of the original dataset m times.

We then impute the missing values setting them to $\overline{Y}_L + \sqrt{T_L}R$ where R is a pseudo random number drawn from a normal distribution with mean 0 and standard deviation 1.

Steps 5 and 6 are further illustrated graphically in Figure 5.7.

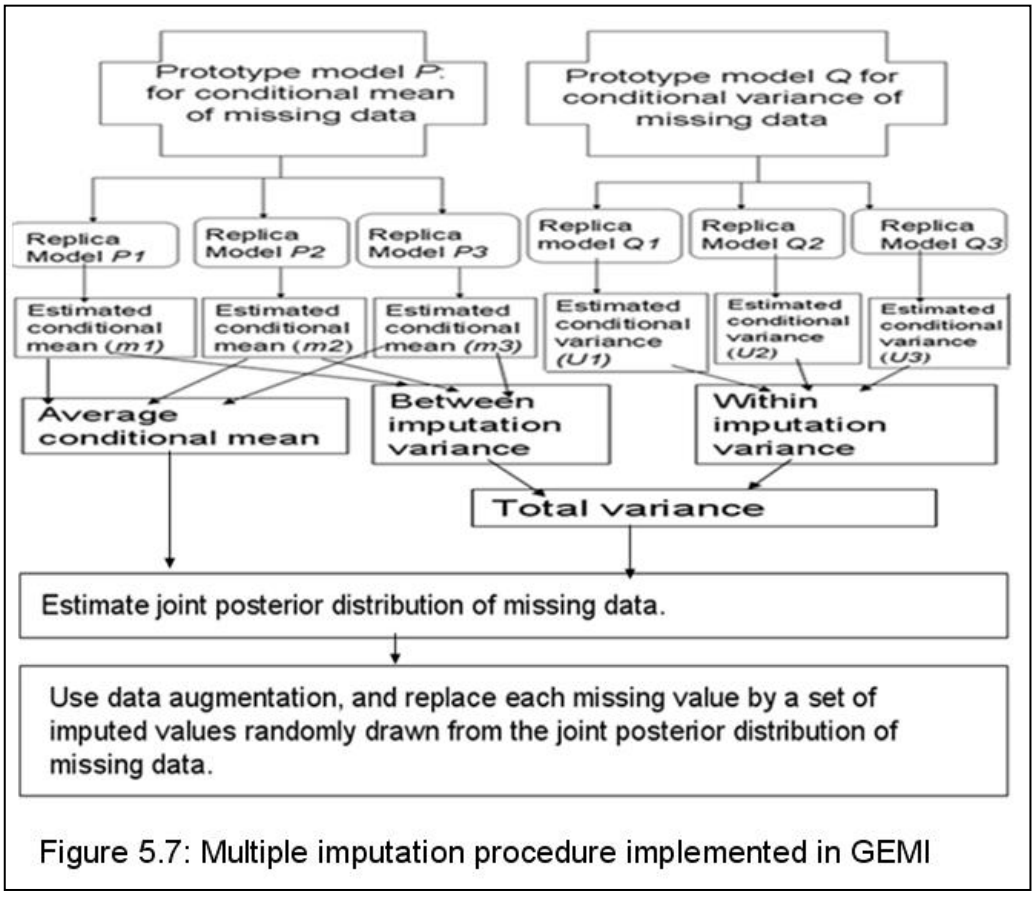

Figure 5.7: Multiple imputation procedure implemented in GEMI

5.3.2 Pseudo code of the Simulated Annealing (SA): As we mentioned before, a proper implementation of SA is tricky, since we need to adjust temperature based on the stopping criteria. **I**n the experiments of chapter 4, the SA was stopped at pre-fixed time intervals. In this study, we use total number of iterations as a stopping criterion. Here we will show how to reduce the temperature of SA with each iteration.

// Initialization Section

Step 1: Encode possible solutions in binary strings, where 1 indicates the presence of the feature (or the base classifier) and 0 indicates its absence.

Step 2: Set the initial temperature **(**T_i**):** $T_i = 100$

Step 3: Set the current temperature T_c: $T_c = T_i$

Step 4: Initialize Population: Randomly select 100 feature subset $I \leftarrow I(:100)$ solutions from the solution space for initial population.

Step 5: Estimate the prediction accuracy of each solution.

Step 6: Evaluate the fitness (fitness score is the prediction accuracy as a fraction, not a percentage) of each solution using 10-fold cross validation**:** based on binary string of each solution I, extract a new dataset D_{new} from the (normalized) original dataset D. Evaluate the fitness scores $E_o \leftarrow E_o(:100)$ of feature subsets using GRNN and store the information (feature subset solutions with feature score) in database.

// Iterative Section

Step 7: For all current feature subset vectors $I \leftarrow I(:100)$ change the bits of vectors with probability $p_m \leftarrow p_m(:100)$: $p_{mi} = 1 - E_{oi}$ (5.5)

where p_{mi} represents the mutation rate of bits within the *i*th solution and E_{oi} represents the fitness score of the *i*th solution.

Step 8: Evaluate the fitness $E_n \leftarrow E_n(:100)$ of the new candidate solutions if not already evaluated (check the database for fitness scores).

Step 9: Determine if this new solution is kept or rejected and update the database:

- If $E_n \geq E_o$, the new solution is accepted. The new solution replaces the old solution and E_o is set to E_n.
- If $E_n < E_o$, calculate the Boltzmann acceptance probability P_{accept}: $P_{accept} = \exp(-(E_o - E_n)/T_c)$ (5.6).
- Generate a random number between 0 and 1. If P_{accept} is greater than or equal to the random number, the new solution is accepted and it replaces the old one: $E_o = E_n$.

Step 10: Update the effective temperature T_c:

- If the fitness of the best solution does not improve: $T_C = T_C - 1 \qquad (5.7)$.
- If the fitness of the best solution improved: $T_C = T_i$

Step 11: If the effective temperature is greater than or equal to zero, return to step 7. Otherwise the run is finished.

5.4 Comparative Performance Analysis

We compare our algorithms (GEMI and GESI) with popular single imputation (SI) and multiple imputation (MI) algorithms: (1) EM (Expectation Maximization) SI, (2) EM MI, (3) GRNN (Generalized Regression Neural Networks) SI, (4) GRNN MI, (5) HD (Hot-Deck) SI, (6) HD MI, (7) HES (Heterogeneous Ensemble of GRNNs with Simple Averaging) SI, (8) HES (Heterogeneous Ensemble of GRNNs with Simple Averaging) MI, (9) HEW (Heterogeneous Ensemble of GRNNs with Simple Averaging) SI, (10) HEW (Heterogeneous Ensemble of GRNNs with Simple Averaging) MI, (11) HOS (Homogeneous Ensemble of GRNNs with Simple Averaging) SI, (12) HOS (Homogeneous Ensemble of GRNNs with Simple Averaging) MI, (13) HOW (Homogeneous Ensemble of GRNNs with Weighted Averaging) SI, (14) HOW (Homogeneous Ensemble of GRNNs with Weighted Averaging) MI, (15) KNN (K-Nearest Neighbour algorithm) SI, (16) KNN MI, (17) MCMC (Markov Chain Monte Carlo), (18) MLP (Multilayer Perceptrons) SI, (19) MLP MI, (20) Mean Substitution (MS), (21) RBFNN (Radial Basis Function Neural Networks) SI, (22) RBFN MI, (23) WKNN (Weighted KNN) SI, (24) WKNN MI, and (25) ZI (Zero Imputation) on 51 datasets (30 synthetic datasets + 20 public real-world datasets+1 new real-world

datasets), using 10 fold cross validation. Conventional missing data imputation algorithms are described in section 3.1.2. Section 3.2.1 presents a brief description of datasets. Public real-world datasets are complete with no missing data. These datasets were used to conduct controlled experiments. We artificially removed data using ignorable and non-ignorable missing value mechanisms at different rates of missing values. Section 3.2.2.3 details the methods adopted to simulate random and non-random missing data mechanisms. Approximately one-third of the total missing values were introduced, using each of the three missing data mechanisms (MCAR, MAR and MNAR), into each of the dataset. Then the imputation algorithms were applied separately to each incomplete dataset for imputation of missing values. We compare performance of algorithms on synthetic and public real-world datasets in three different ways:

First, each multiple imputation algorithm's performance was evaluated based on how accurately the algorithm recovers the nature of the conditional posterior distribution of the missing value. Each multiple imputation algorithm constructs a 95% confidence interval estimate of the missing value. The formula for the 95% confidence interval using the normal approximation is:

$$\left(\text{conditional mean} \pm 1.96 \sqrt{\text{conditional variance}}\right).$$

We examine the relative performance in the sense of interval estimation accuracy of multiple imputation algorithms. In general, an interval forecast is considered to be correct if the actual value falls inside the predicted 95% confidence interval. The higher the accuracy of interval identification, the better the algorithm. If the two algorithms

have the same interval estimation accuracy on a dataset, the algorithms are ranked based on the average length of the estimated confidence interval. A narrow confidence interval implies high precision. For example, if we let the confidence interval to be $(-\infty, \infty)$, then we may achieve 100% accuracy, but this interval is associated with very poor precision. Single imputation algorithms were naturally excluded from this study since they only provide a point estimate of the missing value.

Second, we compare the imputation algorithms with respect to the precision in terms of the accuracy of the missing value estimates using 10-fold cross validations on datasets. In general, the higher the accuracy values, the higher the agreement between observed and predicted data. If the variable of interest is continuous, then MAPE (Mean Absolute Percentage Error) was used to estimate the predictive accuracy of the algorithms.

$$Accuracy(\%) = 100 - MAPE(\%) = 100 - \frac{100}{N}\sum_{i=1}^{N}\frac{\left|X_i - \hat{X}_i\right|}{X_i} \quad (5.8)$$

Where N is the number of test patterns, X_i is the observed value of the variable of interest (X) at the i'th point and $\hat{X}_i$ is the predicted value.

Third, missing data inevitably affect a classifier's performance. Hence, the relative merits of these imputation algorithms were assessed by measuring the impact of imputation on the classification of outcomes. We determine the performance of an imputation algorithm based on the classification accuracy of a GRNN classifier in the imputed dataset using 10-fold cross-validation.

The Friedman test is used to test the null hypothesis that the performance is the same for all algorithms. After applying the Friedman test and noting it is significant, "Comparison of Groups or Conditions with a Control" tests (details are available in [140], p. 181) were performed in order to test the (null) hypothesis that there is no significant difference between any pair of the imputation algorithms under study.

In addition to experiments on public real world datasets, the performance was tested with a new real world dataset "Smoking Dataset". This dataset contains about 37% missing values. We did not artificially remove values from this dataset. We assess each imputation algorithm's performance on this dataset in terms of accuracy of output estimation by the GRNN classifier trained with the imputed dataset. Since all the missing values in this dataset are actually unknown to us, it was not possible to compare imputation algorithms in terms of the estimation error of missing values.

We adopted a set of measures for free and fair competition between imputation algorithms, which we discussed below.

Measures for fair comparative evaluations of imputation algorithms

The following measures were taken to organize fair competition among all imputation algorithms:

- All algorithms use our proposed feature subset selection algorithm SAGA for selecting an optimal subset of features. GEMI, GESI, HOS SI, HOS MI, HOW SI and HOW MI employ SAGA for the selection of optimal subsets of features, as

well as an optimal strategy for selecting ensemble members. All imputation algorithms use SAGA with the parameter settings reported in table 5.1:

- Although none of these algorithms (except EM and MCMC) use maximum likelihood (ML) estimation of model parameters, all algorithms (except MS, ZI, HD, KNN, and WKNN) were implemented as an (EM-style) iterative method to refine imputation models.
- Each multiple imputation (MI) algorithm replaces missing values with a set of 30 plausible values.

5.5. Results & Discussion

Figure 5.8 presents the impact of imputation of missing values by different imputation algorithms on classification accuracy for smoking dataset. Summary results on 50 datasets (synthetic and UCI datasets) are presented in Tables 5.2-5.5. These tables compare our novel imputation algorithms with well-known imputation procedures in terms of (i) the overall mean accuracy of classifying the response variable on imputed datasets with different percentage of missing data (table 5.2), (2) the overall mean accuracy of interval estimation of the missing value (table 5.3) and average length of constructed 95% confidence intervals for missing values (table 5.4), and (3) the overall mean accuracy of point estimation of the missing value (table 5.5). The Friedman tests reveal significant differences ($p < 0.05$) in the performance of imputation algorithms at 10 to 70% missing data. The classification performance of imputation algorithms based on pair wise tests is presented in Table 5.6. The pair-wise test results for other performance measures are shown in appendix tables (5B-5C).

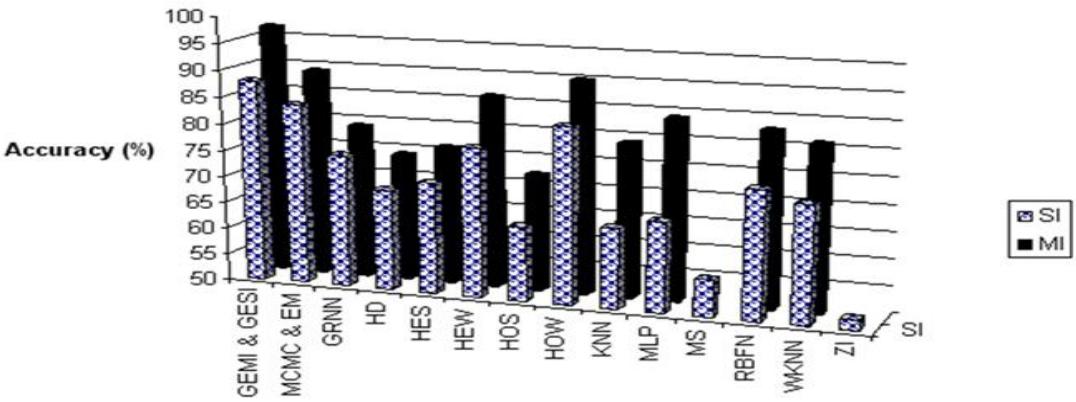

Figure 5.8: Impact of imputation on classification accuracy on smoking dataset

Table 5.2: Output classification accuracy under different levels of missing rates

	Rate of Missing Values								
Algorithm	5%	10%	20%	30%	40%	50%	60%	70%	75%
GEMI	97 (4)	96 (4)	92 (6)	85 (7)	76 (9)	69 (17)	59 (17)	55 (12)	44 (15)
GESI	97 (4)	85 (6)	75 (8)	62 (15)	55 (17)	55 (10)	51 (10)	53 (11)	40 (15)
EM	97 (4)	75 (10)	61 (13)	60 (12)	52 (20)	53 (16)	49 (9)	43 (13)	43 (13)
GRNN MI	97 (4)	79 (10)	71 (11)	68 (10)	51(19)	52 (14)	50 (11)	44 (14)	43 (16)
GRNN SI	97 (4)	80 (11)	64 (13)	57 (19)	55(17)	50 (9)	50 (9)	39 (14)	46 (15)
HD MI	97 (4)	81 (7)	62 (13)	52 (20)	49 (16)	54 (11)	50 (8)	39 (18)	43 (17)
HD SI	97 (4)	80 (8)	61 (8)	51 (20)	50(15)	51 (12)	47 (8)	39 (17)	46 (18)
HES MI	97 (4)	81 (9)	71 (11)	59 (14)	49(23)	59 (17)	50 (10)	39 (19)	38 (17)
HES SI	97 (4)	79 (10)	62 (15)	54 (15)	48 (20)	50 (18)	52 (8)	43 (15)	43 (13)
HEW MI	98 (5)	82 (7)	73 (8)	70 (13)	61(17)	50 (10)	49 (12)	41 (16)	41 (14)
HEW SI	97 (5)	81 (9)	68 (12)	59 (12)	52(16)	49 (9)	50 (10)	42 (17)	42 (18)
HOS MI	96 (4)	79 (10)	68 (17)	66 (19)	52 (17)	51 (16)	52 (9)	44 (15)	39 (17)
HOS SI	97 (4)	73 (15)	53 (16)	52 (18)	57 (18)	52 (13)	51 (10)	40 (19)	47 (15)
HOW MI	97 (5)	86 (8)	80 (9)	75 (12)	67 (13)	58 (12)	50 (11)	43 (15)	41 (17)
HOW SI	96 (4)	82 (8)	71 (14)	63 (19)	43 (16)	47 (15)	47 (9)	40 (18)	46 (15)
KNN MI	97 (4)	79 (11)	61 (13)	54 (19)	56 (16)	53 (14)	48 (11)	40 (15)	39 (16)
KNN SI	97 (4)	73 (15)	56 (18)	51 (22)	55 (18)	48 (10)	48 (8)	44 (15)	43 (15)
MCMC	97 (4)	87 (7)	80 (7)	73 (11)	62 (15)	48 (16)	50 (9)	39 (16)	42 (16)
MLP MI	96 (3)	79 (9)	65 (11)	63 (15)	57 (16)	51 (12)	52 (10)	43 (17)	41 (16)
MLP SI	97 (5)	74 (11)	56 (11)	51 (18)	54(18)	48 (12)	42 (14)	44 (12)	42 (14)
MS	98 (4)	73 (9)	54 (11)	51 (22)	50(21)	52 (9)	49 (11)	43 (14)	41 (15)
RBF MI	97 (4)	79 (12)	69 (12)	64 (11)	45 (20)	49 (13)	48 (11)	40 (14)	39 (19)
RBF SI	97 (4)	78 (11)	60 (11)	49 (17)	51(13)	48 (18)	51 (8)	38 (15)	43 (17)
WKNN MI	98 (4)	78 (11)	66 (11)	60 (21)	50 (20)	51 (12)	50 (9)	43 (15)	41 (17)
WKNN SI	97 (4)	75 (13)	59 (13)	52 (22)	55 (18)	51 (12)	52 (9)	46 (16)	39 (17)
ZI	96 (5)	69 (15)	50 (18)	48 (12)	55 (15)	46 (9)	48 (8)	41 (17)	48 (15)

Table 5.3: Interval estimation accuracy of imputation algorithms under different levels of missing rates

	Rate of Missing Values								
Algorithm	5%	10%	20%	30%	40%	50%	60%	70%	75%
GEMI	94 (5)	97 (2)	96 (4)	90 (9)	82 (11)	79 (14)	71 (15)	64 (16)	49 (15)
GRNN MI	94 (5)	90 (5)	79 (6)	73 (11)	74 (14)	54 (17)	48 (9)	49 (12)	50 (12)
HD MI	93 (4)	88 (4)	70 (10)	70 (11)	70 (10)	55 (14)	55 (13)	50 (13)	51 (14)
HES MI	94 (6)	88 (6)	73 (7)	74 (7)	58 (29)	59 (24)	54 (10)	52 (13)	47 (10)
HEW MI	94 (5)	91 (3)	80 (11)	80 (8)	72 (18)	56 (13)	54 (10)	51 (14)	50 (11)
HOS MI	93 (5)	83 (7)	67 (6)	68 (17)	60 (11)	52 (7)	50 (8)	47 (12)	50 (14)
HOW MI	94 (5)	90 (3)	87 (5)	80 (13)	74 (14)	62 (10)	56 (9)	50 (15)	48 (11)
KNN MI	95 (5)	80 (8)	71 (9)	66 (13)	59 (18)	56 (13)	55 (10)	51 (13)	52 (14)
MCMC	94 (5)	92 (2)	90 (5)	82 (12)	72 (17)	54 (12)	52 (10)	49 (14)	53 (11)
MLP MI	93 (4)	83 (7)	70 (11)	67 (15)	61 (16)	50 (14)	48 (8)	51 (13)	49 (13)
RBF MI	94 (5)	85 (7)	73 (7)	74 (17)	56 (18)	59 (22)	50 (18)	52 (13)	50 (13)
WKNN MI	93 (5)	89 (6)	71 (9)	61 (19)	67 (21)	50 (7)	45 (11)	47 (13)	51 (10)

Table 5.4: The overall width of interval estimates of missing values under different levels of missing rates

	Rate of Missing Values								
Algorithm	5%	10%	20%	30%	40%	50%	60%	70%	75%
GEMI	9 (7)	16 (11)	23 (12)	39 (14)	51 (22)	67 (18)	46 (14)	53 (21)	31(28)
GRNN MI	8 (8)	25 (9)	35 (14)	36 (16)	53 (32)	60 (39)	38 (20)	32 (27)	30 (28)
HD MI	12 (9)	35 (14)	32 (12)	28 (14)	28 (12)	60 (23)	35 (27)	23 (27)	29 (24)
HES MI	10 (8)	30 (9)	36 (20)	44 (24)	32 (25)	64 (39)	48 (29)	33 (30)	29 (27)
HEW MI	10 (8)	29 (10)	28 (9)	49 (28)	82 (36)	66 (22)	55 (34)	28 (29)	30 (26)
HOS MI	9 (9)	27 (15)	47 (24)	57 (32)	60 (19)	53 (19)	40 (30)	24 (30)	32 (25)
HOW MI	9 (8)	20 (8)	27 (14)	49 (26)	64 (30)	67 (33)	37 (26)	28 (25)	34 (28)
KNN MI	11 (9)	24 (13)	35 (27)	37 (24)	40 (25)	53 (31)	37 (30)	31 (32)	33 (28)
MCMC	12 (9)	23 (11)	25 (14)	46 (26)	60 (34)	64 (30)	36 (28)	23 (23)	21(25)
MLP MI	8 (7)	22 (15)	48 (25)	46 (30)	35 (26)	48 (30)	42 (28)	35 (29)	29 (29)
RBF MI	12 (10)	25 (11)	35 (23)	37 (23)	47 (29)	77 (36)	42 (30)	29 (28)	29 (28)
WKNN MI	11 (9)	22 (13)	31 (14)	31 (20)	44 (26)	59 (29)	41 (26)	30 (26)	41 (30)

Table 5.5: The accuracy of estimating the missing values under different levels of missing rates

	Rate of Missing Values								
Algorithm	5%	10%	20%	30%	40%	50%	60%	70%	75%
GEMI	89 (11)	81 (10)	79 (14)	76 (12)	70 (15)	62 (14)	62 (14)	55 (13)	50 (28)
GESI	89 (10)	98 (2)	94 (5)	89 (8)	79 (12)	74 (14)	64 (17)	60 (17)	44 (31)
EM	89 (11)	88 (9)	84 (9)	77 (10)	72 (12)	64 (7)	54 (8)	41 (30)	59 (31)
GRNN MI	91 (8)	73 (8)	64 (8)	63 (10)	57 (17)	50 (11)	53 (12)	46 (34)	53 (32)
GRNN SI	88 (11)	89 (9)	89 (9)	77 (11)	70 (10)	60 (13)	57 (12)	50 (29)	51 (30)
HD MI	89 (10)	66 (19)	58 (13)	61 (11)	54 (12)	50 (6)	43 (10)	53 (29)	43 (32)
HD SI	89 (12)	90 (9)	86 (10)	72 (11)	58 (14)	50 (7)	54 (15)	55 (32)	48 (29)
HES MI	86 (11)	68 (14)	63 (9)	61 (16)	62 (16)	51 (21)	44 (22)	45 (29)	50 (23)
HES SI	89 (9)	80 (16)	69 (13)	74 (15)	55 (22)	58 (13)	52 (12)	53 (31)	47 (34)
HEW MI	89 (11)	75 (11)	66 (13)	64 (13)	57 (11)	54 (17)	52 (12)	42 (28)	50 (34)
HEW SI	88 (11)	88 (6)	85 (9)	78 (13)	68 (11)	63 (11)	58 (13)	50 (35)	57 (30)
HOS MI	90 (11)	70 (13)	60 (14)	55 (14)	57 (13)	53 (8)	49 (13)	56 (33)	45 (29)
HOS SI	89 (8)	81 (11)	81 (12)	72 (14)	66 (18)	55 (15)	56 (13)	49 (28)	46 (33)
HOW MI	91 (11)	77 (8)	71 (17)	64 (16)	64 (12)	55 (17)	51 (13)	46 (32)	57 (34)
HOW SI	89 (10)	89 (9)	85 (10)	72 (11)	69 (15)	57 (13)	58 (13)	45 (29)	48 (35)
KNN MI	90 (10)	73 (11)	65 (9)	64 (15)	56 (18)	41 (17)	44 (17)	49 (33)	53 (31)
KNN SI	92 (10)	87 (7)	81 (13)	80 (13)	62 (14)	55 (18)	45 (15)	53 (29)	45 (32)
MCMC	86 (10)	77 (13)	70 (14)	64 (15)	58 (14)	55 (15)	53 (19)	56 (31)	51 (33)
MLP MI	88 (10)	74 (12)	61 (12)	56 (14)	50 (18)	56 (19)	53 (14)	50 (33)	48 (29)
MLP SI	86 (10)	86 (7)	80 (9)	77 (16)	68 (11)	60 (9)	58 (8)	41 (25)	40 (28)
MS	90 (11)	84 (13)	80 (13)	75 (18)	63 (18)	58 (21)	50 (13)	47 (30)	50 (33)
RBF MI	88 (9)	73 (13)	66 (12)	66 (18)	60 (14)	46 (16)	47 (18)	46 (29)	55 (23)
RBF SI	87 (13)	83 (10)	88 (11)	75 (17)	74 (13)	62 (11)	62 (14)	51 (33)	57 (31)
WKNN MI	87 (10)	75 (14)	64 (10)	67 (12)	54 (14)	48 (12)	47 (15)	57 (30)	50 (30)
WKNN SI	89 (9)	88 (7)	85 (8)	73 (12)	69 (9)	61 (10)	50 (13)	57 (29)	47 (34)
ZI	85 (11)	79 (15)	67 (17)	56 (22)	53 (21)	54 (19)	57 (19)	52 (29)	53 (30)

Table 5.6: Pairwise comparisons among imputation algorithms in terms of output classification accuracy

Rank	Algorithm	Significantly outperformed algorithms
With about 10% Missing Values		
1	GEMI	(1)MCMC, (2)HOW MI, (3)GESI, (4)HOW SI, (5)HEW MI , (6)HES SI, (7)GRNN MI, (8) HD MI, (9)WKNN_MI, (10)HOS SI, (11)HES MI, (12)HEW SI, (13)RBF MI, (14) HD SI, (15)KNN MI, (16) RBF SI, (17) EM, (18) WKNN SI, (19) GRNN SI, (20) MLP MI, (21) MLP SI, (22) HOS MI, (23) KNN SI, (24) MS, (25) ZI
2	MCMC	(1)HEW MI, (2)HES SI, (3)GRNN MI, (4) HD MI, (5)WKNN_MI, (6)HOS SI, (7)HES MI, (8)HEW SI, (9)RBF MI, (10) HD SI, (11)KNN MI, (12) RBF SI, (13) EM, (14) WKNN SI, (15) GRNN SI, (16) MLP MI, (17) MLP SI, (18) HOS MI, (19) KNN SI, (20) MS, (21) ZI
3	HOW MI	(1)RBF MI, (2) HD SI, (3)KNN MI, (4) RBF SI, (5) EM, (6) WKNN SI, (7) GRNN SI, (8) MLP MI, (9) MLP SI, (10) HOS MI, (11) KNN SI, (12) MS, (13) ZI
4	GESI	(1)KNN MI, (2) RBF SI, (3) EM, (4) WKNN SI, (5) GRNN SI, (6) MLP MI, (7) MLP SI, (8) HOS MI, (9) KNN SI, (10) MS, (11) ZI
5	HOW SI	(1) GRNN SI, (2) MLP MI, (3) MLP SI, (4) HOS MI, (5) KNN SI, (6) MS, (7) ZI
6	HEW MI, HES SI, GRNN MI, HD MI, WKNN MI, HOS SI	(1) MLP SI, (2) HOM_MI1, (3) KNN SI, (4) MS, (5) ZI
7	HES MI, HEW SI, RBF MI, HD SI, KNN MI, RBF SI, EM, WKNN SI	(1) HOS MI, (2) KNN SI, (3) MS, (4) ZI
8	GRNN SI, MLP MI, MLP SI, HOS MI	(1) KNN SI, (2) MS, (3) ZI
9	KNN SI	(1) MS, (2) ZI
10	MS	(1) ZI
11	ZI	0
With about 20% Missing values		
1	GEMI	(1) GESI, (2)HOW SI, (3)HEW MI, (4) GRNN MI, (5) HES MI, (6) RBF MI, (7) HEW SI, (8) MLP MI, (9) WKNN MI, (10) HES SI, (11) HD MI, (12) KNN MI, (13) EM, (14) HOS MI, (15) GRNN SI, (16) RBF SI, (17) WKNN SI, (18) HD SI, (19) KNN SI, (20) MLP SI, (21) HOS SI, (22) MS, (23)ZI
2	MCMC, HOW MI	(1) HES MI, (2)GRNN MI, (3)HEW MI,(4) RBF MI, (5) HEW SI, (6) MLP MI, (7) WKNN MI, (8)

		HES SI, (9) HD MI, (10) KNN MI, (11) EM, (12) HOS MI, (13) GRNN SI, (14) RBF SI, (15) WKNN SI, (16) HD SI, (17) KNN SI, (18) MLP SI, (19) HOS SI, (20) MS, (21)ZI
3	GESI	(1) HEW SI, (2) MLP MI, (3) WKNN MI, (4) HES SI, (5) HD MI, (6) KNN MI, (7) EM, (8) HOS MI, (9) GRNN SI, (10) RBF SI, (11) WKNN SI, (12) HD SI, (13) KNN SI, (14) MLP SI, (15) HOS SI, (16) MS, (17)ZI
4	HOW SI, HEW MI, GRNN MI	(1) WKNN MI, (2) HES SI, (3) HD MI, (4) KNN MI, (5) EM, (6) HOS MI, (7) GRNN SI, (8) RBF SI, (9) WKNN SI, (10) HD SI, (11) KNN SI, (12) MLP SI, (13) HOS SI, (14) MS, (15)ZI
5	HES MI	(1) HES SI, (2) HD MI, (3) KNN MI, (4) EM, (5) HOS MI, (6) GRNN SI, (7) RBF SI, (8) WKNN SI, (9) HD SI, (10) KNN SI, (11) MLP SI, (12) HOS SI, (13) MS, (14)ZI
6	RBF MI	(1) EM, (2) HOS MI, (3) GRNN SI, (4) RBF SI, (5) WKNN SI, (6) HD SI, (7) KNN SI, (8) MLP SI, (9) HOS SI, (10) MS, (11)ZI
7	HEW SI, MLP MI, WKNN MI	(1) GRNN SI, (2) RBF SI, (3) WKNN SI, (4) HD SI, (5) KNN SI, (6) MLP SI, (7) HOS SI, (8) MS, (9)ZI
8	HES SI, HD MI, KNN MI, EM, HOS MI	(1) RBF SI, (2) WKNN SI, (3) HD SI, (4) KNN SI, (5) MLP SI, (6) HOM_SI1, (7) MS, (8)ZI
9	GRNN SI, RBF SI	(1) WKNN SI, (2) HD SI, (3) KNN SI, (4) MLP SI, (5) HOS SI, (6) MS, (7)ZI
10	WKNN SI	(1) HD SI, (2) KNN SI, (3) MLP SI, (4) HOS SI, (5) MS, (6)ZI
11	HD SI	(1) KNN SI, (2) MLP SI, (3) HOS SI, (4) MS, (5)ZI
12	KNN SI	(1) MLP SI, (2) HOS SI, (3) MS, (4)ZI
13	MLP SI	(1) HOS SI, (2) MS, (3)ZI
14	HOS SI	(1) MS, (2)ZI
15	MS	(1)ZI
16	ZI	0
With about 30% Missing Values		
1	GEMI	(1)HEW MI, (2) GRNN MI, (3) HOS MI, (4) RBF MI, (5) GESI, (6) MLP MI, (7) HOW SI, (8) HES SI, (9) WKNN SI, (10) MS, (11) KNN MI, (12) KNN SI, (13) EM, (14) HES MI, (15) HD MI, (16) GRNN SI, (17) WKNN MI, (18) HEW SI, (19) HOS SI, (20) HD SI, (21) MLP SI, (22) RBF SI, (23) ZI
2	MCMC, HOW MI	(1) GESI, (2) MLP MI, (3) HOW SI, (4) HES SI, (5) WKNN SI, (6) MS, (7) KNN MI, (8) KNN SI, (9) EM, (10) HES MI, (11) HD MI, (12) GRNN SI,

		(13) WKNN MI, (14) HEW SI, (15) HOS SI, (16) HD SI, (17) MLP SI, (18) RBF SI, (19) ZI
3	HEW MI	(1) HES SI, (2) WKNN SI, (3) MS, (4) KNN MI, (5) KNN SI, (6) EM, (7) HES MI, (8) HD MI, (9) GRNN SI, (10) WKNN MI, (11) HEW SI, (12) HOS SI, (13) HD SI, (14) MLP SI, (15) RBF SI, (16) ZI
4	GRNN MI	(1) MS, (2) KNN MI, (3) KNN SI, (4) EM, (5) HET_MI1, (6) HD MI, (7) GRNN SI, (8) WKNN MI, (9) HET_SI2, (10) HOM_SI1, (11) HD SI, (12) MLP SI, (13) RBF SI, (14) ZI
5	HOS MI	(1) KNN SI, (2) EM, (3) HES MI, (4) HD MI, (5) GRNN SI, (6) WKNN MI, (7) HEW SI, (8) HOS SI, (9) HD SI, (10) MLP SI, (11) RBF SI, (12) ZI
6	RBF MI	(1) EM, (2) HES MI, (3) HD MI, (4) GRNN SI, (5) WKNN MI, (6) HEW SI, (7) HOS SI, (8) HD SI, (9) MLP SI, (10) RBF SI, (11) ZI
7	GESI	(1) HD MI, (2) GRNN SI, (3) WKNN MI, (4) HEW SI, (5) HOS SI, (6) HD SI, (7) MLP SI, (8) RBF SI, (9) ZI
8	MLP MI	(1) GRNN SI, (2) WKNN MI, (3) HEW SI, (4) HOS SI, (5) HD SI, (6) MLP SI, (7) RBF SI, (8) ZI
9	HOW SI, HES SI, WKNN SI, MS	(1) WKNN MI, (2) HEW SI, (3) HOS SI, (4) HD SI, (5) MLP SI, (6) RBF SI, (7) ZI
10	KNN MI, KNN SI	(1) HEW SI, (2) HOS SI, (3) HD SI, (4) MLP SI, (5) RBF SI, (6) ZI
11	EM, HES MI, HD MI, GRNN SI, WKNN MI	(1) HOS SI, (2) HD SI, (3) MLP SI, (4) RBF SI, (5) ZI
12	HEW SI, HOS SI	(1) HD SI, (2) MLP SI, (3) RBF SI, (4) ZI
	HD SI	(1) MLP SI, (2) RBF SI, (3) ZI
13	MLP SI	(1) RBF SI, (2) ZI
14	RBF SI	(1) ZI
15	ZI	0
With about 40% Missing values		
1	GEMI	(1) HOW MI, (2) MCMC, (3) HEW MI, (4) MLP MI, (5) GESI, (6) GRNN SI, (7) KNN MI, (8) MLP SI, (9) WKNN SI, (10) KNN SI, (11) HOS SI, (12) ZI, (13) HEW SI, (14) RBF SI, (15) HD MI, (16) EM, (17) HOS MI, (18) HES MI, (19) HES SI, (20) GRNN MI, (21) RBF MI, (22) HD SI, (23) WKNN MI, (24) MS, (25) HOW SI
2	HOW MI	(1) HOS SI, (2) ZI, (3) HEW SI, (4) RBF SI, (5) HD MI, (6) EM, (7) HOS MI, (8) HES MI, (9) HES SI, (10) GRNN MI, (11) RBF MI, (12) HD SI, (13) WKNN MI, (14) MS, (15) HOW MI

3	MCMC	(1) ZI, (2) HEW SI, (3) RBF SI, (4) HD MI, (5) EM, (6) HOS MI, (7) HES MI, (8) HES SI, (9) GRNN MI, (10) RBF MI, (11) HD SI, (12) WKNN MI, (13) MS, (14) HOW SI
4	HEW MI	(1) HEW SI, (2) RBF SI, (3) HD MI, (4) EM, (5) HOS MI, (6) HES MI, (7) HES SI, (8) GRNN MI, (9) RBF MI, (10) HD SI, (11) WKNN MI, (12) MS, (13) HOW SI
5	MLP MI	(1) RBF MI, (2) HD SI, (3) WKNN MI, (4) MS, (5) HOW SI
6	GESI, GRNN SI, KNN MI, MLP SI, WKNN SI, KNN SI, HOS SI, ZI	(1) HD SI, (2) WKNN MI, (3) MS, (4) HOW SI
7	HEW SI, RBF SI	(1) WKNN MI, (2) MS, (3) HOW SI
8	HD MI, EM, HOS MI	(1) MS, (2) HOW SI
9	HES MI, HES SI, GRNN MI, RBF MI, HD SI, WKNN MI, MS	(1) HOW SI
10	HOW SI	0
	With about 50% Missing values	
1	GEMI	(1) HES MI, (2) GESI, (3) HEW SI, (4) RBF MI, (5) GRNN SI, (6) MLP SI, (7) KNN SI, (8) MCMC, (9) RBF SI, (10) HEW MI, (11) HES SI, (12) HD MI, (13) KNN MI, (14) EM, (15) HOW SI, (16) GRNN MI, (17) HD SI, (18) MLP MI, (19) WKNN MI, (20) HOS SI, (21) MS, (22) HOS MI, (23) ZI
2	HOW MI	(1) KNN MI, (2) EM, (3) HOW SI, (4) GRNN MI, (5) HD SI, (6) MLP MI, (7) WKNN MI, (8) HOS SI, (9) MS, (10) HOS MI, (11) ZI
3	HES MI	(1) GRNN MI, (2) HD SI, (3) MLP MI, (4) WKNN MI, (5) HOS SI, (6) MS, (7) HOS MI, (8) ZI
4	GESI, HEW SI, RBF MI, GRNN SI, MLP SI, KNN SI	(1) MS, (2) HOS MI, (3) ZI
5	MCMC, RBF SI, HEW MI, HES SI, HD MI, KNN MI, EM	(1) HOS MI, (2) ZI
6	HOW SI, GRNN MI, HD SI, MLP MI, WKNN MI, WKNN SI, HOS SI, MS	(1) ZI
7	ZI	0

	With about 60% Missing values	
1	GEMI	(1) ZI, (2) GESI, (3) HOW MI, (4) HOW SI, (5) GRNN MI, (6) GRNN SI, (7) MLP MI, (8) WKNN MI, (9) RBF SI, (10) HOS SI, (11) MS, (12) HOS MI
2	MCMC, HEW MI, HES MI, HEW SI, HES SI, HD MI, RBF MI, HD SI, KNN MI, EM, MLP SI, WKNN SI, KNN SI, ZI	(1) HOS MI
3	GESI, HOW MI, HOW SI, GRNN MI, GRNN SI, MLP MI, WKNN MI, RBF SI, HOS SI, MS, HOS MI	0
	With about 70% Missing values	
1	GEMI	(1) HOW MI, (2) MCMC, (3) HEW MI, (4) MLP MI, (5) GESI, (6) GRNN SI, (7) KNN MI, (8) MLP SI, (9) WKNN SI, (10) KNN SI, (11) HOS SI, (12) ZI, (13) HEW SI, (14) RBF SI, (15) HD MI, (16) EM, (17) HOS MI, (18) HES MI, (19) HES SI, (20) GRNN MI, (21) RBF MI, (22) HD SI, (23) WKNN MI, (24) MS, (25) HOW SI
2	HOW MI, MCMC, HEW MI, MLP MI, GESI, GRNN SI, KNN MI, MLP SI, WKNN SI, KNN SI, HOM_SI1, ZI, HEW SI, RBF SI, HD MI, EM, HOS MI, HES MI, HES SI, GRNN MI, RBF MI, HD SI, WKNN MI, MS, HOW SI	0

Our results lead to the following insights about the imputation algorithms:

- The rates of missing values affect the performance of the imputation algorithms.

There was no significant difference in the performance of algorithms when the

percentage of missing values is either very low (not more than 5%) or very high (above 75%). Thus, it would appear that a difference in the performance of imputation algorithms develops when the percentage of missing values is not too high or too low.

- Our results reveal that GRNN classifier has the highest mean accuracy across all levels of missing data when the classifier is trained on the dataset imputed by GEMI (figure 5.8, tables 5.2 and 5.6). GEMI offers the best performance when the percentage of missing data is between about 10% and 70%. Within this range of missing values, GEMI outperformed all imputation algorithms in terms of the two criteria mentioned earlier: (i) the accuracy of output classification, and (ii) the interval estimation accuracy of missing data.

- In terms of the third criterion which relates to the accuracy of the missing value estimates, GESI has significantly outperformed all imputation algorithms (tables 5.2 and 5.6). In terms of classification accuracy, GESI significantly outperformed all single imputation algorithms (figure 5.8, table 5.2, and table 5.6).

- The "high-level" primary goals of data mining are output prediction and clustering. It is important to bear in mind that the fundamental goal of missing data imputation is to improve and facilitate the practice of these tasks. Imputation is generally not undertaken for its own sake. Both single imputation (SI) and multiple imputation (MI) approaches have been subject to criticisms. According to the critics, SI substantially underestimates uncertainty about the missing value, whereas MI overestimates the variance. However, our empirical results suggest that the performance of imputation algorithms in terms of their classification accuracy is relatively better for the MI than for the SI (figure 5.8, tables 5.2, and 5.6). For instance, the performance of GEMI is better than that of GESI. Similarly, MCMC, HOW MI,

HOS MI, HEW MI, HES MI, GRNN MI, RBF MI , HD MI, KNN MI, WKNN MI, , MLP MI, are generally better than EM SI, HOW SI, HOS SI, HEW SI, HES SI, GRNN SI, RBF SI, HD SI, KNN SI, WKNN SI, and MLP SI, respectively.

- In terms of classification accuracy of the response variable on the imputed data, GEMI, MCMC and HOW MI are the top three imputation algorithms that achieved the first two places across the varying levels of missing data (table 5.2 and 5.6). Similarly, in the sense of interval estimation accuracy of the missing data, GEMI, MCMC, and HOW MI are the top imputation algorithms (appendix 5B). In the sense of point estimation of the missing data (i.e. the accuracy of the missing value estimates), the top imputation algorithms include GESI, HOW SI, HD SI, GRNN SI, EM, WKNN SI, HEW SI, and RBF SI (appendix 5C). These results indicate a direct relation between the classification accuracy of the response variable and the accuracy of constructed confidence intervals for missing values, since in both cases the best algorithms are the same. The results also demonstrate that the accuracy of the missing value estimates is not a good measure for reflecting the goodness of imputations since no top imputation algorithm in terms of missing value estimation accuracy achieved good classification results. It is also interesting to note that the single imputation algorithms achieved the best missing value estimation accuracy, while the multiple imputation algorithms achieved the best classification accuracy of the response variable. This is because multiple imputation algorithms simulate the entire joint distribution of the unknown values. In contrast, single imputation algorithms estimate only the conditional mean of the missing value so that the width of the confidence interval is zero at the imputed value that leads to over-fitting or over-optimization. For all these reasons, single imputation algorithms typically offer higher accuracy in the determination of the

missing data, than do any of the multiple imputation algorism; but provides lower accuracy in downstream analyses.

5.6. Summary and Conclusions

We have presented a multiple imputation algorithm GEMI and a single imputation algorithm GESI. Both algorithms use the Generalized Regression Neural Network Ensemble. We tested new algorithms on 51 real-world datasets. All simulation results show the advantages of GEMI as compared with the conventional algorithms. However, we note that GEMI has heavy memory storage requirements and is expensive computationally. GEMI draws multiple samples from the training set in order to calculate the conditional posterior distribution of the missing value and then the initial training set is augmented several times. GEMI is an instance-based algorithm that stores all instances of the training sample in a memory in order to use them when needed. In addition, GEMI employs a relatively expensive feature subset selection algorithm SAGA to identify not only the good subsets of features, but also an optimal subset of ensemble members. Moreover, GEMI resorts to an EM-style iterative procedure to refine the imputation models. Thus, fitting a joint distribution and generating multiple imputations using GEMI can greatly increase the computational requirements, both in terms of processor speed and storage.

To ensure feasibility with respect to time and resource constraints, one can play with parameter values (the parameters of SAGA and the parameter m) accepting the trade-off between precision and cost. It is found that GEMI is better than other multiple imputation (and single imputation) algorithms, whereas GESI is better than other single

imputation algorithms. Therefore, in a given value of m, GEMI will perform better than the existing ones no matter what value the parameter m takes. Similarly, no matter how much time we give SAGA, SAGA can help us choose a better ensemble, since in our feature subset selection experiments (chapter 4) , SAGA came up with better feature subset solutions compared to conventional search algorithms within all given time frames.

Proper handling of missing values is essential in all analyses. Existing fast but inaccurate imputation methods can hinder downstream analysis of the dataset as they frequently destroy the original distribution of the dataset. Although using GEMI is relatively computationally expensive with associated intensive memory requirements, the significantly better results justify its use. The generation of high-quality imputations always has a greater priority than computational complexity. Besides, modern computers are powerful enough to handle this type of computationally intensive application, for all but the largest datasets.

5.7 Future Work

Our proposed algorithm GEMI uses Rubin's multiple imputation approach. Although Rubin's multiple imputation framework is admittedly rather popular approach, there are other approaches (e.g. Gibbs sampler, fractal hot deck imputation etc.) which can be used to produce the multiple imputes. In the near future, we will specifically examine these approaches to determine which are practical and usable and which are less practical and best forgotten.

The proposed algorithm (GEMI) is designed to be used with cross-sectional data. However, we strongly believe that GEMI can also be used to impute stationary time series data. In the near future, we will investigate this possibility.

Appendix 5A: Impact of number of imputations (*m*) on the performance of GEMI

With about 5% missing values			**With about 10% missing values**			**With about 20% missing values**		
m	Classification Accuracy (%)	Rankings (based on test results)	*m*	Classification Accuracy (%)	Rankings (based on test results)	*m*	Classification Accuracy (%)	Rankings (based on test results)
2	96 (4)	3	2	89 (6)	5	2	84 (9)	5
5	98 (3)	2	5	92 (3)	4	5	87 (8)	4
10	96 (4)	1	10	96 (3)	3	10	88 (8)	3
20	96 (2)	1	20	96 (3)	2	20	90 (5)	2
30	98 (2)	1	30	96 (3)	1	30	94 (5)	1
50	96 (3)	1	50	96 (3)	1	50	94 (3)	1
100	99 (3)	1	100	97 (3)	1	100	95 (3)	1
200	98 (3)	1	200	97 (3)	1	200	95 (3)	1
With about 30% missing values			**With about 40% missing values**			**With about 50% missing values**		
m	Classification Accuracy (%)	Rankings (based on test results)	*m*	Classification Accuracy (%)	Rankings (based on test results)	*m*	Classification Accuracy (%)	Rankings (based on test results)
2	78 (11)	3	2	82 (13)	4	2	68 (14)	5
5	82 (12)	5	5	82 (12)	3	5	71 (15)	3
10	82 (13)	4	10	85 (8)	2	10	76 (9)	4
20	85 (9)	4	20	87 (8)	3	20	75 (7)	3
30	91 (7)	2	30	91 (6)	2	30	82 (7)	2
50	90 (6)	2	50	92 (5)	2	50	82 (7)	1
100	92 (6)	1	100	93 (5)	1	100	83 (6)	1
200	93 (6)	1	200	93 (5)	1	200	83 (6)	1
With about 60% missing values			**With about 70% missing values**			**With about 75% missing values**		
m	Classification Accuracy (%)	Rankings (based on test results)	*m*	Classification Accuracy (%)	Rankings (based on test results)	*m*	Classification Accuracy (%)	Rankings (based on test results)
2	60 (16)	2	2	55 (20)	2	2	46 (12)	1
5	67 (12)	3	5	54 (18)	1	5	46 (18)	1
10	62 (16)	2	10	63 (12)	1	10	50 (16)	1
20	70 (9)	1	20	66 (12)	1	20	51(14)	1
30	76 (7)	1	30	68 (11)	1	30	48 (14)	1
50	80 (8)	1	50	68 (11)	1	50	52 (14)	1
100	80 (7)	1	100	73 (11)	1	100	52 (14)	1
200	80 (7)	1	200	73 (11)	1	200	52 (15)	1

Appendix 5B: Pairwise comparisons among imputation algorithms in terms of interval estimation of missing data

Rank	Algorithm	Significantly outperformed algorithms
		With about 10% missing values
1	GEMI	(1)MCMC, (2) HOW MI, (3) HEW MI, (4) GRNN MI, (5) HD MI, (6) WKNN MI, (7) HES MI, (8) RBF MI, (9) MLP MI, (10) HOS MI, (11) KNN MI
2	MCMC	(1) GRNN MI, (2) HD MI, (3) WKNN MI, (4) HES MI, (5) RBF MI, (6) MLP MI, (7) HOS MI, (8) KNN MI
3	HOW MI, HEW MI, GRNN MI, HD MI, WKNN MI	(1) HES MI, (2) RBF_MI, (3) MLP MI, (4) HOS MI, (5) KNN MI
4	HES MI	(1) RBF MI, (2) MLP MI, (3) HOS MI, (4) KNN MI
5	RBF MI	(1) MLP MI, (2) HOS MI, (3) KNN MI
6	MLP MI	(1) HOS MI, (2) KNN MI
7	HOS MI	(1) KNN MI
8	KNN MI	0
	With about 20% missing values	
1	GEMI	(1) HOW MI, (2) HEW MI, (3) GRNN MI, (4) HES MI, (5) RBF MI, (6) WKNN MI, (7) HD MI, (8) KNN MI, (9) MLP MI, (10) HOS MI
2	MCMC, HOW MI	(1) HEW MI, (2) GRNN MI, (3) HES MI, (4) RBF MI, (5) WKNN MI, (6) HD MI, (7) KNN MI, (8) MLP MI, (9) HOS MI
3	HEW MI	(1) GRNN MI, (2) HES MI, (3) RBF MI, (4) WKNN MI, (5) HD MI, (6) KNN MI, (7) MLP MI, (8) HOS MI
4	GRNN MI	(1) HES MI, (2) RBF MI, (3) WKNN MI, (4) HD MI, (5) KNN MI, (6) MLP MI, (7) HOS MI
5	HES MI, RBF MI	(1) WKNN MI, (2) HD MI, (3) KNN MI, (4) MLP MI, (5) HOS MI
6	WKNN MI	(1) HD MI, (2) KNN MI, (3) MLP MI, (4) HOS MI
7	HD MI	(1) KNN MI, (2) MLP MI, (3) HOS MI
8	KNN MI	(1) MLP MI, (2) HOS MI
9	MLP MI	(1) HOS MI
10	HOS MI	0
	With about 30% missing values	
1	GEMI	(1) MCMC, (2) HOW MI, (3) HEW MI, (4) HOS MI, (5)HES MI, (6) GRNN MI, (7) HD MI, (8) RBF MI, (9) MLP MI, (10)KNN MI, (11) WKNN MI
2	MCMC	(1) HOS MI, (2)HES MI, (3) GRNN MI, (4) HD MI, (5) RBF MI, (6) MLP MI, (7)KNN MI, (8) WKNN MI
3	HOW MI, HEW MI	(1) GRNN MI, (2) HD MI, (3) RBF MI, (4) MLP MI, (5)KNN MI, (6) WKNN MI

4	HOS MI	(1) RBF MI, (2) MLP MI, (3)KNN MI, (4) WKNN MI
5	HES MI, GRNN MI, HD MI, RBF MI	(1) MLP MI, (2)KNN MI, (3) WKNN MI
6	MLP MI	(1)KNN MI, (2) WKNN MI
7	KNN MI	WKNN MI
8	WKNN MI	0
	With about 40% missing values	
1	GEMI	(1) MCMC, (2) HEW MI, (3) HD MI, (4) HES MI, (5) MLP MI, (6) KNN MI, (7) WKNN MI, (8) HOS MI, (9) RBF MI
2	HOW MI, GRNN MI	(1) MLP MI, (2) KNN MI, (3) WKNN MI, (4) HOS MI, (5) RBF MI
3	MCMC,HEW MI, HD MI	(1) KNN MI, (2) WKNN MI, (3) HOS MI, (4) RBF MI
4	HES MI, MLP MI	(1) WKNN MI, (2) HOS MI, (3) RBF MI
5	KNN MI	(1) HOS MI, (2) RBF MI
6	WKNN MI	(1) RBF MI
7	RBF MI	0
	With about 50% missing values	
1	GEMI	(1)HOW MI, (2) HES MI, (3) RBF MI, (4) MCMC, (5) HEW MI, (6) GRNN MI, (7) HD MI, (8) MLP MI, (9) KNN MI, (10) HOS MI, (11) WKNN MI
2	HOW MI	(1) MCMC, (2) HEW MI, (3) GRNN MI, (4) HD MI, (5) MLP MI, (6) KNN MI, (7) HOS MI, (8) WKNN MI
3	HES MI, RBF MI	(1) MLP MI, (2) KNN MI, (3) HOS MI, (4) WKNN MI
4	MCMC, HEW MI, GRNN MI, HD MI	(1)HOS MI, (2) WKNN MI
5	KNN MI, HOS MI	WKNN MI
6	WKNN MI	0
	With about 60% missing values	
1	GEMI	(1) HOW MI, (2) HEW MI, (3) HES MI, (4) HD MI, (5) KNN MI, (6) GRNN MI, (7) HOS MI, (8) MCMC, (9) RBF MI, (10) MLP MI, (11) WKNN MI
2	HOW MI	(1) HOS MI, (2) MCMC, (3) RBF MI, (4) MLP MI, (5) WKNN MI
3	HEW MI, HES MI, HD MI, KNN MI	(1) MCMC, (2) RBF MI, (3) MLP MI, (4) WKNN MI
4	GRNN MI, HOS MI	(1) MLP MI, (2) WKNN MI
5	MCMC, RBF MI, MLP MI	(1) WKNN MI
6	WKNN MI	0

	With about 70% missing values	
1	GEMI	(1) HOW MI, (2) HEW MI, (3) HES MI, (4) HD MI, (5) KNN MI, (6) GRNN MI, (7) HOS MI, (8) MCMC, (9) RBF MI, (10) MLP MI, (11) WKNN MI
2	HOW MI, HEW MI, HES MI, HD MI, KNN MI, GRNN MI, HOS MI, MCMC, RBF MI, MLP MI, WKNN MI	0

Appendix 5C: Pairwise comparisons among imputation algorithms in terms of the accuracy of estimating missing values

Rank	Algorithm	Significantly outperformed algorithms
With about 10% missing values		
1	GESI	(1) HOW SI, (2)HD SI, (3) GRNN SI, (4) EM, (5) WKNN SI, (6) HEW SI, (7) KNN SI, (8)ZI, (9) HES SI, (10)MLP SI, (11) HOS SI, (12) MS, (13) GEMI, (14) RBF SI, (15) MCMC, (16) HOW MI, (17) WKNN MI, (18)HEW MI, (19)MLP MI, (20) KNN MI, (21)GRNN MI, (22) RBF MI, (23) HOS MI, (24) HD MI, (25) HES MI
2	HOW SI, HD SI, GRNN SI, EM, WKNN SI	(1)MLP SI, (2) HOS SI, (3) MS, (4) GEMI, (5) RBF SI, (6) MCMC, (7) HOW MI, (8) WKNN MI, (9)HEW MI, (10)MLP MI, (11) KNN MI, (12)GRNN MI, (13) RBF MI, (14) HOS MI, (15) HD MI, (16) HES MI
3	HEW SI	(1) HOS SI, (2) MS, (3) GEMI, (4) RBF SI, (5) MCMC, (6) HOW MI, (7) WKNN MI, (8)HEW MI, (9)MLP MI, (10) KNN MI, (11)GRNN MI, (12) RBF MI, (13) HOS MI, (14) HD MI, (15) HES MI
4	KNN SI, ZI	(1) GEMI, (2) RBF SI, (3) MCMC, (4) HOW MI, (5) WKNN MI, (6)HEW MI, (7)MLP MI, (8) KNN MI, (9)GRNN MI, (10) RBF MI, (11) HOS MI, (12) HD MI, (13) HES MI
5	HES SI, MLP SI, HOS SI, MS	(1) RBF SI, (2) MCMC, (3) HOW MI, (4) WKNN MI, (5)HEW MI, (6)MLP MI, (7) KNN MI, (8)GRNN MI, (9) RBF MI, (10) HOS MI, (11) HD MI, (12) HES MI
6	GEMI, RBF SI	(1) MCMC, (2) HOW MI, (3) WKNN MI, (4)HEW MI, (5)MLP MI, (6) KNN MI, (7)GRNN MI, (8) RBF MI, (9) HOS MI, (10) HD MI, (11) HES MI
7	MCMC	(1) HOW MI, (2) WKNN MI, (3)HEW MI, (4)MLP MI, (5) KNN MI, (6)GRNN MI, (7) RBF MI, (8) HOS MI, (9) HD MI, (10) HES MI
8	HOW MI	(1) WKNN MI, (2)HET_MI2, (3)MLP MI, (4) KNN MI, (5)GRNN MI, (6) RBF MI, (7) HOM_MI1, (8) HD MI, (9) HET_MI1
9	WKNN MI	(1)HEW MI, (2)MLP MI, (3) KNN MI, (4)GRNN MI, (5) RBF MI, (6) HOS MI, (7) HD MI, (8) HES MI
10	HEW MI	(1)MLP MI, (2) KNN MI, (3)GRNN MI, (4) RBF MI, (5) HOM_MI1, (6) HD MI, (7) HET_MI1

11	MLP MI	(1) KNN MI, (2)GRNN MI, (3) RBF MI, (4) HOS MI, (5) HD MI, (6) HES MI
12	KNN MI	(1)GRNN MI, (2) RBF MI, (3) HOS MI, (4) HD MI, (5) HES MI
13	GRNN MI	(1) RBF MI, (2) HOS MI, (3) HD MI, (4) HES MI
14	RBF MI	(1) HOS MI, (2) HD MI, (3) HES MI
15	HOS MI	(1) HD MI, (2) HES MI
16	HD MI	HES MI
17	HES MI	0
	With about 20% missing values	
1	GESI	(1) EM, (2) MLP SI, (3) KNN SI, (4) HOS SI, (5) MS, (6) GEMI, (7) HD SI, (8) WKNN SI, (9) HOW MI, (10) MCMC, (11) HES SI, (12) ZI, (13) RBF MI, (14) HEW MI, (15) KNN MI, (16) GRNN MI, (17) WKNN MI, (18) HES MI, (19) MLP MI, (20) HOS MI, (21) HD MI
2	GRNN SI	(1) HD SI, (2) WKNN SI, (3) HOW MI, (4) MCMC, (5) HES SI, (6) ZI, (7) RBF MI, (8) HEW MI, (9) KNN MI, (10) GRNN MI, (11) WKNN MI, (12) HES MI, (13) MLP MI, (14) HOS MI, (15) HD MI
3	HOW SI, HEW SI, RBF SI, EM, MLP SI, KNN SI, HOS SI, MS	(1) WKNN SI, (2) HOW MI, (3) MCMC, (4) HES SI, (5) ZI, (6) RBF MI, (7) HEW MI, (8) KNN MI, (9) GRNN MI, (10) WKNN MI, (11) HET_MI1, (12) MLP MI, (13) HOS MI, (14) HD MI
4	GEMI, HD SI, WKNN SI	(1) HOM_MI2, (2) MCMC, (3) HES SI, (4) ZI, (5) RBF MI, (6) HEW MI, (7) KNN MI, (8) GRNN MI, (9) WKNN MI, (10) HES MI, (11) MLP MI, (12) HOS MI, (13) HD MI
5	HOW MI	(1) MCMC, (2) HES SI, (3) ZI, (4) RBF MI, (5) HEW MI, (6) KNN MI, (7) GRNN MI, (8) WKNN MI, (9) HES MI, (10) MLP MI, (11) HOS MI, (12) HD MI
6	MCMC	(1) HES SI, (2) ZI, (3) RBF MI, (4) HEW MI, (5) KNN MI, (6) GRNN MI, (7) WKNN MI, (8) HES MI, (9) MLP MI, (10) HOS MI, (11) HD MI

7	HES SI	(1) ZI, (2) RBF MI, (3) HEW MI, (4) KNN MI, (5) GRNN MI, (6) WKNN MI, (7) HES MI, (8) MLP MI, (9) HOS MI, (10) HD MI
8	ZI	(1) RBF MI, (2) HEW MI, (3) KNN MI, (4) GRNN MI, (5) WKNN MI, (6) HES MI, (7) MLP MI, (8) HOS MI, (9) HD MI
9	RBF MI	(1) HEW MI, (2) KNN MI, (3) GRNN MI, (4) WKNN MI, (5) HES MI, (6) MLP MI, (7) HOS MI, (8) HD MI
10	HEW MI	(1) KNN MI, (2) GRNN MI, (3) WKNN MI, (4) HES MI, (5) MLP MI, (6) HOS MI, (7) HD MI
11	KNN MI	(1) GRNN MI, (2) WKNN MI, (3) HES MI, (4) MLP MI, (5) HOS MI, (6) HD MI
12	GRNN MI	(1) WKNN MI, (2) HES MI, (3) MLP MI, (4) HOS MI1, (5) HD MI
13	WKNN MI	(1) HES MI, (2) MLP MI, (3) HOS MI, (4) HD MI
14	HES MI	(1) MLP MI, (2) HOS MI, (3) HD MI
15	MLP MI	(1) HOS MI, (2) HD MI
16	HOS MI	(1) HD MI
17	HD MI	0
	With about 30% missing values	
1	GESI	(1) HES SI, (2) GRNN SI, (3) EM, (4) MLP SI, (5) KNN SI, (6) GEMI, (7) RBF SI, (8) MS, (9) HES SI, (10) WKNN SI, (11) HOS SI, (12) RBF MI, (13) KNN MI, (14) MCMC, (15) WKNN MI, (16) HOW MI, (17) HOW SI, (18) HD SI, (19) HEW MI, (20) GRNN MI, (21) HD MI, (22) HES MI, (23) ZI, (24) MLP MI, (25) HOS MI
2	HEW SI, GRNN SI, EM, MLP SI, KNN SI,	(1) KNN MI, (2) MCMC, (3) WKNN MI, (4) HOW MI, (5) HOW SI, (6) HD SI, (7) HEW MI, (8) GRNN MI, (9) HD MI, (10) HES MI, (11) ZI, (12) MLP MI, (13) HOS MI
3	GEMI, RBF SI, MS	(1) MCMC, (2) WKNN MI, (3) HOW MI, (4) HOW SI, (5) HD SI, (6) HEW MI, (7) GRNN MI, (8) HD MI, (9) HES MI, (10) ZI, (11) MLP MI, (12) HOS MI
4	HES SI, WKNN SI, HOS SI	(1) WKNN MI, (2) HOW MI, (3) HOW SI, (4) HD SI, (5) HEW MI, (6) GRNN MI, (7) HD MI, (8) HEW MI, (9) ZI, (10) MLP MI, (11) HOS MI
5	RBF MI	(1) HOW MI, (2) HOW SI, (3) HD SI, (4) HEW MI, (5) GRNN MI, (6) HD MI, (7) HES MI, (8) ZI, (9) MLP MI, (10) HOS MI

6	KNN MI	(1) HOW SI, (2) HD SI, (3) HEW MI, (4) GRNN MI, (5) HD MI, (6) HES MI, (7) ZI, (8) MLP MI, (9) HOS MI
7	MCMC, WKNN MI	(1) HD SI, (2) HEW MI, (3) GRNN MI, (4) HD MI, (5) HES MI, (6) ZI, (7) MLP MI, (8) HOS MI
8	HOW MI, HOW SI, HD SI	(1) HEW MI, (2) GRNN MI, (3) HD MI, (4) HES MI, (5) ZI, (6) MLP MI, (7) HOS MI
9	HEW MI	(1) GRNN MI, (2) HD MI, (3) HES MI, (4) ZI, (5) MLP MI, (6) HOS MI
10	GRNN MI	(1) HD MI, (2) HES MI, (3) ZI, (4) MLP MI, (5) HOS MI
11	HD MI	(1) HES MI, (2) ZI, (3) MLP MI, (4) HOS MI
12	HES MI	(1) ZI, (2) MLP MI, (3) HOS MI
13	ZI	(1) MLP MI, (2) HOS MI
14	MLP MI	(1) HOS MI
15	HOS MI	0
	With about 40% missing values	
1	GESI	(1) GRNN SI, (2) MLP SI, (3) WKNN SI, (4)MCMC, (5) HD SI, (6) RBF MI, (7) KNN MI, (8) HES SI, (9) HOS SI, (10) GRNN MI, (11) HOW MI, (12) HEW MI, (13) KNN SI, (14) HES MI, (15) MS, (16) ZI, (17) HOS MI, (18) WKNN MI, (19) HD MI, (20) MLP MI
2	RBF SI	(1) HD SI, (2) RBF MI, (3) KNN MI, (4) HES SI, (5) HOS SI, (6) GRNN MI, (7) HOW MI, (8) HEW MI, (9) KNN SI, (10) HES MI, (11) MS, (12) ZI, (13) HOS MI, (14) WKNN MI, (15) HD MI, (16) MLP MI
3	EM	(1) KNN MI, (2) HES SI, (3) HOS SI, (4) GRNN MI, (5) HOW MI, (6) HEW MI, (7) KNN SI, (8) HES MI, (9) MS, (10) ZI, (11) HOS MI, (12) WKNN MI, (13) HD MI, (14) MLP MI
4	GEMI, HEW SI	(1) HES SI, (2) HOS SI, (3) GRNN MI, (4) HOW MI, (5) HEW MI, (6) KNN SI, (7) HES MI, (8) MS, (9) ZI, (10) HOS MI, (11) WKNN MI, (12) HD MI, (13) MLP MI
5	HOW SI, GRNN SI, MLP SI, WKNN SI	(1) HOS SI, (2) GRNN MI, (3) HOW MI, (4) HEW MI, (5) KNN SI, (6) HES MI, (7) MS, (8) ZI, (9) HOS MI, (10) WKNN MI, (11) HD MI, (12) MLP MI
6	MCMC, HD SI	(1) HEW MI, (2) KNN SI, (3) HES MI, (4) MS, (5) ZI, (6) HOS MI, (7) WKNN MI, (8) HD MI, (9) MLP MI

7	RBF MI, KNN MI	(1) KNN SI, (2) HES MI, (3) MS, (4) ZI, (5) HOS MI, (6) WKNN MI, (7) HD MI, (8) MLP MI
8	HES SI, HOS SI	(1) HES MI, (2) MS, (3) ZI, (4) HOS MI, (5) WKNN MI, (6) HD MI, (7) MLP MI
9	GRNN MI	(1) MS, (2) ZI, (3) HOS MI, (4) WKNN MI, (5) HD MI, (6) MLP MI
10	HOW MI, HEW MI, KNN SI	(1) ZI, (2) HOS MI, (3) WKNN MI, (4) HD MI, (5) MLP MI
11	HES MI, MS, ZI	(1) HOS MI, (2) WKNN MI, (3) HD MI, (4) MLP MI
12	HOS MI	(1) WKNN MI, (2) HD MI, (3) MLP MI
13	WKNN MI	(1) HD MI, (2) MLP MI
14	HD MI	(1) MLP MI
15	MLP MI	0
	With about 50% missing values	
1	GESI	(1) MLP SI, (2) GEMI, (3) GRNN SI, (4) HES SI, (5) MS, (6) MCMC, (7) HEW MI, (8) HES MI, (9) HOS MI, (10) HOW MI, (11) GRNN MI, (12) ZI, (13) HD SI, (14) MLP MI, (15) HOS SI, (16) HOW SI, (17)WKNN MI, (18) KNN
2	EM	(1) MCMC, (2) HEW MI, (3) HES MI, (4) HOS MI, (5) HOW MI, (6) GRNN MI, (7) ZI, (8) HD SI, (9) MLP MI, (10) HOS SI, (11) HOW SI, (12)WKNN MI, (13) KNN SI, (14) HD MI, (15) RBF MI, (16) KNN MI
3	RBF SI	(1) HOS MI, (2) HOW MI, (3) GRNN MI, (4) ZI, (5) HD SI, (6) MLP MI, (7) HOS SI, (8) HOW SI, (9)WKNN MI, (10) KNN SI, (11) HD MI, (12) RBF MI, (13) KNN MI
4	HEW SI	(1) HOW MI, (2) GRNN MI, (3) ZI, (4) HD SI, (5) MLP MI, (6) HOS SI, (7) HOW SI, (8)WKNN MI, (9) KNN SI, (10) HD MI, (11) RBF MI, (12) KNN MI
5	WKNN SI	(1) ZI, (2) HD SI, (3) MLP MI, (4) HOS SI, (5) HOW SI, (6)WKNN MI, (7) KNN SI, (8) HD MI, (9) RBF MI, (10) KNN MI
6	MLP SI	(1) HD SI, (2) MLP MI, (3) HOS SI, (4) HOW SI, (5)WKNN MI, (6) KNN SI, (7) HD MI, (8) RBF MI, (9) KNN MI
7	GEMI, GRNN SI	(1) MLP MI, (2) HOS SI, (3) HOW SI, (4)WKNN MI, (5) KNN SI, (6) HD MI, (7) RBF MI, (8) KNN MI
8	HES SI, MS	(1) HOS SI, (2) HOW SI, (3)WKNN MI, (4) KNN SI, (5) HD MI, (6) RBF MI, (7) KNN MI
9	MCMC, HEW MI, HES MI, HOS MI	(1) HOW SI, (2)WKNN MI, (3) KNN SI, (4) HD MI, (5) RBF MI, (6) KNN MI

10	HOW MI, GRNN MI, ZI	(1)WKNN MI, (2) KNN SI, (3) HD MI, (4) RBF MI, (5) KNN MI
11	HD SI, MLP MI, HOS SI	(1) KNN SI, (2) HD MI, (3) RBF MI, (4) KNN MI
12	HOW SI, WKNN MI, KNN SI	(1) HD MI, (2) RBF MI, (3) KNN MI
13	HD MI	(1) RBF MI, (2) KNN MI
14	RBF MI	KNN MI
15	KNN MI	0
	With about 60% missing values	
1	GESI	(1) HD SI, (2) MLP SI, (3) GEMI, (4) GRNN SI, (5) HES SI, (6) WKNN SI, (7) MS, (8) MCMC, (9) HEW MI, (10) HOS MI, (11) HOW MI, (12) RBF MI, (13) ZI, (14) MLP MI, (15) WKNN MI, (16) HOS SI, (17) HOW SI, (18) KNN SI, (19) HES MI, (20) KNN MI, (21) HD MI
2	EM	(1) WKNN SI, (2) MS, (3) MCMC, (4) HEW MI, (5) HOS MI, (6) HOW MI, (7) RBF MI, (8) ZI, (9) MLP MI, (10) WKNN MI, (11) HOS SI, (12) HOW SI, (13) KNN SI, (14) HES MI, (15) KNN MI, (16) HD MI
3	RBF SI	(1) HEW MI, (2) HOS MI, (3) HOW MI, (4) RBF MI, (5) ZI, (6) MLP MI, (7) WKNN MI, (8) HOS SI, (9) HOW SI, (10) KNN SI, (11) HES MI, (12) KNN MI, (13) HD MI
4	HEW SI	(1) HOS MI, (2) HOW MI, (3) RBF MI, (4) ZI, (5) MLP MI, (6) WKNN MI, (7) HOS SI, (8) HOW SI, (9) KNN SI, (10) HES MI, (11) KNN MI, (12) HD MI
5	GRNN MI, HD SI	(1) RBF MI, (2) ZI, (3) MLP MI, (4) WKNN MI, (5) HOS SI, (6) HOW SI, (7) KNN SI, (8) HES MI, (9) KNN MI, (10) HD MI
6	MLP SI	(1) ZI, (2) MLP MI, (3) WKNN MI, (4) HOS SI, (5) HOW SI, (6) KNN SI, (7) HES MI, (8) KNN MI, (9) HD MI
7	GEMI, GRNN SI	(1) MLP MI, (2) WKNN MI, (3) HOS SI, (4) HOW SI, (5) KNN SI, (6) HES MI, (7) KNN MI, (8) HD MI
8	HES SI, WKNN SI, MS	(1) WKNN MI, (2) HOS SI, (3) HOW SI, (4) KNN SI, (5) HES MI, (6) KNN MI, (7) HD MI
9	MCMC, HEW MI, HOS MI	(1) HOS SI, (2) HOW SI, (3) KNN SI, (4) HES MI, (5) KNN MI, (6) HD MI
10	HOW MI, RBF MI, ZI	(1) HOS SI, (2) KNN SI, (3) HES MI, (4) KNN MI, (5) HD MI
11	MLP MI, WKNN MI, HOS SI	(1) KNN SI, (2) HES MI, (3) KNN MI, (4) HD MI

12	HOW SI, KNN SI	(1) HES MI, (2) KNN MI, (3) HD MI
13	HES MI	(1) KNN MI, (2) HD MI
14	KNN MI	(1) HD MI
15	HD MI	0
	With about 70% missing values	
1	GESI, GEMI	(1) HES SI, (2) GRNN SI, (3) EM, (4) MLP SI, (5) KNN SI, (6) RBF SI, (7) MS, (8) HEW SI, (9) WKNN SI, (10) HOS SI, (11) RBF MI, (12) KNN MI, (13) MCMC, (14) WKNN MI, (15) HOW MI, (16) HOW SI, (17) HD SI, (18) HEW MI, (19) GRNN MI, (20) HD MI, (21) HES MI, (22) ZI, (23) MLP MI, (24) HOS MI
2	HES SI, GRNN SI, EM, MLP SI, KNN SI, RBF SI, MS, HEW SI, WKNN SI, HOS SI, RBF MI, KNN MI, MCMC, WKNN MI, HOW MI, HOS_SI, HD SI, HEW MI, GRNN MI, HD MI, HES MI, ZI, MLP MI, HOS MI	0

CHAPTER 6

Forecasting of Univariate Time Series

6.1. Introduction

Forecasting time series is a very important but very complex area (discussed in detail in section 1.1.3 of Chapter 1). Conventional time series forecasting algorithms are often plagued by parametric resonance effects, many adjustable parameters, the curse of dimensionality, and their insensitivity to local variations (discussed in detail in section 2.4 of Chapter 2). We propose a novel approach with automated feature selection for forecasting univariate time series that minimizes user involvement in the process of model formulation, and thus minimizes user mistakes. The proposed algorithm GEFTS (GRNN Ensemble for Forecasting Time Series) is an ensemble learning technique that combines the advice from several Generalized Regression Neural Networks. We compare GEFTS with several most used algorithms on 36 real datasets. The proposed algorithm appears to be more powerful than existing ones. Although it is an initial phase, GEFTS appears to be equally effective in the presence of seasonal patterns. Our experiments also provide insight into the following unanswered questions:

- Is pre-deseasonalization of the data critical in improving forecasting performance of neural networks?
- Does the hybrid of regression-based approaches (e.g. ARIMA and GARCH) and neural networks achieve significant performance improvement over both the regression based approach and the neural networks?
- This study reveals which conventional algorithms perform well on time series data.

The rest of the chapter is organized as follows: advantages of the proposed algorithms in section 6.2, details of the design and implementation of the new algorithm in section 6.3, comparative performance measurement in section 6.4, results and discussion in section 6.5, summary and conclusions in section 6.6, and future work in section 6.7.

6.2. Advantages of Proposed Algorithm (GEFTS) over Conventional Forecasting Techniques

GEFTS works in two stages. The feature space of a univariate time series includes all of the lagged variables. In stage 1, GEFTS identifies several core subsets of features using SAGA and each subset is used to train a separate GRNN. Each GRNN individually forecasts future values. In stage 2, we apply SAGA to select an optimal subset of GRNNs. The outputs of selected networks are then used to train a new GRNN in order to combine the predictions of individual networks. Figure 6.1 shows the basic architecture of a typical model of GEFTS.

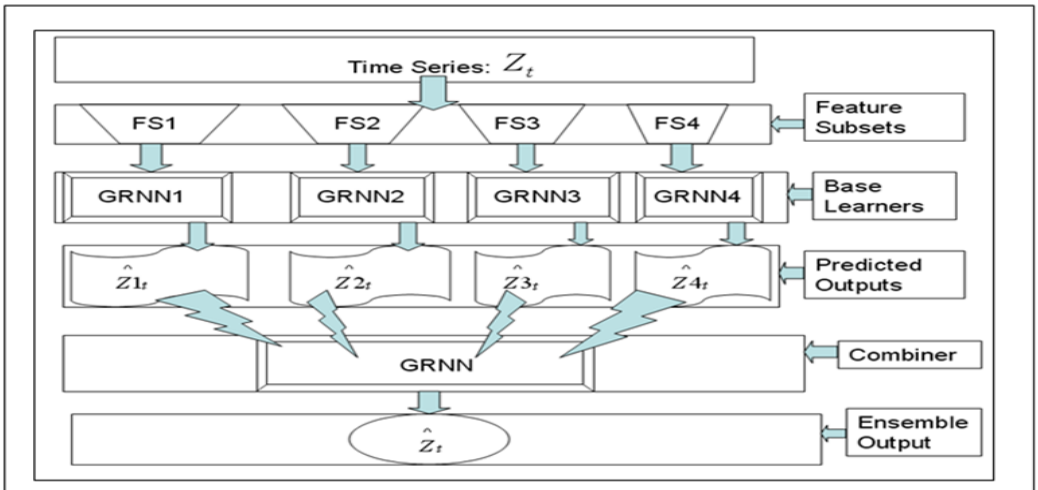

Figure 6.1: Architecture of a typical model in GEFTS

This unique architecture of GEFTS allows for improved performance over standard time series forecasting algorithms because of the following reasons:

First, A key advantage of using our approach for time series forecasting is that it has the potential to detect a greater number of important signals than would be detected with conventional algorithms. Temporal data mining projects a worst case scenario of data mining wherein we have high multi-collinearity, and a small sample size. In time series forecasting, lagged variables are included as inputs. Lagged variables are often highly serially correlated and there are frequently relatively few observations in observed time series since time series data follows one subject's changes over the course of time. Consequently, the forecasting error appears to proportionally increase with an increasing number of input values. This leads to a model with poor precision ability. GEFTS provides a mechanism to segment a GRNN network into multiple subnets (base learners). Each base GRNN learner independently forecasts the output using a distinct subset of lagged values. A combiner GRNN is then trained to predict the final output from the outputs of base learners. GEFTS is effective and robust against the curse of dimensionality, because each GRNN in the ensemble actually depends upon a smaller number of inputs.

Second, the problem with existing homogeneous neural network ensemble methods is that they reduce the curse of dimensionality problem, but cannot cope well with feature interactions since the outputs of base classifiers are combined via static weighted voting approach. By contrast, since GEFTS (our proposed ensemble strategy) applies a GRNN (a dynamic weighted voting scheme) to combine the predictions of multiple

base GRNN learners to make final forecasting, GEFTS can cope with feature interactions.

Third, Most of the conventional time series forecasting algorithms (e.g. ARIMA-GARCH, feedforward MLP, RNN, and ensemble with weighted average) are global approximators, which have a single predictive model holding over the entire data space. A global approach describes the average features for the entire regions. Hence, they cannot deal with heterogeneous short-run dynamics. However, the proposed algorithm (GEFTS) is a fuzzy clustering algorithm based on local approximation of memberships. Hence, GEFTS can deal with heterogeneous signals.

Fourth, GEFTS is a nonparametric algorithm that takes a signal, and divides it into multiple overlapping segments. Hence, GEFTS can handle a wide variety of data.

Fifth, Our proposed algorithm (GEFTS) is able to model seasonality directly so that prior deseasonalization is not necessary. Data pre-processing—both detrending and deseasonalization—is critical for conventional time series forecasting algorithms. As the conventional assumption of stationarity may not hold true for time series signals, deterending is an essential step for time series forecasting. The problem with long term trends is that the exact pattern never recurs when the data include a long term trend. Under such circumstances, no machine learning algorithm can learn. By detrending we remove long term trends from the data. In a (detrended) seasonal time series, some recognizable patterns recur after some regular intervals. In other words, the characteristics of joint distributions change cyclically across the seasonal time series.

Theoretically, seasonal patterns should be predictable. However, our empirical research suggests that most of the conventional time series forecasting algorithms cannot cope well with seasonal patterns for the two major reasons. First, seasonal patterns will likely aggravate the curse of dimensionality. The model of both detrended and deseasonalized data only includes non-seasonal lags, whereas the models of time series with seasonal patterns must include both seasonal and non-seasonal lags that give rise to an increase of the effective dimensionality (here, "effective dimensionality" refers to the smallest number of predictor variables that are necessary to model the time series adequately). In time series, the sample size shrinks dramatically when the model needs to include long seasonal lags. This leads to a very inefficient model and the forecasting of time series becomes less reliable. Second, conventional forecasting algorithms suffer from a weak local approximation of the non-linear process it is modelling. Hence the large seasonal movements can sometimes obscure smaller movements and consequently, produce poor results. Therefore, applying both detrending and deseasonalization simultaneously is the most effective data pre-processing approach in modelling and forecasting time series with conventional forecasting techniques. There are no hard and fast rules for the identification of the seasonal pattern in the data. Hence, it can be very difficult to identify periodicities from random patterns (noise), which often leads to underdifferencing or overdifferencing. Eggleton (1976) showed that subjects were unable to distinguish between alternating sequences and random sequences in contrived time series data [156]. The underdifferenced series exhibits seasonal variations which results in poor forecasting performance for the above discussed reasons. The effects of overdifferencing are even worse. Overdifferencing may lead to an efficiency loss, and a possible deterioration of forecasting, since

overdifferencing can introduce patterns into the original observations which were not actually in the data before the differencing. Our proposed algorithm (GEFTS) can deal with seasonal time series problems due to the particular advantage of being robust to the curse of dimensionality and being a local approximation algorithm.

Sixth, the goal of model optimization is to search for a compromise between the computation time and numerical accuracy of the modelling. Conventional optimization algorithms often fail to provide acceptable solutions due to the increased computational effort and convergence to local minima. By contrast, GEFTS- model optimization is carried out automatically with SAGA (presented in chapter 4). GEFTS employs SAGA for selecting optimal subsets of features both for base GRNN learners and the combiner GRNN learner. SAGA might not always find the best solution but is guaranteed to find a good solution in a reasonable time.

Seventh, in general, conventional forecasting algorithms have many free parameters to tune. In real world exercises, too many choices can create difficulties in using these techniques, especially for users lacking sufficient knowledge of forecasting. One of the key advantages of using the proposed algorithm (GEFTS) is that it requires relatively few parameters to be defined. In addition, we discovered the best default values for the free parameters of the algorithm. The parameters of proposed algorithms include: the parameter of GRNN and the parameters of SAGA. The default parameter setting was adopted on 200 synthetic datasets with various data characteristics. These default parameter values are almost always a good choice. Consequently, users need not worry about which and how to specify parameter values. They can just leave it at default.

However, as the optimal choice of parameters of SAGA mainly depends on the resource constraints, users can also interactively adjust parameter values in order to save computational resources. Since all the information is always digitally saved, SAGA allows the user to stop the algorithm periodically, check the status of the search, adjust the stopping criterion accordingly and then rerun the algorithm without disrupting the process flow.

Table 6.1 presents the default parameter values of SAGA. An overview of the methods that we used to generate synthetic datasets for optimizing default parameters are presented in section 3.2.3.3 of chapter 3.

Table 6.1: Default Parameter Setting of GEFTS

Parameter	Default Value
Parameters of SAGA for selecting optimal subsets of features	
Population Size of SA	≤ 100[a]
Population Size of GA	≤ 100[a]
Population Size of Hill climbing algorithm	≥ 100[a]
Stopping criteria of SA	The SA is stopped when the best solution does not change for 300 successive iterations.
Stopping criteria of GA	The GA is stopped when the best solution does not change for 100 successive iterations.
Parameters of SAGA for selecting an optimal feature subset ensemble (GRNN ensemble)	
Population Size of SA	≤ 100[a]
Population Size of GA	≤ 100[a]
Population Size of Hill climbing algorithm	≤ 100[a]
Stopping criteria of SA	The SA is stopped when the best solution does not change for 300 successive iterations.
Stopping criteria of GA	The GA is stopped when the best solution does not change for 100 successive iterations.
Parameter of GRNN	
Smoothing parameter (σ) of GRNN model	The default value for each centre's width (σ) to 2 times of the average distance to 20 nearest neighbours.

[a] Set the population size to <100 instead of 100 in cases where the list of alternative solutions is less than 100.

6.3. Design and Implementation Strategy of GEFTS

GEFTS forecasts the most likely future values of a time series as well as expected future volatility movements. For this, GEFTS constructs two ensemble models—model-*P* and model-*Q*. Model-*P* forecasts expected future values, whereas model-*Q*

forecasts expected future volatilities. Model-*P* is fitted to the time series data and model-*Q* is fitted to the square of residuals obtained from the model-*P* on the training dataset. Hence the model-*P* is first developed and then the model-*Q* is developed. Once the models are constructed, they forecast future events independently based on historical trends. Figure 6.2 shows two ensemble models.

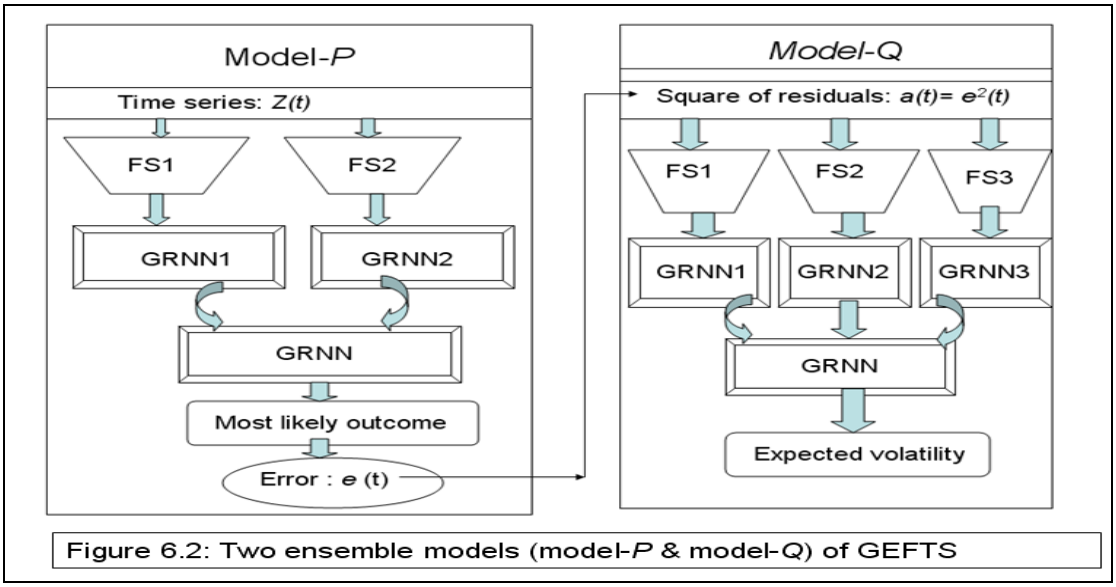

Figure 6.2: Two ensemble models (model-*P* & model-*Q*) of GEFTS

A GEFTS model (model-*P* and model-*Q*) contains only autoregressive parameters (i.e. lagged values of the series) and no moving average parameters (i.e. past values of the error made by the model). The schematic overview and the pseudo-code of the algorithm are presented in section 6.3.1. GEFTS employ our proposed feature subset selection algorithm SAGA to optimize the ensemble makeup. In this study, SAGA utilizes the BIC (which takes into account model parsimony, providing a distinct rank ordering of model fits) to identify the appropriate models. Section 6.3.2 demonstrates

how to assign a solution a fitness score based on its BIC value. BIC is discussed in section 3.4.

6.3.1. Pseudo code of GEFTS

Step 1: Detrend the time series:

i. *Eliminate non-stationarity in variance***:** Ascertain if there is any evidence of non-stationarity in variance by examining time series plots and by the *F*-test of variances of the first and second halves of the time series with a significance level (α) of 0.05. If the variance is not stable through the time, transform the original series into logarithms or square roots to stabilize its variance. If the logarithmic transformation does not stabilize the series variance, then try transforming the original series into its square roots.

ii. *Eliminate non-stationarity in mean:* Then examine the series for stability of its mean by observing time series plots. This can be confirmed by a *t*-test of the means of the first half of the period and the last half with a significance level (α) of 0.05. If the mean is not stable (i.e. the series indicates a trend), resolve this through first or second differencing of the series.

Step 2: Construction of the model (P) for forecasting future values in the time series :

- Normalize the values of stationary series in the range of 0 and 1.
- Select time lags with not less than 10 observations as candidate input variables.
- Apply SAGA to the chosen feature space to select the ≥ 100 best feature subsets (it selects less than 100 feature subsets only if 100 parsimonious diverse features

subsets are not available). Table 6.1 shows the parameter settings that we used in our experiments. The fitness value is assigned to each feature subset solution according to the BIC value. The BIC value for each feature subset on the training set was used to evaluate the feature subset solution. The lower the BIC value, the higher the fitness score.

$$BIC = n\left[\ln\left(\frac{RSS}{n}\right)\right] + (\ln n)k \qquad (6.1)$$

Where, RSS = residual sum-of-squares on the training data from the estimated model, n = Number of data points in training set, and k = number of features in a feature subset.

- Train a GRNN classifier with each selected feature subset.
- Again apply SAGA to select an optimal set of trained GRNNs (i.e. an optimal feature subset ensemble) for the first layer of the ensemble. Table 6.1 presents the parameter settings that we used for these simulations. Each GRNN of the GRNN ensemble is trained to forecast the future value of the time series independently. The outputs of the first layer are the inputs of the second layer. GRNN in the second layer predicts the final output. The fitness value is assigned to each candidate GRNN subset solution according to its BIC value. BIC values are calculated using data from the training dataset using equation 6.1. However, in this context, the k of Equation (6.1) is the number of base GRNN learners in the subset of GRNNs

SAGA selects the GRNN subset (feature subset ensemble) with the minimal BIC value. The selected GRNNs are used as base classifiers. Train the combiner GRNN.

Base classifier outputs are considered new features while training the combiner. Trained combiner learns the fusion rule from the outputs of base classifiers. We name this ensemble model as 'P'. Construction process of the model P is further illustrated in figure 6.3.

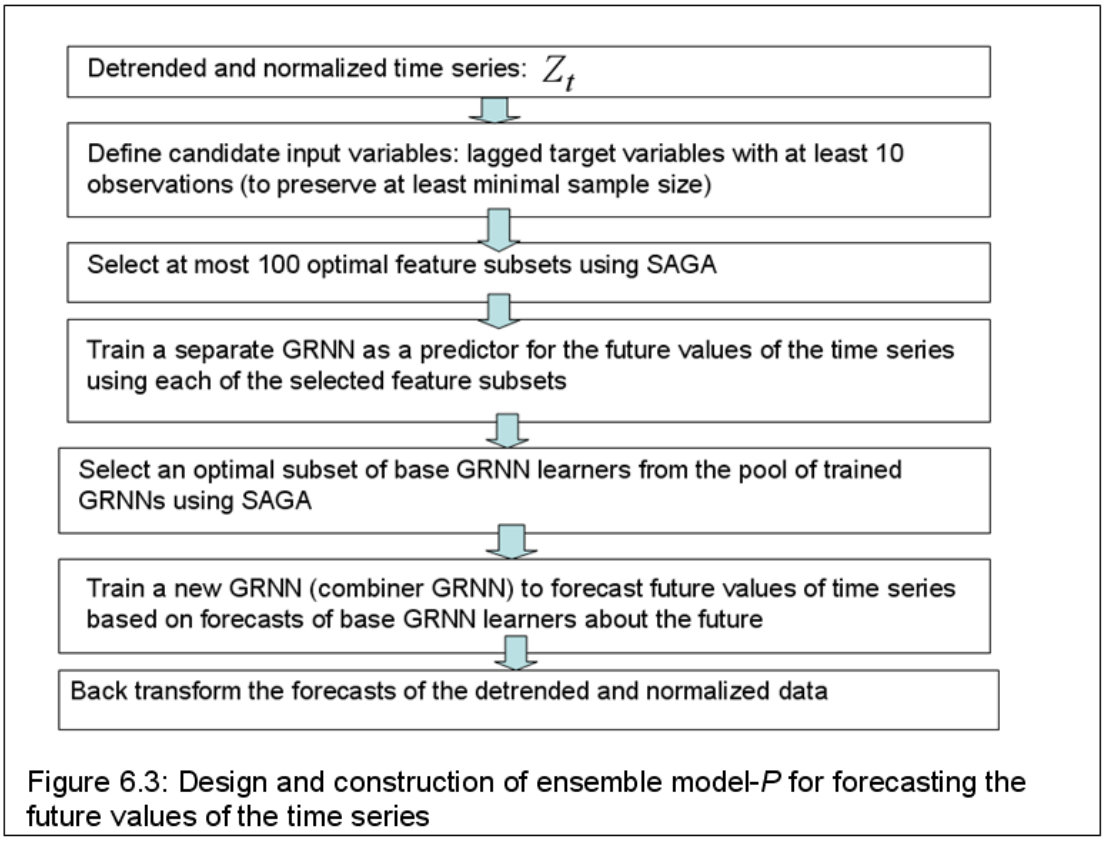

Figure 6.3: Design and construction of ensemble model-*P* for forecasting the future values of the time series

The model 'P' forecasts the future values of time series based upon the lagged (actual and predicted) values of time series. Multistep ahead forecasting was done as iterative one-step ahead forecasting (Figure 6.4). Forecasts are then transformed back into the original scale.

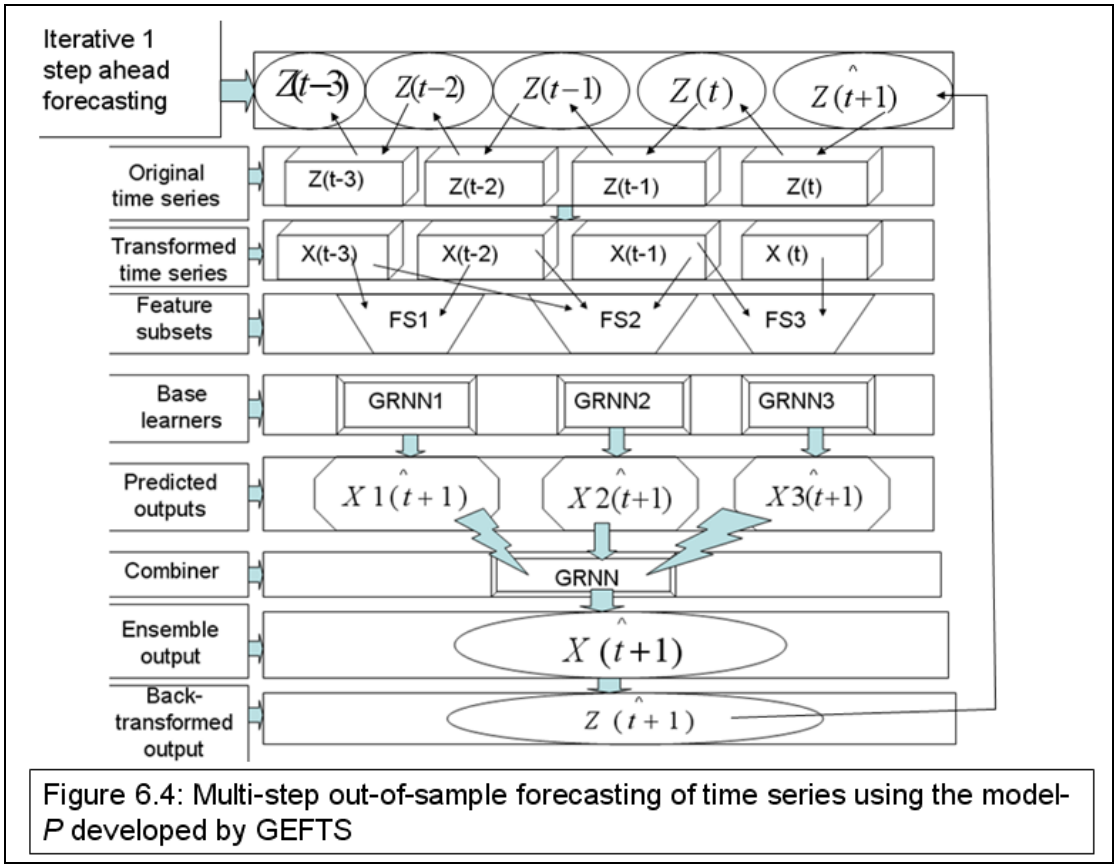

Figure 6.4: Multi-step out-of-sample forecasting of time series using the model-*P* developed by GEFTS

Step 3: Construction of GRNN ensemble model 'Q' for the forecasting volatility of time-series:

- Present training patterns to the model 'P' for prediction purposes. Then find the squared residual series a_t by subtracting the predicted value $\hat{Z}_t$ from the actual value Z_t and by taking the square of the residual: $a_t = (Z_t - \hat{Z}_t)^2$ (6.2).
- The squared residual series is normalized on a 0-1 scale.
- We fit a GRNN ensemble model to this squared residual series as we did for the original time series data in step 2. We name this model as 'Q'. Model 'Q' forecasts future volatility (the conditional mean of the squared residuals) based on the lagged squared residuals.

- Multi-step ahead forecasting of volatility is done by making iterative one-step ahead predictions up to desired step whilst feeding back the predicted output.
- After the forecast of volatility have been calculated on the transformed scale, GEFT translates the forecasted volatility back to the original scale of measurement.
- It is very important to note that during the out-of-sample forecasting, the process variance increases linearly with time due to the accumulation of random noise. Hence GEFTS calculates conditional variance at t-steps-ahead (where t=1, 2, 3….) out-of-sample forecasting, using equation (6.3).

Variance at time horizon t :

$\mathrm{Var}(\hat{Z}_t)$ = (Predicted squared residual at time step t) $\times t$ (6.3)

Step 4: We compute a 95% confidence interval (C.I.) for the forecast of the forecast of the future value of time series ($\hat{Z}_t$) as in equation (6.4):

$95\%\, C.I. = \hat{Z}_t \pm 1.96 * Var(\hat{Z}_t)$ (6.4)

The design and construction of model Q are summarized in Figure 6.5. Figure 6.6 displays the multistep-ahead (out-of-sample) forecasting of model-Q.

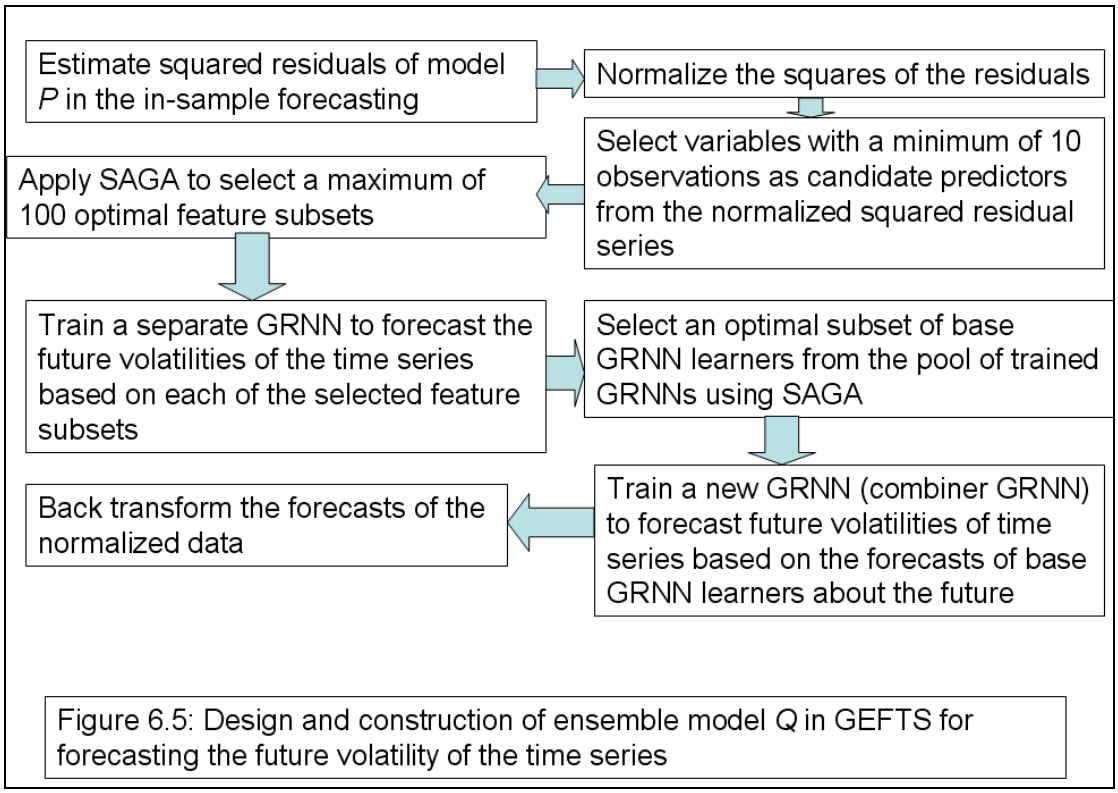

Figure 6.5: Design and construction of ensemble model Q in GEFTS for forecasting the future volatility of the time series

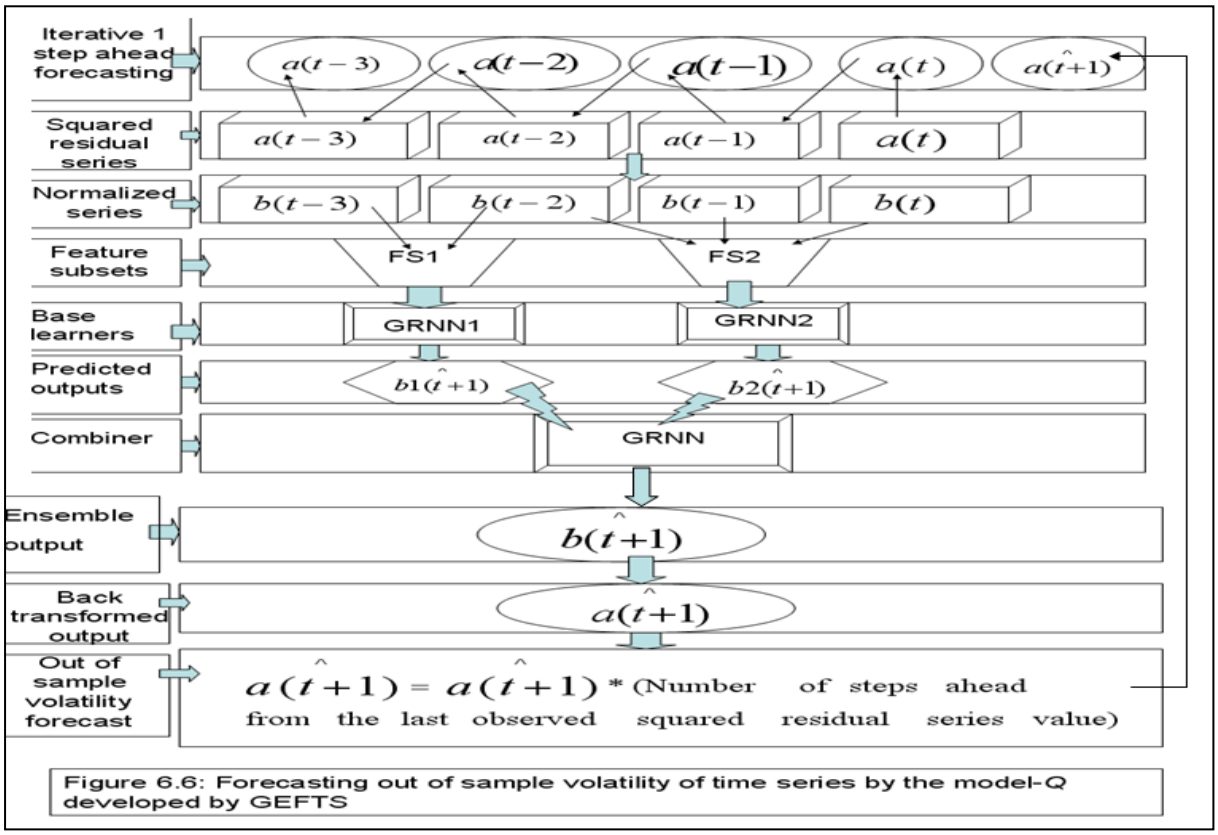

Figure 6.6: Forecasting out of sample volatility of time series by the model-Q developed by GEFTS

6.3.2 Strategy for Assigning Fitness Scores to Candidate Solutions based on Bayesian Information Criterion (BIC)

It is worth emphasizing that the fitness score should be between 0 and 1. So far, we have used the prediction accuracies to obtain fitness scores, where the prediction accuracy ranges from 0 to 100%. A fitness score was obtained for each solution by dividing the percent accuracy by 100. In contrast, the BIC value can range from $-\infty$ to $+\infty$. In this section, we demonstrate how we use BIC values to obtain fitness scores for solutions.

Step 1: BIC is calculated for each solution using equation (6.1). The lower the BIC value, the better the solution.

Step 2: Assign rank $1,\ldots,m$ to the solutions according to their BIC values, where rank '1' is the worst-fitness solution.

Step 3: For each solution, divide its rank value by the rank sum (i.e. the sum of the ranks of the solutions evaluated so far) to obtain the fitness score. The example table 6.3.2.1 illustrates a numerical example for assigning fitness scores to candidate solutions based on the BIC.

Example Table 6.3.2.1: Calculating a fitness score based on the BIC weight

Solution	BIC Weight	Rank	Fitness
Solution 2	-1463.8	4	0.4
Solution 1	-1163.8	3	0.3
Solution 3	-863.8	2	0.2
Solution 4	374.9	1	0.1
Rank sum		10	

The fitness scores of all the solutions were updated following the evaluation of each new solution.

6.4. Comparative Performance Analysis

We compare our proposed algorithm (GEFTS) with popular time series forecasting algorithms: (i) ARIMA-GARCH, (ii) GRNN (Generalized Regression Neural Networks), (iii) MLP (feedforward Multilayer Perceptrons), (iv) ERNN (Elman's Recurrent Neural Networks), (v) HA (a hybrid algorithm of regression-based methods and ERNN), (vi) RBFN (Radial Basis Function Neural Networks), (vi) a homogeneous ensemble of GRNNs with simple averaging approach (HOS), (vii) a homogeneous ensemble of GRNNs with weighted averaging approach (HOW), (viii) a heterogeneous neural network ensemble with simple averaging approach (HES), and (ix) a heterogeneous neural network ensemble with weighted averaging approach (HEW). The implementation methods of existing algorithms are described in section 3.1.3 of chapter 3. We compare the proposed algorithms (GRFTS) with well-known algorithms on 35 publicly available real-world datasets (described in section 3.2.3.2 of chapter 3) and one new problem (described in section 3.2.3.1 of chapter 3) of univariate time series analysis. In this study, we adopted the following strategies to assess the performance of proposed and existing algorithms:

a. ARIMA and GARCH models include autoregressive and moving average parameters. In this study, apart from ARIMA-GARCH, all other algorithms including the proposed algorithm GEFTS consider only autoregressive processes.
b. We determine the parameters of ARIMA and GARCH models by using ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function). Input variables for all other algorithms were selected by using the SAGA. SAGA used BIC model selection criterion to select an optimal model.

c. In ARIMA-GARCH, the fitted ARIMA model forecasts future values of a time series and the GARCH model forecasts future volatilities. Other forecasting algorithms construct two models—one for the future value and another for the future volatility. The first model is fitted to the stationary time series, whereas the second model is fitted to the squared residuals of the first model.

d. The original time series were detrended by standard methods (the most suitable transformation of the data and differencing). We compare the forecasting performance of single and ensemble neural networks before and after deseasonalizing data. On the other hand, the time series are always deseasonalized before being used by the ARIMA methodology.

e. The first two-thirds of a time series were used as the training set, and the remaining third as a test set. We compare the out-of-sample forecasting performance of the algorithms in terms of their interval and point estimation accuracy.

f. The one-step-ahead process is iterated to obtain multi-step-ahead forecasts.

g. We compute the accuracy of algorithms on 1, $\frac{N}{2}$ and N step-ahead forecasts (N denotes the number of out-of-sample forecasts).

h. An interval forecast is considered to be correct if the actual value falls inside the predicted 95% confidence interval. Point estimation accuracy was evaluated by the following equation (6.5).

$$\text{Point estimation accuracy} = 100 - \frac{100}{\text{N}} \sum_{i=1}^{N} \frac{\left| \left(Y_i - \hat{Y}_i \right) \right|}{Y_i} \qquad (6.5)$$

Where, N =number of observation in the test set, Y_i =actual output, $\hat{Y}_i$ =forecasted output.

- In time series forecasting, the magnitude of the forecasting error increases over time, since the uncertainty increases with the horizon of the forecast. When forecasting time series, interval estimates are more informative than simple point estimates. Hence, for each dataset, the algorithms were ranked in terms of their accuracy in the interval estimation. If two algorithms have the same interval-estimation accuracy on a dataset, the algorithms were ranked based on the point estimation accuracy.
- The Friedman test is used to test the null hypothesis that the performance is the same for all algorithms. After applying the Friedman test and noting it is significant "Comparison of Groups or Conditions with a Control" tests (details are available in section 3.8 of this thesis and in [140], p. 181) were performed in order to test the (null) hypothesis that there is no significant difference between any pair of algorithms.

6.5. Results and Discussion

We evaluate the performance of our proposed algorithm (GEFTS) with several well-known forecasting algorithms using 36 time-series datasets, both before and after seasonal adjustments are made. The one, $N/2$ and N-step-ahead predictions are studied (N is the number of out-of-sample forecasts). Comparative overall results (test accuracies and standard deviations in parenthesis, rounded %) are reported in table 6.2. Friedman Test reveals significant differences ($p < 0.05$) in the performance of time series forecasting algorithms at all three time horizons (1, $N/2$ and N -step-ahead). The results of pairwise comparison tests **(** $p < 0.05$ **)** among time series forecasting

algorithms are reported in Table 6.3. Figures 6.7 and 6.8 illustrate the performances of algorithm, in terms of interval estimation accuracy and point estimation accuracy respectively, on Pulse Pressure dataset.

Table 6.2 Summary performance report: A comparison of time series forecasting algorithms (standard deviations in parenthesis)

	1-step-ahead forecast		*N*/2- step-ahead-forecast		*N*-step-ahead forecast	
Algorithm	Overall mean interval estimation Accuracy (rounded %)	Overall mean point estimation accuracy (rounded %)	Overall mean interval estimation Accuracy (rounded %)	Overall mean point estimation accuracy (rounded %)	Overall mean interval estimation Accuracy (rounded %)	Overall mean point estimation accuracy (rounded %)
GEFTS (d[a])	97 (2)	96 (4)	86 (8)	72 (12)	69 (14)	62 (15)
GEFTS (nd[b])	97 (2)	96 (3)	86 (10)	72 (11)	68 (13)	68 (15)
ARIMA-GARCH	95 (6)	87 (9)	80 (11)	61 (22)	61 (17)	55 (22)
HOS (d[a])	92 (8)	86 (11)	69 (16)	65 (17)	60 (11)	49 (23)
HOS (nd[b])	90 (8)	85 (9)	68 (19)	58 (17)	60 (13)	45 (21)
HOW (d[a])	95 (4)	91 (8)	83 (9)	65 (14)	58 (16)	57 (21)
HOW (nd[b])	93 (4)	89 (11)	73 (12)	60 (13)	61 (14)	52 (21)
HES (d[a])	88 (10)	77 (15)	57 (15)	56 (20)	62 (13)	49 (20)
HES (nd[b])	81 (12)	75 (11)	61 (21)	53 (20)	55 (13)	48 (26)
HEW (d[a])	92 (7)	88 (11)	70 (11)	64 (14)	64 (10)	53 (19)
HEW (nd[b])	80 (14)	78 (11)	68 (15)	55 (12)	57 (11)	53 (18)
GRNN (d[a])	87 (10)	71 (15)	69 (12)	62 (13)	52 (20)	53 (24)
GRNN (nd[b])	78 (11)	70 (11)	64 (16)	57 (20)	54 (17)	39 (20)
RBFN (d[a])	86 (10)	68 (16)	66 (15)	58 (17)	50 (16)	46 (23)
RBFN (nd[b])	75 (12)	67 (14)	54 (19)	52 (19)	49 (13)	35 (20)
MLP (d[a])	82 (10)	71 (9)	67 (10)	63 (12)	55 (23)	55 (28)
MLP (nd[b])	77 (11)	57 (20)	60 (20)	55 (23)	48 (15)	40 (25)
HA (d[a])	83 (14)	70 (12)	60 (13)	61 (19)	48 (17)	44 (26)
HA (nd[b])	72 (8)	67 (15)	58 (15)	49 (21)	49 (28)	38 (28)
ERNN (d[a])	93 (5)	90 (8)	80 (9)	66 (19)	51 (15)	56 (29)
ERNN (nd[b])	74 (12)	65 (15)	60 (23)	49 (24)	40 (22)	41 (26)

- [a]d represents deseasonalized data
- [b]nd repreasents non-deseasonalized data

Table 6.3: Pairwise Comparisons between Time Series Forecasting Algorithms (d' in parenthesis stands for 'deseasonalized data' and 'nd' in parenthesis stands for 'non-deseasonalized data'.**)**

	1-step-ahead-forecast	
Rank	Algorithm	significantly outperformed algorithms
1	GEFTS(d), GEFTS (nd)	(1)ARIMA, (2) HOW (nd), (3) ERNN (d), (4) HOS (d), (5) HOW (d), (6) HEW (d), (7) HOS (nd), (8) HES (d), (9) GRNN (d), (10) RBFN (d), (11) HA (d), (12) MLP (d), (13) HEW (nd), (14) HES (nd), (15) GRNN (nd), (16) MLP (nd), (17) RBFN (nd), (18) ERNN (nd), (19) HA (nd)
2	ARIMA	(1) HOS (d), (2) HOW (d), (3) HEW (d), (4) HOS (nd), (5) HES (d), (6) GRNN (d), (7) RBFN (d), (8) HA (d), (9) MLP (d), (10) HEW (nd), (11) HES (nd), (12) GRNN (nd), (13) MLP (nd), (14) RBFN (nd), (15) ERNN (nd), (16) HA (nd)
3	HOW (nd), ERNN (d)	(1) HOW (d), (2) HEW (d), (3) HOS (nd), (4) HES (d), (5) GRNN (d), (6) RBFN (d), (7) HA (d), (8) MLP (d), (9) HEW (nd), (10) HES (nd), (11) GRNN (nd), (12) MLP (nd), (13) RBFN (nd), (14) ERNN (nd), (15) HA (nd)
4	HOS(d), HOW(d)	(1) HEW (d), (2) HOS (nd), (3) HES (d), (4) GRNN (d), (5) RBFN (d), (6) HA (d), (7) MLP (d), (8) HEW (nd), (9) HES (nd), (10) GRNN (nd), (11) MLP (nd), (12) RBFN (nd), (13) ERNN (nd), (14) HA (nd)
5	HEW (d)	(1) HOS (nd), (2) HES (d), (3) GRNN (d), (4) RBFN (d), (5) HA (d), (6) MLP (d), (7) HEW (nd), (8) HES (nd), (9) GRNN (nd), (10) MLP (nd), (11) RBFN (nd), (12) ERNN (nd), (13) HA (nd)
6	HOS(nd)	(1) HES (d), (2) GRNN (d), (3) RBFN (d), (4) HA (d), (5) MLP (d), (6) HEW (nd), (7) HES (nd), (8) GRNN (nd), (9) MLP (nd), (10) RBFN (nd), (11) ERNN (nd), (12) HA (nd)
7	HES (d)	(1) GRNN (d), (2) RBFN (d), (3) HA (d), (4) MLP (d), (5) HEW (nd), (6) HES (nd), (7) GRNN (nd), (8) MLP (nd), (9) RBFN (nd), (10) ERNN (nd), (11) HA (nd)
8	GRNN (d)	(1) RBFN (d), (2) HA (d), (3) MLP (d), (4) HEW (nd), (5) HES (nd), (6) GRNN (nd), (7) MLP (nd), (8) RBFN (nd), (9) ERNN (nd), (10) HA (nd)
9	RBFN (d)	(1) HA (d), (2) MLP (d), (3) HEW (nd), (4) HES (nd), (5) GRNN (nd), (6) MLP (nd), (7) RBFN (nd), (8) ERNN (nd), (9) HA (nd)
10	HA (d)	(1) MLP (d), (2) HEW (nd), (3) HES (nd), (4) GRNN (nd), (5) MLP (nd), (6) RBFN (nd), (7) ERNN (nd), (8) HA (nd)
11	MLP (d)	(1) HEW (nd), (2) HES (nd), (3) GRNN (nd), (4) MLP (nd), (5) RBFN (nd), (6) ERNN (nd), (7) HA (nd)

12	HEW(nd)	(1) HES (nd), (2) GRNN (nd), (3) MLP (nd), (4) RBFN (nd), (5) ERNN (nd), (6) HA (nd)
13	HES(nd)	(1) GRNN (nd), (2) MLP (nd), (3) RBFN (nd), (4) ERNN (nd), (5) HA (nd)
14	GRNN (nd)	(1) MLP (nd), (2) RBFN (nd), (3) ERNN (nd), (4) HA (nd)
15	MLP (nd)	(1) RBFN (nd), (2) ERNN (nd), (3) HA (nd)
16	RBFN (nd)	(1) ERNN (nd), (2) HA (nd)
17	ERNN (nd)	(1) HA (nd)
18	HA (nd)	0
	***N*/2-step-ahead-forecast**	
Rank	Algorithms	Significantly outperformed algorithms
1	GEFTS(d), GEFTS (nd)	(1)HOW(d), (2) ERNN (d), (3) HOW (nd), (4) HOS (nd), (5) HEW (d), (6) GRNN (d), (7) HOS (d), (8) HEW (nd), (9) MLP (d), (10) RBFN (d), (11) GRNN (nd), (12) HES (nd), (13) MLP (nd), (14) ERNN (nd), (15) HA (d), (16) HA (nd), (17) HES (d), (18) RBFN (nd)
2	ARIMA	(1) ERNN (d), (2) HOW (nd), (3) HOS (nd), (4) HEW (d), (5) GRNN (d), (6) HOS (d), (7) HEW (nd), (8) MLP (d), (9) RBFN (d), (10) GRNN (nd), (11) HES (nd), (12) MLP (nd), (13) ERNN (nd), (14) HA (d), (15) HA (nd), (16) HES (d), (17) RBFN (nd)
3	HOW (d), ERNN (d)	(1) HOW (nd), (2) HOS (nd), (3) HEW (d), (4) GRNN (d), (5) HOS (d), (6) HEW (nd), (7) MLP (d), (8) RBFN (d), (9) GRNN (nd), (10) HES (nd), (11) MLP (nd), (12) ERNN (nd), (13) HA (d), (14) HA (nd), (15) HES (d), (16) RBFN (nd)
4	HOW (nd)	(1) HOS (nd), (2) HEW (d), (3) GRNN (d), (4) HOS (d), (5) HEW (nd), (6) MLP (d), (7) RBFN (d), (8) GRNN (nd), (9) HES (nd), (10) MLP (nd), (11) ERNN (nd), (12) HA (d), (13) HA (nd), (14) HES (d), (15) RBFN (nd)
5	HOS (nd), HEW (d), GRNN (d)	(1) HOS (d), (2) HEW (nd), (3) MLP (d), (4) RBFN (d), (5) GRNN (nd), (6) HES (nd), (7) MLP (nd), (8) ERNN (nd), (9) HA (d), (10) HA (nd), (11) HES (d), (12) RBFN (nd)
6	HOS (d), HEW (nd)	(1) MLP (d), (2) RBFN (d), (3) GRNN (nd), (4) HES (nd), (5) MLP (nd), (6) ERNN (nd), (7) HA (d), (8) HA (nd), (9) HES (d), (10) RBFN (nd)
7	MLP (d)	(1) RBFN (d), (2) GRNN (nd), (3) HES (nd), (4) MLP (nd), (5) ERNN (nd), (6) HA (d), (7) HA (nd), (8) HES (d), (9) RBFN (nd)
8	RBFN (d)	(1) GRNN (nd), (2) HES (nd), (3) MLP (nd), (4) ERNN (nd), (5) HA (d), (6) HA (nd), (7) HES (d), (8) RBFN (nd)

9	GRNN (nd)	(1) HES (nd), (2) MLP (nd), (3) ERNN (nd), (4) HA (d), (5) HA (nd), (6) HES (d), (7) RBFN (nd)
10	HES (nd)	(1) MLP (nd), (2) ERNN (nd), (3) HA (d), (4) HA (nd), (5) HES (d), (6) RBFN (nd)
11	MLP(nd)	(1) ERNN (nd), (2) HA (d), (3) HA (nd), (4) HES (d), (5) RBFN (nd)
12	ERNN (nd)	(1) HA (d), (2) HA (nd), (3) HES (d), (4) RBFN (nd)
13	HA (d)	(1) HA (nd), (2) HES (d), (3) RBFN (nd)
14	HA (nd)	(1) HES (d), (2) RBFN (nd)
15	HES (d)	(1) RBFN (nd)
16	RBFN (nd)	0
	***N*-step-ahead forecast**	
Rank	Algorithm	Significantly outperformed algorithms
1	GEFTS (d), GEFTS (nd)	(1) HEW (d), (2) HOS (d), (3) ARIMA, (4)HOS (nd), (5) HES (d), (6) HOW (d), (7) HEW (nd), (8) HES (nd), (9) MLP (d), (10) GRNN (nd), (11) GRNN (d), (12) ERNN (d), (13) HA (nd), (14) RBFN (d), (15) HA (d), (16) MLP (nd), (17) RBFN (nd), (18) ERNN (nd)
2	HOW (nd), HEW (d)	(1) HOW (d), (2) HEW (nd), (3) HES (nd), (4) MLP (d), (5) GRNN (nd), (6) GRNN (d), (7) ERNN (d), (8) HA (nd), (9) RBFN (d), (10) HA (d), (11) MLP (nd), (12) RBFN (nd), (13) ERNN (nd)
3	HOS (d)	(1) HEW (nd), (2) HES (nd), (3) MLP (d), (4) GRNN (nd), (5) GRNN (d), (6) ERNN (d), (7) HA (nd), (8) RBFN (d), (9) HA (d), (10) MLP (nd), (11) RBFN (nd), (12) ERNN (nd)
4	ARIMA, HOS (nd), HES (d)	(1) HES (nd), (2) MLP (d), (3) GRNN (nd), (4) GRNN (d), (5) ERNN (d), (6) HA (nd), (7) RBFN (d), (8) HA (d), (9) MLP (nd), (10) RBFN (nd), (11) ERNN (nd)
5	HOW (d), HEW (nd)	(1) MLP (d), (2) GRNN (nd), (3) GRNN (d), (4) ERNN (d), (5) HA (nd), (6) RBFN (d), (7) HA (d), (8) MLP (nd), (9) RBFN (nd), (10) ERNN (nd)
6	HES (nd), MLP (d)	(1) GRNN (nd), (2) GRNN (d), (3) ERNN (d), (4) HA (nd), (5) RBFN (d), (6) HA (d), (7) MLP (nd), (8) RBFN (nd), (9) ERNN (nd)
7	GRNN (nd)	(1) GRNN (d), (2) ERNN (d), (3) HA (nd), (4) RBFN (d), (5) HA (d), (6) MLP (nd), (7) RBFN (nd), (8) ERNN (nd)
8	GRNN (d)	(1) ERNN (d), (2) HA (nd), (3) RBFN (d), (4) HA (d), (5) MLP (nd), (6) RBFN (nd), (7) ERNN (nd)
9	ERNN (d)	(1) HA (nd), (2) RBFN (d), (3) HA (d), (4) MLP (nd), (5) RBFN (nd), (6) ERNN (nd)

10	HA (nd)	(1) RBFN (d), (2) HA (d), (3) MLP (nd), (4) RBFN (nd), (5) ERNN (nd)
11	RBFN (d)	(1) HA (d), (2) MLP (nd), (3) RBFN (nd), (4) ERNN (nd)
12	HA (d)	(1) MLP (nd), (2) RBFN (nd), (3) ERNN (nd)
13	MLP (nd)	(1) RBFN (nd), (2) ERNN (nd)
14	RBFN (nd)	(1) ERNN (nd)
15	ERNN (nd)	0

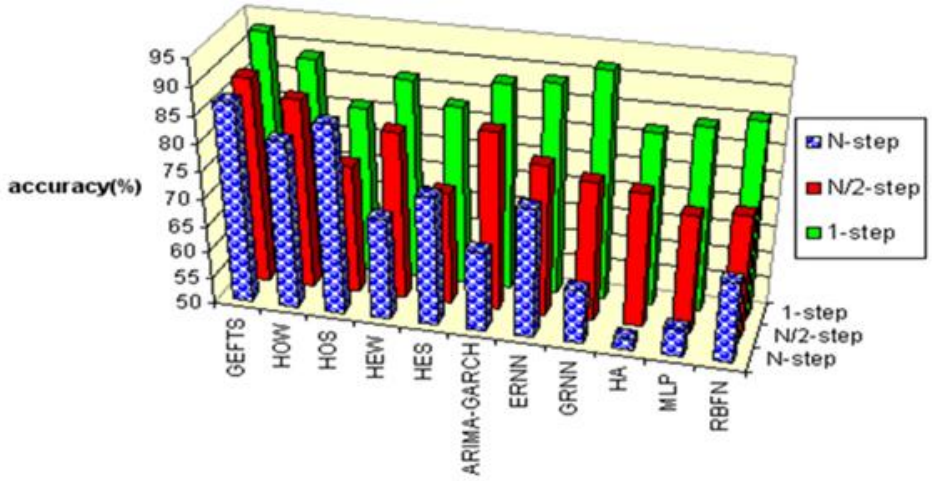

Figure 6.7: Interval forecasting accuracy on 'Pulse Pressure' dataset

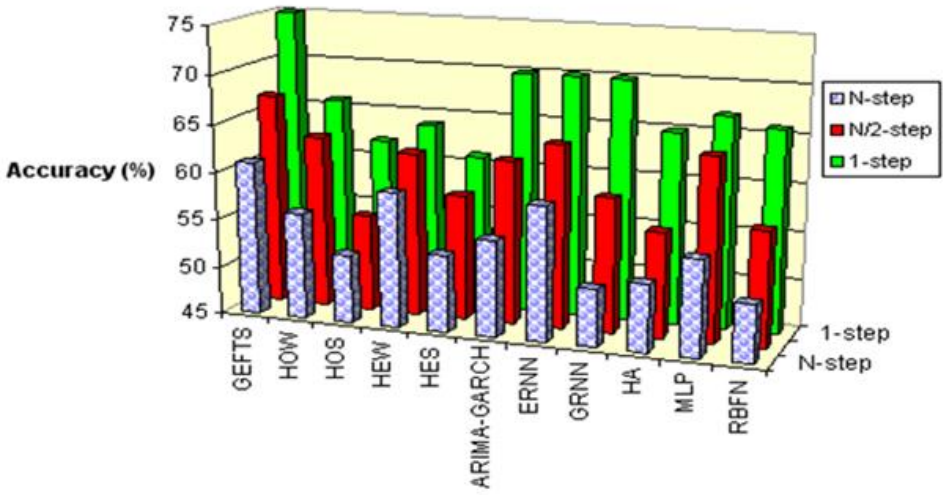

Figure 6.8: Point forecasting accuracy on 'Pulse Pressure' dataset

Key Findings:

- The proposed algorithm (GEFTS) significantly outperformed conventional algorithms both at short horizons (one-step ahead), and at longer horizons ($N/2$ and N -step-ahead) (Figures 6.7-6.8, and Tables 6.2-6.3).

- For GEFTS, apparently the deseasonalization offers no significant performance improvement (no statistically significant differences were found before and after the deseasonalization) (Tables 6.2-6.3). All the single neural networks (GRNN, ERNN, MLP, and RBFN) and HA achieve improved performance when applied to deasonalized data. We found mixed results concerning the effects of deseasonalization on the performance of conventional homogeneous ensemble neural networks (HOS, and HOW). In 1- and N -step-ahead forecasting, the performance of HOW before deseasonalization (HOW(nd)) was significantly better than that of HOW after deseasonalization (HOW(d)), whereas in $N/2$ -step-ahead forecasting, the performance after deseasonalization (HOW(d)) was significantly better than the performance prior to

deasonalization (HOW(nd)). On the other hand, in $N/2$-step-ahead forecasting, the performance of HOS before deseasonalization was significantly better than that of HOS after deseaonalization (HOS(d)), whereas, in 1- and N-step-ahead forecasting the performance after deseasonalization (HOS(d)) was significantly better than the performance before deseasonalization (HOS(nd)). Hence, for conventional homogeneous neural network ensemble models (HOS and HOW), our simulation results are inconclusive and further study is necessary.

However, prior deseasonalization step led to improved accuracy for conventional heterogeneous ensemble networks (HES and HEW).

- In general, the performance of neural network ensembles is better than that of single neural networks (Figures 6.1-6.2, and Tables 6.2-6.3). The homogeneous neural network ensemble with weighted voting (HOW) is the next best time series forecasting algorithm to our proposed algorithm GEFTS. It is worthy to note that deseasonalization was not performed on 'Pulse Pressure' series (figures 6.7-6.8), since we could not recognize seasonal patterns in it.

- The hybrid algorithm (HA) did not show significant improvements over the constituent algorithms (Tables 6.2 and 6.3). Therefore, our simulation results reinforce the arguments that ARIMA neural network hybrids are not better than single models.

6.6 Summary and Conclusions

We proposed an automatic algorithm (GEFTS) for univariate time series forecasting. GEFTS is an ensemble algorithm wherein both the base learners and the combiner are GRNN learners. GEFTS can simulate all the seasonal effects and prior

deseasonalization is not necessary. GEFTS employs our proposed feature subset selection algorithm SAGA for automatic optimization of control parameters. We compare our proposed algorithm with the best known algorithms on 36 real datasets. The one-step process is iterated to obtain multi-step-ahead forecasts. The results obtained from experiments show that GEFTS is superior to conventional algorithms, both for short-term and long-term out-of-sample forecasting.

6.7 Future Work:

Our proposed algorithm GEFTS allows only autoregressive terms without moving average terms. In future we will study the effect of moving average terms on the performance of GEFTS.

The scope of this study was limited to univariate time series forecasting which is focused on attempting to forecast the future values of a random variable based on its past values. Although the previous values of the variable of interest are good predictors of its future values, it is not the only predictor. The evolution of time series *X* may be dependent on the evolution of time series *Y*. Multivariate time series forecasting takes into consideration the evolution of time series variable of interest *X* along with the evolutions of other time series variables (e.g. *Y*) affecting *X*, to come up with the forecasted values of *X*. To promote the applicability of the proposed algorithm, we would like to investigate if there is an elegant way to extend our algorithm so that it can handle further complexities of multivariate time series forecasting.

Chapter 7

Conclusion

7.1 Research Problems and Proposed Solutions

This thesis deals with the non-trivial task of dimensionality reduction. Many real world applications are confronted with high dimensional data. The curses of high dimensionality include difficulty in fully exploring a vast parameter space, the difficulty in interpreting patterns, the possibility of overfitting in small samples and subsequent validity shrinkage. Thus dimensionality reduction, and hence feature subset selection, is critical for data mining applications. Feature subset selection involves choosing the smallest subset of features that maximizes the prediction accuracy. An exhaustive search of all possible subsets will guarantee finding the best solution, but exhaustive search for the best feature subset is prohibitively expensive. A search strategy is needed to explore the feature space. A good search algorithm should facilitate the exploration of new areas in the search space, the rapid identification of the most promising zones, and the highly localized search in promising areas. No existing search strategy can attain all three of these purposes. In this thesis, we present an improved algorithm (SAGA) for feature subset selection. SAGA is a hybrid scheme that combines the strengths of three search algorithms (a simulated annealing (SA), a genetic algorithm (GA) and a greedy search algorithm) and one learning algorithm (Generalized Regression Neural Networks (GRNN)). Among existing search algorithms, SA is a good tool if the goal is to travel a long distance quickly, since it travels by making long jumps. SA promotes the exploration of new regions in the solution space having several local minima. It has the ability to jump over high peaks

and it can go easily up or down. It may fall down and reach a position close to the bottom, but it is also very likely that it may jump up ending in a position even higher than the initial position. First, our proposed algorithm SAGA employs a SA to quickly explore the vast feature space. The SA generates a pool of solutions, some will be better than the others and the cross of these solutions could yield a better one still. Next, our proposed algorithm SAGA applies a GA exclusively to several best-to-date feature subsets of the SA for the exploitation of already sampled regions in the search space. Genetic algorithms (GA) evolve new and better solutions through the application of genetic operators (crossover, mutation, and selection and survival of the fittest). In the final step, SAGA further fine-tunes the search by using a greedy search algorithm. SAGA employs a fast learning algorithm (Generalized Regression Neural Networks) as a means to ensure computational efficiency. GRNN is a one-pass algorithm. The highest computational efficiency is necessary for exploring a vast unknown space of possible solutions.

SAGA reduces the dimensionality of the original feature space by removing irrelevant and redundant features. However, our research suggests that the problem may continue to persist even after we select the most optimally parsimonious subset of features. This happens when the sample size is not sufficient to accommodate all our independent variables. Determining the necessary sample size is not a simple task. The required total sample size depends on many factors. To tackle this problem, we introduce a homogeneous ensemble network that combines several GRNN classifiers. It trains individual GRNN (ensemble members) independently using different feature subsets formed by SAGA and then integrates the outputs from these base GRNN

classifiers to produce a combined output. The main novelties of our ensemble neural networks are:

(1) Our proposed algorithm SAGA is used to achieve automatic optimization of ensemble of models that enhances performance by improving the configuration of the ensemble network. Unless an optimal subset of base classifiers has been selected, an ensemble of classifiers will be no better than a single classifier. Conventional search algorithms are plagued with the problems of premature convergence, slow iterations to reach the global optimal solution and getting stuck at a local optimum that prevent them from being effective search algorithms. In contrast, our proposed search algorithm SAGA demonstrates the greater persistence and better performance.

(2) In order for our ensemble network to inherit the qualities of GRNN, we employ the GRNN as a base classifier (members of the ensemble). GRNN offers exceptional qualities found in no other algorithm. First off, it is nonparametric in the sense it makes no assumptions concerning the form of the underlying distributions and requires no prior knowledge of the dynamics. Second, due to the construction of a similarity matrix and the fuzzy clustering of patterns, it can detect even small alterations in patterns. Hence, when confronted with a fresh set of inputs for a new situation, GRNN predicts outcomes with increasing reliability over time. Third, since it is a similarity based algorithm, it is less sensitive to potential outliers. This enables it to form the most "plausible" outputs (in relation to the learning that it has undergone) for any inputs. Fourth, GRNN has only one free parameter to adjust, which eases the process to set it

to give its best performance in a given application. Other algorithms have many free parameters to select and estimating them all at once may induce error.

(3) At present, most neural network ensemble approaches use simple majority voting or weighted voting of classifiers to combine single classifier votes. Majority voting does not consider the performance level of each individual classifier, therefore results may deteriorate because of poor performing classifiers. In weighted voting systems, the vote of each base classifier is weighted. A single scalar weight is assigned to each base classifier, where the weight of each classifier depends on the accuracy of the classifier on the training set. However, the true performance of a classifier may vary across population subgroups. The conventional weighted voting schemes do not make use of information specific to the instance when calculating the weights of the ensemble members. This may turn out to be a missed opportunity as this information may help to determine which base classifiers are likely to predict correctly. In our proposed ensemble network, the outputs from the ensemble of base classifiers are combined to produce an overall decision by employing another GRNN classifier. GRNN uses a dynamic instance-based weighting scheme based on the combination of fuzzy clustering and weighted voting. Thus, instead of assigning a weight to a base classifier that is fixed for all instances, our proposed ensemble approach dynamically assigns weights to base classifiers based on how well they are expected to predict on the output of the test instance.

Two tailor-made algorithms (GEMI and GEFTS), based on our proposed fuzzy ensemble approach, are developed to impute missing values and to forecast the

univariate time series. Both problems are subject to the curse of dimensionality. Imputation models have to take into account the nature of dependencies of the variable of interest on the observed variables as well as on missing data mechanisms of samples. This substantially increases the effective dimensionality (the dimensionality of the smallest feature subset that adequately represents the data) of the problem. On the other hand, in the univariate time series forecasting process, the dependent variable Z_t is a function of current and lagged values of Z_t. Consequently, 'the curse of dimensionality' begins to set in very quickly and the accuracy of the algorithm gets worse.

Our proposed algorithm GEMI is a multiple imputation algorithm that utilizes an iterative method for imputing missing values. By contrast, our proposed univariate forecasting algorithm GEFTS is a one-pass algorithm that learns two ensemble models, one for forecasting future values of the series and one for future volatilities. We find that GEFTS is able to model seasonality directly and prior deseasonalization is not necessary.

Manual design of neural network ensembles is a complex task. A great advantage of using GEMI and GEFTS is that they can automatically evolve ensembles with the help of SAGA. In our proposed algorithms, the designers need to specify just a few parameters. The main parameter of GEMI and GEFTS is the width of the Gaussian function. Our research suggests that there exists one best default value for the spread parameter. This default value is obtained by setting each centre's width to two times the

average Euclidean distance to 20 nearest members. A decisive factor in the choice of remaining parameter values depends on how much storage and computational cost can be afforded. However, we have defined default values for all parameters. The empirical evidence suggests that default values are almost always a good choice.

We compare our algorithms (SAGA, GEMI, and GEFTS) with existing best-known algorithms. The experimental results highlight the effectiveness of our proposed algorithms.

The proposed algorithms (GEMI and GEFTS) are robust against sample size effects. Hence, we encourage the use of the proposed framework without regarding the size of the sample, because we never know whether the sample size is sufficient for a reliable inference.

7.2 Concluding Remarks and Recommendations

. We propose new machine learning algorithms to resolve three major challenges facing the data mining: (1) feature subset selection in large dimensionality domains, (2) reconstruction of incomplete datasets, and (3) univariate time series forecasting. The simulation results show that our proposed algorithms are better than existing algorithms. However, these algorithms were tried and evaluated on a small number of datasets. Hence, the extensive evaluation of proposed algorithms using data from a wide variety of areas of applications would be an interesting topic for future research.

Data Mining is still in its infancy and virtually there is no existing algorithm that meets all criteria (accuracy, simplicity, and versatility) of an ideal machine learning

algorithm. However, the results of our research demonstrate that these not-so-good existing machine learning algorithms can be converted to a near-optimal algorithm through careful mixing. For instance, let us take a look at three existing search algorithms—SA, GA, and hill-climbing. SA seems to have been lost from the literature since it can be painfully slow. But, this search algorithm has a remarkable strength. SA exhibits asymptotic convergence (that is, as run time approaches infinity, the probability of finding an optimal solution approaches one) because SA search is robust against premature convergence. GA is faster than SA, but is highly susceptible to premature convergence to local optima. Hill-climbing, on the other hand, is the fastest known search algorithm. But this search converges to the nearest local minimum. However, SA, GA and hill-climbing algorithm together is a winning combination to attain the desired result.

We have rediscovered the potential of the Generalized Regression Neural Networks (GRNN) in our work. Generalized Regression Neural Networks (GRNN) uniquely possesses most of the desirable characteristics that an ideal machine learning algorithm should have. GRNN is a nonparametric algorithm that can be used to quickly construct accurate non-linear models. It is a local approximation algorithm. The benefit from being a local approximation method is that it accurately predicts heterogeneous phenomena. GRNN is arguably the simplest algorithm that contains only one free parameter. However, the GRNN has not really taken off as expected. Although GRNN is a powerful machine learning algorithm, many of its benefits are often intangible and difficult to prove because of its "downside" in terms of its higher sensitivity to the negative effects of high-dimensionality. To address this, we propose a two-step

approach. First, using the proposed feature subset selection algorithm (SAGA), the dimension of the working space should be reduced significantly. Second, a GRNN ensemble, instead of a single GRNN, is to be used as a machine learning algorithm. We have successfully applied the proposed scheme to impute missing values and also to forecast univariate time series. We recommend the use of this proposed ensemble method in other data mining tasks to test its effectiveness.

7.3 Promising Research Ideas

We always look for opportunities of advancement. Although the proposed algorithms significantly and substantially improve accuracy and robustness, we think there is still ample room for further advancement. We discuss briefly in this section a number of major improvement opportunities that deserve attention.

7.3.1 Possible Improvements to the Proposed Feature Subset Selection Algorithm (SAGA): SAGA sequentially applies three conventional search algorithms (simulated annealing, then genetic algorithm, and finally hill climbing algorithm). SA allows SAGA to escape local optima. Hence, exploring new regions in the search space is number one priority for SA. Currently, the initial population of SAGA, randomly distributed throughout the problem space, are allowed to freely explore regions with the aid of the SA. SAGA incorporates a genetic algorithm search on the k best-to-date solutions achieved by the SA. A limitation of our scheme is that there might be some promising areas in the solution space which have been only minimally explored, or have not been explored at all. One way we plan to overcome this limitation in future experiments is to partition the search space into k disjoint subspaces. During simulated

annealing search, SAGA will randomly pick one individual from each subspace and restrict its search to the subspace in order to allow a uniform search across all subspaces. Upon the end of the simulated annealing search, the personal best solution achieved by each individual will be selected to form the initial population for the GA. Finally, SAGA will launch a local search by hill-climbing on a number of global best solutions as it currently does. This strategy should offer a uniform global search capability in the given search space.

7.3.2 Possible improvements to the Proposed Ensemble Network with GRNN (GEMI and GEFTS) Classifiers: The development of an ensemble of classifiers entails addressing two issues: the selection of base classifiers that constitute the ensemble and the assignment of a vote weight to each base learner. Our proposed ensemble neural network consists of two layers of GRNN. In the first layer, there are multiple base classifiers (GRNN), each trained on different subsets of features. The second layer contains a single GRNN that constructs the ensemble output based on the predictions of base classifiers. Since GRNN is a local approximation algorithm, the combiner GRNN can adjust the weighting of each GRNN in the first layer according to whether there is evidence suggesting that it may predict well for that specific test instance. Therefore, our proposed ensemble neural network dynamically assigns weights to the individual base learner based on a given input pattern. But still, our proposed algorithm is not completely dynamic. A major drawback of our ensemble learning scheme is that the selection of base learners (i.e. ensemble feature selection) is fixed for all instances. It would be beneficial to have an ensemble method that also dynamically selects an optimum mix of base learners across hundreds of trained GRNN

based on a given input pattern. In order to do this, we have to partition the instance space into regions and then we have to build one ensemble model for each region separately. For a given new query, we will assign the closest cluster to the query, and then we will apply the ensemble network of GRNN associated to the cluster. Applying dynamic selection of base classifiers as well as using dynamic weights for base classifiers to introduce a greater sensitivity to local variations, we could improve the performance of the proposed ensemble networks further.

Reference:

1. Lyman, P., Varian, H.R. (2003). How *Much Information.* Available at: http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/ (Accessed 2 June, 2009).
2. Nwana, H.S., Azarmi, N., & Smith, R. (1997). The rise of Machine Intelligence. *Lecture Notes in Computer Science*, 1198, 251-261.
3. Guyan, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(2003), 1157-1182.
4. Mitra, P., Murthy, C.A., & Pal, S.K. (2002). Unsupervised feature selection using feature similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24, 301-312.
5. Robnik-Sikonja, M., & Kononenko, I. (2003). Theoretical and empirical analysis of Relief and ReliefF. *Machine Learning*, 53, 23-69.
6. Dash, M., Choi, K., Scheuermann, P., & Liu, H. (2002). Feature selection for clustering—A Filter Solution. *In Proceedings of the Second IEEE International Conference on Data Mining (ICDM'02)*, IEEE Computer Society Washington, DC, USA, 2002, pp. 115-122.
7. Hastie, T., Tibshirani, R., & Friedman, J.H. (2001). *The Elements of Statistical learning.* (2nd ed.). Springer.
8. Bell, D. A., & Wang, H. (2004). A formalism for relevance and its application in feature subset selection. *Machine Learning*, 41,175-195.
9. Parzen, E. (1982). ARARMA models for time series analysis and forecasting. *Journal of Forecasting*, 1, 67-87.
10. Albrecht, A.A. (2006). Stochastic local search for the feature set problem, with applications to microarray data. *Applied Mathematics and Computation*, 183, 1148-1164.
11. Rubin, D.B. (1987). *Multiple imputation for nonresponse in surveys*. New York: Wiley.
12. Murray, C.D., & Dermott, S.E. (1999). *Solar System Dynamics*. Cambridge University Press.
13. Tso, G.K.F., & Yao, K.K.W. (2007). Predicting electricity energy consumption: A comparison of regression analysis, decision tree and neural networks. *Energy*, 32(9), 1761-1768.
14. Rooney, N., Patterson, D., & Galushka, M. (2004). A comprehensive review of recursive Naïve Bayes Classifiers, *Intelligent Data Analysis*, 8(6), 615-628.
15. Kothari, R., & Dong, M. (2000). Decision trees for classification: A review and some new results. *CiteSeer*$^{X}_{beta}$.
16. Manocha, S., & Girolami, M.A. (2007). An empirical analysis of the probabilistic *K*-nearest neighbour classifier. *Pattern Recognition Letters.* 28 (13), 1818-1824.
17. Hand, D.J., & Yu, K. (2001). Idiot's Bayes—not so stupid after all? *International Statistical Review*, 69(3), 385-399.
18. Zheng, Z. (1998). Naïve Bayesian classifier committees. *Lecture Notes in Computer Science*, 1398, 196-207.

19. Dudani, S.A. (1976). A note on distance-weighted k-nearest neighbour rules. *IEEE Transactions on Systems, Man, and Cybernetics*, 8, 311-313.
20. Wang, J., Neskovic, P., Cooper, L.N. (2005). Neighborhood size selection in the *k*-nearest-neighbor rule using statistical confidence. *Pattern Recognition*, 39(3), 417-423.
21. Cheng, B., & Titterington, D.M. (1994). Neural Networks: A review from a statistical perspective. *Statistical Science*, 9(1), 2-30.
22. Cherkassky, V., Shao, X., Mulier, F.M., & Vapnik, V.N. (1999). Model control for regression using VC generalization bounds, *IEEE Transactions on Neural Networks*, 10(5), 1075-1089.
23. Vapnik, V.N. (1995).*The Nature of Statistical Learning Theory*, New York: Springer-Verlag.
24. Vapnik, V.N. (1998). *Statistical Learning Theory*. New York: Wiley.
25. Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge, U.K.: Cambridge Univ. Press.
26. Tay, F.E.H., & Cao, L.J. (2003). Support vector machine with adaptive parameters in financial time series forecasting. *IEEE Transactions on Neural Networks*, 14(6), 1506-1518.
27. Zhou, Z., Jiang, Y., & Chen, S. (2003). Extracting symbolic rules from trained neural network ensembles, *AI Communications*, 16(1), 3-15.
28. Tang, J., Sun, Z., & Zhu, J. (2005). Using ensemble information in swarming artificial neural networks. *Lecture Notes in Computer Science*, 3496, 515-519.
29. Dietterich, T.G. (1997). Machine learning research: Four current directions. *AI Magazine*, 18(4), 97-136.
30. Ho, T.K. (1998). The random subspace method for constructing decision foests. *IEEE Transactions on Pattern Analysis Machine Intelligence,* 20(8), 832-844.
31. Chapin, J.K., & Nicolelis, M.A.L. (1999). Principal component analysis of neuronal ensemble activity reveals multidimensional somatosensory representations. *Journal of Neuroscience Methods*, 94(1), 121-140.
32. Kim, K., & Cho, S. (2006). Ensemble classifiers based on correlation analysis for DNA microarray classification, *Neurocomputing*, 70(1-3), 187-199.
33. Dondeti, S., Kannan, K., & Manavalan, R. (2005). Genetic algorithm optimized neural networks ensemble for estimation of mefenamic acid and paracetamol in tablets. *Acta Chim. Slov.*, 52, 440-449.
34. Chen, Y., & Zhao, Y. (2008). A novel ensemble of classifiers for microarray data classification. *Applied Soft Computing*, 8(4), 1664-1669.
35. Tsymbal, A., Puuronen, S., Patterson, D. (2002). Feature selection for ensembles of simple Bayesian classifiers. *Lecture Notes in Computer Science*, 2366, 1611-3349.
36. Bauer, E., & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: bagging, boosting, and variants. *Machine Learning*, 36(1,2), 105-139.
37. Dimitrakakis, C., Bengio, S. (2005). Online adaptive policies for ensemble classifiers. *Neurocomputing*, 64, 211-221.
38. Shen, Z., & Kong, F. (2004). Optimizing weights by Genetic Algorithm for Neural Network Ensemble. *Lecture Notes in Computer Science*, 3173, 323-331.

39. Chen, Y., Yang, B., & Abraham, A. (2007). Flexible neural trees ensemble for stock index modelling, *Neurocomputing*, 70, 697-703.
40. Jacobs, R.A., & Jordan, M.I. (1991). Adaptive mixture of local experts. *Neural Computation*, 3, 79-87.
41. Zhou, N., Wang, L. (2007). A modified t-test feature selection method and its application on the HapMap Genotype Data. *Geometrics, Proteomics & Bioinformatics*, 5(3-4), 242-249.
42. Liao, C., Li, S., & Luo, Z. (2007). Gene selection using Wilcoxon rank sum test and support vector machine for cancer. *Lecture Notes in Computer Science*, 4456, 57-66.
43. Jin, X., Xu, A., Bie, R., & Guo, P. (2006). Machine learning techniques and chi-square feature selection for cancer classification using SAGE gene expression profiles. *Lecture Notes in Computer Science*, 3916, 106-105.
44. Kira, K., & Rendell, L. (1992). A practical approach to feature selection. *In Proceedings of the 9th International Workshop on Machine Learning*, pp. 249-256.
45. Lutu, P.E., & Engelbrecht, A.P. (2010). A decision rule-based method for feature selection in predictive data mining, *Expert Systems with Applications*, 37(1), 602-609.
46. Tourassi, G.D., Frederick, E.D., & Markey, M.K. (2001). Application of the mutual information criterion for feature selection in computer-aided diagnosis. *Medical Physics*, 28(12), 2394-2402.
47. Yu, L., & Liu, H. (2004). Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research*, 5, 1205-1224.
48. Rocchi, L., Chiari, L., & Cappello, A. (2004). Feature selection of stabilometric parameters based on principal component analysis. *Medical and Biological Engineering and Computing*, 42(1), 71-79.
49. Saes, Y., Inza, I., & Larranaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(9), 2507-2517.
50. Somol, P., Baesons, B., Pudil, P., & Vanthienen, J. (2005). Filter-versus wrapper-based feature selection for credit scoring. *International Journal of Intelligent Systems*, 20, 985-999.
51. Freitas, A.A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. New York: Springer-Verlag.
52. Colak, S., & Isik, C. (2003). Feature subset selection for blood pressure classification using orthogonal forward selection. *In Proceedings of 2003 IEEE 29th Annual Bioengineering Conference*, 22-23 March 2003, 122-123.
53. Cotter, S.F., Kreutz-Delgado, K., & Rao, B.D. (2001). Backward sequential elimination for sparse vector selection. *Signal Processing*, 81(9), 1849-1864.
54. Yang, J., & Honavar, V. (1998). Feature subset selection using genetic algorithm. *IEEE Intelligent Systems and their Applications*, 13(2), 44-49.
55. Pudil, P., & Novovicov, J., & Kittler, J. (1994). Floating search methods in feature selection. *Pattern Recognition Letters*,15(11), 1119-1125.
56. Bensch, M., Schroder, M., Bogdan, M., Rosenstiel, W., Czerner, P., Montino, R., Soberger, G., Linke, P., & Schmidt, R. (2005). Feature selection for high-dimensional industrial data. *ESANN 2005*, Brugge, 27-29 April 2005.

57. Ng., H.T., Goh, W.B., & Low, K.L. (1997). Feature selection, perceptron learning, and a usability case study for text categorization. *In Proceedings of 20th International Conference on Research and Development in Information Retrieval*, Philadelphia, 27-31 July 1997, 67-73.
58. Robbins, K.R., Zhang, W., & Bertrand, J.K. (2007). The ant colony algorithm for feature selection in high-dimension gene expression data for disease classification. *Mathematical Medicine and Biology*, 24(4), 413-426.
59. Liu, Y., Qin, Z., Xu, Z., & He, X. (2005). Feature selection with particle swarms. *Lecture notes in computer science*, 3314, 425-430.
60. Debuse, J.C.W., & Smith, V.J.R. (1997). Feature subset selection within a simulated annealing data mining algorithm. *Journal of Intelligent Information Systems*, 9(1), 57-81.
61. Shazzad, K.M., & Park, J.S. (2005). Optimization of intrusion detection through fast hybrid feature selection. *In Proceedings of Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05),* Dalian, China, 05-08 December 2005.
62. Tan, F., Fu, X., Wang, H., & Zhang, Y., & Bourgeois, A. (2006). A hybrid feature selection approach for microarray gene expression data, *Lecture Notes in Computer Science,* 3992, 678-685.
63. Zhang, L., Wang, L.; Zhao, Y., & Yang, Z. (2003). A novel hybrid feature selection algorithm: using ReliefF estimation for Ga-Wrapper search. *In Proceedings of 2003 International Conference on Machine Learning and Cybernetics*, 1, 380-384.
64. Yan, Z., & Yuan, C. (2004). Ant colony optimization for feature selection in face recognition. *Lecture notes in Computer Science,* 3072, 221-226.
65. Osei-Bryson, K-M, Giles, K., & Kositanurit, B. (2003). Exploration of a hybrid feature selection algorithm, *The Journal of the Operational Research Society,* 54(7), 790-797.
66. Huang, J., Cai, Y., Xu, X. (2007). A hybrid genetic algorithm for feature selection based on mutual information. *Pattern Recognition Letters*, 28(13), 1825-1844.
67. Jiang, B., Ding, X., Ma, L., He, Y., Wang, T., & Xie, W. (2008). A hybrid feature selection algorithm: Combination of symmetrical uncertainty and genetic algorithms. *In Proceedings of the Second International Symposium on Optimization and Systems Biology (OSB'08)*, Lijiang, China, October 31-November 3, 2008, 152-157.
68. Little, R.J.A., & Rubin, D.B. (1987). *Statistical analysis with missing data.* New York: Wiley.
69. Schafer, J. (1997). *Analysis of incomplete multivariate data.* London: Chapman and Hall.
70. Miettinen, O.S. (1985*). Theoretical epidemiology: Principles of occurrence research in medicine*. New York: Wiley.
71. Tuikkala, J., Elo, L.L., Nevalainen, O., & Aittokallio, T. (2008). Missing value imputation improves clustering and interpretation of gene expression microarray data. *BMC Bioinformatics*, 9(202).

72. Hawthorne, G., & Elliott, P. (2005). Imputing cross-sectional missing data: comparison of common techniques. *Australian and New Zealand Journal of Psychiatry*, 37(7), 583-590.
73. Bo, T.H., Dysvik, B., & Jonassen, I. (2004). LSImpute: accurate estimation of missing values in microarray data with least squares method. *Nucleic Acids Research*, 32(3).
74. Carlo, G., & Yao, J. (2003). A multiple-imputation metropolis version of the EM algorithm. *Biometrika*, 90(3), 643-654.
75. Hobert, J.P., & Marchev, D. (2008). A theoretical comparison of the data augmentation, marginal augmentation and PX-DA algorithms. *Annals of Statistics*, 2, 532-554.
76. Penell, S. (1993). Cross-sectional imputation and longitudinal editing procedures in the survey of income and program participation. *Technical report*, Institute of Social Research, University of Michigan, Ann Arbor.
77. Chen, J., & Shao, J. (2000). Nearest neighbour imputation for survey data. *Journal of Official Statistics*, 16(2), 113-131.
78. Ling, W., & Dong-Mei, F. (2009). Estimation of missing values using a Weighted K-Nearest Neighbors Algorithm. *In Proceedings of 2009 International Conference on Environmental Science and Information Application Technology*, 3, 660-663.
79. Lingras, P.J., Zhong, M., & Sharma, S.C. (2003). Evolutionary regression and neural imputations of missing values. *Studies in Fuzziness and Soft Computing*, 226, 151-163.
80. Zufiria, P.J., & Rivero, C. (2002). EM-based radial basis function training with partial information. *Lecture Notes in Computer Science*,2415, 138.
81. Rafat, A. (2004). Locally tuned general regression for learning mixture models using small incomplete data sets with outliers and overlapping classes. *Lecture Notes in Computer Science*, 3177, 774-779.
82. Tomandi, D., & Schober, A. (2001). A modified general regression neural network (MGRNN) with new, efficient training algorithms as a robust 'black-box' tool for data analysis. Neural Networks, 14(8), 1023-1034.
83. Bratu, C.V., Muresan, T., & Potolea, R. (2008). Improving classification performance on real data through imputation, *In Proceedings of 2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, Cluj-Napoca, Romania, May 22-May 25, 2008.
84. Hopke, P.K., Liu, C., & Rubin, D.B. (2001). Multiple imputation for multivariate data with missing and below-threshold measurements: time series concentrations of pollutants in the Arctic. *Biometrics*, 57(1), 22-33.
85. Rubin, D.B. (1978). *Multiple imputations in sample surveys*. New York: Wiley.
86. Rubin, D.B., & Schenker, N. (1986). Multiple imputation for interval estimation from simple random samples with ignorable response. *Journal of the American Statistical Association*, 81(394), 366-374.
87. Schafer, J.L., & Olsen, M.K. (1998). Multiple imputation for multivariate missing-data problems: a data analyst's perspective. *Multivariate Behavioral Research*, 33(4), 547-571.
88. Fay, R.E. (1996). Alternative paradigms for the analysis of imputed survey data. *Journal of the American Statistical Association*. 91(434), 490-498.

89. Kim, J.K., & Fuller, W. (2004). Fractional hot deck imputation. *Biometrika,* 91(3), 559-578.
90. Brocklebank, J.C., & Dickey, D.A. (2003). *SAS for forecasting time series*. (2nd ed.). New York: Wiley.
91. Li, W., Liu, J., & Le, J. (2005). Using GARCH-GRNN model to forecast financial time series. *Lecture Notes in Computer Science,* 3733, 565-574.
92. Tikunov, D., & Nishimura, T. (2007). Traffic prediction for mobile network using Holt-Winter's exponential smoothing. *In Proceedings of 15th International Conference on Software, Telecommunications and Computer Networks*, 27-29 Sept, 2007, pp.1-5.
93. Hansen, J.V., McDonald, J.B., & Nelson, R.D. (2002). Time series prediction with genetic-algorithm designed neural networks: An empirical comparison with modern statistical models. *Computational Intelligence,* 15(3), pp. 171-184.
94. Wang, D., Huang, J., Lan, W., & Li, X. (2008). Neural network-based robust adaptive control of nonlinear systems with unmodeled dynamics. *Mathematics and Computers in Simulation*, 79(5), 1745-1753.
95. Hill, T., O'Connor, M., & Remus, W. (1996). Neural network models for time series forecasts. *Management Science*, 42(7), 1082-1092.
96. Nelson, M., Hill, T., Remus, W., & O'Connor, M. (1999). Time series forecasting using neural networks: Should the data be deseasonalized first? *Journal of Forecasting*, 18(5), 359-367.
97. Gorr, W.L. (1994). Research perspective on neural network forecasting (Editorial). *International Journal of Forecasting*, 10(1), 5-15.
98. Franses, P.H. (1991). Seasonality, non-stationarity and the forecasting of monthly time series. *International Journal of forecasting*, 7, 199-208.
99. Kang, S. (1991). An investigation of the use of feedforward neural networks for forecasting. *PhD Dissertation*, Kent State.
100. Tang, Z., Almeida, C., & Fishwick, P.A. (1991). Time series forecasting using neural networks vs. Box-Jenkins methodology. *Simulation*, 57(5), 317-323.
101. Marseguerra, M.. Minoggio, S., Rossie, A., & Zio, E. (1992). Neural networks prediction and and fault diagnosis applied to stationary and Non stationary ARMA modelled time series. *Progress in Nuclear Energy*, 27(1), 25-36.
102. Tasakaya-Temizel, T., & Casey, M.C. (2005). Configuration of neural networks for the analysis of seasonal time series. *Lecture Notes in Computer Science*, 3686, 297-304.
103. Zhang, G.P., & Qi, M. (2005). Neural network forecasting for seasonal and trend time series. *European Journal of Operational research*, 160, 501-514.
104. Tseng, F.M., Yu, H.C., & Tzeng, G.H. (2002). Combining neural network model with seasonal time series ARIMA model. *Technological Forecasting and Social Change*, 69, pp. 71-87.
105. Virili, F., & Freisleben, B. (2000). Nonstationarity and data preprocessing for neural network predictions of an economic time series. *In Proceedings of International Joint Conference on Neural Networks*, 5, p.129.

106. Chu, C., & Zhang, G.P. (2003). A comparative study of linear and nonlinear models for aggregate retail sales forecasting. *International Journal of Economics*, 86, 217-231.
107. Pajares, R. G., Bentez, G.S., & Palmero, I. (2008). Feature selection for time series forecasting: A case study. *In Proceedings of 2008 Eighth International Conference on Hybrid Systems,* 10-12 September, 2008.
108. Parviz, R.K., Nasser, M., & Motlagh, M.R.J. (2008). Mutual information based input variable selection algorithm and wavelet neural network for time series prediction. *Lecture Notes in Computer Science*, 5163, 798-807.
109. Amjady, N., & Daraeepour, A. (2009). Design of input vector for day-ahead price forecasting of electricity markets. *Expert Systems with Applications*, 36(10), 12281-12294.
110. Lee, M. (2009). Using support vector machine with a hybrid feature selection method to the stock trend prediction. *Expert Systems with Applications*, 36(8), 10896-10904.
111. Tikka, J., & Hollmen, J. (2008). Sequential input selection algorithm for long-term prediction of time series. *Neurocomputing*, 71(13-15), 2604-2615.
112. Chen, L., & Hsiao, H. (2008). Feature selection to diagnose a business crisis by using a real GA-based support vector machine: An empirical study. *Expert Systems with Applications*, 35(3), 1145-1155.
113. Kalapanidas, E., & Avouris, N. (2003). Feature selection for air quality forecasting: a genetic algorithm approach. *AI Communications*, 16(4), 235-251.
114. Jha, G.K., Thulasiraman, P., & Thulasiram, R.K. (2009). PSO based neural network for time series forecasting. *In Proceedings of 2009 International Joint Conference on Neural networks*, Atlanta, GA, USA, 14-19 June, 2009, pp. 1422-1427.
115. Morchen, F. (2003). Time series feature extraction for data mining using DWT and DFT. *Technical Report No 33*, Department of Mathematics and Computer Science, University of Marburg, Germany, Available at: http://www.mybytes.de/papers/moerchen03time.pdf (accessed 8 October, 2009).
116. Shiblee, M., Kalra, P.K., & Chandra, B. (2009). Time series prediction with multilayer perceptron (MLP): A new generalized error based approach. *Lecture Notes in Computer Science*, 5507, 37-44.
117. Lipinski, P. (2007). Discovering stock market trading rules using multi-layer perceptrons. *Lecture Notes in Computer Science*, 4507, 1114-1121.
118. Husken, M.; Stagge, P. 2003. Recurrent neural networks for time series classification. Neurocomputing, 50, 223-235.
119. Kumar, D.N., Raju, K.S., & Sathish, T. (2004). River flow forecasting using recurrent neural networks. *Water Resources Management*, 8(2), 143-161.
120. Adya, M., & Collopy, F. (1998). How effective are neural networks at forecasting and prediction? A review and evaluation. Journal of Forecasting, 17, 481-495.
121. Zhang, G., Patuwo, E.P., Hu, M.Y. (1998). Forecasting with artificial neural networks: the state of art. *International Journal of Forecasting*, 14, 35-62.

122. Faraway, J., & Chatfield, C. (1998). Time series forecasting with neural networks: a comparative study using the airline data. *Applied Statistics*, 47(2), 231-250.
123. Barrile, V., Cacciola, M., D'Amico, S., Greco, A., Morabito, F.C., & Parrillo, F. (2006). Radial basis function neural networks to foresee aftershocks in seismic sequences related to large earthquakes. *Lecture Notes in Computer Science*, 4233, 909-916.
124. Cigizoglu, H.K., & Alp, M. (2006). Generalized regression neural network in modelling river sediment yield. Advances in Engineering Software. 37(2), 63-68.
125. Haykin, S. (1999). Neural networks: A comprehensive foundation.(2nd
126. Tomandi, D., & Schober, A. (2001). A modified general regression neural network (MGRNN) with new, efficient training algorithms as a robust 'black-box' tool for data analysis. *Neural Networks*, 14(8), 1023-1034.
127. Zhang, G.P. (2003). Time series forecasting using a hybrid of ARIMA and neural network model. *Neurocomputing*, 50, 159-175.
128. Taskaya-Temizel, T., Casey, M.C. (2005). A comparative study of autoregressive neural network hybrids. *Neural Networks*, 18, 781-789.
129. Landassuri-Moreno, V.M., & Bullinaria, J.A. (2009). Neural network ensemble for time series forecasting. *In Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, Montreal, Quebec, Canada, 1235-1242.
130. Zhang, G.P. (2007). A neural network ensemble method with jittered training data for time series forecasting. *Information Sciences*, 177(23), 5329-5346.
131. Sivanandam, S.N., & Deepa, S.N. (2007). *Introduction to Genetic Algorithms*. Springer.
132. Dorigo, M. (1992). Learning and natural algorithms. *PhD thesis*, Politecnico di Milano, Italie.
133. Holland, J. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control and artificial intelligence*. MIT Press.
134. Zhong, J., Hu, X., Zhang, J., & Gu, M. (2005). Comparison of performance between different selection strategies on simple genetic algorithms. *In Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, 2, 1115-1121.
135. Correia, D.S., Gonvalves, C.V., Cunha, S.S., & Ferraresi, V.A. (2005). Comparison between genetic algorithms and response surface methodology in GMAW welding optimization. *Journal of Materials Processing Technology*, 160(1), 70-76.
136. Kennedy, J., & Eberhart, R.C. (1995). Particle swarm optimization. *In Proceedings of IEEE International Conference on Neural Networks*. 4, 1942-1948.

137. Wang, X., Yang, J., Teng, X., Xia, W., & Jensen, R. (2007). Feature selection based on rough sets and particle swarm optimization. *Pattern Recognition Letters*, 4(1), 459-471.
138. Kirkpatric, S., Gelatt, C.D., & Veccie, M.P. (1983). Optimization by simulated annealing. *Science*, 220, 671-680.
139. Tan, F., Fu, X., Wang, H., Zhang, Y., & Bourgeois, A. (2006). A hybrid feature selection approach for microarray gene expression data. *Lecture Notes in Computer Science*, 3992, 678-685.
140. Siegel, S., & Castellan Jr, N.J. (1988). *Nonparametric statistics: for the behavioural sciences.* (2nd ed.). New York: McGraw-Hill.
141. Kira, K., & Rendell, L. (1992). A practical approach to feature selection. *In Proceedings of the Ninth International Workshop on Machine Learning*, 249-256.
142. Box, G., & Jenkins, G. (1970). *Time series analysis: Forecasting and control*, San Francisco: Holden-Day.
143. Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31, 307-327.
144. Elman, J.L. (1990). Finding structure in time. *Cognitive Science*, 14, 179-211.
145. Specht, D.F. (1991). A general regression neural network. *IEEE Transactions on Neural Networks*, 20(6), 568-576.
146. Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). 'Learning internal representations by error propagation'. In *Parallel distributed processing: explorations in the microstructure of cognition*, MIT Press, Cambridge, MA, USA, 1, 318-362.
147. Broomhead D.S., & Lowe D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2, 321-355.
148. UCI Irvine Machine Learning Repository. [Online]. Available: http://archive.ics.uci.edu/ml/
149. A First Course on Time Series Analysis. [Online]. Available at: http://statistic.mathematic.uni-wuerzburg.de/timeseries/index.php
150. FreeLunch.com: Why Pay Anything. [Online]. Available at: http://www.economy.com/freelunch/default.asp
151. Vapnik, V., & Chervonenkis, A. (1974). *Theory of Pattern Recognition* [in Russian!], Moscow: Nauka.
152. Kranakis, E., Krizanc, D., Urrutia, J., & Woeginger, G.J. (1994). VC-dimensions for graphs. *CiteSeer*$^{X}_{beta}$.
153. Tormandl, D., & Schober, A. (2001). A modified general regression neural network (MGRNN) with new, efficient training algorithm as a robust 'black box' tool for data analysis. *Neural Networks*, 14 (8), 1023-1034.
154. Cooper, J., & Hinde, J. (2003). Improving genetic algorithms' efficiency using intelligent fitness functions. *Lecture Notes in Computer Science*, 2718, 1-58.
155. Valdovinos, R.M., & Sanchez, J.S. (2009). Combining multiple classifiers with dynamic weighted voting. *Lecture Notes in Artificial Intelligence*, 1, 510-516.

156. Eggleton, I. (1976). Intuitive time series extrapolation. *Journal of Accounting Research*, Supplement, 68-131.