

COMPARING GENERALISED ADDITIVE NEURAL NETWORKS WITH DECISION TREES AND ALTERNATING CONDITIONAL EXPECTATIONS

Susanna E. S. Campher

10974318

Dissertation submitted in partial fulfilment of the requirements for the degree
Master of Science in
Computer Science at the Potchefstroom Campus of the North-West University.

Supervisor:
Prof. D.A. De Waal

Potchefstroom

May, 2008



ACKNOWLEDGEMENTS

The completion of the requirements for a Master of Science degree while working fulltime has been one of the most challenging, but fulfilling undertakings of my career. I am grateful for the support of family and friends, my husband Andries in particular, who committed to more than his share of domestic responsibilities over the past three years.

I wish to thank my supervisor, Prof. André de Waal, who not only provided much appreciated guidance and encouragement but also taught me research skills and values. My interest in artificial intelligence and data-mining is owed to him and his commitment to it kept me focused. I am grateful for the advice received from Dr. Tiny du Toit whose own research was inspirational.

Appreciation is extended to Ms. Cecilia van der Walt for the language editing, to the SAS Institute for providing SAS[®] Enterprise Miner[™] and to MathWorks[™] for providing MATLAB[®] to students and researchers.

I am pleased to acknowledge Telkom, Grintek Telecom and THRIP for supporting research performed at the Telkom Centre of Excellence, North-West University.

Soli Deo gloria

ABSTRACT

In this dissertation generalised additive neural networks (GANNs), decision trees and alternating conditional expectations (ACE) are studied and compared as computerised techniques in the field of predictive data mining. De Waal and Du Toit (2003) developed the automated construction of GANNs called AutoGANN. The aim is to better understand the contribution that this new technology may bring to the field of data mining. Decision trees and ACE were chosen as comparative techniques since both have been praised for their performance as data analysis tools.

The ACE, AutoGANN and decision tree modelling methods are explained and applied to several regression problems. The methods and empirical results are compared and discussed in terms of criteria such as predictive capability or accuracy, novelty, utility, interpretability and stability. Finally, the conclusion is made that the AutoGANN system should contribute substantially to the field of data mining.

Keywords: Alternating conditional expectations; artificial neural networks; data mining; decision trees; generalised additive models; predictive models; regression; smoother.

UITTREKSEL

In hierdie verhandeling, getitel *Vergelyking van veralgemeende additiewe neurale netwerke met besluitnemingsbome en alternerende voorwaardelike verwagtinge*, word hierdie drie gerekenariseerde tegnieke binne die veld van data-ontginning bestudeer en vergelyk. De Waal en Du Toit (2003) het 'n geoutomatiseerde konstruksiemetode ontwikkel vir veralgemeende additiewe neurale netwerke genaamd *AutoGANN*. Die doel van die studie is om die moontlike bydrae wat hierdie nuwe tegniek tot die veld van data-ontginning kan maak, beter te verstaan. Besluitnemingsbome en alternerende voorwaardelike verwagtinge (*ACE*) is gekies as vergelykende tegnieke aangesien beide reeds in die literatuur as hoog aangeskrewe data-ontledingshulpmiddele bestempel is.

Die *ACE*, *AutoGANN* en besluitnemingsbome modelleringstegnieke word verduidelik en op verskeie regressieprobleme toegepas. Die metodes en empiriese resultate word vergelyk en bespreek ooreenkomstig kriteria soos voorspellingsvermoë en –akkuraatheid, uniekheid, utiliteit, interpreteerbaarheid en stabiliteit. Laastens word tot die gevolgtrekking gekom dat die *AutoGANN*-stelsel 'n aansienlike bydrae tot die veld van data-ontginning behoort te lewer.

Sleutelwoorde: Alternerende voorwaardelike verwagtinge; besluitnemingsbome; data-ontginning; gladstryker; kunsmatige neurale netwerke; regressie; veralgemeende additiewe modelle; voorspellingsmodelle.

CONTENTS

1	Introduction	1
1.1	Predictive data mining techniques	1
1.2	Comparison of modelling methods	2
1.3	Description of examples	2
1.3.1	Simulated examples	3
1.3.2	Ethanol fuel example	5
1.3.3	EXPCAR example	5
1.3.4	SO ₄ example	6
1.3.5	Boston Housing example	6
1.3.6	Ozone example	7
1.4	Outline	8
2	Alternating conditional expectations	9
2.1	Smoothers	9
2.2	The super-smoother	10
2.2.1	A linear fit as a measure of average	11
2.2.2	The use of a variable span	12
2.2.3	The need for bass enhancement	13
2.2.4	Implementation of the super-smoother	14
2.2.5	Example application of the super-smoother	15
2.3	The ACE procedure	17
2.3.1	An iterative procedure for variable transformation	17
2.3.2	Creating the additive model	19
2.3.3	Implementation of the ACE procedure	20
2.4	ACE examples	22
2.4.1	Simulated examples	23
2.4.2	Ethanol fuel example	28
2.4.3	EXPCAR example	29
2.4.4	SO ₄ example	31
2.4.5	Boston Housing example	32
2.4.6	Ozone example	36

2.5	Conclusions.....	38
3	Automated generalised additive neural networks.....	39
3.1	Generalised additive models	39
3.2	Generalised additive neural networks.....	40
3.3	Partial residual plots	44
3.4	Constructing a generalised additive neural network.....	45
3.5	Automated construction of generalised additive neural networks.....	46
3.5.1	Selection of activation functions	46
3.5.2	GANN architecture encoding	47
3.5.3	Model selection criteria.....	48
3.5.4	Search tree expansion method.....	49
3.5.5	Improvements on performance.....	49
3.5.6	An intelligent start.....	50
3.5.7	Model averaging.....	53
3.5.8	The AutoGANN implementation	55
3.6	AutoGANN examples	59
3.6.1	Simulated examples	60
3.6.2	Ethanol fuel example.....	64
3.6.3	EXPCAR example	65
3.6.4	SO ₄ example	66
3.6.5	Boston Housing example.....	69
3.6.6	Ozone example	73
3.7	Conclusions.....	75
4	Decision trees.....	76
4.1	An overview of decision trees	76
4.2	The cultivation of decision trees	77
4.2.1	Split search	77
4.2.2	Splitting criteria.....	78
4.2.3	Stopping and pruning rules.....	79
4.3	Decision tree examples	80
4.3.1	Simulated examples	81
4.3.2	Ethanol fuel example	83
4.3.3	EXPCAR example	83
4.3.4	SO ₄ example	84

4.3.5 Boston Housing example.....	84
4.3.6 Ozone example	85
4.4 Conclusions.....	85
5 Prediction problem examples.....	86
5.1 Simulated examples	86
5.1.1 First simulated example.....	86
5.1.2 Second simulated example.....	88
5.1.3 Third simulated example	89
5.2 Ethanol fuel example.....	92
5.3 EXPCAR example.....	94
5.4 SO ₄ example	95
5.5 Boston Housing example.....	97
5.6 Ozone example	98
5.7 Conclusions.....	100
6 Comparative discussion on ACE, AutoGANN, and decision trees	101
6.1 Predictive accuracy	101
6.2 Utility	101
6.3 Stability	103
6.4 Understandability and interpretability.....	104
6.5 Novelty.....	104
6.6 Additional comments.....	105
7 Conclusions.....	106
Addendum – MATLAB® ACE code.....	108
Main functions	108
Function <i>ace</i>	108
Function <i>aceModel</i>	110
Function <i>aceScore</i>	110
Function <i>superSmoother</i>	111
Auxiliary functions	113
Function <i>aceInterp</i>	113
Function <i>mse</i>	114

(Contents continued)

Function <i>neighbourhoodInterval</i>	115
Function <i>smoothLeastSq</i>	115
Function <i>ssSort</i>	117
Function <i>ssResort</i>	117
Function <i>toMonotone</i>	118
Bibliography	119

1

INTRODUCTION

In an information-driven world, massive amounts of data are generated and collected on a daily basis. Analysis of large volumes of raw data to extract useful information has become a science. According to Walsh (2002), it is largely driven by business decision support functions – strategic marketing, credit scoring and fraud detection for example, and is concerned with finding patterns in order to anticipate or predict occurrences of events. Terms such as *knowledge discovery in databases* and *data mining* are widely used to describe this research area. The field of predictive data mining combined with computer technology is the subject of study.

Section 1.1 contains descriptions of the predictive data mining techniques that were selected for this comparative study. In Section 1.2 the means of comparing the modelling techniques are given. The prediction problem examples used throughout the study are described in Section 1.3. Finally, an outline of the rest of the dissertation is given in Section 1.4.

1.1 Predictive data mining techniques

In recent years there has been ongoing research and development of computerised methods for building predictive models. These include the use of artificial neural networks (ANNs). Zhang *et al.* (1998) conclude that, although extensive research has been done on ANNs, findings are inconclusive as to whether and when ANNs are better than other predictive modelling methods. ANNs are extremely flexible models and are considered to be general function approximators. They do pose a few difficulties, especially the uncertainties in interpreting the models (the underlying variable relationships depicted by the models) and the neural network architecture to be chosen for a specific problem.

De Waal and Du Toit (2003) developed a method for using ANNs which addresses both these problems – AutoGANN: automated construction of generalised additive neural networks. Generalised additive neural networks (GANNs) are based on additive models and are therefore a simplified family of neural networks compared to the more general multilayer perceptrons.

1

INTRODUCTION

In an information-driven world, massive amounts of data are generated and collected on a daily basis. Analysis of large volumes of raw data to extract useful information has become a science. According to Walsh (2002), it is largely driven by business decision support functions – strategic marketing, credit scoring and fraud detection for example, and is concerned with finding patterns in order to anticipate or predict occurrences of events. Terms such as *knowledge discovery in databases* and *data mining* are widely used to describe this research area. The field of predictive data mining combined with computer technology is the subject of study.

Section 1.1 contains descriptions of the predictive data mining techniques that were selected for this comparative study. In Section 1.2 the means of comparing the modelling techniques are given. The prediction problem examples used throughout the study are described in Section 1.3. Finally, an outline of the rest of the dissertation is given in Section 1.4.

1.1 Predictive data mining techniques

In recent years there has been ongoing research and development of computerised methods for building predictive models. These include the use of artificial neural networks (ANNs). Zhang *et al.* (1998) conclude that, although extensive research has been done on ANNs, findings are inconclusive as to whether and when ANNs are better than other predictive modelling methods. ANNs are extremely flexible models and are considered to be general function approximators. They do pose a few difficulties, especially the uncertainties in interpreting the models (the underlying variable relationships depicted by the models) and the neural network architecture to be chosen for a specific problem.

De Waal and Du Toit (2003) developed a method for using ANNs which addresses both these problems – AutoGANN: automated construction of generalised additive neural networks. Generalised additive neural networks (GANNs) are based on additive models and are therefore a simplified family of neural networks compared to the more general multilayer perceptrons.

Additive models are also easier to interpret than general function approximators or multilayer perceptrons. Optimal or near-optimal network architectures are chosen automatically through the use of objective model selection criteria together with a model search algorithm. The aim of the study is to gain a better understanding of the contribution this new technology may bring to the field of data mining. GANNs, and more specifically, the AutoGANN system, is compared to alternating conditional expectations (ACE) and decision trees, both of which have been praised as predictive data mining techniques. In their comment on ACE, Pregibon and Vardi (1985) from Bell Laboratories complimented Breiman and Friedman for providing the data analyst with a powerful tool and, with that, narrowing the gap between mathematical statistics and data analysis.

Decision trees and ACE are, like ANNs, considered to be multivariate function estimators which can be applied to almost any predictive data mining problem (Potts, 2000). Decision trees are not only highly flexible predictive models capable of modelling nonlinear trends; they are also generally easy to interpret and computationally fast. They are, however, considered to be unstable or sensitive to changes in the data on which the model was built (Potts, 2001).

ACE is a method that aims at finding the best fitting additive model for a prediction problem by means of estimating optimal variable transformations. Breiman and Friedman (1985) demonstrate ACE using a variable span smoother to estimate the conditional expectations used for the variable transformations.

1.2 Comparison of modelling methods

The criteria that are used to compare the data mining techniques include predictive capability or accuracy, novelty, utility, understandability or interpretability and stability. The means of assessment of these criteria are mainly through the direct application of the techniques to regression problems that serve as examples throughout the study. The metrics used in all the examples are the coefficient of determination (R^2) and the average squared error (ASE). Conclusions from existing studies are also considered.

1.3 Description of examples

A set of examples was selected to use through-out the comparative study. The examples include simulated and observed data sets that have previously been used in literature relating to data analysis using the three predictive modelling methods being studied, for example the Boston Housing data set (Harrison & Rubinfeld, 1978) and the SO_4 data set (Xiang, 2001). The

examples also vary in sample size, in the number of predictor variables and in the distribution of the dependant variable. This section contains a description of each of these examples.

1.3.1 Simulated examples

Simulated data sets are helpful in the testing and evaluation of data mining techniques. The analyst is able to simulate specific scenarios in variable relationships and the optimal data model is usually known. Three simulated examples are used by Breiman and Friedman (1984) to illustrate the ACE procedure. Data sets based on the same simulation models were generated and the ACE procedure, the AutoGANN system and decision trees were applied to these data sets.

For the first simulated example a set of 200 data points, $\{(y_i, x_i), 1 \leq i \leq 200\}$, was generated from a model with two independent variables, X and ε :

$$Y = \exp(X^3 + \varepsilon) \quad (1.1)$$

with x_i^3 and ε_i drawn randomly from standard normal distributions. The optimal variable transformations for this regression problem are $\ln(Y)$ and X^3 . Figure 1.1 contains a scatter plot of the data.

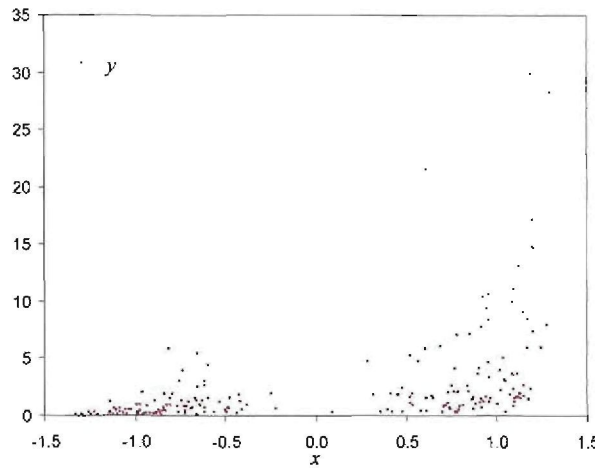


Figure 1.1: First simulated example - data scatter plot

For the second example a set of 200 data points, $\{(y_i, x_i), 1 \leq i \leq 200\}$, was generated from the model

$$Y = \exp[\sin(2\pi X) + \varepsilon/2] \quad (1.2)$$

with x_i drawn from a uniform distribution between 0 and 1, and ε drawn independently from a standard normal distribution. Figure 1.2 contains a scatter plot of the data. The variable

transformations $\ln(Y)$ and $\sin(2\pi X)$ are close to the optimal transformations (it is not optimal since $\sin(2\pi X)$ has a non-normal distribution).

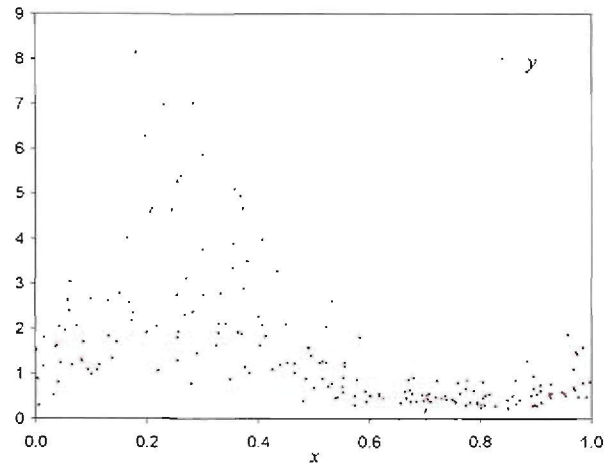


Figure 1.2: Second simulated example - data scatter plot

The third example consists of 200 data points, $\{(y_i, s_i, t_i), 1 \leq i \leq 200\}$, generated from

$$Y = ST \quad (1.3)$$

where s and t are two independent variables with uniform distributions between -1 and 1. Figure 1.3 and Figure 1.4 are scatter plots of y versus s and t respectively. Since y is positive when s and t have the same sign and negative when s and t have opposite signs, a 3-dimensional plot of the data produces a saddle-like scatter. The optimal transformations are $\ln|Y|$, $\ln|S|$ and $\ln|T|$. The absolute values are required because of the negative values the variables may assume.

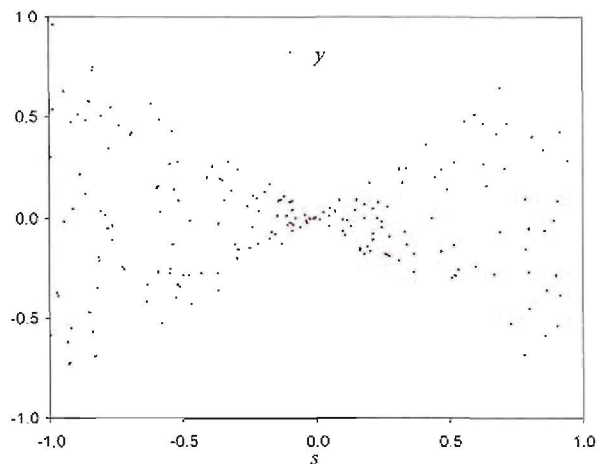


Figure 1.3: Third simulated example – scatter plot of y vs. s

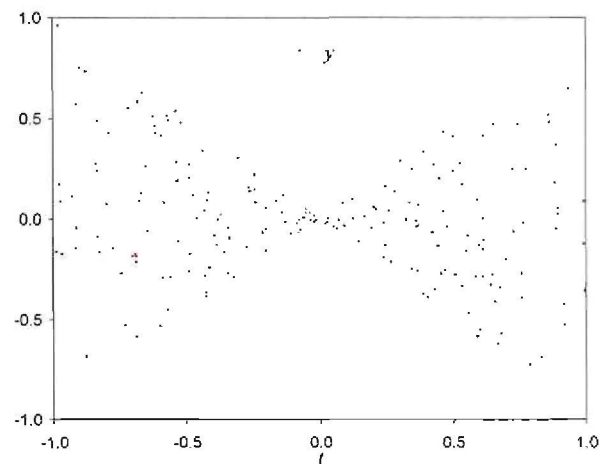


Figure 1.4: Third simulated example – scatter plot of y vs. t

1.3.2 Ethanol fuel example

The ethanol dataset (Brinkman, 1981) was used for this example. The predictor variable is E . It is the equivalence ratio that an engine was run at while burning ethanol. The response variable is NOX , the concentration of nitric oxide and nitrogen dioxide in the engine exhaust emissions. Figure 1.5 contains a scatter plot of the data, with 88 observations.

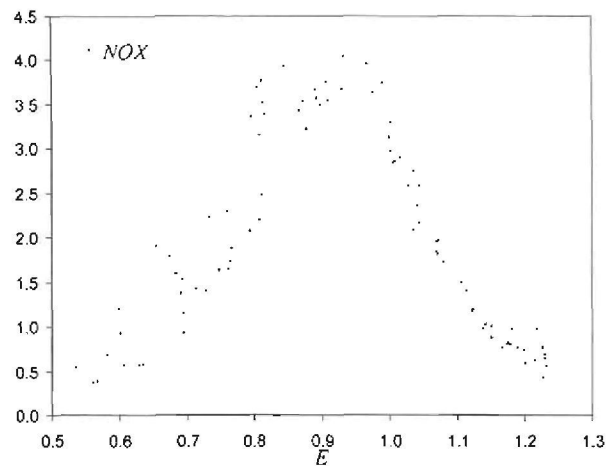


Figure 1.5: Ethanol fuel example – data scatter plot

1.3.3 EXPCAR example

The EXPCAR dataset (Potts, 2000) was used for this example. It is a data set consisting of 100 data points with predictor variable x and response variable y . Figure 1.6 contains a scatter plot of the data.

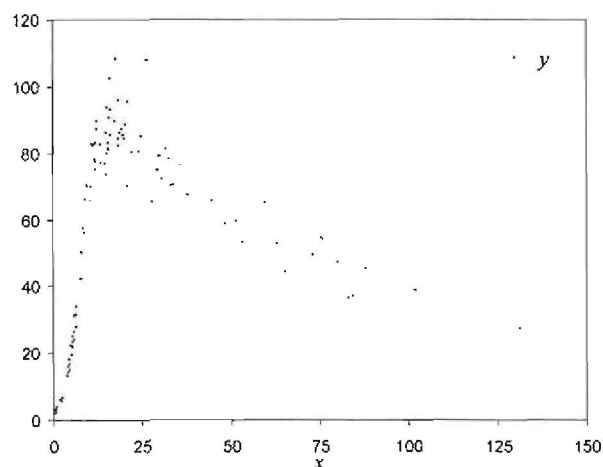


Figure 1.6: EXPCAR example – data scatter plot

1.3.4 SO_4 example

The SO_4 dataset (Xiang, 2001) with 179 observations has predictor variables LAT and LON . They represent the latitude and longitude respectively, of different locations in the United States. The response variable is SO_4 , the amount of sulphate deposits measured at the corresponding locations. Figure 1.7 contains a stem plot of the data set.

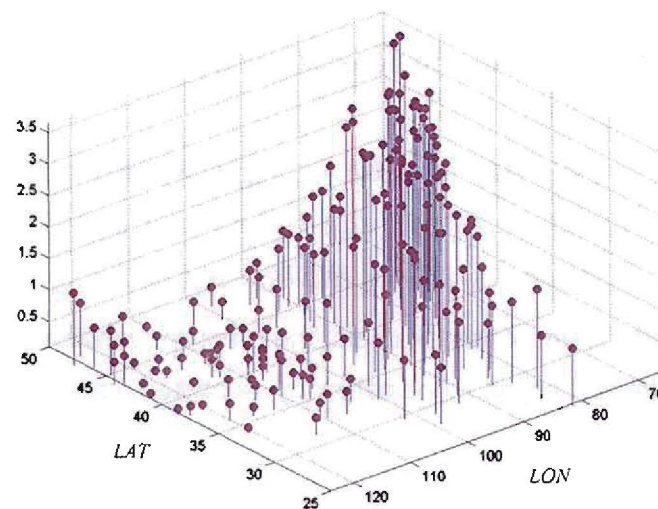


Figure 1.7: SO_4 example – data stem plot

1.3.5 Boston Housing example

The Boston Housing data set (Harrison & Rubinfeld, 1978) has 506 observations. The predictor variables are a set of housing attributes measured in the Boston (USA) metropolitan area around 1970. These attributes include structural characteristics of the homes in the neighbourhoods themselves, accessibility variables, neighbourhood characteristics and air pollution concentrations:

RM	average number of rooms in owner units;
AGE	proportion of owner units in the neighbourhood built prior to 1940;
B	proportion of the population that is black;
$LSTAT$	proportion of the neighbourhood population that is lower status;
$CRIM$	crime rate;
ZN	proportion of residential land zones for lots larger than 25 000 square feet;
$INDUS$	proportion of non-retail business acres;
TAX	full value property tax rate;
$PTRAT$	school pupil to teacher ratio;

<i>CHAS</i>	Charles river tract bound indicator;
<i>DIS</i>	weighted distance to Boston area employment centres;
<i>RAD</i>	index of accessibility to radial highways;
<i>NOX</i>	the concentration of nitrogen oxides (measure of air pollution).

The response variable is *MEDV*, the median value of the homes that are occupied by owners (from the 1970 USA census data).

Harrison and Rubinfeld suggested a basic equation for the problem which includes a few variable transformations:

$$\begin{aligned}\log(MEDV) = & a_1 + a_2(RM)^2 + a_3AGE + a_4 \log(DIS) + a_5 \log(RAD) + \\ & a_6TAX + a_7PTRAT + a_8(B - 0.63)^2 + a_9 \log(LSTAT) + \\ & a_{10}CRIM + a_{11}ZN + a_{12}INDUS + a_{13}CHAS + a_{14}(NOX)^2 + \varepsilon.\end{aligned}\tag{1.4}$$

During the further development of this example, the predictor variables are used as input with these transformations already applied (indicated by accented variable names, for example *MEDV*).

1.3.6 Ozone example

The Ozone data set is used by Breiman and Friedman (1985) to illustrate the application of ACE and, specifically, the use of variable transformation plots to help understand variable relationships.

Eight meteorological measurements together with the ozone concentration were taken over 330 days in the Los Angeles basin. The predictor variables are:

<i>SBTP</i>	Sandburg Air Force Base temperature;
<i>IBHT</i>	inversion base height;
<i>DGPG</i>	Dagget pressure gradient;
<i>VSTY</i>	visibility;
<i>VDHT</i>	Vandenburg 500 millibar height;
<i>HMDT</i>	humidity;
<i>IBTP</i>	inversion base temperature;
<i>WDSP</i>	wind speed;
<i>DOY</i>	day of year.

The response variable is *UPO3*, the upland ozone concentration.

1.4 Outline

The rest of the dissertation is organised as follows. In Chapter 2 the method of alternating conditional expectations is discussed. The chapter starts with a general description of a data smoother after which the super-smoother used by the ACE method is explained. A detailed description of ACE and a summary of a MATLAB® implementation are given after which the ACE results of the problem examples follow. Chapter 3 contains material on the AutoGANN system. Generalised additive models, generalised additive neural networks (GANNs), partial residual plots and the interactive construction of GANNs are discussed. The factors that are needed to be considered for the automated construction of GANNs are explained together with the implemented algorithm. The results are given of applying the SAS® Enterprise Miner™ AutoGANN model node to the problem examples. Decision trees are discussed in Chapter 4. An overview is given after which the cultivation of regression trees is explained in more detail. The results of applying the Decision Tree modelling node of SAS® Enterprise Miner™ to the examples are given. Chapter 5 contains the combined results of the three modelling methods. In Chapter 6 the modelling methods are compared. Finally, in Chapter 7 conclusions are drawn from the study.

2

ALTERNATING CONDITIONAL EXPECTATIONS

When developing linear regression models, analysts often make use of variable transformations as remedial measures when some of the model assumptions are violated. The process of selecting a transformation usually requires the analyst to make a subjective choice based on known guides.

The procedure of alternating conditional expectations, or ACE in short, was developed by Breiman and Friedman (1985) and was intended to be used as a tool to estimate the optimal transformations for multiple regression. It is a non-parametric procedure which uses data smoothers to estimate the conditional expectations.

This chapter is devoted to explaining the procedure and its implementation. Section 2.1 contains a high-level description of a smoother. The super-smoother is described in Section 2.2. The ACE procedure is explained in Section 2.3 and illustrated with several examples in Section 2.4.

2.1 Smoothers

Smoothers are often used to explore regression relationships between an independent variable X and a dependant variable Y , especially in cases where the relationship is difficult to be guessed (for example from a very complex scatter plot of the data). It is also used in multiple regression algorithms such as ACE and generalised additive models (Hastie & Tibshirani, 1986) which are based on the summation of individual unspecified univariate functions (Potts, 1999).

Smoothers are nonparametric regression methods used to determine a function that represents a generalisation of the dependent relationship of Y on X . The function is estimated from a set of observations representing the relationship and is referred to as the smooth. For the finite bivariate case of n observations, $(x_1, y_1), \dots, (x_n, y_n)$, smoothers produce a model of the form

$$y_i = s(x_i) + r_i \text{ for } i = 1 \dots n \quad (2.1)$$

where s is the smooth and r the residuals (Friedman, 1984).

Consider the function $f(X)$, defined to be the optimal transformation of X so that it is maximally correlated to Y . The function $f(X)$ is the conditional expectation of Y given x :

$$f(X) = E(Y|X = x). \quad (2.2)$$

When it is assumed that

$$Y = f(X) + \varepsilon \quad (2.3)$$

with ε being an independent random variable with expected value 0, the smooth, s , can be considered an estimate of f .

Various smoothing methods have been developed, mostly based on a variant of local averaging. Local averaging entails exploring a neighbourhood, N , containing J observations, around a specific value of X and calculating some measure for the average value of Y :

$$s(x_i) = M_{avg}(y_j | x_j \in N_i). \quad (2.4)$$

Examples of local averaging smoothing methods include moving average, running median, and the lowess method which uses local weighted linear regression as a measure of average.

The size of the neighbourhood, J , also called the span of the smooth, essentially controls the smoothness of the result. Using $J=1$ would reproduce a smooth that fits the observations perfectly. Increasing J would result in a smooth that gives a more general representation of the regression relationship - decreasing the variance whilst increasing the bias. This is also known as the bias-variance trade-off.

Implementing the smoother of Equation (2.4) poses two problems:

1. how to select the span to use; and
2. which measure of average to use.

The section that follows contains a discussion of a local averaging smoother that addresses these problems.

2.2 The super-smoother

In the previous section it was explained that local averaging smoothers operate by calculating a measure of average for a neighbourhood around each observation in the problem domain. The ACE procedure utilises the super-smoother to determine conditional expectation estimates (Breiman & Friedman, 1985).

Friedman (1984) describes the super-smoother as a variable span smoother based on linear fits using cross-validation to determine the optimal span. This short circumscription indicates how the super-smoother addresses the two problems posed in the previous section. Sections 2.2.1 to 2.2.3 describe the main characteristics of the super-smoother. Section 2.2.4 contains a breakdown of the super-smoother algorithm implementation in MATLAB® and Section 2.2.5 illustrates the super-smoother application with a simulated example.

2.2.1 A linear fit as a measure of average

The super-smoother uses local averaging by fitting a least squares straight line through the neighbouring points of an observation, x_i . The value of the linear fit at x_i is taken to be the measure of average for the y -values in the neighbourhood, N_i . The reason for using a least squares straight line rather than a simple average is twofold. Firstly, the x -values are almost never equally spaced in practice. Using a simple average will not reproduce straight lines. Secondly, a simple average calculation will be less accurate near the boundaries of the x -domain as it is not possible to keep the span symmetric.

Using a linear fit as the measure of average, Equation (2.4) may be rewritten as:

$$s(x_i) = \beta_i x_i + \alpha_i \quad (2.5)$$

where the parameters of the fitted line are calculated using

$$\alpha_i = \bar{y}_{N_i} - \beta_i \bar{x}_{N_i} \quad (2.6)$$

and

$$\beta_i = \frac{C_{N_i}}{V_{N_i}} \quad (2.7)$$

with

$$\bar{y}_{N_i} = \frac{1}{J} \sum y_j, \quad (2.8)$$

$$\bar{x}_{N_i} = \frac{1}{J} \sum x_j, \quad (2.9)$$

$$C_{N_i} = \sum (x_j - \bar{x}_{N_i})(y_j - \bar{y}_{N_i}), \quad (2.10)$$

$$V_{N_i} = \sum (x_j - \bar{x}_{N_i})^2, \quad (2.11)$$

given $x_j \in N_i, j = 1 \dots J$.

2.2.2 The use of a variable span

In the previous section, the measure of average of the neighbourhood, N_i , was defined. The next issue to be addressed is the definition of the neighbourhood itself and more specifically, how to select the span of the neighbourhood.

A neighbourhood of size J may be defined as the selection of J observations having x -values closest to x_i . This is referred to as the nearest neighbour selection. Another method would be to select observations which would form a symmetric neighbourhood around x_i . Friedman (1984) elects to use the symmetric neighbourhood for its computational advantage while pointing out that no one of these approaches deliver better results in general.

The key feature of the super-smoother is, however, the automatic calculation of the span to use at each x -value. Usually, the analyst chooses the best span to use for the specific problem and applies it over the entire problem domain. Using a constant span over the entire domain is not optimal, especially in cases of heteroscedasticity or where the second derivative of f changes over the domain (Friedman, 1984).

The optimal span to use at each x -value (as well as the corresponding smooth value) may be obtained by selecting that span that minimises an estimate for the expected squared error:

$$e^2(s, J|x) = E_Y \left[(Y - s(x|J))^2 | x \right]. \quad (2.12)$$

Since the error at observation x_i is always minimised by $J=1$ in the case of the normal residual, that is $y_i - s(x_i|J)$, the need for employing a cross-validation method arises. A simple cross-validation method entails removing an observation and performing optimisation with the remaining set of observations (Stone, 1974). It is therefore more appropriate to use the cross-validated residual:

$$r_{(i)}(J) = y_i - s_{(i)}(x_i|J) \quad (2.13)$$

where $s_{(i)}$ indicates that observation i has been removed from the calculation. It can also be shown (Friedman, 1984) that the cross-validated residual may be calculated using the normal residual since s is a linear smoother:

$$r_{(i)}(J) = \frac{y_i - s(x_i|J)}{1 - \frac{1}{J} - \frac{(x_i - \bar{x}_{N_i})^2}{V_{N_i}}}. \quad (2.14)$$

Friedman suggests smoothing the absolute residuals themselves according to x to obtain better estimates. Taking the smoothed value as an estimate for the expected error, $\hat{e}(s, J|x)$,

will ensure that the error itself has less variance and is not based on a single observation, but rather on an average in the neighbourhood of x_i . The span value obtained by minimising $\hat{e}$ may then in turn be used to calculate the value of the smooth.

When evaluating the residuals, Friedman suggests using three to five discrete values for the span - selected from the range $[1, n]$ - to adequately represent the main parts from the frequency spectrum of $f(x)$. Since small to moderate changes in the span have little effect on the residuals, using more than five different values is usually unnecessary. To further ensure a smooth variance of the span (and therefore the smooth) over the predictor domain, the calculated span-values themselves are smoothed according to x . The result is an estimated span value constrained by the lowest and highest span values chosen at the start of the procedure. The final smooth value is calculated by interpolating the smooth values associated with the two span values closest to the estimated optimal span.

2.2.3 The need for bass enhancement

The procedure explained above is driven by minimised expected errors, which is good practice when creating regression curves, but often produces curves with high variance even when the underlying relationship is assumed to be simple. A few techniques exist that could be applied to obtain a smoother curve. One might apply a smoothing method recursively, or wish to choose larger span values at the start of the super-smoother procedure. Friedman (1984) suggests the following: increase the span value at each x by an amount which is inversely proportional to the amount of increase in the expected error resulting from the suggested span increase. In this way, high variance is reduced, while error restriction is still applied.

Let $J_{cv}(x_i)$ denote the span that minimises the smoothed absolute cross-validated residual at x_i . Let J_L be a large span value associated with the lower frequency range, also known as the bass component, of the resulting smooth. Let $0 \leq \alpha \leq 10$ be a variable indicating the extent of bass enhancement to be performed, where $\alpha = 0$ is equivalent to minimal bass enhancement and $\alpha = 10$ is equivalent to maximum bass enhancement. The revised estimated span may then be calculated using

$$J(x_i) = J_{cv}(x_i) + [J_L - J_{cv}(x_i)] R_i^{10-\alpha} \quad (2.15)$$

where

$$R_i = \frac{\hat{e}[J_{cv}(x_i)|x_i]}{\hat{e}(J_L|x_i)}. \quad (2.16)$$

Note that when α is chosen equal to zero, $J(x_i) = J_{cv}(x_i)$ and when α is chosen equal to 10, $J(x_i) = J_L$. For any specific value of $\alpha \neq 10$, a smaller value of the error ratio (R_i) will result in a smaller bass enhancement factor while a larger error ratio will result in a larger bass enhancement factor. In other words, if the increase in span would result in a large increase of the error, the increase in span is restricted, whereas a smaller increase in error would allow for larger span growth. Figure 2.1 shows the bass enhancement factor, $[J(x_i) - J_{cv}(x_i)] / [J_L - J_{cv}(x_i)]$, plotted against R_i for different values of α .

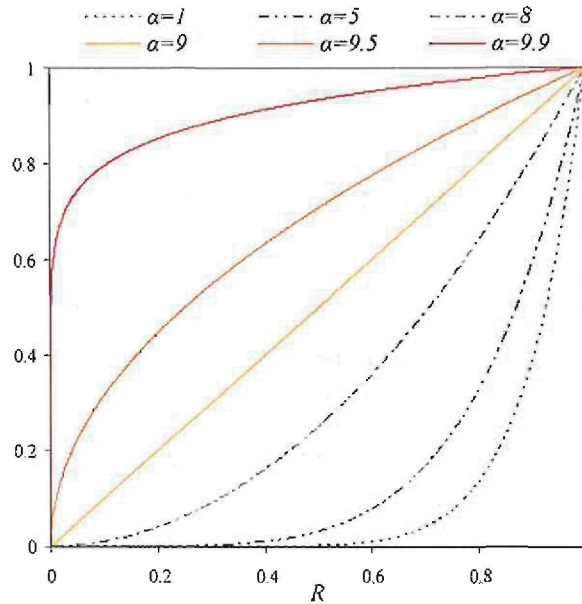


Figure 2.1: Bass enhancement factor vs. the error ratio

2.2.4 Implementation of the super-smoother

The super-smoother procedure was implemented in MATLAB® using the algorithm described below. The algorithm is based on the procedure described by Friedman (1984) in a technical report about the variable span smoother. The report contains a compact description of the passes that the smoother makes over the data, with detail of each step explained in separate sections within the report.

- (i) Receive the input parameters:
 - (a) the observations, $(x_1, y_1), \dots, (x_n, y_n)$, from which to compute the smooth;
 - (b) two or more smoother indicators on which to base the variable span, for example, $0.05n$, $0.2n$, and $0.5n$ are known as the tweeter, midrange and woofer

smoothers and are intended to reproduce the three main parts of the frequency spectrum of $f(x)$ (Friedman, 1984);

- (c) the bass enhancement indicator (bass control) – a value in the interval $[0,10]$.
- (ii) Order the observations according to x in ascending order.
- (iii) Compute the smooth and cross-validated absolute residuals corresponding to each span value represented by the smoother indicators defined in step (i)(b).
- (iv) Smooth the cross-validated absolute residuals according to x using the midrange smoother.
- (v) Compute the estimated error as the minimum smoothed absolute residual at each x -value as well as the span value associated with the estimated error.
- (vi) Apply the bass enhancement to the estimated span values using the bass enhancement indicator defined in step (i)(c).
- (vii) Smooth the bass enhanced estimated optimal span values according to x using the midrange smoother.
- (viii) Compute the final smooth value at each x -value as the average between the two smooths having associated span values closest to the estimated optimal span value obtained in step (vii).
- (ix) Arrange the data into its original order and return the resulting smooth.

2.2.5 Example application of the super-smoother

The MATLAB[®] implementation of the super-smoother described in the previous section was applied to a simulated example used by Friedman and Silverman (1989). The example was specifically chosen to illustrate the behaviour of a smoother in a case of both varying curvature and noise levels across the problem domain. In such a case an adaptable span is more desirable than a constant span, since the latter has no means of automatically adjusting for the bias-variance trade-off.

A data set was generated using

$$\begin{aligned} y_i &= \varepsilon_i, \quad x_i \leq 0; \\ y_i &= \sin\left[2\pi(1-x_i)^2\right] + \varepsilon_i, \quad 0 < x_i \leq 1 \end{aligned} \tag{2.17}$$

for $i = 1, \dots, 50$, with x_i drawn randomly from a uniform distribution in $[-0.2, 1.0]$ and ε_i drawn randomly from a normal distribution with a standard deviation of $\max(0.05, x_i)$. Figure 2.2 contains a scatter plot of the data set with results of the woofer, midrange and tweeter (constant

span) smoothers superimposed. The tweeter smooth has the most variance and the woofer smooth the least.

Figure 2.3 contains a scatter plot of the data set with the super-smoother result superimposed. Figure 2.4 illustrates the selected optimal span values as well as the midrange-smoothed span values. In the interval $x_i < 0.2$, which represents the high-curvature, low-noise region, a small span value was selected as optimal. This result is a less biased smooth. In the interval $x_i > 0.6$, which represents the low-curvature, high-noise region, a large span value was selected. This in turn results in a less variant smooth. It is clear how the super-smoother exhibits adaptability to the characteristics of the problem domain. Friedman and Silverman (1989) do, however, point out that the super-smoother has a tendency to over-fit the data in low-noise scenarios while performing poorly in very high-noise scenarios.

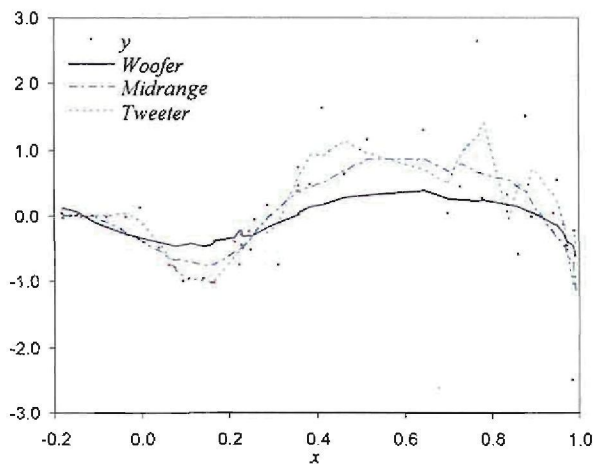


Figure 2.2: Super-smoother example – constant span smoothers

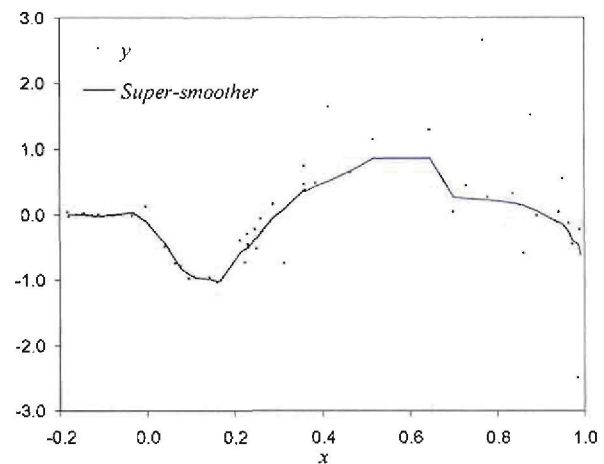


Figure 2.3: Super-smoother example – super-smoother

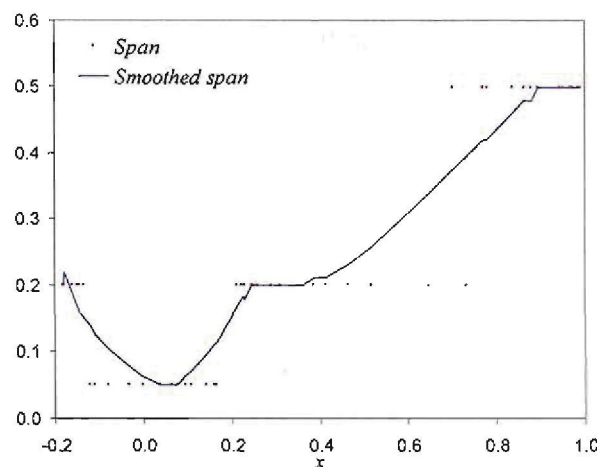


Figure 2.4: Super-smoother example – estimated optimal span values

2.3 The ACE procedure

The ACE procedure introduced by Breiman and Friedman (1985) is praised as a novel and remarkable achievement (Fowlkes & Kettenring, 1985) – a powerful tool bringing objectivity to the area of variable transformations in data analysis (Pregibon & Vardi, 1985) which is conceptually simple, mathematically elegant and useful in its ability to uncover nonlinear relationships that might otherwise be missed (Buja & Kass, 1985). The procedure was developed to estimate the optimal transformations for variables in multiple regression problems and to use the transformations to build additive models.

Consider the multivariate case of n observations of the form $(y_i, \mathbf{x}_i)$, where $\mathbf{x}_i = (x_{1i}, x_{2i}, \dots, x_{pi})$ is a vector of p predictor variable observations. When it is assumed that the response variable Y is dependant on the predictor variable $\mathbf{X} = (X_1, X_2, \dots, X_p)$ by a relation of the form

$$Y = f(\mathbf{X}) + \varepsilon, \quad (2.18)$$

the value $f(\mathbf{X})$ is considered to be the conditional expectation of Y given $\mathbf{X}$. Creating a model to estimate $f(\mathbf{X})$ would not only be useful in predicting Y in cases where only $\mathbf{X}$ is known, but would also help in gaining better insight into the predictive relationship between the variables.

While estimating $f(\mathbf{X})$ as a multivariate function could prove to be challenging, additive modelling simplifies the problem - it involves the estimation of p one-dimensional functions of the components of $\mathbf{X}$:

$$f(\mathbf{X}) = \sum_{j=1}^p f_j(X_j). \quad (2.19)$$

The ACE procedure creates a model of additive form and maps each one-dimensional function to a variable transformation. In addition, it also transforms the response variable, Y .

Section 2.3.1 describes an iterative procedure for finding optimal variable transformations using bivariate conditional expectations. Section 2.3.2 describes the procedure followed to create the ACE predictive model using variable selection and Section 2.3.3 contains a decomposition of the ACE algorithm.

2.3.1 An iterative procedure for variable transformation

Variable transformation is a method commonly used when building regression models. It exploits the relationships between variables to linearise regression models, to create normally distributed variables or error terms, to stabilise the error variance and it may also be used to

remove multicollinearities. The ACE procedure also uses variable transformations as a basis for creating an additive predictive model.

Finding transformations to achieve the desirable results may be difficult and time consuming. Breiman and Friedman (1985) provide mathematical proof that optimal variable transformations exist and that the transformations calculated by the ACE procedure converges to the optimal transformations. There are, however, restrictions in the way the conditional expectations are calculated.

ACE is a nonparametric procedure based on the iterative calculation of bivariate conditional expectations. Say Y is a response variable to the predictor variables $X_1, X_2, \dots, X_p$ and $\theta(Y), \phi_1(X_1), \phi_2(X_2), \dots, \phi_p(X_p)$ are transformation functions on the variables. The aim is to find optimal transformations $\theta^*, \phi_1^*, \phi_2^*, \dots, \phi_p^*$ that will minimise the error of a regression of $\theta(Y)$ on $\sum \phi_j(X_j)$:

$$e^2 = \frac{E \left\{ \left(\theta(Y) - \sum_{j=1}^p \phi_j(X_j) \right)^2 \right\}}{E \theta^2(Y)}. \quad (2.20)$$

Breiman and Friedman (1985) explains the procedure on the assumptions of

1. a known distribution of the problem variables;
2. $E \theta^2(Y) = 1$, and
3. the transformation functions having zero means.

Under these assumptions the bivariate case of Equation (2.20) simplifies to

$$e^2(\theta, \phi) = E [\theta(Y) - \phi(X)]^2. \quad (2.21)$$

For a given function $\phi(X)$, e^2 is minimised by

$$\theta(Y) = \frac{E[\phi(X)|Y]}{\|E[\phi(X)|Y]\|} \quad (2.22)$$

where $\|(\cdot)\| \equiv \sqrt{E(\cdot)^2}$. For a given function $\theta(Y)$, e^2 is minimised by

$$\phi(X) = E[\theta(Y)|X]. \quad (2.23)$$

The ACE algorithm starts with the initialisation of θ : $\theta_0 = Y/\|Y\|$. The two conditional expectations from Equations (2.22) and (2.23) are alternately calculated until θ and ϕ converges to θ^* and ϕ^* .

The algorithm is extended to the multivariate case with p predictor variables by replacing $\phi(X)$ in Equations (2.21) and (2.22) with $\sum \phi_j(X_j)$ and by replacing Equation (2.23) with $k = 1, \dots, p$ equations of the form

$$\phi_k(X_k) = E \left[\theta(Y) - \sum_{j \neq k} \phi_j(X_j) \mid X_k \right]. \quad (2.24)$$

The application of the iterative procedure to the multivariate case thus involves a sub-iteration over k to optimise the predictor variable transformations for a specific response variable transformation.

Some of the quantities described above need to be estimated when working with finite data sets. A data analyst usually has a set of observations which acts as a sample taken from the problem variables. In the case of continuous variables, the conditional expectations may be estimated using a smoothing method applied to the set of observations (Breiman and Friedman (1985) provide proof of convergence and consistency of the ACE algorithm for smoothing methods that are self-adjoint only, although they find that a variety of non-self-adjoint smooths also converge). For finite data sets, e^2 may be estimated using the mean squared error for regression:

$$\hat{e}^2 = \frac{1}{n} \sum_{i=1}^n \left[\theta(y_i) - \sum_{j=1}^p \phi_j(x_{ij}) \right]^2. \quad (2.25)$$

$\|(\cdot)\|$ is estimated using

$$\|(\cdot)\| = \sqrt{\frac{1}{n} \sum_{i=1}^n (\cdot)^2}. \quad (2.26)$$

Using a smoother to estimate the conditional expectations together with these estimates for e^2 and $\|(\cdot)\|$ produces transformations that are estimates of the optimal transformations.

2.3.2 Creating the additive model

Once the optimal variable transformations, $\hat{\theta}^*, \hat{\phi}_1^*, \hat{\phi}_2^*, \dots, \hat{\phi}_p^*$, have been estimated, a prediction of $\theta^*(y)$ may be obtained for a new observation where y is unknown using

$$\hat{\theta}^*(y) = \sum_{j=1}^p \hat{\phi}_j^*(x_j). \quad (2.27)$$

To obtain the corresponding y -value, one needs to calculate the inverse of the estimated optimal transformation for Y , that is $\hat{\theta}^{*-1}$. Alternatively, one could smooth Y according to

$Z = \sum \hat{\phi}_j(X_j)$ using the observation data set. This would give the best least squares predictor of Y , since it is an estimate of the expected value of Y given Z (Breiman & Friedman, 1985). The resulting smooth is referred to as the model smooth. Both the set of transformation smooths and the model smooth are used to find the response value for the new observation. This may be done by means of interpolation, since the smooths are usually represented by a set of data points. The algorithm described in the following section contains steps to explain this procedure. Apart from the predictive ability provided by the ACE model, graphic presentations of the variable transformations provide insights into the relationships between the response and predictor variables. It is also helpful by providing an indication of the predictive contribution of each input variable. Breiman and Friedman (1985) propose an iterative application of the ACE procedure in a forward selection manner to assist in variable selection when building the ACE model. First apply the ACE procedure to p bivariate problem cases, each having Y as the response variable and one component of X , X_j , as the predictor variable. Determine which case results in the smallest error, assign it to $\hat{e}_1^{*2}$, and use the corresponding predictor variable as the first of the final model variables, X_{F1} . The next iteration includes $p-1$ trivariate problem cases each having two predictor variables, X_{F1} and one other of the remaining $p-1$ variables. Again, select the variable corresponding to the case which produces the smallest error, now $\hat{e}_2^{*2}$, as the next variable, X_{F2} , to include in the final model. Continue this iterative procedure until all the variables have been added to the final model or until two consecutive errors, $\hat{e}_j^{*2}$ and $\hat{e}_{j+1}^{*2}$, do not differ by less than some predetermined value, for example, 0.01. This procedure adds variables to the final model in order of predictive significance.

2.3.3 Implementation of the ACE procedure

Breiman and Friedman (1985) point out that their implementation of the ACE procedure may be applied to either continuous or categorical variables. For simplicity reasons, the implementation presented here caters for continuous variables only. It uses the super-smoother to estimate the conditional expectations. There is also a limit to the maximum number of iterations as well as a stop-value for the change in the error value, $\hat{e}_j^{*2}$, when estimating the conditional expectations. The procedure will terminate as soon as the error cannot be reduced by more than the stop-value in a single iteration. A description of the algorithm is subsequently given.

- (i) Receive the input parameters for the procedure:
 - (a) the observations, $(x_1, y_1), \dots, (x_n, y_n)$, where x_i is a vector of p predictor variables;
 - (b) the maximum number of allowed iterations;
 - (c) the stop-value for the change in error value;
 - (d) the super-smoother span-option as described in Section 2.2.4 – 0 to use the default super-smoother span indicators (representing the tweeter, midrange and woofer spans), or a vector of user-specified smoother indicators;
 - (e) the super-smoother bass enhancement indicator.
- (ii) Initialise the response variable transformation to $Y/\|Y\|$.
- (iii) Initialise the predictor variable transformations to zero.
- (iv) Iterate until the error fails to decrease by more than the stop-value or until the maximum number of iterations have been reached:
 - (a) Iterate until the error fails to decrease by more than the stop-value:

For $k = 1, \dots, p$ compute the transformation, $\hat{\phi}_k(X_k)$, for each predictor variable. This includes smoothing $\hat{\theta}(Y) - \sum_{j \neq k} \hat{\phi}_j(X_j)$ according to X_k using the super-smoother as well as subtracting the mean from the resulting smooth so that the resulting function has a zero mean.
 - (b) Compute the transformation for the response variable. This includes smoothing $\sum \hat{\phi}_j(X_j)$ according to Y , subtracting the mean from the resulting smooth and calculating $\hat{\theta}/\|\hat{\theta}\|$.
- (v) The results include the optimal transformation estimates as well as the final error estimate, $\hat{e}^{*2} = e^2(\hat{\theta}^*, \hat{\phi}_1^*, \hat{\phi}_2^*, \dots, \hat{\phi}_p^*)$. One may calculate the coefficient of determination for a regression of $\hat{\theta}^*(Y)$ on $\sum \hat{\phi}_j^*(X_j)$ as $\hat{R}^{*2} = 1 - \hat{e}^{*2}$ which will give an indication of the linear association the procedure established between $\hat{\theta}^*(Y)$ and $\sum \hat{\phi}_j^*(X_j)$.

The transformation results may also be used to predict the response values for new observations:

- (vi) Compute the model smooth by smoothing Y according to the sum of the transformed predictor variables, $Z = \sum \hat{\phi}_j^*(X_j)$, using the midrange smoother.

- (vii) Transform the predictor variables of the new observation according to the transformation functions:
 - (a) Transform each predictor transformation, $\hat{\phi}_j^*$, to be strictly monotone over X_j .
 Since each variable transformation is a data smooth represented by a sequence of pairs, $(x_{j1}, \hat{\phi}_{j1}^*), \dots, (x_{jn}, \hat{\phi}_{jn}^*)$, rearranging the sequence so that it is increasing, will simplify the interpolation step that follows.
 - (b) Use interpolation to determine the transformed predictor variables.
- (viii) Compute the sum of the transformed predictor variables.
- (ix) Compute the prediction from the model smooth:
 - (a) Transform the model smooth to be strictly monotone over the sum of the transformed predictor variables. The same reason applies here as that for the predictor variable transformations.
 - (b) Use interpolation to determine the prediction result from the sum of the transformed predictor variables.

The algorithm automatically estimates the optimal variable transformations for multivariate prediction problems without any restrictive assumptions concerning the variable relationships. It also provides an additive model from which new predictions can be made. Scatter plots of the *estimated transformations provide graphic indications of the relationships between the problem variables*. When the ACE procedure is run on already transformed variables, the scatter plots indicate the appropriateness of the previously applied transformations – the more linear the ACE transformations, the closer the original transformations were to the optimal transformations.

The following section contains results of applying this ACE algorithm to the set of examples described in Section 1.3.

2.4 ACE examples

The ACE procedure was applied to all the example cases described in Section 1.3. The maximum number of iterations was set at 50 and the stop-value for the error was set at 0.001. The super-smoother span-option was set at 0, indicating the use of the default super-smoother span values of $0.05n$, $0.2n$ and $0.5n$. ACE was applied with a bass enhancement indicator of 9, unless indicated otherwise in the example itself.

For each example, the variable transformations are plotted together with scatter plots describing the resulting additive model, named *ACE*. The final estimated error, $\hat{e}^{*2}$, and the

corresponding value for $\hat{R}^{*2}$ is given ($\hat{R}^{*2} = 1 - \hat{e}^{*2}$). The latter is an indication of the linearity that the variable transformations establish in the additive model and should be viewed in conjunction with a scatter plot of $\hat{\theta}^*(Y)$ versus $\sum \hat{\phi}_j(X_j)$.

2.4.1 Simulated examples

First simulated example

A set of 200 data points was generated using $y = \exp(x^3 + \varepsilon)$ with x^3 and ε drawn randomly from standard normal distributions. Figure 2.5 and Figure 2.6 show the variable transformations $\hat{\phi}^*(x)$ and $\hat{\theta}^*(y)$ respectively, together with the known optimal transformations, x^3 and $\ln(y)$. The estimated transformations are very close to the known optimal transformations.

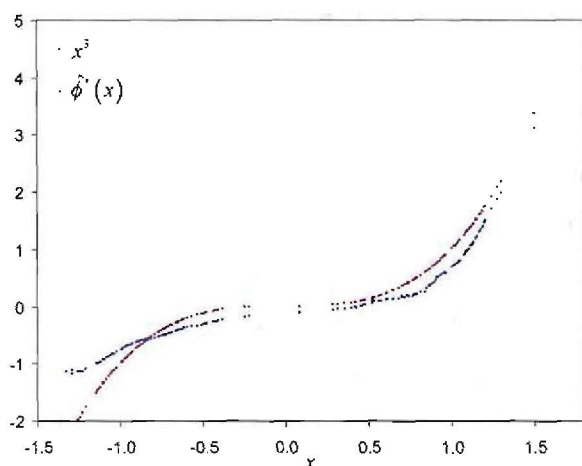


Figure 2.5: First simulated example – predictor variable transformation

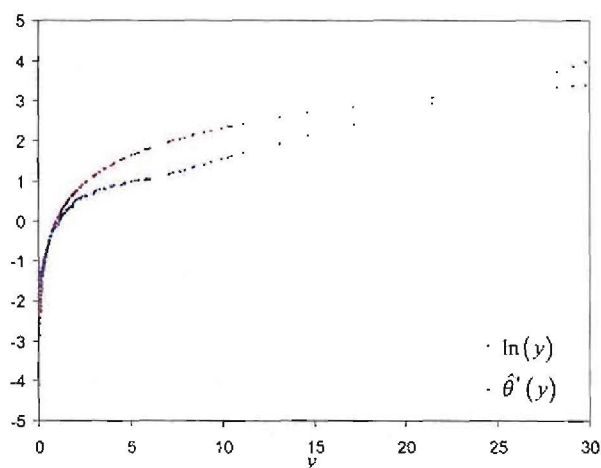


Figure 2.6: First simulated example – response variable transformation

Figure 2.7 contains a plot of $\hat{\theta}^*(y)$ versus $\hat{\phi}^*(x)$. The transformations enhanced the linear relationship between the predictor and response variables ($\hat{R}^{*2} = 0.51$) with the error minimised to $\hat{e}^{*2} = 0.49$ in 3 iterations of the ACE algorithm. Figure 2.8 contains a plot of the ACE model as a smooth of y according to $\hat{\phi}^*(x)$.

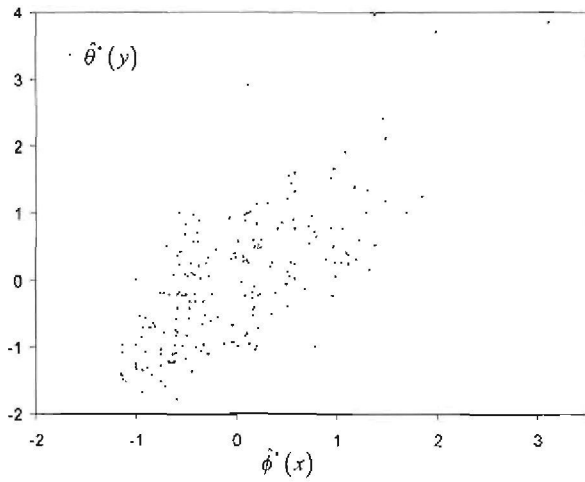


Figure 2.7: First simulated example – $\hat{\theta}^*$ vs. $\hat{\phi}^*$

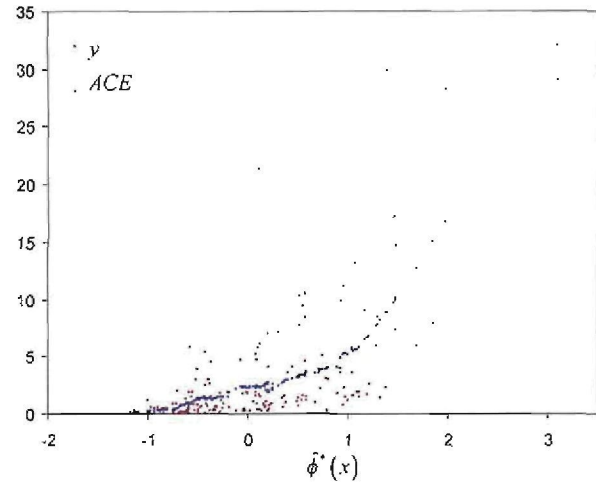


Figure 2.8: First simulated example – ACE model vs. $\hat{\phi}^*$

Figure 2.9 contains a plot of the ACE model superimposed upon the original training input data. The model has an R^2 value of 0.50 and an average square error of 10.18.

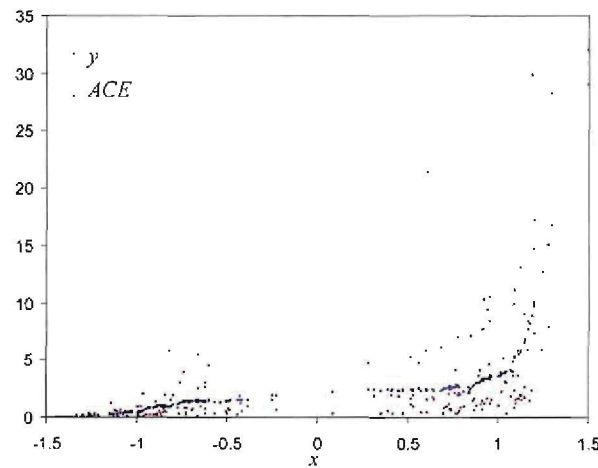


Figure 2.9: First simulated example – ACE model results

Second simulated example

A set of 200 data points was generated using $y = \exp[\sin(2\pi x) + \varepsilon/2]$ with x drawn from a uniform distribution between 0 and 1, and ε drawn independently from a standard normal distribution. Figure 2.10 and Figure 2.11 show the variable transformations $\hat{\phi}^*(x)$ and $\hat{\theta}^*(y)$

respectively, together with the known optimal transformations $\sin(2\pi x)$ and $\ln(y)$. The estimated transformations are very close to the known optimal transformations.

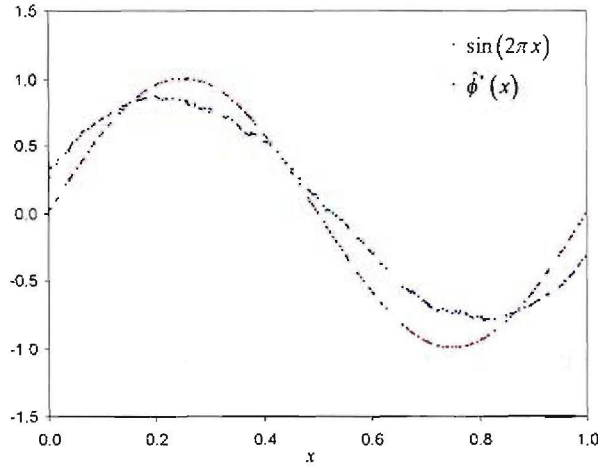


Figure 2.10: Second simulated example – predictor variable transformation

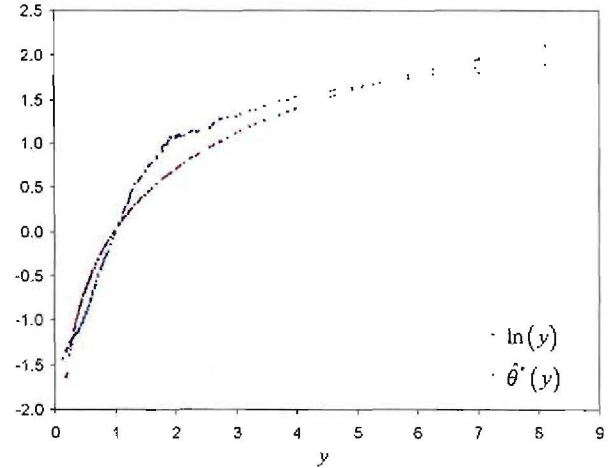


Figure 2.11: Second simulated example – response variable transformation

Figure 2.12 contains a plot of $\hat{\theta}^*(y)$ versus $\hat{\phi}^*(x)$. The transformations enhanced the linear relationship between the response and predictor variables ($\hat{R}^{*2} = 0.65$) with the error minimised to $\hat{e}^{*2} = 0.35$ in 3 iterations of the ACE algorithm. Figure 2.13 contains a plot of the ACE model as a smooth of y according to $\hat{\phi}^*(x)$.

Figure 2.14 contains a plot of the ACE model results superimposed upon the original training input data. The model has an R^2 value of 0.53 and an average square error of 0.94.

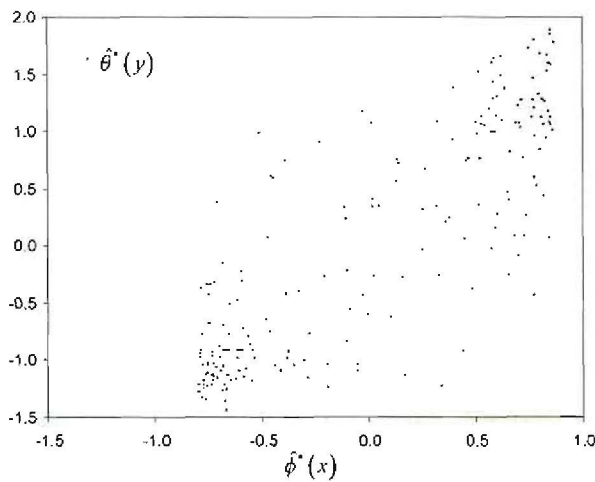


Figure 2.12: Second simulated example – $\hat{\theta}^*$ vs. $\hat{\phi}^*$

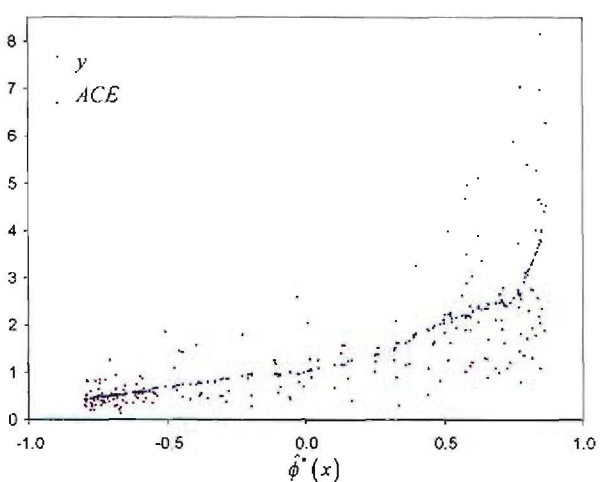


Figure 2.13: Second simulated example – ACE model vs. $\hat{\phi}^*$

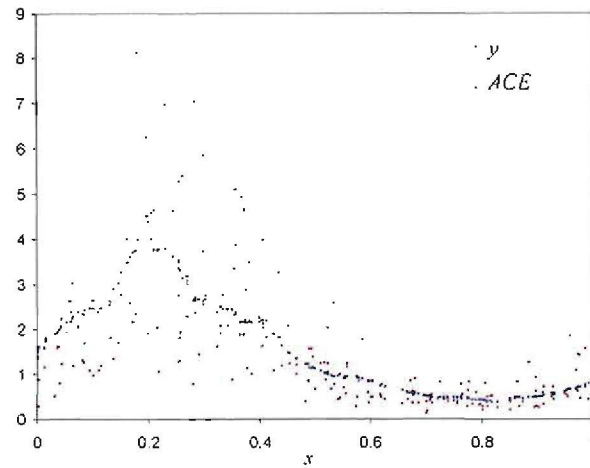


Figure 2.14: Second simulated example – ACE model results

Third simulated example

A set of 200 data points, $\{(y_i, s_i, t_i), 1 \leq i \leq 200\}$, was generated from $y = st$ where s and t are two independent variables with uniform distributions between -1 and 1. In this example a bias enhancement indicator of zero was used since there is no error term involved in the model used to generate the problem data set.

Figure 2.15, Figure 2.16 and Figure 2.17 show the variable transformations $\hat{\phi}_1^*(s)$, $\hat{\phi}_2^*(t)$ and $\hat{\theta}^*(y)$ respectively, together with the known optimal transformations $\ln|s|$, $\ln|t|$ and $\ln|y|$. The estimated transformations have the same form as the known optimal transformations.

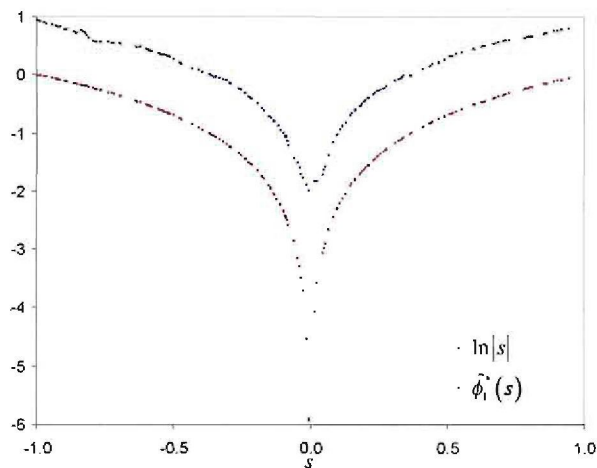


Figure 2.15: Third simulated example – predictor variable transformation for s

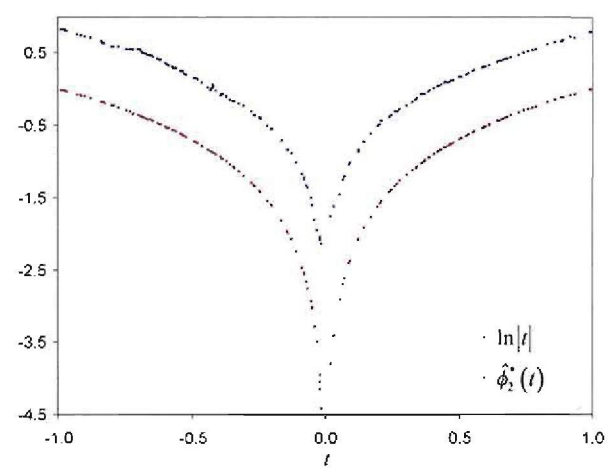


Figure 2.16: Third simulated example – predictor variable transformation for t

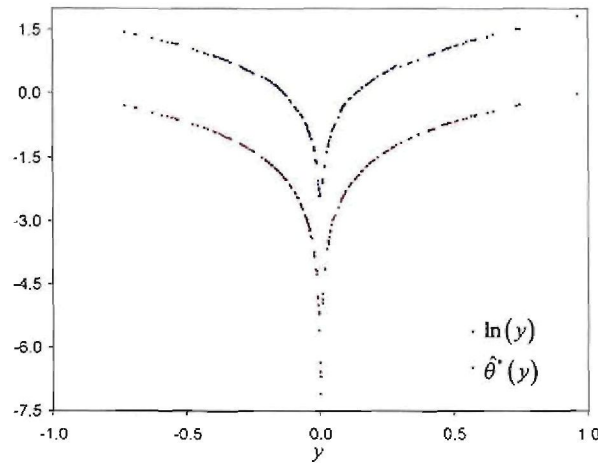


Figure 2.17: Third simulated example – response variable transformation

Figure 2.18 contains a plot of $\hat{\theta}^*(y)$ versus $\sum \hat{\phi}^*$. The transformations enhanced the linear relationship between the response variable and the sum of the predictor variables ($\hat{R}^{*2} = 0.98$) with the error minimised to $\hat{e}^{*2} = 0.02$ in 6 iterations of the ACE algorithm. Figure 2.19 contains a plot of the ACE model as a smooth of y according to $\sum \hat{\phi}^*$. From this plot it is clear the ACE procedure cannot produce a model that will reflect the interaction between the two input variables correctly, since it assumes an additive problem domain. One would also expect the value of R^2 to be low. The model produced a value of 0.05 with an average square error of 0.09. Breiman and Friedman (1985) do, however, use this example to illustrate the ability of ACE to estimate non-monotonic transformations. Included in Section 5.1.3, is a 3-dimensional graphic illustration of the model predictions on a more densely populated input space.

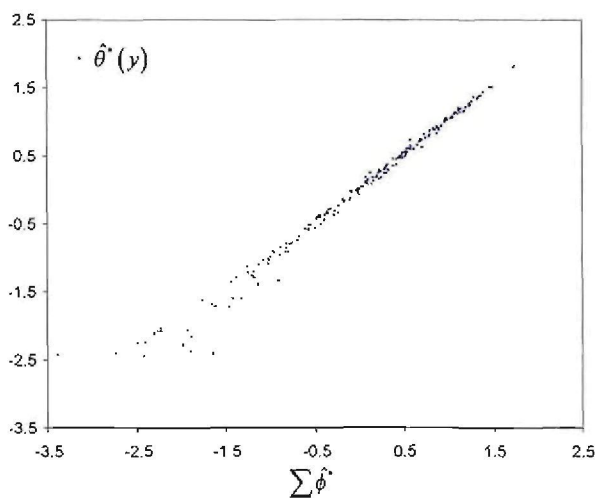


Figure 2.18: Third simulated example – $\hat{\theta}^*$ vs. $\sum \hat{\phi}^*$

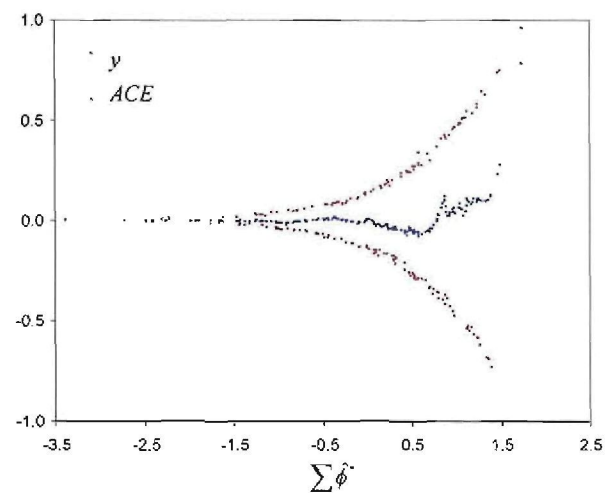


Figure 2.19: Third simulated example – ACE model vs. $\sum \hat{\phi}^*$

2.4.2 Ethanol fuel example

In this problem the predictor variable is E and the response variable is NOX . Figure 2.20 and Figure 2.21 show the variable transformations $\hat{\phi}^*(E)$ and $\hat{\theta}^*(NOX)$ respectively.

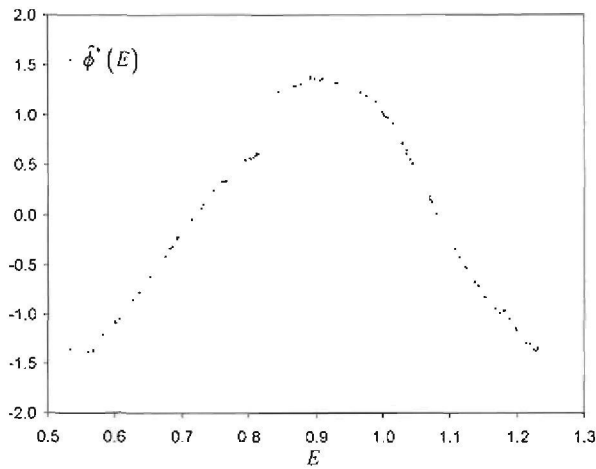


Figure 2.20: Ethanol fuel example – predictor variable transformation

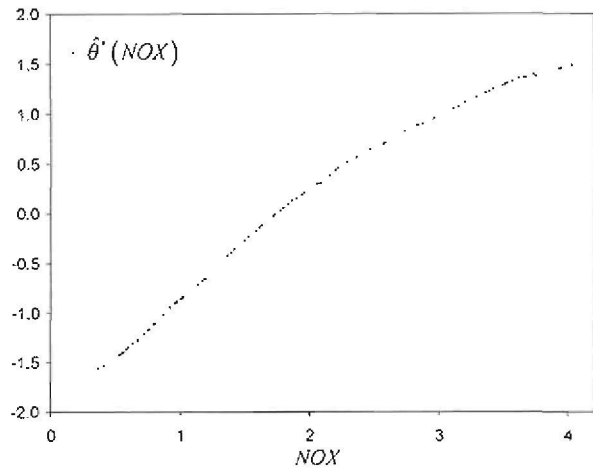


Figure 2.21: Ethanol fuel example – response variable transformation

Figure 2.22 contains a plot of $\hat{\theta}^*(NOX)$ versus $\hat{\phi}^*(E)$. The transformations enhanced the linear relationship between the predictor and response variables ($\hat{R}^{*2} = 0.91$) with the error minimised to $\hat{\epsilon}^{*2} = 0.09$ in 4 iterations of the ACE algorithm. Figure 2.23 contains a plot of the ACE model as a smooth of NOX according to $\hat{\phi}^*(E)$.

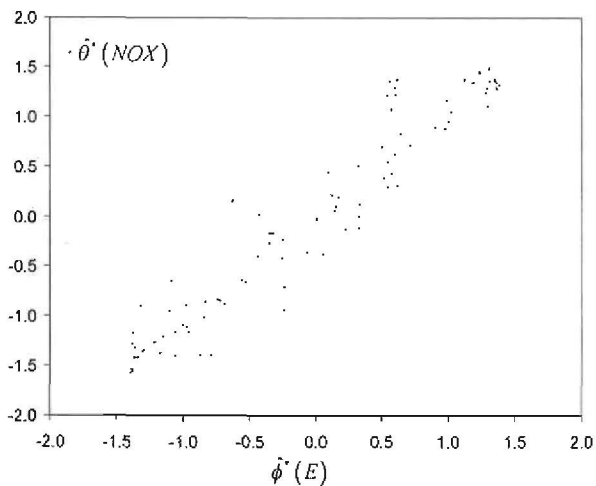


Figure 2.22: Ethanol fuel example – $\hat{\theta}^*$ vs. $\hat{\phi}^*$

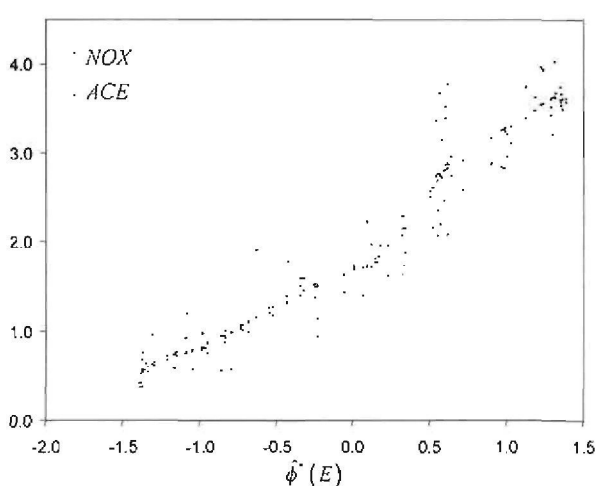


Figure 2.23: Ethanol fuel example – ACE model vs. $\hat{\phi}^*$

Figure 2.24 contains a plot of the ACE model results superimposed upon the original training input data. The model has an R^2 value of 0.91 and an average square error of 0.11.

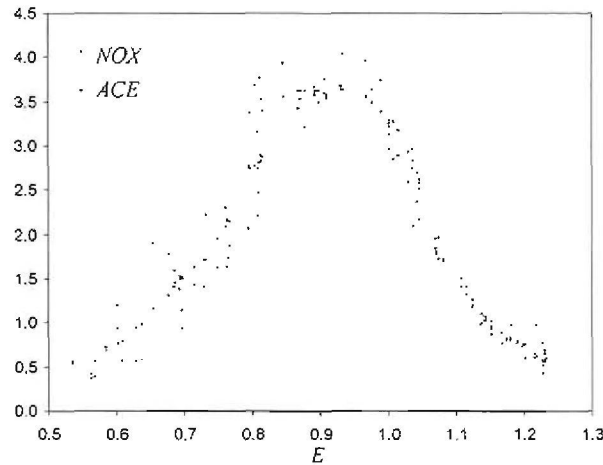


Figure 2.24: Ethanol fuel example – ACE model results

2.4.3 EXPCAR example

The predictor variable is x and the response variable is y . Figure 2.25 and Figure 2.26 show the variable transformations $\hat{\phi}^*(x)$ and $\hat{\theta}^*(y)$ respectively. The shape of the predictor variable transformation might suggest a logarithmic transformation.

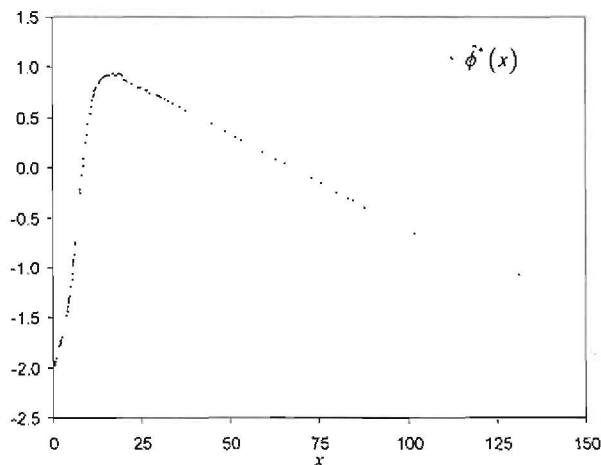


Figure 2.25: EXPCAR example – predictor variable transformation

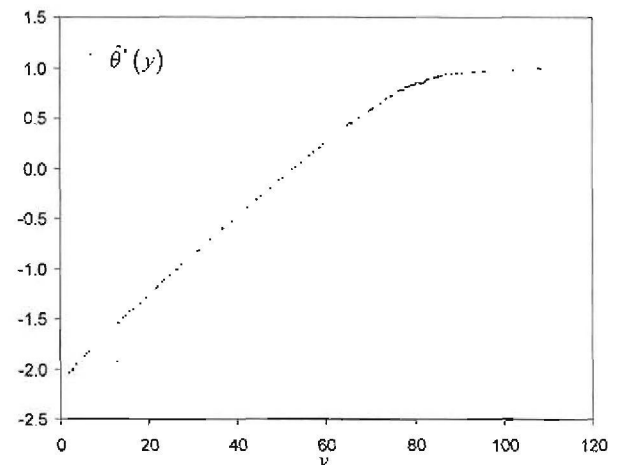


Figure 2.26: EXPCAR example – response variable transformation

Figure 2.27 contains a plot of $\hat{\theta}^*(y)$ versus $\hat{\phi}^*(x)$. The transformations enhanced the linear relationship between the predictor and response variables ($\hat{R}^{*2} = 0.98$) with the error minimised

to $\hat{e}^{*2} = 0.02$ in 2 iterations of the ACE algorithm. Figure 2.28 contains a plot of the ACE model as a smooth of y according to $\hat{\phi}^*(x)$.

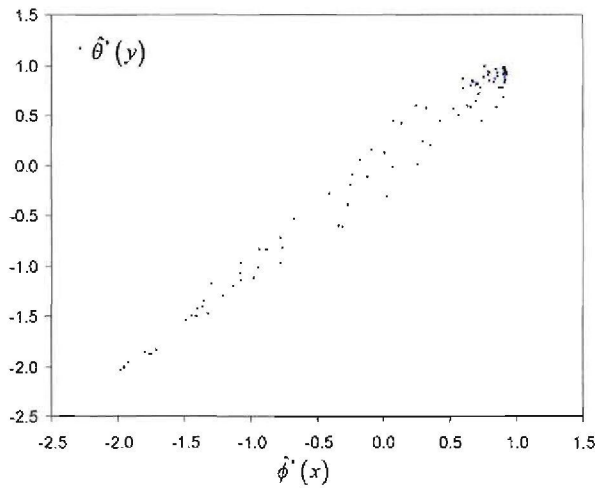


Figure 2.27: EXPCAR example – $\hat{\theta}^*$ vs. $\hat{\phi}^*$

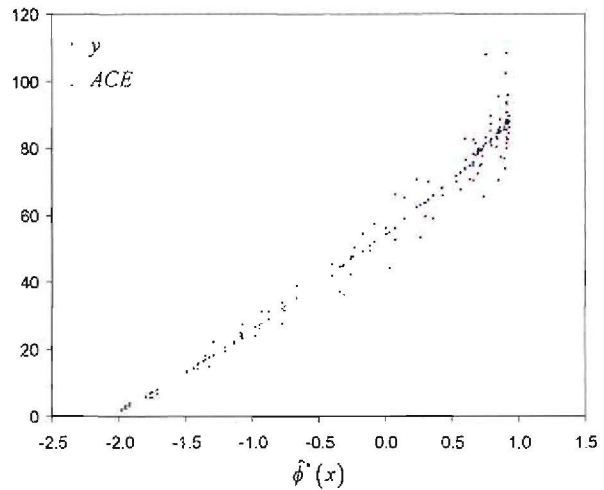


Figure 2.28: EXPCAR example – ACE model vs. $\hat{\phi}^*$

Figure 2.29 contains a plot of the ACE model results superimposed upon the original training input data. The model has an R^2 value of 0.96 and an average square error of 38.32.

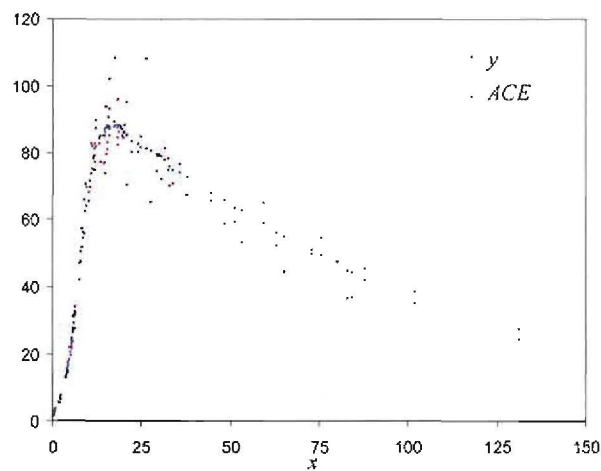


Figure 2.29: EXPCAR example – ACE model results

2.4.4 SO_4 example

The SO_4 example has predictor variables LAT and LON with response variable SO_4 . Figure 2.30, Figure 2.31 and Figure 2.32 show the variable transformations $\hat{\phi}_1^*(LAT)$, $\hat{\phi}_2^*(LON)$ and $\hat{\theta}^*(SO_4)$ respectively. The transformation graphs have sinusoidal forms.

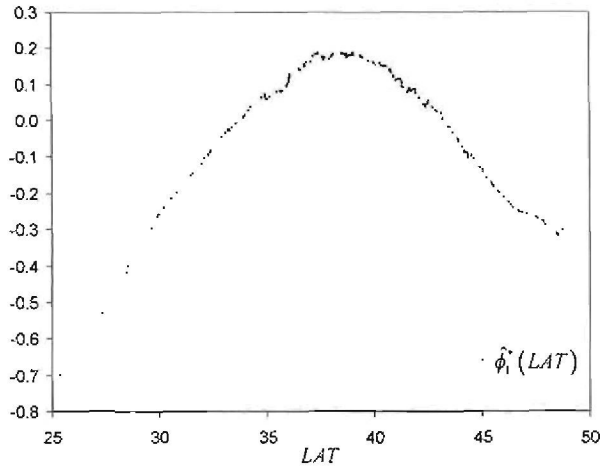


Figure 2.30: SO_4 example – predictor variable transformation for LAT

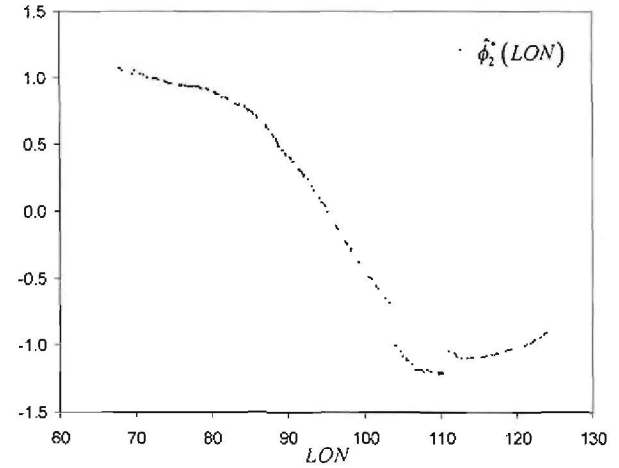


Figure 2.31: SO_4 example – predictor variable transformation for LON

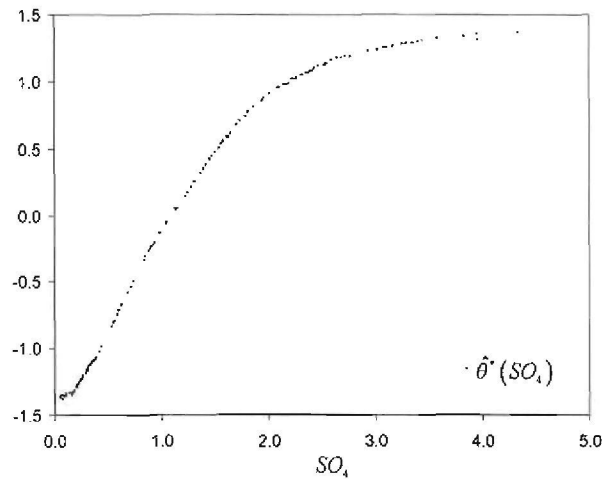


Figure 2.32: SO_4 example – response variable transformation

Figure 2.33 contains a plot of $\hat{\theta}^*(SO_4)$ versus $\sum \hat{\phi}^*$. The transformations enhanced the linear relationship between the sum of the predictor variables and response variable ($\hat{R}^{*2} = 0.85$) with the error minimised to $\hat{e}^{*2} = 0.15$ in 4 iterations of the ACE algorithm. Figure 2.34 contains a plot of the ACE model as a smooth of SO_4 according to $\sum \hat{\phi}^*$.

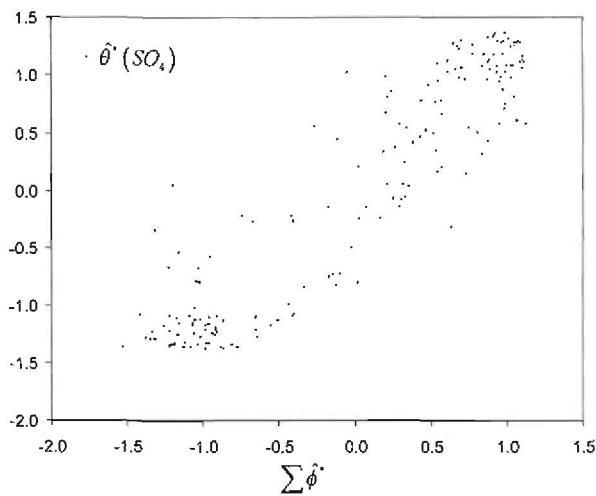


Figure 2.33: SO_4 example – $\hat{\theta}^*$ vs. $\sum \hat{\phi}^*$

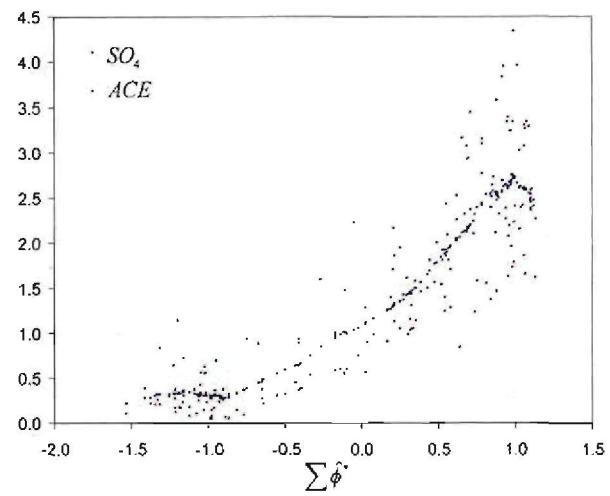


Figure 2.34: SO_4 example – ACE model vs. $\sum \hat{\phi}^*$

The model has an R^2 value of 0.79 and an average square error of 0.24. Included in Section 5.4, is a 3-dimensional graphic illustration of the model predictions on a more densely populated input space.

2.4.5 Boston Housing example

The Boston Housing data set has a large number of predictor variables, RM' , AGE , B' , $LSTAT'$, $CRIM$, ZN , $INDUS$, TAX , $PTRAT$, $CHAS$, DIS' , RAD' and NOX' . The response variable is $MEDV'$. The application of ACE involving all 13 predictor variables and with the bass enhancement indicator set at 9, produced an $\hat{R}^{*2}$ value of 0.86 with the error minimised to $\hat{e}^{*2} = 0.14$ in 5 iterations of the ACE algorithm (note that $CHAS$ is treated as a continuous variable in this implementation of ACE, although it assumes binary values). Applying ACE with no bass enhancement results in an $\hat{R}^{*2}$ value of 0.91, which is similar to the value reported by Breiman and Friedman (1985).

Including more predictor variables in a predictive model does not necessarily result in a better model. Some predictors may have little or no effect on the response variable. Others may be highly correlated which may cause model instability. Selecting a small subset of significant predictors in problems such as this one, usually results in a model with adequate predictive capability while gaining performance as well as understandability of the effects.

To illustrate the forward stepwise application of ACE for variable selection, the ACE procedure was first applied to 13 bivariate problems, each including $MEDV'$ and one of the predictor variables. Table 2.1 is a list of the predictor variables associated with each problem

together with the values for $\hat{e}^{*2}$ and $\hat{R}^{*2} = 1 - \hat{e}^{*2}$. The first iteration identified $LSTAT'$ to be the first variable to be included in the model. It produced an error of 0.28 and an $\hat{R}^{*2}$ value of 0.72. Next, ACE was applied to 12 trivariate problems including $LSTAT'$ and one of the remaining predictor variables. Table 2.2 lists the results, indicating that RM' was the next variable to be included in the model. The decrease in error associated with adding RM' to the model was 0.06.

This forward stepwise procedure was continued with a third and fourth iteration. Table 2.3 shows the result of the third iteration which included the variables $LSTAT'$, RM' and each one of the remaining variables listed in the first column of the table. The variables $PTRAT$ and $CHAS$ both produced the smallest error in the third iteration – that of 0.20 which is a decrease in the error of 0.02. Either one of these variables may be chosen as the variable to be included in the next iteration. $PTRAT$ was chosen since it has a higher correlation with $MEDV'$ than $CHAS$.

Table 2.4 shows the results of the fourth iteration which resulted in DIS' being selected as the next variable to be included in the final model. The decrease in error associated with adding DIS' to the set of predictors identified in the previous iteration is 0.02, which is still more than the stopping value for the procedure. The smallest error in the fifth iteration was produced by the variable NOX' , but the decrease in error for that iteration was less than 0.01, indicating that the variable selection procedure should be stopped and the final model should include the variables $LSTAT'$, RM' , $PTRAT$ and DIS' .

First variable to select	$\hat{e}^{*2}$	$\hat{R}^{*2}$
$LSTAT'$	0.280	0.720
RM'	0.339	0.661
ZN	0.487	0.513
NOX'	0.566	0.434
TAX	0.570	0.431
$INDUS$	0.579	0.421
$PTRAT$	0.594	0.406
$CHAS$	0.603	0.397
RAD'	0.604	0.396
$CRIM$	0.610	0.390
DIS'	0.620	0.380
AGE	0.658	0.342
B'	0.741	0.259

Table 2.1: ACE stepwise variable selection – results for the first iteration

Second variable to select	$\hat{e}^{*2}$	$\hat{R}^{*2}$
RM'	0.220	0.780
$PTRAT$	0.239	0.761
$CHAS$	0.246	0.754
$CRIM$	0.250	0.750
DIS'	0.262	0.738
B'	0.262	0.738
TAX	0.262	0.738
RAD'	0.269	0.731
NOX'	0.273	0.727
ZN	0.273	0.727
AGE	0.275	0.725
$INDUS$	0.276	0.724

Table 2.2: ACE stepwise variable selection – results for the second iteration

Third variable to select	$\hat{e}^{*2}$	$\hat{R}^{*2}$
<i>PTRAT</i>	0.200	0.800
<i>CHAS</i>	0.200	0.800
<i>DIS'</i>	0.201	0.799
<i>CRIM</i>	0.207	0.793
<i>TAX</i>	0.208	0.792
<i>B'</i>	0.211	0.790
<i>RAD'</i>	0.215	0.785
<i>ZN</i>	0.216	0.784
<i>AGE</i>	0.217	0.783
<i>NOX'</i>	0.219	0.781
<i>INDUS</i>	0.219	0.781

Table 2.3: ACE stepwise variable selection – results for the third iteration

Fourth variable to select	$\hat{e}^{*2}$	$\hat{R}^{*2}$
<i>DIS'</i>	0.1798	0.820
<i>CHAS</i>	0.1893	0.811
<i>CRIM</i>	0.1896	0.810
<i>TAX</i>	0.1904	0.810
<i>NOX'</i>	0.1921	0.808
<i>B'</i>	0.1934	0.807
<i>RAD'</i>	0.1973	0.803
<i>ZN</i>	0.1977	0.802
<i>INDUS</i>	0.1985	0.802
<i>AGE</i>	0.199	0.801

Table 2.4: ACE stepwise variable selection – results for the fourth iteration

The final model proposed by Breiman and Friedman (1985) involves the four variables *LSTAT'*, *RM'*, *PTRAT* and *TAX* instead of *DIS'*. This may be due to factors including slight differences in the original data set, the order of the variables when running the ACE procedure (Breiman & Friedman, 1985b), running the ACE procedure with different indicator values for the procedure as well as slight differences in the implementation itself. As mentioned before, the implementation used here is slightly simplified and does not include special handling of categorical variables or weighting factors for observations.

Furthermore, the ACE procedure was also applied to the Boston Housing data with the four predictor variables used by Breiman and Friedman (1985). The bass enhancement indicator was set at 9. Figure 2.35 to Figure 2.39 show the variable transformations for *LSTAT'*, *RM'*, *PTRAT*, *TAX* and *MEDV'*.

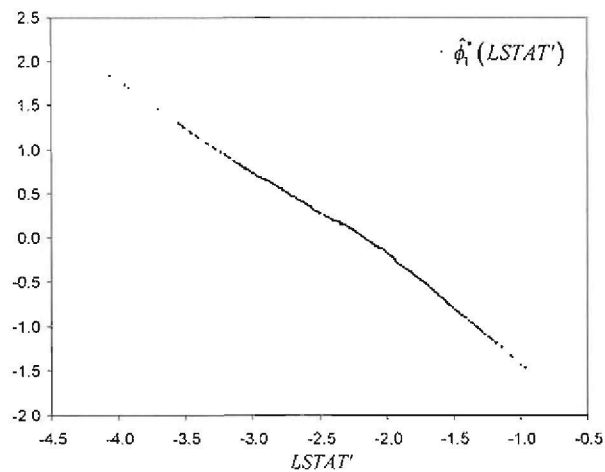


Figure 2.35: Boston Housing example – predictor variable transformation for $LSTAT'$

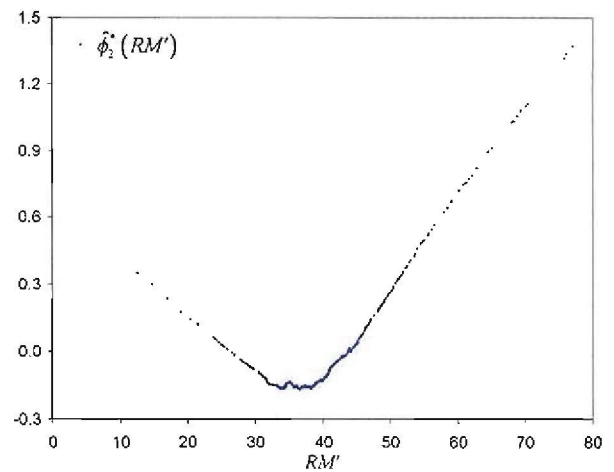


Figure 2.36: Boston Housing example – predictor variable transformation for RM'

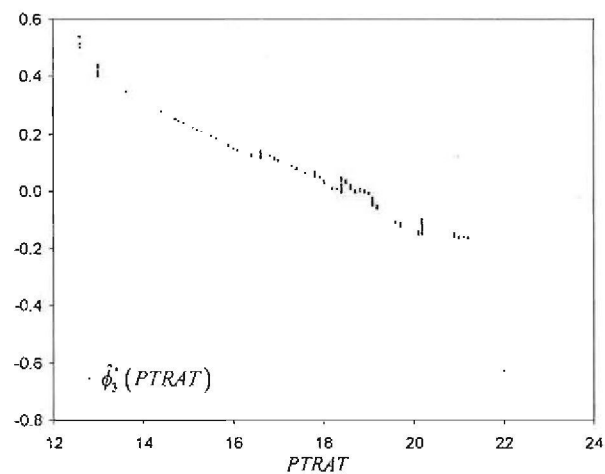


Figure 2.37: Boston Housing example – predictor variable transformation for $PTRAT'$

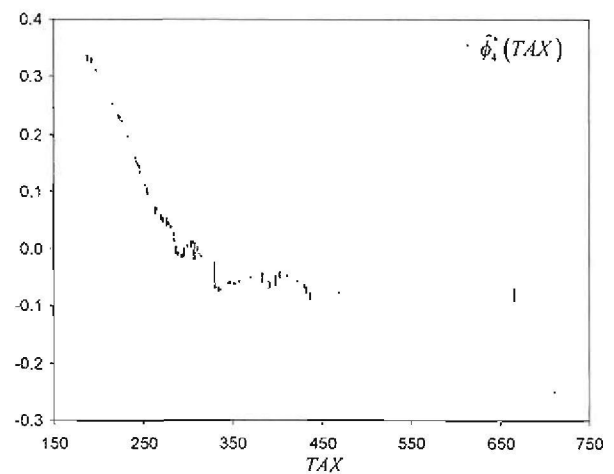


Figure 2.38: Boston Housing example – predictor variable transformation for TAX'

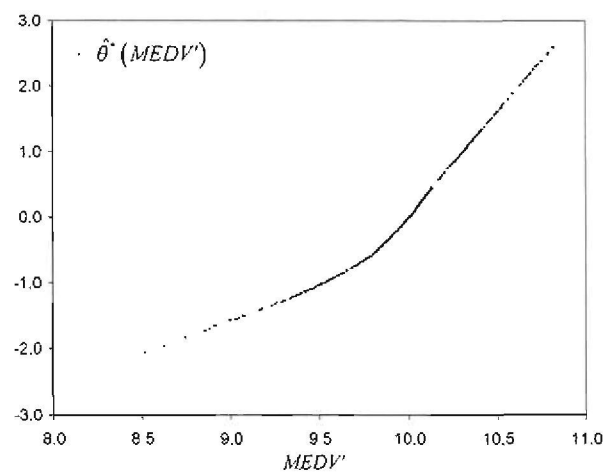


Figure 2.39: Boston Housing example – response variable transformation

Figure 2.40 contains a plot of $\hat{\theta}^*(MEDV')$ versus $\sum \hat{\phi}^*$. The transformations enhanced the linear relationship between $MEDV'$ and the sum of the transformed predictor variables ($\hat{R}^{*2} = 0.81$) with the error minimised to $\hat{\epsilon}^{*2} = 0.19$ in 4 iterations of the ACE algorithm. Figure 2.41 contains a plot of the ACE model as a smooth of $MEDV'$ according to $\sum \hat{\phi}^*$. The model has an R^2 value of 0.78 and an average square error of 0.04.

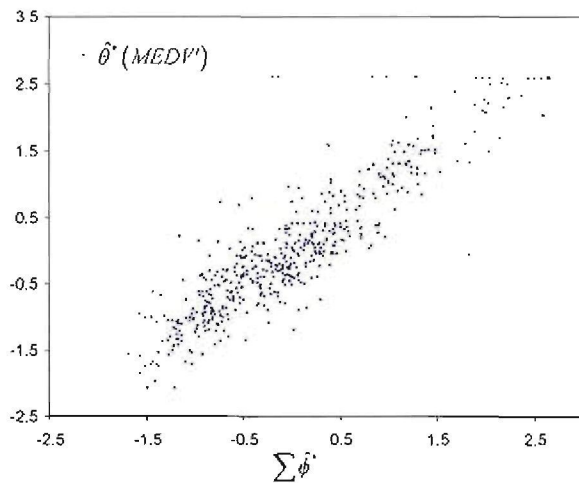


Figure 2.40: Boston Housing example – $\hat{\theta}^*$ vs. $\sum \hat{\phi}^*$

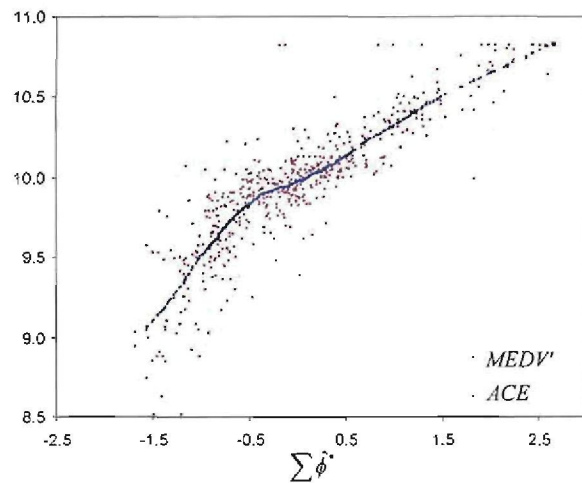


Figure 2.41: Boston Housing example – ACE model vs. $\sum \hat{\phi}^*$

2.4.6 Ozone example

The Ozone dataset consists of 9 predictor variables, $SBTP$, $IBHT$, $DGPG$, $VSTY$, $VDHT$, $HMDT$, $IBTP$, $WDSP$ and DOY and the response variable is $UPO3$. The same forward stepwise variable selection method was applied to the Ozone data set with all predictor variables except the day of year, DOY , which was used by Breiman and Friedman to point out other factors. The variables selected were:

- iteration 1 – $SBTP$, with an error of 0.3;
- iteration 2 – $IBHT$, with an error of 0.27;
- iteration 3 – $DGPG$, with an error of 0.24;
- iteration 4 – $VSTY$, with an error of 0.23.

On the fifth iteration, $WDSP$ was selected with an error of 0.23. Since adding this variable to the model would not decrease the error, the final model includes the variables selected in the first four iterations only.

Figure 2.42 to Figure 2.46 show the transformations the ACE procedure produced for the variables $SBTP$, $IBHT$, $DGPG$, $VSTY$ and the response variable, $UPO3$.

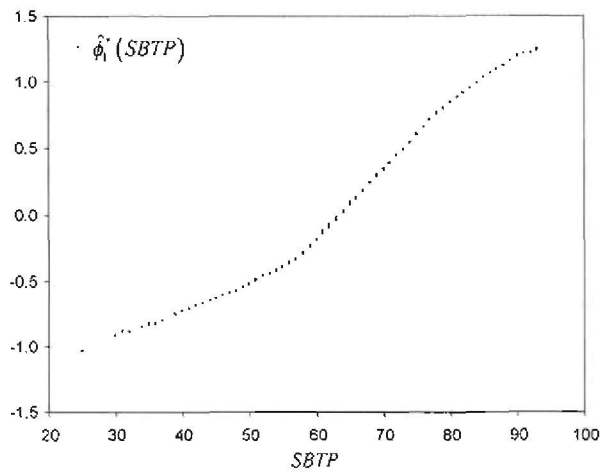


Figure 2.42: Ozone example – predictor variable transformation for *SBTP*

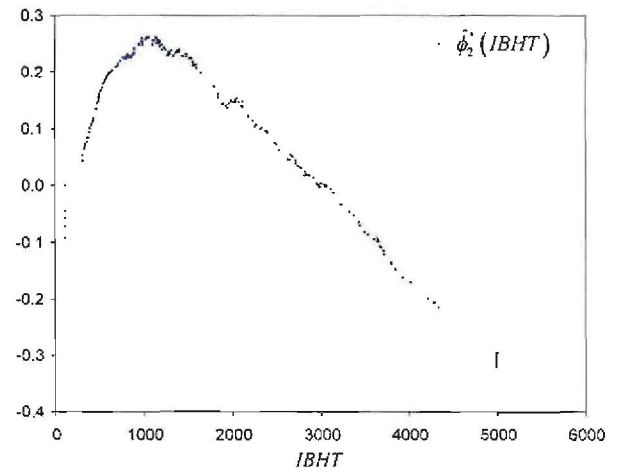


Figure 2.43: Ozone example – predictor variable transformation for *IBHT*

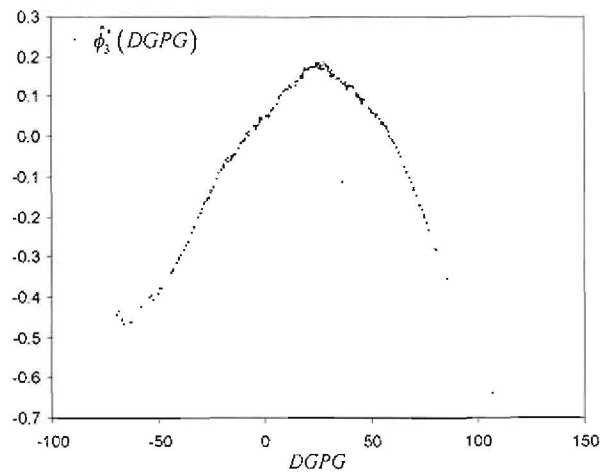


Figure 2.44: Ozone example – predictor variable transformation for *DGPG*

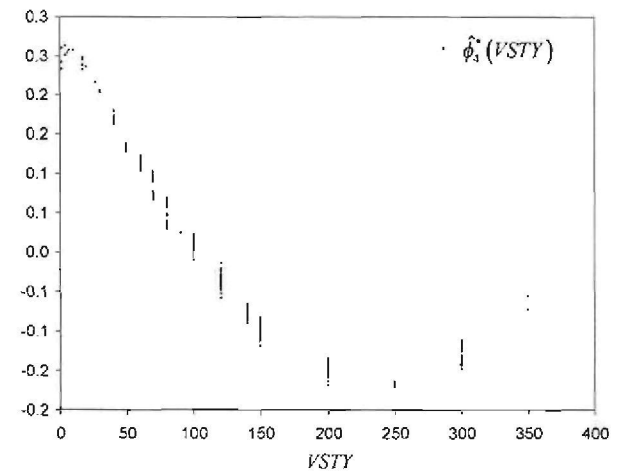


Figure 2.45: Ozone example – predictor variable transformation for *VSTY*

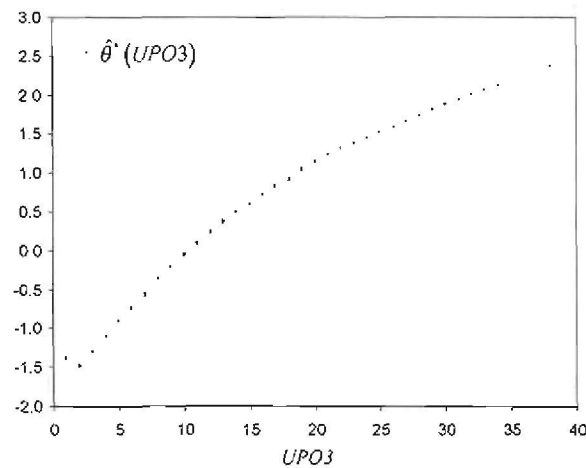


Figure 2.46: Ozone example – response variable transformation

Figure 2.47 contains a plot of $\hat{\theta}^*(UPO3)$ versus $\sum \hat{\phi}^*$. The transformations enhanced the linear relationship between $UPO3$ and the sum of the transformed predictor variables ($\hat{R}^2 = 0.77$) with the error minimised to $\hat{\epsilon}^2 = 0.23$ in 3 iterations of the ACE algorithm. Figure 2.48 contains a plot of the ACE model as a smooth of $UPO3$ according to $\sum \hat{\phi}^*$. The model has an R^2 value of 0.75 and an average square error of 15.69.

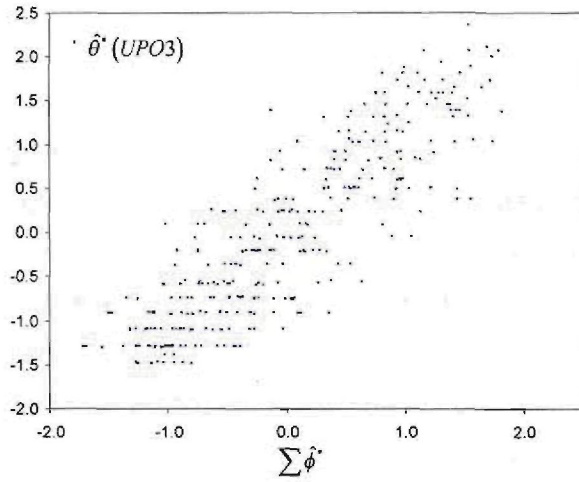


Figure 2.47: Ozone example – $\hat{\theta}^*$ vs. $\sum \hat{\phi}^*$

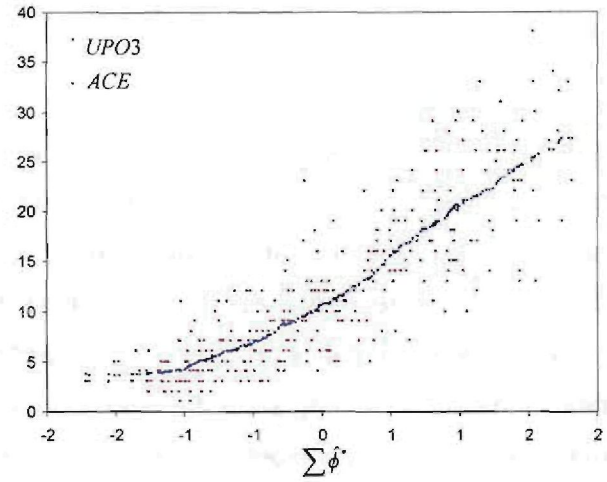


Figure 2.48: Ozone example – ACE model vs. $\sum \hat{\phi}^*$

2.5 Conclusions

In this chapter, the method of alternating conditional expectations was discussed and demonstrated using several regression problem examples. The method is helpful in determining optimal variable transformations. It was also shown how an ACE predictive model may be implemented and used.

In the following chapter, a method for creating automated generalised additive neural networks will be discussed and demonstrated using the regression problem examples.

3

AUTOMATED GENERALISED ADDITIVE NEURAL NETWORKS

Neural network modelling is known to be a simple, flexible and powerful nonlinear modelling method that may be applied effectively to very large data sets (Kutner *et al.*, 2005). Although neural network models are highly flexible, some of the problems commonly associated with these models are interpretability as well as the selection of the neural network architecture (model selection). Since neural network architectures can be extremely complex due to their flexible nature, selecting a good model is not a trivial task and automation of the model selection method becomes a necessity. In addition to simplifying the model selection method, the restriction to additive models also aids model interpretability.

This chapter is devoted to the explanation of an automated approach to the construction of generalised additive neural networks. In Section 3.1, generalised additive models (GAMs) are discussed and Section 3.2 explains the neural network implementation of GAMs, called GANNs. Section 3.3 contains a brief description of partial residual plots used in an iterative GANN construction procedure described in Section 3.4. An automated procedure for selecting a GANN model architecture, implemented as the AutoGANN model node in SAS® Enterprise Miner™, is discussed in Section 3.5. In Section 3.6, results are given of applying the AutoGANN model node to the set of examples used throughout this study.

3.1 Generalised additive models

A brief description of additive modelling was given in Section 2.3 - it involves the estimation of p one-dimensional functions of the components of X :

$$f(X) = \beta_0 + \sum_{j=1}^p f_j(X_j) \quad (3.1)$$

where $f(X)$ is considered to be the conditional expectation of Y given X .

The generalised linear model (GLM) were introduced by Nelder and Wedderburn (1972) and involves the addition of a link function, g , to the linear model. The link function relates $f(X)$ to the expected value of the response variable, $\mu = E\{Y\}$:

$$g(\mu) = \beta_0 + \sum_{j=1}^p \beta_j X_j. \quad (3.2)$$

The link function is usually assumed to be known - in the case of a linear logistic model, for example, $g(\mu) = \ln\left(\frac{\mu}{1-\mu}\right)$, and the response value is bound by zero and one.

A generalised additive model (GAM) combines the generalised linear and additive models by transforming the generalised linear model to a generalised sum of (potentially) non-linear functions:

$$g(\mu) = \beta_0 + \sum_{j=1}^p f_j(X_j). \quad (3.3)$$

Hastie and Tibshirani (1986) introduced the generalised additive model where each function f_j is estimated with a scatterplot smoother, s_j , so that the model becomes a (nonparametric) sum of smooths. To estimate the functions, they propose an iterative smoother procedure called the local scoring algorithm. It involves the repeated fit of an additive model - using the backfitting algorithm - to the adjusted dependent variable, Z , where $Z = \hat{\eta} + (Y - \hat{\mu})\left(\frac{d\eta}{d\mu}\right)$ and $\hat{\eta}$ is a current estimate of $g(\mu)$.

This generalised additive model from Equation (3.3) is similar to the ACE additive model from Equation (2.27), except that ACE generalises the additive model with a variable transformation on the response variable which also absorbs the bias term, β_0 . As with the ACE procedure, the smoother functions are normalised to have zero means to avoid the need for free constants in each of the functions.

3.2 Generalised additive neural networks

A generalised additive neural network (GANN) is the artificial neural network implementation of a generalised additive model. In order to understand the GANN implementation, it is necessary to look at the statistical representation of neural networks and relate the models and associated terms between GAMs and GANNs.

The most commonly used artificial neural networks, multilayer perceptrons, are considered to be universal function approximators (Potts, 1999) due to their flexibility in representing simple to

very complex nonlinear relationships. First consider the simple perceptron that computes a linear combination of inputs (or predictors), adds an intercept (or bias term) and applies a (potentially) nonlinear activation function:

$$E\{Y\} = \text{act}_0 \left(w_0 + \sum_{j=1}^p w_j X_j \right). \quad (3.4)$$

The (regression) coefficients of the neural network model are referred to as weights and are usually represented by w . The process of estimating the (weight) parameters for a neural network is referred to as the training or learning process. As can be seen from Equation (3.2) and Equation (3.4), a simple perceptron is analogous to a generalised linear model where the activation function of a perceptron relates to the inverse of the link function for generalised linear models. In neural network literature the activation function itself is often indicated by g so that Equation (3.4) could be rewritten as

$$g_0^{-1}(E\{Y\}) = w_0 + \sum_{j=1}^p w_j X_j \quad (3.5)$$

where g_0^{-1} is the inverse of the activation function, g_0 , and is referred to as the link function in accordance with Equation (3.2). According to Sarle (1994), a few common activation functions used in perceptrons are:

- the linear or identity function: $g_0(x) = x$;
- the hyperbolic tangent function: $g_0(x) = \tanh(x)$;
- the logistic function: $g_0(x) = (1 + e^{-x})^{-1} = \frac{\tanh\left(\frac{x}{2}\right) + 1}{2}$;
- the threshold function: $g_0(x) = 0$ if $x < 0$, else 1, and
- the Gaussian function: $g_0(x) = e^{-(x^2/2)}$

where x is referred to as the combination function. In addition to the identity function, sigmoid functions such as the hyperbolic tangent and logistic functions are the most popular because they are bounded, monotonically increasing and differentiable (Zhang, 1998). Potts (2000) points out that the link function is used to constrain the predicted response to reasonable values and that the distribution of the response variable should guide the choice of the activation function. For example, the linear link function could be used for response variables in the interval $(-\infty, \infty)$; the logit function would constrain the response to $(0,1)$ and would be suitable

for binary response values; the inverse hyperbolic tangent function would constrain the response to $(-1,1)$ and using a logarithmic function, would constrain it to $(0,\infty)$, which is suitable for nonnegative response values.

Multilayer perceptrons (MLPs) establish nonlinearity by adding one or more derived input sets, called hidden layers. Each hidden layer has its own number of derived inputs or hidden nodes, so that $g_0^{-1}(E\{Y\})$ becomes a linear combination of nonlinear functions of linear combinations of the inputs (Potts, 1999). An MLP with one hidden layer and h hidden nodes may be written as

$$g_0^{-1}(E\{Y\}) = w_0 + \sum_{k=1}^h w_k H_k \quad (3.6)$$

with

$$H_k = g_k \left(w_{k0} + \sum_{j=1}^p w_{kj} X_j \right). \quad (3.7)$$

A skip layer is (commonly) added for each input to represent the main effects of the predictors:

$$g_0^{-1}(E\{Y\}) = w_0 + \sum_{k=1}^h w_k H_k + \sum_{j=1}^p w_{0j} X_j. \quad (3.8)$$

The nonlinearity is provided by the second term involving the hidden units. When removed, the equation simplifies to that of (3.5).

Multilayer perceptrons are usually trained using the back propagation chain rule (also known as backfitting, or the method of steepest descent) and is known to be very slow. Training involves the estimation of the model parameters so that it minimises some pre-defined error function. Apart from the commonly used least squares error function, Potts (2000) also discuss several other error functions as they are appropriate when using different activation (or link) functions. These include the normal, Poisson, Gamma and Bernoulli error functions, all based on maximising the likelihood (the joint probability of the observed data) or, equivalently, minimising the deviance which has numerical advantages (Potts, 2000). The definitions of these error functions are given below. The normal least squares error function is suitable for interval response values. For binary (or ordinal) response values, the multiple Bernoulli error function is appropriate. For nonnegative response values the Poisson or Gamma error functions are useful and should be used together with the log link function to constrain the response to positive values.

- Normal: $\sum (Y - \mu)^2$;
- Poisson: $2 \sum \left[Y \ln \left(\frac{Y}{\mu} \right) - (Y - \mu) \right]$;
- Gamma: $2 \sum \left[-\ln \left(\frac{Y}{\mu} \right) + \frac{Y - \mu}{\mu} \right]$;
- Bernoulli: $-2 \sum [Y \ln(\mu) + (1 - Y) \ln(1 - \mu)]$.

A generalised additive neural network implements the generalised additive model by placing restrictions on the form of the flexible multilayer perceptron. According to Potts (1999), the basic architecture of the GANN consists of a separate MLP with a single hidden layer of h units for each input variable and uses the hyperbolic tangent function as activation function. A GANN model has the form:

$$g_0^{-1}(E\{Y\}) = w_0 + \sum_{j=1}^p f_j(X_j) \quad (3.9)$$

where

$$f_j(X_j) = w_{0j}X_j + \sum_{k=1}^h \left[w_{kj} \tanh(w_{0kj} + w_{1kj}X_j) \right]. \quad (3.10)$$

Equation (3.9) clearly represents the form of the generalised additive model. When each w_{kj} is equal to zero, it simplifies to the form of the generalised linear model (Equation (3.2)).

A couple of reasons for using the simplified neural network model of the GANN are mentioned in literature. Potts (1999) discuss three main difficulties associated with applying complex MLPs in predictive data mining – inscrutability, model selection and troublesome training. The GANN modelling method brings relief to these difficulties to a great extent. Firstly, the flexible multilayer perceptron is often criticised for being a black box model where underlying relationships are difficult to understand. An additive model is easier to interpret. The use of partial residual plots generated from the GANN training procedure may also be used to understand the effect each predictor has on the response. The use of partial residual plots will be discussed in the following section. Secondly, selecting an MLP model (the neural network architecture) is not a trivial process, requires a large amount of knowledge and expertise (Sarle, 1994) and in most cases trial and error is the modeller's only way of finding a suitable model. Restrictions placed on the network architecture by the GANN model simplify this process to a large extent and Potts (1999) also proposes a recipe for constructing a GANN to aid the modeller. This process will be discussed in more detail in Section 3.4. Thirdly, training an MLP requires a huge amount of computational effort because of the large number of model

parameters to be optimised. Local minima also pose a problem. The simpler GANN architecture brings some relief to this problem. Since the GANN is based on MLP architecture, any method of training used for MLPs may also be used to train GANNs, eliminating the need for a comprehensive backfitting algorithm. Due to these facts, it can be said that GANNs offer a compromise between the flexible but troublesome universal approximators and the inflexible, but docile, linear models (Potts, 1999).

3.3 Partial residual plots

Residual analysis is commonly used to examine the aptness of statistical models. Various residual plots may be examined to determine features of the model such as nonlinearity, the presence of outliers, nonindependence and nonnormality of error terms. In multivariate regression, residual plots against each predictor variable may be used to check whether a curvature effect for that variable is required in the model (Kutner *et al.*, 2005), but it is not sufficient for seeing the correct shape of the curve (Berk & Booth, 1995). Partial residual plots provide graphic information on the marginal importance of a predictor variable, given the other predictor variables already in the model. It also provides insight into the possible nonlinearity of the specific predictor variable and may suggest possible transformations. While residual plots indicate deviations from linearity, partial residual plots indicate the extent of nonlinearity.

The partial residuals, r_j^* , for the j^{th} predictor variable of a generalised additive model, is obtained by adding the residuals from the full model, r , to the fitted function $\hat{f}_j(X_j)$ where

$$\begin{aligned} r &= g_0^{-1}(y) - g_0^{-1}(\hat{y}) \\ &= g_0^{-1}(y) - \left[\hat{\beta}_0 + \sum_{j=1}^p \hat{f}_j(x_j) \right] \end{aligned} \quad (3.11)$$

so that

$$\begin{aligned} r_j^* &= g_0^{-1}(y) - \left[\hat{\beta}_0 + \sum_{j=1}^p \hat{f}_j(x_j) \right] + \hat{f}_j(x_j) \\ &= g_0^{-1}(y) - \hat{\beta}_0 - \sum_{k \neq j} \hat{f}_k(x_k). \end{aligned} \quad (3.12)$$

The partial residual plot for the j^{th} predictor is obtained by plotting r_j^* versus x_j . A regression of r_j^* on x_j would give an estimate for $f_j(X_j)$ (Larsen & McCleary, 1972). This principle may be used in an iterative backfitting algorithm to fit the additive model in the same manner as discussed in the section describing the ACE algorithm (Section 2.3.1).

3.4 Constructing a generalised additive neural network

As mentioned in Section 3.2, the GANN model alleviates difficulties associated with universal approximators. One of the difficulties that still remain to an extent is the process of selecting a suitable GANN architecture. Potts (1999) suggests the following recipe for constructing a GANN model:

- (i) Construct a GANN with one neuron and a skip layer for each input that has been standardised:

$$f_j(X_j) = w_{0j}X_j + w_{1j} \tanh(w_{01j} + w_{11j}X_j). \quad (3.13)$$

- (ii) Fit a generalised linear model to give initial estimates for w_0 and w_{0j} .
- (iii) Initialise the remaining parameters in each hidden layer as random values from a normal distribution with a zero mean and variance equal to 0.1.
- (iv) Iterate:
 - (a) Fit the full GANN model.
 - (b) Examine each of the fitted functions overlaid on their partial residuals. Prune (reduce the number of hidden nodes in) the hidden layers with apparently linear effects and add nodes to hidden layers where the nonlinear trend appears to be underfitted.
 - (c) Set the new parameter estimates equal to those obtained in the current iteration.

Potts (1999) explains that the use of standardised inputs, the hyperbolic tangent activation function and parameter initialisation values close to zero all contribute to a relatively stable fitting algorithm that does not take large leaps into so-called treacherous regions of the parameter space. He warns that local minima may still occur, but that these can usually be seen in the partial residual plots and serve as an indication to retry with new random initialisations.

Potts also notes that the use of partial residual plots for model selection is subjective and he suggests the incorporation of formal measures of fit such as performance on external validation data or the less expensive Bayesian Information Criterion. In addition to the process being subjective, De Waal and Du Toit (2007) note that the interactive process of pruning and adding nodes over several iterations could become tedious and time consuming. They suggest a stepwise model selection method for GANNs that is both objective and fully automatic, but is still based on this recipe for constructing a GANN. The following section contains a description of this approach.

3.5 Automated construction of generalised additive neural networks

The stepwise model selection method for GANNs suggested by De Waal and Du Toit (2007) is both fully automatic and uses objective model selection criteria such as the Schwarz Bayesian Criterion. It therefore proposes a solution to the drawbacks of the GANN construction recipe of Potts (1999) discussed in the previous section. The stepwise algorithm utilises a greedy best-first search algorithm that makes multi-steps over a search tree constructed from possible GANN model architectures.

The implementation of such an algorithm involves several factors that have to be taken into consideration. These include:

- selection of activation functions;
- a GANN architecture encoding method to represent the search space of possible architectures;
- objective model selection criteria to be used;
- a tree expansion method to form the basis of the search algorithm;
- methods for improving performance;
- an intelligent start, and
- model averaging as a means of accounting for model uncertainty.

These factors are discussed in the following sections with a final section explaining the implementation of the algorithm.

3.5.1 Selection of activation functions

A subset of possible activation functions for the GANN output layer was chosen for the implementation of the automatic GANN construction method (the hidden units would use the hyperbolic tangent activation function). These activation functions together with appropriate error functions are listed in Table 3.1 and are able to solve a large class of problems. The activation function for each problem is selected according to the range of the observed data for the response or target variable. An interval target variable with an identity activation function is assumed by default. If the target variable has a minimum value greater than or equal to zero, the exponential activation and Poisson error functions are applied. If the target variable has a minimum value greater than or equal to minus one and a maximum value less than or equal to one, the hyperbolic tangent activation and normal error functions are applied. Finally, if the target variable assumes binary values, the multiple logistic activation and multiple Bernoulli error functions are used.

<i>Activation function</i>	<i>Link function</i>	<i>Target range</i>	<i>Error function</i>
Identity	Identity	$(-\infty, \infty)$	Normal
Hyperbolic tangent	Inverse hyperbolic tangent	$[-1, 1]$	Normal
Exponential	Log	$[0, \infty)$	Poisson
Multiple Logistic	Logit	Binary	Multiple Bernoulli

Table 3.1: Activation, link and error functions used in the GANN construction algorithm

3.5.2 GANN architecture encoding

As described in Section 3.2, a GANN consists of a separate MLP for each input variable. Each separate MLP may have its own sub-architectural form. Du Toit (2006) suggests the restriction of the possible sub-architectures to the possibilities listed in Table 3.2. Since it restricts the possible complexity of the underlying neural network and therefore the size of the search space, this restriction is intended to simplify the implementation of the search algorithm and not the model itself. De Waal and Du Toit (2007) point out that the model selection criteria are used to prevent unnecessary complex neural networks from being exploited in a specific modelling problem.

<i>MLP present for input</i>	<i>Skip layer present</i>	<i>Number of hidden nodes</i>	<i>GANN sub- architecture identifier</i>
No	-	-	0
Yes	Yes	0	1
Yes	No	1	2
Yes	Yes	1	3
Yes	No	2	4
Yes	Yes	2	5
Yes	No	3	6
Yes	Yes	3	7
Yes	No	4	8
Yes	Yes	4	9

Table 3.2: GANN sub-architecture possibilities and identifiers

Table 3.2 also displays the corresponding sub-architecture identifiers used to make up the encoding method for the search algorithm. A specific GANN architecture may be represented by

a string of digits from the set of sub-architecture identifiers $S = \{0,1,2,3,4,5,6,7,8,9\}$. For example, a GANN architecture with three inputs, the first having an MLP with a skip layer and 2 hidden nodes, the second input removed and the third input having an MLP with no skip layer and 1 hidden node, is represented by the string 502. A complete search space for the GANN architecture in this case would consist of all the 3-length permutations (with repetitions) from S and would have a size of $10^3 = 1000$. Even with the sub-architecture restriction, the size of the search space increases with an order of 10 for every additional input variable, confirming the importance of an appropriate combination of model selection criteria used as an ordering method and a greedy best-first search algorithm.

3.5.3 Model selection criteria

The set of model selection criteria opted for by Du Toit (2006) includes the Akaike Information Criterion (AIC) and the Schwarz Bayesian Criterion (SBC). These criteria are based on different assumptions, but they both penalise large complex models that tend to overfit (Du Toit, 2006). They are therefore not only appropriate for predictive data mining, but also for restricting the GANN architecture search algorithm.

The Akaike Information Criterion is based on the assumption that the model being evaluated can be close to, but never equal to the true model. It therefore attempts to measure the amount of deviation, or distance, from the true model. AIC is referred to as being an asymptotically efficient criterion, able to identify the model closest to the true model from an increasingly larger set of candidates. AIC selects the model that best approximates the true model and will select the model with the smallest amount of parameters that fit the data well. The AIC is defined as

$$AIC = -2 \ln \left[L(\hat{\theta}|y) \right] + 2W \quad (3.14)$$

where $L(\hat{\theta}|y)$ is the likelihood function of the parameter vector θ , given y , and W is the number of parameters to be estimated. When least squares estimation is used with normally distributed errors and constant variance, Equation (3.14) may be expressed as

$$AIC = n \ln(\hat{\sigma}^2) + 2W \quad (3.15)$$

where $\hat{\sigma}^2 = \frac{\sum \hat{\varepsilon}_m^2}{n}$, $\hat{\varepsilon}_m$ are the estimated errors for candidate model m , and W is the number of parameters to be estimated including the constant term and σ^2 . Minimising AIC will result in the optimal model being selected.

The Schwarz Bayesian Criterion is based on the assumption that the model being evaluated is a candidate for being the true model. The SBC is referred to as being asymptotically

consistent, able to identify the correct model within the candidate set. The generic SBC is defined as

$$\text{SBC} = -2 \ln \left[L(\hat{\theta}|y) \right] + W \ln(n). \quad (3.16)$$

When the Gaussian error model applies, Equation (3.16) may be expressed as

$$\text{SBC} = n \ln(\hat{\sigma}^2) + W \ln(n) \quad (3.17)$$

in the same manner as Equation (3.15). According to Du Toit (2006), SBC penalises model complexity more than AIC when $W > 7$.

3.5.4 Search tree expansion method

In order to implement the automation of the iterative adjustment of model complexity, rules need to be defined on which to expand or grow the search tree of possible GANN architectures.

De Waal and Du Toit (2007) propose the starting point of the search to be a linear model: for each input an MLP is created with a skip layer and without any hidden nodes. Starting with a nonlinear architecture as proposed by Potts (1999), helps with interpreting the partial residual plots and identifying slight nonlinear trends. Since the automated procedure relies on an objective model selection criterion, the need for interpreting the partial residual plots falls away together with the need of a nonlinear starting model. The advantage of starting with a linear model becomes apparent with the proposed search tree expansion method: for each leave node add $2p$ child nodes, one for each input sub-architecture change to one value lower than the parent value while keeping the rest of the architecture the same, and one for each input sub-architecture change to one value higher than the parent value while keeping the rest of the architecture the same. For example, a starting node with three variables, 111, will gain six child nodes, 011, 101, 110, 211, 121 and 112. All the models are fitted and evaluated with the model selection criterion. The criterion orders all the models, and the most appropriate model's architecture will be the first to be expanded further. Note that all three input variables are removed from the proposed models in this first expansion step. If the selection criterion determines that it is worth removing the particular input, variations on architectures excluding that variable will be explored first.

3.5.5 Improvements on performance

The generation of the GANN sub-architectures explained above would allow for previously generated GANN architectures to be generated again during the search procedure. Previously searched areas in the tree can be avoided in subsequent search iterations by keeping a history

of the search tree. Before evaluating a new node and adding it to the search tree history, a step is performed to check whether it has been generated previously.

The expansion method described above allows for single-step expansions only – in a single iteration of the search procedure all the new child nodes that are generated differ from the parent nodes with only a single sub-architecture change. In problems with a large number of predictor variables, the number of search iterations it takes to arrive at an optimal GANN architecture could become extremely large. One way of improving the performance of the procedure would be to allow for multi-step expansions: start each expansion iteration with the single-step expansion described above; using the model selection criterion, for each parent node, identify the set of child nodes and thus the single-step sub-architecture changes, which show improvement on the parent node; create one new child node that combines these sub-architecture changes and evaluate the new node with the model selection criterion; finalise the iteration by selecting the best node (from all the nodes) to be expanded further in the next iteration. This approach takes advantage of the possibility that a combination of the single-step sub-architecture changes that showed improvement on the parent architecture would most likely also show improvement on the parent architecture and could even show improvement on the independent changes themselves. This multi-step expansion method improves on the performance by including relatively promising architectures that would have been generated in subsequent iterations, in the current iteration.

The final improvement focuses on memory optimisation. Since an iteration of the expansion method requires that all newly and previously generated nodes be evaluated for both existence and the model selection criterion, it could require a huge amount of computing memory. De Waal and Du Toit (2007) note that the multi-step expansion method causes the search to move to another area in the search tree and that it is very unlikely that the multi-step would ever reach the same search area again. The greedy search strategy would avoid revisiting areas of the search space considered to be bad in the context of the model selection criterion. With this in mind they propose that the search tree history be cleared after a single iteration and that the last selected tree node becomes the new root node of the search tree.

3.5.6 An intelligent start

In Section 3.5.4 it was explained that the linear model is a good starting point for the automated GANN construction algorithm. Fitting the linear model would involve the estimation of $p+1$ weights including the bias, w_0 , and one for each input variable, w_{0j} . The starting point suggested by Potts (1999), which is a GANN with a skip layer and one node in the hidden layer for each input, would also involve initialisation of the weights in the hidden nodes. With the

search tree expansion method and the initialising method suggested by Potts, it would be possible to use any starting architecture and still arrive at the same optimal model. The only difference would be the number of search tree iterations it would take to find the optimal node in the search tree. Using an intelligent starting architecture that is closer to the optimal one, could significantly decrease the number of iterations in the search process.

De Waal and Du Toit (2007) propose a pre-processing task to find a more promising starting architecture by making an intelligent guess of the optimal model architecture. It is based on the analysis of stepwise regression and Pearson product-moment correlation coefficients. The following two paragraphs contain brief descriptions of stepwise regression and the Pearson product-moment correlation coefficient.

The best subsets regression and stepwise regression methods are discussed by Kutner *et al.* (2005) as the two most commonly used automatic search procedures for linear model selection. Such procedures identify one or more “good” subsets of predictor variables to include in the final model according to a specified criterion instead of evaluating all possible subsets. The results of these methods would give the analyst a good indication of what the optimal model might look like. The stepwise regression method is computationally even more efficient in cases of a very large number of predictor variables because of the sequential manner of adding and deleting potential predictor variables. Where the best subsets regression identifies a subset of best models to be considered, the stepwise regression, however, identifies only a single best model. This is considered a weakness of the method - it does not show the fact that several other variable subsets may also be good. Kutner *et al.* (2005) therefore suggest using the model from the stepwise regression as a starting point for searching for other good models, which is exactly what the automated GANN construction method is used for. (Note that Breiman and Friedman (1985) also utilise stepwise regression for variable selection in the ACE procedure discussed in Section 2.3.)

The coefficient of correlation between two random variables, Y and X , provides information on the degree of linear correlation between the variables and can be expressed as:

$$\rho\{Y, X\} = \frac{\sigma\{Y, X\}}{\sqrt{\sigma^2\{Y\}\sigma^2\{X\}}} \quad (3.18)$$

where $\sigma\{Y, X\} = E\{(Y - E\{Y\})(X - E\{X\})\}$ is the covariance of Y and X and $\sigma^2\{Y\} = E\{Y^2\} - (E\{Y\})^2$ is the variance of Y . The coefficient of correlation can take on a value in the range $[-1, 1]$. When $\sigma\{Y, X\} = 0$ then $\rho\{Y, X\} = 0$ and variables Y and X are said to be (linearly) uncorrelated. The Pearson product-moment correlation coefficient (or simply the

Pearson correlation coefficient) is the maximum likelihood estimator of $\rho\{Y, X\}$ (Kutner *et al.*, 2005) and is given by:

$$p_{Y,X} = \frac{\sum(Y - \bar{Y})(X - \bar{X})}{\sqrt{\sum(Y - \bar{Y})^2 \sum(X - \bar{X})^2}}. \quad (3.19)$$

The Pearson correlation coefficient may be used without the assumption of Y and X being random variables. As with the coefficient of correlation, a Pearson correlation coefficient value close to -1 or 1 indicates a strong linear relationship between the variables and a value close to zero indicates either no relationship or a complex non-linear relationship.

The pre-processing task proposed by De Waal and Du Toit (2007) uses the Pearson correlation coefficient as an additional criterion to obtain a stepwise regression decision matrix as depicted in Table 3.3. The decision matrix is used to determine the GANN search starting architecture. The process would follow a normal stepwise regression procedure where, in each step, a predictor variable is added to the model and another variable already in the model is potentially dropped from the model. For the variable under consideration, X_j , suppose the stepwise regression decision is to add it to the existing model. Examine the Pearson correlation coefficient: if the absolute value is greater than or equal to 0.25, a linear association is considered appropriate and a GANN sub-architecture of 1 is used; if it is less than 0.25, linear correlation is very low but the variable still contributes to the response so that a sub-architecture of 2 (a slight nonlinear architecture) is used. If, on the other hand, the stepwise regression decision is to remove variable X_j and the absolute Pearson correlation coefficient is greater than or equal to 0.25, the variable is removed from the GANN architecture. The only remaining case occurs when the stepwise regression indicates removal but the absolute Pearson correlation coefficient is less than 0.25. In such a case uncertainty exists about the form of the relation to the response – it either does not exist or is a very complex nonlinear relation. De Waal and Du Toit (2007) suggest keeping the variable in the GANN search starting architecture with a linear sub-architecture.

		Stepwise regression decision	
		Remove	Add
Pearson correlation coefficient	$ p_{Y,X_j} < 0.25$	Uncertain (1)	Nonlinear (2)
	$ p_{Y,X_j} \geq 0.25$	Remove (0)	Linear (1)

Table 3.3: Pre-processing decision matrix

3.5.7 Model averaging

The result of using a data-dependant model selection method based on a selection criterion, is a single approximating model that does not account for model uncertainty – the question whether the selected model fits the underlying relationships best rather than the finite sample data set. Model uncertainty may therefore be considered components of the variance in the estimated model parameter estimates that occur over more than one selected model (Anderson & Burnham, 1998). Not accounting for model uncertainty may lead to overconfidence in the predictive power of a particular model and Anderson and Burnham strongly recommend using model averaging for inference rather than using a single selected model. Du Toit (2006) proposes using approximating Bayesian model averaging for generalised additive models based on the findings made by Lee (1999).

Bayesian model averaging accounts for model uncertainty since response inferences are based on a weighted average over a set of models instead of one specific model. The weights are assigned according to the posterior probabilities of each of the models. The posterior probability of a quantity of interest, Δ , such as a parameter, prediction or a course of action in each model, may be calculated using the standard Bayesian formalism (Leamer, 1978):

$$P(\Delta|D) = \sum_{k=1}^K P(\Delta|D, M_k) P(M_k|D) \quad (3.20)$$

where D is the data used to build the models and M_k is the k^{th} model from the set of K models. $P(\Delta|D, M_k)$ is the marginal posterior density of Δ given the data and a specific model, M_k . $P(M_k|D)$ is the posterior probability of model M_k , given the data. To compute the posterior probability of a model according to Bayes' theorem, it is necessary to have a prior probability, $P(M_k)$, for each of the models under consideration (if little or no prior probability of a model is available, a reasonable assumption is that all models are equally likely (Hoeting *et al.*, 1999; Lee, 1999):

$$P(M_k|D) = \frac{P(D|M_k)P(M_k)}{\sum_{l=1}^K [P(D|M_l)P(M_l)]} \quad (3.21)$$

Lee (1999) studied the asymptotic behaviour of Bayesian model averaging for neural networks and investigated several methods for estimating the integral when computing the marginal posterior density of model M_k :

$$P(D|M_k) = \int P(D|\theta_k, M_k) P(\theta_k|M_k) d\theta_k \quad (3.22)$$

where θ_k is the parameter vector for model M_k , $P(\theta_k|M_k)$ is the prior distribution of θ_k and $P(D|\theta_k, M_k)$ is the likelihood. Lee concludes that the SBC is one of the most robust methods to use (compared to quadrature and Monte Carlo numerical methods, Laplace approximation, importance sampling, reciprocal importance sampling, bridge and path sampling, density-based approximation and partial analytical integration). Other examples used in literature also show improved out of sample predictive performance (Hoeting *et al.*, 1999) when using Bayesian model averaging. With the SBC as defined in Equation (3.16), approximating Bayesian model averaging for multilayer perceptrons involves estimating (3.21) with

$$P(M_k|D) \approx \frac{P(D|M_k)}{\sum_{l=1}^K P(D|M_l)} \approx \frac{e^{-SBC_k}}{\sum_{l=1}^K e^{-SBC_l}}. \quad (3.23)$$

To implement the model averaging for the automatic GANN construction, recall that a GANN model has the form:

$$g_0^{-1}(E\{Y\}) = w_0 + \sum_{j=1}^p f_j(X_j). \quad (3.24)$$

Du Toit (2006) defines

$$P(\Delta|M_k, D) = E\{\Lambda_k\} = g_{0_k} \left(w_{0_k} + \sum_{j=1}^p f_{j_k}(X_j) \right) \quad (3.25)$$

where $\Lambda_k = I(\Delta|M_k, D)$ and I is an indicator function. It then follows from Equations (3.20) and (3.23) that

$$P(\Delta|D) \approx \sum_{k=1}^K E\{\Lambda_k\} \frac{e^{-SBC_k}}{\sum_{l=1}^K e^{-SBC_l}}. \quad (3.26)$$

The indicator function is used to rank the models according to the SBC. Equation (3.26) is used to approximate Bayesian model averaging in the automated GANN construction algorithm. Apart from averaging the predicted response, the univariate functions, $f_1(X_1), \dots, f_j(X_j)$, are also averaged to obtain the partial residual plots for the combined model.

Although model averaging using Equation (3.26) would produce better prediction results than any of the models under consideration themselves (De Waal & Du Toit, 2007), Lee (1999) warns that model averaging reduces the interpretability of the final model and that it should not be used in cases where it is essential to understand the model.

With the GANN activation function selection, architecture encoding, model selection criteria, search tree expansion method, performance considerations, model averaging and (optional)

pre-processing step explained, the implementation of the automatic GANN model construction algorithm is explained in the following section.

3.5.8 The AutoGANN implementation

The automated GANN construction algorithm devised by Du Toit (2006) is implemented in the SAS® programming language and integrated into the SAS® Enterprise Miner™ solution as the AutoGANN model node. The user would utilise the standard functionality available in SAS® Enterprise Miner™ to create an input data set with input variables and a target variable from the observed data, $(x_{1i}, x_{2i}, \dots, x_{pi}, y_i)$ for $i = 1, \dots, n$. An AutoGANN model node is connected to the input, and the parameters for the model are set. An option may be invoked to determine an intelligent start architecture for the AutoGANN (refer to Section 3.5.6). The AutoGANN model node has six user-specified parameters:

- **The model selection criterion.** The user may select a model selection criterion. The options include variations on AIC and the SBC. The default is the SBC for a least squares analysis (*SBC_dev*).
- **The start architecture.** The user is able to specify the start architecture for the model search procedure in the form of the GANN architecture encoding described in Section 3.5.2. The default is a linear architecture (*Linear*).
- **The sub-architecture search space.** The user may restrict the possible GANN sub-architectures by listing only those required from the sub-architecture identifiers listed in Table 3.2. The default value is *0123* and would restrict the GANN sub-architecture complexity to an MLP with a skip layer and two hidden nodes.
- **Partial residual plots.** The user may turn the creation of partial residual plots on or off by specifying *Yes* or *No*. The default value is *Yes*.
- **Time limit.** The user may select a time limit for the model search: if an optimal GANN model has not been found when reaching the time limit, the best model found so far is reported. The default value is *Initialise* which would only report the model resulting from the initialised starting architecture. Other options range from 15 seconds to 2 days.
- **The number of models.** The user is able to specify the number of models to use for model averaging. The procedure would then average over the top number of models ranked according to the SBC. The default value is 1, indicating that no model averaging will be performed.

After setting the parameters, the user may execute the AutoGANN model node. A description of the algorithm followed by the AutoGANN system is given next.

- (i) Determine appropriate activation, link and error functions according to the range of the target variable.
- (ii) Construct a GANN according to the start architecture specified by the user. If the default value was specified, it would be a GANN with a skip layer and no hidden nodes with the univariate functions initialised to $f_j(x_{ji}) = w_{0j}x_{ji}$.
- (iii) Initialise the GANN.
 - (a) A generalised linear model is fitted to give initial estimates for w_0 and w_{0j} .
 - (b) If the start architecture contains hidden nodes, the remaining weights are initialised to random values from a normal distribution with a zero mean and a variance of 0.1.
- (iv) Fit the GANN.
 - (a) The full GANN model is fitted using the selected error function.
 - (b) Evaluate the fitted GANN using the model selection criterion specified by the user.
 - (c) Denote the node as the root of the search tree.
 - (d) Denote the node as *unexpanded* (the model is available for expansion).
- (v) Iterate until the search space has been exhausted or until the time limit has been reached:
 - (a) Rank all the generated GANN nodes according to the model selection criterion.
 - (b) If all the nodes have been *expanded* or the time limit has been reached, terminate step (v).
 - (c) Apply the search tree expansion method to the highest ranked *unexpanded* node. If any of the newly generated child nodes already exist in the search tree, remove it from the current set of newly generated nodes.
 - (d) For each of newly generated nodes, evaluate the associated model, m , with the selection criterion, add it to the search tree and denote it as *unexpanded*.
- (vi) Report the final model:
 - (a) With all the generated models ranked according to the selection criterion, perform model averaging on the top K models, where K is the number of models specified by the user. Both the predicted target values and the univariate function values are averaged for $i = 1, \dots, n$.

- (b) Generate the partial residual plots if indicated so by the user-specified parameter.

The implemented procedure also creates additional reports for example fit statistics, the populate search space, the solution path and the architecture of the final model. Resulting data sets are exported to disc and score code is generated as part of the model node. Adding newly observed data as a score input data set to the existing model node would create predictions based on the selected model. Figure 3.1 is an example screen of the AutoGANN model node applied to an example in SAS® Enterprise Miner™ 5.1. The example contains both input and score data. The default model selection and model averaging options were used, except the time limit that was set at 5 minutes. The status tab indicates that the AutoGANN model node has completed execution, has found the optimal model and has scored the new data within 19 seconds.

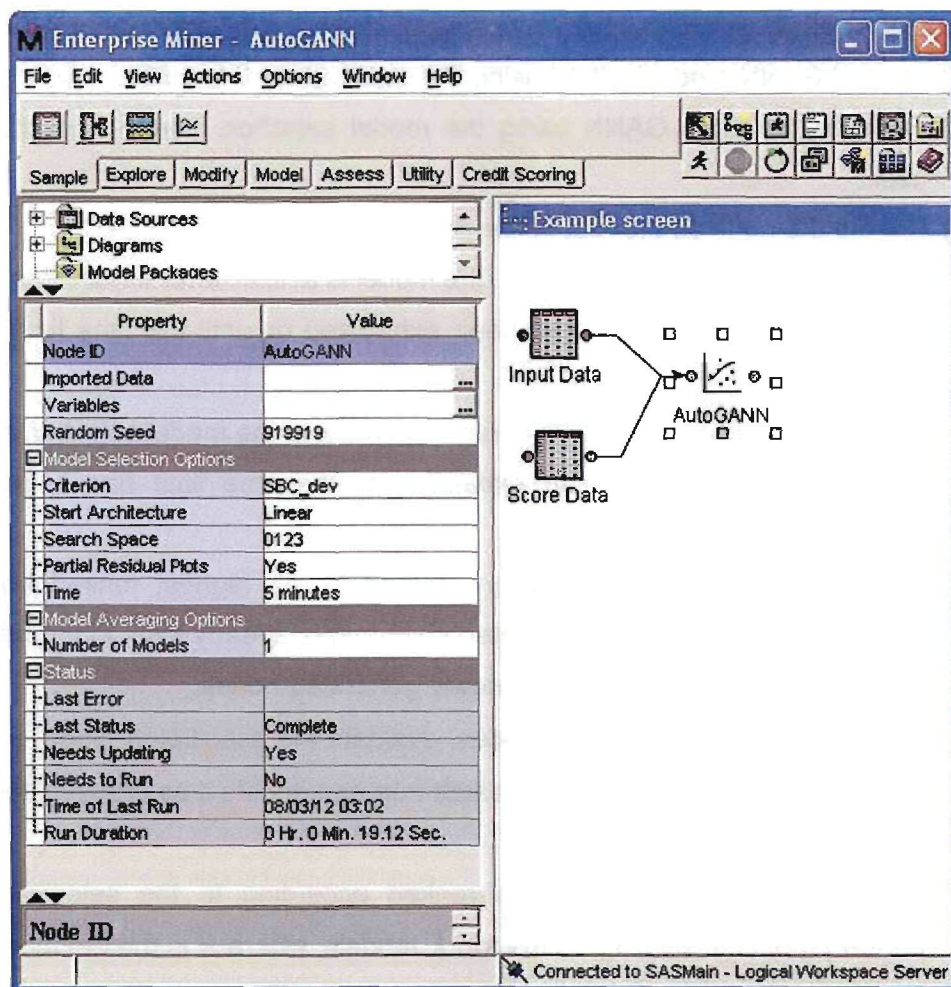


Figure 3.1: Example screen of the AutoGANN model node in Enterprise Miner™

Figure 3.2 shows the AutoGANN model node result screen displaying some of the available output for the example. It shows that the optimal model has a SBC value of 171.65 and was found at the third level of the search tree (DEPTH = 2). The optimal model has a GANN sub-architecture code of 3 (an MLP with a skip layer and one hidden node) for the one input variable and has 5 model weights (PARAMS). The output and log files also contain values for the weight estimates and the activation functions of the optimal model. In this case the GANN model has the form:

$$\ln(E\{Y\}) = w_0 + w_{01}x_1 + w_{11} \tanh(w_{011} + w_{111}x_1). \quad (3.27)$$

Results - AutoGANN

File Edit View Window

Populated Search Space

I	SBC_DEV	NETWORKID	USED	DEPTH	NODENUM	PARENTNUM	PARAMS
1	171.652085233		1	2	3	2	5
2	182.566146062		1	1	2	1	4
3	196.564978751		1	0	1	0	2

Solution Path

I	SBC_DEV	NETWORKID	USED	DEPTH	NODENUM	PARENTNUM	PARAMS
1	171.652085233		1	2	3	2	5
2	182.566146062		1	1	2	1	4
3	196.564978751		1	0	1	0	2

Inputs

I	INPUT	SUBARCHITECTURE ▲	LABEL
1x		3	

Fit Statistics

TARGET	Fit statistics	Statistics Label	Train	Validation	Test
y	DFT_	Total Degrees of Freedom.	200		
y	DFE_	Degrees of Freedom for Error.	194		
y	DFM_	Model Degrees of Freedom.	6		
y	NW_	Number of Estimated Weights.	6		
y	AIC_	Akaike's Information Criterion.	425.2777		
y	SBC_	Schwarz's Bayesian Criterion.	445.0676		
y	ASE_	Average Squared Error.	11.1208		
y	MAX_	Maximum Absolute Error.	22.24874		
y	DIV_	Divisor for ASE.	200		
y	NOBS_	Sum of Frequencies.	200		
y	RASE_	Root Average Squared Error.	3.334787		
y	SSE_	Sum of Squared Errors.	2224.161		
y	SUMW_	Sum of Case Weights Times Freq.	200		
y	FPE_	Final Prediction Error.	11.80869		
y	MSE_	Mean Squared Error.	11.46475		
y	RFPE_	Root Final Prediction Error.	3.436377		
y	RMSE_	Root Mean Squared Error.	3.385963		
y	AVERR_	Average Error Function.	2.066389		
y	ERR_	Error Function.	413.2777		
y	MISC_	Misclassification Rate.			
y	WRONG_	Number of Wrong Classifications.			

Figure 3.2: Example screen of the AutoGANN model node results in Enterprise Miner™

Note that the SBC value in the Fit Statistics window is incorrect. SAS® Enterprise Miner™ is not able to infer the correct degrees of freedom of the special GANN architecture due to some frozen links (Potts, 2000).

Finally, examination of the partial residual plots is made easy with automatically generated graphic files. Figure 3.3 contains the partial residual plot generated for the variable of this example. It is superimposed on the fitted spline and contains tick marks for easy examination of the function values.

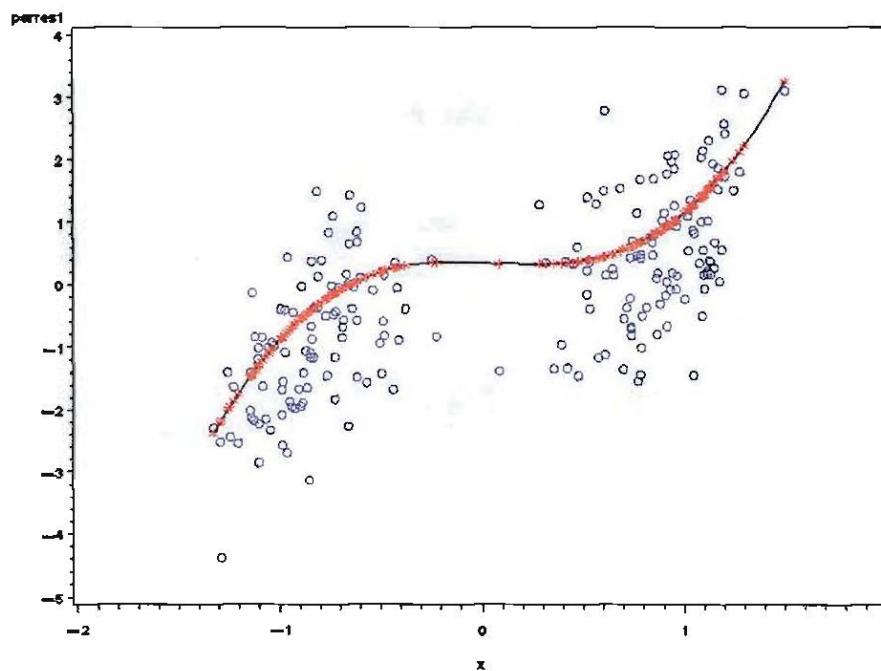


Figure 3.3: Example of a partial residual plot generated by the AutoGANN system

The following section contains results obtained by applying the AutoGANN system to the examples presented in Section 1.3.

3.6 AutoGANN examples

The AutoGANN system was applied to all the example cases described in Section 1.3. In each case the procedure was run for a maximum duration of 5 minutes (unless indicated otherwise), used a linear starting GANN architecture and the SBC as the model selection criterion. The sub-architecture search space was limited to $\{0,1,2,3\}$ to allow for a maximum of one hidden node. No model averaging was performed.

For each example, the partial residuals are plotted against each variable as well as the scatter plots describing the resulting model.

3.6.1 Simulated examples

First simulated example

A set of 200 data points was generated using $y = \exp(x^3 + \varepsilon)$ with x^3 and ε drawn randomly from standard normal distributions. The AutoGANN system found the optimal model within 15 seconds. The selected sub-architecture for input variable x is 3 – an MLP with a skip layer and a hidden node. Figure 3.4 contains the partial residual plot for the input. It clearly shows the nonlinear relationship with the target and the fitted univariate function has the same form as the known optimal transformation for the input variable, namely x^3 . Figure 3.5 contains a plot of the AutoGANN model results superimposed upon the original training input data. The model has an R^2 value of 0.45 and an average square error of 11.12.

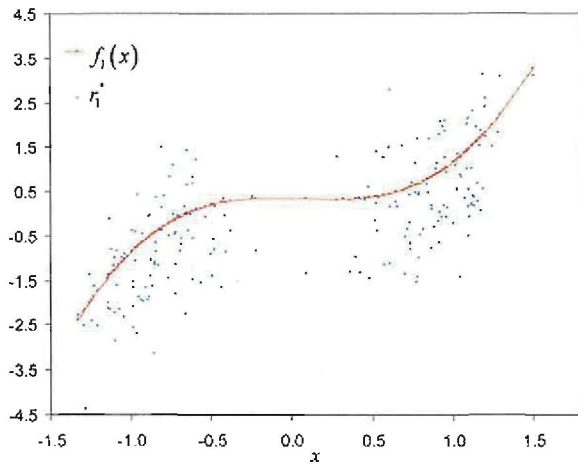


Figure 3.4: First simulated example – partial residual plot

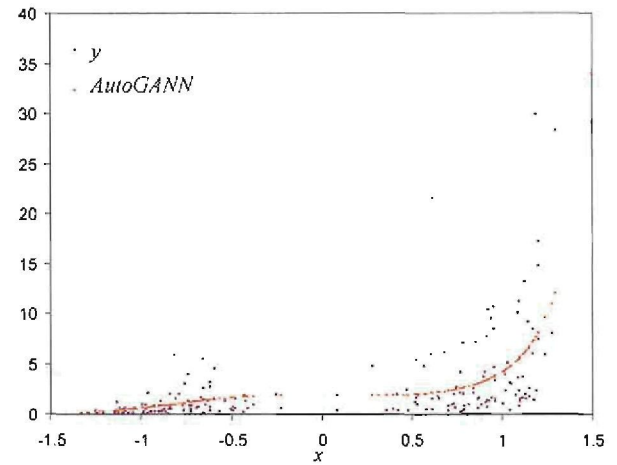


Figure 3.5: First simulated example – AutoGANN model results

Second simulated example

A set of 200 data points was generated using $y = \exp[\sin(2\pi x) + \varepsilon/2]$ with x drawn from a uniform distribution between 0 and 1, and ε drawn independently from a standard normal distribution. The AutoGANN system found the optimal model within 21 seconds. The selected sub-architecture for input variable x is 3 – an MLP with a skip layer and a hidden node. Figure 3.6 contains the partial residual plot for the input. It clearly shows the nonlinear relationship with target and the fitted univariate function has the same form as the known optimal transformation for the input variable, $\sin(2\pi x)$. Figure 3.7 contains a plot of the AutoGANN model results superimposed upon the original training input data. The model has an R^2 value of 0.54 and an average square error of 0.92.

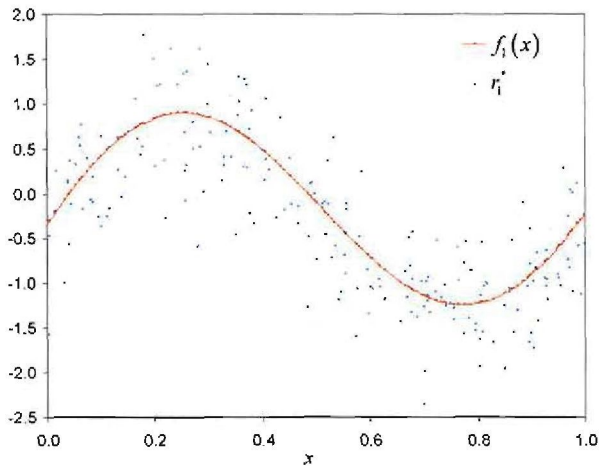


Figure 3.6: Second simulated example – partial residual plot

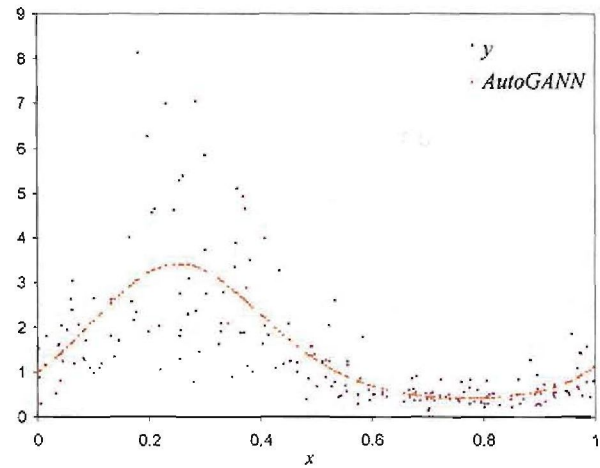


Figure 3.7: Second simulated example – AutoGANN model results

Third simulated example

A set of 200 data points, $\{(y_i, s_i, t_i), 1 \leq i \leq 200\}$, was generated from $y = st$ where s and t are two independent variables with uniform distributions between -1 and 1. The AutoGANN application to this example was performed three times – in each case different a setup was used in SAS® Enterprise Miner™.

The first case simply followed the default setup used for all the examples. The AutoGANN system found the optimal model within 35 seconds. The selected sub-architectures were 0 for input s and 1 for input t , indicating s was discarded and a linear relationship assumed for t . Figure 3.8 contains the partial residual plot for the input variable t . Figure 3.9 shows the AutoGANN model results superimposed upon the original training input data plotted against st . The AutoGANN was not able to model the input interaction. The model has an R^2 value of 0.06 and an average square error of 0.09.

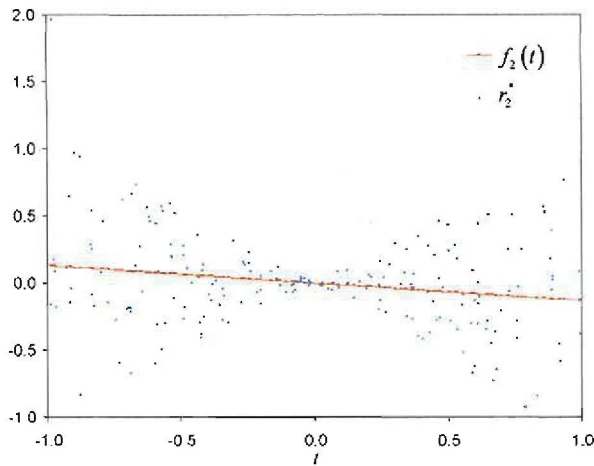


Figure 3.8: Third simulated example case 1 – partial residual plot for t

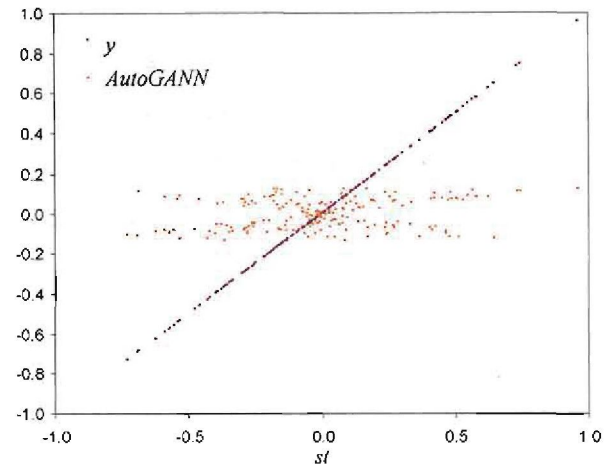


Figure 3.9: Third simulated example case 1 – AutoGANN model results

For the second application of this example an *Interactions* node was added before applying the AutoGANN model node. This is necessary since an additive model is not able to model input variable interactions as seen in case 1. This is a trivial task since no additional settings need to be specified for the *Interactions* node. The node derives two additional input variables, s^2 and st . Figure 3.10 shows the SAS® Enterprise Miner™ setup for the *Interactions* node.

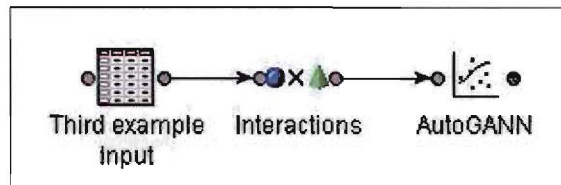


Figure 3.10: Third simulated example case 2 – *Interactions* node setup

Since the data was generated from st , one would expect the AutoGANN model to discard variables s , t and s^2 and fit the model with only st in a linear sub-architecture. When the sub-architecture search space is restricted to $\{0,1\}$, this is exactly the case. When the default setting of $\{0,1,2,3\}$ is used, however, the AutoGANN system creates a model with an unnecessary complex architecture: a sub-architecture of 3 for input variables s and st , 2 for variable t and discards s^2 . The AutoGANN used all the available time (5 minutes) and reported this model as the final model. Figure 3.11, Figure 3.12 and Figure 3.13 contain the partial residual plots for the input variables s , t and st respectively. The partial residual plots for s and t clearly show that these variables do not contribute to the variance in the target variable by themselves and that they should be removed. The partial residual plot for st indicates a fairly

linear relationship but not a perfect linear relationship as one would expect. The partial residuals are small but become larger near the endpoints of the plot. This is also the case when inspecting the AutoGANN model results superimposed upon the original training input data plotted against st (Figure 3.14). The AutoGANN was not able to fit the data perfectly, even though one of the available input variables explains the target variable perfectly. The model is close, however, with an R^2 value of 9.99×10^{-1} and an average square error of 6.36×10^{-5} .

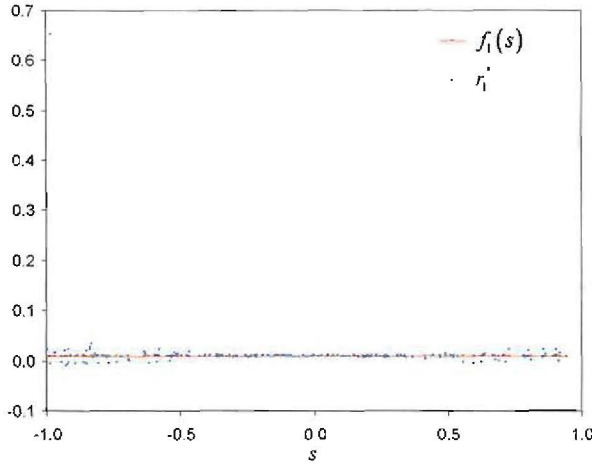


Figure 3.11: Third simulated example case 2 – partial residual plot for s

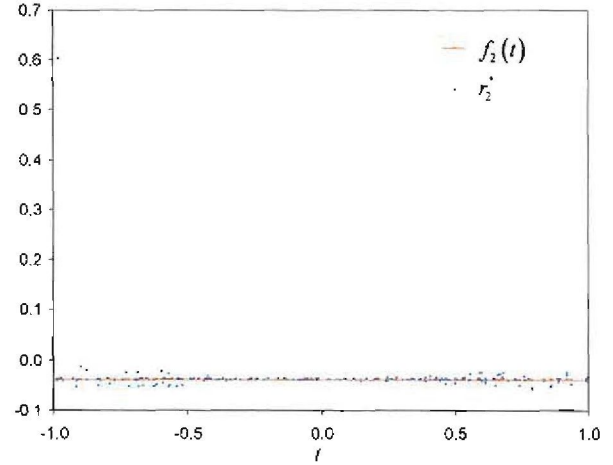


Figure 3.12: Third simulated example case 2 – partial residual plot for t

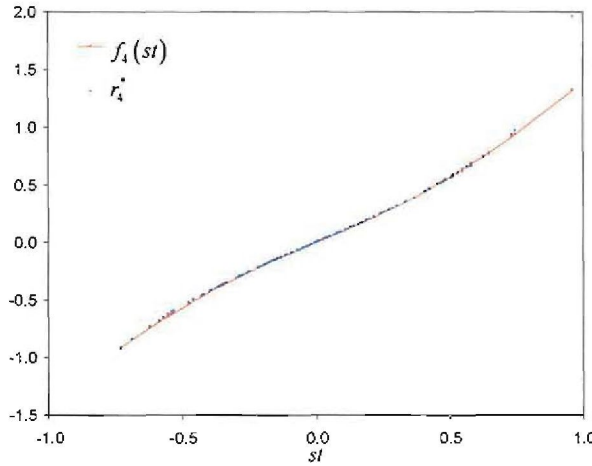


Figure 3.13: Third simulated example case 2 – partial residual plot for st

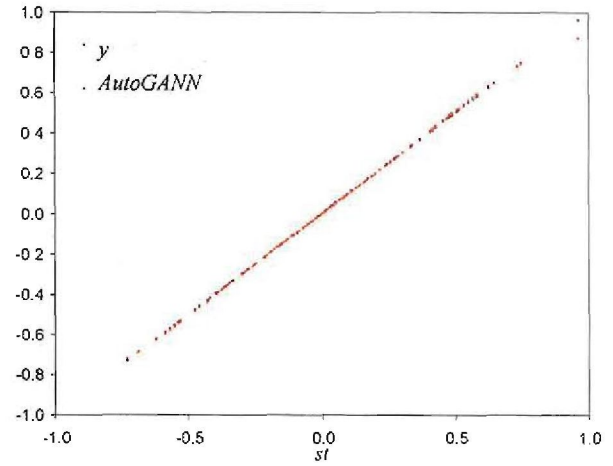


Figure 3.14: Third simulated example case 2 – AutoGANN model results

This behaviour of creating an unnecessary complex architecture could be ascribed to the manner in which the activation function is selected for the GANN output layer. Because the target variable falls in the interval $[-1,1]$, the hyperbolic tangent activation function is selected, adding an unnecessary component of nonlinearity to the model search space. For the third case

of this example a slight change was made to the simulated input data by simply altering one of the 200 data points to produce a target value outside the range $[-1,1]$. Reusing the setup of case 2 with this new input data set produced a much better result since the identity activation function was selected for the output layer. The AutoGANN system ran for the entire 5 minutes allowed and produced the expected model architecture: inputs s , t and s^2 were removed (sub-architecture 0) and a linear sub-architecture was created for input st . Figure 3.15 contains the partial residual plot for the input variable st . It now depicts the linear relationship between st and y . Figure 3.16 contains a plot of the AutoGANN model superimposed upon the original training input data plotted against st . The model is a perfect fit to the data with an R^2 value of 1.00 and an average square error of 1.74×10^{-19} .

Included in Section 5.1.3, are 3-dimensional graphic illustrations of the model predictions on a more densely populated input space for all three cases discussed in this section.

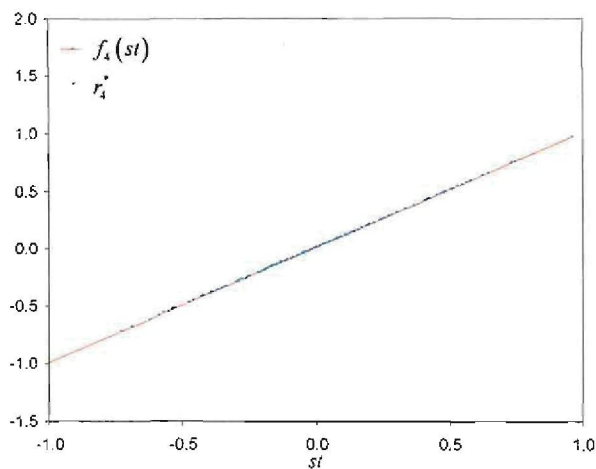


Figure 3.15: Third simulated example case 3 – partial residual plot for st

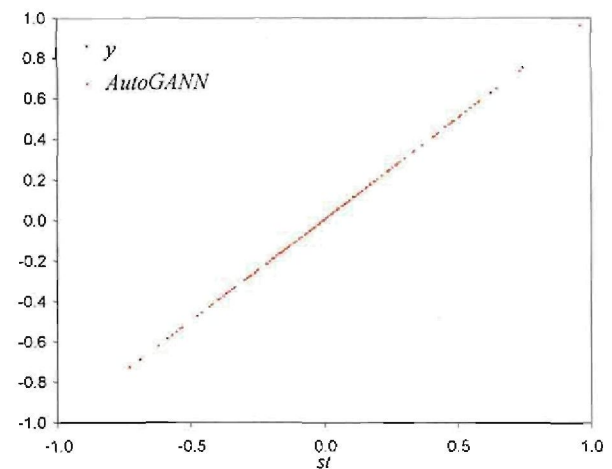


Figure 3.16: Third simulated example case 3 – AutoGANN model results

3.6.2 Ethanol fuel example

In this problem the input variable is E and the target variable is NOX . The AutoGANN system found the optimal model within 38 seconds. The selected sub-architecture for input variable E is 3 – an MLP with a skip layer and a hidden node. Figure 3.17 contains the partial residual plot for the input. It depicts a nonlinear relationship with the target. Figure 3.18 contains a plot of the AutoGANN model results superimposed upon the original training input data. The model has an R^2 value of 0.92 and an average square error of 0.10.

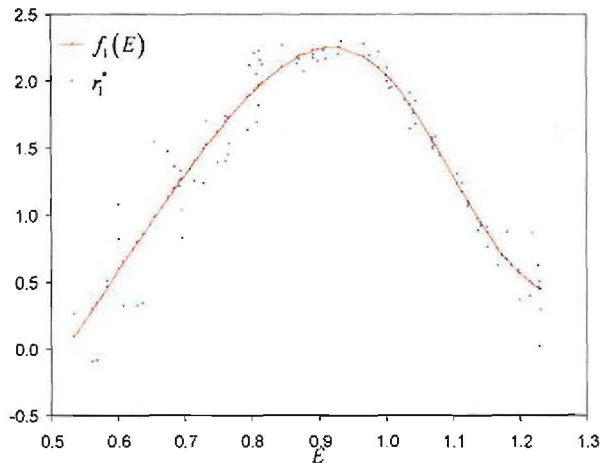


Figure 3.17: Ethanol fuel example – partial residual plot

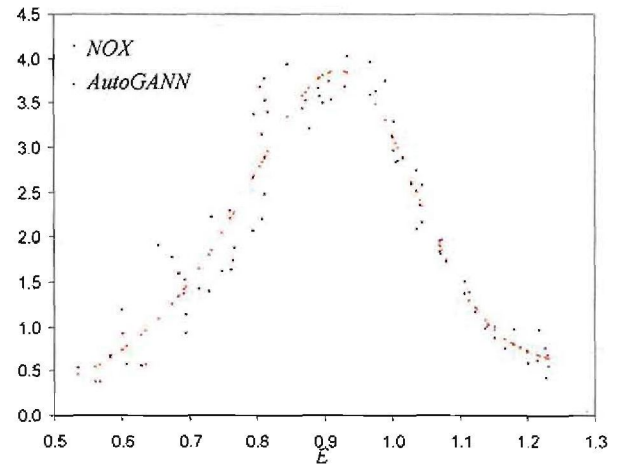


Figure 3.18: Ethanol fuel example – AutoGANN model results

3.6.3 EXPCAR example

The input variable is x and the target variable is y . The AutoGANN system found the optimal model within 39 seconds. The selected sub-architecture for input variable x is 3 – an MLP with a skip layer and a hidden node. Figure 3.19 contains the partial residual plot for the input. It depicts a nonlinear relationship with the target. Figure 3.20 contains a plot of the AutoGANN model results superimposed upon the original training input data. The model has an R^2 value of 0.96 and an average square error of 36.27.

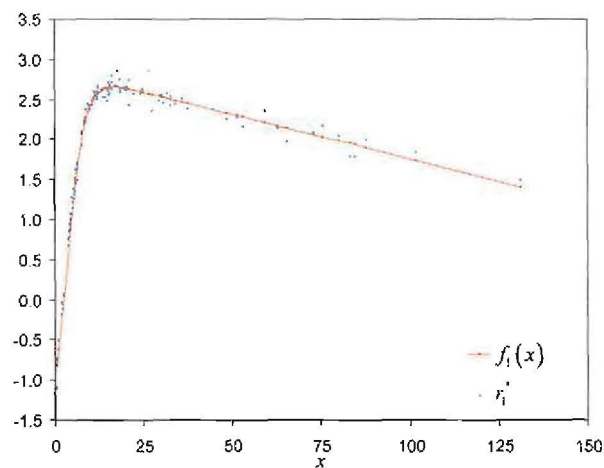


Figure 3.19: EXPCAR example – partial residual plot

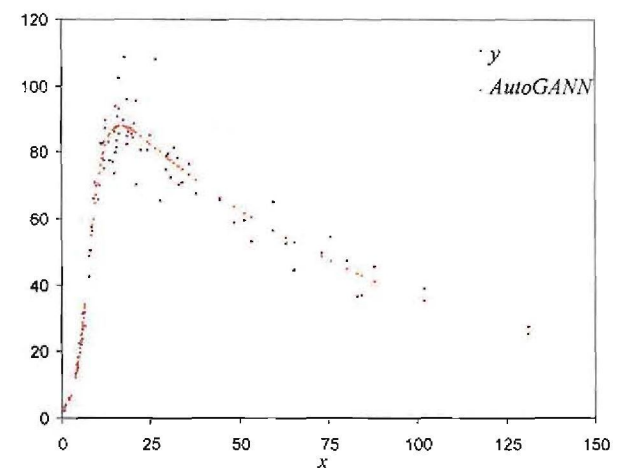


Figure 3.20: EXPCAR example – AutoGANN model results

3.6.4 SO_4 example

The SO_4 example has input variables LAT and LON and target variable SO_4 . The AutoGANN system found the optimal model within 42 seconds. The selected sub-architecture is 3 for both LAT and LON . Figure 3.21 and Figure 3.22 contain the partial residual plots for the input variables LAT and LON respectively. Both depict nonlinear trends with the target variable. The AutoGANN system selected the exponential activation function. Figure 3.23 contains a plot of the AutoGANN model results superimposed upon the original training input data plotted against $w_0 + f_1(LAT) + f_2(LON)$. The model has an R^2 value of 0.85 and an average square error of 0.18.

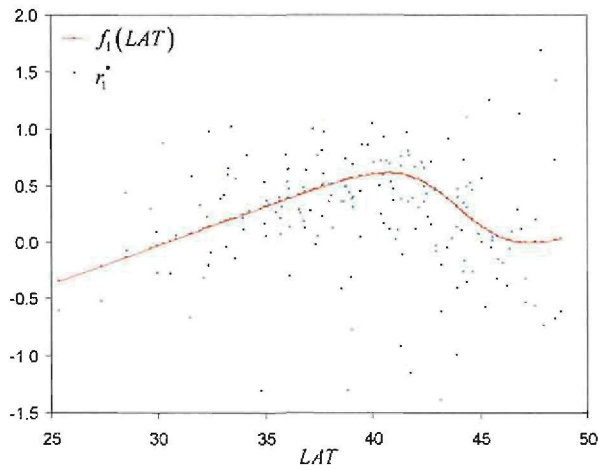


Figure 3.21: SO_4 example case 1 – partial residual plot for LAT

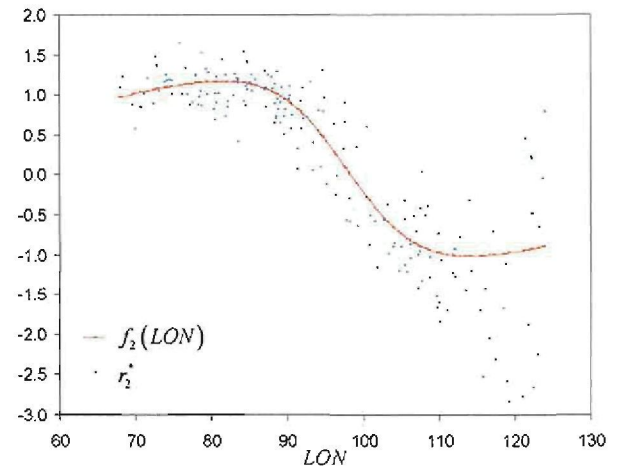


Figure 3.22: SO_4 example case 1 – partial residual plot for LON

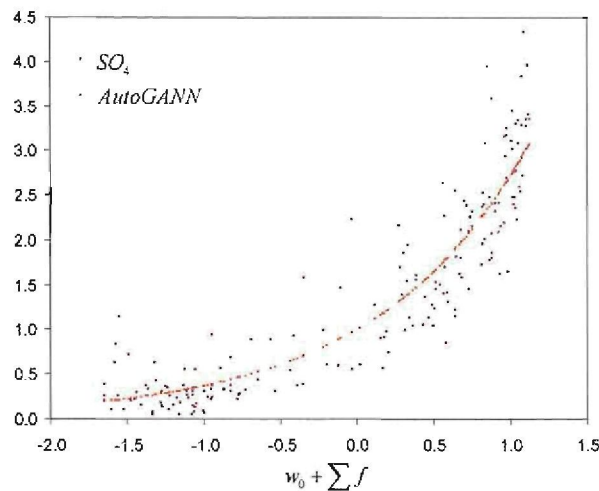


Figure 3.23: SO_4 example case 1 – AutoGANN model results

It is possible to obtain a better fit to the observed data by adding an *Interactions* node as was done in the third simulated example. The input variables LAT^2 and $LAT \times LON$ were added to the input data set. The AutoGANN system used the full time period allowed (5 minutes) to obtain a model with a linear sub-architecture for variable LAT and sub-architectures of 3 for the remaining variables. Figure 3.24 to Figure 3.27 contain the partial residual plots for the input variables. The partial residual plot for LAT depicts a linear association and the remaining plots depict nonlinear trends with the target variable. The AutoGANN system again selected the exponential activation function. Figure 3.28 contains a plot of the AutoGANN model results superimposed upon the original training input data plotted against $w_0 + \sum f_j$. The model has an R^2 value of 0.92 and an average square error of 0.09.

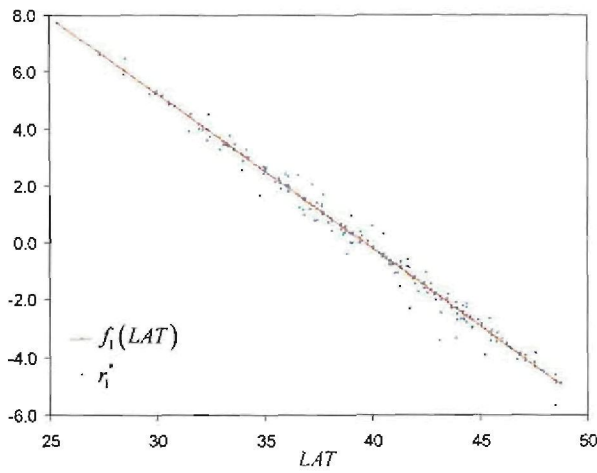


Figure 3.24: SO_4 example case 2 – partial residual plot for LAT

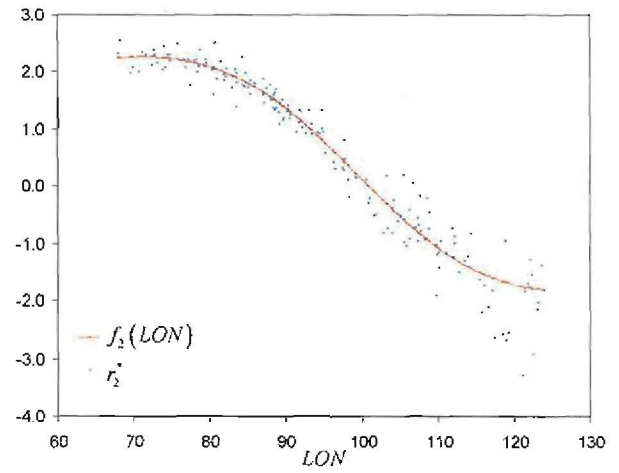


Figure 3.25: SO_4 example case 2 – partial residual plot for LON

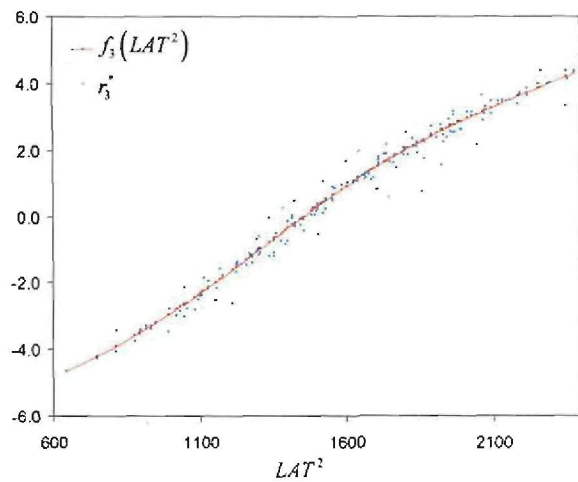


Figure 3.26: SO_4 example case 2 – partial residual plot for LAT^2

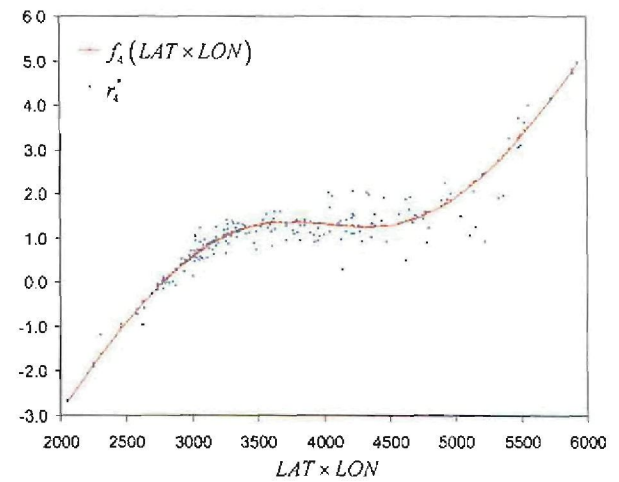


Figure 3.27: SO_4 example case2 – partial residual plot for $LAT \times LON$

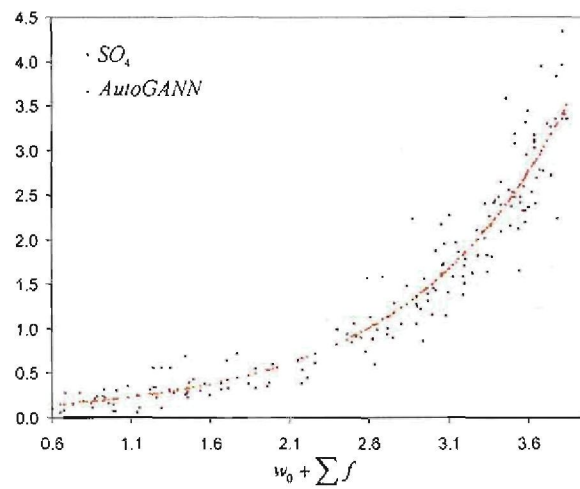


Figure 3.28: SO_4 example case 2 – AutoGANN model results

Included in Section 5.4, are 3-dimensional graphic illustrations of the model predictions on a more densely populated input space for both the cases discussed in this section.

3.6.5 Boston Housing example

The Boston Housing data set consists of the 13 input variables listed in Table 3.4. The target variable is $MEDV'$. The AutoGANN system was first applied to a problem involving all 13 input variables. Twenty minutes was allowed for this example in stead of five to compensate for the large number of input variables. The best model the AutoGANN system could find in the total time allowed has the GANN architecture depicted in Table 3.4. Variables ZN , $INDUS$, AGE and DIS' were removed from the model.

Input number	Input variable	GANN sub-architecture code	GANN sub-architecture description
1	$LSTAT'$	1	Linear
2	RM'	2	MLP, no skip layer, 1 hidden node
3	$PTRAT$	1	Linear
4	TAX	1	Linear
5	$CRIM$	1	Linear
6	ZN	0	Input removed
7	$INDUS$	0	Input removed
8	$CHAS$	1	Linear
9	NOX'	3	MLP, skip layer, 1 hidden node
10	AGE	0	Input removed
11	DIS'	0	Input removed
12	RAD'	3	MLP, skip layer, 1 hidden node
13	B'	1	Linear

Table 3.4: Boston Housing example case 1 – GANN architecture of the best model

Figure 3.29 to Figure 3.37 contain the partial residual plots for the input variables. The partial residual plots for $LSTAT'$, $PTRAT$, TAX , $CRIM$, $CHAS$ and B' clearly show linear trends. The plots for RM' , NOX' and RAD' depict nonlinear trends. The AutoGANN system selected the exponential activation function. Figure 3.38 contains a plot of the AutoGANN model results superimposed upon the original training input data plotted against $w_0 + \sum f_j$. The model has an R^2 value of 0.85 and an average square error of 0.03.

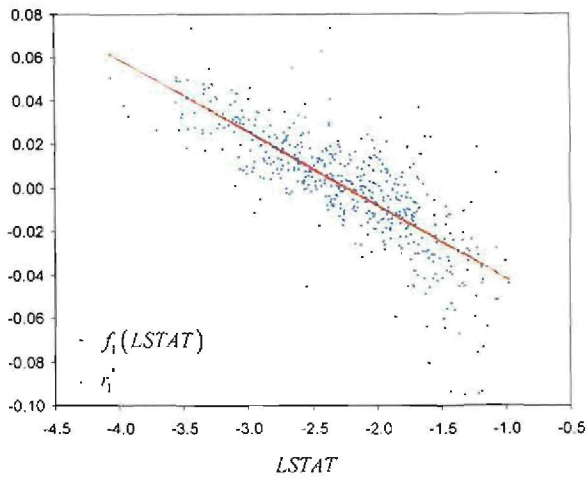


Figure 3.29: Boston Housing example case 1 – partial residual plot for $LSTAT'$

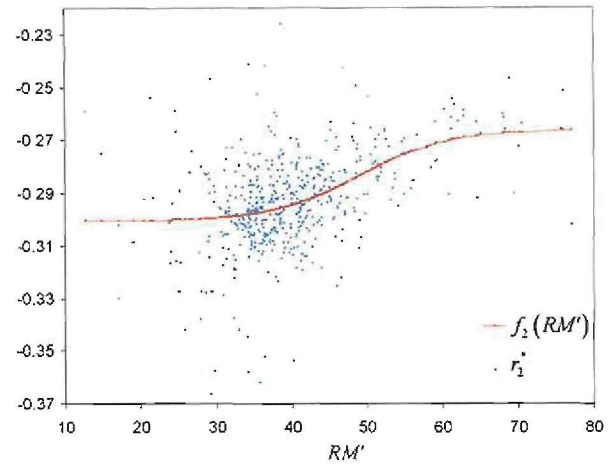


Figure 3.30: Boston Housing example case 1 – partial residual plot for RM'

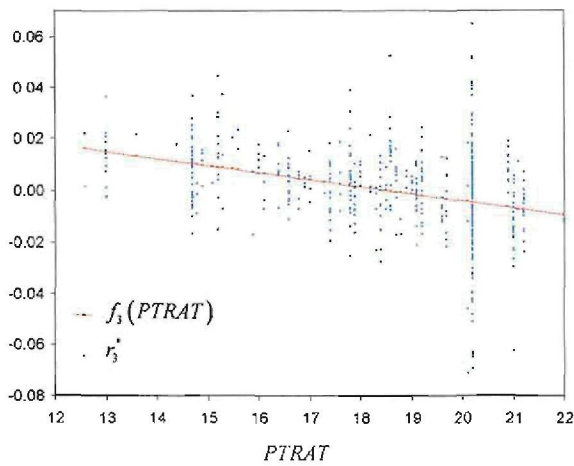


Figure 3.31: Boston Housing example case 1 – partial residual plot for $PTRAT$

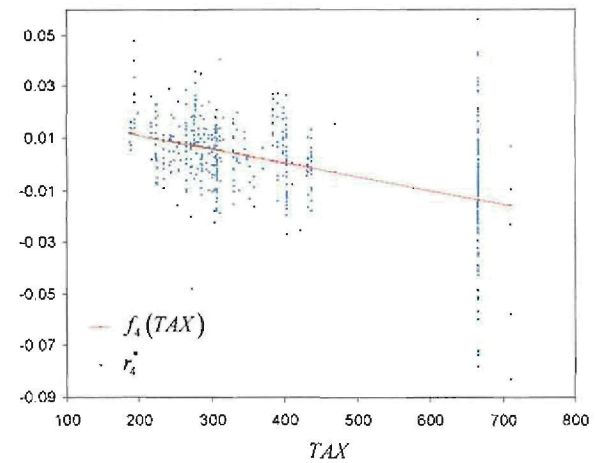


Figure 3.32: Boston Housing example case 1 – partial residual plot for TAX

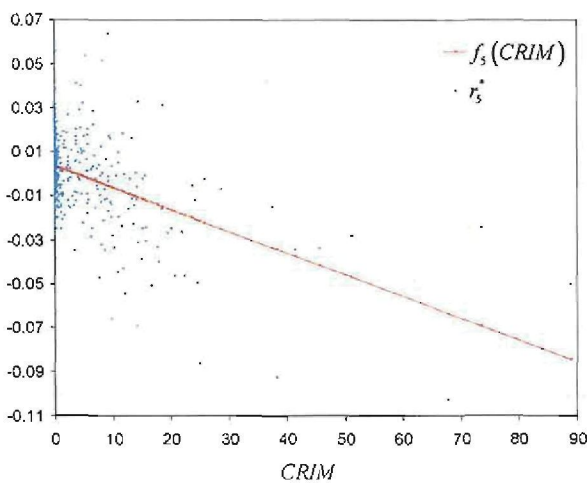


Figure 3.33: Boston Housing example case 1 – partial residual plot for $CRIM$

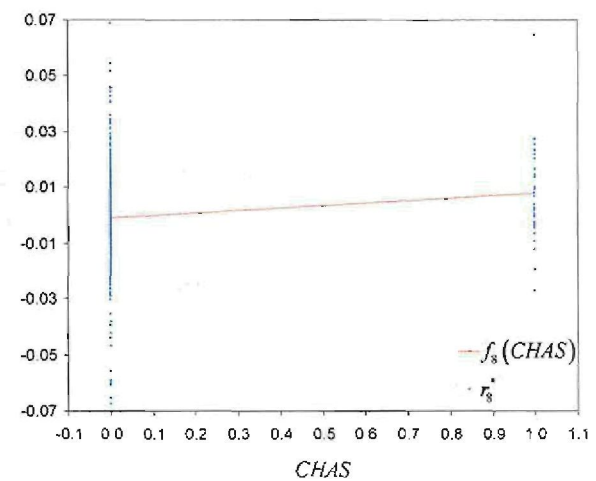


Figure 3.34: Boston Housing example case 1 – partial residual plot for $CHAS$

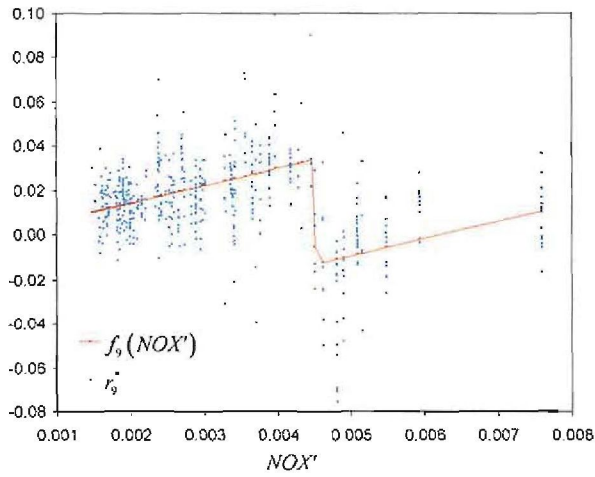


Figure 3.35: Boston Housing example case 1 – partial residual plot for NOX'

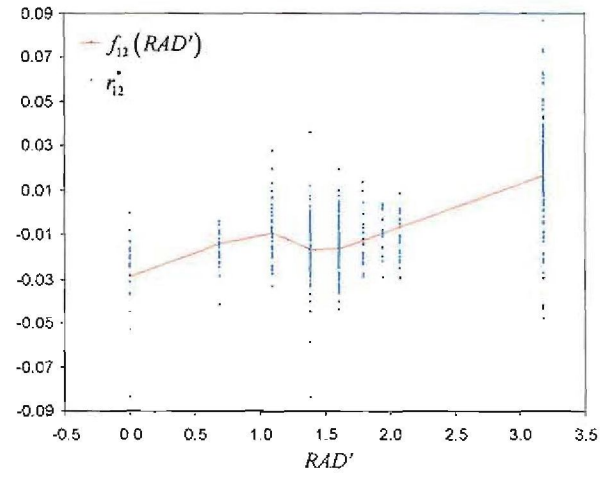


Figure 3.36: Boston Housing example case 1 – partial residual plot for RAD'

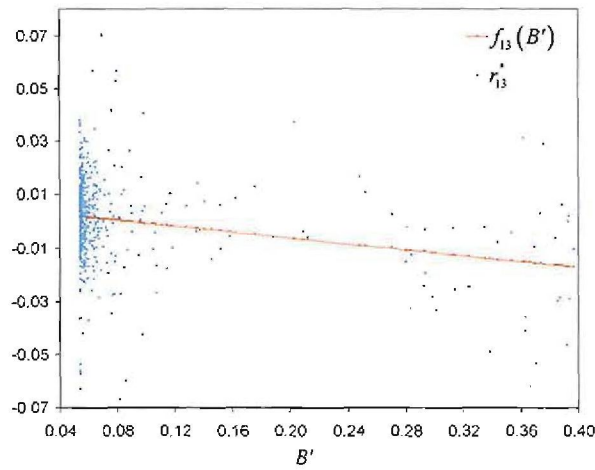


Figure 3.37: Boston Housing example case 1 – partial residual plot for B'

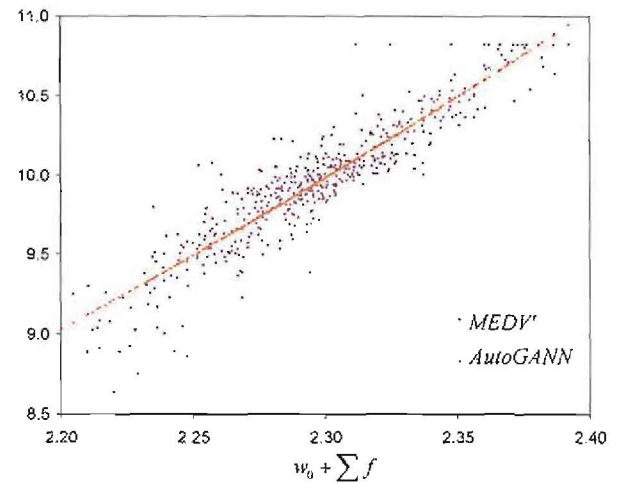


Figure 3.38: Boston Housing example case 1 – AutoGANN model results

A second case was also created in which the AutoGANN system was applied to a problem involving the four input variables used by Breiman and Friedman (1985). A total execution time of 10 minutes was allowed. The best model found by the AutoGANN system has a sub-architecture of 3 for variables $LSTAT'$ and RM' and linear sub-architectures for variables $PTRAT'$ and TAX' – the same architecture as in the first case. Figure 3.39 to Figure 3.42 show the partial residual plots for the input variables. They are very similar to those obtained in the first case. Figure 3.43 contains a plot of the AutoGANN model results superimposed upon the original training input data plotted against $w_0 + \sum f_j$. The model has an R^2 value of 0.77 and an average square error of 0.04.

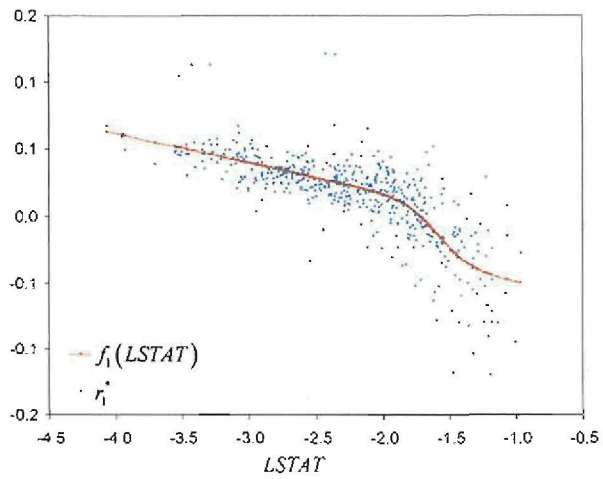


Figure 3.39: Boston Housing example case 2 – partial residual plot for $LSTAT'$

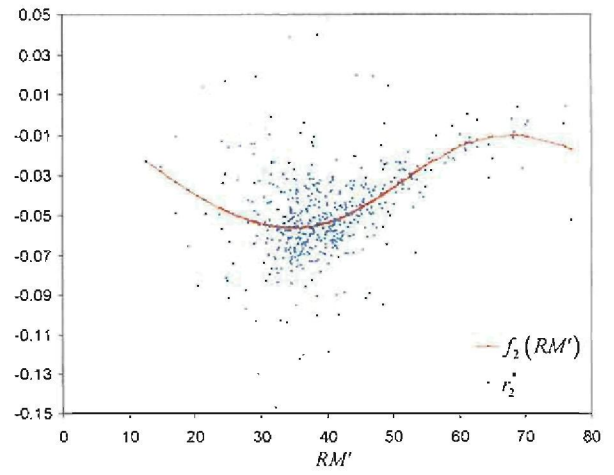


Figure 3.40: Boston Housing example case 2 – partial residual plot for RM'

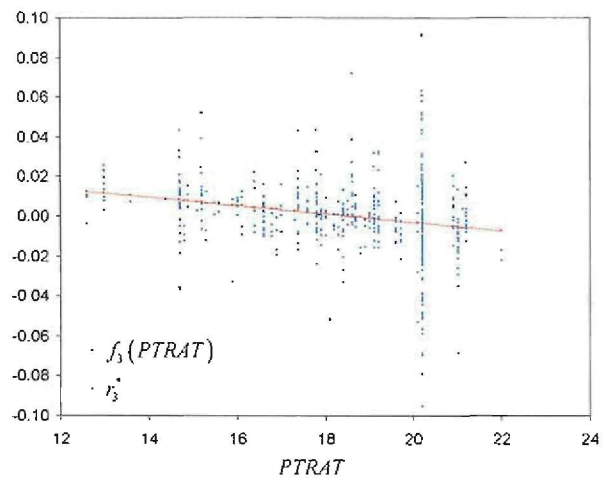


Figure 3.41: Boston Housing example case 2 – partial residual plot for $PTRAT'$

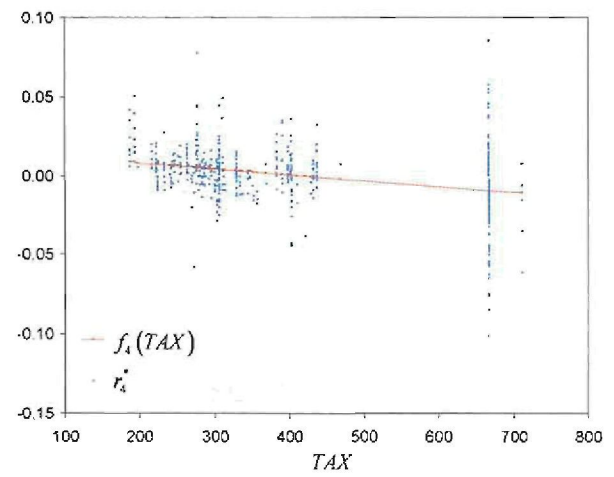


Figure 3.42: Boston Housing example case 2 – partial residual plot for TAX'

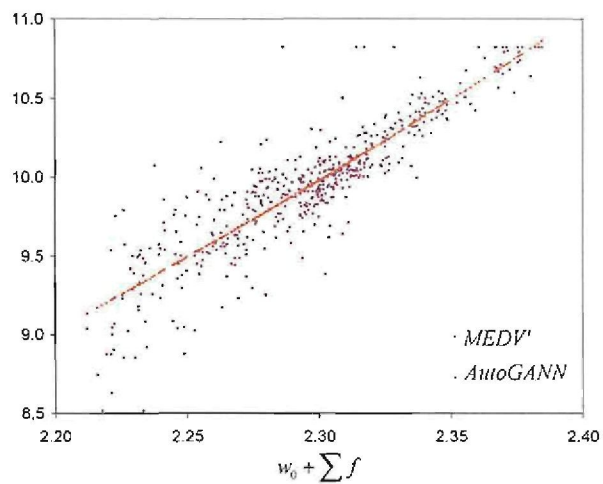


Figure 3.43: Boston Housing example case 2 – AutoGANN model results

3.6.6 Ozone example

The Ozone dataset consists of the input variables listed in Table 3.5 and a target variable, *UPO3*. The AutoGANN system was applied to a problem involving all 9 input variables. Twenty minutes was allowed for this example in stead of five to compensate for the large number of input variables. The best model the AutoGANN system could find in the total time allowed has the GANN architecture depicted in Table 3.5.

Input number	Input variable	GANN sub-architecture code	GANN sub-architecture description
1	<i>VDHT</i>	1	Linear
2	<i>WDSP</i>	0	Input removed
3	<i>HMDT</i>	2	MLP, no skip layer, 1 hidden node
4	<i>SBTP</i>	0	Input removed
5	<i>IBHT</i>	0	Input removed
6	<i>DGPG</i>	3	MLP, skip layer, 1 hidden node
7	<i>IBTP</i>	3	MLP, skip layer, 1 hidden node
8	<i>VSTY</i>	1	Linear
9	<i>DOY</i>	3	MLP, skip layer, 1 hidden node

Table 3.5: Ozone example – GANN architecture of the best model

Figure 3.44 to Figure 3.49 contain the partial residual plots for the input variables. The partial residual plots for variables *VDHT* and *VSTY* clearly show linear trends. The plots for variables *DGPG*, *IBTP* and *DOY* depict nonlinear trends, while the plot for *HMDT* depicts a near-linear trend.

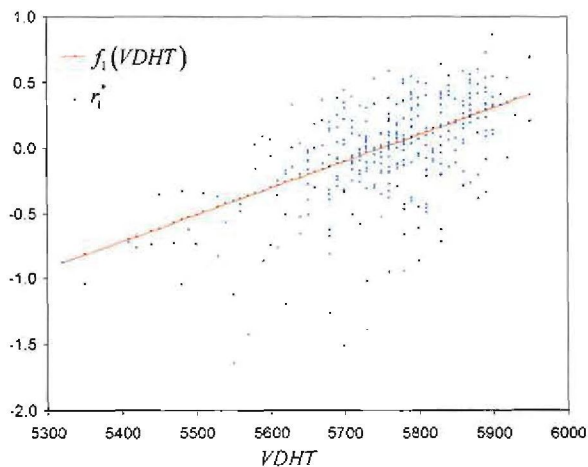


Figure 3.44: Ozone example – partial residual plot for *VDHT*

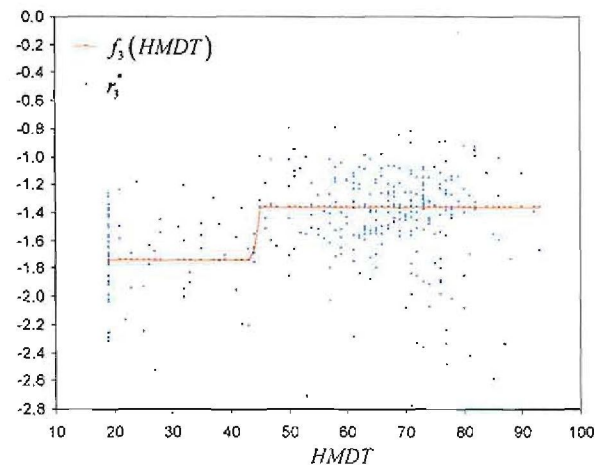


Figure 3.45: Ozone example – partial residual plot for *HMDT*

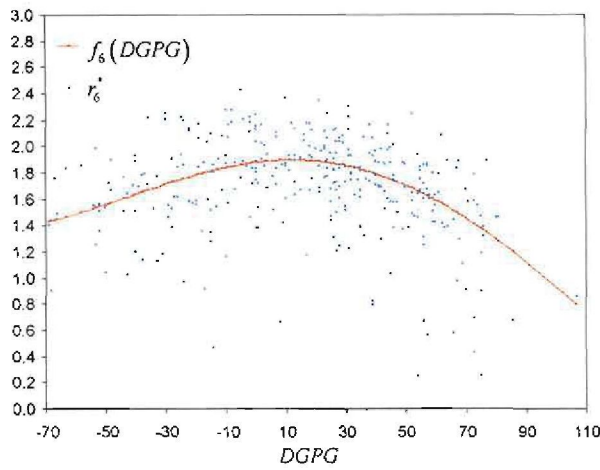


Figure 3.46: Ozone example – partial residual plot for *DGPG*

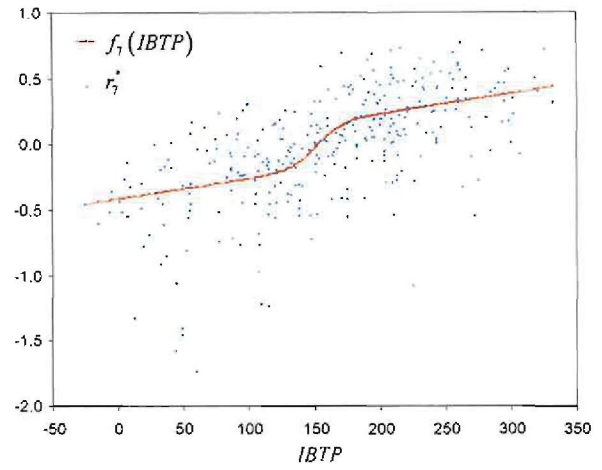


Figure 3.47: Ozone example – partial residual plot for *IBTP*

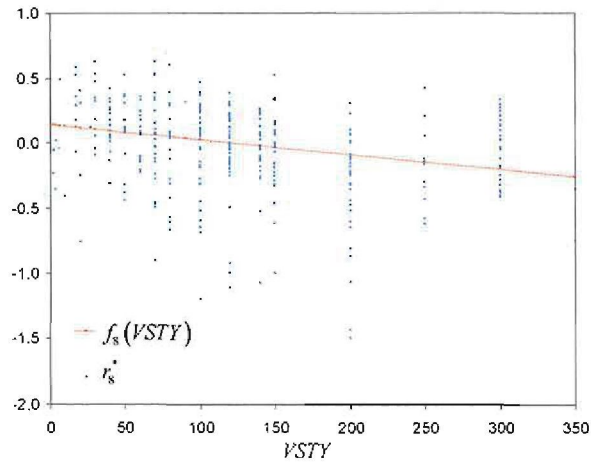


Figure 3.48: Ozone example – partial residual plot for *VSTY*

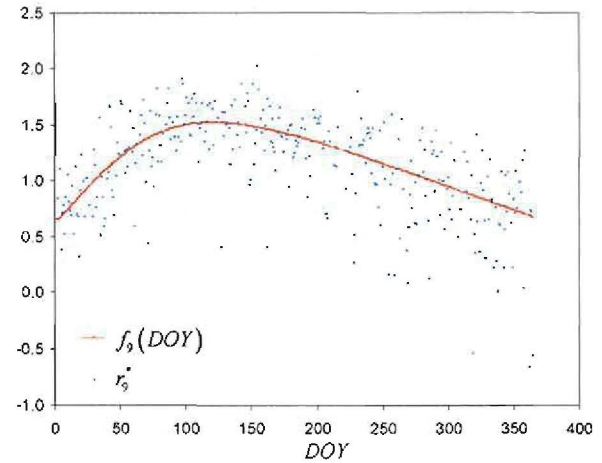


Figure 3.49: Ozone example – partial residual plot for *DOY*

The AutoGANN system selected the exponential activation function. Figure 3.50 contains a plot of the AutoGANN model results superimposed upon the original training input data plotted against $w_0 + \sum f_j$. The model has an R^2 value of 0.82 and an average square error of 11.25.

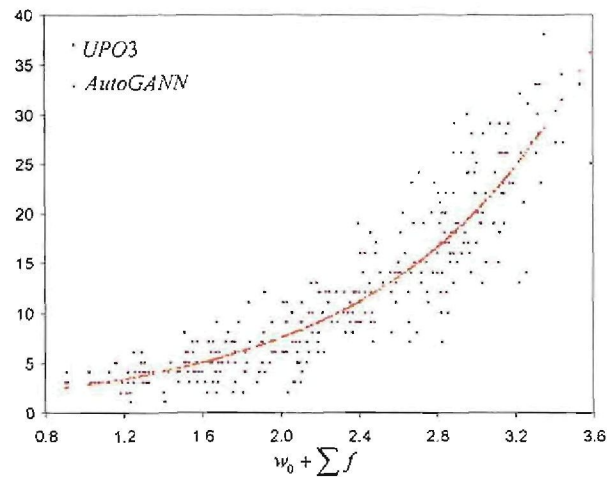


Figure 3.50: Ozone example – AutoGANN model results

3.7 Conclusions

In this chapter, the AutoGANN system was discussed and demonstrated using the regression problem examples. The method overcomes several problems associated with artificial neural networks – it automatically creates the neural network architecture, and the generalised additive model is easy to interpret using the partial residual plots.

The following chapter contains a discussion of decision trees as well as the results obtained by applying it to the regression problem examples.

4

DECISION TREES

Decision trees, also known as regression trees when applied to regression problems, are widely used as nonparametric regression methods. A regression tree is both simple to implement and simple to interpret. It divides the problem space into blocked regions with each region having a constant response value. This gives a regression tree model its characteristic stepped appearance. The main problem to solve when building a tree model is to determine these regions. It is usually chosen to minimise or maximise a regression criterion, for example the sum of the squared errors.

An overview of decision trees is given in Section 4.1. Section 4.2 contains descriptions of the different methods used to fit regression trees. In Section 4.3, the results are given of applying regression trees to the different examples used in the study.

4.1 An overview of decision trees

Decision trees owe their name to the tree-like structure obtained when representing the predictive model graphically. Figure 4.1 contains a generic representation of a decision tree model. The predictive model is a recursive series of decisions or rules used to split the problem space into partitions that contain similar response values. Each of the internal nodes describes a rule based on one of the predictor variables and splits the problem space into two or more disjoint subspaces. Following the decision path from the root node up to a leaf node for a particular observation would give an estimate of the expected response value. In the case of a regression tree, each partition represents a constant value for the expected response so that the model represents a multivariate step function. These step functions are highly flexible and capable of modelling nonlinear trends (Potts, 2001). In the case of a categorical response variable the decision tree is called a classification tree and each partition is associated with a single response class determined by the posterior probability of the response.

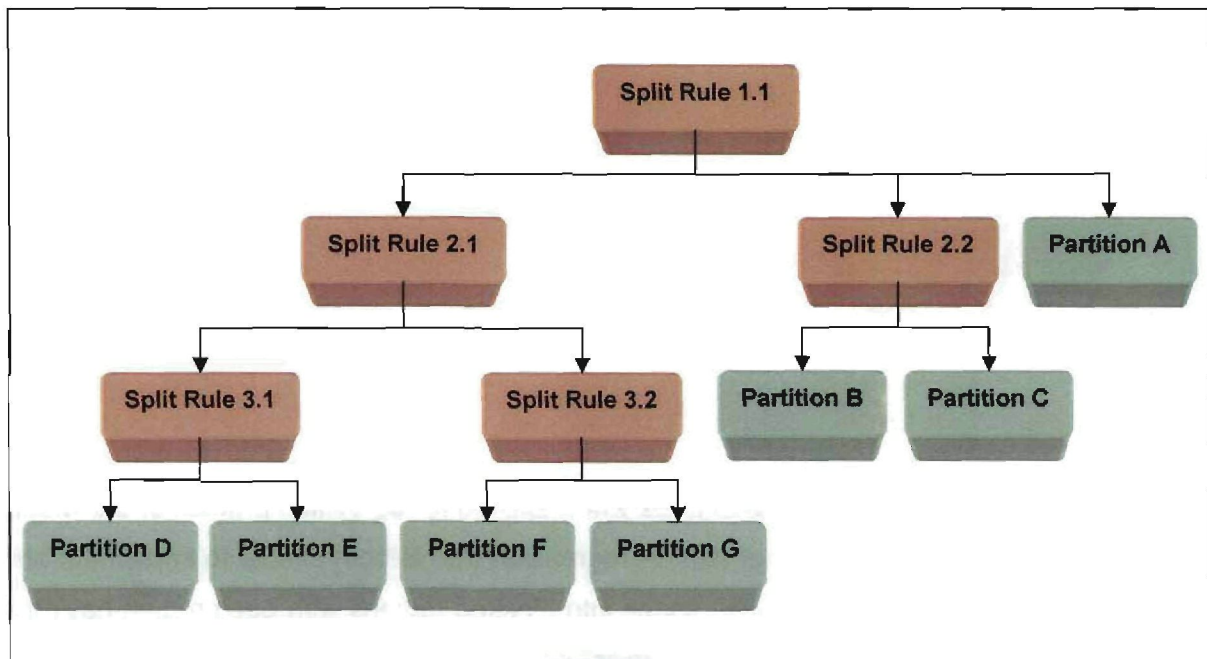


Figure 4.1: A generic decision tree structure

A decision tree model may have any number of nodes and levels. Each decision node may have any number of child nodes. Fitting a decision tree to the data is also referred to as the cultivation of the tree (Potts, 2002) and is discussed in more detail in the following section.

4.2 The cultivation of decision trees

Various algorithms have been developed for the cultivation of decision trees. Commonly known algorithms include those of Morgan and Sonquist (1963) called AID (automatic interaction detection), Kass (1980) called CHAID (Chi-squared automatic interaction detection), Breiman and Friedman (1984) called CART (classification and regression trees), Quinlan (1993) called C4.5 and its follower, C5 and Loh and Shih (1997) called QUEST (quick, unbiased, efficient statistical tree). The cultivation of decision trees involves several factors that include the split search, the splitting criterion, stopping rules and pruning rules. This section describes the cultivation methods of regression trees used by algorithms from SAS®. These algorithms incorporate and extend most of the functionality accepted as good from the algorithms mentioned above (Neville, 1999).

4.2.1 Split search

Each split rule (or decision node) in a decision tree involves a single predictor variable and splits the problem space into two or more subspaces. When cultivating decision trees, the split search

is used to determine the split rule possibilities to be considered at each decision node in the tree. Each possibility describes both the predictor variable on which to base the split and the way the split will be performed. For example, one possibility might be to use predictor variable X_1 and split the data into three branches where $X_1 < a$, $a \leq X_1 \leq b$ and $b < X_1$. Split possibilities for a continuous predictor variable are determined by sorting the n observation cases and considering each case value as a possible cut-off point on the scale. Because of the extremely large number of possibilities that exist for a number of interval predictor variables, p , a split search usually restricts the possibilities. The most common way is to restrict the number of branches created by a split rule to a low number and only consider binary splits, for example.

4.2.2 Splitting criteria

A splitting criterion is used to determine the best split rule to use from the set of candidate splitting rules created by the split search. A two-step process is followed (Potts, 2001): the best split for each predictor variable is determined first and then the best split is chosen from the set of best splits determined in the first step. Since a regression tree creates partitions where the response values are alike, suitable splitting criteria include variance reduction and the P -value of the F test associated with the partition variance. For a response variable, Y , and a specific node, t , the (maximum likelihood) sample variance estimate would be

$$\sigma^2(t) = \frac{1}{n_t} \sum_{j=1}^{n_t} (Y_{tj} - \bar{Y}_t)^2 \quad (4.1)$$

where n_t is the number of cases in node t and $\bar{Y}_t$ is the mean of the response. Let θ denote the decision node under consideration. The best split for node θ when using variance reduction would minimise

$$\Delta\sigma^2 = \sigma^2(\theta) - \sum_{k=1}^B \frac{n_k}{n_\theta} \sigma^2(k) \quad (4.2)$$

where B is the number of branches created by the split and n_k is the number of cases in child node k .

Variance reduction does not take into account the number of branches created by the split (the degrees of freedom) and should only be used with a restricted branching factor. The F test, however, may be used to include the branching factor:

$$F^* = \left(\frac{SS_{between}}{SS_{within}} \right) \left(\frac{n-B}{B-1} \right) \quad (4.3)$$

where

$$SS_{between} = \sum_{k=1}^B n_k (\bar{Y}_0 - \bar{Y}_k)^2 \quad (4.4)$$

is a measure of distance between the mean of node 0 and the mean of its child nodes and

$$SS_{within} = \sum_{k=1}^B \sum_{j=1}^{n_k} (Y_{kj} - \bar{Y}_k)^2 \quad (4.5)$$

is a measure of variability within the child nodes. The F test is used in the same manner as with analysis of variance (ANOVA) models. A split of the data based on a single predictor variable into B branches is analogous to single factor studies with B treatments. Values of F^* close to one support the null hypothesis that the mean response values in the child nodes are equal and that F^* follows the F distribution, $F(B-1, n-B)$. From larger values follows the conclusion that the response means are not equal and that F^* follows the noncentral F distribution (Kutner *et al.*, 2005). The P -value of the F test is the probability that the F statistic has a value exceeding the one observed: $P\{F(B-1, n-B) > F^*\}$. Large values of F^* yield small P -values so that split possibilities with smaller P -values are favoured. It may be more convenient to use the logworth of the P -value, $-\log_{10}(P\text{-value})$, when the P -values are very small, in which case larger logworth values are favoured. Using the P -value ensures unbiased comparisons of split possibilities when selecting the best split option in terms of the branching factor.

Potts (2001) points out that the F test is relatively robust to departures from the assumption of normality, but that it does not perform well when heteroscedasticity is present. In this case it favours smaller partitions where the predictor variable values are larger and have larger variance.

4.2.3 Stopping and pruning rules

A stopping rule or pruning rule is required to keep the tree from partitioning the data into too many partitions and by that means causing overfitting. These rules essentially control the bias-variance trade-off, where harsh rules would in turn cause high bias and low variance. A complex tree model has a high number of leaves, splits and levels. Selecting the tree complexity can be done by means of pre-pruning or post-pruning. Pre-pruning uses a forward stopping rule to stunt the growth of a tree. Post-pruning involves growing a large tree and pruning back the branches.

Various limits may be employed for a stopping rule. The depth of the tree may be limited to a certain number of levels. Another method may be to limit the amount of fragmentation – no node may contain a number of cases less than some threshold (which is usually expressed as a

percentage of the total number of cases). Statistical significance is also used where the P -values of the F test are not allowed to become less than some predefined significance level, α (maximum acceptable P -value). A depth adjustment may also be applied to the P -value where the P -value is multiplied by $\prod_{i=0}^d B_i$. The value of d is the level number of the node being tested and B_i is the number of branches created by the split at each parent node in the path from the root node to the current node. This depth adjustment causes the P -value to become increasingly smaller deeper in the tree, thereby limiting the growth of the tree.

For post-pruning, a maximal tree is grown after which a sequence of subtrees is created, one for each possible number of leaves. All the subtrees are evaluated using a performance measure. The smallest subtree with the best performance measure is selected. Performance may be measured against a validation data set or by using k -fold cross-validation. For regression trees the performance measure is the average square error.

According to Potts (2001), pre-pruning is faster but less effective than growing a maximal tree and then applying post-pruning. A combination of pre-pruning and post-pruning could be used for determining the right-sized tree and may be implemented using a liberal stopping rule for growing a large tree and then performing post-pruning.

The following section contains the results of applying regression trees to the different examples used in the study.

4.3 Decision tree examples

The Decision Tree model node in SAS® Enterprise Miner™ was applied to all the example cases described in Section 1.3. In each case all the variables were specified to be continuous (interval variables) and the default user settings for the Decision Tree model node were used. The most influential settings are:

- Splitting criterion = PropF, the P -value of the F test;
- Significance level = 0.2, the maximum acceptable P -value;
- Leaf size = 5, the minimum number of cases a leaf node is allowed to have;
- Maximum branches = 2, only binary splits are allowed;
- Maximum depth = 6, the maximum number of levels in the tree, counted from the root node;
- Subtree = Assessment, the smallest subtree with the best assessment value is selected when performing post-pruning;

- Variable selection = Yes, all variables with an importance value less than 0.05 are removed from the model. Variable importance is determined by summing the split worth values of each variable across the tree model and scaling it from zero to one;
- Assessment measure = Average Square Error;
- P -value split adjustment = Yes, a depth adjustment is applied to the P -value.

4.3.1 Simulated examples

First simulated example

A set of 200 data points was generated using $y = \exp(x^3 + \varepsilon)$ with x^3 and ε drawn randomly from standard normal distributions. The resulting tree model has a depth of 3 with 3 split rules based on x . Figure 4.2 contains a plot of the decision tree model results superimposed upon the original training input data. The model has an R^2 value of 0.53 and an average square error of 9.58.

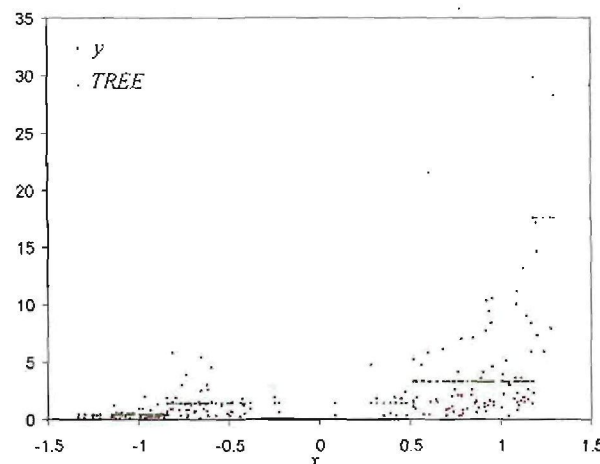


Figure 4.2: First simulated example – decision tree model results

Second simulated example

A set of 200 data points was generated using $y = \exp[\sin(2\pi x) + \varepsilon / 2]$ with x drawn from a uniform distribution between 0 and 1, and ε drawn independently from a standard normal distribution. The resulting tree model has a depth of 4 with 5 split rules based on x . Figure 4.3 contains a plot of the decision tree model results superimposed upon the original training input data. The model has an R^2 value of 0.53 and an average square error of 0.94.

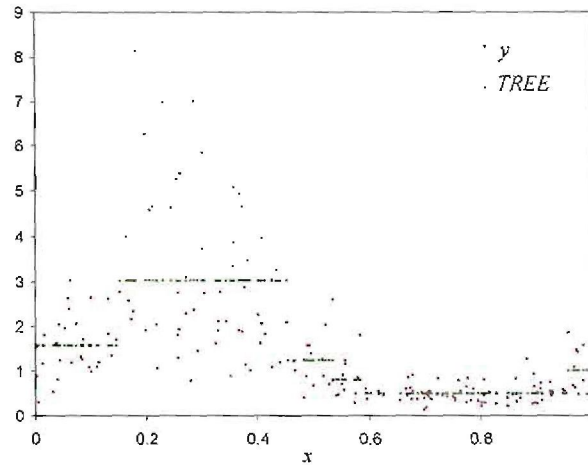


Figure 4.3: Second simulated example – decision tree model results

Third simulated example

A set of 200 data points, $\{(y_i, s_i, t_i), 1 \leq i \leq 200\}$, was generated from $y = st$ where s and t are two independent variables with uniform distributions between -1 and 1. The resulting tree model has a depth of 6 with 12 split rules based on s and 9 rules based on t . Figure 4.4 contains a plot of the decision tree model results superimposed upon st . The model has an R^2 value of 0.95 and an average square error of 4.99×10^{-3} . The decision tree model is not able to fit a simple linear relationship well – the step function produces an unnecessary complex model. Included in Section 5.1.3 is a 3-dimensional graphic illustration of the model predictions on a more densely populated input space.

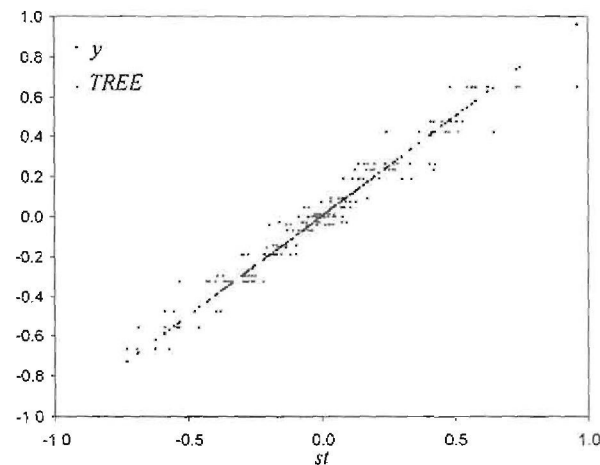


Figure 4.4: Third simulated example – decision tree model results

4.3.2 Ethanol fuel example

In this problem the predictor variable is E and the response variable is NOX . The resulting tree model has a depth of 4 with 8 split rules based on E . Figure 4.5 contains a plot of the decision tree model results superimposed upon the original training input data. The model has an R^2 value of 0.93 and an average square error of 0.09.

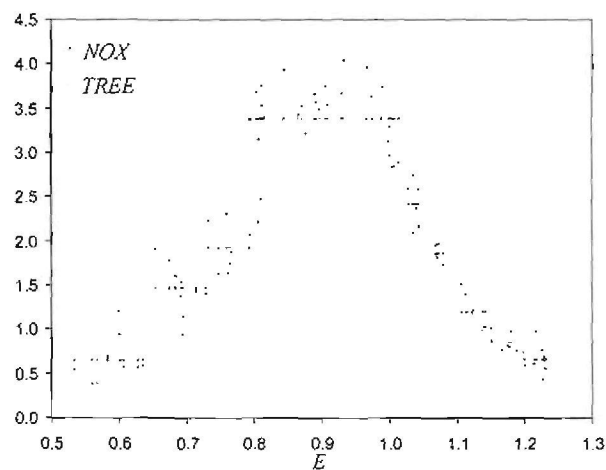


Figure 4.5: Ethanol fuel example – decision tree model results

4.3.3 EXPCAR example

The predictor variable is x and the response variable is y . The resulting tree model has a depth of 4 with 8 split rules based on x . Figure 4.6 contains a plot of the decision tree model results superimposed upon the original training input data. The model has a R^2 value of 0.95 and average square error of 40.45.

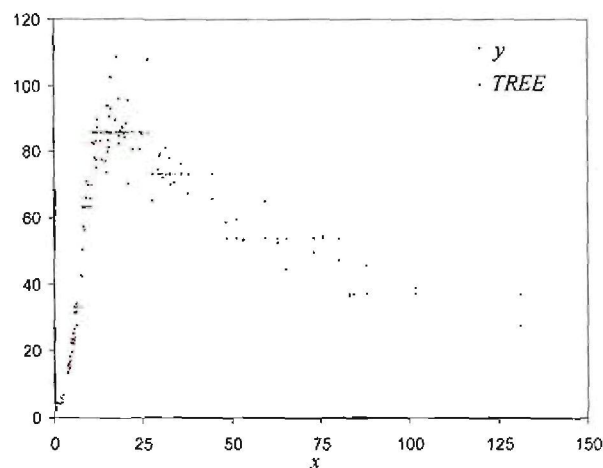


Figure 4.6: EXPCAR example – decision tree model results

4.3.4 SO_4 example

The SO_4 example has predictor variables LAT and LON and response variable SO_4 . The resulting tree model has a depth of 5 with 6 split rules based on LAT and 3 rules based on LON . The model has an R^2 value of 0.89 and an average square error of 0.13. Included in Section 5.4 is a 3-dimensional graphic illustration of the model predictions on a more densely populated input space.

4.3.5 Boston Housing example

The Boston Housing data set consists of the 13 predictor variables described in Section 1.3.5. The response variable is $MEDV'$. The Decision Tree model node was first applied to a problem involving all 13 predictor variables. The resulting tree model has a depth of 6, which is the maximum number allowed according to the stopping rule. Table 4.1 lists the variables that were selected by the decision tree in order of importance together with the number of split rules based on each variable. Variables ZN , $INDUS$, $CHAS$ and B' were not used in the tree model. The model has an R^2 value of 0.89 and an average square error of 0.02.

<i>Input variable</i>	<i>Number of split rules</i>
$LSTAT'$	6
RM'	4
$CRIM$	2
NOX'	3
RAD'	1
$PTRAT$	2
DIS'	2
TAX	2
AGE	1

Table 4.1: Boston Housing example case 1 – decision tree variables and number of split rules

A second case was also created in which the Decision Tree model node was applied to the four predictor variables used by Breiman and Friedman (1985), namely $LSTAT'$, RM' , $PTRAT$ and TAX . The resulting tree model has a depth of 6. All four variables were used by the tree model and Table 4.2 lists the variables in order of importance together with the number of split rules based on each. The model has an R^2 value of 0.83 and an average square error of 0.03.

<i>Input variable</i>	<i>Number of split rules</i>
<i>LSTAT'</i>	5
<i>RM'</i>	5
<i>TAX</i>	4
<i>PTRAT</i>	3

Table 4.2: Boston Housing example case 2 – decision tree variables and number of split rules

4.3.6 Ozone example

The Ozone data set consists of 9 predictor variables and a response variable, *UPO3*. The Decision Tree model node was applied to a problem involving all the predictor variables. The resulting tree model has a depth of 5. Table 4.3 lists the variables that were selected by the decision tree in order of importance together with the number of split rules based on each variable. Variables *VSTY*, *VDHT* and *WDSP* were not used in the tree model. The model has an R^2 value of 0.79 and an average square error of 13.70.

<i>Input variable</i>	<i>Number of split rules</i>
<i>SBTP</i>	3
<i>IBTP</i>	2
<i>IBHT</i>	2
<i>DOY</i>	2
<i>DGPG</i>	1
<i>HMDT</i>	2

Table 4.3: Ozone example – decision tree variables and number of split rules

4.4 Conclusions

In this chapter, decision trees were discussed and demonstrated using the regression problem examples. Decision trees are very easy to implement and produce models that are easy to interpret.

In the following chapter, the results obtained by applying ACE, the AutoGANN system and decision trees to all the prediction problem examples are presented together and discussed.

5

PREDICTION PROBLEM EXAMPLES

The set of examples described in Chapter 1 was used to compare the predictive power and characteristics of the models created by ACE, the AutoGANN system and decision trees. Details of the results for each of these methods are given in Chapters 2, 3 and 4 respectively. In this chapter the results are compared.

5.1 Simulated examples

5.1.1 First simulated example

A set of 200 data points was generated using $y = \exp(x^3 + \varepsilon)$ with x^3 and ε drawn randomly from standard normal distributions. A score data set of 601 data points was also created using $y_i = \exp(x_i^3)$ and $x_{i+1} = x_i + 0.005$ for $i = 1, \dots, 601$ with $x_1 = -1.5$. After estimation of the models, the models were applied to the score data set. Figure 5.1 contains a plot of the ACE variable transformation for x superimposed on the known optimal transformation, x^3 . Figure 5.2 contains a plot of the AutoGANN univariate function. Since the univariate function assumes values on a different scale, it is shown on a separate plot. Both the transformations have a form similar to the form of the optimal transformation and an analyst could derive the optimal transformation from these plots. The transformation created by the AutoGANN system assumes a much smoother form than the one created by ACE because of the sigmoidal quality of the GANN activation function.

Figure 5.3 contains plots of the prediction results of all three models (on the score data set) together with the response variable, y , from the training and score data sets. The AutoGANN predictions are very close to the score data response, y .

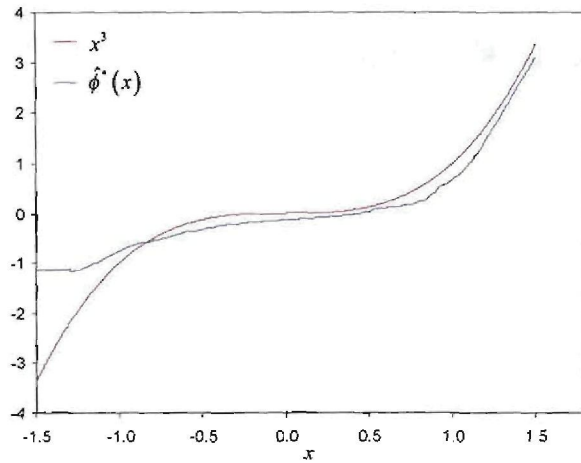


Figure 5.1: First simulated example – ACE variable transformation

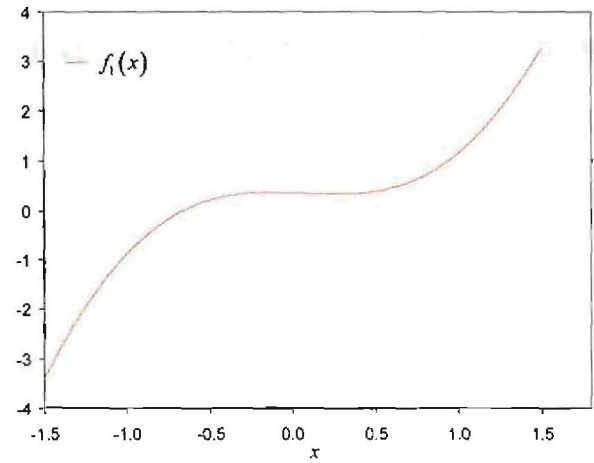


Figure 5.2: First simulated example – AutoGANN univariate function

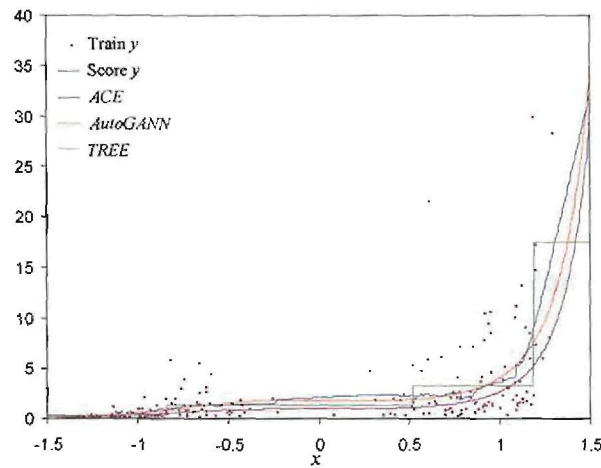


Figure 5.3: First simulated example – prediction results

Table 5.1 shows the values of the coefficient of determination (R^2) and the average squared error (ASE) for each model on the training and score data sets. Although the AutoGANN model fitted the training data worst, it outperformed ACE and the decision tree on the score data set.

Model	Train R^2	Score R^2	Train ASE	Score ASE
ACE	0.50	0.67	10.18	6.46
AutoGANN	0.45	0.90	11.12	2.02
TREE	0.53	0.63	9.58	7.19

Table 5.1: First simulated example – train and score R^2 and ASE

5.1.2 Second simulated example

A set of 200 data points was generated using $y = \exp[\sin(2\pi x) + \varepsilon/2]$ with x drawn from a uniform distribution between 0 and 1, and ε drawn independently from a standard normal distribution. A score data set of 501 data points was also created using $y_i = \exp(2\pi x_i)$ and $x_{i+1} = x_i + 0.002$ for $i = 1, \dots, 501$ with $x_1 = 0$. The estimated models were applied to the score data set. Figure 5.4 contains a plot of the ACE variable transformation for x superimposed on the known optimal transformation. Figure 5.5 contains a plot of the AutoGANN univariate function. Both the transformations have the same form as the optimal transformation.

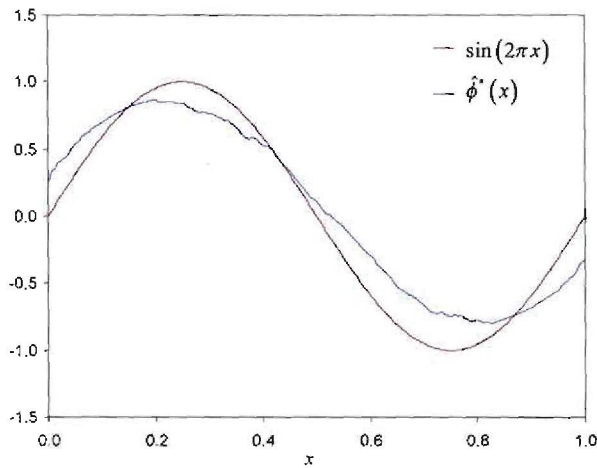


Figure 5.4: Second simulated example – ACE variable transformation

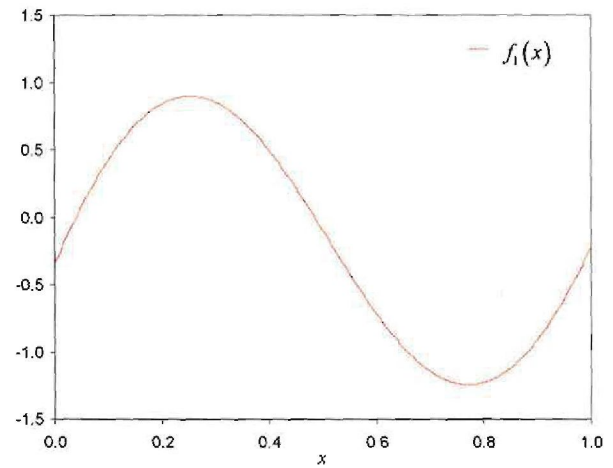


Figure 5.5: Second simulated example – AutoGANN univariate function

Figure 5.6 contains plots of the prediction results of all three models together with the response variable, y , from the training and score data sets. Once again the AutoGANN prediction results show close resemblance to that of the score data response values with a smooth curve.

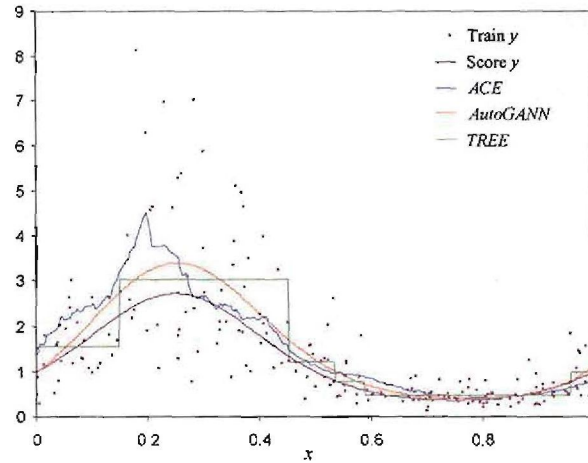


Figure 5.6: Second simulated example – prediction results

Table 5.2 shows the values of R^2 and ASE for each model on the training and score data sets. The AutoGANN model outperformed ACE and the decision tree on both the training and score data sets.

<i>Model</i>	<i>Train R^2</i>	<i>Score R^2</i>	<i>Train ASE</i>	<i>Score ASE</i>
<i>ACE</i>	0.53	0.61	0.94	0.27
<i>AutoGANN</i>	0.54	0.82	0.92	0.12
<i>TREE</i>	0.53	0.64	0.94	0.24

Table 5.2: Second simulated example – train and score R^2 and ASE

5.1.3 Third simulated example

A set of 200 data points, $\{(y_i, s_i, t_i), 1 \leq i \leq 200\}$, was generated from $y = st$ where s and t are two independent variables with uniform distributions between -1 and 1. A score data set of 10201 data points was also created by first defining $x_{j+1} = x_j + 0.02$ for $j = 1, \dots, 101$ with $x_1 = -1$ and then $y_i = x_j x_k$ for $i = 1, \dots, 10201$ with $k = ((i - j)/101) + 1$.

The estimated models were applied to the score dataset. The models $AutoGANN_1$, $AutoGANN_2$ and $AutoGANN_3$ correspond to cases 1, 2 and 3 described in Section 3.6.1. Figure 5.7, Figure 5.8 and Figure 5.9 contain surface plots of the prediction results obtained from these models. The response values from the score data set are represented by the mesh surface in each of the figures.

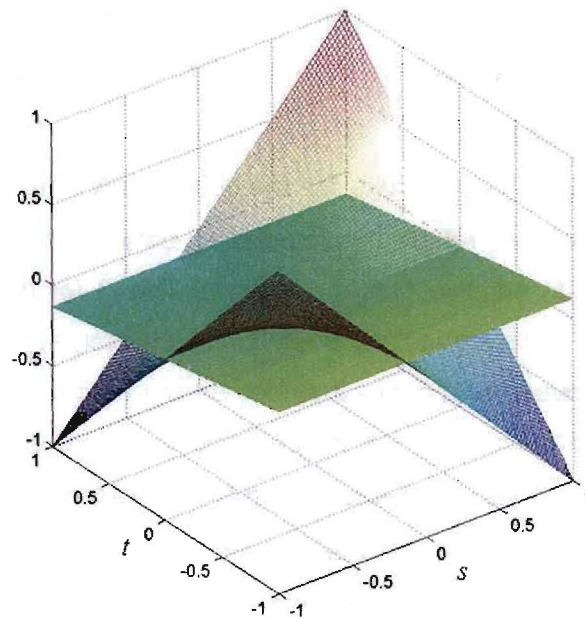


Figure 5.7: Third simulated example – $AutoGANN_1$ prediction results

Recall that, in the first case, the AutoGANN model was not able to model the interaction between s_i and t_i ; hence the flat surface in Figure 5.7. For the second case an *Interactions* node was added before applying the AutoGANN model node. Figure 5.8 depicts a much better fitted model for this case, but since the hyperbolic tangent activation function was selected for the GANN output layer, an unnecessary component of nonlinearity was added causing the flared endpoints observable in the prediction surface.

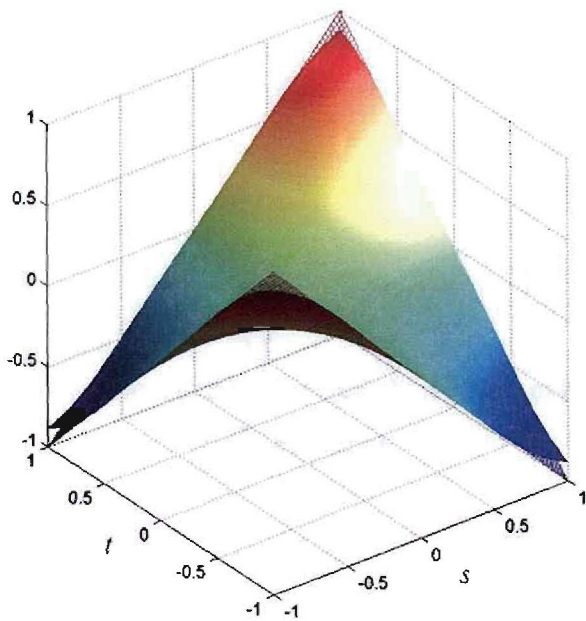


Figure 5.8: Third simulated example – $AutoGANN_2$ prediction results

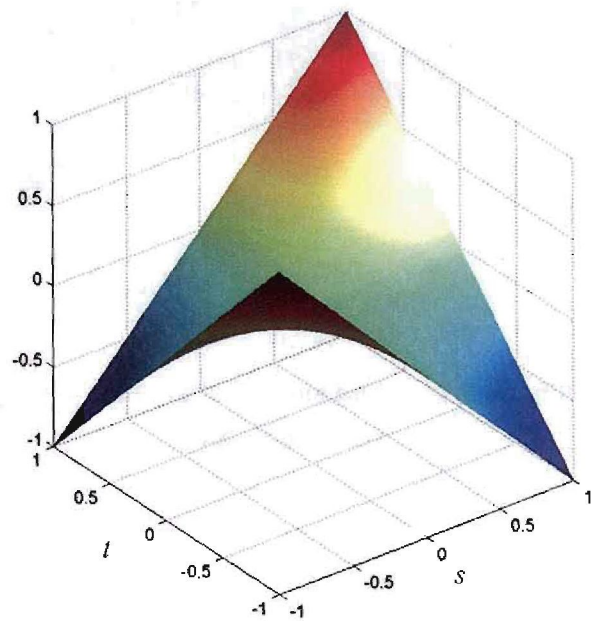


Figure 5.9: Third simulated example – $AutoGANN_3$ prediction results

For the third case the training data set was slightly altered to contain one response case outside the range $[-1,1]$. The identity activation function was selected, producing an optimal model that is evident from Figure 5.9.

Figure 5.10 shows the prediction results for the ACE model, ACE_1 . The ACE procedure cannot produce a model that will reflect the interaction between the two predictor variables correctly, since it assumes an additive problem domain. The fit for the south-east and south-west corners in Figure 5.10 were particularly poor. The prediction surface owes its form to the way the ACE procedure derives the model - it smooths the response values according to the sum of the transformed predictor variables as depicted in Figure 5.11. A second case was created with an additional input variable representing the interaction between variables s and t . Figure 5.12 shows the prediction results for the second ACE model, ACE_2 . This model fitted the data much better, but since it is not defined outside the training problem space it results in poor predictions at the endpoints of the score data surface.

Figure 5.13 shows the prediction results for the decision tree model. Although it depicts a complex step function, it fitted the data quite well.

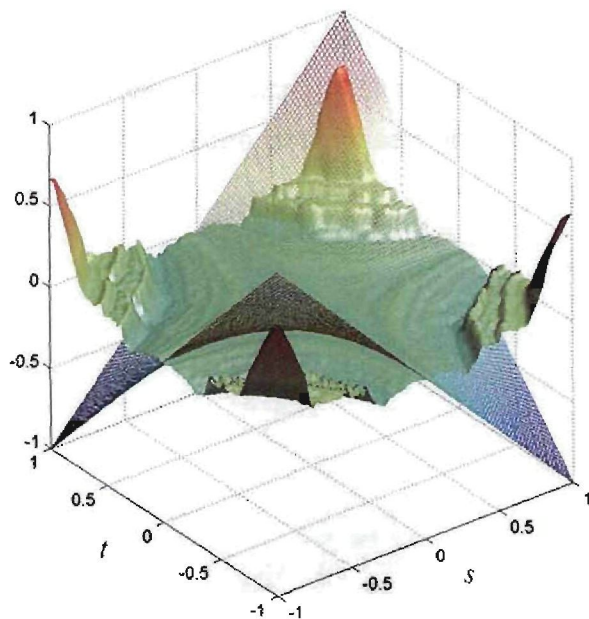


Figure 5.10: Third simulated example – ACE_1 prediction results

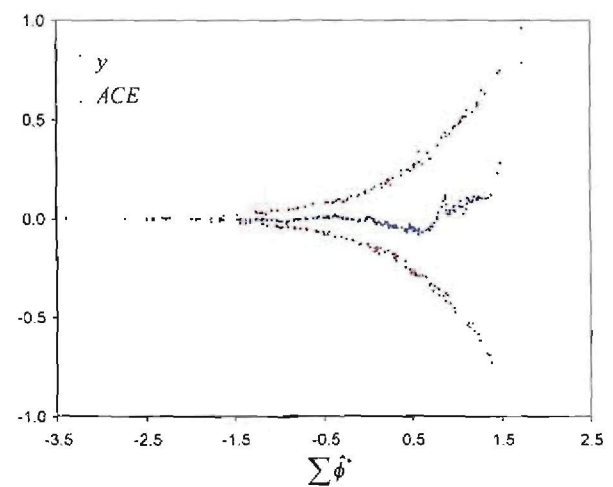


Figure 5.11: Third simulated example – ACE_1 model vs. $\sum \hat{\phi}$

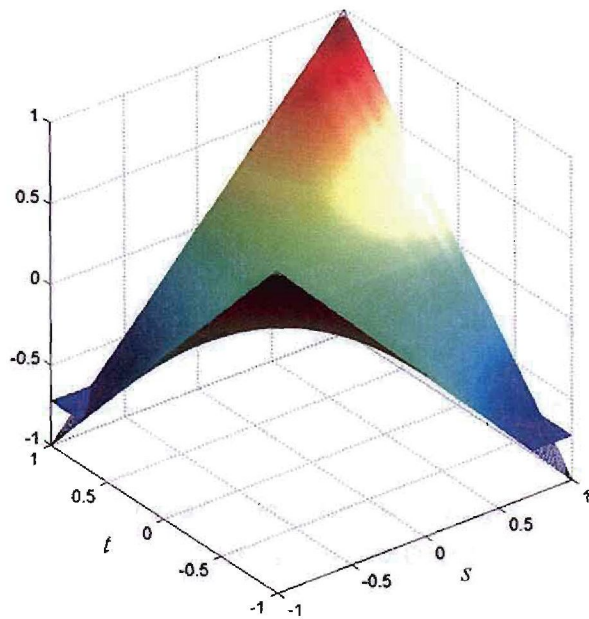


Figure 5.12: Third simulated example – ACE_2 prediction results

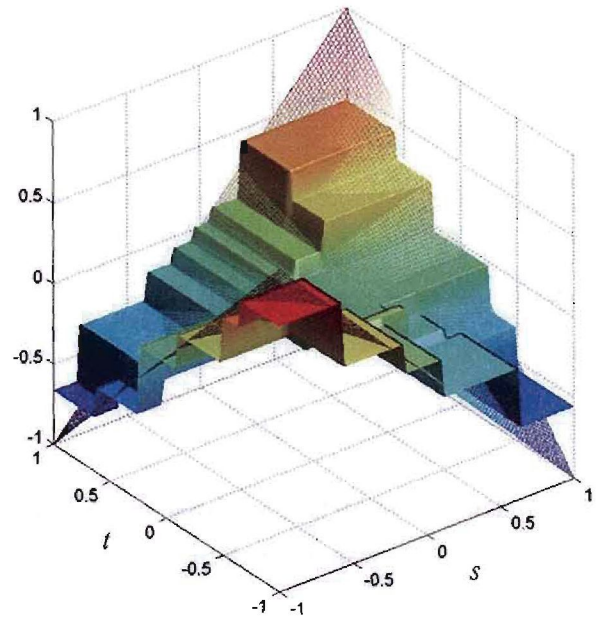


Figure 5.13: Third simulated example – decision tree prediction results

Table 5.3 shows the values of R^2 and ASE for the models on the training and score data sets. The AutoGANN model performed best on the score data set.

Model	Train R^2	Score R^2	Train ASE	Score ASE
ACE_2	1.00	0.99	0.00	0.00
<i>AutoGANN₃</i>	1.00	1.00	0.00	0.00
<i>TREE</i>	0.95	0.88	0.00	0.01

Table 5.3: Third simulated example – train and score R^2 and ASE

5.2 Ethanol fuel example

The ethanol fuel data set has 88 observation cases with predictor variable E and response variable NOX . The estimated models were applied to an input data set with $E = 0.530, 0.532, 0.534, \dots, 1.236$. Figure 5.14 contains a plot of the ACE variable transformation for E and Figure 5.15 contains a plot of the AutoGANN univariate function. Both plots have a parabolic form and may be used to derive a quadratic variable transformation for E .

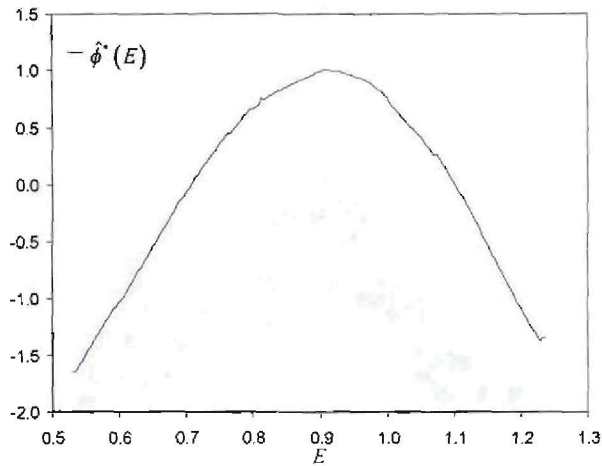


Figure 5.14: Ethanol fuel example – ACE variable transformation

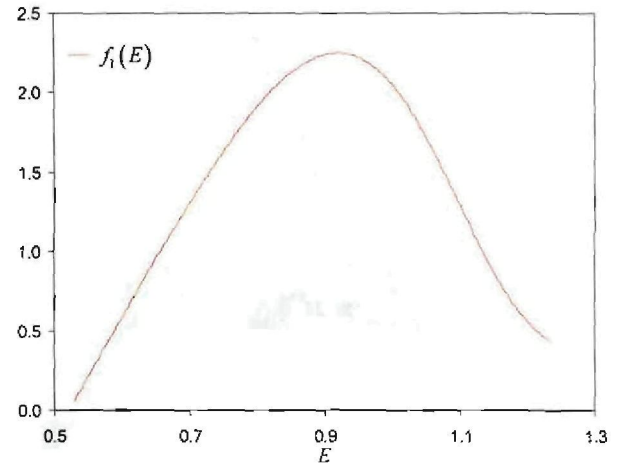


Figure 5.15: Ethanol fuel example – AutoGANN univariate function

Figure 5.16 contains plots of the prediction results of all three models together with the response variable, *NOX*, from the training data set. All the models fitted the data well with the decision tree model having a slightly higher R^2 value as shown in Table 5.4.

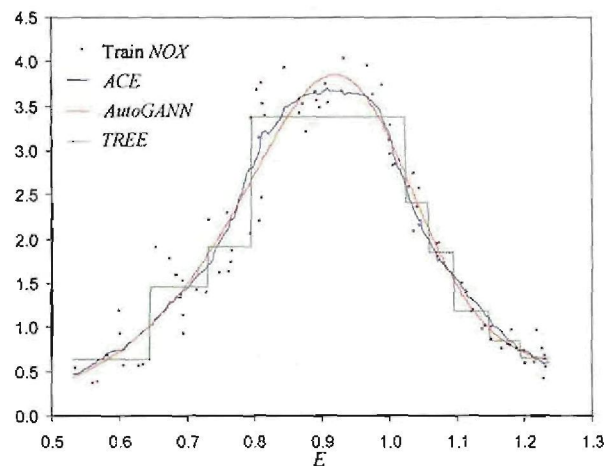


Figure 5.16: Ethanol fuel example – prediction results

Model	Train R^2	Train ASE
ACE	0.91	0.11
AutoGANN	0.92	0.10
TREE	0.93	0.09

Table 5.4: Ethanol fuel example – training R^2 and ASE

5.3 EXPCAR example

The EXPCAR dataset has 100 observation cases with predictor variable x and response variable y . The estimated models were applied to an input data set with $x = 0.0, 0.2, 0.4, \dots, 130$. Figure 5.17 contains a plot of the ACE variable transformation for x and Figure 5.18 contains a plot of the AutoGANN univariate function. Both plots show graphs similar to that of the data scatter plot.

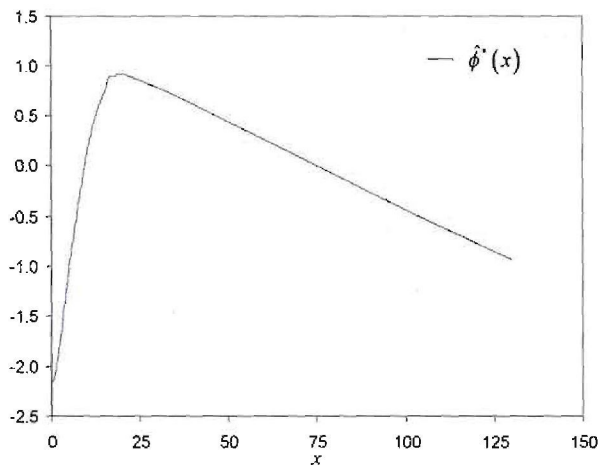


Figure 5.17: EXPCAR example – ACE variable transformation

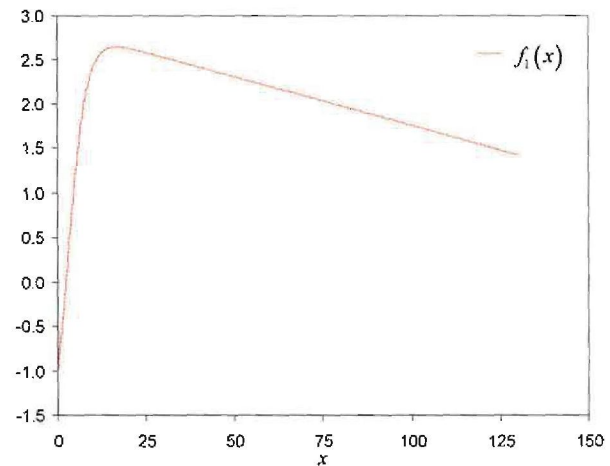


Figure 5.18: EXPCAR example – AutoGANN univariate function

Figure 5.19 contains plots of the prediction results of all three models together with the response variable, y , from the training data set. All the models fitted the data well with the AutoGANN model response values having an approving smooth curve.

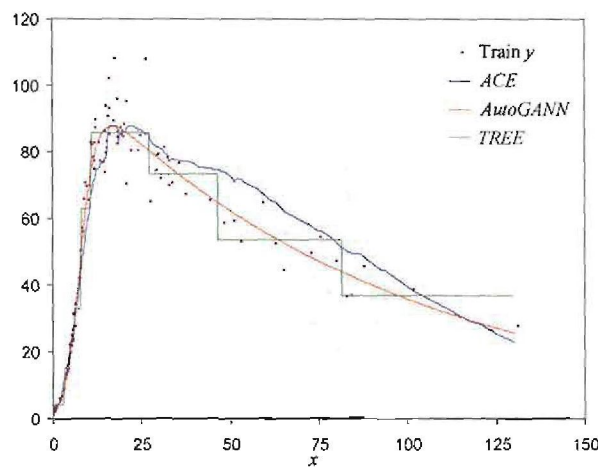


Figure 5.19: EXPCAR example – prediction results

Table 5.5 shows the values of R^2 and ASE for each model on the training data set. All the models fitted the data very well.

<i>Model</i>	<i>Train R^2</i>	<i>Train ASE</i>
<i>ACE</i>	0.96	38.32
<i>AutoGANN</i>	0.96	36.27
<i>TREE</i>	0.95	40.45

Table 5.5: EXPCAR example – training R^2 and ASE

5.4 SO_4 example

The SO_4 example has predictor variables LAT and LON and response variable SO_4 . Figure 5.20 is a stem plot of the 179 observation cases. The estimated models were applied to an input data set with $n=3969$ cases of (LAT_i, LON_j) where $LAT_i = LAT_{i-1} + 0.39$, $LAT_1 = 25.30$, $LON_j = LON_{j-1} + 0.9$, $LON_1 = 68$ and $i, j = 1, \dots, 63$.

The models $AutoGANN_1$ and $AutoGANN_2$ correspond to cases 1 and 2 described in Section 3.6.4. Figure 5.21 and Figure 5.22 contain plots of the prediction results obtained from these models. $AutoGANN_1$ was created using the default AutoGANN model node settings. $AutoGANN_2$ was created with interactions added for the two predictor variables and produced a much better fit to the data set. Figure 5.23 and Figure 5.24 contain plots of the prediction results obtained from the ACE and decision tree models respectively.

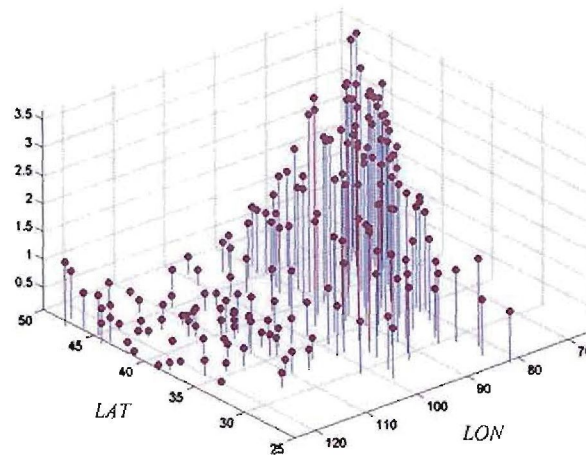


Figure 5.20: SO_4 example – observation data stem plot

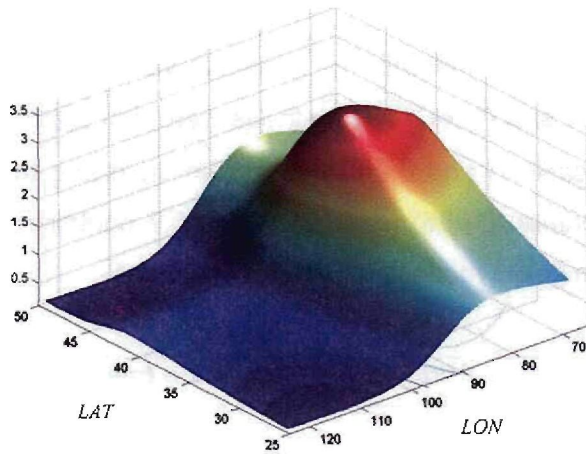


Figure 5.21: SO_4 example – AutoGANN_1 prediction results

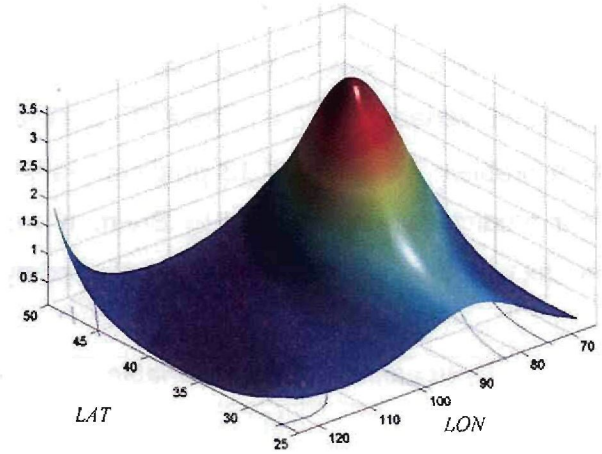


Figure 5.22: SO_4 example – AutoGANN_2 prediction results

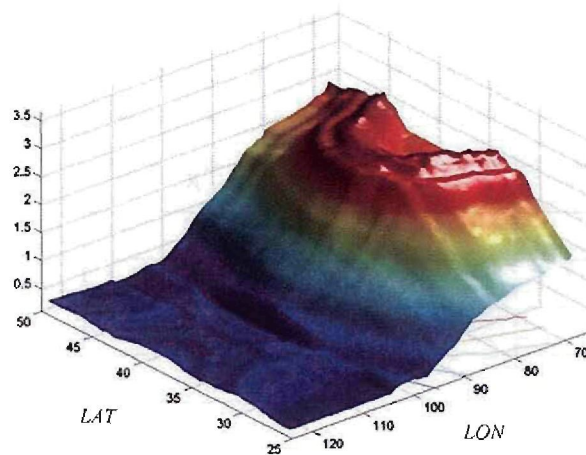


Figure 5.23: SO_4 example – ACE prediction results

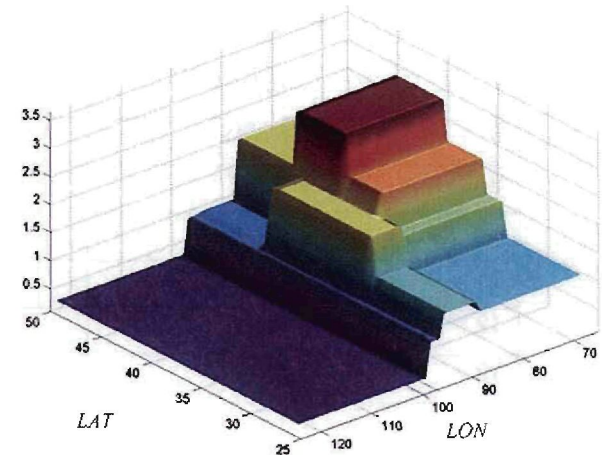


Figure 5.24: SO_4 example – decision tree prediction results

Table 5.6 shows the values of R^2 and ASE for each model. The decision tree fitted the data better than ACE and slightly better than AutoGANN_1 , but with the variable interaction, AutoGANN_2 produced the best fit.

<i>Model</i>	<i>Train R^2</i>	<i>Train ASE</i>
<i>ACE</i>	0.79	0.24
<i>AutoGANN₁</i>	0.85	0.18
<i>AutoGANN₂</i>	0.92	0.09
<i>TREE</i>	0.89	0.13

Table 5.6: SO_4 example – training R^2 and ASE

5.5 Boston Housing example

The Boston Housing data set consists of 507 observations with 13 predictor variables and response variable, $MEDV'$. The ACE, decision trees and AutoGANN methods were first applied to a problem involving all 13 predictor variables (case 1) and then to a problem involving the four predictor variables used by Breiman and Friedman (1985) (case 2). Table 5.7 contains a matrix with the variables that were selected by each of the modelling methods in case 1.

<i>Input number</i>	<i>Input variable</i>	<i>ACE</i>	<i>AutoGANN</i>	<i>Decision tree</i>
1	$LSTAT'$	X	X	X
2	RM'	X	X	X
3	$PTRAT$	X	X	X
4	TAX		X	X
5	$CRIM$		X	X
6	ZN			
7	$INDUS$			
8	$CHAS$		X	
9	NOX'		X	X
10	AGE			X
11	DIS'	X		X
12	RAD'		X	X
13	B'		X	

Table 5.7: Boston Housing example case 1 – model variable selection matrix

The variables $LSTAT'$, RM' and $PTRAT$ were selected by all three the methods indicating that these variables are probably the most important in describing $MEDV'$. The ACE procedure selected only one other variable, DIS' , while the AutoGANN system and the decision tree both included 6 more variables of which TAX , $CRIM$, NOX' and RAD' are present in both models. Variables ZN and $INDUS$ were rejected by all the models.

Table 5.8 shows the values of R^2 and ASE for each model. The decision tree fitted the data best in both cases 1 and 2.

<i>Model</i>	<i>Train R^2</i>	<i>Train ASE</i>
<i>ACE₁</i>	0.83	0.03
<i>AutoGANN₁</i>	0.85	0.03
<i>TREE₁</i>	0.89	0.02
<i>ACE₂</i>	0.78	0.04
<i>AutoGANN₂</i>	0.77	0.04
<i>TREE₂</i>	0.83	0.03

Table 5.8: Boston Housing example case 1 and 2 – training R^2 and ASE

For a third case the Boston Housing data set was randomly divided into 2 data sets, one consisting of 70% of the original data set while the other consists of the remaining 30%. The larger set was used to build the models for ACE, AutoGANN and the decision tree while the smaller set was used as a score data set. Table 5.9 shows the values of R^2 and ASE for each new model on both the training and score data sets. Once again the decision tree model fitted the training data best, but it performed worst on the score data set indicating that it probably overfitted the training data. The AutoGANN model performed best on the score data set.

<i>Model</i>	<i>Train R^2</i>	<i>Score R^2</i>	<i>Train ASE</i>	<i>Score ASE</i>
<i>ACE₃</i>	0.84	0.82	0.02	0.04
<i>AutoGANN₃</i>	0.85	0.83	0.02	0.04
<i>TREE₃</i>	0.86	0.78	0.02	0.04

Table 5.9: Boston Housing example case 3 – training and score R^2 and ASE

5.6 Ozone example

The Ozone dataset consists of 330 observations with 9 predictor variables and response variable *UPO3*. The ACE, decision trees and AutoGANN methods were applied to a problem involving all 9 predictor variables. Table 5.10 contains a matrix with the variables that were selected by each of the modelling methods. The variable *DGPG* was the only variable selected by all three the methods and *WDSP* was the only variable rejected by all.

<i>Input number</i>	<i>Input variable</i>	<i>ACE</i>	<i>AutoGANN</i>	<i>Decision tree</i>
1	<i>VDHT</i>		X	
2	<i>WDSP</i>			
3	<i>HMDT</i>		X	X
4	<i>SBTP</i>	X		X
5	<i>IBHT</i>	X		X
6	<i>DGPG</i>	X	X	X
7	<i>IBTP</i>		X	X
8	<i>VSTY</i>	X	X	
9	<i>DOY</i>		X	X

Table 5.10: Ozone example case 1 – model variable selection matrix

Table 5.11 shows the values of R^2 and ASE for each model. The AutoGANN model fitted the data the best.

<i>Model</i>	<i>Train R^2</i>	<i>Train ASE</i>
<i>ACE₁</i>	0.75	15.69
<i>AutoGANN₁</i>	0.82	11.25
<i>TREE₁</i>	0.79	13.70

Table 5.11: Ozone example case 1 – training R^2 and ASE

A second case was also created by randomly dividing the Ozone data set into 2 data sets with 70% of original data set used for training the models and the remaining 30% used as a score data set. Table 5.12 shows the values of R^2 and ASE for each new model on both the training and score data sets. Although the AutoGANN model fitted the training data best, the ACE model slightly outperformed it on the score data set. The decision tree model did not perform well on the score data set.

<i>Model</i>	<i>Train R^2</i>	<i>Score R^2</i>	<i>Train ASE</i>	<i>Score ASE</i>
<i>ACE₂</i>	0.79	0.79	13.82	11.69
<i>AutoGANN₂</i>	0.82	0.78	12.30	12.42
<i>TREE₂</i>	0.78	0.66	14.62	18.63

Table 5.12: Ozone example case 2 – training and score R^2 and ASE

5.7 Conclusions

In this chapter, the results obtained by applying ACE, the AutoGANN system and decision trees to all the prediction problem examples were presented together and discussed. All three methods performed generally well.

The following chapter contains a comparative discussion on ACE, the AutoGANN system and decision trees in terms of criteria such as predictive capability or accuracy, novelty, utility, interpretability and stability.

6

COMPARATIVE DISCUSSION ON ACE, AUTOGANN, AND DECISION TREES

From the literature study and the results obtained by applying ACE, the AutoGANN system and decision trees to the prediction problem examples, it is possible to compare these methods in terms of criteria such as predictive accuracy, utility, stability, understandability, interpretability and novelty.

6.1 Predictive accuracy

The predictive accuracy of all three methods proved to be good overall. From the set of 5 examples in which a score data set was used (the three simulated examples, the Boston Housing and the Ozone examples) the AutoGANN system outperformed the decision tree and the ACE models on all but the Ozone example where the ACE model predictions on the score data were slightly better. The decision tree did not perform well in problems with a higher number of predictor variables, indicating that overfitting is probable in cases of a high predictor-variable-to-training-observations ratio.

A problem with ACE and decision trees is the inference of response values when the new case falls outside the training problem space. Both methods produce prediction values which are the same as the model values defined on the closest edge of the observed problem space. This is not the case for the AutoGANN model, since the univariate functions remain defined outside the boundaries of the observed predictors. The predictions made just outside the observed problem space may be inaccurate but could still give an indication of what the response would be if the observed relationships were to continue their current trends.

6.2 Utility

All three methods are applicable to regression problems and are easy to use. The application of ACE depends on the particular implementation. The detail of the MATLAB® implementation developed for purposes of this study is given in the addendum and is based on the algorithm

described in Chapter 2. The stepwise variable selection method used by Breiman and Friedman (1985) does not form part of the original standard ACE procedure and requires user interaction unlike the variable selection methods of decision trees and the AutoGANN system. However, it could easily be implemented. The decision tree and AutoGANN model integration with SAS® Enterprise Miner™ makes it easy to use within a complete data mining environment.

Almost all the regression problems contained in the selected examples were solved by using the default setup for ACE, decision trees and the AutoGANN system. The third simulated example containing the extreme variable interaction ($y = st$) posed to be problematic for the additive methods of ACE and AutoGANN. ACE was not able to solve the prediction problem without the addition of a variable representing the interaction. Breiman and Friedman (1985) did, however, use this example to illustrate the ability of the ACE procedure to estimate non-monotonic transformations. Adding variable interactions to the AutoGANN modelling problem using the build-in functionality provided by SAS® Enterprise Miner™, is simple and it solved the problem for the AutoGANN system. The decision tree solved the problem by creating a complex step function, repeatedly using the individual predictor variables in split rules at different levels of the decision tree.

Specifying the user parameter values for the ACE procedure is not trivial. This includes the bass enhancement indicator used for the super-smoother. Recall that the bass enhancement indicator is used to reduce high variance in the super-smoother curves and values from 0 to 10 may be used. A value of zero results in no bass enhancement and the normal variable span smoother is applied. Using a value of 10 is equivalent to applying a constant woofer span. From experimental values used in this study it seems as though a bass enhancement indicator smaller than 9 causes too much bias in the models; hence the models overfit the training data. In this sense, the overhead needed to apply the variable span smoother is not really justifiable, since applying a constant woofer span (50% of n or, alternatively, using a bass enhancement indicator of 10) would generate very similar results. In their rejoinder, Breiman and Friedman (1985b) highlight that the super-smoother variable span selection produces less accurate estimates and that forcing a larger span (greater bass enhancement) produces better estimates. They also give an example in which the threshold value for the iterative procedure termination rule have to be reduced in order to produce acceptable variable transformations.

The default user parameters for the decision tree and AutoGANN model nodes proved to be sufficient in most examples. These methods are quite capable of balancing the variance-bias trade-off automatically. Relative to the AutoGANN system, the decision trees did tend to overfit the training data. According to Genton *et al.* (2008), utilisation of the ANOVA F test as a splitting criterion tends to overfit a tree model. Its performance also degrades with increasing variable correlations. Decision trees can be easily overfitted or underfitted with the combined influence of

the training data set characteristics (such as size) and the stopping rule default values. A data set with less cases results in a decision tree with less bias than a larger data set with same default stopping rules. The same set of stopping rules are not equally appropriate for all data sets.

The limit the user should place on the execution time of the AutoGANN algorithm is not always apparent. Since the model search space explodes exponentially with the addition of more predictor variables, it is advisable to increase the time limit accordingly. It would have been easier if the procedure had been able to suggest an appropriate time limit to serve as a starting point for the analyst, at least after a preliminary run.

In the context of artificial neural networks, the AutoGANN system resolves the problem of selecting appropriate neural network architectures by automating the interactive network construction methodology proposed by Potts (1999). Du Toit (2006) also demonstrates how the automated construction algorithm produces a better model in much less time than using the subjective interactive construction method it is based on.

6.3 Stability

Not all three modelling methods produce equally stable models. The ACE method has a few factors contributing to instability. Breiman and Friedman (1985) points out that an ACE model is highly dependant on the type of smoother used in the algorithm. Even the super-smoother does not always produce desirable results, especially near the boundaries of the problem input space. The ACE model is also unstable in the order in which the problem variables are entered, especially in the case of weak association between the predictor and response variables (Breiman & Friedman, 1985b). This was seen in the Boston Housing example variable selection procedure. At some point in the stepwise procedure the addition of two predictors have very similar prediction results and the rest of the variable selection procedure is then highly dependant on the variables selected at some earlier stage. The same problem applies to decision trees since the tree cultivation method also constitutes a stepwise procedure. Slight changes in the training data set could produce very different models. This problem is referred to by Arentze *et al.* (2006) as variable masking. They point out that this problem raises questions towards the stability and interpretation of the models, but it could be overcome by model averaging or sampling with replacement. The AutoGANN model search strategy does not display this behaviour of instability, provided the time allowed for the procedure is set at an appropriate value. The AutoGANN implementation also allows for model averaging to account for model uncertainty.

6.4 Understandability and interpretability

Apart from predictive accuracy and stability, understandability of a particular model is also important. More often than not predictive model users need to understand the relationships between problem variables and have to be able to interpret the prediction results. Decision tree models are easy to interpret, since every model may be represented as a set of rules defined in ordinary language. The decision rules give insight into the relation of a response to the predictor variable. In the case of ACE, the method supplies the user with (optimal) variable transformations to interpret the variable relationships. They give insight into the relationships between the inputs and the target. These transformations together with the additive nature of the model make the model easier to interpret. The same applies to the partial residual plots and the model architecture restrictions of the AutoGANN system. Since a GANN is based on a generalised additive model, the black-box association with neural networks is overcome.

Decision trees are often used to determine variable importance. The ACE and AutoGANN methods may also be used for this purpose. As can be seen from the AutoGANN modelling results (especially in the Boston Housing example), almost all the variables deemed important by the decision tree, were also selected by the AutoGANN system. In terms of variables importance, decision tree results include a rank for the variables. It would be helpful if a similar functionality could be added to the AutoGANN output.

6.5 Novelty

Linear statistical models for regression have been around for a long time and the theory and application thereof has been studied intensively. With the growth of interest in knowledge discovery and data mining, more flexible non-linear modelling methods such as ACE, decision trees and ANNs have gained interest. Problems associated with extremely flexible models such as these include spurious relationships being identified and the tendency to overfit the data (Hand, 1998). Strategies that may be used to ease these problems include the use of objective model selection criteria (with penalised goodness-of-fit functions), restriction of model complexity and the shrinking of an overfitted model. ACE, decision trees and GANNs all make use of some of or all these strategies. ACE has a *bass* control that can be set to obtain better generalisation, it restricts the model to be additive and the stepwise variable selection procedure shrinks the model. There are various ways by means of which the model complexity of a decision tree may be restricted, for example the restriction of branching factors and the use of stopping and pruning rules. Decision trees make use of adjusted *P*-values to penalise model complexity. Post-pruning constitutes the shrinking of an overfitted model. The AutoGANN system uses objective model selection criteria such as AIC and the SBC that penalise

complexity. The model form is restricted to a generalised additive model. The GANN sub-architectures may be restricted to a specific set of possibilities. A shortcoming in the GANN model selection process still exists in the selection of the neural network output layer activation function. As was seen in the third simulated example, activation function selection based merely on the response variable interval may not always result in the best option.

In general, ACE has been widely praised for its contributions to data analysis, mainly in terms of finding optimal variable transformations. Esteban-Díez *et al.* (2006) ascribe the scarce practical application of ACE to its susceptibility to noise and the fact that it requires a large observations-to-variable ratio in order to obtain reliable results. To resolve this issue they have developed a modified ACE method, GA-ACE, incorporating genetic algorithms and data compression. They have found the results to be considerably better than the original ACE method when applied to multivariate calibration problems.

Decision trees are unique in many ways. The main distinction from other non-linear modelling methods is the statistically non-traditional, rules-based and easily interpretable model definitions. Apart from prediction, other uses of decision trees include the discretising of interval predictor variables, interaction detection and missing value imputation (Potts, 2001).

GANN models compare well to ACE and decision trees in terms of prediction. One of the main goals of the development of the AutoGANN system was the automatic selection of the neural network architecture. From the example results obtained in this study, it proved to be very successful.

6.6 Additional comments

Fowlkes and Kettenring (1985) demonstrate that the ACE variable transformations based on maximum correlation in finite data sets do not always give the data analyst the best choice of data analysis. Buja and Kass (1985) demonstrate the instability of ACE in cases of several suboptimal, approximately equal solutions as well as the lack of robustness to outliers.

Kim (2008) compares decision trees, ANNs and linear regression using 60 simulated examples which differ in sample size as well as in the number and types of independent variables. The conclusion is that, for continuous independent variables, linear regression performs better than decision trees and ANNs, regardless of the number of variables and the sample size. When categorical variables are present, ANNs perform better and the rate of performance increase becomes larger as the number of categorical variables increases.

De Waal and Du Toit (2007) demonstrate that the AutoGANN system would have achieved a fourth place in the KDD Cup 2004 competition. The competition challenged data miners to predict 100,000 new cases with a model trained on a data set with 50,000 cases, 78 predictor variables and a binary response.

7

CONCLUSIONS



The purpose of this study was to investigate and compare ACE, GANNs and decision trees as computerised techniques in the field of predictive data mining. De Waal and Du Toit (2003) developed the automated construction of GANNs called AutoGANN. The aim was to better understand the contribution that this new technology may bring to the field of data mining. ACE and decision trees were chosen as comparative techniques since both have been praised for their performance. According to Wilkinson (1992), decision trees have become so popular that commercial programs are highly competitive in the market. ACE has been deemed extremely powerful by several statistical researchers (Buja & Kass, 1985; Fowlkes & Kettenring, 1985; Pregibon & Vardi, 1985; Lim *et al.*, 1999).

In all, the AutoGANN system compares well with existing approved methods, and in the context of the variety of regression problems presented in this study, it is quite capable of outperforming these methods. The (default) GANN models fitted the data well and predicted new cases on average more accurately than ACE and decision trees. The application of the AutoGANN system is easy and the automatic construction of the neural network architecture simplifies a successful ANN application. The AutoGANN models are more stable compared to ACE and decision trees. It produces output that has similar interpretability to ACE. It gives insight into variable relationships with smooth partial residual curves. The AutoGANN method is well suited to regression problems and should contribute substantially to the field of data mining.

Improvement to the AutoGANN implementation should continue, in particular on the model search performance. Development is under way for the incorporation of genetic algorithms into the AutoGANN system. It is expected that it would improve the model search algorithm. De Pauw *et al.* (2006) have successfully used genetic algorithms together with decision trees and neural networks for predictive ecosystem models. Esteban-Diez *et al.* (2006) have also done some work with genetic algorithms in combination with ACE. Another modification to the AutoGANN system currently being researched is the development of a GANN output activation function selection procedure. Such a procedure should overcome the anomaly observed in one of the simulated prediction problem examples.

Chapter 7 - Conclusions

This study provides insight into the contribution of the AutoGANN system to the field of data mining in terms of regression analysis compared to ACE and decision trees. A promising area of future research would be in applying this approach to compare the performance on classification problems.

ADDENDUM – MATLAB® ACE CODE

Main functions

Function ace

%Alternating conditional expectations - based on the algorithm by Breiman and
%Friedman (1985)

%-----

```
function [theta, phi, error, iter] =  
ace(X,Y,span,bassControl,max_iter,e_change,NN_indc);
```

%-----Input parameters-----

```
% 1. X is an n by p matrix containing the p predictor variables column wise;  
%    the n rows represent the n observation cases.  
% 2. Y is an n by 1 column vector containing the n response variable  
%    observations.  
% 3. span is a column vector of span values in [0,1] to be used by the  
%    super-smoother. A value of [0] causes the default spans to be used,  
%    [0.05;0.2;0.5].  
% 4. bassControl is a value in [0,10] used for bass enhancement by the  
%    super-smoother.  
% 5. max_iter is the maximum number of allowed ACE iterations.  
% 6. e_change is the stop-value for the change in error.  
% 7. NN_indc is an indicator for the use of nearest neighbours smoothing:  
%    anything other than 0 -> use nearest neighbours, 0 -> use symmetric  
%    span.
```

%-----Output parameters-----

```
% 1. theta is an n by 1 column vector containing the n response variable  
%    transformation values.  
% 2. phi is an n by p matrix containing the p predictor variable  
%    transformations values.  
% 3. error is the final error that has been minimised.  
% 4. iter is the number of ACE iterations that has been executed.
```

%-----BEGIN ace-----

```
%Determine the size of the sample, n
```

```
n = size(X,1);
```

```
%Test whether the number of observation cases are the same in X and Y.
```

```
%If not, display an error message and terminate.
```

```
if (n ~= size(Y,1))
```

```
    error('X and Y not of the same size');
```

```
end
```

```
%Determine the number of predictor variables, p
```

```
p = size(X,2);
```

```
%Determine whether the default super-smoother span values should be used
```

```
if (span == 0)
```

Addendum

```
span = [0.05;0.2;0.5];
end

%Initialise the response variable transformation to  $Y/||Y||$ 
theta = Y./((mean(Y.^2)).^(1/2));

%Initialise the predictor variable transformations to zero
phi = zeros(n,p);

%Initialise the error, the difference in error and the number of iterations
error = mse(phi,theta); %call to mse
diff = 1;
iter = 0;

%Iterate until the error fails to decrease by more than the stop-value
%or until the maximum number of iterations is reached for the outer loop
while ((diff > e_change) & (iter < max_iter))
    %Increment the number of outer loop iterations
    iter = iter + 1;
    %Initialise the inner loop iterations
    iter_inner = 0;
    %Iterate until the error fails to decrease by more than the stop-value
    %or until the maximum number of iterations is reached for the inner loop
    while ((diff > e_change) & (iter_inner < max_iter))
        %Increment the number of inner loop iterations
        iter_inner = iter_inner + 1;
        %Save the previous predictor variable transformations
        phi_previous = phi;
        %For every predictor variable, calculate the new variable
        %transformation
        for k = 1:p
            %Calculate SUM(phi) excluding the current predictor variable
            phi_k_sum = sum(phi,2) - phi(:,k);
            %Smooth theta-SUM(phi) according to the current predictor variable
            [phi(:,k),vs,vss] = superSmoother(X(:,k), %call to superSmoother
            (theta - phi_k_sum),span,bassControl,NN_indc);
            %Subtract the mean
            phi(:,k) = phi(:,k) - mean(phi(:,k));
        end;
        %Determine the change in error
        error_previous = error;
        error = mse(phi,theta); %call to mse
        diff = error_previous - error;
        %If the error has increased, revert back to the previous transformation
        if (diff < 0)
            phi = phi_previous;
            error = error_previous;
        end;
    end; %Inner loop

    %Calculate theta, the new response variable transformation
    %Save the previous response variable transformation
    theta_previous = theta;
    %Smooth SUM(phi) according to Y
    [theta,vs,vss] = superSmoother(Y,sum(phi,2),span,bassControl,NN_indc);
    %Subtract the mean
    theta = theta - mean(theta);
    %Normalise the variable transformation
    theta = theta./((mean(theta.^2)).^(1/2));

    %Determine the change in error
```


Addendum

```
error_previous = error;
error = mse(phi,theta); %call to mse
diff = error_previous - error;
%If the error has increased, revert back to the previous transformation
if (diff < 0)
    theta = theta_previous;
    error = error_previous;
end;
end %outer loop

%-----END ace-----
```

Function *aceModel*

```
%Alternating conditional expectations model smooth
%Determines the ACE predictive function from the ACE variable transformations
%-----
function [f,t] = aceModel(Y,phi);

%-----Input parameters-----
% 1. Y is an n by 1 column vector containing the n response variable
%    observations.
% 2. phi is an n by p matrix containing the p predictor variable
%    transformations values.

%-----Output parameters-----
% 1. f is an n by 1 column vector containing the n ACE model smooth values.
% 2. t is an n by 1 column vector containing values for the sum of the p
%    predictor variable transformations values.

%-----BEGIN aceModel-----

%Calculate the sum of the transformed predictor values for each observation
t = sum(phi,2);

%smooth Y over the sum of the transformed predictor values using a midrange
%smoother
[f,vs,vss] = superSmoother(t,Y,0.2,0,0); %call to superSmoother

%-----END aceModel-----
```

Function *aceScore*

```
%ACE scoring function
%Calculate response estimates using an ACE model
%-----
function [P,ItS,It] = aceScore(I,X,phi,f,t);

%-----Input parameters-----
% 1. I is an m by p matrix containing the p new predictor variables.
% 2. X is an n by p matrix containing the p original predictor variables.
% 3. phi is an n by p matrix containing the p predictor variable
%    transformations values.
% 4. f is an n by 1 column vector containing the n ACE model smooth values.
% 5. t is an n by 1 column vector containing values for the sum of the p
%    predictor variable transformations values.
```

Addendum

```
%-----Output parameters-----
% 1. P is an m by 1 column vector containing the newly predicted response
%    values.
% 2. ItS is an m by 1 column vector containing values for the sum of the p
%    new predictor variable transformations values.
% 3. It is an m by p matrix containing the p predictor variable
%    transformations for I.

%-----BEGIN aceScore-----

%Determine the number of predictor variables, p
p = size(X,2);

%Calculate the transformation for each predictor using linear interpolation
It = I;
for k = 1:p
    Ik = I(:,k);
    %Transform phi to be strictly monotone over X
    [XM,phiM] = toMonotone(X(:,k),phi(:,k)); %call to toMonotone
    %Use interpolation to determine the transformed new predictor values
    It(:,k) = aceInterp(XM,phiM,Ik); %call to aceInterp
end

%Calculate the sum of the transformed new predictors
ItS = sum(It,2);

%Compute the prediction value from the model smooth
%Transform f to be monotone over t
[tm,fm] = toMonotone(t,f); %call to toMonotone
%Use interpolation to determine the prediction result
P = aceInterp(tm,fm,ItS); %call to aceInterp

%-----END aceScore-----
```

Function *superSmoother*

```
%Super-smoother - based on the algorithm by Friedman (1984)
%A local least squares smoother with automatic variable span selection
%-----

function [smooth,var_span,var_span_smoothed] =
superSmoother(x,y,spans,bassControl,NN_indc);

%-----Input parameters-----
% 1. x is an n by 1 column vector containing the data points to smooth by.
% 2. y is an n by 1 column vector containing the data points to be smoothed
%    according to x.
% 3. spans is a column vector of span values in [0,1].
% 4. bassControl is a value in [0,10] used for bass enhancement.
% 5. NN_indc is an indicator for the use of nearest neighbours smoothing:
%    anything other than 0 -> use nearest neighbours, 0 -> use symmetric
%    span.

%-----Output parameters-----
% 1. smooth is an n by 1 column vector containing the smoothed values for y.
% 2. var_span is an n by 1 column vector containing the selected span
%    values.
% 3. var_span_smoothed is an n by 1 column vector containing the smoothed
%    span values.
```


Addendum

```
%-----BEGIN superSmoother-----

%Determine the size of the sample, n
n = size(x,1);
%Test whether x and y are of the same size.
%If not, display an error message and terminate.
if (n ~= size(y,1))
    error('x and y not of the same size');
end

%Sort x and y according to x in ascending order
[x,y,o] = ssSort(x,y); %call to ssSort

%Set the value for the midrange smoother
midspan = 0.2;
%Set the value for a small number
small = 1.0e-6;
%Determine the number of span values to evaluate
nr_of_spans = size(spans,1);
%Determine the width of one side of a symmetric span
ws = fix((spans*n)/2);
%Determine the width of a nearest neighbours span
wnn = fix(spans*n);

%Initialise the smooth values, the residuals and smoothed residuals for each
%span value to zero
smooths = zeros(n,nr_of_spans);
res = zeros(n,nr_of_spans);
res_smoothed = zeros(n,nr_of_spans);

%For each span value
for k = 1:nr_of_spans
    %Calculate the smooth value and cross-validated absolute residuals

    %For each observation
    for i = 1:n
        %Find the neighbourhood N of x at i
        [min_index,max_index] = neighbourhoodInterval(i,n,ws(k),
        wnn(k),x,NN_indc); %call to neighbourhoodInterval
        NXi = x(min_index:max_index);
        NYi = y(min_index:max_index);
        %Determine the size of the neighbourhood
        J = size(NXi,1);
        %Fit a least squares straight line through the points in the
        %neighbourhood
        mean_x = mean(NXi);
        mean_y = mean(NYi);
        Vj = sum((NXi - mean_x).^2);
        %If the slope of the line is not infinity
        if (Vj ~= 0)
            Cj = sum((NXi - mean_x).*(NYi - mean_y));
            beta = double(Cj)/Vj;
            alpha = mean_y - beta*mean_x;
            %set the smoothed value as the value of the line at i
            smooths(i,k) = beta*x(i) + alpha;
        %else
        else
            %take the average of the y-values
            smooths(i,k) = mean_y;
        end
    end
end
```

Addendum

```
%Calculate the cross-validated residuals
if (Vj == 0)
    Vj = small;
end
res(i,k) = ((y(i)-smooths(i,k))) /
    ((1- 1/J- (((x(i)-mean_x).^2)/Vj)));
end %for each observation

%Smooth the absolute residuals over x using the midrange smoother
res_abs = abs(res(:,k));
res_smoothed(:,k) = smoothLeastSq(x,res_abs,midspan,
NN_indc); %call to smoothLeastSq
end %for each span value

%Calculate the estimated error as the minimum smoothed residual at each x
%and determine the corresponding optimal span values
err = zeros(n,1);
var_span = zeros(n,1);
for i = 1:n
    [err(i),index] = min(res_smoothed(i,:));
    var_span(i) = spans(index);
end

%Apply bass enhancement if desired
if ((bassControl > 0) & (bassControl <= 10))
    [woofer,w_index] = max(spans);
    Ri = err./res_smoothed(:,w_index);
    var_span = var_span + (woofer-var_span).*(Ri.^(10-bassControl));
end

%Smooth the selected span values over x with the midrange smoother
var_span_smoothed = smoothLeastSq(x,var_span,midspan,
NN_indc); %call to smoothLeastSq

%Calculate the final smooth as the average between the 2 smooths
%with their spans closest (in distance) to the smoothed variable span value
smooth = zeros(n,1);
for i = 1:n
    distance = abs(spans-var_span_smoothed(i));
    [v,index1] = min(distance);
    distance(index1) = -1;
    [v,index2] = min(distance);
    smooth(i) = (smooths(i,index1) + smooths(i,index2))/2;
end

%Arrange the data into the original order
[smooth,var_span,var_span_smoothed] =
ssResort(smooth,var_span,var_span_smoothed,o); %call to ssResort

%-----END superSmoother-----
```

Auxiliary functions

Function aceInterp

```
%ACE interpolation function
%Calculates the interpolated values t at s using vectors x and y.
%-----
```

Addendum

```
function [t] = aceInterp(x,y,s);

%-----Input parameters-----
% 1. x is an n by 1 column vector that is monotone increasing.
% 2. y is an n by 1 column vector containing the values used for
%    interpolation.
% 3. s is an m by 1 column vector containing the new x-points for
%    interpolation.

%-----Output parameters-----
% 1. t is an m by 1 column vector containing the interpolated values at s.

%-----BEGIN aceInterp-----

%Determine the number of point available for interpolation
n = size(x,1);
%Initialise t to be equal to s
t = s;

%For each new data point in s
for (k = 1:size(s,1))
    %If the new point falls outside the interval of x use the boundary values
    %of y
    if (s(k) <= x(1)) t(k) = y(1);
    elseif (s(k) >= x(n)) t(k) = y(n);
    %Else
    else
        %Find existing points that matches the new point
        [i,j] = find(x==s(k));
        %If the new point matches only one existing point, use the corresponding
        %y-value
        if (size(i,1)==1)
            t(k) = y(i);
        %Else if the new point does not exist in x
        elseif (size(i,1)==0)
            %Find the indexes of the two existing data points either side of the
            %new point
            [i,j] = find(s(k) > x);
            i1 = max(i);
            [i,j] = find(s(k) < x);
            i2 = min(i);
            %and calculate the interpolated value
            t(k) = y(i1) + ((y(i2)-y(i1))*(s(k)-x(i1)))/(x(i2)-x(i1));
        %Else x is not monotone, display an error and terminate
        else
            error('x not monotone');
        end
    end
end %for each new data point in s

%-----END aceInterp-----
```

Function mse

```
% Mean square error function
%-----
function [e] = mse(X,Y);

%-----Input parameters-----
% 1. X is an n by p matrix.
% 2. Y is an n by 1 column vector.
```


Addendum

```
%-----Output parameters-----
% 1. e returns the mean square error for a regression of Y on SUM(X)

%-----BEGIN mse-----

e = (sum((Y - sum(X,2)).^2))/size(X,1);

%-----END mse-----
```

Function *neighbourhoodInterval*

```
%Super-smoother neighbourhood interval
%-----
function [i1,i2] = neighbourhoodInterval(i,n,ws,wnn,x,NN_indc);

%-----Input parameters-----
% 1. i is a scalar indicating the index around which a neighbourhood of x
%    should be determined.
% 2. n is a scalar indicating the size of the vector, x.
% 3. ws is a scalar indicating the width of one side of a symmetric
%    neighbourhood.
% 4. wnn is a scalar indicating the width of a nearest neighbours
%    neighbourhood.
% 5. x is an n by 1 column vector containing the data points for which a
%    neighbourhood should be determined.
% 6. NN_indc is an indicator for the determination of the neighbourhood:
%    anything other than 0 -> nearest neighbours, 0 -> symmetric
%    neighbourhood.

%-----Output parameters-----
% 1. i1 is the minimum index for the neighbourhood.
% 2. i2 is the maximum index for the neighbourhood.

%-----BEGIN neighbourhoodInterval---

%If NN_indc = 0, determine a symmetric neighbourhood around x at i
if (NN_indc == 0)
    i1 = max([i-ws;1]);
    i2 = min([i+ws;n]);
%Else, determine a nearest neighbours neighbourhood around x at i
else
    index1 = max([1;i-wnn+1]);
    index2 = min([i+wnn-1;n]);
    interval = index1:index2;
    sz = size(interval',1);
    distance = [interval' x(interval) zeros(sz,1)];
    distance(:,3) = abs(distance(:,2) - x(i));
    sorted = sortrows(distance,3);
    i1 = min(sorted(1:wnn,1));
    i2 = max(sorted(1:wnn,1));
end %if, else

%-----END neighbourhoodInterval---
```

Function *smoothLeastSq*

```
%Local least squares smoother
%-----
```

Addendum

```
function [smooth] = smoothLeastSq(x,y,span,NN_indc);

%-----Input parameters-----
% 1. x is an n by 1 column vector containing the data points to smooth by.
% 2. y is an n by 1 column vector containing the data points to be smoothed
%    according to x.
% 3. span is a scalar value in [0,1].
% 4. NN_indc is an indicator for the use of nearest neighbours smoothing:
%    anything other than 0 -> use nearest neighbours, 0 -> use symmetric
%    span.

%-----Output parameters-----
% 1. smooth is an n by 1 column vector containing the smoothed values for y.

%-----BEGIN smoothLeastSq-----

%Determine the size of the sample, n
n = size(x,1);
%Test whether x and y are of the same size.
%If not, display an error message and terminate.
if (n ~= size(y,1))
    error('x and y not of the same size');
end

%Determine the width of one side of a symmetric span
ws = fix((span*n)/2);
%Determine the width of a nearest neighbours span
wnn = fix(span*n);

%Initialise the smooth values to zero
smooth = zeros(n,1);

%For each observation
for i = 1:n
    %Find the neighbourhood N of x at i
    [min_index,max_index] = neighbourhoodInterval(i,n,ws,wnn,
    x,NN_indc); %call to neighbourhoodInterval
    NXi = x(min_index:max_index);
    NYi = y(min_index:max_index);
    %Determine the size of the neighbourhood
    J = size(NXi,1);
    %Fit a least squares straight line through the points in the neighbourhood
    mean_x = mean(NXi);
    mean_y = mean(NYi);
    Vj = sum((NXi - mean_x).^2);
    %If the slope of the line is not infinity
    if (Vj ~= 0)
        Cj = sum((NXi - mean_x).*(NYi - mean_y));
        beta = double(Cj)/Vj;
        alpha = mean_y - beta*mean_x;
        %set the smoothed value as the value of the line at i
        smooth(i) = beta*x(i) + alpha;
    else
        %take the average of the y-values
        smooth(i) = mean_y;
    end
end %For each observation

%-----END smoothLeastSq-----
```

Function ssSort

```
%Super-smoother sort function
%-----
function [X,Y,O] = ssSort(x,y);

%-----Input parameters-----
% 1. x is an n by 1 column vector containing the data points to sort by.
% 2. y is an n by 1 column vector containing the data points to be sorted
%    according to x.

%-----Output parameters-----
% 1. X is an n by 1 column vector containing the sorted x values.
% 2. Y is an n by 1 column vector containing the data points of y sorted
%    according to x.
% 3. O is an n by 1 column vector containing the original positions of the
%    data points in X and Y

%-----BEGIN ssSort-----

o = 1:size(x,1);
temp = [x y o'];
t = sortrows(temp,1); %call to the MATLAB sortrows function
X = t(:,1);
Y = t(:,2);
O = t(:,3);

%-----END ssSort-----
```

Function ssResort

```
%Super-smoother resort function
%-----
function [X,Y,Z] = ssResort(x,y,z,o);

%-----Input parameters-----
% 1. x,y and z are n by 1 column vectors containing the data points to
%    resort.
% 2. o is an n by 1 column vector containing the original positions of the
%    data points.

%-----Output parameters-----
% 1. X,Y and Z are n by 1 column vectors containing the data from x,y and z
%    sorted according to o

%-----BEGIN ssResort-----

temp = [o x y z];
t = sortrows(temp,1); %call to the MATLAB sortrows function
X = t(:,2);
Y = t(:,3);
Z = t(:,4);

%-----END ssResort-----
```


Function *toMonotone*

```

%Transformation function to strictly monotone
%Returns vectors xm and ym sorted according to x and duplicate x values
%eliminated
%-----

function [xm,ym] = toMonotone(x,y);

%-----Input parameters-----
% 1. x is an n by 1 column vector to be rearranged to be strictly monotone.
% 2. y is an n by 1 column vector to be rearranged to be strictly monotone
%    over x.

%-----Output parameters-----
% 1. xm is an n by 1 column vector rearranged to be strictly monotone.
% 2. ym is an n by 1 column vector rearranged to be strictly monotone over
%    x.

%-----BEGIN toMonotone-----

%Sort x and y according to x
[xS,yS,order] = ssSort(x,y); %call to ssSort

%Find unique values of x
[b,i,j] = unique(xS);
xm = xS(i);
ym = xm;

%Remove duplicate values of y
start = 1;
for k = 1:size(xm,1)
    ym(k) = mean(yS(start:i(k)));
    start = i(k) + 1;
end

%-----END toMonotone-----

```


BIBLIOGRAPHY

ANDERSON, D.R. & BURNHAM, K.P. 1998. Model selection and inference: a practical information-theoretic approach. New York : Springer-Verlag.

ARENTZE, T., BRIJS, T., JANSSENS, D., TIMMERMANS, H., VANHOOF, K. & WETS, G. 2006. Integrating Bayesian networks and decision trees in a sequential rule-based transportation model. *European journal of operational research*, 175(1):16-34, Nov.

BERK, K.N. & BOOTH, D.E. 1995. Seeing a curve in multiple regression. *Technometrics*, 37(4):385-398, Nov.

BREIMAN, L. 1996. Technical note: some properties of splitting criteria. *Machine learning*, 24(1):41-47, Jul.

BREIMAN, L., FRIEDMAN, J.H., OLSHEN, R.A., & STONE, C.J. 1984. Classification and regression trees. Belmont, CA : Wadsworth Inc.

BREIMAN, L. & FRIEDMAN, J.H. 1985. Estimating optimal transformations for multiple regression and correlation. *Journal of the American Statistical Association*, 80(391):580-598, Sep.

BREIMAN, L. & FRIEDMAN, J.H. 1985b. Rejoinder for 'Estimating optimal transformations for multiple regression and correlation'. *Journal of the American Statistical Association*, 80(391):614-619, Sep.

BRINKMAN, N. D. 1981. Ethanol fuel - a single-cylinder engine study of efficiency and exhaust emissions. *SAE transactions*, 90(810345):1410-1424.

BUJA, A. & KASS, R.E. 1985. Comments on 'Estimating optimal transformations for multiple regression and correlation'. *Journal of the American Statistical Association*, 80(391):602-607, Sep.

Bibliography

DE PAUW, N., D'HEYGERE, T. & GOETHALS, P.L.M. 2006. Genetic algorithms for optimisation of predictive ecosystems models based on decision trees and neural networks. *Ecological modelling*, 195:20-29.

DE WAAL, D.A. & DU TOIT, J.V. 2003. Automated construction of generalized additive neural networks for predictive data mining. (*In* Mende, J. & Sanders, I., eds. Proceedings of the 33rd annual conference of the South African Computer Lecturer's Association. Johannesburg : University of the Witwatersrand.)

DE WAAL, D.A. & DU TOIT, J.V. 2007. Stepwise model selection for generalised additive neural networks. Technical report FABWI-N-RA/RKW: 2007-206. Potchefstroom : Business Mathematics and Informatics, North-West University.

DU TOIT, J.V. 2006. Automated construction of generalized additive neural networks for predictive data mining. Potchefstroom : North-West University. (Thesis – PhD.)

ESTEBAN-DÍEZ, I., FORINA, M., CONZÁLEZ-SÁIZ, J. & PIZARRO, C. 2006. GA-ACE: alternating conditional expectations regression with selection of significant predictors by genetic algorithms. *Analytica chimica acta*, 555:96-106.

FOWLKES, E.B. & KETTENRING, J.R. 1985. Comments on 'Estimating optimal transformations for multiple regression and correlation'. *Journal of the American Statistical Association*, 80(391):607-613, Sep.

FRIEDMAN, J.H. 1984. A variable span smoother. LCS technical report no. 5 SLAC PUB-3477. Stanford, CA : Department of Statistics, Stanford University.

FRIEDMAN, J.H. & SILVERMAN, B.W. 1989. Flexible parsimonious smoothing and additive modeling. *Technometrics*, 31(1):3-39, Feb.

GENTON, M.G., STEFANSKI, L.A. & WU, Y. 2008. A comparison of node-splitting rules in recursive partitioning analysis of multivariate quantitative structure activity data. *Statistics in biopharmaceutical research*, to appear.

<http://www.unige.ch/ses/metri/genton/2008.WGS.SBR.proofs.pdf> Date of access: 4 Apr. 2008.

HAND, D.J. 1998. Data mining: statistics and more? *The American statistician*, 52(2):112-118.

Bibliography

- HARRISON, D. & RUBINFELD, D.L. 1978. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5:81-102.
- HASTIE, T.J. & TIBSHIRANI, R.J. 1986. Generalized additive models. *Statistical science*, 1(3):297-318, Aug.
- HOETING, J.A., MADIGAN, D., RAFTERY, A.E. & VOLINSKY, C.T. 1999. Bayesian model averaging: a tutorial. *Statistical science*, 14(4):382-417.
- KASS, G.V. 1980. An exploratory technique for investigating large quantities of categorical data. *Applied statistics*, 29(2):119-127.
- KIM, Y.S. 2008. Comparison of the decision tree, artificial neural network, and linear regression methods based on the number and types of independent variables and sample size. *Expert systems with applications*, 34:1227-1234.
- KUTNER, M.H., LI, W., NACHTSHEIM, C.J. & NETER, J. 2005. Applied linear statistical models. 5th ed. New York : McGraw-Hill.
- LARSEN, W.A., McCLEARY, S.J. 1972. The use of partial residual plots in regression analysis. *Technometrics*, 14(3):781-790, Aug.
- LEAMER, E.E. 1978. Specification searches: ad hoc inference with nonexperimental data. New York : Wiley.
- LEE, H.K.H. 1999. Model selection and model averaging for neural networks. Pittsburgh : Carnegie Mellon University. (Thesis – PhD.)
- LIM, H., QUEK, S. & SUM, C. 1999. Analysing interaction effects on MRP implementation using ACE. *International journal of production economics*, 58:303-318.
- LOH, W. & SHIH, Y. 1997. Split selection methods for classification trees. *Statistica Sinica*, 7:815-840.
- MORGAN, J.N. & SONQUIST, J.A. 1963. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58(302):415-434.

Bibliography

NELDER, J.A. & WEDDERBURN, R.W.M. 1972. Generalized linear models. *Journal of the Royal Statistical Society A*, 135(3):370-384.

NEVILLE, P.G. 1999. Decision trees for predictive modeling. Technical report. Cary, NC : SAS Institute Inc.

POTTS, W.J.E. 1999. Generalized additive neural networks. (*In* KDD-99 Conference Proceedings. San Diego, CA.)

POTTS, W.J.E. 2000. Neural network modeling course notes. Cary, NC : SAS Institute Inc.

POTTS, W.J.E. 2001. Decision tree modeling course notes. Cary, NC : SAS Institute Inc.

POTTS, W.J.E. 2002. Applying data mining techniques using Enterprise Miner™ course notes. Cary, NC : SAS Institute Inc.

PREGIBON, D. & VARDI, Y. 1985. Comments on 'Estimating optimal transformations for multiple regression and correlation'. *Journal of the American Statistical Association*, 80(391): 598-601, Sep.

QUINLAN, R.J. 1993. C4.5: Programs for machine learning. San Mateo, CA : Morgan Kaufmann Publishers Inc.

SARLE, W.S. 1994. Neural networks and statistical models. (*In* SAS Institute Inc. Proceedings of the nineteenth annual SAS Users Group international conference. Cary, NC.)

STONE, M. 1974. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B*, 36(2):111-147.

WALSH, S. 2002. Applying data mining techniques using Enterprise Miner™: course notes (ADMT). Cary, NC : SAS Institute Inc.

WILKINSON, L. 1992. Tree structured data analysis: AID, CHAID and CART. (*In* Sawtooth/SYSTAT. Paper presented at the Sawtooth/SYSTAT joint software conference. Sun Valley, ID)

Bibliography

XIANG, D. 2001. Fitting generalized additive models with the GAM procedure. (*In* SAS Institute Inc. Proceedings of the twenty-sixth annual SAS Users Group international conference. Cary, NC.)

ZHANG, G., PATUWO, B.E., HU, M.Y. 1998. Forecasting with artificial neural networks: the state of the art. *International journal of forecasting*, 14:35-62.