Manchester Metropolitan University

United Kingdom

Tree Based Methods for Rule Extraction from Artificial Neural Networks

A thesis submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in the Department of Computing and Mathematics

by

Darren Dancey

August, 2008

Acknowledgments

I would like to express my gratitude to my supervisors Dr. Zuhair Bandar and Dr. David McLean for the guidance and support.

During the course of my this research I have had many office/lab mates who have provided encouragement, advice and in many cases a good deal of much needed distraction. By naming them I will inevitable miss out people who are more than deserving of mention but Jon, Jay, Andy, Nic, Bob, Ben, Tim, David, and Naresh have become what I am sure will be life-long friends.

The Department of Computing and Mathematics has provided me with a friendly and supportive environment. Dr Martin Stanton, and Dr David Britch have provided much insight and advice[1]. Gill Grundy has made sure I get at least one meal a day that does not come from a vending machine for which I am grateful.

Last, but not least, I would also like to thank my family for their support, patience and understanding.

[1]most significantly on beer

Abstract of the Thesis

Tree Based Methods for Rule Extraction from Artificial Neural Networks

Artificial Neural Networks are powerful and flexible tools for pattern recognition, inspired by biological neurons, such as those making up the human brain. Artificial neural networks have successfully found widespread use, but further adoption is hindered in many areas because, like their biological counterparts, artificial neural networks do not reveal the knowledge they have learnt in a readily understandable form.

This thesis presents new algorithms for extracting decision trees from artificial neural networks. Decision trees, unlike neural networks, are a graphical representation of a decision process that are intuitively easy to understand. This thesis extends previous algorithms in this area by making use of new developments in decision trees. The algorithms developed do not require specialised neural network architectures or training algorithms and can be applied to existing neural networks and other classifier types that are black boxes.

In addition to algorithms for the extraction from classification domains, this thesis also presents algorithms to extract model trees from artificial neural networks trained on regression problems. Artificial neural networks make excellent models for function appromimation, but extraction from such neural networks has been a neglected area of research.

To show the real-world applicability of these algorithms an empirical evaluation was completed on 16 real-world datasets from the standard machine learning benchmark repositories. This evaluation confirms that the algorithms are capable of extracting decision trees that achieve higher predictive classification accuracy than decision trees directly induced on the datasets, and also maintains high level of fidelity with the neural network from which they are extracted.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Pattern Recognition encompasses a wide range of information processing problems, such as recognising a family member's face, recognising their voice on a poor quality telephone line, deciding which move to make next in chess, diagnosing diseases, to predicting future movements of the stock market. The human brain seems remarkably well adapted to solving many of these types of tasks. For example, we are often able to recognise old school friends even after an absence of many years, regardless of the fact that their features may have changed dramatically in the intervening years.

However, creating computer programs to solve such problems has been problematic. It is often difficult to express a solution to these types of problems as a series of procedural steps, which is the traditional form of a computer program. Neural networks provide an alternative style of information processing that is well suited to solving the pattern recognition problem. Neural networks are a collection of computational models and techniques based on biological neural networks such as the human brain. However, neural networks are 'black boxes' and do not reveal how they make their decisions. This thesis shows how a comprehensible representation of the decision process in the form of a decision tree can be extracted from trained neural networks.

The Pattern Recognition problem as described by Ripley [69] is:

> *Given some examples of complex signals and the correct decisions for them, make decisions automatically for a stream of future examples.*

The basic framework for classification[69] is that certain objects need to be classified as coming from a number of classes $C_1, \ldots, C_K$. A process called feature extraction takes a number (p) of measurements from the object. This produces a vector of features, X, commonly called an instance. X therefore belongs to an instance space $\mathscr{X}$. The instance space, $\mathscr{X}$, being $\mathscr{X}_1 \times \mathscr{X}_2 \times \cdots \times \mathscr{X}_p$ where $\mathscr{X}_i$ is either the set of Reals, $\mathbb{R}$, for real valued features or a finite set for nominal valued feature. The task is then to build a classifier, $\hat{C}$, that given an instance, $X = x$, will classify it as one of the K classes, that is

$$\hat{C} : \mathscr{X} \to 1 \ldots K. \tag{1.1}$$

1.1 Artificial Intelligence and Artificial Neural Networks

The field of Artificial Intelligence can be considered an attempt to solve computationally tasks that would usually be assumed to require intelligence[1]. Defining artificial intelligence requires a definition of (natural) intelligence which quickly becomes philosophical. Turing attempted to side-step the issue of definition by setting the well-known Turing test[88] for artificial intelligence. The test required an artificial intelligence program to convince someone through text-based conversation that they were in conversation with a human. But the test sets a very high bar; a program passing the test would certainly be considered artificially intelligent, but a program that fails the test could still be considered intelligent. For example, a young child that could not read would fail the test.

Symbolic AI techniques have attempted to create artificial intelligence at an abstract level. It is characterised by attempts to reason using sets of facts and rules. Symbolic AI has a long history with some early successes such as Mycin[78] an expert system capable of diagnosing infectious blood diseases and recommending antibiotics. The CYC project[47] aimed to create a database of 'common sense' using a formal symbolic representation based on predicate logic which although it may not have achieved its grandiose aim of building a system which acts with common sense[98], its database of common sense

[1]Of course once a task can be shown to be performed without requiring real intelligence the resulting system which performed this task ceases to meet the definition – hence one reason for the sometimes perceived failure of AI

rules has been used for providing a natural language interface to databases in commercial applications[19].

In contrast to symbolic AI, neural networks, or, more precisely artificial neural networks, take a more biologically inspired approach to AI and attempt to model the neurons of the brain. It was originally hoped that intelligence would emerge from systems that modelled a significant number of neurons. Although the more ambitious hopes for neural networks may not have been realised the field has matured into providing a set of very practical and powerful tools for solving the specific problem of pattern recognition as described in the previous section. Neural networks have been widely applied to multiple problem domains including financial[59], engineering[57], medical[22], and computer games domains[82].

1.2 Rule Extraction From Artificial Neural Networks

Andrews[4] gives an excellent overview of the importance of rule extraction from neural networks. Making a neural network transparent has several benefits. Users of symbolic AI systems, such as Case-Based Reasoning[92], Expert Systems[36], and Decision Trees[65], benefit from explicit declarative explanations of the systems reasoning and decision making processes. This explanation often includes a step-by-step explanation of the decision process, and it has been shown that this level of detail is often required for user acceptance[20]. In contrast users of neural networks are denied these benefits because of the opaqueness of neural networks. Safety critical systems seldom use neural networks because of their lack of transparency. In addition, a safety critical system needs to be able to be interpreted unambiguously. Increasingly, software systems are required to be verified to be correct or match the system specification. A neural network because of its black box nature is difficult to verify, but it is possible to verify the rule sets extracted. The extracted rules can then be manipulated using logic to verify a system meets the system requirements.

1.3 Thesis Aims and Objectives

Aims

The aim of this research project is to create a series of algorithms which will meet the following criteria:

- extract rules from artificial neural networks in both classification and regression domains;
- create rules that are easy to comprehend;
- outperform direct decision tree induction;
- be widely applicable to a wide range of different artificial neural network architectures;
- will not require specialist training algorithms.

Objectives

To achieve the above aim the following objectives will be investigated:

- research and review the pattern recognition problem with respect to classification and regression domains;
- identify by examining artificial neural networks and their architectures and training algorithms why the knowledge within an artificial neural networks is difficult to interpret;
- compare and contrast artificial neural networks with decision trees to identify the strength and weaknesses of both approaches;
- review current rule extraction algorithms to identify the current strengths and weaknesses of these algorithms;
- develop new algorithms to overcome the limitations identified in the review of current rule extraction algorithms;

- implement new algorithms for rule extraction;
- implement an evaluation methodology for comparing the performance of rule extraction algorithms on multiple datasets;
- analyse the new algorithms on synthetic datasets to gain insight into the operation of the component parts of the algorithm;
- evaluate the algorithms on real-world datasets to show the real-world applicability of the algorithms.

1.4 Thesis Overview

Chapter 2 will review artificial neural networks. The review will concentrate on the multilayer perceptron architecture and error backpropagation, which is the most popular training algorithm for this architecture. Chapter 3 examines decision trees and the information theory approach to growing decision trees will be examined. Chapter 4 examines the rule extraction task and reviews current rule extraction algorithms, highlighting the deficiencies addressed by the new algorithms presented in the subsequent chapters. Chapter 5 will introduce, ExTree and ExLMT, new rule extraction algorithms for classification problems. In addition to introducing these algorithms this Chapter will analyse the algorithms on synthetic datasets. Chapter 6 will give the results of an empirical evaluation of these algorithms using real-world datasets. Chapter 7 will present the ExMT algorithm which extracts rules in the form of model trees from artificial neural networks in regression-based domains. This chapter will also give the results of an empirical evaluation of ExMT on real-world regression datasets. Chapter 8 will conclude the thesis summarising the earlier chapters and giving details of potential future research directions.

CHAPTER 2

Artificial Neural Networks

The field of (artificial) neural networks consists of a large collection of information processing models and techniques that were originally inspired by biological nervous systems such as the human brain. A biological nervous system is based around a collection of interconnected neurons. Figure 2.1 shows a typical biological neuron.

Neurons communicate using electrical pulses. A neuron receives pulses at its dendrites and sends pulses down its axon. The electrical pulse starts at or near the cell membrane then propagates down the axon to the other neurons. A synapse is the connection between the firing neuron at its axon and the receiving neurons dendrites. These synaptic connections can either impose an excitation or inhibition effect on the pulse. A typical neuron is constantly receiving thousands of pulses sent from other neurons, with the strength of the pulses regulated by the synapses. The cell body acts as a leaky integrator of these pulses and if the sum of these pulses reach a threshold limit then the neuron itself fires, sending a pulse on to other neurons.

Biological neurons and neural networks are one of the most complex naturally occurring systems and there are many aspects that the above description does not take into account. For example, brain chemistry such as hormonal changes can affect brain function. However, most of the research into artificial neural networks is based on this model. The human brain has an estimated 100 billion neurons and 60 trillion synaptic connections[96], so, although an individual neuron may seem relatively simple it is the size and complexity of the interconnected network that makes the complex animal behaviour that is

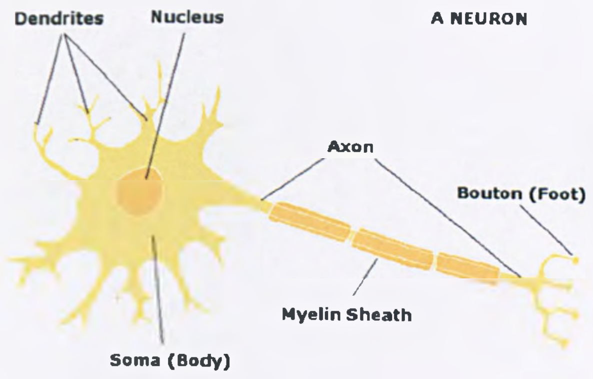
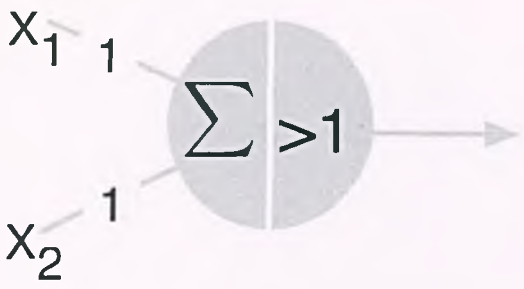


Figure 2.1: A biological neuron showing the dendrites at the top which receive pulses from the other connected neurons.

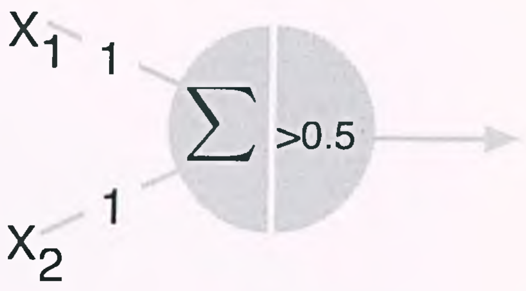
observed in the natural world, and even human consciousness, possible.

Although often considered a new field, intrinsically linked to the computer age, neural networks are rooted in the purely logical simulations of biological neurons created by McCulloch and Pitts[50]. Neural networks are based around a number of individual models of neurons arranged in a network. These artificial neurons accept a number of *weighted* inputs and process this input in some way to produce an output. The weight on the input is used to model the synaptic inhibitory or excitation effect. It is the value of these weights that determine the function of the neural network. Figure 2.2(a) models the logical AND operator such that given two binary inputs, $X \in \{0, 1\}^2$, the output will be 1, if and only if X is equal to $(1, 1)$. Figure 2.2(b) models the OR operator which results in 1 if either or both inputs has the value 1.

Minsky[53] showed that the processing power of a single (artificial) neuron had a substantial limitation: classification tasks that could not be solved by a single decision surface could not be solved by a single neuron. For example, a single neuron could not implement the logical XOR operator. Therefore, a neuron is limited to problems which are linear separable. For a two input classification task this means a single line must be able to separate the two classes.



(a) A neuron implementing the AND function.



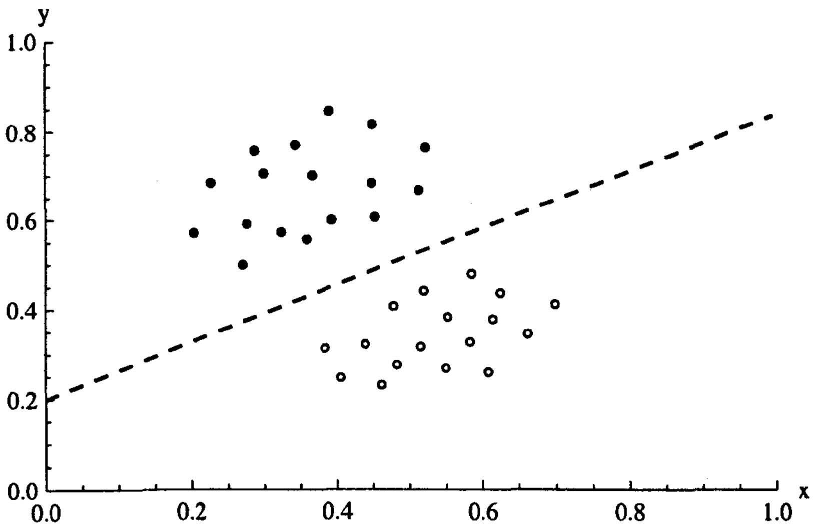
(b) A neuron implementing the OR function.

Figure 2.2: Artificial neurons implementing the AND and OR logical functions.

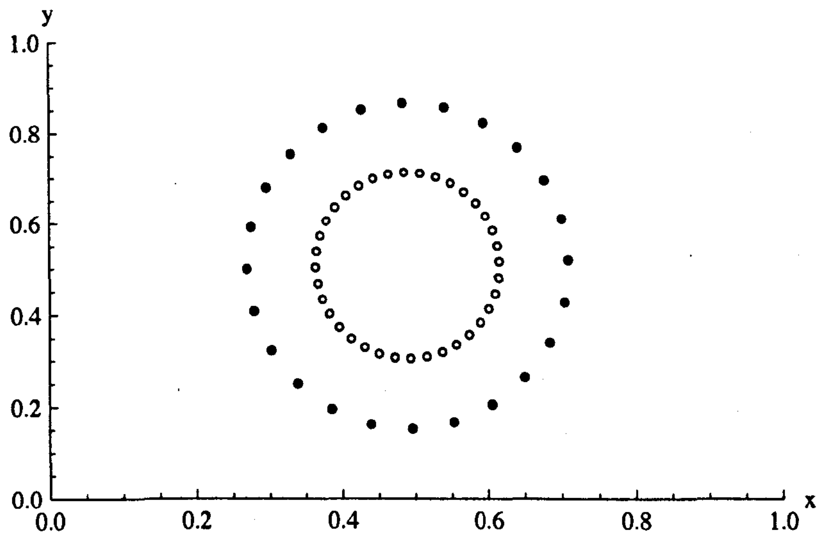
In higher input dimensions the classes must be separable by a single hyperplane. Figure 2.3 illustrates linear-separable and nonlinear-separable problems. In Figure 2.3(a) the two classes are separable by a single straight line as indicated by the dashed line and is therefore linearly separable. In Figure 2.3(b) the points are arranged in two concentric circles precluding a single line separating the two classes and therefore represents a nonlinear-separable problem.

The inability of a single neuron to separate nonlinear-separable classes meant that neural networks were not practical for pattern recognition tasks. However, it is possible by arranging multiple interconnected neurons together in an architecture known as a multilayer perceptrons(MLP) to solve nonlinear-separable classification problems. Although this was known at the time when the limitations of single neurons were being recognised[53], it was not until the discovery of the error back-propagation by gradient descent method[94, 71] allowing MLPs to be trained that they became useful. It is this type of model that will be used in this thesis.

A typical two layer MLP is shown in Figure 2.5. For each layer in the neural network a set of weights is required. The matrix **W** represents the in-



(a) Linearly separable.



(b) Nonlinear separable.

Figure 2.3: Linear and nonlinear separably pattern spaces.

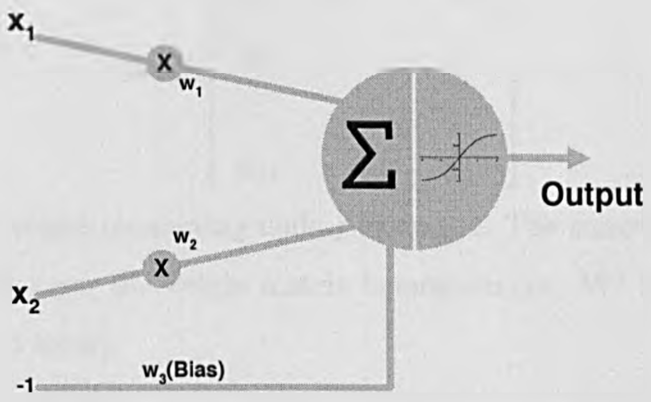


Figure 2.4: An artificial neuron with inputs $X_1 and X_2$ and bias with weights W_1, W_2, W_3.

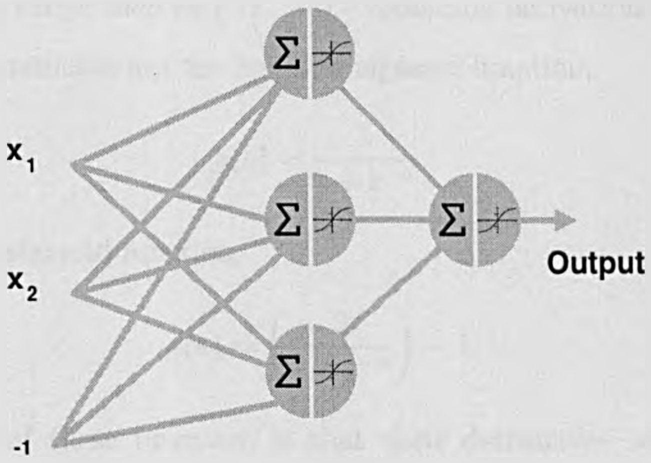


Figure 2.5: A two layer MLP with two inputs, 3 hidden nodes and a single output node.

terconnecting weights between layers and is normally represented in the matrix form,

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1i} \\ w_{21} & \ddots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ w_{j1} & \cdots & \cdots & w_{ji} \end{bmatrix}$$

where $w_{ji}$is the weight connecting node j to node i. The superscript is used to distinguish which layer the weight matrix belongs to (i.e. $\mathbf{W}^1$ is the first layer, $\mathbf{W}^2$ is the second layer).

The neuron performs two processing steps. First, it calculates the net input, a, which is simply the inputs, z, to the neuron multiplied by the corresponding weights,

$$a_j = \sum_i w_{ji} z_i. \tag{2.1}$$

Second, the output for the neuron is obtained by applying a nonlinear activation function, $g(\cdot)$ to the net input. The activation function must be nonlinear to allow the network to approximate nonlinear functions. In addition for back-propagation learning, the activation function must be differentiable and have a bounded output range such as $[-1 \ldots 1]$. Common activations functions that meet these requirements are the unipolar sigmoid function,

$$g(a) = \frac{1}{1+e^{-a}}, \tag{2.2}$$

and the bipolar sigmoid function,

$$g(a) = \left(\frac{2}{1+e^{-a}}\right) - 1. \tag{2.3}$$

The advantage of these functions is that their derivatives can be expressed simply in terms of the activation function itself, giving

$$g'(a) = g(a)\left(1 - g(a)\right), \tag{2.4}$$

for the unipolar sigmoid function and

$$g'(a) = \frac{1}{2}\left(1 - g(a)^2\right). \tag{2.5}$$

for the bipolar sigmoid function.

Training is the process of selecting a set of weights that produce a desired network function $\hat{C}$. This is done by presenting to the neural network a set of instances with known classifications and adjusting the weights until the error between the output of the neural network and the true classification is minimised.

The most popular training algorithm is error back-propagation using gradient descent, which is often, but somewhat misleadingly, referred to simply as back-propagation. Using summed square error (SSE) as the error measure the error for pattern n and the whole dataset can be written as

$$E^n = \frac{1}{2}\sum_{k=1}^{c}(y_k - t_k)^2 \tag{2.6}$$

$$,E = \sum_n E^n, \tag{2.7}$$

where $y_k$is the output of the network and t_k is the *a priori* known classification of that pattern.

The derivative of E^n with respect to the weights w_{ji}, is

$$\frac{\partial E^n}{\partial w_{ji}}, \tag{2.8}$$

then by application of the chain rule this can be written as

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{2.9}$$

where a is the net input. The activation with respect to the weights can then be written as

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \tag{2.10}$$

Substituting Eq. (2.10) into Eq. (2.9) gives

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial a_j} z_i. \tag{2.11}$$

The $\frac{\partial E^n}{\partial a_j}$ term is usually referred to as the error term and is represented as δ giving

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j z_i. \tag{2.12}$$

For the output layers this can be calculated as

$$\delta_k = \frac{\partial E^n}{\partial y_k}\frac{\partial y_k}{\partial a_k} = g\prime(a_k)\frac{\partial E^n}{\partial y_k}. \tag{2.13}$$

The δs for the neurons in the hidden layer become

$$\delta_j = \sum_k \frac{\partial E^n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = g\prime(a_j)\sum w_{kj}\delta_k, \tag{2.14}$$

where the sum runs over all k neurons to which neuron j sends connections. This has the effect that δ_j, (the 'blame') assigned to the neuron j is passed back to the neurons that connect to it, in proportion to how much it contributed to the output of neuron j.

Using the unipolar activation given in Eq. (2.2), and SSE given in Eq. (2.6) for the error function the δs becomes

$$\delta_k = y_k - t_k, \tag{2.15}$$

$$\delta_j = z_j(1-z_j)\sum_k w_{kj}\delta_k, \tag{2.16}$$

where the sum runs over all neurons k to which neuron j sends connections.

Having obtained the derivatives of the error with respect to the weights, the gradient descent method can be used to find a set of weights to minimise the error. Starting with a random set of weights[1], the weights are updated such that the E, as defined in Eq. (2.6), is decreased. This is achieved by taking a small step (in weight space) in the direction that results in the greatest descent, which will be the negative of the gradient that was just found. Therefore

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \Delta\mathbf{W}^t \tag{2.17}$$

where

$$\Delta w^t_{ji} = \eta\delta_j x_i. \tag{2.18}$$

This update to the weights is recalculated for each instance in the training data. After all the instances in the training set have been processed (1 epoch of

[1]Although small random weights are sufficient, initialising the weights using algorithms such as Nguyen and Widrow[58] will increase the convergence speed.

training) if E is below the predetermined limit $E_{\max}$ then training is complete else further epochs of training are repeated until E falls below $E_{\max}$.

The algorithm described is the standard algorithm of MLP training but there have been numerous improvements proposed that can be made to the algorithm[9, 31, 69, 23].

The basic back-propagation algorithm has three main problems:

- It can be very slow to converge to a solution, taking several thousand epochs to converge to a set of weights that will solve even simple problems like representing the XOR function.
- Gradient descent is not guaranteed to converge to the optimal solution, and can get 'stuck' in local minima as shown in Figure 2.6.
- It can overfit the training set reducing the neural networks ability to generalise to previously unseen instances.

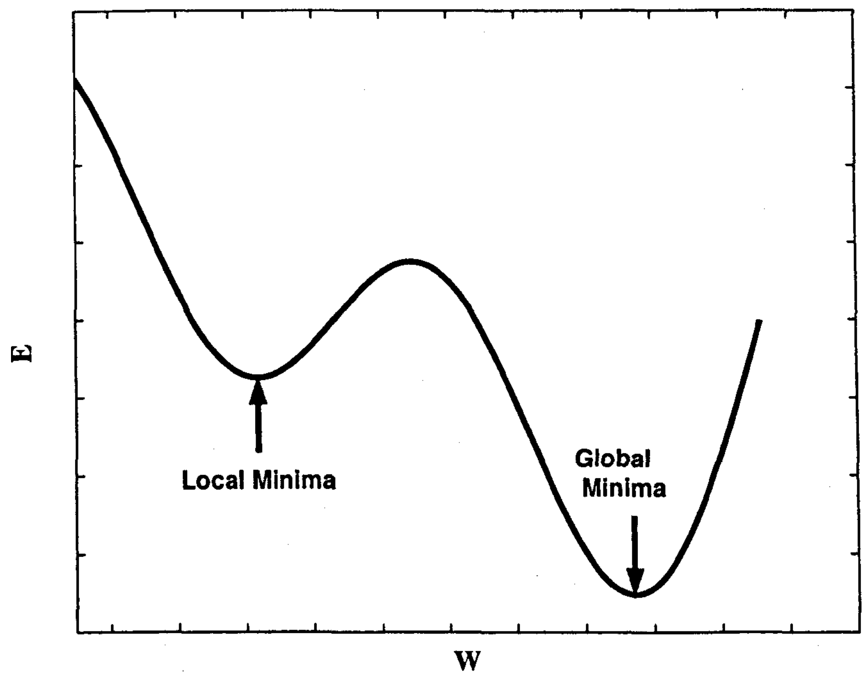


Figure 2.6: Minima on the Error Surface: E plotted against $\mathbf{W}$

The speed of convergence is dependent on the value of the learning rate η in Eq. (2.18). If η is small enough then the algorithm should converge to a minima in a smooth descent, but this will take an unpractical number of iterations. Increasing η will speed the descent but may result in an unstable descent that may 'overshoot' the minima and actually increase E. It can be shown analytically[40] that the optimal value for η decreases in proportion to the time-step such that

$$\eta^t \propto \frac{1}{t}. \tag{2.19}$$

However for practical applications this results in convergence taking too long. Therefore, η tends to be set at a higher value using trial and error. A related improvement is the inclusion of a momentum term[60] to the delta, such that Eq.(2.17) becomes

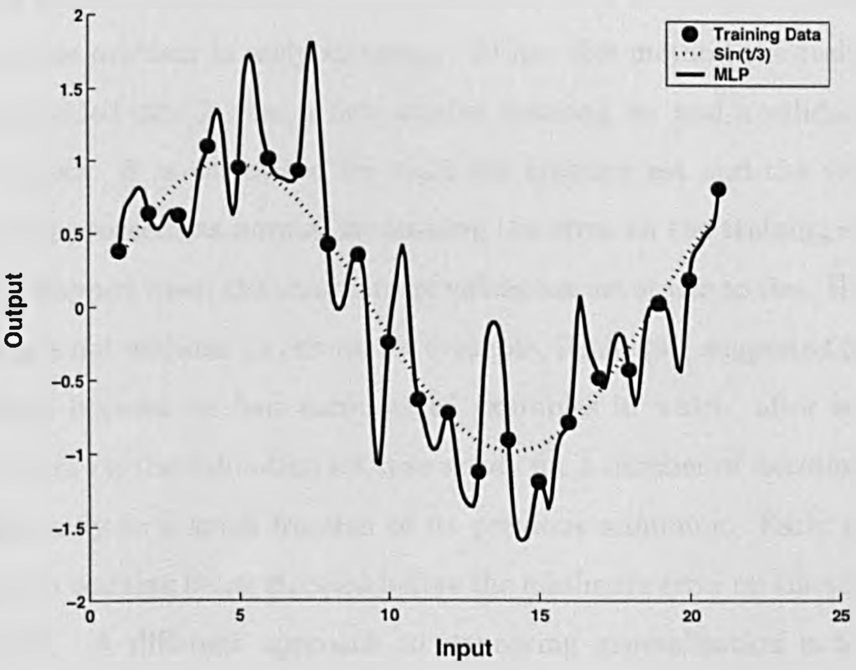
$$\mathbf{W}^{t+1} = \mathbf{W}^t + \Delta\mathbf{W}^t + \mu\mathbf{W}^{t-1}. \tag{2.20}$$

The additional term added to the delta includes a fraction, μ, of the previous delta. This has the effect of altering the step size according to the error surface. When the error surface is constantly downhill in the same direction the delta term will 'gather momentum' increasing the step size, but when the surface is uneven or 'bumpy' the momentum term will dampen the rate of descent resulting in smaller, more cautious steps. Fausett[23] reports an experiment where adding a momentum term speeded up convergence from 387 epochs to 38 epochs. Riedmiller[68] took a more radical approach to the problem of step size and proposed the Resilient Back-propagation(Rprop) algorithm which attempted to 'eliminate the harmful influence of the size of the partial derivative on the weight step.' In Rprop the derivative is used only to indicate the direction of the step with the size of the step being calculated so that it is proportionate to the time-step. This is somewhat similar to the update of the learning rate previously discussed.

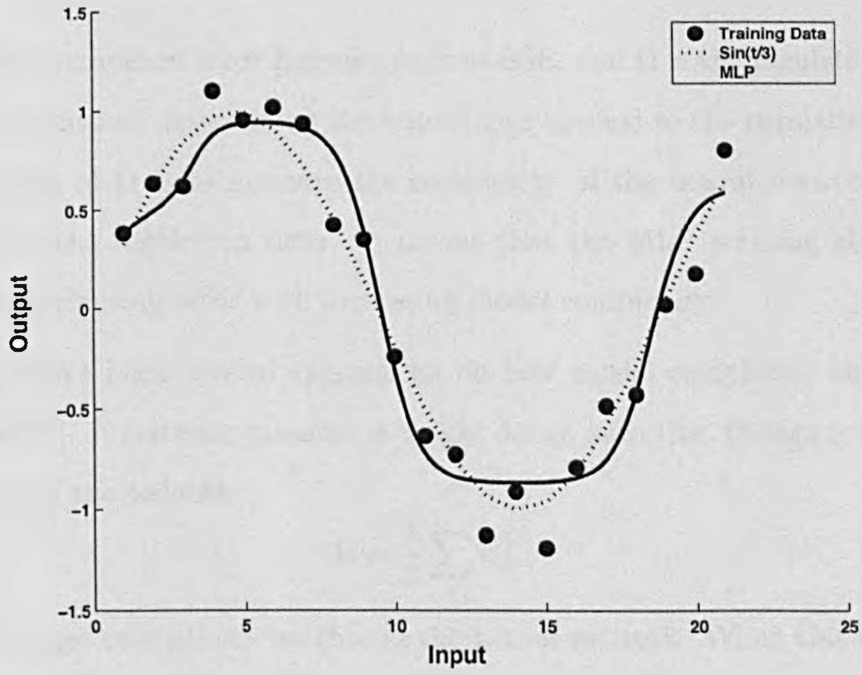
However, the biggest improvement in terms of speed of convergence has come from the realisation that the back-propagation method of finding the derivatives can be combined with optimisation techniques other than gradient descent, such as Quasi-Newton, conjugate gradients, and Levenberg-Marquardt (L-M) techniques. For example, Demuth[21] showed that the L-M algorithm was six times faster on the well-known diabetes dataset and was 25 times faster learning the Sine function. Masters[49, 48] gives details on how these more advanced algorithms can be applied to MLP training. However it should be noted that the neural networks community has, in general, been slow to adopt these more advanced optimisation techniques. Reasons for this reluctance include the gradient descent technique being 'tried and tested' and relatively simple to implement, but also because it is slightly closer to the biological neuron model discussed earlier in this Chapter.

As already discussed a potential problem of training MLPs is that the algorithm can converge to a local minima instead of the global minima. There are currently no known ways of eliminating this problem entirely. Although approaches that change the error landscape such as changing the error function in Eq. (2.6) to a cross-entropy based measure[39] and scaling the individual input vectors can lessen the problem but not eliminate it. An alternative approach is to attempt to test whether the minima found is really the global minima by trying to 'escape' from it. Masters[49] shows how a form of simulated annealing can be used to achieve this. In practice the most common way to avoid sub-optimal solutions is to retrain the MLP with different initial weights and choose the MLP with the lowest E.

A further problem with the standard back-propagation algorithm is that if E is minimised to its lowest point using the training set then the neural networks will most likely have overfitted the training set. To show the effects of overfitting, a dataset was created by sampling 21 points from the function $y = \sin(3/t) \pm v$ where v is a uniform random number between 0 and 0.25, with t being $1, 2 \ldots 21$. A MLP with 20 neurons in the hidden layer was trained using this data. Figure 2.7(a) shows the extreme overfitting that occurred. Although the E for the dataset is zero the neural network failed to interpolate or generalise to points outside the original dataset. Figure 2.7(b) shows a MLP with 5 neurons in the hidden layer. Although the error on the training data will be higher for this neural network its ability to generalise to unseen instances is



(a) 20 Hidden Neurons



(b) 5 Hidden Neurons

Figure 2.7: Overfitting Comparison of a MLP with 20 and 5 Hidden Neurons

far greater than the previous 20 neuron network. The most common approach to solving this problem is early stopping. To use this method the training set is further divided into 2 sets: a new smaller training set and a validation set. For each epoch, E is calculated for both the training set and the validation set. Training proceeds as normal minimising the error on the training data but training is stopped when the error for the validation set starts to rise. However/ the scheme is not without its critics, for example, Ripley[69] suggested that this is dangerous because he had encountered examples in which, after an initial drop, the error on the validation set rose slowly for a number of iterations, then fell dramatically to a small fraction of its previous minimum. Early stopping may result in training being stopped before the minimum error on the validation set is found. A different approach to improving generalisation is to add a regulation term to the error function, such that Eq. (2.6) is replaced by

$$\tilde{E} = E + v\Omega, \tag{2.21}$$

where E is a standard error function such as SSE, and Ω is the regulation term and v is a constant determining the importance applied to the regulation term. The purpose of Ω is to measure the complexity of the neural network. The addition of the regulation term, Ω, means that the MLP training algorithm balances minimising error with increasing model complexity.

There have been several suggestions on how model complexity should be measured[31]. A common measure is weight decay, such that Omega is the sum of squares of the weights,

$$\Omega = \frac{1}{2}\sum_i w_i^2. \tag{2.22}$$

where i ranges over all the weights in the neural network. When this measure is used it has the effect that when $\tilde{E}$ is minimised for a given value of E the weights will decay towards zero, hence the name weight decay.

2.1 Analysis of Neural Network Training

This section presents an analysis of how artificial neural networks learn a given pattern classification task or approximate a given function.

2.1.1 The XOR Classification Task

The XOR classification is of historical importance to artificial neural networks because it cannot be solved by a single layer of perceptrons as demonstrated by Minsky[53]. However, it can be solved with a multilayer perceptron with a single hidden layer. The task is to classify a two attribute pattern according to the eXclusive-OR function as illustrated by the truth table in Table 2.1. Figure 2.8 shows the pattern space for the XOR problem. Notice that the two classes are not linear separable. A MLP of the type in Figure 2.5 can be trained to solve

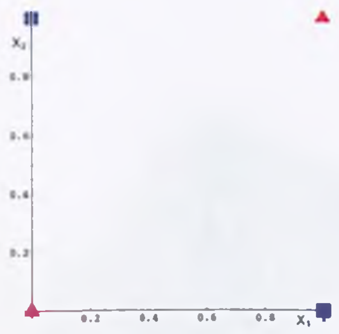


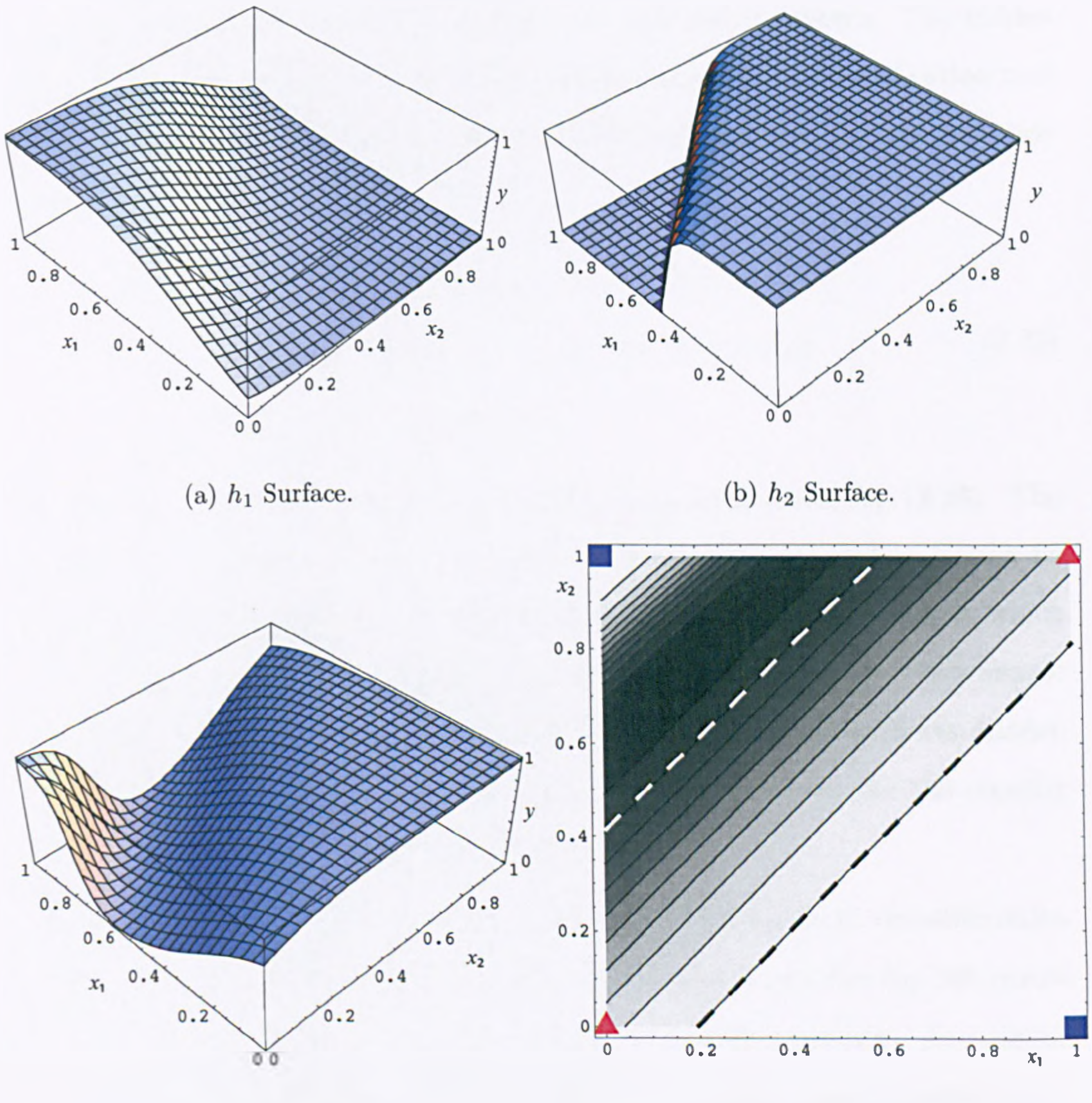
Figure 2.8: Pattern Space for XOR function.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1: Truth table for the XOR function.

the XOR classification problem. The hidden layer consisting of two neurons h_1 and h_2 each represent a nonlinear function of the pattern space. Figure 2.9(a) shows the output from h_1 which gives a value of 1 at (1,0) before gradually sloping away to give a value 0 at (0,1). Figure 2.9(b) shows the output from h_2 which gives a value of 0 at (1,0) before steeply rising to give a value 1 at (0,1).

Figure 2.9(c) shows the surface created by the output node that combines the h_1 and h_2 hidden nodes. Figure 2.9(d) shows the pattern space with the decision boundaries formed by the hidden neurons. Because the hidden neurons used smooth continuous functions the decision boundaries are not sharp, but if a classification threshold of 0.5 is applied then the boundaries become distinct as shown by the dotted lines in Figure 2.9(d).



(a) h_1 Surface.

(b) h_2 Surface.

(c) Output Surface.

(d) Decision boundaries in pattern space.

Figure 2.9: Decision surfaces and boundaries for the XOR problem formed by the hidden and output neurons of the MLP.

2.2 Hidden Layer Transform

The previous section looked at neural networks solving the XOR problem from the perspective of the hidden neurons being decision planes in pattern space. In this section an alternative but equivalent view will be shown. The hidden layer (or layers) can be seen as a transformation such that the classification task becomes linearly separable. Consider the following two-input (x_1, x_2) two-class classification problem,

$$y = \begin{cases} 1 & \text{, where } x_2 > 2x_1 + 1 \wedge x_1 < 0 \\ 1 & \text{, where } -x_2 < 2x - 1 \wedge x > 0 \\ 0 & \text{, elsewhere .} \end{cases} \tag{2.23}$$

The dataset was 10,000 data points uniformly sampled from Eq. (2.23). The dataset in pattern space for this problem is shown in Figure 2.10. As can be seen from the pattern space, the task is to separate the red class 1 points, which form a triangle, from the remaining blue class 0 points. Although a very simple classification task it is clearly not linearly separable. A MLP with two hidden nodes h_1 and h_2 was trained for 125 epochs. The error over the 120 steadily reduced towards near 0 as shown in Figure 2.11.

Figure 2.12(a) shows how before training the hidden layer transformation starts in a random state which squashes all the points into the top left corner with the decision plane not separating any of the data points. As training progresses, shown in Figures 2.12(a)- 2.12(f), the hidden layer transformation 'fans out' the points simultaneously keeping the class 1(red) points in the top left corner and moving the class 0(blue) points away from the top left corner. At the same time the decision plane of the output neuron moves up to the top-left hand corner to separate these now linearly separable classes. This shows how a problem which is not linearly separable in $x_1 \times x_2$ can be transformed by the hidden layer into a problem that is linearly separable in $h_1 \times h_2$. In this illustrative example the number of inputs and outputs was limited to two for

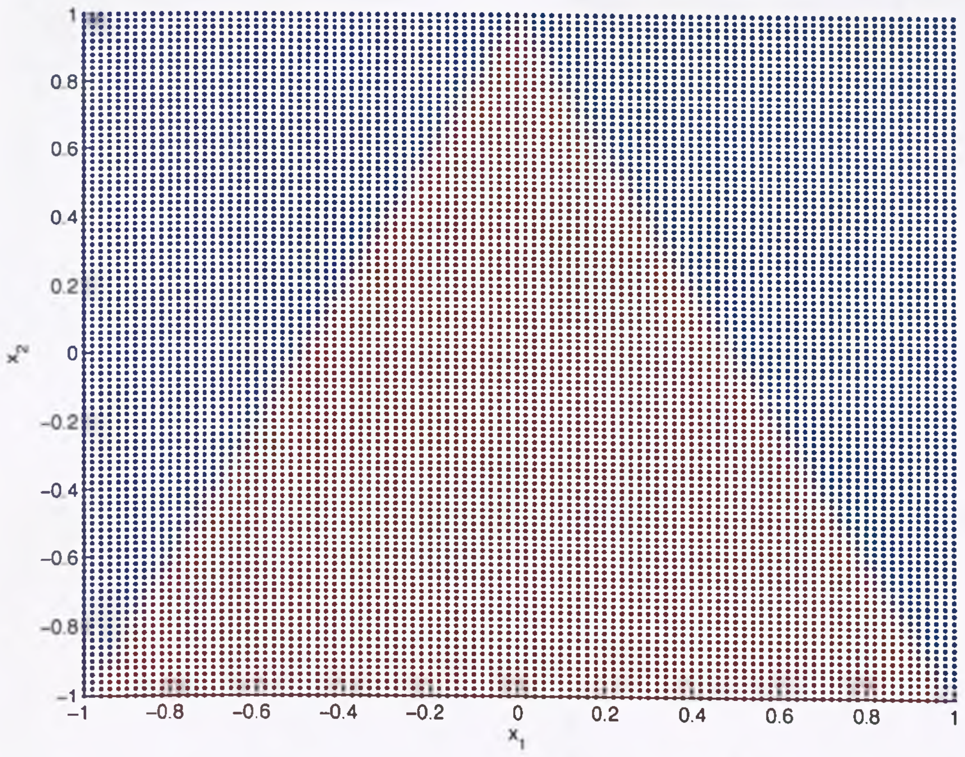


Figure 2.10: Pattern space for the triangle problem.

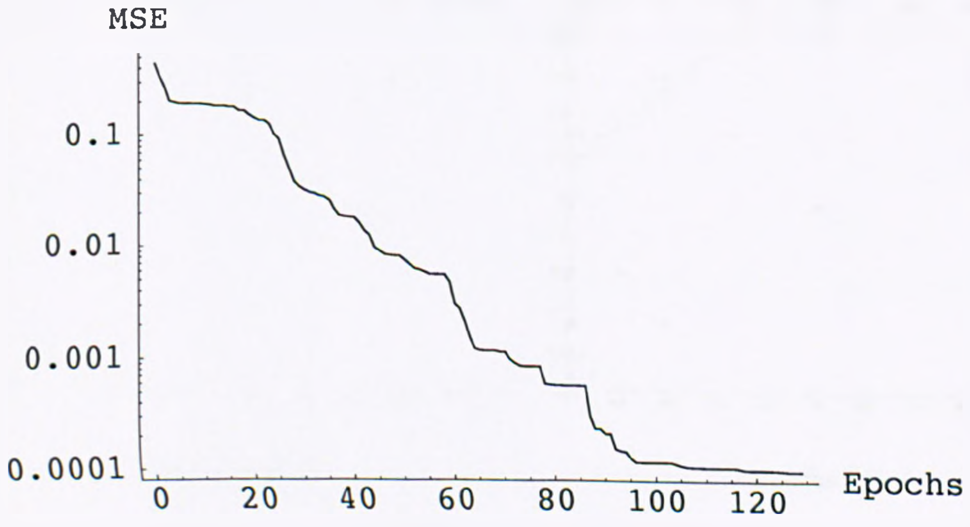
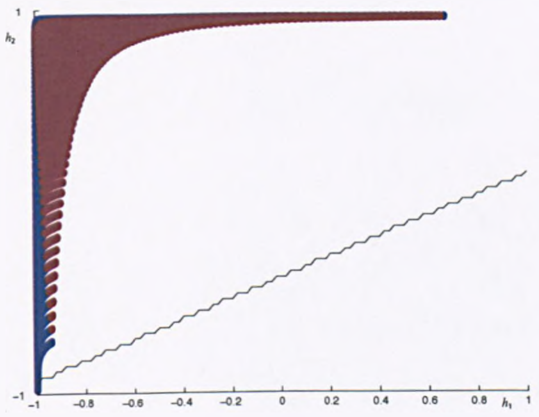
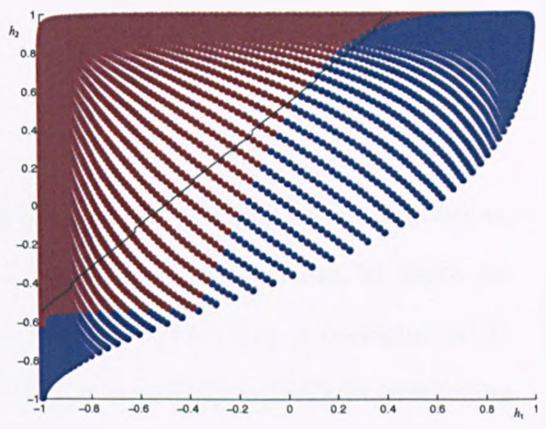


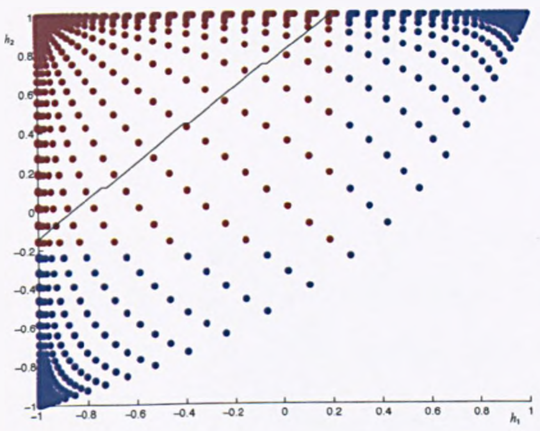
Figure 2.11: Mean squared error over time for the triangle problem.



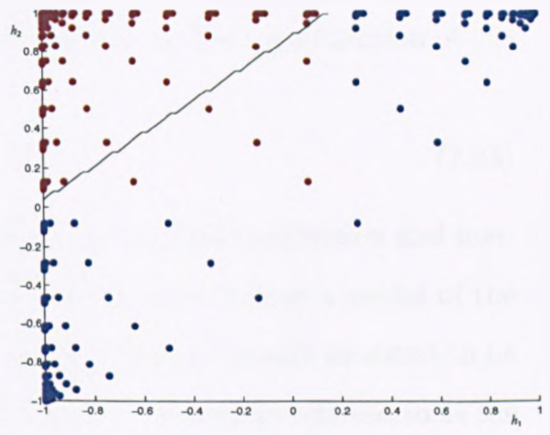
(a) Epochs 0



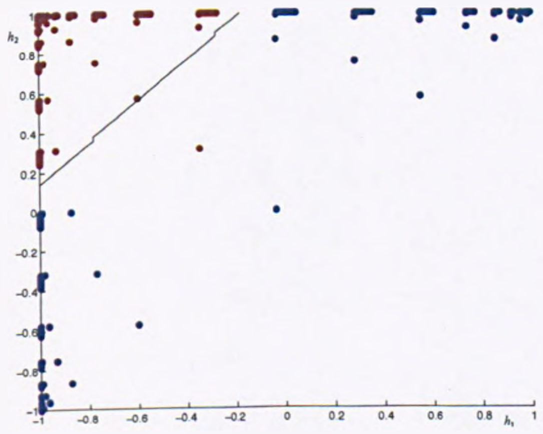
(b) Epochs 25



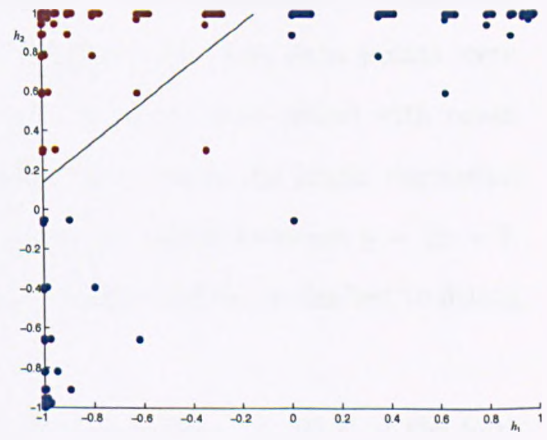
(c) Epochs 50



(d) Epochs 75



(e) Epochs 100



(f) Epochs 125

Figure 2.12: As the neural network passes through an increasing number of training epochs the hidden layer separates the blue and red points until they are linearly separable.

ease of plotting but, of course, both can be of a higher dimension.

2.3 Regression Problems

The previous examples examine how MLPs can be applied to classification, but the other significant use of MLPs is function approximation to solve regression problems. Regression problems are those that have a continuous($\mathbb{R}$) valued output. Real world examples of regression problems include predicting house prices, stock prices, and population growth. The input, X, can also be continuous($\mathbb{R}$) or a nominal valued attribute as in the classification task. Mirroring Eq. (1.1), the regression task can be specified as finding a function of the form,

$$\hat{f} : \mathscr{X} \to \mathbb{R}. \tag{2.24}$$

A useful categorisation of regression problems is into linear regression and nonlinear regression. Linear regression can be expressed as finding a model of the form $Y = X\beta + \varepsilon$ where ε models the 'error' in Y and is usually assumed to be Gaussian distributed. The value to be predicted, Y, is usually referred to as the dependent variable and the values used to make the prediction(attributes) are referred to as the dependent variable. A simple linear regression problem with a single independent variable is shown in Figure 2.13. The data points were sampled from the function $y = 2x + 3$ with Gaussian noise added with mean 0 and standard deviation of 1.5. The dashed line shows the linear regression fit with least squares and the solid line shows the target function $y = 2x + 3$. Linear regression fitted using least squares performs well but is limited to fitting linear models.

Real world phenomena are often nonlinear in nature. A MLP of the form $y = g^1(\sum g^2(w_i x_i))$ where $g^1(\cdot)$ is the activation function for the output layer and $g^2(\cdot)$ is the activation function for the hidden layer, provides a model that can approximate the regression task as given in Eq. (2.24) with some practically-satisfiable assumptions[95]. The back-propagation training method previously described can be used to train the neural network on regression problems. The

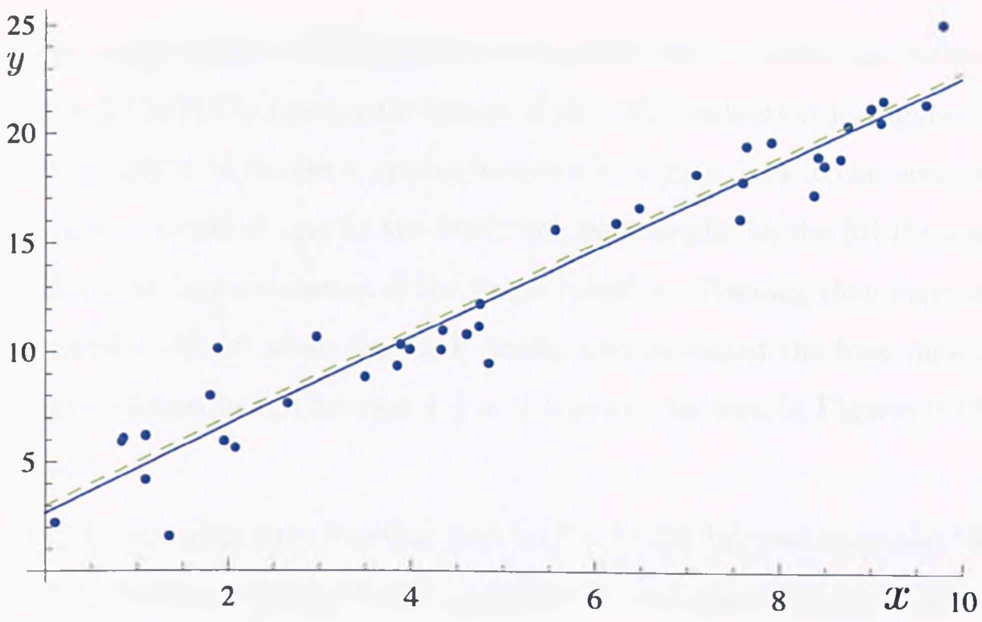


Figure 2.13: An example linear regression on points sampled from the function $y = 2x + 3 + \epsilon$.

output function is normally changed to a linear function such as the identity function $g(a) = a$. This simplifies the calculation of the derivative for the output layer, because $g\prime(a) = 1$, so that Eq. (2.13) simplifies to $\frac{\partial E^n}{\partial y_k}$. To demonstrate how a neural network can be used to solve a regression problem, a neural network was trained to approximate the function,

$$y = 5\sin(x) + \sin(3x). \tag{2.25}$$

A dataset was created by linearly sampling the function at 121 points. Initial experiments were carried out to find a suitable architecture and parameters for the neural network. These experiments suggested a MLP with one hidden layer of 7 neurons could be trained to solve this problem. The hidden neurons used an unipolar sigmoid function(Eq. (2.2)). The single output neuron used the Identity function($g(a) = a$). Figure 2.14 shows how the MLP training reduced the MSE over the 120 epochs.

Figure. 2.15 shows how the MLP approximation becomes closer to the target function(Eq. (2.25)) during the training. Starting with the random untrained MLP shown in Figure 2.15(a) with a MSE of 13.56, after being trained for 120 epochs the error of the MLP approximation is reduced to 0.00054 as shown in the Figure 2.15(f). Comparing the graph of the MSE reduction in Figure 2.14 with the sequence of function approximations in Figure 2.15 it can seen how a after only a couple of epochs the MSE dropped rapidly as the MLP quickly provided a gross approximation of the target function. Training then plateaued until around epoch 80 when the MLP finally approximated the finer detail of the target function in the interval $4 \leq x \leq 5$ as can be seen in Figures 2.15(e) and 2.15(f).

A MLP approximates a function such as Eq. (2.25) by combining the functions of the hidden neurons. In this case the unipolar sigmoid, Eq. (2.2), was used as the transfer function. The input weights and the output layer weights for each hidden neuron transforms the sigmoid transfer function. The output neuron then combines the hidden neuron functions into the final output function for the MLP.

Figure 2.16 shows each of the hidden neuron functions for the trained MLP and how they combine to form the final output function. In a similar manner to the analysis of the classification based neural network in Section 2.2 the regression neural networks hidden layer can be viewed as transforming the nonlinear function in $\mathscr{X}$ into a linear function which can then be fitted by the linear output neuron.

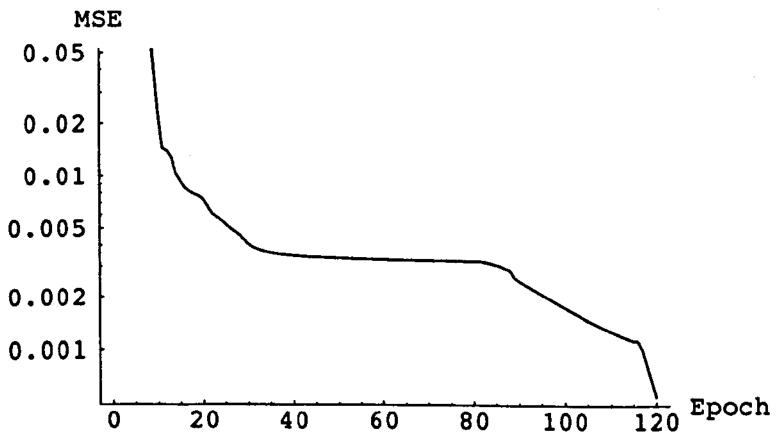
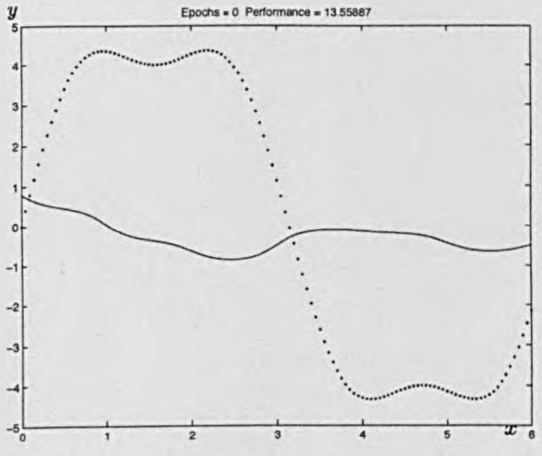


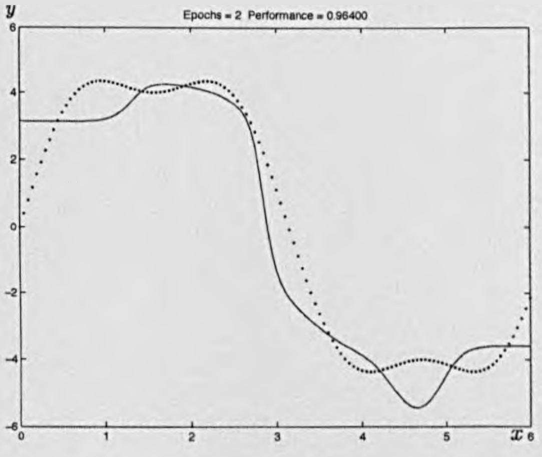
Figure 2.14: MSE reduction on $y = 5\sin(x) + \sin(3x)$ over 120 epochs.

2.4 Summary

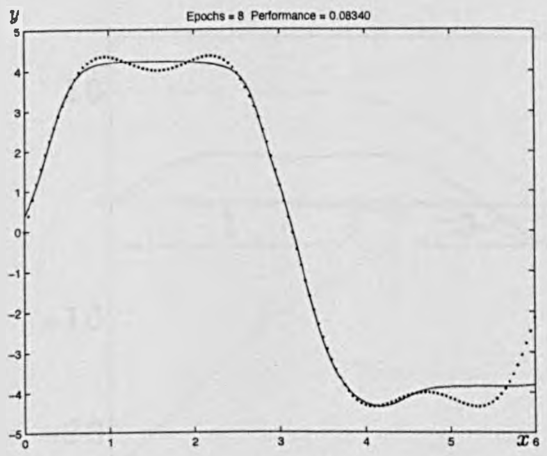
This chapter began by looking at the biological motivation behind neural networks. The artificial neuron model was examined and used to implement simple logical operators. However, the power of a single neuron is limited because it cannot solve nonlinear-separable classification problems. Therefore, the multilayer perceptron which can solve nonlinear-separable classification problems was also examined. The MLP will be the standard neural architecture used within this thesis. The gradient descent using error back-propagation training algorithm as the most popular and well-known neural network training algorithm was then derived. A review of the more advanced training algorithms used in this thesis was then given. Illustrative examples of how MLP learn and store their 'knowledge' for classification was given with an emphasis on how the hidden layer transforms the pattern space. Finally, in addition to solving classification problems, neural networks are universal approximators, so are particularly effective at solving regression problems. The regression MLP was contrasted with the classification MLP and an example of how regression MLPs can approximate a given function was demonstrated.



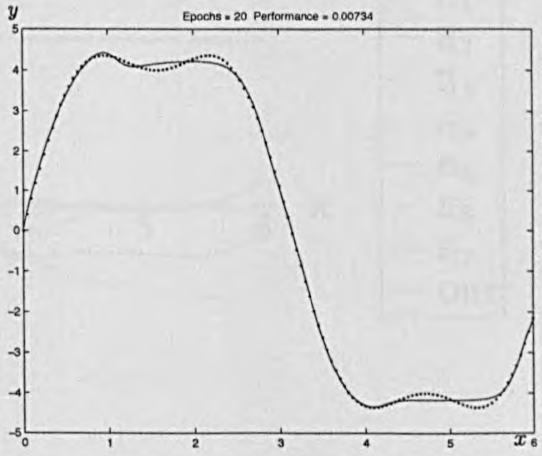
(a) Epochs 0



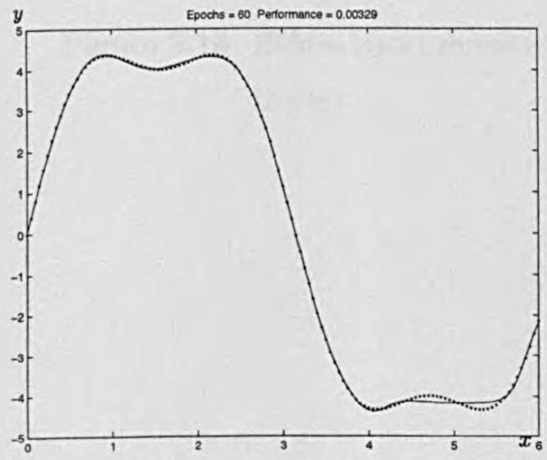
(b) Epochs 2



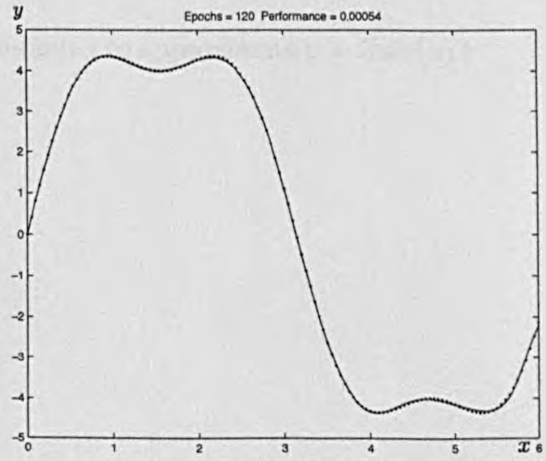
(c) Epochs 8



(d) Epochs 20



(e) Epochs 60



(f) Epochs 120

Figure 2.15: MLP function fitting of $y = 5\sin(x) + \sin(3x)$ over 120 epochs.

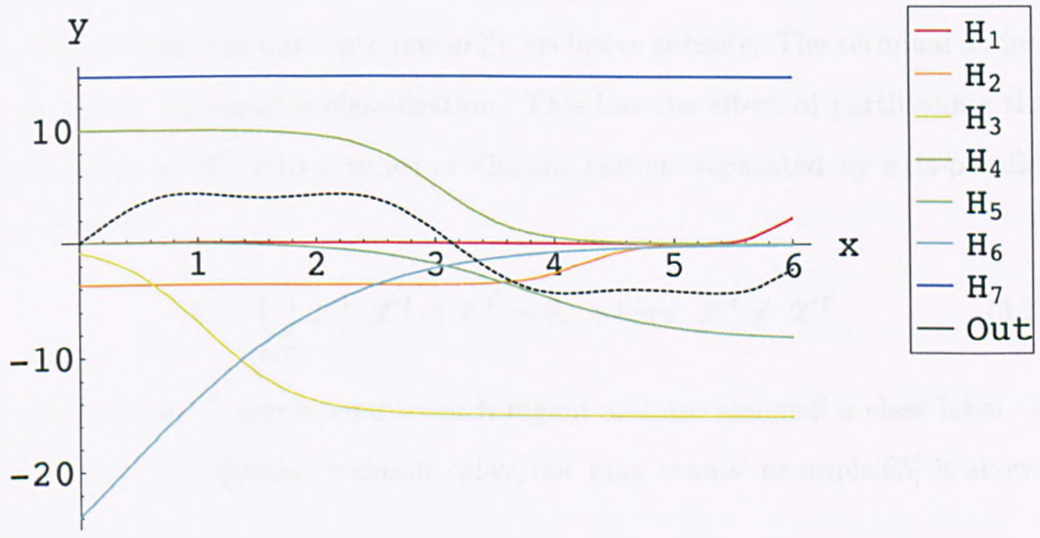


Figure 2.16: Hidden layer neurons combining to approximate $y = 5\sin(x) + \sin(3x)$.

CHAPTER 3

Decision Trees

Decision trees are one of the most widely used classifier models. Decision trees are directed acyclic graphs consisting of nodes and connections (edges) that illustrate decision rules. Each non-terminal node has an associated splitting test, which splits the data into mutually exclusive subsets. The terminal nodes, called leaves, represent a classification. This has the effect of partitioning the instance space, $\mathscr{X}$, into a series of disjoint regions separated by axis-parallel hyperplanes,

$$\mathscr{X} = \bigcup_{t \in T} \mathscr{X}^t, \mathscr{X}^t \cap \mathscr{X}^{t'} = \emptyset, \text{ where } \mathscr{X}^t \neq \mathscr{X}^{t'}. \tag{3.1}$$

The leaf nodes, T, correspond to each region and are assigned a class label. A decision tree for Quinlan's classic 'play/not play tennis' example[63] is shown in Figure 3.1.

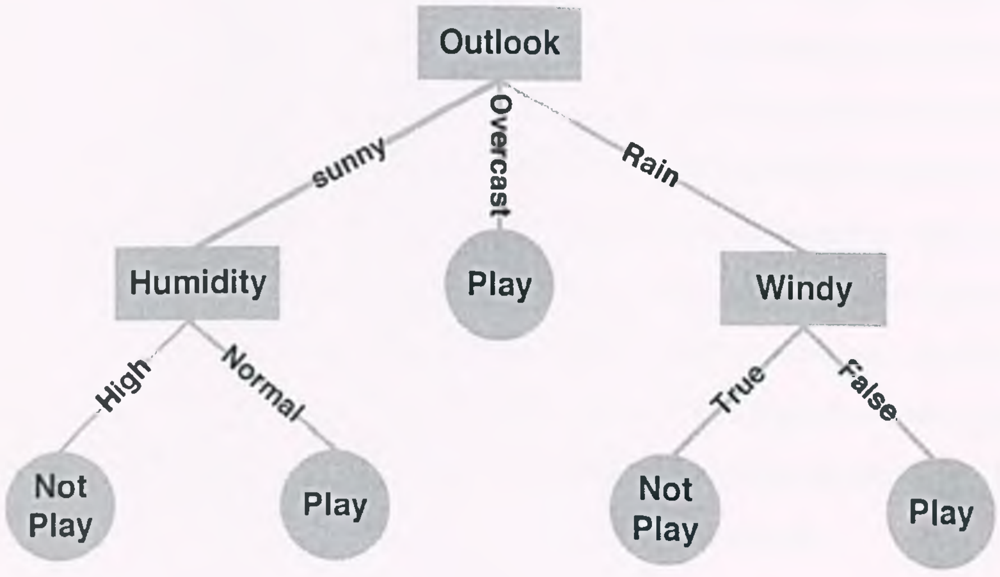


Figure 3.1: A decision tree for Quinlan's tennis problem.

Attribute	Value
Outlook	Sunny
Windy	True
Humidity	Normal

Table 3.1: An instance from the tennis dataset

To classify an instance using a decision tree: start at the root node and follow the tree down the branches according to the splitting tests until a leaf node representing a class is reached. For example, to classify the instance in Table 3.1 using the tree given in Figure 3.1, start at the root node which tests the value of the attribute *Outlook*, which, for this instance, is *sunny*. Following the leftmost branch requires a test on the *Humidity* attribute, therefore the rightmost branch, representing *normal*, should be followed which terminates in a leaf node labelled *Play* and this instance(day) should thus be classified as a good day to play tennis.

Although Decision Trees are simple to understand, the method of creating or inducing a decision tree from a dataset of examples is a nontrivial task, in fact, it has been shown to be NP complete[35]. Modern Decision Tree induction algorithms have their roots in the work of CART by Breiman[12] and ID3 by Quinlan[63]. Breiman comes from a statistical background and produced a set of decision tree algorithms based on established statistical measures such as the Gini index. Around the same time, Quinlan, who had an AI/machine learning background, was developing his own algorithm based around an information measure derived from information entropy. The CART algorithm was quickly commercialised by its authors into a data-mining tool whereas Quinlan's ID3, has become the benchmark for modern Decision Tree research.

Nearly all Decision Tree algorithms take a divide and conquer approach, using a technique known as recursive partitioning:

1. If all instances at a node all belong to the same class C then stop and label the node C.
2. If the instances do not all belong to the same class then:
    (a) Find all possible splits
    (b) Find the best possible split by applying splitting test
    (c) Label edge resulting from split
    (d) goto 1 for each subnode

The splitting test is the heart of the algorithm and measures the purity (one class dominating the set) of the subsets created by a split. Popular measures of impurity include information gain used by ID3,C4.5[65], chi-squared used by CHAID[43] and gini used by CART[12]. This thesis concentrates on information gain as it has been shown empirically[52] to be able to produce trees that compare favourably with the other main decision tree algorithms over a wide range of datasets.

ID3 and its successor C4.5 use information entropy to determine on which attribute to split. These methods select the attribute that results in the most information being obtained. Information here is not the ambiguous common notion of information, but a precise concept that was defined in Shannon's information theory[77]. The information entropy can be considered a purity measure where a set containing instances that all belong to the same class would have a value 0. Conversely, sets of instances containing a mix of classes would have an entropy value greater than 0, indicating the impurity of the set.

More formally, the information conveyed by a message can be thought of as the number of possibilities it eliminates. A message that eliminates all but a fraction p of the possibilities can be given the value $-\log_2(p)$. The logarithm is used to make the values additive and when base 2 is used for the logarithm the units of information are called bits. The average amount of information to classify a pattern in a training set, S, can be calculated as

$$\text{info}(S) = -\sum_{i=1}^{K} p\left(C_i\right) \log_2\left(p(C_i)\right) \text{ bits,} \tag{3.2}$$

where

$$p(C_i) = \frac{\text{freq}(C_i, S)}{|S|}, \tag{3.3}$$

and the freq(C_i, S) is the number of objects belonging to C_i in set S and K is the number of classes. The information gained by splitting the data can be found by calculating the average amount of information needed to classify an instance before splitting the data and subtracting the amount of information needed to classify an instance for each of the subsets created by the split. Therefore, for a split which results in N subsets, the sum of the average information of the N subsets $S = \{S_1, S_2, \ldots, S_N\}$ is

$$\text{info}_{split}(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \times \text{info}(S_i) \text{ bits.} \tag{3.4}$$

The total information gained by the split can be calculated as

$$\text{gain}(S) = \text{info}(S) - \text{info}_{split}(S). \tag{3.5}$$

Information gain has a bias towards selecting tests with many outcomes. For example, in the extreme case of a split which resulted in a unique subset for every instance, each subset would be 100% pure resulting in the maximium amount of information gain. Quinlan[65] proposed a modification to information gain giving information gain ratio. This is calculated by dividing the information gain by the information gained solely by splitting the data into the number of outcomes resulting from the test. The information gained by arbitrarily splitting a set S into N subsets is given by

$$\text{split info}(X) = \sum_{i=1}^{N} \frac{|S_i|}{|S|} \times \log_2\left(\frac{|S_i|}{|S|}\right). \tag{3.6}$$

The gain ratio of test X can thus be calculated as

$$\text{gain ratio}(X) = \frac{\text{gain}(X)}{\text{split info}(X)}. \tag{3.7}$$

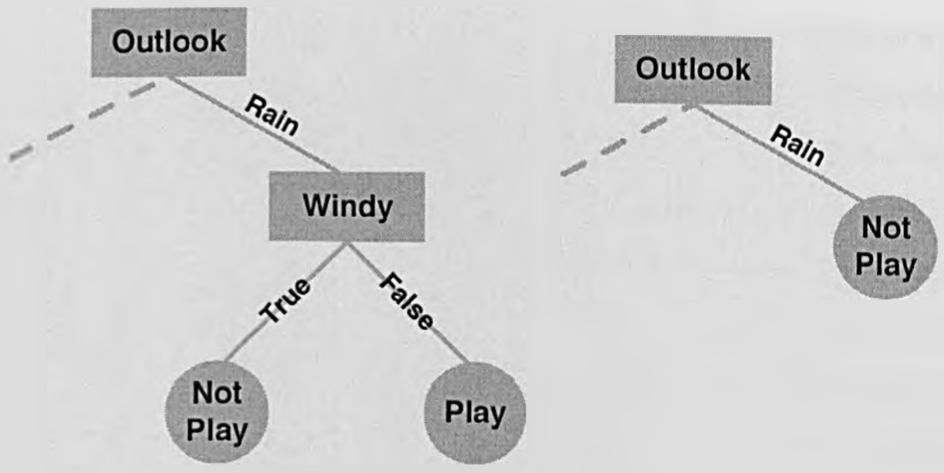


Figure 3.2: A subtree replacement operation removes the Windy subtree and replaces it with the Not Play leaf node.

The best split S^* is the split that maximises the information gain ratio

$$S^* = \arg\max_i \left(\text{gainratio}(X)\right). \tag{3.8}$$

3.1 Pruning

A flaw of the recursive partitioning algorithm is that it tends to overfit the training data. The model is too specific to the noise (outliers, anomalies and measurement error) contained in the training data. Overfitting reduces the model's ability to generalise and thus predict the classification of previously unseen instances. Overfitting also results in larger trees that reduce the comprehensibility of a model. A solution to this problem is to apply Occam's Razor to the tree in a process aptly named pruning. In the top down construction of decision trees the most general concepts are at the root and concepts become more specific towards the leaf nodes. Pruning, therefore, starts at the bottom of the tree with the most specific concepts, and estimates the difference in error of replacing each subtree with a single leaf node. Figure 3.2 shows a typical pruning operation: the *is Windy* subtree is replaced with single leaf node which classifies the instances as *Not Play.* The challenge with pruning is determining whether the pruning was beneficial or not. Any pruning will increase the error on the original training data; if it did not then the subtree would not have been grown initially. A simple solution to this problem is to set aside a subset of the training data that is to be used exclusively for pruning. An example of this approach is reduced error pruning[65]. This pruning set can then be used to evaluate the error of each potential pruning operation. This approach has the significant disadvantage that not all the information in the training set is used to grow the tree initially. The second approach is to use all the training set to grow the tree and then use an estimate of the error due to the replacement using the training set. Any pruning operation will increase the error on the original training dataset. Therefore, Quinlan proposed pessimistic pruning[65], a heuristically based method, that attempts to estimate the true error of a pruning operation using only the original training set. The method is loosely based on the familiar statistical concept of confidence limits but violates some of the rigorous statistical underpinnings. The pessimistic error estimate, e, is given by

$$e = \frac{f + \frac{z^2}{2N} + z\sqrt{-\frac{f^2}{N} + \frac{f}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}, \tag{3.9}$$

where N is the number of instances that reached that node, and f is the fraction of incorrectly classified instances. z is the number of standard deviations corresponding to a desired confidence value. The statistical underpinnings for pessimistic pruning are not rigorous, but in practice it performs well, and has the substantial benefit of allowing the whole dataset to be used in the tree building stage.

3.2 Example of Decision Tree Induction

This section will illustrate the induction and pruning of a C4.5 style decision tree from a synthetic dataset. The classification problem is to decide whether a given berry is edible or poisonous. The decision tree will be induced from a

Colour	Taste	# Instances	Edible
Red	Sour	30	Poison
Red	Sweet	30	Edible
Red	Salty	30	Edible
Blue	Sour	1	Edible
Blue	Sweet	40	Poison
Blue	Salty	40	Poison

Table 3.2: An abbreviated version of the berry dataset showing how many instances of the 171 instances match the given colour taste combinations.

dataset of 171 examples. For simplicity only two attributes will be considered: colour which can be either *red* or *blue*, and *taste* which can be either *sour*, *sweet* or *salty*. Table 3.2 summaries the significant attributes of the dataset.

3.2.1 Induction

Before deciding which attribute to split on, the information content of the Berry dataset before it is split must be found. This can be considered how much information is expected to be required to classify a single instance in the Berry dataset. Using Eq. (3.2) this can be calculated as

$$\text{info}(Berry) = -\sum_{i=\{eat,poison\}} P(C_i)\log_2(C_i)\text{bits}, \tag{3.10}$$

where

$$P(C_{eat}) = \frac{\text{num. edible}}{\text{total instances}} = \frac{61}{171}$$
$$, P(C_{poison}) = \frac{\text{num. poison}}{\text{total instances}} = \frac{110}{171}.$$

Substituting this back into Eq. (3.10) gives

$$\begin{aligned}\text{info}(Berry) &= -\left(\frac{61}{171}\log_2\left(\frac{61}{171}\right) + \frac{110}{171}\log_2\left(\frac{110}{171}\right)\right)\text{bits}\\ &= 1.05664.\end{aligned} \tag{3.11}$$

The next stage is decide whether to split the data on *colour* or *taste*. The information gained by spitting on *colour* can be calculated using Eq. (3.4) as

$$\text{info}_{colour}(Berry) = -\frac{90}{171}\text{info}(Red) + \frac{81}{171}\text{info}(Blue). \tag{3.12}$$

Using Eq. (3.2) info(*Red*) can be calculated as

$$\begin{aligned}\text{info}(Red) &= -\sum_{i}^{i\in\{poison,eat\}} P(C_i)\log_2(P(C_i)) \\ &= -\left(\frac{30}{90}\log_2\left(\frac{30}{90}\right) + \frac{60}{90}\log_2\left(\frac{60}{90}\right)\right)\text{bits} \\ &= 1.05664\text{bits}.\end{aligned} \tag{3.13}$$

Similarly, info(*Blue*) can be calculated as

$$\begin{aligned}\text{info}(Blue) &= -\sum_{i}^{i\in\{poison,eat\}} P(C_i)\log_2(P(C_i))\text{bits} \\ &= -\left(\frac{80}{81}\log_2\left(\frac{80}{81}\right) + \frac{1}{81}\log_2\left(\frac{1}{81}\right)\right)\text{bits} \\ &= 0.09599704\text{bits}.\end{aligned} \tag{3.14}$$

Combining Eq. (3.12), Eq. (3.13) and Eq. (3.14) the total expected information required after splitting on *colour* can now be calculated as $\text{info}_{colour} = \frac{90}{171}\text{info}(Red) + \frac{81}{171}\text{info}(Blue) = 0.601587$. The information gained by this split can then be calculated as

$$\begin{aligned}\text{gain}(Colour) &= \text{info}(Berry) - \text{info}(Colour) \\ &= 0.939931 - 0.601587 \\ &= 0.33834\ .\end{aligned} \tag{3.15}$$

Because the *taste* attribute will result in a multiway split, Quinlan's information gain ratio, Eq. (3.7) is required. The expected information gained by splitting the Berry dataset into two subsets containing 90 and 81 instances respectively can be calculated as

$$\begin{aligned}\text{split info}(berry) &= -\frac{90}{171}\log_2\left(\frac{90}{171}\right) + \frac{81}{171}\log_2\left(\frac{81}{171}\right) \\ &= 0.998\ .\end{aligned} \tag{3.16}$$

Using the results of Eq. (3.15) and Eq. (3.16) the information gain ratio, previously defined in Eq. (3.7), can now be calculated as

$$\begin{aligned} \text{gain ratio}(\mathit{Colour}) &= \frac{\text{gain}(\mathit{Colour})}{\text{split info}(\mathit{Berry})} \\ &= 0.339022\ . \end{aligned} \tag{3.17}$$

Similarly, the information gain ratio for the *taste* split can be calculated as

$$\begin{aligned} \text{info}_{taste} &= \frac{31}{171}\text{info}(\mathit{Sour}) + \frac{70}{171}\text{info}(\mathit{Sweet}) + \frac{70}{171}\text{info}(\mathit{Salty}) \\ &= 0.843891\ , \end{aligned} \tag{3.18}$$

where

$$\begin{aligned} \text{info}_{Sour} &= -\frac{1}{31}\log_2\left(\frac{1}{31}\right) + \frac{30}{31}\log_2\left(\frac{30}{31}\right) \\ &= -0.205593\ , \end{aligned} \tag{3.19}$$

and

$$\begin{aligned} \text{info}_(\mathit{Sweet}) &= -\frac{30}{70}\log_2\left(\frac{30}{70}\right) + \frac{40}{70}\log_2\left(\frac{40}{70}\right) \\ &= -0.985228\ , \end{aligned}$$

and

$$\begin{aligned} \text{info}_{Salty} &= -\frac{30}{70}\log_2\left(\frac{30}{70}\right) + \frac{40}{70}\log_2\left(\frac{40}{70}\right) \\ &= -0.985228\ , \\ \text{gain}(\mathit{taste}) &= \text{info}(\mathit{Berry}) - \text{info}(\mathit{Taste}) \\ &= 0.939931 - 0.843891 \\ &= 0.0960405\ . \end{aligned} \tag{3.20}$$

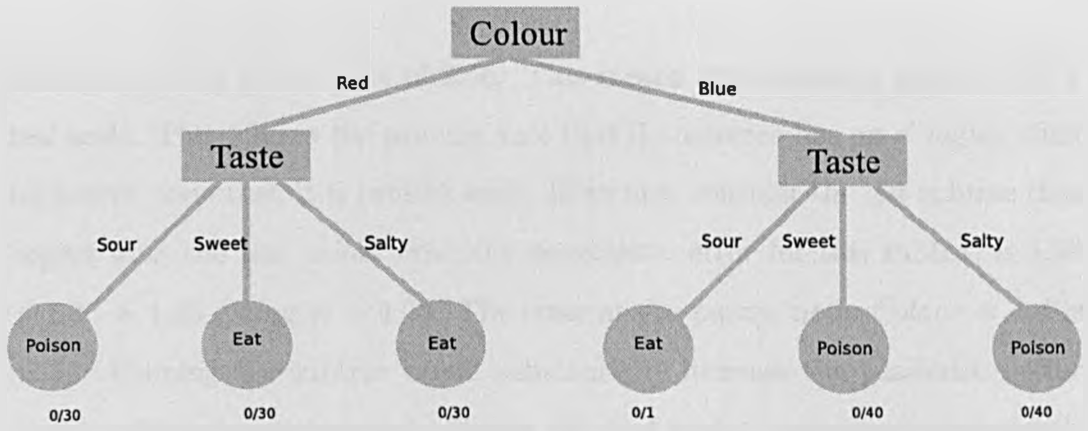


Figure 3.3: The full decision tree for the berry dataset.

Finally, the gain ratio can be calculated as

$$\begin{aligned}\text{split info}_3(Berry) &= \frac{31}{171}\log_2\left(\frac{31}{171}\right) + \frac{70}{171}\log_2\left(\frac{70}{171}\right) + \frac{70}{171}\log_2\left(\frac{70}{171}\right)\\ &= 1.5016\ .\\ \text{gain ratio}(Taste) &= \frac{\text{gain}(Taste)}{\text{split info}_3(Berry)}\\ &= 0.639589\ . \end{aligned} \tag{3.21}$$

Because gain ratio(*Colour*) > gain ratio(*Taste*) the root split of this decision tree should be based on *colour*. This process is applied recursively for each branch. The induced decision tree for the Berry dataset is shown in Figure 3.3.

3.2.1.1 Pruning the Berry Tree

After the tree is grown it normally overfits the data and needs therefore to be pruned back to improve generalisation. For example, using the decision tree in Figure 3.3, starting at the right subtree, *Colour = Blue*, the error for this subtree can be calculated by summing the pessimistic error for the three leaf nodes. Calculating the pessimistic error for each node from left to right gives *blue* and *sour* $e\prime = 0.75$, *blue* and *sweet* $e\prime = 1.36$, *blue* and *salty* $e\prime = 1.36$. This gives a combined pessimistic error of $e\prime = 3.47$. If the parent node *colour = blue* was turned into a *poison* leaf node it would have $f = \frac{1}{81}$ and classify 81 instances giving an $e\prime$ value of 2.56. This means the subtree is replaced by a leaf node. This follows the pruning rule that if a subtree has an e' higher than its parent node then it is pruned away. If we now consider the left subtree that begins with the test *colour=red*, the pessimistic error for this subtree is 1.35 + 1.35 + 1.35 giving $e\prime = 4.05$. The error at the parent node *Colour = red* is 33.59. Pruning the subtree would substantially increase the pessimistic error so the subtree is not replaced. Finally the root node is examined; replacing it with the majority class *poison* would classify 61 of the 171 instances incorrectly which would give a pessimistic error of 65.79. The left and right branches of the subtree are calculated, as previously, giving a pessimistic error of 4.05 + 2.56 = 6.61. Again, this replacement would substantially increase the pessimistic error so no pruning takes place. The pruning procedure for this tree, therefore, only replaced the right subtree *colour = blue* with a *poison* leaf node which produces the pruned decision tree shown in Figure 3.4 and the pattern space in Figure 3.5.

3.3 Refinements to Decision Trees

The previous sections described the basic algorithm for decision tree induction. This section briefly reviews some of the refinements and modifications made to decision tree induction.

3.3.1 Continuous Attributes

The examples previously given only dealt with discrete attributes from a finite set, for example *Colour* $\in$ {Red, Green, Blue}. However, many attributes are, of course, continuous in nature, such as temperature. Various methods have been proposed to overcome this limitation. When using the original ID3 algorithm it is normal to prediscretize the continuous attributes. For example, temperature could be changed from the continuous valued Celsius scale to *Temperature* $\in$ {Cold, Warm, Hot}. This can be done either manually using domain

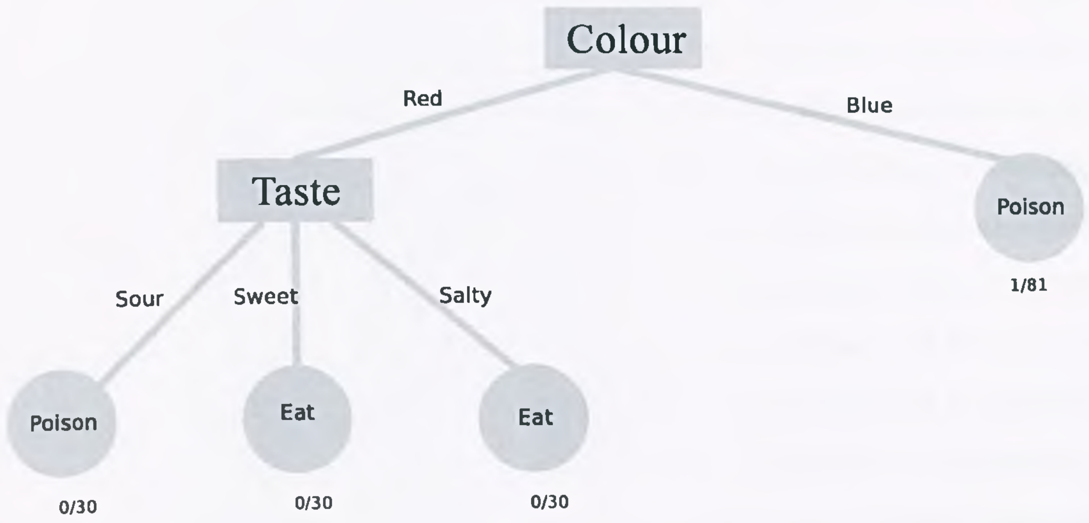


Figure 3.4: The pruned berry decision tree.

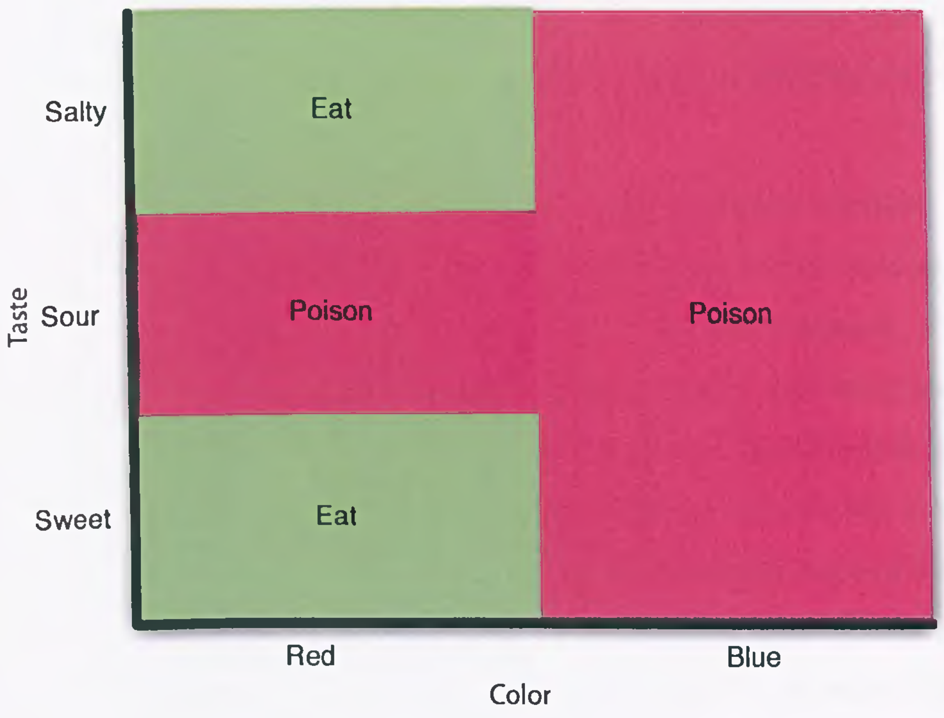
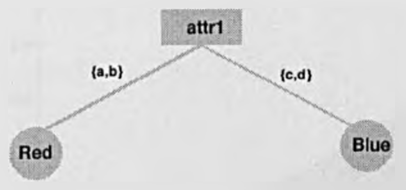


Figure 3.5: The pattern space of the berry decision tree.

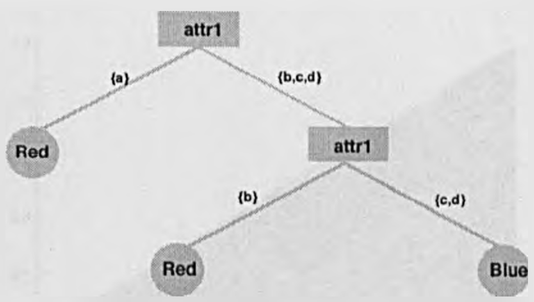
knowledge, or automatically. Automatic approaches include equal-interval binning where the range of the continuous variable is split into v bins. Unless the data is distributed uniformly along its range this can result in very uneven binning with some bins containing many instances and other bins containing only a few, or even no, instances. An alternative is equal-frequency binning which splits the numeric attribute into a set of v bins each containing an equal number of instances. This still has the significant disadvantage of ignoring the class attribute which can result in poorly chosen boundaries. C4.5 does not require continuous attributes to be prediscretized. At each node C4.5 considers each of the possible ways to create a binary partition of the data. For example, given a 10 instance data set with an attribute *temperature* with the following set of values $\{10, 28, 31, 38, 50, 66, 76, 84, 90, 100\}$, C4.5 would consider the information gained by splitting at each of the possible 9 split points. The procedure is computationally complex because it first requires the v values in the set to be sorted and then for up to $v-1$ splits to be considered. In practice, many datasets do not contain unique values for each instance so fewer than $v-1$ splits need to be considered. In addition to the computational requirement, this approach results only in binary splits which can be suboptimal in terms of clarity of explanation.

Quinlan's original ID3 algorithm always splits a discrete attribute with v possible values into v branches. For example, splitting on the taste attribute (Table 3.2) would result in a 3 way split. This raises two concerns. First, by splitting the data into many subsets, each containing only a few instances means patterns in the data become undetectable. Second, the gain ratio heuristic is unsuitable for comparing attributes which differ greatly in the number of possible values. Quinlan's C4.5 introduced an algorithm[65] to group attributes then create splits based on these groups. For an attribute with 4 possible values there are 15 possible groupings. The relation between the number of possible values v and the number of possible groupings is the Bell number[1] which quickly

[1]This is given by $B_v = \sum_{i=0}^{v} S(v,i)$ where $S(x,y)$ is a Stirling number of the second kind. Therefore, this grows exponentially. For example, the number of groupings for an attribute with 10 possible values is $B_{10} = 115975$



(a) Multivalued Binary Split



(b) {1}{k-1} Binary Split

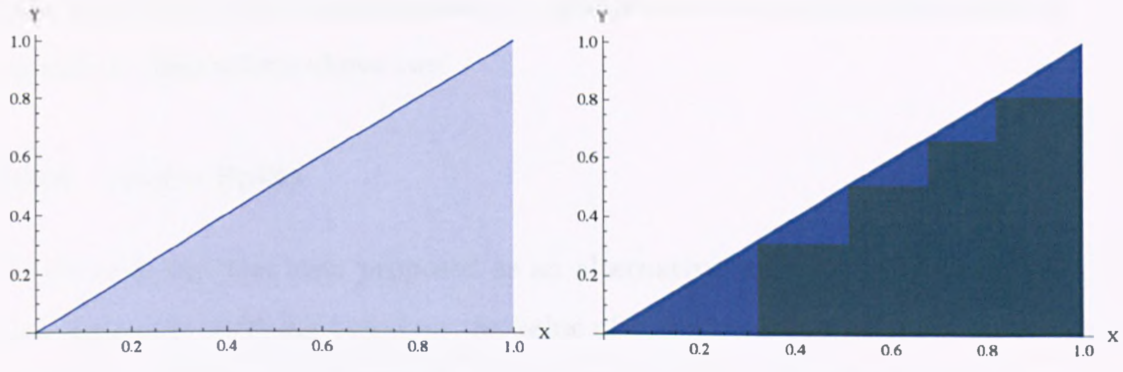
Figure 3.6: Any general multivalued split can be replaced with a {1}{k-1} binary split.

becomes too large.

Quinlan in the C4.5 algorithm used another heuristic-based algorithm to find groupings. It is an iterative algorithm which starts with the v potential splits and then compares each pairwise merger of these v partitions. After the best pairwise merger in terms of gain ratio is found the algorithm merges the two attributes then repeats again comparing each possible pairwise merge. The algorithm continues iterating until merging does not increase the gain ratio or only two subgroups remain. This algorithm results in far fewer comparisons than B_v but is still computationally intensive and being heuristic has no guarantee that it is close to finding the optimal grouping.

3.3.2 Binary Splits

The technique used by CART considers only binary splits such that for for an attribute with v possible values there are 2^{v-1} possible groupings. A simpler approach is to restrict the binary groupings into two groups: one containing only 1 value and the other group containing all the other possible values. For example, given the possible value set {a,b,c}, only consider the groupings {{a},{b,c}}, {{b},{a,c}}, {{c},{a,b}}. This leads to a much more tractable number of groupings to test. It is obvious that any grouping can be emulated by repeated binary grouping of this type, as in Figure 3.6. However, because not all groupings are



(a) Two Class Classification (b) Step Approximation

Figure 3.7: The decision boundary is not parallel to an axis so must be approximated with a step function.

considered, patterns in the data may still become undetectable.

3.3.3 Oblique Hyperplanes

The split types considered to this point have all resulted in splits which are parallel to the axis, as demonstrated in Figure 3.5. The advantage of such splits is the ease in which they can be understood, but the disadvantage is that they can result in overly large trees for relatively simple concepts. Oblique decision trees[89] use a linear combination of attributes which, by combining two attributes, can form non-axis parallel decision planes. Consider the simple partition of the pattern space shown in Figure 3.7; a decision tree restricted to only axis-parallel splits would have to approximate the split with a series of steps as shown. Not only would this result in degraded performance it would also fail to capture the real relationship between x and y. In contrast, an oblique test of the form

$$\sum_{i=1}^{d} a_i x_i + a_{d+1} > 0, \tag{3.22}$$

where $a_1, \ldots, a_{d+1}$ are real-valued coefficients could easily represent the function with a split test such as $x - y > 0$. This would not only result in a smaller tree but it is also a closer representation of the 'true' relationship between the variables. The drawback is the creation of such splits is computationally intensive[55] and often much harder to understand than axis-parallel splits especially in dimensions above two.

3.3.4 m-of-n Splits

The m-of-n test has been proposed as an alternative splitting test[54]. In this case instead of splitting based on the value of a single attribute, the tree splits are based on whether m conditions out of the set of n conditions can be met. For example, in the test, 2-of$\{colour = blue, taste = salty, size = big\}$, any 2 of the 3 conditions must be true. This type of split has the advantage that it can represent some concepts more concisely, which results in smaller decision trees. However, it is not inherently more powerful as any m-of-n test can be rewritten in standard Boolean(DNF) form as used in C4.5. The m-of-n type test is used in the Trepan rule extraction method(See Section 4)

3.3.5 Fuzzy Decision Trees

A significant disadvantage of decision trees is the sharp decision boundaries. In a decision tree split such as the root node in Figure 3.1 which has a split based on the *Outlook* attribute, if the attribute *outlook* = *rain* then with this type of split the decision is made based solely on the rightmost branch ignoring the 'knowledge' captured in the *sunny* and *overcast* branches. This, at first, may seem entirely reasonable; however, weather is not a discrete phenomenon, as some rain storms are heavier than others.

Fuzzy set theory[99] is an extension to set theory to incorporate the concept of membership of a set. For example, a day with a heavy downpour would have a high degree of membership to the class rain, a very low membership of sunny and low membership of overcast. A day with light rain may have a medium degree of membership with rain but also a medium level of membership with overcast.

In terms of decision trees the last example of a day with light rain will have a decision based on the *rain* branch but will also be influenced by the

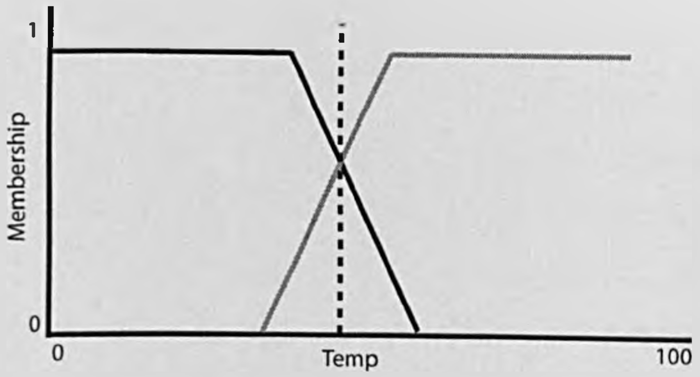


Figure 3.8: A fuzzy membership function for heat which shows as temperature increases membership of cold decreases and membership of hot increases.

overcast and *sunny* branches but to a lesser degree, based on the degree of membership. As discussed in Section 3.3.1, continuous attributes are discretized either before tree induction or as part of the algorithm. This can have a drastic effect on instances which are close to the boundaries. Fuzzy sets can be used to alleviate this problem by creating membership functions that give instances a membership degree based on how far the instance is from the mean. Instances close to the mean would have a high degree of membership with the degree of membership reducing as the instances move away from the mean. There have been several attempts to combine fuzzy set theory and decision trees[38, 18].

Figure 3.8 shows a potential split on temperature at 50. Using traditional crisp splits an instance with a temperature of 48 and another instance which is identical, apart from having a temperature of 51 could have very different outcomes based on the divergent paths each instance has taken through the decision tree. If fuzzy membership is used the decision tree would be more likely to classify these instances the same by combining the results of the divergent paths.

3.4 Summary

This chapter has introduced the pattern classification technique of decision trees. Decision trees are an easy to comprehend, graphical representation of a decision process. Algorithms for inducing and pruning a decision tree were discussed with an emphasis on the C4.5 algorithm. The focus on C4.5 was because the tree induction parts of the rule extraction algorithms developed in Chapter 5 are likewise based on information entropy. An illustrative example of tree induction and pruning was given. The chapter concluded with a brief review of refinements and developments of decision trees.

CHAPTER 4

Rule Extraction From Artificial Neural Networks

Neural networks' greatest weakness is their opaqueness. Unlike symbolic rule based systems, which explicitly show their reasoning, neural networks hide their knowledge in the complex interrelationships of their weights. This means that, although neural networks provide excellent models for prediction, they provide no insight into the relationships between input variables and output that the model may have found. For example, researchers at MMU have created a neural network that can classify a person's responses as either deceptive or truthful, using clues in their nonverbal behaviour (e.g. eye moments, shrugs). Although the neural network has good predictive accuracy it does not reveal the relationships it has found between nonverbal behaviour and deception[70].

This chapter begins by looking at the rule extraction task and examines why neural networks are black-boxes. A comparison of neural networks and decisions trees is then given. Previous work in rule extraction is then reviewed. Finally, the chapter concludes with an overview of the contributions the novel algorithms, presented in the remainder of this thesis, will make to the field of rule extraction.

4.1 The Rule Extraction Task

The aim of rule extraction is to reduce the complexity of a neural network into a more easily understood symbolic form. These rules can then be analysed for trustworthiness for safety critical systems or used to provide insights into the

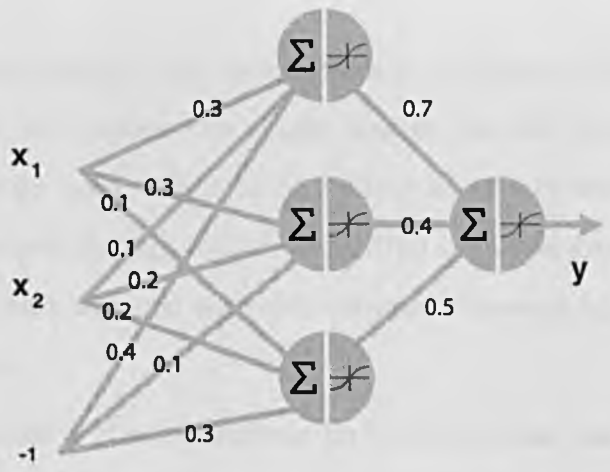


Figure 4.1: A multilayer neural network illustrating the weights between inputs and neurons and between neurons.

relationships found by the neural network. As shown in Figure 4.1 the neural network model is difficult to interpret. The knowledge within the neural network is hidden in the complex interrelationships of the neurons. In examining the weight matrix for meaning it can be tempting to think that a large weight value attached to an input indicates that the input is important. However, this can be misleading because although a large weight does indeed indicate that the input's contribution to the connected hidden neuron is significant, that hidden neuron may itself contribute very little to the output value. Alternatively, the same input may connect to another hidden neuron with an equal weight, but of opposing sign cancelling out any effect. Moreover, several small weights that may seem insignificant may have an aggregate effect that is far more significant than a single large weight. If two inputs are strongly correlated, but both irrelevant to the classification, and one ends up with a large weight, then through the training process the neural network may balance this out, not through decreasing that weight, but by increasing the other input's weight to cancel out the effect of the original large weight. This gives the illusion that both inputs are important to the classification when, in reality, neither is important.

This means making conclusions about the relationships between the input and output values based on the weight matrix is difficult. However, an early method of trying to visualize the weight matrix was the Hinton diagram[71]. Such diagrams show each weight as an individual square with the magnitude indicated by area and the sign by the colour (red negative, green positive). The weights are organised on a grid such that columns represent inputs and the rows represent neurons.

Figure 4.2 shows a Hinton diagram for a neural that has been trained to solve the XOR problem. The diagram shows that each neuron is connected to the inputs by 2 weights of opposing signs but equal magnitude. This has the effect that if both inputs have a value of 1 then each hidden node will receive two weighted sums of equal magnitude, but opposite signs, which will then cancel out. This creates the desired behaviour for the XOR function such that if both inputs have a value of 1 then the output value is 0. It can also be seen that the other input patterns are handled correctly: if both inputs are 0 then the hidden layer output must be 0; if only one input is a 1 then the output of the hidden layer will be have a high positive or negative value. This leaves the output neuron to act as an absolute function.

Although the Hinton map allows a visual representation of the weight matrix it is not easy to interpret. It also becomes more complex as the number of inputs and neurons increase as shown in Figure 4.3, which shows a Hinton diagram for a neural network for a breast cancer dataset that will be used in Chapter 6. The Hinton diagram can be useful for seeing if an input has little or no effect, and there are techniques for rearranging the grid to make certain relationships between inputs and outputs more apparent, but these generally require domain knowledge. The Hinton diagram is a limited tool, but does further illustrate how it is difficult to elicit meaning directly from the weight matrix of a neural network.

Although neural networks are good classifiers they provide no insight into the relationships between input and output variables the model may have found. In comparison, symbolic rule-based systems, such as decision trees, explicitly reveal

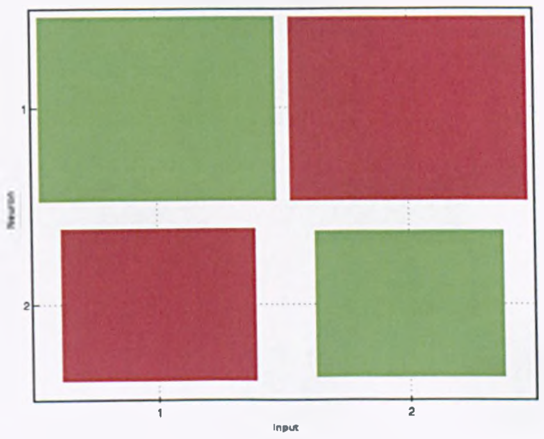


Figure 4.2: A simple Hinton diagram for XOR neural network.

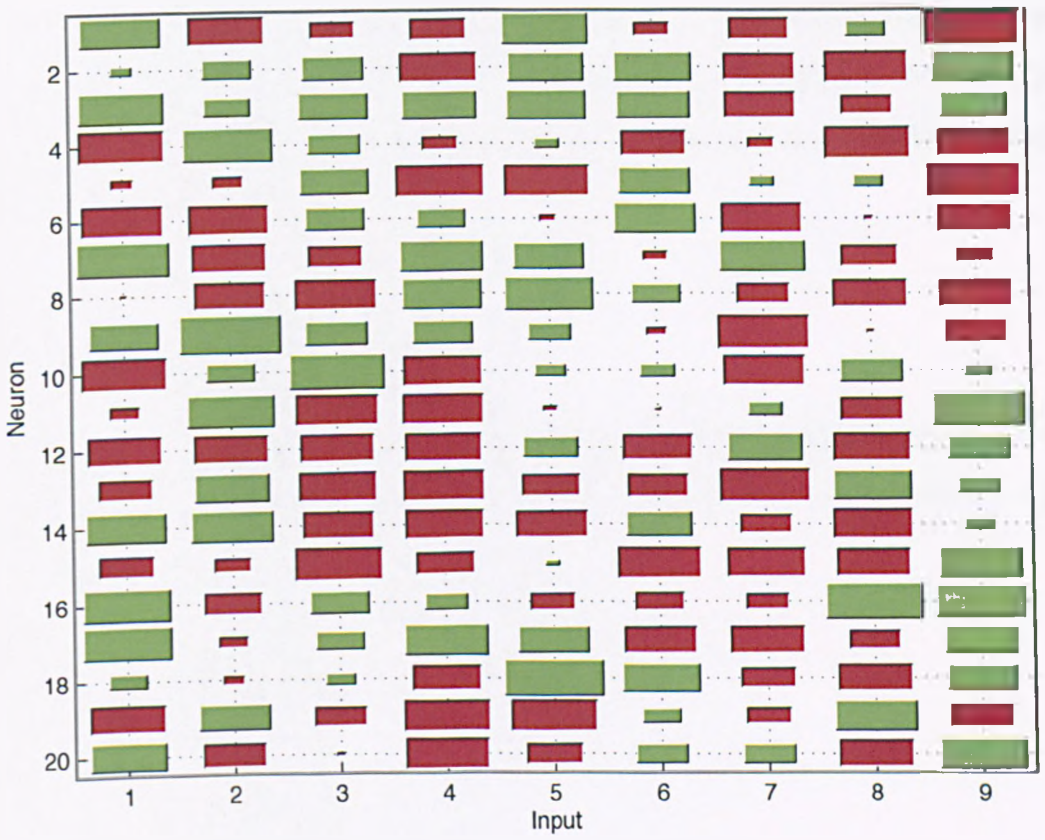


Figure 4.3: A Hinton diagram for breast cancer neural network.

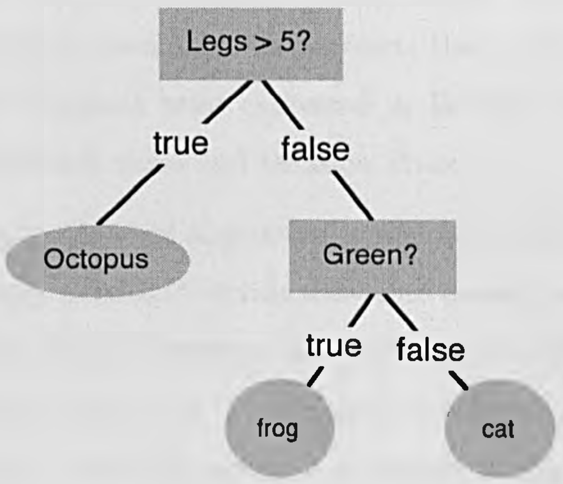


Figure 4.4: A decision tree for a simple animal classification task.

the knowledge they have acquired in an easy to comprehend form. For example, looking at the decision tree in Figure 4.4 it is easy to see the relationship between the input attributes *Legs* and *Green*, and the output classification. No specialist knowledge or further processing is required to understand the knowledge within the model.

4.2 Previous Work

A taxonomy for ordering and grouping rule extraction algorithms has been proposed[4], and considers the following aspects of the rule extraction algorithms:

1. Expressive power
2. Translucency
3. Specialised Training Regimes
4. Quality of the extracted rules
5. Algorithmic complexity.

Expressive power refers to the type of rules extracted from the neural network as separate from the technique that extracts them. Rule extraction techniques so far have extracted rules expressed in Boolean logic, Fuzzy logic, IF...Then...rules, M-of-N rules, and Decision Trees.

Translucency means the level of granularity with which the neural network is examined. Craven and Shavlik[15] divide these into decompositional techniques which examined the individual weights, and pedagogical techniques which treat the neural network as a black box. Pedagogical techniques use the neural network as an oracle by repeatedly asking it questions in the form of instances and then using the classifications to create a descriptive model of the neural network.

Specialised Training Regimes are required by many of the rule extraction algorithms. Either modifications to the standard neural network training algorithms, or the use of a specific architecture, or both are needed. Although these techniques have been successful in extracting rules, such methods cannot be applied to existing neural networks models. This is a significant disadvantage because many existing proofs of desirable properties, such as the universal approximator proof, do not then apply to these modified versions. Furthermore, such methods cannot be applied to pre-existing trained neural networks which are already being used effectively.

Andrews[85] proposes four measures for the *quality of the rules* extracted from neural networks: accuracy, fidelity, consistency and comprehensibility. Accuracy measures the ability of the rule set to correctly classify previously unseen instances from the problem domain. Fidelity is how well the extracted rule set corresponds to the original neural network. Consistency, in this context, is whether the extracted rule set is the same under different training sessions of the neural network. Comprehensibility is a measure of the number of rules produced by the extraction algorithm.

The final dimension is *algorithmic complexity*, which attempts to provide a measure for the efficiency of an algorithm; examining such aspects as whether the algorithm scales exponentially with the number of hidden neurons or inputs, which would mean the algorithm would not be applicable to large neural networks.

Several approaches to rule extraction from neural networks have been proposed and a few of the significant ones are now discussed. An early approach to rule extraction was the *The Connectionist Scientist Game*[51]. Inspired by the induction of scientific hypotheses, this method uses an iterative technique. First, a neural network is trained, which in their analogy is like a scientist developing intuitions about a problem domain. Second, rules are extracted from the network, the analogy being that this is like forming explicit hypotheses which can then be checked to see if they cover the domain. Third, the rules are translated back into weights to be injected back into the neural network. This sequence of extraction and injection is repeated until the rule base adequately characterises the domain. In the same paper, RuleNet, an architecture which implements the Connectionist Scientist Game is presented. RuleNet maps input strings of n symbols to output strings of n symbols. The RuleNet system uses a fixed 3 layer architecture with an input layer of n units, a hidden layer of m condition unit and an output layer of n units. Each unit in the condition layer represented an IF..THEN rule which can be extracted in the extraction phase. As RuleNet requires a fixed architecture and learning algorithm, it is not applicable to pre-existing trained neural networks, but does represent an early recognition of the need for rule extraction.

The Subset rule extraction method[87] is typical of the decompositional approach, and similar methods have been proposed by Saito and Nakano[72] and Fu[28]. The subset method extracts a series of rules from each node in the network. A rule is created for each combination (or subset) of inputs that could cause a node to activate. For example given the node in Figure 4.5 the following rules could be extracted:$a \land b \land c \implies y$, $a \land b \land \neg d \implies y$, $a \land c \land \neg d \implies y$, $b \land c \land \neg d \implies y$, $b \land c \land \neg d \implies y$. This particular implementation requires the outputs of the nodes to be binary and therefore

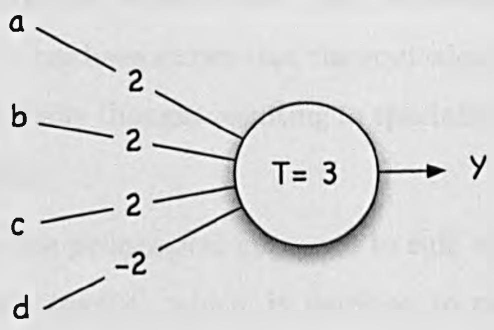


Figure 4.5: A neuron with 4 inputs and a threshold value of 3.

cannot be applied to pre-existing MLPs that normally have neurons with real value outputs. Moreover, the number of rules increases exponentially with the number of nodes making the algorithm intractable for large networks. This approach was extended by the n-of-m algorithm by Towell and Shavlik[87]. A n-of-m rule contains a set, m, of tests of which n must be satisfied for the rule to be evaluated as *true*. For example, the n-of-m rule 2-of-$\{r_1, r_2, r_3\}$ is equivalent to $(r_1 \wedge r_2) \vee (r_1 \wedge r_3) \vee (r_2 \wedge r_3)$. This style of rule is particularly appropriate for representing the activation of a neural node. For example, the rules of Figure 4.5 can be represented as 3-of-$\{a, b, c, \neg d\}$. However, for a multiple layered network to be represented in a concise number of n-of-m rules and overcome the exponential growth problem of the subset algorithm, the antecedents of the rules should be equivalent, i.e. it does not matter which n are *true*. Standard back-propagation has no predisposition to favour such an arrangement. Therefore, either the neural network needs to be initialised using a pre-existing rule set and/or trained using a special training algorithm[87].

Algorithms[42, 56] have been proposed which extract fuzzy rule sets from neural networks. These approaches usually require a domain expert to label the resulting fuzzy sets and/or require specialised neural network architectures and training algorithms. These approaches have generally been applied to rule refinement where a pre-existing set of fuzzy rules have their membership functions refined by the neural network. Jang and Sun[37] note a functional equivalence between radial-basis-function networks and fuzzy inference systems under some conditions. However, it has been shown that the equivalence conditions are more restrictive than was initially thought resulting in specialised training algorithms again being required[3].

Trepan[16] follows the pedagogical approach to rule extraction. Trepan creates an n-of-m decision tree[54], which, in addition to the C4.5-style splitting rule, can make use of a n-of-m splitting rule at any of the nodes. The use of n-of-m splits can fit certain concepts more naturally than C4.5-style splits at the cost of a certain amount of comprehensibility. Another interesting feature of Trepan is its use of best-first tree expansion in contrast to the more usual depth-first expansion. The next node to expand is the node that maximises the function,

$$n^* = \arg\max_n \left(\text{reach}\,(n)\,(1 - \text{fidelity}\,(n))\right), \tag{4.1}$$

where reach(n) is the number of instances that have reached node n and fidelity(n) is the percentage of instances at node n that the decision tree and the neural network classify as belonging to the same class. This has the effect of concentrating growth of the tree in the region that most increases fidelity. However, after the tree is fully grown and pruned the difference between the two methods is negligible. The real advantage of this approach is the ability to more precisely control whether the tree should be large, which increases accuracy but decreases comprehensibility, or the tree should be small, which decreases accuracy but increases comprehensibility. Trepan uses information gain to decide on which attribute to base the splitting test. To extract 'knowledge' from the neural network Trepan uses a sampling and querying approach. The neural network is used as an oracle which can be queried for the class assignment of a sampled instance. To create a query instance Trepan models the original dataset using an empirical distribution for nominal attributes and kernel density estimation[79] for the continuous attributes. The empirical distribution means the nominal values are sampled with a probability based on their frequency in the original dataset. For continuous attributes, a probability density function (PDF) using a kernel density estimate with a Gaussian kernel is sampled.

4.3 Comparison of Neural Networks and Decision Trees

Decision trees and neural networks most significant difference is their comprehensibility: decision trees are easy to understand, neural networks are block boxes. However, neural networks and decisions trees also differ in the way they partition the instance space. Neural networks can represent any borel-measurable function[33][13], whereas traditional decision trees are only capable of representing concepts that are representable as Boolean functions.

Consider, a classification task as represented in Figure 4.6, the two classes to be separated are the Red points in the circle, and the Blue points covering the remainder of the instance space. A neural network can learn this function easily as shown in Figure 4.7. The neural network fits a function over the instance space, which starts at a value of 0 at the 4 corners of $\mathscr{X}$ before steeply rising to one in the centre of $\mathscr{X}$. If a threshold value on the output is set to 0.5 then a decision boundary is created that models the original in Figure 4.6.

In contrast, a decision tree is limited to solving this problem through repeated binary partitioning of the instance space. A decision tree for this dataset is shown in Figure 4.8. It starts by partitioning off the area $x > 0.8$ and the area $x < -0.8$ as belonging to the Blue class. The remaining area is then partitioned on y. First the area $y < -0.8$ and then the area above 0.8 is classed as Blue. This leaves the centre square to be classified as Red. The decision tree in Figure 4.8(a) is far more comprehensible than the neural network in Figure 4.7(a), but the decision tree is limited by the axis-parallel splits which results in errors as shown in Figure 4.8(b). The points outside the circle but within the axis-parallel decision planes are incorrectly labelled as class *Red*, but should be classified as class *Blue*, and a few points which are inside the circle but outside the decision planes are incorrectly labelled as class *Blue* when they should be labelled as class *Red*.

This discussion illustrates a difference between neural networks and de-

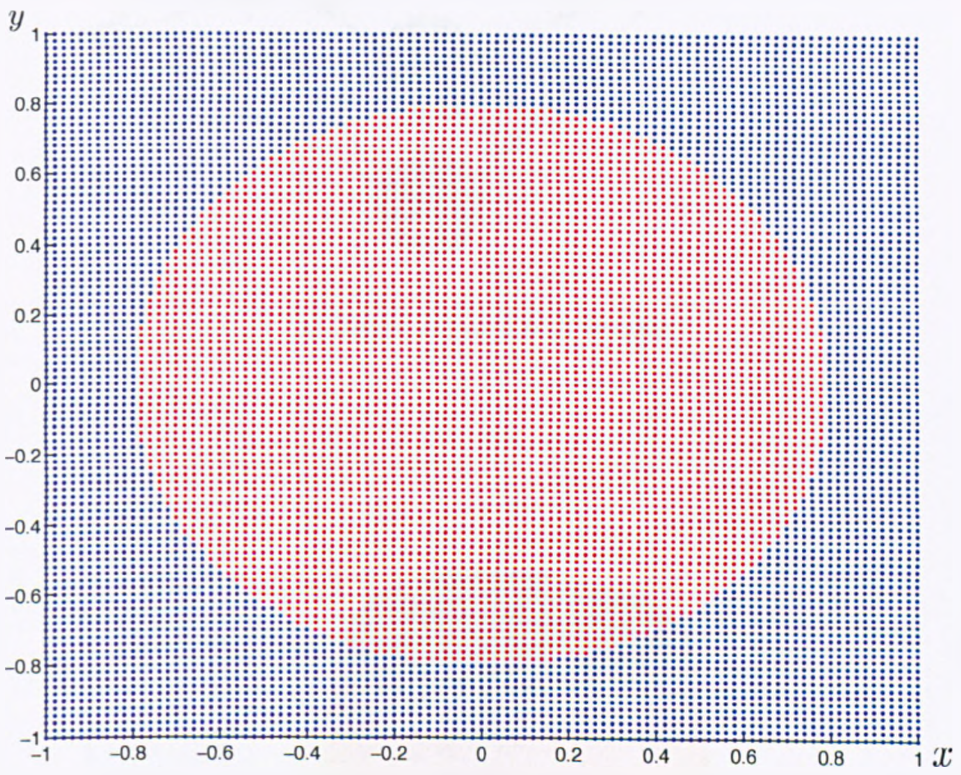
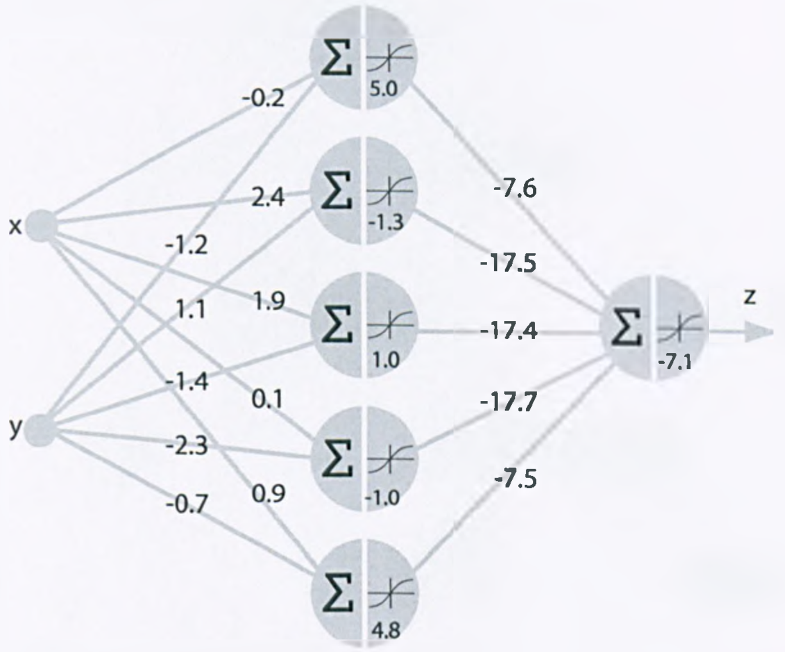
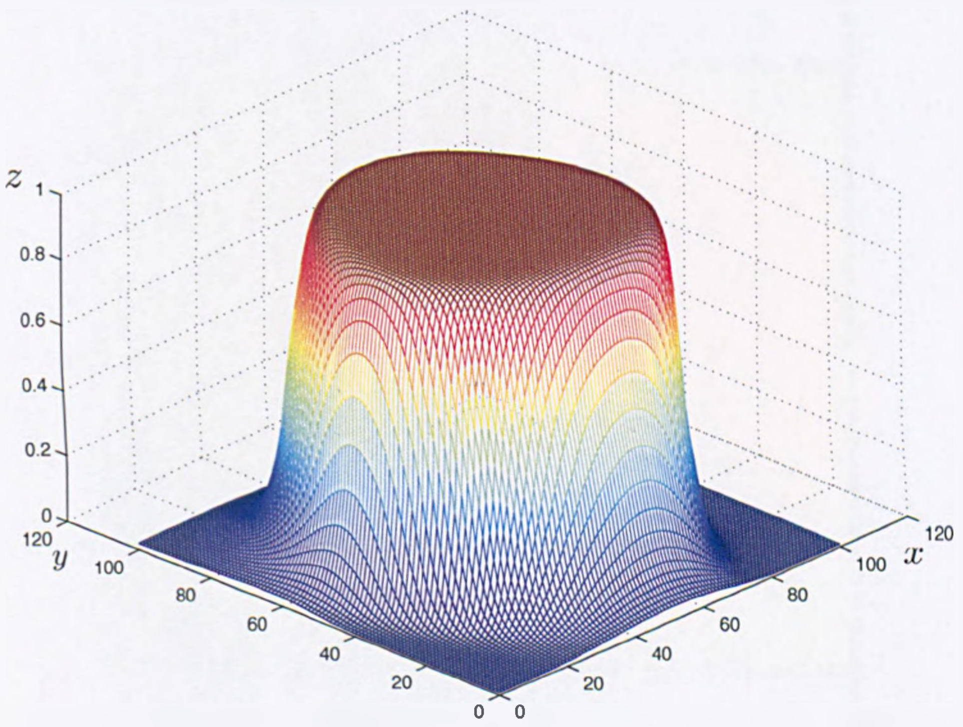


Figure 4.6: Pattern space of circle dataset with a decision boundary at $x^2 + y^2 = 0.8$.

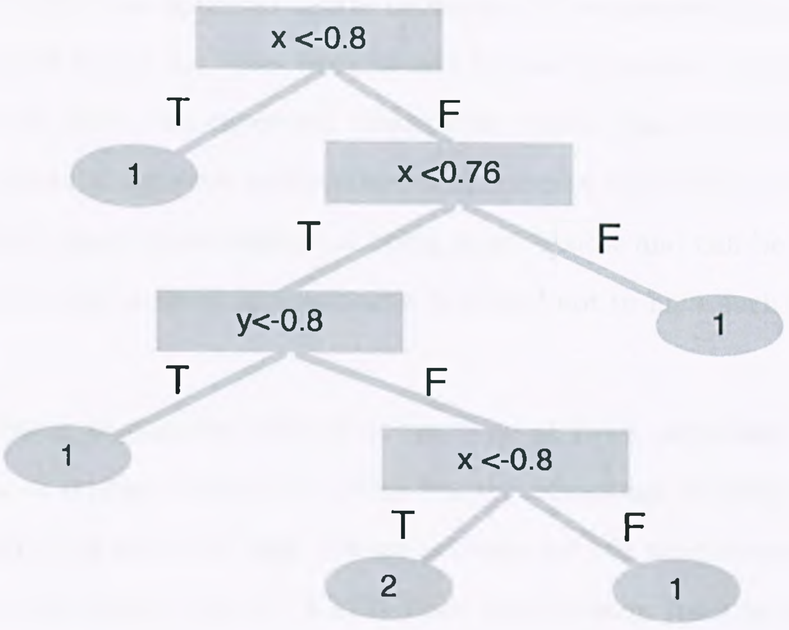


(a) Neural network with weights

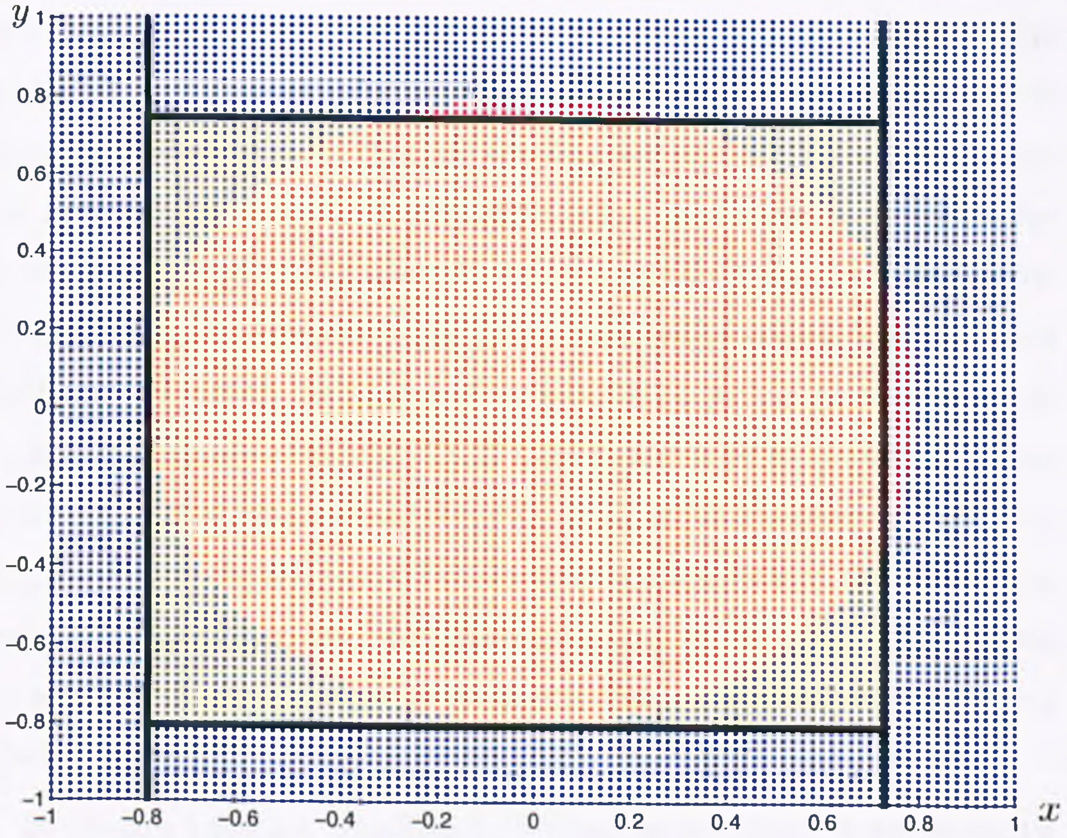


(b) Instance space

Figure 4.7: A neural network and resulting instance space that solves the circle dataset.



(a) Decision tree



(b) Decision tree instance space

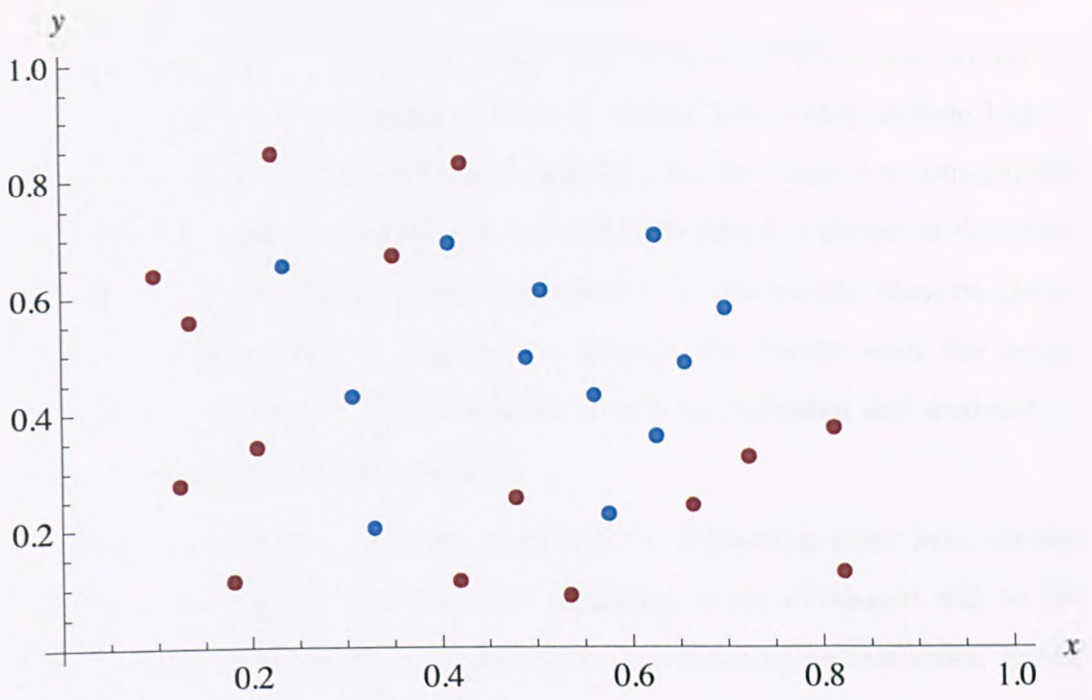
Figure 4.8: A Decision Tree and resulting instance space that solves the circle dataset.

algorithms and/or pre-specified neural network architectures which means they cannot be used to extract rules from *in situ* neural networks. Therefore, they do not benefit from the important theoretical results that have been proved for standard neural network architectures and training algorithms. Pedagogical techniques[17], have the advantage of being more flexible and can be applied to a number of neural network architectures, but tend not to have such high levels of fidelity.

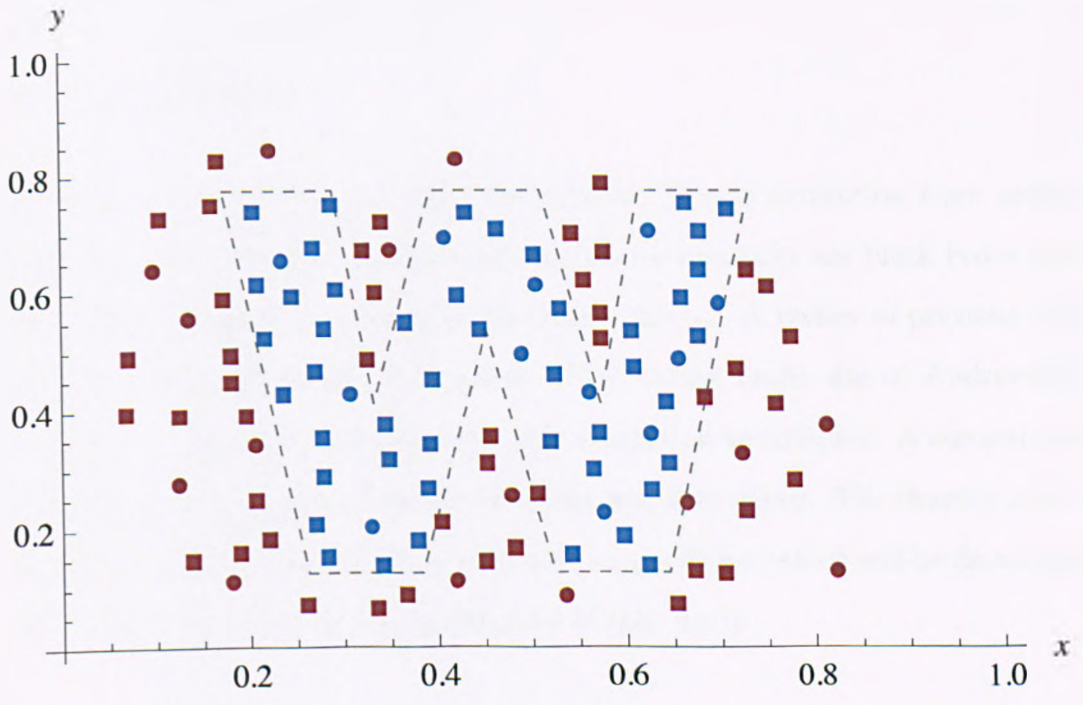
The existing algorithms differed in the type of rules extracted; some use formal logic to represent the rules, which has the advantage of being able to be manipulated using standard logic but are perhaps not the most comprehensible form, other algorithms used IF..THEN rules and decision trees as the output form, which are more easily comprehended.

In the remaining chapters several new algorithms are developed. These algorithms are pedagogical and will use various forms of decision trees for the output form so that the rules produced will be easy to comprehend. All the algorithms will make use of Craven's sampling and querying to extract information from the new network. The dataset used to train a neural network is the set of points in the instance space $\mathscr{X}$ that the neural network has been trained to classify, but the neural network will be expected to generalise to all points within $\mathscr{X}$. To capture the totality of the information within the neural network a rule extraction algorithm can sample more points in $\mathscr{X}$ by generating queries in the form of new instances and then requesting the neural network, which acts as an oracle, to classify them. For example, in Figure 4.9(a) the relationship between the blue points and the red points is unclear. However, if more points are added as shown by the square markers in Figure 4.9(b) then the underlying relationship (the **W** pattern) becomes clear.

In Chapter 5 the first algorithm ExTree will be developed. It will be similar to the Trepan algorithm but will produce a standard C4.5-style decision tree as opposed to the n-of-m decision tree the Trepan algorithm produces. ExTree will then be analysed on a number of synthetic datasets to assess how the various



(a) Original Dataset



(b) Augmented Dataset

Figure 4.9: In the original dataset the relationship between the two classes is unclear but by sampling extra points the underlying relationship is revealed.

aspects of the algorithm contribute to classification accuracy and fidelity.

The second algorithm to be developed will be ExLMT which will extend the ExTree algorithm by extracting a Logistic Model Tree which include logistic models at the leaf nodes. Traditional decision trees are limited to axis-parallel splits whereas neural networks have more flexible decision planes as described in Chapter 2. The Logistic Model Tree allows non-axis parallel decision planes in the leaf nodes which is expected to increase the fidelity with the neural network. In Chapter 6 ExTree and ExLMT will be evaluated and analysed on a large number of real-world datasets.

ExTree and ExLMT are only applicable to extracting rules from classification neural networks, and the final algorithm to be developed will be the ExMT algorithm which will be applicable to regression neural networks. ExMT achieves this by extracting a Model Tree also explained in Chapter 7. ExMT will also be evaluated and analysed on a number of real-world regression datasets.

4.5 Summary

This chapter began by examining the problem of rule extraction from neural networks. The first section discussed why neural networks are black boxes and used Hinton diagrams to visualize the weight matrix. A review of previous rule extraction algorithms was then given. This review made use of Andrews'[4] taxonomy to analyse and define the rule extraction techniques. A comparison between decision trees and neural networks was then given. The chapter ended with an overview of the novel rule extraction algorithms, which will be developed and evaluated in the remaining chapters of this thesis.

CHAPTER 5

ExTree and ExLMT

In this Chapter ExTree, a novel algorithm for extracting decision trees from artificial neural networks, is developed. An extension to this algorithm, ExLMT, which extracts a Logistic Model tree is also presented. Both these algorithms are for extracting rules from the neural networks trained on classification problems. Extraction from regression problems will be considered in Chapter 7.

It has been previously established that a substantial weakness of neural networks is their inability to explain their reasoning. This opaqueness limits the applications where neural networks can be used. Although several rule extraction methods have been proposed, as discussed in Section 4, the ExTree algorithm presented in this chapter possesses additional desirable properties. It extracts a C4.5-style decision tree which is a familiar, graphical and easy to understand representation of a decision process.

ExTree being a pedagogic method requires a trained neural network to act as an oracle. By repeatedly asking the neural network questions, in the form of requests to classify instances, the knowledge hidden within the neural network can be extracted. In Chapter 6, ExTree will be applied to MLPs trained with backpropagation but ExTree can easily be extended to other neural network types such as Radial Basis Function Networks(RBF), or other pattern recognition techniques that are opaque. ExTree does not require the neural network to use a specialised training algorithm, or pre-specified architecture and only requires that it fulfils the pattern recognition mapping, $\hat{C} : \mathscr{X} \rightarrow 1 \ldots K$, defined in Chapter 1. Once a trained neural network is available, ExTree proceeds in a similar manner to decision tree induction algorithms: recursively splitting the tree by finding the best attribute to split on.

ExTree is similar to standard decision tree induction algorithms such as CART and C4.5, discussed in Chapter 3. However, unlike C4.5 and CART, ExTree has access not only to a training set but also to a neural network which has already been trained on that training set. Standard decision tree induction algorithms have the limitation that the selection of the splitting test is based on fewer and fewer instances as the tree grows downwards. The splitting tests that are near the bottom of the tree are often poorly chosen because there is insufficient data to find the optimal split. ExTree alleviates this problem by generating new instances, and then querying the neural network, which acts as an oracle, with the newly created instances. ExTree can then select a splitting test based on the newly created instances as well as the original dataset.

5.1 The Algorithm

The algorithm for ExTree is given in Figure 5.1. The algorithm follows the basic decision tree algorithm given in Chapter 3. An overview of the algorithm is now presented, followed by a more detailed discussion of the main components of the algorithm.

The algorithm extracts a decision tree from a neural network which has been previously trained to a high level of accuracy using a standard training algorithm on a dataset S. The ExTree algorithm starts with the original dataset, S, which is relabelled using the neural network. The algorithm also requires a set of constraints, *Const*, defining the values that each of the attributes in the dataset can hold. Initially, these constraints represent the whole of the instance space, $\mathscr{X}$ and will be further refined as the algorithm moves down the tree. A set of N new unlabelled instances is then created that satisfies the constraints in *Const* by creating a model of the dataset and sampling from it, as discussed in Section 4.4. The new instances are labelled using the neural network in its role as an oracle, and are added to the existing dataset. If all the instances belong to the same class C_k then the current node is labelled as Class C_k. If the instances do not all belong to the same class then a split which creates subsets of instances which are purer than the original set is found. For each subset, a set of constraints is created from the parent constraints that define the area of the $\mathscr{X}$ that would go into the subset. The algorithm is then applied recursively for each subset until the subsets meet an acceptable level of purity.

5.1.1 New Instances and Oracle Querying

An advantage of the ExTree algorithm is that new instances can be created and classified by the neural network. New instances are created at a node to provide more information on which to base the selection of a split test. ExTree uses Craven's sampling and querying approach[16] to elicit knowledge from the neural network. It models the original dataset then samples this model to create new instances. These new instances are then used to query the neural network to obtain class labels, which has the effect of expanding the original dataset. It is desirable to model instances based on the current dataset because in real world datasets much of the potential pattern space $\mathscr{X}$ is of little interest. For example, in a house price dataset the instances which represent 1 bedroom houses with 5 bathrooms are valid datapoints but are of little real world interest. The new instances, therefore, are created by fitting a distribution to each attribute which can then be sampled to create new instances.

For nominal attributes an empirical distribution is used for the model. This means nominal values in the new instances are sampled according to their frequency in the original dataset. For example, if an attribute *Colour* belonged to the set $\{red, green, blue\}$ in a dataset with 15 red instances, 25 green instances and 10 blue instances then the new instances would be created such that approximately $\frac{3}{10}$ of the instances are red, $\frac{5}{10}$ of the instances are green, $\frac{2}{10}$ of the instances are blue.

For numeric continuous attributes a probability density function (PDF) is

```
ExTree( dataset $S$, constraints $Const$ )
BEGIN
NewInstances :=
      Create $N$ new instances constrained by $Const$;
FOR each instance in NewInstances
   Label instance using neural network
$S$ := $S$ + NewInstances;
IF all $S$ belongs to $C_k$ THEN
   label node as leaf $C_k$
    RETURN
ELSE
   Find Best Split $S*$
   Split the $S$ into subsets $S_1..S_n$according to $S*$
   FOR each subset$S_i$
   BEGIN
      IF the number of instances in subset is 0
      THEN mark node as dominating class of parent
ELSE IF node is a mixture of classes
   Create new Constraint $Const_i$ from $Const$,
   ExTree(oracle,$S_i$,$Const_i$)
END
```

Figure 5.1: The pseduo code of the ExTree algorithm.

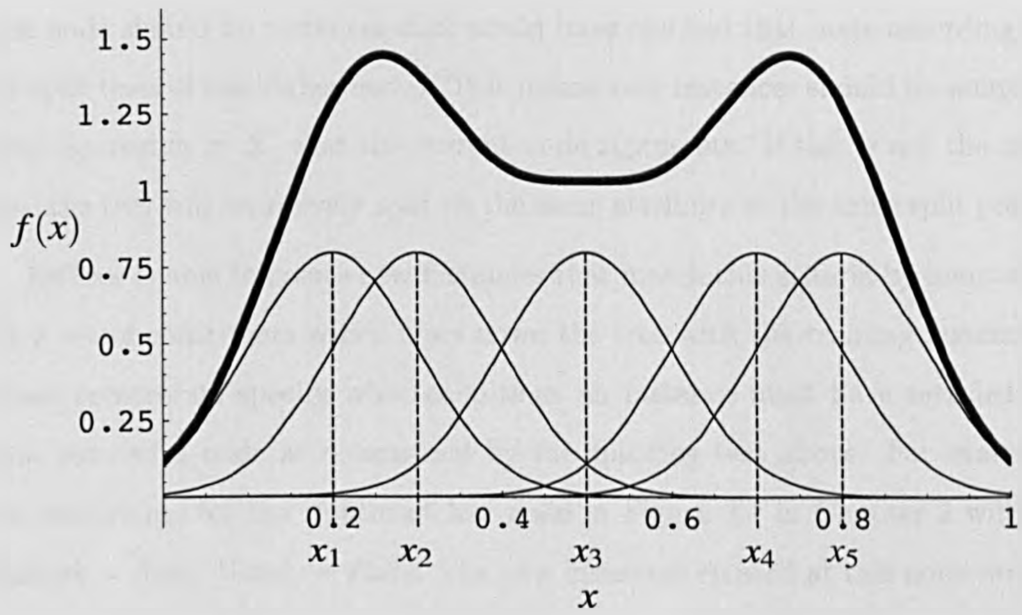


Figure 5.2: A example Kernel-density estimation using five gaussian kernels.

estimated using kernel density estimation with a Gaussian kernel such that,

$$f(x) = \frac{1}{m}\sum_{i}^{m}\left(\frac{1}{\sqrt{2\pi\sigma}}e^{-\left(\frac{x-u_i}{2\sigma}\right)^2}\right), \tag{5.1}$$

where m is the number of original instances, u_i is the attribute value for the ith examples, and σ is the width for the Gaussian kernel. As illustrated in Figure 5.2, kernel density estimation can be thought of as creating a PDF by summing a series of Gaussian functions centred on the current data points with a standard deviation or width determined by the size of the dataset. Based on preliminary experiments, ExTree uses a width of $1/\sqrt{m}$ which provides acceptable performance over several datasets but could be further optimised. The new instances' attribute values are then sampled from this PDF. The new instances $\mathscr{T}^*$ are then labelled by the neural network to produce a set of instances to be added to the relabelled dataset $\mathscr{R}$ to create the new training set $\mathscr{T}'$ giving,

$$\mathscr{T}' = \mathscr{R} \cup \mathscr{T}^*. \tag{5.2}$$

When new instances are created at the root node all instances in the dataset can be used to model the attribute distributions. As the tree grows each splitting test partitions the pattern space. Any new instances created at nodes below the root node should be instances that would have reached that node according to the split tests of the higher nodes. This means new instances should be sampled from the region of $\mathscr{X}$ that the current node represents. If this is not the case then the tree will recursively split on the same attribute at the same split point.

ExTree is able to create new instances that match this criteria by maintaining a set of constraints which flows down the tree with the training instances. These constraints specify what conditions an instance must have satisfied to have reached a node as determined by the splitting test above. For example the constraints for the rightmost leaf node in Figure 3.1 in Chapter 3 will be *Outlook* = *Rain*, *Windy* = *False*. The new instances created at this node would have to fulfil the constraints, so, in this case, only the *humidity* value is free. New instances are created by sampling from the region of $\mathscr{X}$ that is bounded by the constraints.

Consider a simple two class classification problem with classes Red and Blue. The patterns space $\mathscr{X} = x \times y$ where $0 \leq 0x \leq 1$ and $0 \leq y \leq 1$. The Blue class is Gaussian distributed and centred on the point (0.3, 0.25) with standard deviation of 0.25. The Red class is Gaussian distributed and centred around the point (0.75,0.75) with a standard deviation of 0.25.

Figure 5.4 shows this instance space. A fragment of the ExTree extracted decision tree is shown in Figure 5.3 with the nodes labelled 1 through 4. Figure 5.4 shows, for each of the 4 labelled nodes, how the constraints restrict the region of $\mathscr{X}$ that ExTree can sample. In Figure 5.4(a) there are no constraints on the instance space. In Figures 5.4(b)- 5.4(d) the area that each node represents in instance space is further constrained. This illustrates how, as the decision tree grows, the number of original instances used to select the splitting test decreases. New instances are created such that each bounded area has a constant number of instances.

Figure 5.5 shows how the kernel density estimates change as we move down the tree. By recalculating the KDE at each new node in the tree the local nature

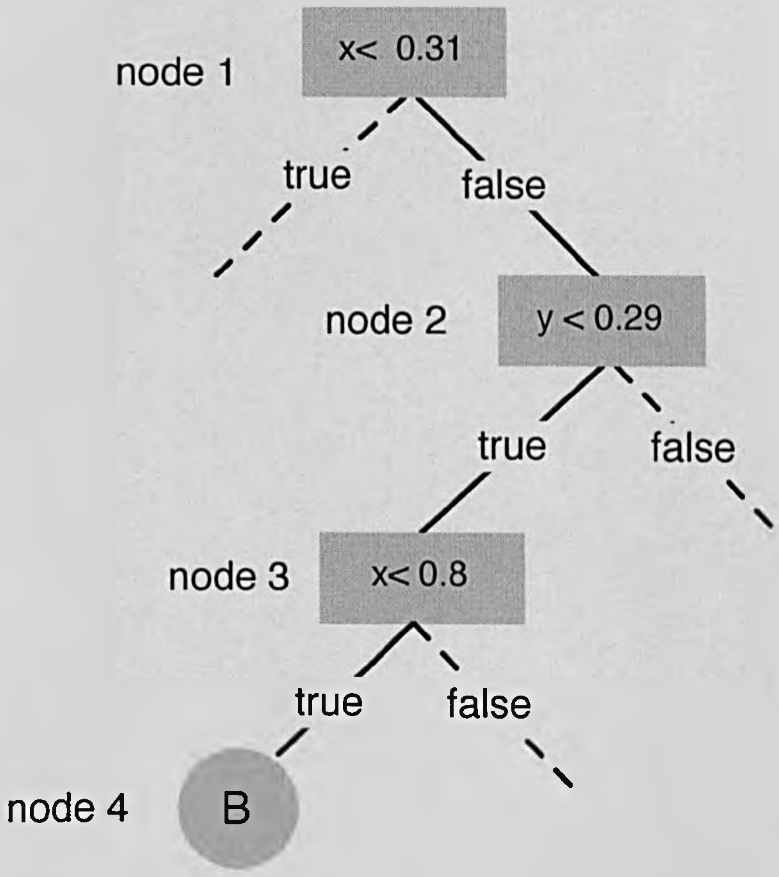
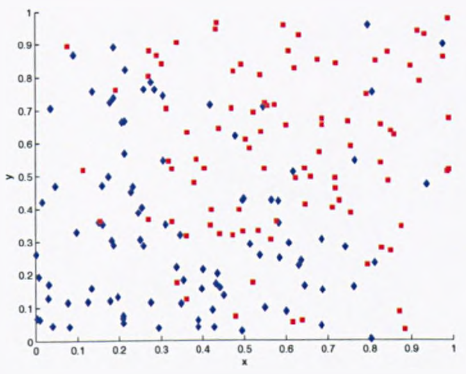


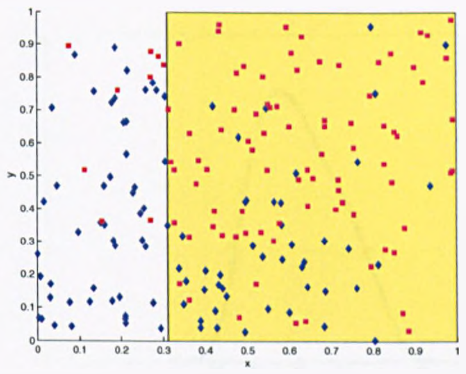
Figure 5.3: Decision tree fragment for red/blue problem.

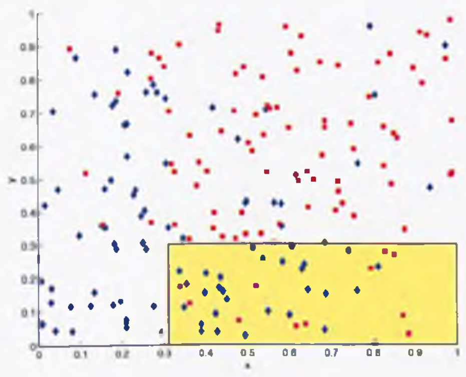
of relationships between the attributes is captured. Notice that the difference between node 3 and node 4 is a split on y but this still affects the kernel density estimate of x because the area of $\mathscr{X}$ is still further constrained, thus it is taking into account the relationship between x and y.

5.1.2 Class Relabelling

Real-world datasets often contain noise in the output attribute in the form of misclassified instances. If noisy data is used for decision tree induction it can result in overly large trees because it attempts to fit the noise. This can obscure the true relationship between the predictor attributes and the class variable. ExTree overcomes this by using the neural networks as an oracle


(a) Unconstrained Instance Space at Node 1


(b) Node 2


(c) Node 3

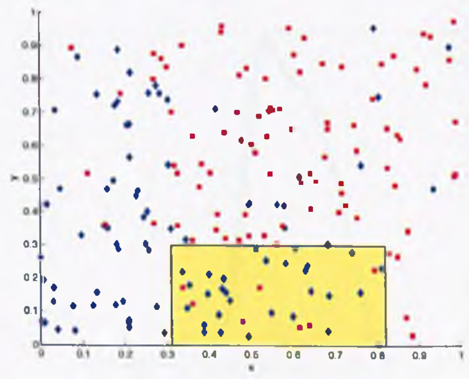
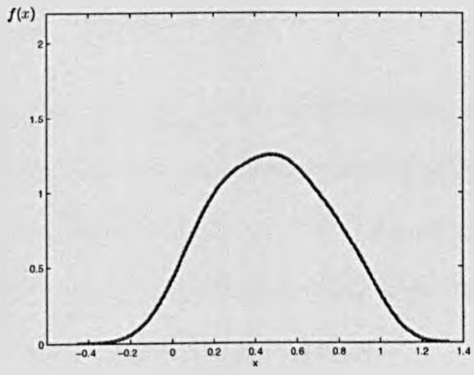
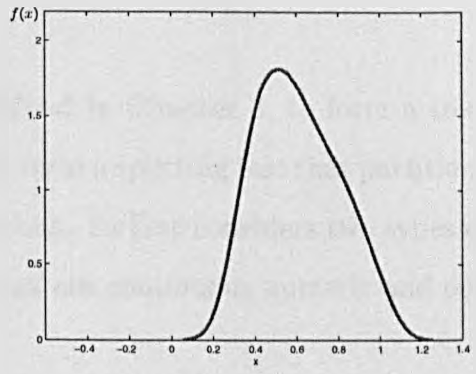

(d) Node 4

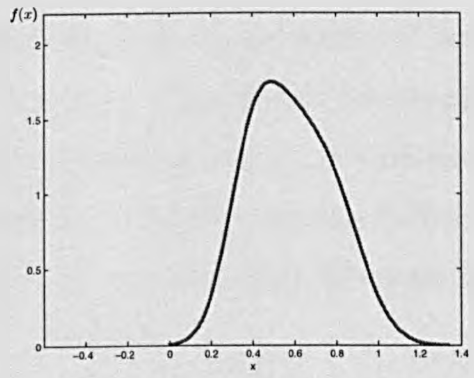
Figure 5.4: The yellow areas show the subspace of $\mathscr{X}$ that is bounded by the constraints at nodes 1 to 4.



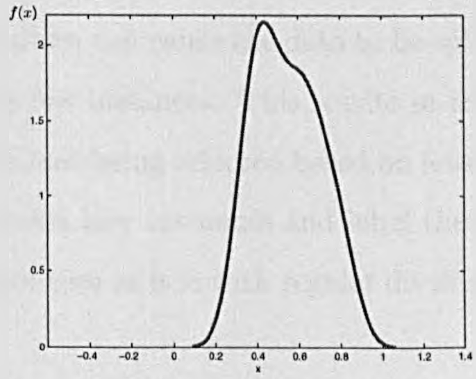
(a) No Constraints at Node 1 the root node



(b) Node 2



(c) Node 3



(d) Node 4

Figure 5.5: The kernel density estimation of x at the four labelled nodes.

to relabel the entire dataset. Because the primary aim of rule extraction is to model the neural network, relabelling the dataset using the neural network creates a noise-free dataset. A new dataset, $\mathscr{R}$, is created by relabelling the instances in the training dataset with the classification given to them by the neural network, such that

$$\mathscr{R} = \{X, \hat{c}(X)\}. \tag{5.3}$$

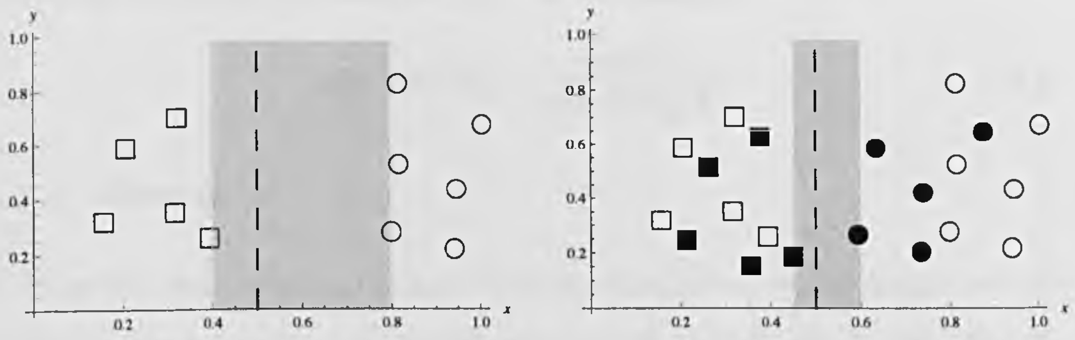
5.1.3 Split Types

ExTree uses recursive partitioning, as described in Chapter 3, to form a tree. Therefore, at each interior node ExTree must form a splitting test that partitions the instance space, $\mathscr{X}$, into two or more regions. ExTree considers two types of tests: one for attributes that have values that are continuous numeric and one for attributes that are discrete.

For discrete attributes, ExTree creates a branch for each possible value of the attribute. This can have the disadvantage, as described in Section 3.3.2, as attributes with a large number of possible values, can cause the data to be split into many partitions each containing only a few instances. This results in the splitting test created at the next level in the tree being selected based on fewer instances. However, because ExTree can create new instances and label them using the neural network this is not such a concern as it is with regular decision tree induction.

For a continuous numeric attribute, $V = v$, a binary split is made with two outcomes $v \leq z$ and $v > z$. The threshold value, z, is determined by first sorting the set of instances on the value of attribute V. For a set with n unique values in the dataset for attribute V there will be $n-1$ possible split points that could partition the set into two. ExTree uses the heuristic of choosing the midpoint of the bounding values as the split point, so for the set $\{10, 14, 20, 30\}$ the split points considered would be 12, 17, 25. Again, it is expected that by creating new instances ExTree will find a split point z closer to the optimum.

Consider the dataset in Figure 5.6(a), assume the optimal split point is at



(a) Original dataset (b) Augmented dataset

Figure 5.6: The optimal split point is shown by the dotted line. The shading indicates the region in which the splitpoint must lie given the dataset.

$x = 0.5$ as illustrated by the dotted line. The best potential split according to the dataset lies between 0.4 and 0.8, which is shown by the shaded area. Using the midpoint of the bounding values heuristic, the split point would be 0.6. In Figure 5.6(b) the dataset is augmented by additional instances (shown as solid markers) this reduces the bounds of the region in which the optimal split point must lie to the region bounded by 0.45 and 0.6. Applying the midpoint rule finds a split point at 0.525 which is much closer to the optimal.

5.1.4 Split Selection

To determine which one of the possible splits to use, ExTree uses Information Gain Ratio, which is a modification by Quinlan[65] to his Information Gain measure described in Chapter 3. Information Gain has an intrinsic bias towards selecting tests with many outcomes. Gain Ratio is determined by dividing the Information Gain by the Information gained by arbitrarily splitting the data into the number of outcomes resulting from the test. The information gained by arbitrarily splitting a set S into n subsets is given by

$$\text{split info}_n(X) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \times \log_2\left(\frac{|S_i|}{|S|}\right). \qquad (5.4)$$

The gain ratio of test X can, therefore, be calculated as

$$\text{gain ratio}(X) = \frac{\text{gain}(X)}{\text{split info}_n(X)}. \tag{5.5}$$

5.1.5 Pruning

ExTree only stops growing the tree when the set of instances reaching a node all belong to the same class, or the instances cannot be split any further. For the majority of datasets which contain noise this will lead to overfitting as described in Chapter 3. ExTree uses the method of subtree replacement to post-prune the decision tree. Pruning creates a smaller decision tree that would be expected to generalise better and be more comprehensible. Starting at the leaf nodes and working back towards the root, each subtree is tested to determine whether or not replacement would be beneficial. To determine if the replacement is beneficial the classification error on a validation set for the tree before and after the replacement is compared. If the tree after the replacement has a lower error then the subtree is replaced. As discussed in Section 3.1 using a subset of the training instances reduces the number of instances that can be used for the training phase. ExTree overcomes this limitation because it can create new instances and use these to augment the original training set. The pruning set can also be made as large as the original training set of instances.

5.2 Analysis on Synthetic Data

Once upon a time, in July 1991, the monks of Corsendonk Priory were faced with a school held in their priory, namely the 2nd European Summer School on Machine Learning. After listening more than one week to a wide variety of learning algorithms, they felt rather confused: Which algorithm would be optimal? And which one to avoid? As a consequence of this dilemma, they created a simple task on which all learning algorithms ought to be be compared: the three MONK's problems. – Sebastian Thrun.

x_1:Head shape	$\in$	{round,square,octagon}
x_2:Body shape	$\in$	{round,square,octagon}
x_3:Is smiling	$\in$	{yes,no}
x_4:Holding	$\in$	{sword,balloon,flag}
x_5:Jacket colour	$\in$	{red,yellow,green,blue}
x_6:Wearing a tie	$\in$	{yes,no}

Table 5.1: The attributes for the monk's robots dataset.

The Monk's problems[84] are a set of synthetic machine learning problems which illustrate certain characteristics found in real world problems. In the original study[84] 25 machine learning algorithms were compared, and the datasets have also been used in several other studies[5, 100]. Because the datasets are synthetic the mapping relationship, Eq. (1.1), between the inputs and outputs is known. This makes the problems particularly useful for both comparing and analysing machine learning algorithms. In this section ExTree is compared with C4.5 on the Monk's problems. This will show the deficiencies in C4.5 induced trees and how ExTree is able to overcome them by extracting a decision tree from a well-trained neural network

The three Monk's problems are based on a fictional robot domain. All three Monk's datasets are two-class classification problems with 6 attributes. For each dataset, the problem is to determine to which of the two potential classes a given robot belongs. The attributes, listed in Table 5.1, are all discrete and describe various characteristics of the robots. Because the pattern space, $\mathscr{X}$, is discrete its total size can be easily found by multiplying the number of elements in each attribute. This gives $3 \times 3 \times 2 \times 3 \times 4 \times 2 = 432$. The test set for each problem is the full 432 instances covering all of $\mathscr{X}$. The training set for each problem is a smaller subset of these instances. The two classes are coded such that class 1 is 'doesn't belong' and class 2 is 'belongs'.

Initial experiments were carried out to find suitable neural network training parameters and architectures. Neural networks generally performed well across

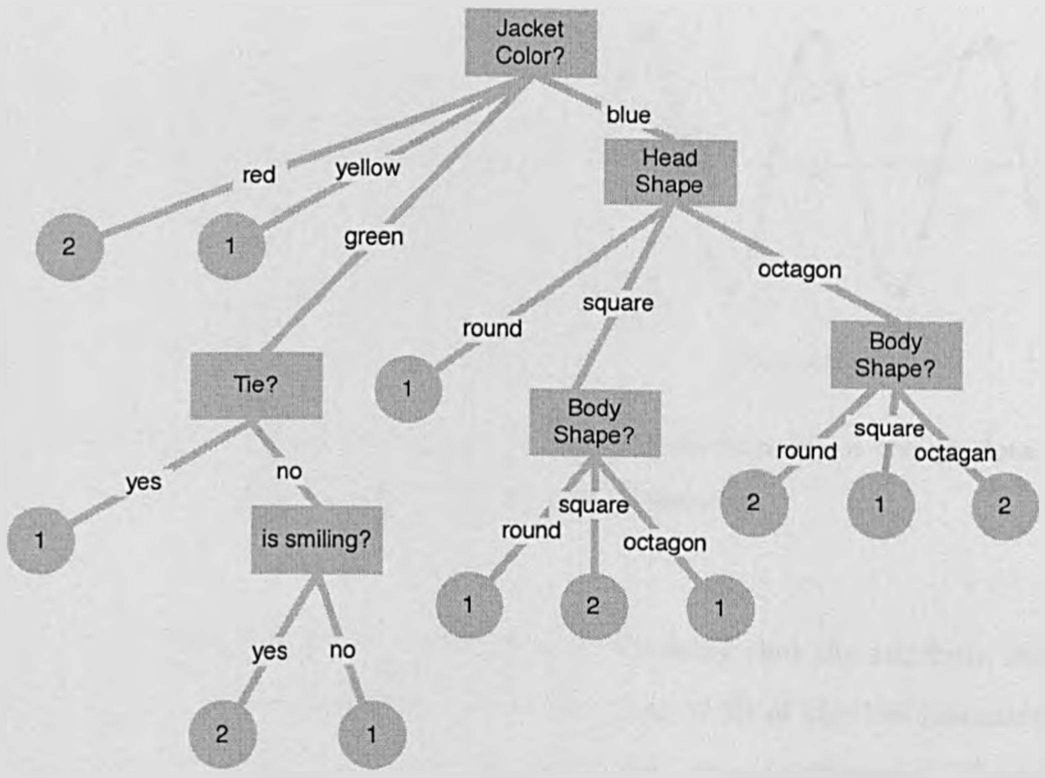


Figure 5.7: A C4.5 tree induced for monk training dataset 1.

a number of training parameters on the Monk's problems, demonstrating that neural networks are well-suited to these tasks.

5.2.1 Problem 1

The function for problem 1 is (*Head Shape* = *Body Shape*) or (*Jacket Colour* = *Red*). The training set is 124 randomly selected instances with no added noise. A neural network MLP was trained using error backpropagation to minimise the summed squared error for 500 epochs. The MLP had a single hidden layer with 5 neurons. The hidden and output layers used the unipolar sigmoid transfer function. A learning rate of 0.2 and momentum term of 0.3 was used. This neural network obtained 100% classification on both the training and test sets.

Figure 5.7 shows a C4.5 decision tree that was induced on the training set. This decision tree classified 76% of the testset correctly. This highlights some deficiencies with decision induction. In this case the attribute which gives the

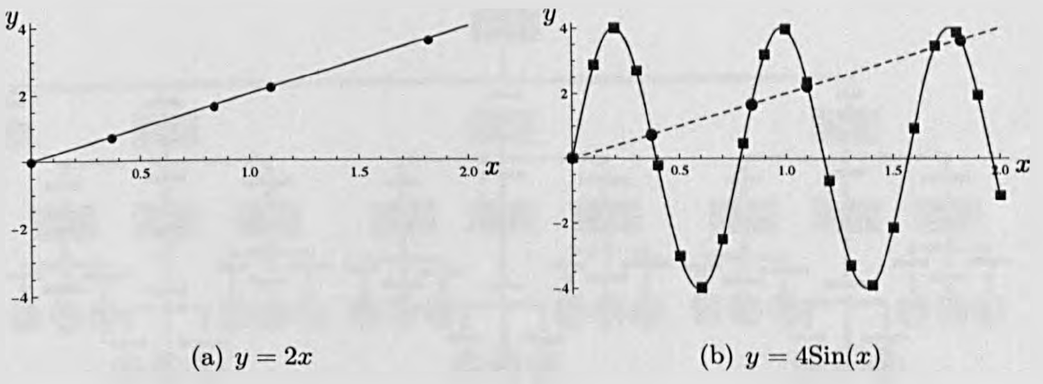


Figure 5.8: Phantom relations can appear when there is not enough data points to determine the true relationship.

most information initially is *Jacket Colour.* Knowing that the attribute *Jacket Colour* has a value of *Red* determines the class of 29 of the 124 instances so this is the correct first split from an information entropy viewpoint. However, after splitting on *Jacket Colour* this leaves only 95 instances to determine the more complex part of the problem *Head shape = Body shape.* The problem is exacerbated because the multiway split means that the remaining 95 instances are split between the red, green and blue branches. At this stage there are insufficient instances to determine the 'true' relationship between the attributes and information theory 'breaks down' selecting attributes which have no bearing on the classification. For example on the *Jacket Colour = Green* branch, a split is made based on *Tie*, then *is Smiling* this is a result of 'phantom' relationships in the data appearing stronger than the true global relationship because of lack of instances.

These 'phantom' relationships can arise because either the lack of instances allows strong local relationships in the instance space to be mistaken for global relationships or random relationships in the data can appear to be the true relationship. Consider the points in Figure 5.8(a), at first it appears the relationship between x and y is linear such as the function $y = 2x$ but after adding more points, as in Figure 5.8(b), the true relationship, $y = 4\text{Sin}(8x)$, becomes clear. The solution is to acquire more information by creating new instances.

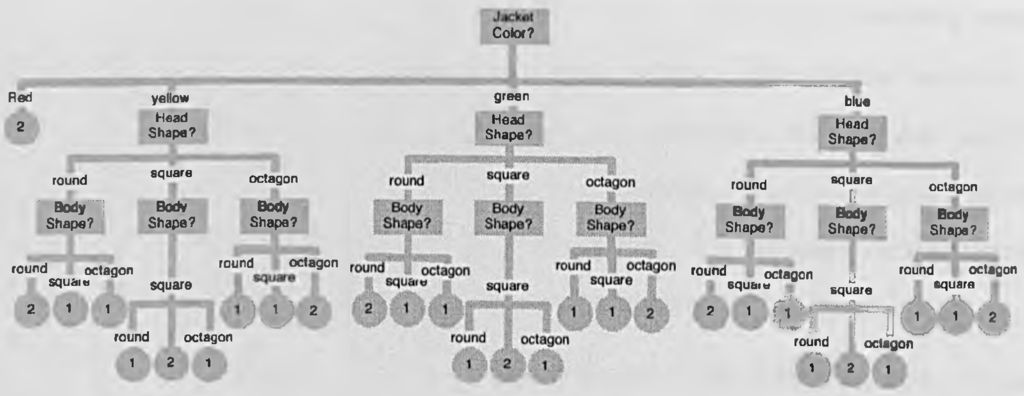


Figure 5.9: An ExTree tree for the monk training dataset 1.

Figure 5.9 shows a decision tree extracted from the previously trained neural network using ExTree, which sampled an extra 500 points at each node. This decision tree achieved 100% accuracy on both the training and test set and therefore, also 100% fidelity with the neural network. The ExTree extracted decision tree starts the same as the C4.5 decision tree with a split on *Jacket Colour*. After this point we can see the advantage of the sampling and querying approach. At each of the 3 colour branches another 500 instances are created and labelled by the neural network. This allowed the decision to represent the correct relationship *Head Shape* == *Body Shape* at each node. This example shows the advantage of the sampling and querying and how this leads to better splitting decisions.

5.2.2 Problem 2

The function for problem 2 is that exactly two of the six attributes must have their first value. For example, if *Body Shape* and *Head Shape* both equal their first value, *round*, this would imply that for the robot to be a member of class 2 the robot's other attributes were not equal to their first value that is *is smiling* ≠ *yes*, *holding* ≠ *sword*, *jacket colour* ≠ *red*, and *has tie* ≠ *yes*. This problem is similar to parity problems and is particularly difficult for decision tree algorithms[84]. The training set for this problem is 169 randomly selected instances, with no noise added.

A neural network was trained on the data. The neural network used a learning rate of 0.5 and a momentum term of 0.2. The neural network, as before, used 5 hidden neurons and one output neuron. All neurons used the unipolar sigmoid function as the activation function. The neural network was trained for 500 epochs of error backpropagation. The trained neural network obtained a 100% classification rate for both training and test sets.

A C4.5 decision was induced on the dataset. This tree achieved a classification rate of 65% which is lower than selecting the majority class (class 1) for every instance, which, for this test set, would have correctly classified 67% of the instances. This problem requires global knowledge meaning all six of the attributes' values are required to make a decision at any point in $\mathscr{X}$. This is in contrast to other problems where in certain parts of $\mathscr{X}$ only local knowledge represented by a subset of the attribute values is required. This is not a problem for neural networks such as MLPs which learn such tasks well because they consider each instance as a whole. In contrast, decision tree induction performs badly on these problems because they consider each attribute in turn. With this dataset, no attribute in isolation gives more information about the class than the others. It is only after each attribute has been considered that a classification is possible. However, it is possible for a decision tree to represent such functions by splitting on each attribute in turn, regardless of the small amount of information gained, to create a full tree where each path between the root and leaf nodes tests each attribute. In practice, decision tree algorithms can rarely achieve this because, as in Monk Problem 1, the number of instances in the dataset does not sufficiently cover $\mathscr{X}$.

ExTree with its sampling and querying technique can generate these new instances to provide adequate coverage of $\mathscr{X}$. An ExTree extracted decision tree from the trained neural network achieved a 100% classification rate on both the training and test set.

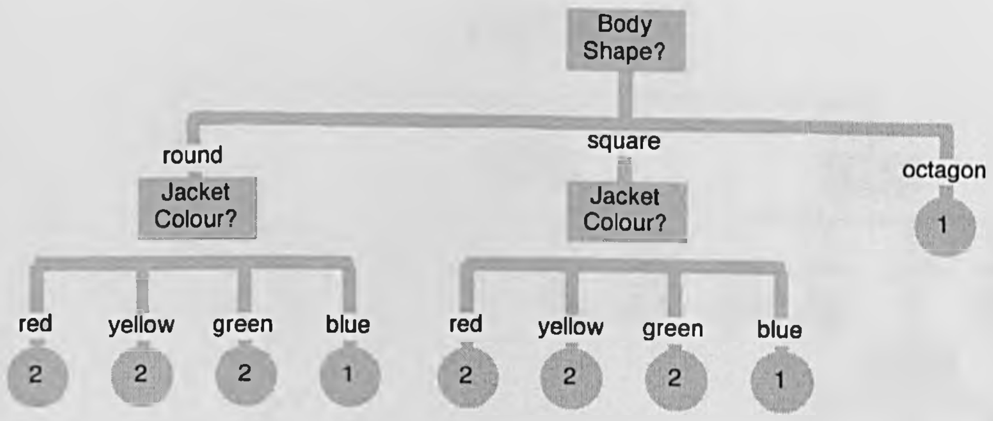


Figure 5.10: A C.45 induced decision tree for Monk problem 3.

5.2.3 Problem 3

The function for problem 3 is (*jacket colour* = *green* and *holding* = *sword*) $\vee$ (*jacket colour* $\neq$ *blue* and *body shape* $\neq$ *octagon*) . This is not a particularly hard problem for decision trees or neural networks, but in the training dataset for this problem 5% of the instances are misclassified. This introduces an element of noise into the dataset. Figure 5.10 shows a decision tree induced on the training set. C4.5 does well on this problem and correctly classifies 93% of the training data and 97% of the test data. However, the C4.5 tree ignores the second part of the function (*jacket colour* $\neq$ *blue* and *body shape* $\neq$ *octagon*) capturing only the relationship (*jacket colour* = *green* and *holding* = *sword*). This results in 12 misclassified instances in the test set. A neural network was created which correctly classified 95% of the training data and 100% of the noise-free test data. The neural network used 2 hidden layers with bipolar sigmoid activation functions. The ExTree extracted decision tree is shown in Figure 5.11. The ExTree decision tree discovered the full relationship between the attributes including the (*jacket colour* $\neq$ *blue* and *body shape* $\neq$ *octagon*) part missed by the C4.5 tree. By using the neural network as an oracle to relabel the misclassified instances in the training set the extracted tree was unaffected by the noise and therefore achieved 100% classification on the test set.

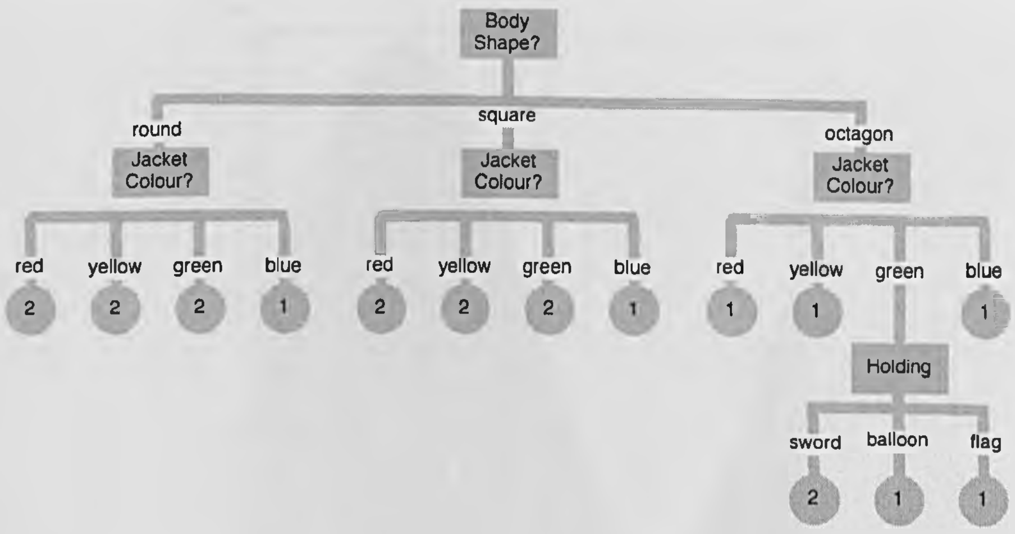


Figure 5.11: An ExTree decision tree for Monk problem 3.

5.2.4 Continuous Problem 1

The monk datasets are useful for the analysis of machine learning classifiers but do not include continuous attributes in the dataset. To examine how ExTree handles continuous data a simple synthetic dataset was created. This classification problem is a two class classification problem. The pattern space is defined by $\mathscr{X} = x \times y : 0 \leq x \leq 2\pi, -1 \leq y \leq 1$. The point (x, y), if in the region defined by $y < \sin(x)$, is of class 1, else it is class 2. Figure 5.12 illustrates the pattern space.

The training set for this problem consisted of 100 random points sampled from $\mathscr{X}$. The test consisted of 400 uniformly sampled points covering $\mathscr{X}$. A neural network with 3 hidden neurons can learn this problem with an accuracy on the test set of 97%. A C4.5 induced decision tree, as shown in Figure 5.13(a), only learns the problem to an accuracy of 85%.

The approximation to the decision boundary is shown in Figure 5.14(a). The region that classified correctly as Class 1 is shaded blue. The region incorrectly classified as Class 1 is shaded green. The region correctly classified as Class 2 as white without shading. The region incorrectly classified as Class 2 is shaded

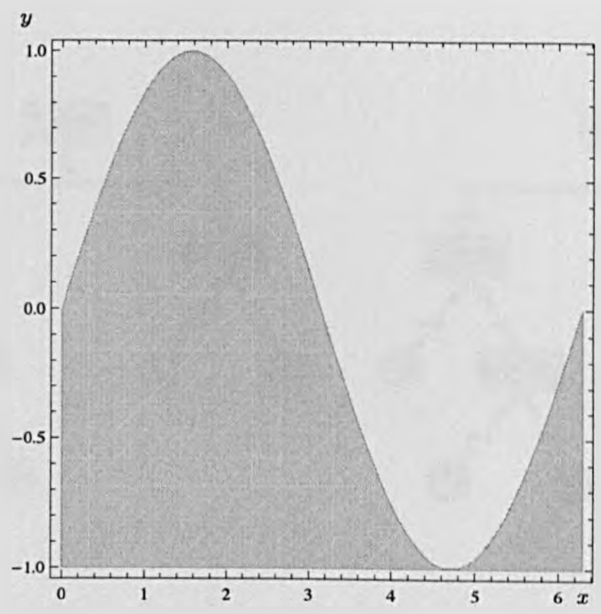


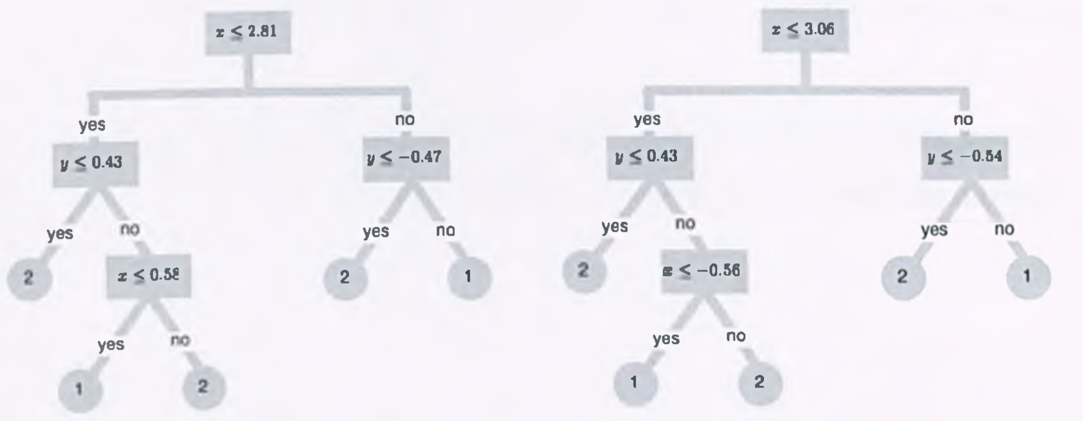
Figure 5.12: The pattern space for continuous problem 1. The shaded region is class 1 and the unshaded region is class 2.

Instances	Accuracy	Fidelity	Nodes	Leaves	Depth
0	87%	84%	9	5	4
100	88%	89%	13	7	5
200	93%	94%	21	11	6

Table 5.2: As the number of new instances increases the fidelity and accuracy for ExTree increases on the Sine dataset.

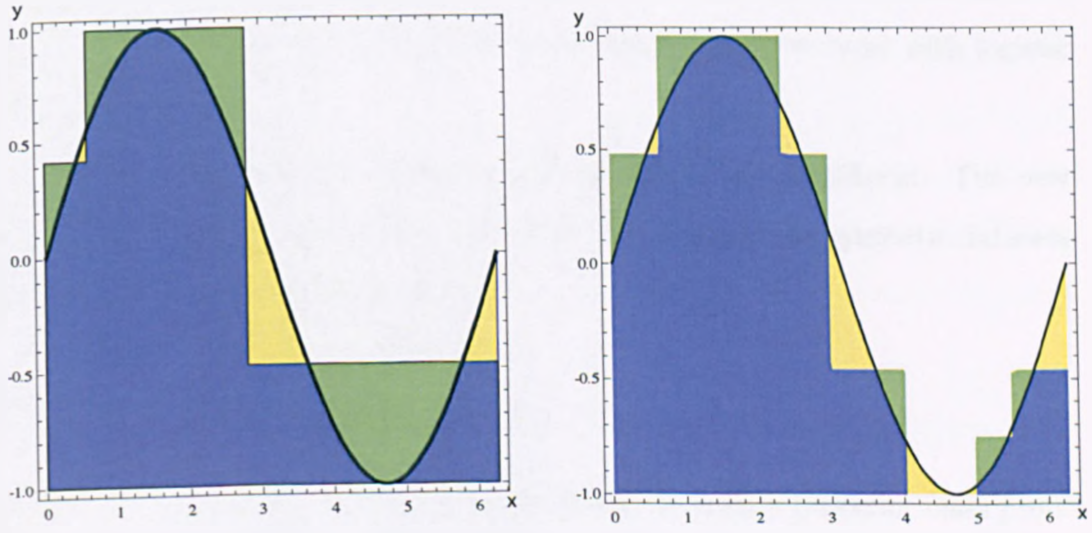
yellow. As shown in Figure 5.14(a) the stepwise approximation results in many errors close to the decision boundary.

ExTree can improve on this performance in two ways. First, it can find better split points. To demonstrate this ExTree was confined to creating a tree of the same size as the C4.5 tree but allowed to refine the split points. The resulting tree is shown in Figure 5.13(b). The slight refinement in the split points increased the accuracy to 87% and the fidelity to 84%. Second, by



(a) C4.5 (b) ExTree

Figure 5.13: C4.5 and ExTree decision trees for the Sine dataset .



(a) C4.5 (b) ExTree

Figure 5.14: Decision boundaries for C4.5 and ExTree on the Sine dataset.

increasing the number of instances ExTree can more closely approximate the curved decision boundary, as shown in Figure 5.14(b), but the tree grows in complexity.

Table 5.2 shows the improvement in fidelity with the neural network and the accuracy as the number of new instances are created and the resulting increase in tree size.

5.3 ExLMT

In the first part of this Chapter, ExTree, a rule extraction method for classification problems was developed. In this section an extension of this method will be developed. The results on the real-world datasets in the next Chapter shows that on many datasets there remains a significant gap between the knowledge discovered by the neural network and the knowledge extracted by ExTree. To attempt to overcome this ExLMT was developed. This new method extracts Logistic Model Trees(LMTs)[46]. These are a recent addition to the field of decision trees that replace the leaf nodes of regular decision trees with logistic regression functions.

The following sections review logistic regression and logitBoost. The new ExLMT algorithm is described. ExLMT is then analysed on synthetic datasets to see how it can improve on ExTree.

5.3.1 Logistic Regression

Logistic regression is a statistical method used to predict posterior class probabilities $P(C = k \mid X = x)$ for the K classes. Logistic regression fits a logistic function of the form

$$y = \frac{e^{\beta_0+\beta_1 x_1}}{1 + e^{\beta_0+\beta_1 x_1}} \tag{5.6}$$

to the class probabilities, where β_i are the parameters to be estimated. The motivation for using the logistic function can be demonstrated by considering

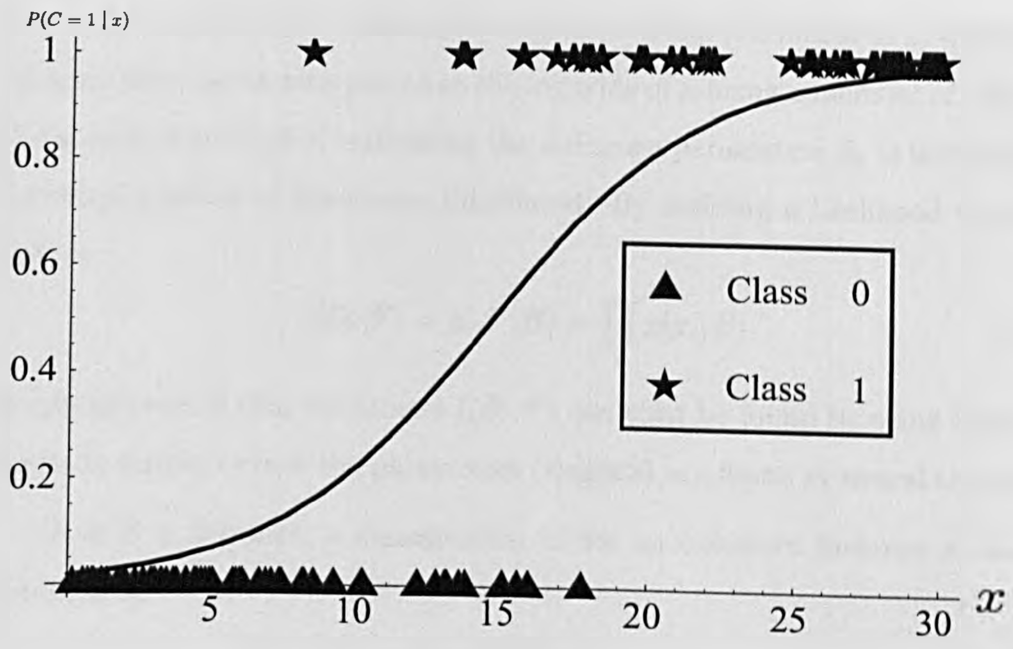


Figure 5.15: Logistic regression curve giving posterior probabilities a two class problem.

the univariate case with a dichotomous dependent class variable.

Figure 5.15 shows a typical two class problem separable by the independent variable x. For low values of x, C_1 is more probable and for high values C_2 becomes more probable. As can be seen in Figure 5.15 the logistic curve provides an excellent approximation to the posterior probability $P(C = 1 \mid x)$. Points close to $x = 0$ have a low probability of belonging to Class 1 which increases rapidly as the points move closer to $x = 30$.

When dealing with the multivariate case the posterior probabilities become

$$P\left(C = k \mid x\right) = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ \cdots\ +\beta_p x_p}}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ \cdots\ +\beta_p x_p}}. \tag{5.7}$$

By applying the logit transform,

$$\text{logit}(x) = \log\left(\frac{x}{1-x}\right), \tag{5.8}$$

to Eq. (5.7) achieves

$$\log\left(\frac{P(C = k \mid x)}{1 - P(C = k \mid x)}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ \cdots\ + \beta_p x_p\ . \tag{5.9}$$

This has the advantage of making the right side of Eq. (5.9)linear in x, whilst the left hand side can be interpreted as the log odds of x being a member of Class k. The standard method of estimating the unknown parameters β_n is through the statistical method of Maximum Likelihood. By defining a likelihood function such as

$$\ell(\beta; \mathscr{T}) = p(\mathscr{T}; \beta) = \prod_i p(x_i; \beta), \tag{5.10}$$

the parameters $\hat{\beta}$ that maximises $\ell(\beta; \mathscr{T})$ can then be found by using iterative methods similar to how the parameters (weights) are found in neural networks.

Once $\hat{\beta}$ is obtained, a classification k^* for an unknown instance x can be obtained by

$$k^* = \arg\max_k P(C = k \mid X = x). \tag{5.11}$$

5.3.1.1 LogitBoost

Friedman[27] analysed boosting algorithms by recasting them as additive models, which are well-documented in the statistical literature. This analysis resulted in a new boosting algorithm, LogitBoost.

Figure 5.16 shows an overview of the algorithm. LogitBoost uses an ensemble of functions F_k to predict classes $1...K$ using M 'weak learners'

$$F_k(x) = \sum_{m=1}^{K} f_{mk}(x). \tag{5.12}$$

Each of the 'weak learners' f_{mk} can be any algorithm that fulfils Eq. (1.1). When f_{mk} are linear functions in x then F_{mk} is equivalent to the logistic model previously described. Therefore, the LogitBoost algorithm can be seen as an iterative Newton method of fitting the logistic regression function. This iterative nature is exploited by the logistic model tree growing stage of the ExLMT algorithm described in Section 5.4.

```
1. Start with weights $w_{ik} = 1/N, i = 1, \ldots, N, k = 1, \ldots, J, F_k(x) = 0$
   and $p_k(x) = 1/K \quad \forall k$

2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Repeat for $k = 1, \ldots, K$:
       i. Compute working responses and weights in the $k$th class
          $$z_{ik} = \frac{y^*_{ik} - p_k(x_i)}{p_k(x_i)(1 - p_k(x_i))}$$
          $$w_{ik} = p_k(x_i)(1 - p_k(x_i))$$
       ii. Fit the function $f_mk(x)$ by a weighted least-squares regression
           of $z_{ik}$ to $x_i$ with weights $w_{ik}$
   (b) Set
       $$f_{mk}(x) \leftarrow \frac{K-1}{K}\left(f_{mk}(x) - \frac{1}{K}\sum_{j=1}^{K} f_{mj}(x)\right)$$
       $$F_k(x) \leftarrow F_k(x) + f_{mk}$$
   (c) $P_k(x) \leftarrow \frac{e^{F}_{k}(x)}{\sum_{j=1}^{K} e^{F_j(x)}}$,

3. Output Classifier
   $$\arg\max_k F_k(x)$$
```

Figure 5.16: LogitBoost: an adaptive newton algorithm(adapted from Friedman[27]).

5.4 The ExLMT Algorithm

The ExLMT algorithm is similar to the ExTree algorithm given previously. It follows the same top down recursive partitioning approach and uses information gain ratio to determine the attribute to split on. The main difference is the additional creation and refinement of the logistic models.

Figure 5.17 shows a flowchart of the ExLMT algorithm. Comparing the ExTree algorithm (Figure 5.1) with the ExLMT algorithm the main difference is the creation and refinement of the logistic regression model.

Create Initial Logistic Regression Model An initial Logistic regression model is built using all the data in the relabelled and expanded training set $\mathscr{T}'$. The logistic regression model is then fitted using the LogitBoost method[27]. LogitBoost uses an ensemble of functions F_k to predict classes $1...K$ using M 'weak learners'. Figure 5.16 details the LogitBoost algorithm as originally given by Friedman[27].

Refine Logistic Model For each node resulting from the split created at the previous stage the logistic regression function is refined based only on the subset of $\mathscr{T}$ that reached that node. This refinement means that as the tree grows the logistic regression models capture information local to the region of $\mathscr{X}$ that the tree structure above has partitioned. Because of the iterative and additive nature of the LogitBoost algorithm, the refinement is simply running more iterations on a copy of the LogitBoost model of the node above but using only the subset of $\mathscr{T}$ that reached the node.

5.4.1 Illustrative Example

Consider the simple 2 attribute 2 classification problem

$$c(x,y) = \begin{cases} 1 & (y > \ 0.7 + 0.65 \wedge x \leq 0.5) \\ 1 & (y > -0.4x + 0.4 \wedge x > 0.5) \\ 0 & \text{elsewhere.} \end{cases} \tag{5.13}$$

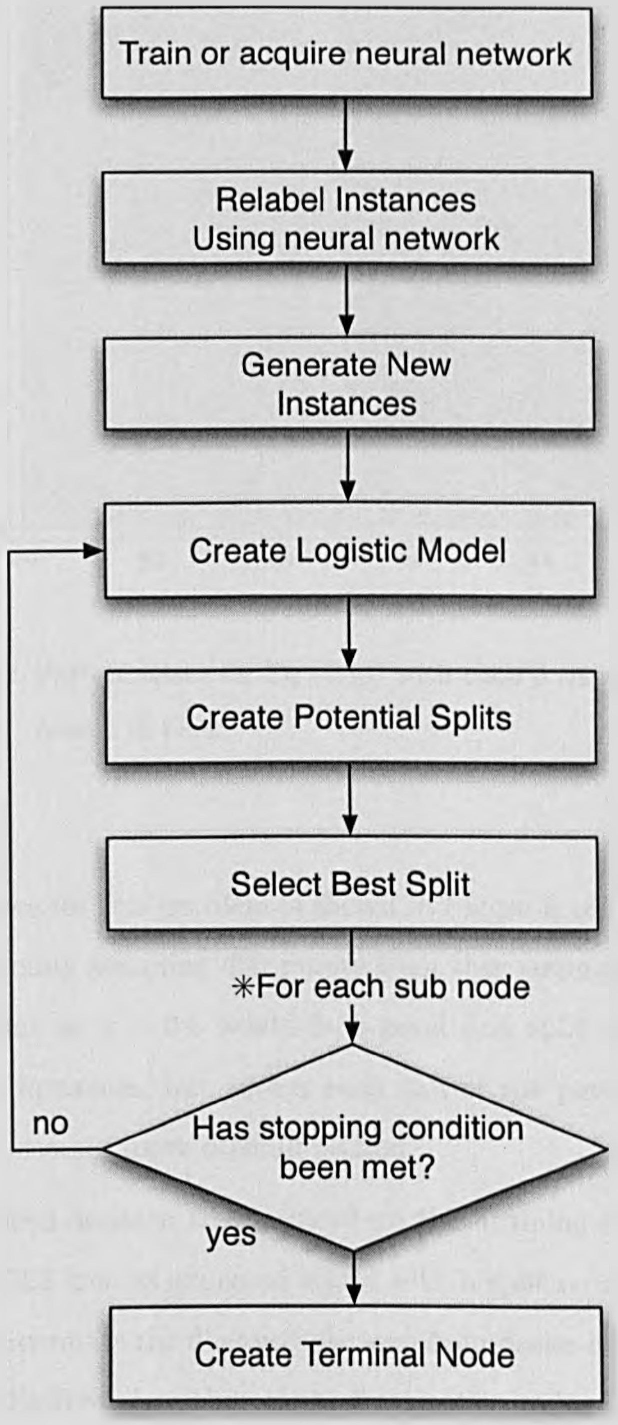


Figure 5.17: Pseudo-code for the ExLMT algorithm.

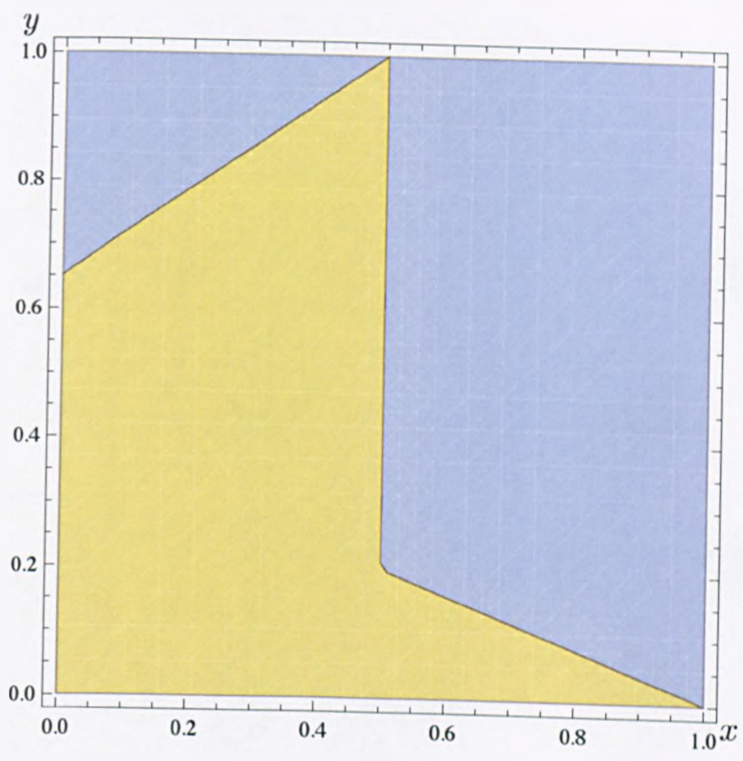


Figure 5.18: Pattern space for Eq (5.13) with class 0 shown in yellow and class 1 in blue.

The pattern space for this problem is shown in Figure 5.18. A training set was created by randomly sampling 400 points from this mapping. It can be easily seen that splitting at $x = 0.5$ would be a good first split which separates the majority of the instances, but within each half of the pattern space the next splitting point becomes more difficult to find.

A C4.5 induced decision tree induced on this training set is shown in Figure 5.19. The C4.5 tree as expected starts with a split on x around 0.5. Then proceeds to approximate the diagonal decision boundaries in each half with step functions. The ExTree algorithm of the last section only made small improvements to this. It finds slightly better split points and by creating new instances can increase the accuracy of the approximation to the diagonal decision boundaries, but at the cost of increasing the complexity of the tree. By replacing the leaf nodes of the decision tree with the logistic models much smaller trees can

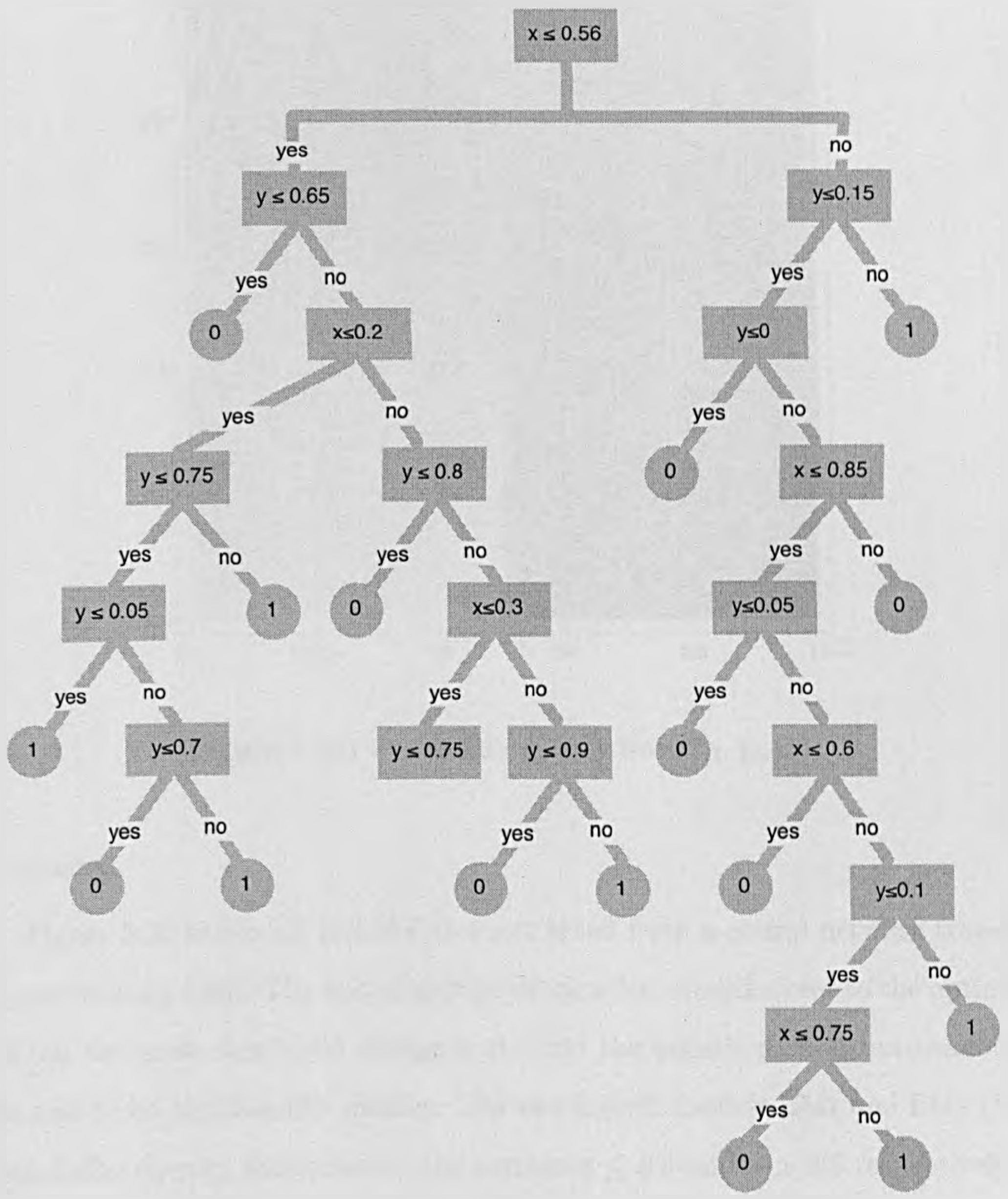
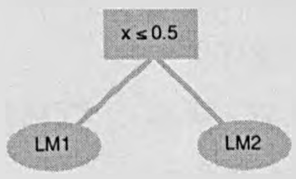


Figure 5.19: A C4.5 tree approximation to Eq. (5.13).



LMT1: $15.34 + 16.84x + -23.64y$

LMT2: $-12.52 - 12.49x + 31.05y$

Figure 5.20: ExLMT approximation to Eq. (5.13).

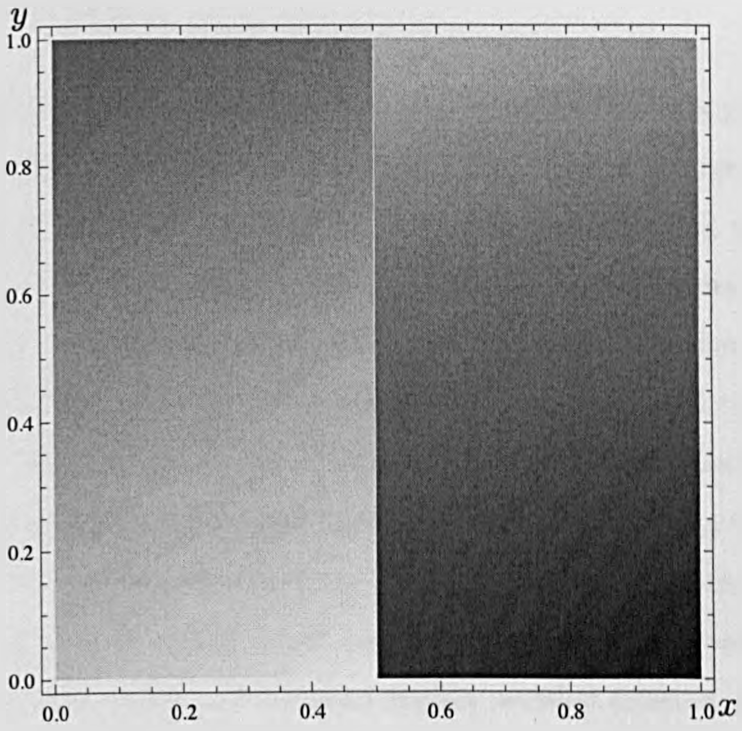


Figure 5.21: Probability density from Eq. (5.13).

be created.

Figure 5.20 shows an ExLMT tree extracted from a neural network trained on the training data. The initial split point on x has moved closer to the optimal 0.5 but the most significant change is the way the logistic models have enabled the tree to be significantly smaller. The two logistic models LM1 and LM2 give probability density functions for the regions $x \leq 0.5$ and $x > 0.5$ respectively.

The probability density function for the pattern space is shown in Figure 5.21. The exact decision boundaries can be calculated from the Logistic models by finding the sign boundaries LM1=LM2=0 and rearranging for y

$$\begin{aligned} \text{LMT1:}\quad & 15.34 + 16.84\text{x} + \text{-}23.64\text{y} &= 0 \\ & \text{y} = 0.71\text{x} + 0.65 & \\ \text{LMT2:}\quad & \text{-}12.52 - 12.49\text{x} + 31.05\text{y} &= 0 \\ & \text{y} = 0.4\text{x} + 0.4\ \ . & \end{aligned}$$

This shows that the decision boundaries are very close approximations to the original generating function Eq. (5.13).

5.5 Summary

In this Chapter the new rule extraction algorithm ExTree was presented. Using the taxonomy provided by Andrews[4], discussed, in Chapter 1, ExTree is a pedagogical method and uses the sampling and querying approach first proposed by Craven[16] meaning it is independent of the neural network architecture and training algorithm. The new algorithm was then analysed on the synthetic monk datasets which showed that the extracted rules were able to achieve 100% fidelity with the neural network and 100% classification accuracy. This improvement, although significant, does have a cost in computational time and also increases the size of the tree. The Monk's problems contain only discrete attributes therefore a continuous attribute problem was created this demonstrated the ability of ExTree to extract decision trees on such problems and also produced trees which had improved classification accuracy over C4.5. This dataset also demonstrated a limitation of ExTree which is that it can only create axis-parallel decision boundaries like C4.5. Therefore, this chapter also introduced an extension of the previous ExTree algorithm, ExLMT, which replaced the leaf nodes with logistic models. This allowed the leaf nodes to model non-axis parallel splits. An example of ExLMT was then given which showed how it was possible to extract a very close approximation to the original generating function of a dataset from a neural network trained on that dataset. In the next chapter both these algorithms will be evaluated on real-world datasets.

CHAPTER 6

Empirical Evaluation of ExTree

The novel ExTree algorithm was introduced in the previous chapter. The algorithm was successfully applied to the synthetic Monk's datasets[84]. Although an algorithm can be demonstrated to work on idealised synthetic data it is important to demonstrate it on real world data[61]. Real-world datasets by their nature have unknown characteristics which present unique challenges to machine learning algorithms. In this Chapter ExTree is evaluated on a number of real-world datasets. The performance on these datasets is compared to C4.5 and MLPs.

6.1 Evaluation Methodology

To evaluate the effectiveness of ExTree an evaluation methodology was required. There are three measures used to evaluate the ExTree algorithm with other machine learning algorithms: predictive accuracy, fidelity and tree size. Predictive accuracy is a measure of how well a machine learning algorithm classifies previously unseen instances and is simply the percentage of instances in a test set which the classifier $\hat{C}$ classifies correctly. Given a set of n instances with known classifications $t_1 \dots t_n$ and the classifications, $y_1 \dots y_n$, given to these instances by a classifier, the classification accuracy is

$$\text{accuracy} = \frac{\sum_i^n \text{eq}(y_i, t_i)}{n}, \tag{6.1}$$

where

$$\text{eq}(x, y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{elsewhere.} \end{cases} \tag{6.2}$$

To measure how well the extracted decision tree models the knowledge within the neural network the fidelity measure is used. This is the percentage of instances in a given dataset for which two classifiers predict the same class. Given the classifications from the first classifier, $y_1^1 \dots y_n^1$, and the classifications from the second classifier, $y_1^2 \dots y_n^2$, fidelity can be calculated as

$$\text{fidelity} = \frac{\sum_i^n \text{eq}(y_i^1, y_i^2)}{n}. \tag{6.3}$$

To evaluate comprehensibility the tree size is used, measured as the total number of nodes in the tree including the root node, interior nodes and leaf nodes.

6.1.1 k-Fold Cross Validation

In evaluating a classification algorithm, the dataset plays two roles: first as training data from which to build the model, and second as test data to estimate how well the model can predict. The obvious way to achieve this is to split the data into two subsets and use the first to train and the second to test the model. A problem using this approach is that with small datasets, such as some of the common benchmark datasets, 50% of the dataset is not sufficiently representative enough of the problem domain to build a model. Furthermore, this approach is very unstable, which means that the prediction of classification accuracy can fluctuate substantially depending on which instances are assigned to which subset. To overcome this problem k-fold cross validation[83] can be used. The first step is to randomly split the data into k subsets or folds. For each subset a model is trained on all the $k-1$ other subsets then tested on the remaining subset. The final result is found by summing the correctly classified instances in each of the k folds and dividing by the total number of instances. For example, if the dataset had 100 instances and 5-fold validation was used each of the 5 subsets would contain 20 instances.

Figure 6.1 shows how 5-fold cross validation would split a dataset into training and test subsets. A model would then be created for each of the 5 subsets. Each model would be tested on one of the 5 subsets with the remaining 4 sub-

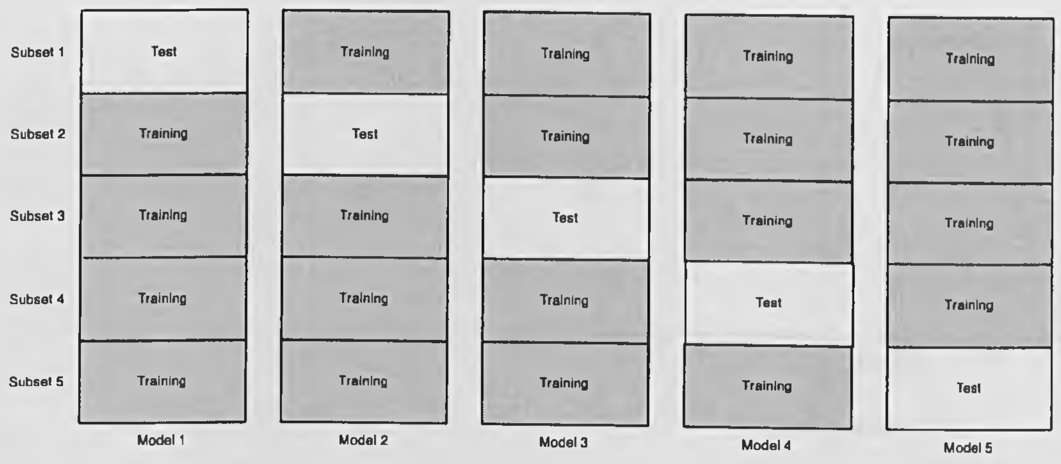
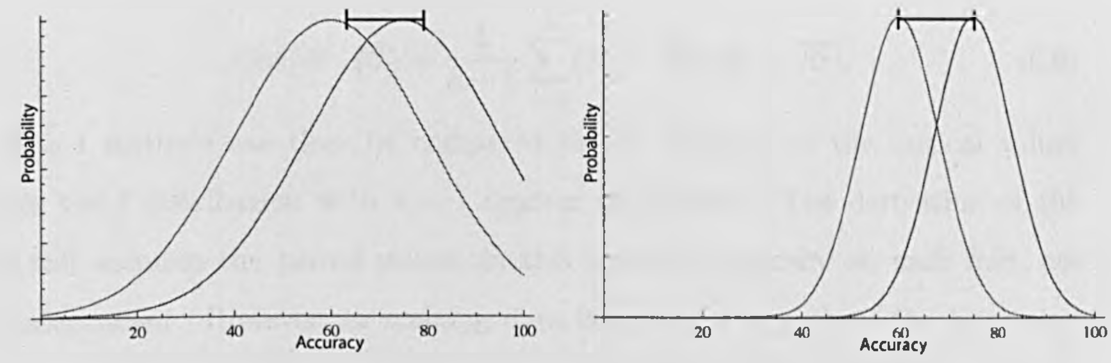


Figure 6.1: k-Fold Cross Validation for k=5.

sets being used as training data. The classification accuracy for the dataset can be calculated by summing how many of the 20 instances each of the 5 models classified correctly by dividing by 100 which is the total number of instances in the dataset. This produces an unbiased[6] estimate of the predictive accuracy of the model on the dataset.

A further refinement to cross validation is stratification which attempts to keep the proportion of classes in each fold equal to the proportion in the whole dataset. A k-fold cross validation can still be sensitive to which instances are in which fold. This is particularly problematic for small datasets or datasets with a large amount of noise. To improve the estimate of the predictive accuracy, the whole of the k-fold cross validation process itself can be repeated a number of times with the instances being randomly assigned to each fold. This, however, increases the amount of computation required to estimate the predictive accuracy of the machine learning algorithm being tested. The estimates of predictive accuracy, fidelity and tree size in this Chapter use 10-fold cross validation. The use of 10 folds is somewhat arbitrary but is the standard for estimating predictive accuracy and has been used in many machine learning comparisons such as the Statlog project[52]. The 10 fold validation itself is repeated 10 times for each dataset, requiring the machine learning algorithm to be run 100 times to obtain estimates for just one dataset.



(a) Comparison 1 (b) Comparison 2

Figure 6.2: A comparison of algorithms with different deviations but same difference in mean.

6.1.2 Comparing Two Machine Learning Algorithms on a Single Dataset

The k-fold cross validation results in k results for each algorithm. Therefore, comparing two machine learning algorithms creates two sets of results, R^1, R^2, each containing k results. It is reasonable to assume that these results will be normally distributed for each machine learning algorithm as shown in Figure 6.2.

The further apart the distributions are the more likely it is that the difference between the algorithms is significant. However, as shown in Figure 6.2, the deviation of the results also needs to be taken into account. The more the distributions overlap the more likely it becomes that there is no significant difference between the algorithms. The paired t-test[62] is a statistical measure that allows quantification of the difference between two sets of paired values such as the results from the k-fold cross validation. The t-test divides the difference in the mean of the two results by the standard error, S_d which measures the spread of the two sets of results. This gives the following formulae to calculate the t value,

$$t = \frac{\overline{R^1} - \overline{R^2}}{S_d}, \tag{6.4}$$

$$S_d = \sqrt{\frac{\mathrm{Var}(R^1) + \mathrm{Var}(R^2) - 2\,\mathrm{Cov}(R^1, R^2)}{k}}, \tag{6.5}$$

$$\text{Cov}(R^1, R^2) = \frac{1}{k-1}\sum_{i=1}^{k}(R_i^1 - \overline{R^1})(R_i^2 - \overline{R^2}). \tag{6.6}$$

The t statistic can then be compared to the tables[1] of the critical values for the t distribution with $k - 1$ degrees of freedom. The derivation of the t-test assumes the paired values, in this case the accuracy on each fold, are independent. However, as training data is repeated in each of the k-training datasets(Figure 6.1), this assumption is violated. Therefore, although it is common practice within the machine learning community[6] the results of a t-test on a k-fold cross validation should be viewed with some caution. In the results tables that follow in this section t-tests shown to be significant are indicated in bold.

6.1.3 Comparing Two Machine Learning Algorithms on Multiple Datasets

The last section showed how a paired t-test method can be used to test the significance of two machine learning algorithms on a single dataset. It is also useful to compare learning algorithms across multiple datasets. Again, there are two algorithms with a number of paired results, so, superficially, the t-test may seem appropriate. However, in this situation the t-test is unsuitable as there is no reason to expect the results for an algorithm to be normally distributed over different datasets. The datasets are completely independent of one another because the accuracy achieved on one dataset has no influence on the accuracy achieved on another dataset.

Comparing two machine learning algorithms on m datasets creates $d_1 \dots d_m$ differences. For functionally identical algorithms these differences would be 0. The larger the magnitude of the differences the more likely the difference between the two algorithms is significant. The Wilcoxon signed-rank test allows us to test whether the magnitude of the differences are significant. This test is similar to the paired t-test but is non-parametric so does not make the assumption that the observations are normally distributed. The null hypothesis for this test is that the two samples were drawn from identical populations, that is the difference between the algorithms is insignificant. It is calculated by ranking the $d_1 \ldots d_m$ by magnitude in ascending order such that each d_j is assigned a rank of r_i. The W^+ statistic can then be calculated as

$$W^+ = \sum_{r_i>0} r_i. \tag{6.7}$$

The Wilcoxon signed-rank test is determining whether the differences are distributed symmetrically around 0 which would indicate there is no significant difference between the algorithms being tested. The W^+ statistic is then evaluated against standard statistical tables to determine if it is significant[1].

6.2 The Datasets

ExTree was evaluated using 12 benchmark machine learning datasets, which except for the Grub Damage dataset, are part of the well-known UCI machine-learning repository[32]. The following section gives a brief description of these datasets.

Balance Scale Klahr and Seiger[45] generated this dataset to model the results of a psychological experiment that they carried out. The data models a balance scale with the inputs representing the mass on each side and the distance from the fulcrum. The output class represents whether the balance scale was tipped to the left or right, or balanced.

Sonar This dataset was originally used by Gorman and Sejnowski[29] in their study on using neural networks to classify sonar signals. The dataset was obtained by bouncing a sonar signal off a series of rocks and metal cylinders (mines). The input was the amount of energy in 60 frequency bands integrated over a set interval. The output class indicates whether the object was a rock or a metal cylinder.

Diabetes This dataset is based on data from the National Institute of Diabetes, Digestive and Kidney Diseases. The data describes various medical attributes of patients with the output class representing whether that patient was diabetic. The data in its current form was first used by Smith[81], and is a subset of the original dataset containing only females who are at least 21 years old and of Pima Indian heritage.

Heart Disease Originally collected by Robert Detrano, MD., Ph.D at the V.A. Medical Centre, Long Beach and Cleveland Clinic Foundation. The version used here is the processed form used in the Statlog project[52]. Statlog included only 13 of the original 75 medical attributes as inputs and removed 6 patient records with missing attribute values and an additional 27 records for unknown reasons, leaving a dataset of 270 records. The output class was whether the patient had heart disease.

Hepatitis A medical dataset containing the results of various medical tests and relevant attributes of a patient. The output class being whether they died of the hepatitis infection.

Housing The dataset contains details of houses in the Boston, MA area. Originally the dataset included the house price in dollars and this was used as the output variable in regression studies[8, 66]. Boz[10] modified this dataset to create a classification problem which is used in this evaluation, by discretizing the house price attribute to two classes corresponding to whether the house price was above or below $2,000.

Labor The dataset is based on data from labor negotiations in Canada during 1987-88 for the business and personal service sector. The input attributes describe relevant details of the contracts and the output class is whether the negotiation was successful. Although the positive cases are examples of actual successful contracts, the unsuccessful examples were obtained by interviewing experts or inventing near misses.

Wine The dataset contains the results of a chemical analysis of wines grown in a region of Italy using three different cultivars. Unfortunately the exact meaning of the inputs has been lost to time. The output class is which of the three cultivars the wine was produced from.

Grub Damage Unlike the other datasets that come from the machine learning dataset collection at UCI[32], this dataset was from R. J Townsend at AgResearch. Grass Grub insects are a major cause of pasture damage. The dataset includes measures thought to affect grub population. The output class is whether the grub population was low, average, high or very high.

Zoo Regarded as a relatively simple classification task, the zoo dataset contains the attributes of various animals with the output class being which of 7 groups they belong to. The groups are defined by the creator of the dataset, Richard S. Forsyth

Iris A well-known dataset from the seminal paper by Fisher[24]. The dataset inputs are the features of various individual iris plants and the class attribute is whether it is an example of an Iris Sentosa, Iris Verisilour or Iris Virginica.

Primary Tumor Donated by M. Zwitter and M. Soklic at the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. It was introduced to the machine learning community via Kononenko[14]. The dataset details medical attributes of various patients who had been diagnosed with a tumor. The output class for this dataset is the location of the primary tumor.

A summary of the main features of the twelve datasets is given in Table 6.1. The datasets represent a variety of problem domains and present different challenges to machine-learning algorithms. The machine-learning community has a bias towards using datasets from the medical domain and this is represented by four of the datasets being medical in nature. With the exception of the Balance Scale dataset all the datasets are based on measured or observed data, and are, therefore, likely to contain noise. Four of the datasets only have continuous inputs and four datasets are purely nominal. The remaining eight datasets have a mixture of continuous and nominal inputs. Five of the datasets have more than two classes; the remaining seven datasets have a dichotomous class variable.

Dataset	Instances	Inputs	Continuous	Classes
Balance Scale	625	4	4	3
Sonar	208	60	60	2
Diabetes	768	8	8	2
Heart	270	13	13	2
Hepatitis	155	19	6	2
Housing	506	13	13	2
Labor	57	16	8	2
Wine	178	13	0	3
Grub Damage	155	8	2	4
Zoo	101	16	0	7
Iris	150	4	0	3
Primary Tumor	339	16	0	21

Table 6.1: The characteristics of the 12 datasets used in the evaluation.

6.3 Neural Networks - MLP

To create the neural networks from which to extract rules using ExTree an evaluation was undertaken. The purpose of this evaluation was to find a series of networks that had greater accuracy than C4.5. A basic type of Multilayer perceptron(MLP) was used as described in Chapter 2. For datasets with more than one output class 1-of-N encoding was used. This assigns an output neuron to each class. The classification of an instance is determined by which output node had the highest value. Additionally, the output nodes used the softmax activation function[39] which forces the sum of the output nodes to always equal 1. This allows the output values for each neuron to be interpreted as the probability that the instance belongs to that class. The softmax activation function changes the activation function for an output node i to

$$g(a_i) = \frac{\exp a_i}{\sum_j^K (a_j)}, \tag{6.8}$$

where j runs over the number of the K output neurons for the K classes. For datasets with two classes a single output node is able to adequately represent the dichotomous class attribute. Experiments with different combinations of network topology and learning algorithm parameters were carried out using 10-fold cross validation. The best network configuration for each dataset was then used as the basis for the rule extraction phase. The aim was to find a neural network which outperformed C4.5, for each of the datasets, to make the rule extraction by ExTree worthwhile. For many of the datasets it is probable that a better performing neural network could be found. The accuracy of the neural networks compared to C4.5 are given in Table 6.2. For the best performing neural network on each dataset Table 6.3 gives the learning rate, momentum term, validation set size, maximum number of epochs, number of neurons in the hidden layer and whether a decay term was used.

For purposes of comparison, predictive classification accuracy results were obtained for these datasets using Quinlan's C4.5 algorithm[1]. The C4.5 algorithm used confidence based pruning with a confidence parameter of 0.25. Binary splitting of nominal attributes was not used in these experiments.

6.4 Initial Results

Table 6.4 shows the results of the 10-times repeated 10-fold cross validation. For all the datasets, apart from the Primary-Tumor dataset, the decision tree extracted by ExTree had a higher accuracy than the C4.5 tree grown on the

[1]This was not the 'official' C4.5 released by Quinlan but a C++ work-a-like implementation which shares much of the code base of the ExTree implementation to ensure a fair comparison

Dataset	C4.5	ANN
Balance Scale	77.82	96.00
Sonar	73.61	79.31
Diabetes	73.15	77.61
Heart	77.02	83.46
Hepatitis	77.02	85.82
Housing	83.12	88.93
Labor	80.7	91.57
Wine	89.34	93.65
Grub Damage	39.43	46
Zoo	92.61	94.69
Iris	94.73	95.8
Primary Tumor	41.39	44.28

Table 6.2: Neural network and C4.5 accuracy results.

Dataset	Hidden Layers	Epochs	Learning	Validation	Momentum	Decay
Balance Scale	10,5	2500	0.1	0	0.9	No
Sonar	20	500	0.2	0	0.3	Yes
Diabetes	20	500	0.2	0	0.3	Yes
Heart	20	500	0.2	0	0.3	Yes
Hepatitis	20	500	0.2	0	0.3	Yes
Housing	2	2000	0.2	0	0.3	No
Labor	25	2500	0.1	25	0.9	Yes
Wine	25	2500	0.1	25	0.9	Yes
Grub Damage	25	2500	0.1	25	0.9	Yes
Zoo	10	2000	0.3	0	0.2	No
Iris	10	2000	0.3	0	0.2	No
Primary Tumor	10	2000	0.3	25	0.2	No

Table 6.3: Neural network parameters.

Dataset	C4.5	ExTree (-1)
Balance Scale	77.82	**81.05**
Sonar	73.61	73.69
Diabetes	73.15	**76.03**
Heart Statlog	77.02	**82.74**
Hepatitis	77.02	**82.81**
Housing	83.12	83.96
Labor	80.70	**85.83**
Wine	89.34	**90.78**
Grub Damage	35.96	**42.39**
Zoo	92.61	92.70
Iris	94.73	95.00
Primary Tumor	41.01	40.95

Table 6.4: Comparison of predictive classification accuracy for C4.5 and ExTree.

same data. The largest improvement is on the Grub Damage dataset with an improvement of 6.43%. The Primary-Tumor dataset made a slight decrease in predictive accuracy of 0.06%. A paired t-test showed the difference was statistically significant at $p \leq 0.05$. This is not emboldened in the Table because it was a decrease. A Wilcoxon Rank Sign Test was carried out and again showed the difference between C4.5 and ExTree to be significant ($p = 0.0034$)

Dataset	C4.5	ExTree (-1)
Balance Scale	77.48	**80.84**
Sonar	73.44	**77.28**
Diabetes	83.91	**92.38**
Heart Statlog	85.81	**92.26**
Hepatitis	86.98	**92.82**
Housing	85.22	**89.72**
Labor	81.2	**90.40**
Wine	90.56	**92.98**
Grub Damage	53.14	**79.25**
Zoo	94.75	**95.95**
Iris	97.07	97.00
Primary Tumor	55.89	**70.80**

Table 6.5: Comparison of Fidelity for C4.5 and ExTree.

Table 6.5 shows the fidelity results for ExTree compared with the C4.5 algorithm. C4.5 is not a rule extraction algorithm and therefore, it would be expected that the ExTree algorithm would have significantly better fidelity than C4.5. However, the results are given here to give an indication of how much the fidelity is increased by the extraction process. For example, two algorithms that obtained 100% accuracy would also have 100% fidelity even though there was no extraction link between the algorithms. This can be seen in the Iris dataset which superficially has a high fidelity of 97% but because it is a reasonably easy dataset as reflected in the accuracy results in Table 6.2 this result is in fact the worst result as C4.5 achieved approximately the same result. In terms of gain over C4.5 the largest increase was on the Primary Tumor dataset which increased fidelity by nearly 15% but the overall fidelity was still only 70.8% which means a significant amount of knowledge was not extracted. A Wilcoxon Sign Rank test showed the overall difference in fidelity between C4.5 and ExTree to be significant which shows the extraction process works ($p = 0.00097$).

Dataset	C4.5	ExTree
Balance Scale	77.82	193.4
Sonar	27.9	53.32
Diabetes	43.4	55.2
Heart Statlog	34.64	46.68
Hepatitis	20.12	22.5
Housing	38.04	93.08
Labor	7.92	15.7
Wine	17.35	28.09
Grub Damage	56.73	31.89
Zoo	15.7	36.48
Iris	8.28	22.16
Primary Tumor	89.9	136.19

Table 6.6: Comparision of tree size of C4.5 and ExTree.

Table 6.6 gives the average number of nodes including leaf nodes in the decision tree created for each dataset. Apart from the Grub Damage dataset the trees for the extracted trees are significantly larger than those created by C4.5. It should be noted however that tree node growth is exponential, i.e. a fully grown binary tree of depth 4 has 15 nodes whereas a tree with only one more level has 31 nodes. By comparing this with the accuracy results from Table 6.4 it is seen that increase in size is representative of the increase of information extracted. However, the large increase in size and therefore the

Dataset	No Relabel	Relabel
Balance Scale	77.82	77.76
Sonar	73.61	74.99
Diabetes	73.15	75.48
Heart Statlog	77.02	**82.00**
Hepatitis	77.02	**81.94**
Housing	83.12	**84.28**
Labor	80.70	82.00
Wine	89.34	**90.51**
Grub Damage	35.96	**42.43**
Zoo	92.61	92.30
Iris	94.73	95.00
Primary Tumor	41.01	39.58

Table 6.7: Effect of relabelling on accuracy.

reduced overall comprehensibility of the model may not be worth the modest improvement in accuracy. Of course, the cost–benefit of the comprehensibility–accuracy trade-off is problem-domain specific.

6.5 Effect of Relabelling

This section will examine how relabelling the dataset affects the accuracy of the decision tree as discussed in Chapter 5. As discussed in the previous chapter the purpose of relabelling is to take advantage of the neural network's ability to deal with noise in the dataset. A t-test showed the difference between a decision tree grown on only the original dataset and the decision tree grown on the relabelled dataset was significant on five of the twelve datasets, suggesting these are the noisiest of the datasets, or more precisely the datasets with most noise that the neural network could 'filter out'. Although the *t*-test did not show the difference between the accuracy for the Primary Tumor and Zoo datasets

Dataset	No Relabel	Relabel
Balance Scale	77.82	81.32
Sonar	27.9	**24.36**
Diabetes	43.4	45.94
Heart Statlog	34.64	**26.14**
Hepatitis	20.12	**10.08**
Housing	38.04	**35.46**
Labor	7.92	7.75
Wine	17.35	17.08
Grub Damage	56.73	**32.93**
Zoo	15.7	15.7
Iris	8.28	**7.76**
Primary Tumor	89.9	**84.02**

Table 6.8: Effect of relabelling on tree size

to be significant, accuracy did drop slightly after relabelling. An explanation could be that the neural network was 'smoothing' too much. The advantage of relabelling the data is that the neural network removes some of the noise in the dataset (smooths) but, potentially, if the neural network smooths the data too much the decision tree part of the algorithm could underfit the data.

Relabelling had a significant effect on tree size. In six of the eleven datasets it reduced the size of the tree significantly. This is likely to be because the neural network's ability to 'see through' the noise resulted in a decision tree that did not overfit the data. The Balance Scale dataset, known to be noise free, increased in size, however, from table 6.2 it is known that the neural network achieved only 96% accuracy so relabelling the dataset will have introduced noise to the dataset resulting in an increase in decision tree size.

Table 6.9 shows the effect of relabelling on the fidelity between the neural network and the extracted decision tree. As predicted, all except one of the

Dataset	No Relabel	Relabel
Balance Scale	77.48	77.37
Sonar	73.44	**77.9**
Diabetes	83.91	**91.94**
Heart Statlog	85.81	**90.93**
Hepatitis	86.98	**92.65**
Housing	85.22	**89.96**
Labor	81.2	**84.37**
Wine	90.56	**91.4**
Grub Damage	53.14	**77.74**
Zoo	94.75	**96.06**
Iris	97.07	96.87
Primary Tumor	55.89	**68.08**

Table 6.9: Effect of relabelling on fidelity

datasets showed an increase in fidelity after relabelling. Using a series of t-tests it was found that 10 of the datasets had significant increases in fidelity. The grub damage dataset showed the largest increase in fidelity, from relabelling alone, increasing from 53.14% to 77.4%. This could have been due to the suspected large amount of noise in this dataset being smoothed out by the neural network. The noise-free Balance Scale dataset showed no improvement in fidelity with a statistically insignificant drop of 0.11%. Although there is no noise to remove it was expected that the fidelity would have increased by relabelling because the tree would be modelling the function of the neural network and not the original dataset. The reason this did not occur could be because there are insufficient instances in the dataset for a decision tree to model, neither the generating function of the original dataset nor the function learnt by the neural network, close enough to achieve an accuracy above approximately 77%. This reasoning is supported by the results in the next section on generating new instances.

Dataset	0	1	2	4
Balance Scale	77.76	**81.05**	**83.18**	**87.39**
Sonar	74.99	73.69	71.68	72.47
Diabetes	75.48	76.03	**76.05**	**76.13**
Heart Statlog	**82.00**	**82.74**	81.59	**81.78**
Hepatitis	**81.94**	**82.81**	81.46	82.03
Housing	**84.28**	**83.96**	**84.80**	84.54
Labor	82.00	**85.83**	83.57	86.67
Wine	**90.51**	**90.78**	89.28	**89.49**
Grub Damage	**42.43**	**42.39**	**42.74**	**43.53**
Zoo	92.30	92.7	92.70	91.8
Iris	95.00	95	95.00	95.07
Primary Tumor	39.58	40.95	41.54	40.8

Table 6.10: The effect of new instances on accuracy.

6.6 New Instances

This section examines the effect of adding new instances to the dataset. For each of the twelve datasets, a decision tree was extracted from the neural network with 1, 2, and 4 times the number of instances that were in the original dataset. Table 6.10 shows the results of this experiment. When the number of new instances was set to double the number in the original dataset, all eleven datasets showed an improvement in accuracy over both C4.5 and the decision tree extracted from the relabelled data. A t-test showed the improvement over C4.5 to be statistically significant for 8 of the datasets. As the number of datasets was increased to 2 and 4 times the original number the accuracy started to fall for about half the datasets.

Table 6.11 shows that increasing the number of new instances increased the fidelity for 8 datasets. However somewhat counter-intuitively, the fidelity dropped for some of the datasets as the number of new instances increased.

Dataset	0	1	2	4
Balance Scale	77.37	80.84	83.33	88.12
Sonar	77.9	77.28	75.55	76.53
Diabetes	91.94	92.38	92.77	93.16
Heart Statlog	90.93	92.26	92.77	92.93
Hepatitis	92.65	92.82	91.59	92.85
Housing	89.96	89.72	89.56	90.14
Labor	84.37	90.4	89.27	92.23
Wine	91.4	92.98	91.07	92.64
Grub Damage	77.74	79.25	78.08	80.55
Zoo	96.06	95.95	95.85	94.93
Iris	96.87	97	97.33	97.6
Primary Tumor	68.08	70.8	74.81	70.95

Table 6.11: The effect of new instances on fidelity.

The likely explanation for this comes from the way fidelity is measured. Because fidelity is measured against the original dataset even if the fidelity of the decision tree is increased in $\mathscr{X}$ overall, if the dataset has strong interlinked dependencies between attributes, the fidelity in the area of $\mathscr{X}$ covered by the original dataset may be reduced. That is, the fidelity within an area of the input space that is of little interest may increase at the cost of reducing the fidelity within an area of greater interest.

Previously it was speculated that there was not enough instances in the dataset for a decision tree to model the Balance Scale dataset above about 77% and the results here are consistent with that prediction. Increasing the number of new instances increased the fidelity of the Balance Scale datasets smoothly from 77.37% to 88.12% with accuracy also improving from 77.76% to 87.38%.

6.7 Ensembles

This section shows the general applicability of the ExTree technique by applying it to an ensemble of neural networks. Ensemble methods, also called committee methods, combine the results of multiple classifiers to classify an instance. Although there are several methods to create ensembles such as Stacking[97], Boosting[73], and Bagging[11]. The results presented in this section were obtained using Bagging, which is the simplest of these methods. The often-used analogy for ensemble methods is that a decision reached by a group of experts is often better than a decision made by a single expert. The Bagging method gives equal weighting to all the individual classifiers. Following the 'group of experts' analogy, this is the same as considering all experts to have equal expertise for all problems, which is rarely the case. However, generally bagged ensembles perform as well or better than a single classifier.

In the simplest case of a group of g neural networks trained on the same problem, assuming the neural networks started with the same weights, all the individual networks would be identical and there would be no benefit of combining the results of the individual neural networks. Initialising the neural networks with different weights does introduce some variation but training still tends to converge the neural networks to sets of weights which produce similar final classifications. This is normally a desirable property but, in this case, means the ensemble would be no more powerful as a classifier than an individual neural network.

The bagging method provides a different training set for each classifier in the ensemble which produces a group of quite different neural networks. The bagging method applied to g classifiers with a training dataset containing n instances, creates g new datasets with n instances in each by re-sampling with replacement from the original dataset. This has the effect that some instances will not appear in all of the g new datasets and some instances will be repeated multiple times within the same dataset. This produces an ensemble of classifiers that each have expertise in a slightly different area of the pattern space, $\mathscr{X}$.

Ensembles which combine the predictions of multiple classifiers are more opaque than a single classifier. To demonstrate the flexibility of the ExTree algorithm, it was used to extract a decision tree from an ensemble of neural networks trained on each of the twelve datasets. For each of the datasets an ensemble of ten neural networks using bagging was created. The ExTree algorithm, relabelled the instances and doubled the number of instances.

Dataset	C4.5	ExTree(Ensemble)
Balance Scale	77.82	**81.42**
Sonar	73.61	**71.34**
Diabetes	73.15	**75.82**
Heart Statlog	77.02	**81.15**
Hepatitis	77.02	**82.68**
Housing	83.12	**84.46**
Labor	80.70	**83.03**
Wine	89.34	90.63
Grub Damage	35.96	**39.46**
Zoo	92.61	92.42
Iris	94.73	95.07
Primary Tumor	41.01	**42.52**

Table 6.12: Accuracy for decision tree extracted from ensembles using exTree compared with C4.5.

Table 6.12 shows that as on a single neural network the ExTree algorithm extracted a decision tree that had higher accuracy than a decision tree induced using C4.5. The *t*-tests showed that for 9 of the datasets the difference was significant. Applying the more sophisticated Boosting algorithm would have improved the accuracy results of the ensemble itself which would have led to better accuracy results for the extracted decision tree. The results prove that the ExTree algorithm is applicable to ensembles. As expected the Wilcoxon

Dataset	C4.5	ExTree(Ensemble)
Balance Scale	77.48	**81.42**
Sonar	73.44	**76.45**
Diabetes	83.91	**92.2**
Heart Statlog	85.81	**91.59**
Hepatitis	86.98	**92.51**
Housing	85.22	**90.37**
Labor	81.2	**88.5**
Wine	90.56	**92.77**
Grub Damage	53.14	**62.92**
Zoo	94.75	94.88
Iris	97.07	97.53
Primary Tumor	55.89	**71.95**

Table 6.13: Fidelity for decision tree extracted from ensembles using ExTree compared with C4.5.

Sign Rank test confirmed the algorithms to be significantly different.

Table 6.13 shows the fidelity of the extracted trees with the ensemble. The C4.5 fidelity results are provided for comparison. The t-tests showed the fidelity was significantly higher for 10 of the datasets. The increase in fidelity will, in part, be due to the increase in accuracy of the ensemble. The fidelity results further demonstrate the flexibility of the ExTree algorithm to be applied to other classifier types.

6.8 ExLMT

In this section the ExLMT algorithm, introduced in Section 5.3, which produces the Logistic Model Trees instead of the traditional C4.5-style trees is evaluated.

Dataset	C4.5	ExTree	ExLMT
Balance Scale	77.82	81.05	93.33*
Sonar	73.61	73.69	77.99*
Diabetes	73.15	76.03	77.1 *
Heart Statlog	77.02	82.74	83.44*
Hepatitis	77.02	82.81	85.06*
Housing	83.12	83.96	85.57*
Labor	80.70	85.83	90.67*
Wine	89.34	90.78	92.32*
Grub Damage	35.96	42.39	44.03*
Zoo	92.61	92.70	93.7 *
Iris	94.73	95.00	95.8 *
Primary Tumor	41.01	40.95	42.48*

Table 6.14: Comparison of predictive classification accuracy for C4.5 and ExTree and ExLMT.

Table 6.14 compares the accuracy of C4.5, ExTree and ExLMT. ExTree and ExLMT both used the neural network to relabel the dataset and doubled the size of the dataset by creating new instances. The decision trees extracted using ExLMT had significantly better accuracy than C4.5 and ExTree for all eleven datasets.

Dataset	C4.5	ExTree	ExLMT
Balance Scale	77.37	80.84	93.94
Sonar	77.9	77.28	84.53
Diabetes	91.94	92.38	98.28
Heart Statlog	90.93	92.26	98.44
Hepatitis	92.65	92.82	96.29
Housing	89.96	89.72	93.04
Labor	84.37	90.4	96.63
Wine	91.4	92.98	96.2
Grub Damage	77.74	79.25	88.18
Zoo	96.06	95.95	93.7
Iris	96.87	97	95.8
Primary Tumor	68.08	70.8	84.22

Table 6.15: Comparison of fidelity of C4.5, ExTree, ExLMT.

Table 6.15 compares the fidelity of C4.5, ExTree and ExLMT. Again, ExLMT had a significantly higher fidelity with the neural network than either C4.5 or ExTree.

Table 6.16 compares the resulting size of the trees from C4.5, ExTree and ExLMT. The ExLMT grown trees have substantially fewer nodes than the C4.5 and ExTree decision trees, and although smaller trees are easier to comprehend, the logistic models at the leaf nodes are far less comprehensible than the simple class labels of the C4.5 and ExTree leaf nodes. Wilcoxon Rank Sign tests showed the differences in both fidelity and accuracy to be significant when ExLMT was

Dataset	C4.5	ExTree	ExLMT
Balance Scale	77.82	193.4	18
Sonar	27.9	53.32	7.18
Diabetes	43.4	55.2	3.92
Heart Statlog	34.64	46.68	4.52
Hepatitis	20.12	22.5	2.18
Housing	38.04	93.08	35.76
Labor	7.92	15.7	1.81
Wine	17.35	28.09	3.37
Grub Damage	56.73	31.89	27.37
Zoo	15.7	36.48	1.68
Iris	8.28	22.16	5.56
Primary Tumor	89.9	136.19	21.24

Table 6.16: Comparison of tree size for C4.5, ExTree, ExLMT.

compared with both C4.5 and ExTree. Overall the results showed, as predicted, that ExLMT increases the accuracy and fidelity significantly and also reduces tree size, but at the cost of introducing more complicated leaf nodes.

6.9 Summary

In this chapter ExTree was evaluated on twelve real-world datasets. To ensure the results and the conclusions drawn from them were valid, a comprehensive evaluation methodology was developed. This evaluation methodology used repeated k-fold cross validation, t-tests and Wilcoxon hypothesis testing.

The results clearly showed the feasibility of the ExTree rule extraction method by successfully extracting a decision tree from each of the neural networks tested. Comparing ExTree with C4.5 showed ExTree produced trees with higher predictive accuracy, with ExTree achieving higher predictive accuracy on eleven of the twelve datasets and a statistically significant improvement on 7 of the datasets. In addition to predictive accuracy, the fidelity of ExTree to the original neural network was evaluated, and, as expected, ExTree had much higher fidelity with the neural network than C4.5. This confirmed that the decision tree extracted by ExTree was modelling the knowledge within the neural network and not just producing decision trees with better predictive accuracy.

To better understand ExTree performance on real-world data, experiments testing the relabelling and instance generation components of the ExTree algorithm were conducted. These experiments showed that relabelling contributed to the increase in predictive accuracy and reduced tree size for many of the datasets. The experiments also showed that increasing the number of instances created in the instance generation phase of ExTree substantially increased fidelity. To demonstrate the ability of ExTree to be applied to other machine learning algorithms that produce oblique classifiers, ExTree was successfully applied to a bagged ensemble of neural networks.

Finally, ExLMT, a modified version of ExTree that replaces the leaf nodes with logistic models was evaluated. The results showed that this modification increased accuracy, fidelity and reduced tree size at the cost of more complex leaf nodes.

CHAPTER 7

ExMT: Extraction in Regression Domains

In the previous two chapters, ExTree, a novel algorithm for extraction of decision trees from neural networks was developed and evaluated. ExTree is only applicable to neural networks trained on classification problems. Previous research into rule extraction techniques has been predominately concerned with neural networks trained on classification problems. However, neural networks are universal approximators and make powerful models for regression based problems. ExMT is a new rule extraction algorithm that extracts a model tree[64] from neural networks trained on regression problems.

The next section examines previous work in rule extraction in regressions domains. Section 7.2 reviews the model tree representation, which is the basis of the new rule extraction algorithm. Section 7.3 presents the new ExMT rule extraction algorithm itself. Section 7.4 presents and discusses the results of an empirical evaluation of ExMT on real-world datasets. Section 7.4.4 provides an empirical comparison of ExMT and ANN-DT[75], a previous rule extraction algorithm. Finally, the last Section provides a summary of the chapter.

7.1 Previous Techniques

Rule extraction from neural networks trained for regression problems is a neglected area in the rule extraction literature[86]. This section briefly reviews two algorithms developed for regression domains. Setiono suggest a decompositional approach called REFANN[76] that extracts IF..THEN rules from the neurons in the hidden layer. This is analogous to the SUBSET style methods[87] used for classification based rule extraction. The IF *region* THEN *linear function of*

x rules are extracted in two parts. The *linear* function is a three-piece linear equation that approximates the activation function of each hidden neuron as illustrated in Figure 7.1.

This linear approximation to the sinusoid introduces error as shown by the grey shading. The *region* is the hyperplane defined by the product of the inputs and weights. To aid comprehension a further stage can be applied that replaces the oblique hyperplanes used in the conditions with axis-parallel hyperplanes, which are found using C4.5. This is achieved by creating a classification dataset from the original regression training set by replacing the continuous output value with a region number, then training the C4.5 algorithm using this dataset. IF..THEN rule extraction is applied to the C4.5 tree and finally the regions are replaced with the corresponding 3-piece approximation extracted from the neuron for that region.

Being a decompositional approach REFANN has the associated benefits and drawbacks[4]. Empirical evaluations show the approach to have a high degree of accuracy and exceptional fidelity. The drawbacks are the constraints on the neural network architecture. For example, because the number of rules is 3 times the number of hidden neurons, the neural network model needs to be pruned to reduce the number of neurons, and therefore the number of rules extracted. Not only does this pruning of the neurons mean that special training algorithms are required it is also likely to reduce the classification accuracy of the neural network.

The pedagogical extraction technique ANN-DT[74] was extended to regression problems[75]. This algorithm also uses the sampling and querying approach[16]. New query instances are created which are 'near' existing instances in the instance space as defined by either Euclidean distance or a distance metric proposed by Gower[30], suitable for mixed continuous and discrete attributes. The tree induction aspects of ANN-DT(e) are based on CART[12] and therefore uses weighted variance as the split minimisation criteria. The

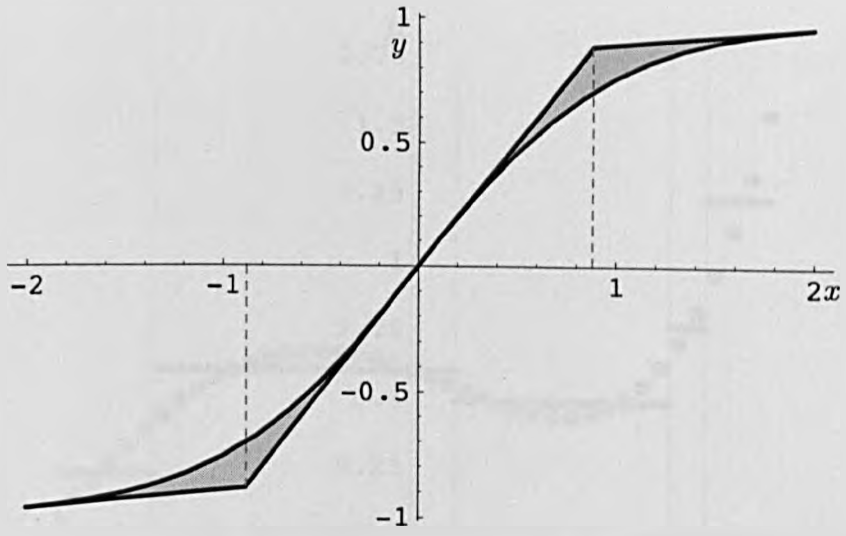


Figure 7.1: Three piece approximation to tanh.

weighted variance of split of the data S into n subsets can be calculated as

$$V = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \mathrm{Var}(S_i), \tag{7.1}$$

where $\mathrm{Var}(S_i)$ is the variance of output attribute of the i^{th} subset of S. A second version of the algorithm ANN-DT(s) uses a form of significance analysis on the neural network to determine the most significant attribute. The split point on the attribute found to be most significant is determined using the weighted variance measure(Eq. (7.1).

The regression tree formed by ANN-DT has the same form as CART's regression trees with leaf nodes being assigned the mean value of the instances that reach that node, so ANN-DT gives a piecewise *constant* approximation of the neural network. Figure 7.2 shows a typical piecewise constant approximation to a curve. The datapoints shown clearly follow a curve, but the only way a piecewise constant classifier, such as CART or ANN-DT, can model this is through splitting the curve into sections and predicting the mean value for each section. This produces a step approximation to the curve. This will inherently create errors which are equal to the distance between the datapoints and the mean line for each section.

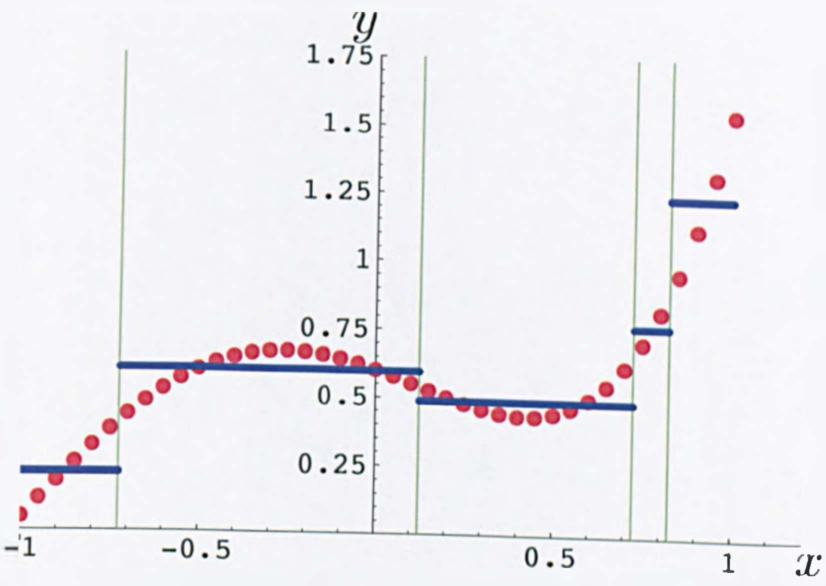


Figure 7.2: Piecewise constant approximation to a nonlinear function.

7.2 Model Trees and M5 Induction

Decision Trees induction techniques such as ID3/C4.5[65], and CHAID[43] have proved popular in pattern classification domains. The main advantage is that the inherent graphical representation is easy to comprehend. However, these trees are only capable of predicting categorical values and cannot, therefore, be applied to regression problems which require the prediction of a continuous output variable.

Model Trees[64, 91] attempt to take the advantages that decision trees have in the classification domain and transfer them to the regression domain. Previously, decision tree algorithms that have been applied to regression problems required the class variable to be discretized which meant only average values could be predicted. This is shown in Figure 7.2, which shows an approximation to a continuous value through discretizing the value. The curve to be approximated is split into sections and for each section the mean value of the datapoints in that section is used as the predicted value. Unless the function being approximated is a step function this will result in an error. This error will increase the further the function being approximated diverges from being an axis-parallel step function. To reduce this error the function can be split into more sections, but this will increase the size of the decision tree and therefore reduce comprehensibility. The most significant improvement model trees provide is that they can predict a continuous class value directly, so do not need to discretize the target variable.

The M5 model tree induction algorithm follows the same top down recursive partitioning approach as classification decision tree algorithms such as C4.5, CART and CHAID. Standard deviation of the numeric class value is used to determine which attribute to split on[64]. Discretization is not necessary because the leaf nodes are replaced with a multivariate linear model predicting a continuous class value. Prediction is then accomplished in two stages. First, the tree structure defines in which area of instance space the predictive value lies. Second, the linear model for that region then predicts the final class value. The Model Tree is, therefore, a piecewise *linear* representation as shown in Figure 7.3.

7.3 ExMT

The previous two sections reviewed previous rule extraction techniques and provided a summary of the model tree and the M5 algorithm for inducing model trees. In this section two new algorithms, ExMT(a) and ExMT(b) for rule extraction in regression domains are developed. The difference between the two versions is the measure used for splitting. ExMT(a) like M5 uses standard deviation and ExMT(b) uses a measure based on Root Mean Squared Error(RMSE). Using standard deviation has the advantage that it is quick to calculate, but has a significant weakness when used for the induction of model trees. The next section will examine this weakness before demonstrating that RMSE provides a more accurate measure but at the cost of increasing computational time. The way ExMT handles nominal values will then be discussed. The section ends with a step-by-step description of the algorithm.

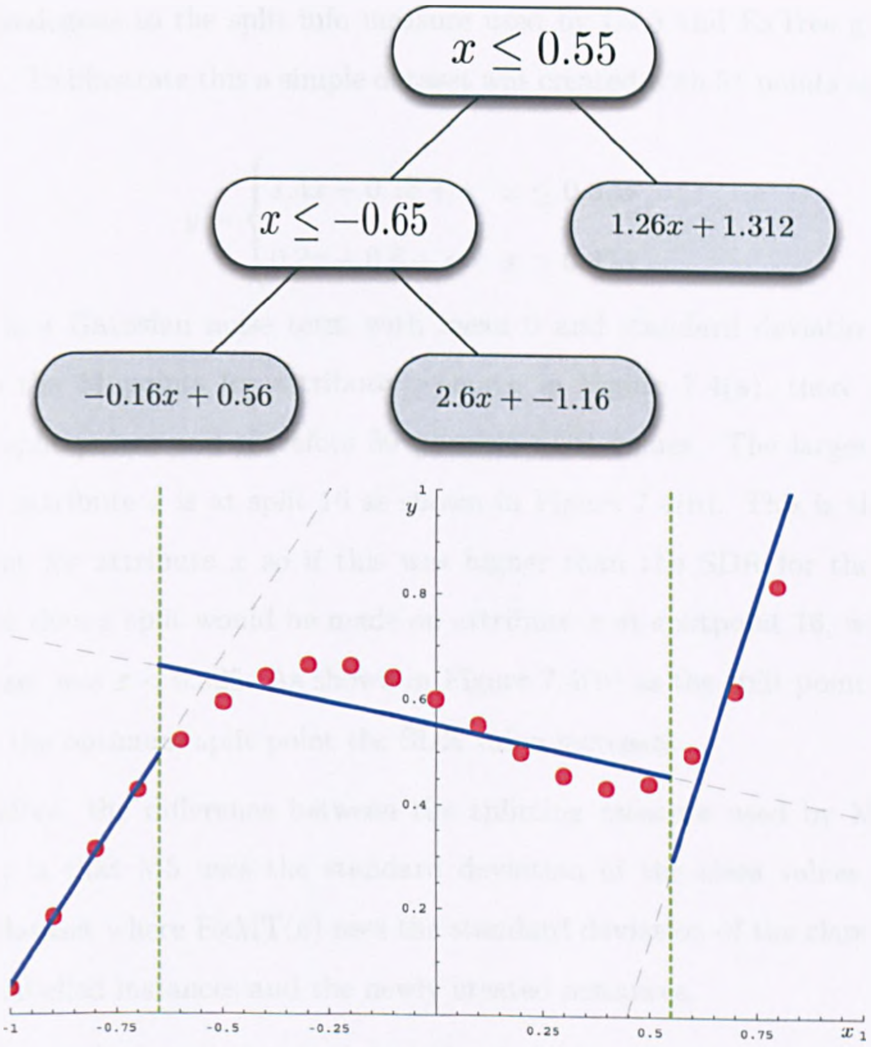


Figure 7.3: An example M5 approximation to a curve.

7.3.1 Split Selection

ExMT(a) uses the same standard deviation based measure as M5 to determine the correct attribute and point to split on. All possible split points are considered so for an attribute with m distinct values in the training set it will consider all $m - 1$ possible split points. The best split is the one which most reduces the standard deviation of the class values. The reduction in standard deviation by splitting a set S into N subsets is given by

$$\text{sdr} = \text{StdDev}(S) - \sum_{i=1}^{N} \frac{|S_i|}{|S|} \text{StdDev}(S_i). \tag{7.2}$$

This is analogous to the split info measure used by C4.5 and ExTree given in Eq. (3.5). To illustrate this a simple dataset was created with 51 points sampled from

$$y = \begin{cases} 1.4x + 0.15 + \epsilon & x \leq 0.325 \\ 0.2x + 0.6 + \epsilon & x > 0.325\ . \end{cases} \tag{7.3}$$

where ϵ is a Gaussian noise term with mean 0 and standard deviation 0.05. Consider the 51 points for attribute x shown in Figure 7.4(a), there are 50 possible split points and therefore 50 possible SDR values. The largest SDR value for attribute x is at split 16 as shown in Figure 7.4(b). This is the best split point for attribute x so if this was higher than the SDR for the other attributes then a split would be made on attribute x at splitpoint 16, which in this dataset was $x < 0.325$. As shown in Figure 7.4(b) as the split point moves closer to the optimum split point the SDR value increases.

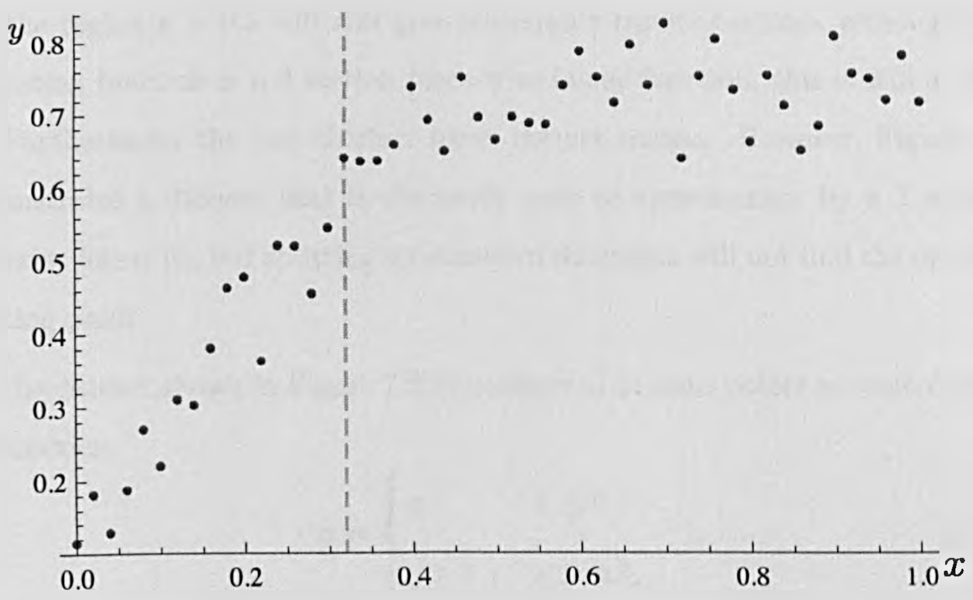
Therefore, the difference between the splitting measure used by M5 and ExMT(a) is that M5 uses the standard deviation of the class values in the original dataset where ExMT(a) uses the standard deviation of the class values of the relabelled instances and the newly created instances.

However, splitting on standard deviation is best applied to situations when the points are clustered around distinct mean values. Figure 7.5(a) demonstrates a situation where the points are clustered around 2 values (0.25 and 0.75) and splitting based on reducing variance works well. A classifier which is limited to predicting mean values(CART,ANN-DT) will create a tree predicting a class value of 0.25 for $x \leq 0.5$ and 0.75 for $x > 0.5$. In this case standard deviation reduction works well and finds the optimal classifier.

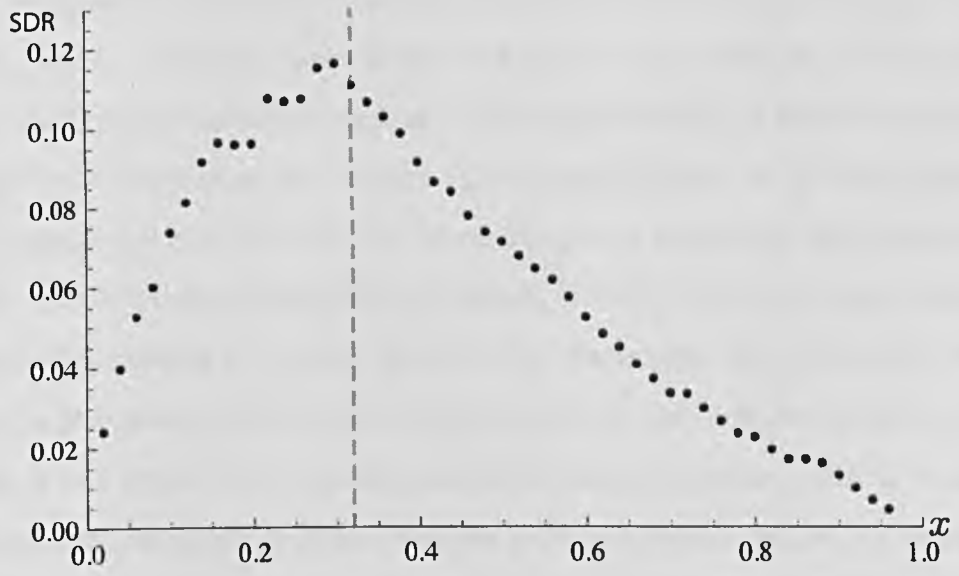
In Figure 7.5(b) the points are generated from the function

$$y = \begin{cases} 0.2x + 0.25 & x \leq 0.5 \\ -0.3x + 0.9 & x > 0.5, \end{cases} \tag{7.4}$$

with a small amount of Gaussian noise added. The optimal classifier is obviously Eq (7.4) but the optimal split point $x = 0.5$ can still be easily found using



(a) Data Points



(b) sdr values

Figure 7.4: Splitting based on standard deviation ratio. Splitting at the peak of the SDR ratio splits x into two linear segments at the optimal point in (a).

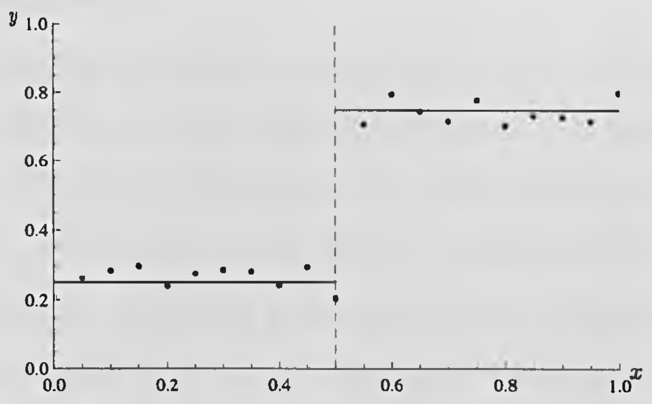
standard deviation. A classifier predicting mean values for the region $x \leq 0.5$ and the region $x > 0.5$ will still give reasonable results because, although the generating function is a 2 section piece-wise linear function, this is still a close fit. Furthermore, the two clusters have distinct means. However, Figure 7.5 demonstrates a dataset that is obviously easy to approximate by a 2 section piecewise linear fit, but splitting on standard deviation will not find the optimal splitting point.

The dataset shown in Figure 7.5(c) consists of 21 data points generated from the function,

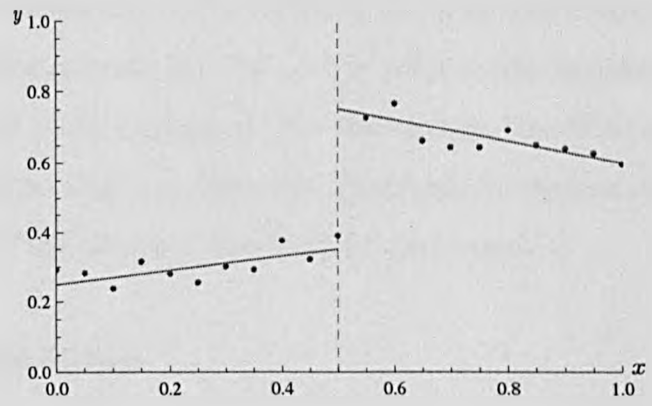
$$y = \begin{cases} x & x \leq 0.5 \\ -x+1 & x > 0.5, \end{cases} \tag{7.5}$$

The dataset has Standard Deviation of 0.156. The optimal splitting point for a 2 piece piecewise linear fit is $x \leq 0.5$. This creates two subsets which also have a Standard Deviation of approximately 0.156 resulting in this split not being chosen. The leaf node would then predict the mean value of 0.24 for piece-wise constant methods such as CART and ANN-DT. A Model Tree based method will attempt to fit a single linear regression line to all the points in the range $0 \leq x \leq 1$, but the best linear fit that is possible is axis-parallel so, again, it is the same as predicting the mean of 0.24. In the previous examples Standard Deviation is a good measure for determining the split point when fitting a piecewise constant model, but when the model being fitted is piecewise linear it can prove to be a poor measure in many situations. This is because the standard deviation reduction chooses split points that reduce the spread of the data points, which is only optimal when predicting the mean of the values. If a linear model is to be fitted then the best measure for determining the split point is to reduce the error between the points and the best regression line through those points. The error can be measured using Root Mean Squared Error, RMSE(Eq. (7.10)).

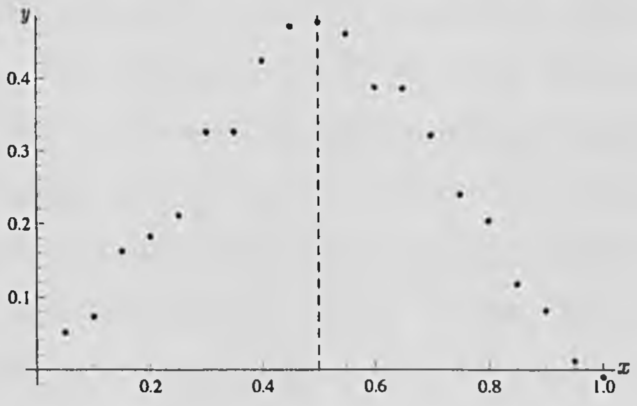
Figure 7.6(a) shows a split point of 0.2; the linear regression to the left of the split point has no error, but the linear regression to the right of the split



(a) Points clustered around a Mean



(b) Points clustered but not axis parallel



(c) Points linear but not clustered

Figure 7.5: Splitting based on standard deviation.

point has substantial error.

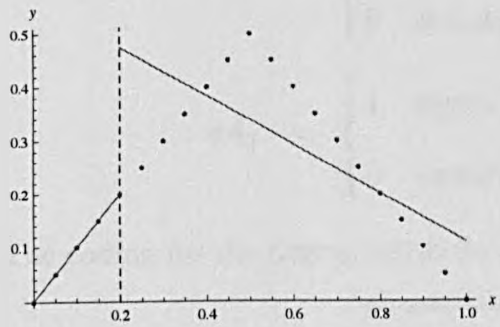
Figure 7.6(b) shows how moving the split point to $x = 0.4$, which is closer to the optimal split point, substantially reduces the error in the right-hand-side regression line. Moving the split point to the optimal split point of $x = 0.5$, as shown in Figure 7.6(c), reduces the error to 0 for both regression lines.

Figure 7.6(d) shows that as the split point moves to the right of the optimal split point the error of the left-hand-side regression line increases. If the criteria for splitting is based on reducing the RMSE of the regression line then the optimal $x = 0.5$ split point would be selected.

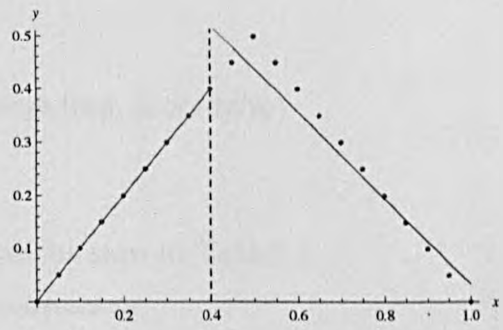
Basing the split criteria on the RMSE of the linear regression fit does come at substantial computational cost. For each possible splitting point it requires two linear regressions to be performed. For this reason, two versions of ExMT are evaluated ExMT(a) that uses Standard Deviation for determining the splitting point, and ExMT(b) that uses the RMSE based measure.

7.3.2 Nominal Values

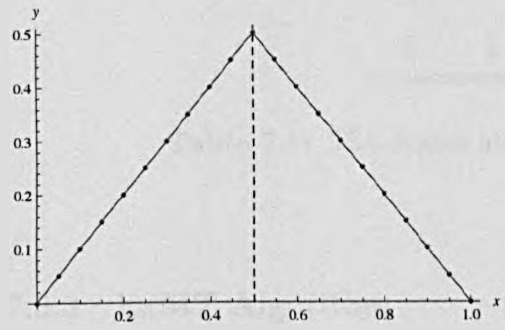
The linear models used by the model tree require the nominal values to be converted to numeric values. One proposed solution[91] is to apply the method used by CART[12] to model trees. Consider an attribute, A, with m possible values $A_1, \ldots, A_m$. The average class value for each of the m values is calculated and the m values are then ordered by this average class value. This tends to produce an ordering of the m values that is positively correlated with the class value. CART recalculates this ordering at every node of the tree based on the instances that have reached that node. In contrast, the M5 algorithm [91] makes use of the same ordering for the whole tree. Because the ordering is fixed for the whole tree, it is possible to improve the efficiency of the algorithm by converting the attribute A at the Root node into $m - 1$ synthetic binary attributes. The $m - 1$ synthetic binary attributes correspond to whether the original attribute was a member of the ith ordered attribute values. For ex-



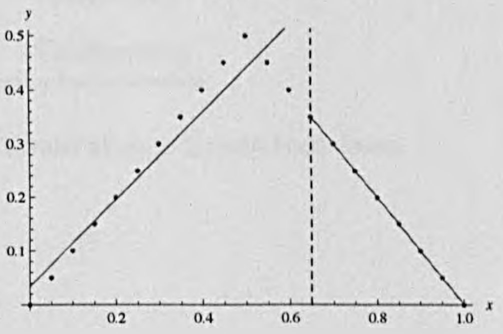
(a) Split Point, $x \leq 0.2$



(b) Split Point, $x \leq 0.4$



(c) Split Point, $x \leq 0.5$



(d) Split Point, $x \leq 0.65$

Figure 7.6: Splitting on RMSE between linear regression and data points.

ample, consider the attribute *degree* $\in$ {geography, computing, drama} and the class attribute *mark* $\in \{1, \ldots 100\}$. If the average class mark for geography, computing, and drama were 80, 65, 90 respectively then the ordering would be computing = 0, geography = 1, and drama = 2. The synthetic binary attributes would then be

$$sA_1 = \begin{cases} 1 & degree \in \{computing\} \\ 0 & elsewhere. \end{cases} \tag{7.6}$$

$$sA_2 = \begin{cases} 1 & degree \in \{computing, geography\} \\ 0 & elsewhere. \end{cases} \tag{7.7}$$

The coding for the degree attribute can then be seen in Table7.1.

sA1	sA2	Degree
0	0	Drama
0	1	Geography
1	1	Computing

Table 7.1: The degree attribute after a binary transform.

7.3.3 ExMT Algorithm

The algorithm starts with a neural network and the dataset used to train it. The dataset is then relabelled by replacing the output attribute for each instance in the dataset with the output value of the neural network for that instance. New instances are then created by modelling the dataset and sampling from it. These new instances are then labelled using the neural network in its role as an oracle. The new instances are then added to the original dataset. A Model Tree is then induced from the new enlarged dataset in a similar way to the M5 algorithm. ExMT(a) uses Standard Deviation as the split criteria, like M5, but the ExMT(b) uses a RMSE based split measure.

A step-by-step description of the algorithm is now presented.

Stage 1: Obtain the trained neural network ExMT places no constraints on the topology or training algorithm of the neural network

Stage 2: Relabel the dataset The dataset used to initially train the neural network is relabelled by the fully trained neural network. This aims to reduce noise in the class attribute of the dataset.

Stage 3: Generate new data instances To elicit the knowledge stored in the neural network, a number of new instances are created and then labelled by the neural network. The new instances are created based on the original dataset. Nominal attributes are sampled according to their frequency in the original dataset. For example, in a dataset with 20 instances, with a colour attribute with 10 instances being Red, 5 Green and 5 Blue, the colour attribute of the new instances would have the same probabilities, Red(0.5), Green(0.25), Blue(0.25). For continuous attributes a Kernel Density estimate[79] is made of the distribution. Random values are then sampled from this distribution, and the new instances are then labelled by the neural network.

Stage 4 - 6 Create Model Tree

Stage 4: Create Candidate Splits ExMT only considers binary splits for both continuous and nominal attributes. For continuous attributes the values in the dataset are sorted with each midpoint between the values being considered as a splitting point.

Stage 5: Select Best Split To measure the benefit of each possible split the reduction in the standard deviation(StdDev) of the class value after the split is used. For a Split T that splits the dataset S into subsets S_l and S_r, the measure used is

$$i(T,S) = \text{StdDev}(S) - \left(\frac{|S_l|}{|S|}\text{StdDev}(S_l) + \frac{|S_r|}{|S|}\text{StdDev}(S_r)\right) \quad (7.8)$$

for the method ExMT(a). For method ExMT(b) the following measure is used

$$i(T,S) = \mathrm{MSD}(S) - \left(\frac{|S_l|}{|S|}\mathrm{MSD}(S_l) + \frac{|S_r|}{|S|}\mathrm{MSD}(S_r),\right) \qquad (7.9)$$

where MSD(S) is the mean squared difference between the predictions from the neural network and a linear regression model for set S.

Stage 6: Create the Linear models for leaf nodes A linear model for each node in the tree is created.

Stage 7: Compare the Linear Model The ideal way of pruning a tree is to compare the error for each interior node linear model to the subtree below it. If the error is lower for the interior node the subtree below is removed and the interior node becomes a leaf node. However, in real world problems the expected error is, of course, unknown so it is approximated. A subset of the training data can be set aside and used as a pruning set, but this reduces the amount of data that can be used to induce the model. The residual error of a model is the difference between the predicted values and the actual values. When pruning decision trees, such as C4.5 the residual error cannot be reduced at the pruning stage because the growing of the tree has already reduced the residual. In contrast, model trees reduce another measure of error such as standard deviation at the tree growing stage so can reduce the residual error at pruning time. A parsimonious multiplicative factor of $(n+v)/(n-v)$ is used, where n is the number of training instances and v is the number of parameters in the model. This has the effect of favouring smaller, simpler models at the cost of accuracy.

7.4 Evaluation

The evaluation of the ExMT algorithms follows the same general approach as used in evaluating ExTree in Chapter 6. Again, stratified k-fold validation is used with a k value of 10 chosen, as is standard in machine learning. The k-fold evaluation itself was repeated 10 times to produce reliable, unbiased estimates of predictive accuracy and fidelity and tree size. This results in the classifier being executed 100 times for each estimate of classification accuracy or fidelity. To compare two classifiers on the same dataset a standard two-tailed t-test was used. In the tables, bold indicates a result was a significant improvement over M5($p < 0.05$). To compare the performance of two classifiers over a series of datasets the Wilcoxon signed rank test is used. However, the measures used in classification domains for predictive classification accuracy (6.1) and fidelity (6.3) have to be modified for regression domains.

Two measures were used to assess the predictive accuracy of the methods. The basic measure is mean squared error, which is the average of the squares of the differences between the predicted value and the actual values. Normally the square root of MSE is used so that the measure is comparable to the dimensions of the values being predicted giving root mean squared error (RMSE). This can be calculated as

$$\text{RMSE} = \sqrt{\frac{1}{M}\sum_{i}^{M}(t_i - y_i)^2}, \qquad (7.10)$$

where t_i and y_i are the actual and predicted values, respectively, for instance i of the M instances.

A variation of RMSE is root relative mean squared error(RRMSE). This measures the performance of the predictor relative to merely predicting the mean value, $\bar{t} = \frac{1}{M}\sum_i^M t_i$, of the target value. RRMSE can be calculated as

$$\text{RRMSE} = 100\sqrt{\frac{\sum_i^M (t_i - y_i)^2}{\sum_i^M (t_i - \bar{t})^2}}. \qquad (7.11)$$

Fidelity is how closely the extracted rules model the neural network. Fidelity was measured in two ways. First, the mean squared difference

$$\mathit{Fidelity} = \frac{1}{M}\sum_{i}^{p}(y_i - nn_i))^2, \qquad (7.12)$$

where M is the number of instances, nn_i is the output from the neural network, and y_i is the output from the tree for instance i. Second, a relative fidelity measure was calculated that is relative to predicting the mean target value,

$$\mathit{RelativeFidelity} = 100 \times \frac{\sum_i^p (y_i - nn_i)^2}{\sum_i^p (\bar{y} - y_p)^2}. \qquad (7.13)$$

7.4.1 Datasets

The datasets were chosen from the UCI machine learning repository[32]. Based on some initial experiments only datasets on which neural networks outperformed M5 were chosen. If it is already possible to produce a model tree using M5 that outperforms a neural network then there is little incentive to extract a model tree from the neural network. The datasets represent a range of real-world problem domains. The predictor variables include nominal and continuous attributes. The total number of instances, the number of numeric attributes and the number of nominal attributes of each of the datasets is given in Table 7.2.

Dataset	Instances	Numeric	Nominal
bolts	40	7	0
fishcatch	158	5	2
sleep	62	7	0
vineyard	52	4	0
cpu-rm	209	6	1

Table 7.2: Characteristics of datasets used in evaluation.

Bolts The Bolts dataset comes from the operation of an industrial machine that counts the number of bolts to be packaged. The predictor variables include various settings of the machine, such as conveyor belt speed, and the sensitivity of the 'electric eye'. The value to predict is the time the machine will take to count 20 bolts.

Fishcatch The Fishcatch dataset[44] was obtained from data taken from a 158 fish caught in a Finnish lake. The predictor variables are attributes such as species, sex, width, and height of the fish. The value to predict is the weight of the fish.

Sleep The task provided by the Sleep dataset[2] is to predict how many hours of sleep a mammal requires, based on attributes such as brain weight, body weight, life span etc.

Vineyard The Vineyard dataset[80] is the yield of the vine rows in a vineyard close to Lake Erie. The task is to predict the yield for 1991 based on yields from previous years.

CPU The task provided by the CPU dataset[44] is to predict the relative performance of a CPU based on a number of technical attributes of the CPU.

7.4.2 Neural Network Setup

The neural networks used in the evaluation were standard multilayer feed-forward neural networks. The hidden neurons used bipolar sigmoid as the transfer function. For the output nodes a linear function ($f(x) = x$) was used as the transfer function. All nodes used the dot product for the net function. The networks were trained using the GNBR[25] training algorithm that combines the speed of the Levenberg-Marquardt optimisation with Bayesian regularisation as discussed in Chapter 2. The regularisation favours smoother functions that are less likely to overfit and therefore improves the generalisation ability of the neural network. Initial experiments were carried out using standard gradient descent back-propagation, as used in Chapter 5 for the evaluation of ExTree on classification datasets. However, it was found that using gradient descent caused the experiments to take an impractical amount of time to complete. Every result in the evaluation of an algorithm is found by averaging 10 runs of 10-fold validation requiring the neural network to be trained 100 times for each dataset for every set of ExMT parameters evaluated. The line search in the Levenberg-Marquardt algorithm significantly speeds up the training, making the large number of runs required to produce unbiased estimates of predictive accuracy and fidelity feasible. The switch of training algorithm is due to increased complexity of fitting an exact regression curve over finding a decision boundary and does not affect the architecture of the neural network, which remains a multilayer perceptron with one or more hidden layers with bipolar

Dataset	MLP	CART	M5	ExMT(a)	ExMT(b)
bolts	3.72	11.16	6.39	**5.81**	**4.94**
fishcatch	52.51	92.11	57.31	65.65	**50.19**
sleep	3.47	4.12	3.82	**3.52**	**3.52**
vineyard	2.56	3.06	3.82	**2.62**	**2.54**
cpu-rm	20.22	36.75	27.45	**20.54**	**21.77**

Table 7.3: Comparison of Root-MSE.

sigmoid nodes in the hidden layer. Moreover, because the ExMT method takes as its starting point a trained neural network, the actual training algorithm that finds the weights can be any optimisation method; the rule extraction process is unaffected.

7.4.3 Results

The RMSE results (Table 7.3) clearly show that the CART method, which only uses a mean value at the nodes, produces trees with significantly higher error rates than the model tree based methods. The ExMT(a) method produced trees with a lower RMSE than M5 for all but the fishcatch dataset, for which it produced a tree with a higher error rate but a smaller tree size. The ExMT(b) method produced tees with significantly lower errors than both M5 and ExMT(a). Wilcox Signed Rank tests showed the difference between both versions of ExMT and M5 to be significant and also showed the difference between the two version to be significant but less so.

Tables 7.5 and 7.6 show that ExMT(a) had a higher degree of fidelity than the M5 tree except again for the fishcatch dataset. ExMT(b) had a higher degree of fidelity with the neural network than the M5 on all the datasets. It also had a higher degree of fidelity than ExMT(a). Wilcoxon Signed Rank

Dataset	MLP	CART	M5	ExMT(a)	ExMT(b)
bolts	15.6	45.43	25.19	**22.62**	**18.94**
fishcatch	15.14	26.55	16.19	18.67	**14.27**
sleep	84.74	103.2	94.82	**89.2**	**86.11**
vineyard	66.32	79.66	101.93	**65.61**	**66.7**
cpu-rm	11.57	29.81	20.04	**16.22**	**12.98**

Table 7.4: Comparison of root-relative-MSE.

Dataset	M5	ExMT(a)	ExMT(b)
bolts	23.51	**19.79**	**16.79**
fishcatch	46.87	**52.51**	**36.36**
sleep	1.23	**0.31**	**0.29**
vineyard	1.27	**0.58**	**0.51**
cpu-rm	27.45	**20.22**	**19.05**

Table 7.5: Comparison of fidelity root-MSE.

Dataset	M5	ExMT(a)	ExMT(b)
bolts	5.86	**5.13**	**4.37**
fishcatch	13.27	14.86	**10.32**
sleep	29.89	**7.58**	**6.78**
vineyard	35.63	**16.78**	**14.01**
cpu-rm	19.89	**16.86**	**12.6**

Table 7.6: Comparison of fidelity root-relative-MSE.

Dataset	CART	M5	ExMT(a)	ExMT(b)
bolts	6.40	4.45	3.74	10.69
fishcatch	27.82	4.08	3.92	20.06
sleep	11.96	2.30	1.00	14.84
vineyard	35.50	2.33	5.68	15.32
cpu-rm	31.1	2.68	7.04	24.00

Table 7.7: Comparison of tree size for CART, M5, ExMT(a), ExMT(b).

test showed the difference between M5 and both ExMT(a) and ExMT(b) to be significant and also the difference between ExMT(a) and ExMT(b) to be statistically significant.

Table 7.7 shows the average number of nodes in the tree for the various tree induction methods. As expected, the CART approach that is restricted to predicting mean values produces the largest trees. The ExMT(a) method produces trees which are smaller than the equivalent M5 tree on 3 of the 5 datasets, which possibly demonstrates the smoothing potential of extracting from the neural network. The results for ExMT(b) shows the large improvements in error reduction and fidelity have come at the cost of increased tree size.

The results show the potential of extracting model trees from artificial neural networks. The ExMT(a) method being the closest to the original M5 method showed that it was possible to produce trees that were smaller than M5 but have higher accuracy. This indicates that M5 does produce suboptimal model trees. The ExMT(b) method improved on the accuracy and fidelity results of M5 at the cost of increased computational time and larger trees. If we consider the Bolts dataset, as shown in the results section, the M5 obtained an average RRMSE of 25.19 whereas the ExMT(a) algorithm reduced the RRMSE to 22.62.

Figures 7.7 and 7.8 show typical trees created by the algorithms. Both algorithms created a three node tree with three corresponding linear regression models. Although both trees split on the same two attributes, TIME and TOTAL, the split points were different. The linear regression models for the ExMT(a) tree used made use of more of the attributes than the M5 tree. This can be explained by the ExMT(a) method having more data due to the sampling at Stage 3. Furthermore, because the purity measures of Stage 5 are based on the outputs of the neural network, this has the effect of reducing noise in the dataset resulting in more accurate trees.

7.4.4 Comparison with ANN-DT

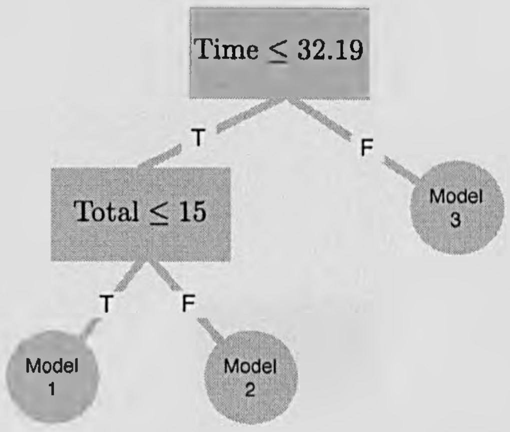
In this section ExMT will be evaluated against the ANN-DT algorithm by Schmitz, *et al*[75]. In the original study introducing ANN-DT it was evaluated using an artificial dataset containing 3 continuous attributes (θ, x, s) and one discrete(ϕ). The attributes are bounded such that $0 \leq (\theta, x, s) \leq 1)$ and $\phi \in \{0, \frac{\pi}{2}\}$. The target attribute, y, is given by

$$y = \sin(4\pi\theta - \phi) + ax + bs + c\epsilon. \tag{7.14}$$

The $c\epsilon$ term adds noise to the dataset with ϵ being a random variable from a normal distribution with mean 0 and standard deviation 1. The training data was 300 randomly sampled points with the constants having values $a = 0.3, b = 0.0, c = 0.2$. Because b is set to 0 the s attribute has no effect, forcing the algorithms to deal with a superfluous variable.

The test set consisted of a further 1,200 randomly sampled points but with the c constant having a value of 0, which results in a test dataset containing no noise. As observed by Schmitz this dataset presents a difficult learning task as several splits on both ϕ and θ are required before the error is reduced significantly and the error term in the training dataset can cause overfitting.

As can be seen from the results in Table 7.8 the ExMT(b) method significantly outperformed the ANN-DT algorithms. This shows the clear advantage of ExMT(b) linear nodes in representing this function. Moreover, the ExMT(b)

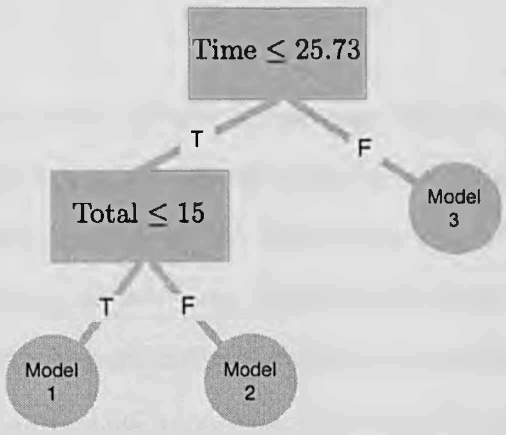


```
Model 1
T20BOLT = 2 * TIME


Model 2
T20BOLT = 0.4649 * TIME + 5.2598


Model 3
T20BOLT = 74.9445
```

Figure 7.7: M5 tree for Bolts dataset .

Dataset	ANN	CART	M5	ANN-DT(e)	ANN-DT(s)	ExMT(b)
Sin Cos R^2	86.1	32.9	61	77.6	82.3	92
Sin Cos Fidelity	-	27.5	56	84	87.2	88

Table 7.8: RMSE for Sine Cosine dataset.



```
Model 1
T20BOLT = -3.3344 * TOTAL + -0.1002 * SENS + 1.9909 * TIME + 34.2875

Model 2
T20BOLT = 0.032  * RUN + 0.6192 * SPEED1 + -0.2163 * TOTAL + 0.6414 * TIME + 5.6058

Model 3
T20BOLT = -0.1146 * RUN + -2.2886 * TOTAL + 1.3716 * NUMBER2 + 0.6851 * TIME + 67.2983
```

Figure 7.8: M5 tree for Bolts dataset .

performed better than a M5 induced tree which shows the benefit of extracting the model from the trained neural network. The fidelity was also higher than the ANN-DT algorithms, again due to ExMT(b) using linear nodes. The standard tree induction methods CART and M5 had very low fidelity, as expected, because they make no attempt to describe the neural network and are included merely as a baseline for comparison.

7.5 Summary

This chapter has introduced, ExMT, a new algorithm for rule extraction from neural networks trained to solve regression problems. The chapter started with a review of the current algorithms for rule extraction, which established that, in comparison to rule extraction from classification based neural networks, extraction from regression neural networks had been a neglected area of research.

The previous classifications extraction algorithms introduced in Chapter 5 had successfully used decision trees to represent the rules extracted and was particularly influenced by C4.5. For regression rules C4.5 is not appropriate so Model Trees were used as the tree type for the extracted rules. Model Trees have the advantage of being a graphical representation of a decision process but can predict a continuous output class. Therefore, Model Trees and the model tree induction algorithm M5 were analysed.

The new extraction algorithm, ExMT, itself was then described. The M5 algorithm used standard deviation for determining the split points which is reasonably fast to calculate, but a problem with this measure and its interaction with the linear models at the leaf nodes was highlighted. For this reason two extraction methods were proposed ExMT(a) that used standard deviation like M5, and ExMT(b) that used a new measure based on the difference between the root mean square error of the nodes linear model and the neural network.

Both algorithms were then evaluated on a number of real world datasets. The results were compared to the M5 algorithm. The results showed that the model tree extraction algorithm produced rules which had a high level of fidelity with the neural network and also were more accurate than the M5 algorithm. Furthermore, the ExMT(b) algorithm significantly outperformed both M5 and the ExMT(a) algorithm. This means the limitations previously discussed in the chapter when using standard deviation for split points does impact real world datasets. Finally, the ExMT(b) algorithm was evaluated against the ANN-DT algorithm using a synthetic dataset which had been introduced in the original ANN-DT paper. This evaluation showed that the ExMT(b) outperformed both versions of ANN-DT.

CHAPTER 8

Conclusion and Future Directions

This thesis has developed new algorithms to extract decision trees from neural networks to solve the well-known 'black box' problem.

8.1 Summary

The aim of this thesis as stated in Chapter 1 was to create a series of algorithms to extract rules from artificial neural networks. This research covered in the preceding chapters has fulfilled this aim. This section will now summarise how the research in each chapter contributed to meeting the objectives.

Chapter 1 provided a review of the general pattern recognition task. The terminology of pattern recognition used throughout the thesis was introduced. A brief introduction to neural networks and decision trees was presented.

Chapter 2 provided an analysis of neural networks. The chapter started by examining the biological origins of artificial neural networks and how this was developed into the model of an artificial neuron. The inability of a single artificial neuron to model non-linear separable classes, such as the XOR function, were highlighted. This limitation of a single artificial neuron led to the development of the multilayer perceptron. Although multilayer neural networks overcame the linear separability problem, it made training more complicated, requiring backpropagation of errors through the neural network. An analysis of how backpropagation and related algorithms trained the weight matrix was presented. A significant problem with moving to multilayered neural networks was the black box nature of the resulting neural networks. The complex weight matrices trained by the backpropagation algorithms give no insight into the relationship between inputs and outputs that the neural network has learnt. Solving this black box problem was the focus of this thesis.

Chapter 3 provided an analysis of decision trees and their respective induction algorithms. Decisions trees, in contrast to neural networks, provide a graphical model of a decision process that is easy to comprehend. The recursive partitioning method with particular emphasis on the information entropy based C4.5 algorithm was analysed. Although much easier to understand, decision trees and the induction algorithms have a number of limitations which were highlighted.

Chapter 4 examined the rule extraction task. The chapter started with a review of the black box nature of neural networks. The Hinton map, an early method for visualising neural network weight matrix, was examined and shown to be insufficient for interpreting the weight matrix. Previous rule extraction techniques were reviewed and analysed using the rule extraction taxonomy provided by Andrews[4]. An outline of rule extraction algorithms that were developed in the remainder of the thesis was presented. Central to these algorithms was the Sampling and Querying approach by Craven[16] which was discussed.

In Chapter 5 the novel rule extraction algorithms ExTree and ExLMT were developed. These algorithms were developed to extract decision trees from neural networks trained in classification domains. Both algorithms were pedagogic and used Sampling and Querying to extract the knowledge hidden in the neural network. The ExTree algorithm was analysed on the synthetic Monk's datasets. This analysis confirmed that the decision trees extracted using ExTree outperformed the C4.5 induced algorithms by finding better split points, eliminating noise in the dataset, and overcoming the limitation of decision trees where decisions towards the bottom of the tree are based on too few instances. ExTree was also analysed on datasets with continuous input attributes. This showed that increasing the number of instances, by sampling and querying, created approximations much closer to the nonlinear decisions boundaries. The second algorithm developed in this chapter was the ExLMT algorithm. This algorithm replaced the leaf nodes of ExTree with logistic models which created a significantly closer approximation to nonlinear decision boundaries. This increased both the predictive classification accuracy and the fidelity with the neural network at the cost of reducing the comprehensibility of the decision tree.

Chapter 6 presented an empirical evaluation of ExTree and ExLMT on 12 real-world datasets. A comprehensive evaluation methodology was developed that was based around repeated k-fold evaluation. This included statistical tests to compare two algorithms on a single dataset, and tests to compare two algorithms over a range of datasets. The extraction algorithms were compared to multilayer peceptrons and the C4.5 induction algorithm. The evaluation measured and compared the predictive classification accuracy, fidelity and comprehensibility of the algorithms. Various aspects of the algorithm were individually evaluated to examine the effect of each, for example, relabelling. The evaluation confirmed the results of the synthetic datasets. As expected the extracted decision trees had substantially higher fidelity with the neural network, but also achieved higher classification accuracy. The ExLMT achieved significantly higher fidelity and classification accuracy than ExTree. To demonstrate the flexibility of ExTree to be applied to different classifier types, it was successfully applied to ensembles of multilayer perceptrons.

In Chapter 7, ExMT a novel algorithm for extracting decision trees in regression domains was developed. This replaces the leaf nodes of a regular decision tree with a linear regression function creating a Model Tree. This chapter started with a review of the regression task and previous algorithms for rule extraction in regression domains. The ExMT algorithm was based on the M5 Model Tree induction algorithm and therefore this was analysed. A significant limitation in the way the M5 algorithm found split points was found and, because of this, two versions of ExMT were developed: one which used the same flawed measure as M5 and a more computationally expensive method of finding the split points. An empirical evaluation on regression datasets was then presented. This compared the extracted model trees with M5 trees and the MLP. This used an evaluation methodology based on that used in Chapter 6 but used new measures for the predictive accuracy and fidelity which were suited to the regression domain. As expected this evaluation showed significant improvements in fidelity but also predictive accuracy. The chapter concluded with a comparison of ExMT with ANN-DT, an earlier rule extraction technique for regression domains, which showed the that ExMT outperformed ANN-DT in both classification accuracy and fidelity.

8.2 Overall Conclusion

Neural networks are powerful classifiers and function approximators but their adoption in many areas has been impeded due to their black box nature. Every successful neural network has learnt an important mapping between the input attributes and the output, but this knowledge is trapped within the complex weight matrix. In this thesis algorithms for extracting decision trees from neural networks have been developed. Extracting decision trees from neural networks has two benefits. First, they open up the black box allowing us to see inside a trained neural network. Second, they produce better classifiers and function approximators than directly inducing decision trees. The cost of this benefit is some reduction in comprehensibility, in comparison to directly induced decision trees, and an increase in computational time. Rule extraction algorithms have been proposed before but the algorithms in this thesis have extended the sampling and querying approach to newer, more powerful decision tree types: logistic model trees and model trees. By using the model tree as form of output, extraction from neural networks trained in regression domains was possible, which has been a neglected area of rule extraction.

8.3 Limitations and Future Directions

This section will briefly examine future research directions resulting from the work presented in this thesis.

8.3.1 Comprehensibility Measures

Andrews et al.[4], as described in Chapter 1 proposed four dimensions with which to measure the quality of the rules extracted by rule extraction algorithms: accuracy, fidelity, consistency, and comprehensibility. This thesis has concentrated on accuracy and fidelity. Comprehensibility was measured through tree size but this is acknowledged to be only one aspect of comprehensibility and further research is required to measure and improve the rule extraction algorithms' performance in this area. Comprehensibility is difficult to measure and difficult to represent in a single number. Although heuristic measures such as tree size can give an indication to the likely comprehensibility of the model, the real test of comprehensibility remains how easily the user of the model can understand the knowledge represented by the model. This type of test is, of course, subjective. In Chapter 6 and Section 7.4 it was shown that the ExTree and ExMT algorithms were applicable to a wide range of real world problem domains. This testing shows the real-world applicability of the algorithms but does not show the algorithms in actual use on a current real-world problem where domain experts are eager to examine the resulting rules. The next stage, now the benchmark testing has been completed, would be to apply these algorithms to current real-world problems. Applying the algorithms in this way would test how well the extracted model could be comprehended by a domain expert.

Chapters 5 and 6 showed how changing the leaf nodes to logistic models decreased the tree size. However, the effect on comprehensibility is hard to measure, as a logistic model is not as simple to understand as a single attribute split but is easier to comprehend than a large subtree. Again, real world testing on current real-world problems in collaboration with domain experts would enable decisions to be made about how much the comprehensibility was affected by the logistic model nodes.

8.3.2 Instance Generation

The current modeling of data attributes is based on randomly sampling from empirical distributions for nominal attributes and from a PDF obtained using kernel density estimation for continuous attributes. The random sampling means the rules are not consistent between runs of the algorithm. Another problem is the relationships between the attributes are not taken into account. Although the random sampling approach is widely applicable and does not require domain knowledge to use, it fails to capture known dependencies among the attributes.

Dependent on the problem domain there may be better ways of creating the new instances. For example, in domains where input data is cheap to capture but the classification is expensive a large number of unlabelled data may exist. For the 'lie detector' mentioned in Chapter 1 it is relatively easy to capture video of people talking and turn those clips into instances, but to label those instances requires prior knowledge of whether the person was being deceitful or not. If an algorithm could use this unlabelled data instead of randomly sampling from the input space this would ensure the new instances maintain any relationships between attributes. In other problem domains relationships between attributes are well-known and the 'new instance generator' used in the extraction algorithms could enforce these relationships. It is expected that by maintaining the attribute relationships the extracted rules would be of higher quality (accuracy and fidelity) and fewer new instances would be required because the instances would be focused on the relevant parts of the instance space.

8.3.3 Active Learning

In domains where there is neither an abundance of unlabelled data or known relationships between attributes it may be possible to use active learning[90] to guide the creation of new instances. Active learning refers to a learning system where the learner guides the selection of instances. An active learner selects the instance whose classification will reveal the most information. The game of twenty questions where an object or person is identified by asking no more than twenty 'yes or no' questions is an example of active learning. A random sample of 20 questions and answers about an object is not nearly as useful as 20 guided questions and answers. Active learning has been applied to neural network learning by Baum[7], but active learning, although not random, can still create instances which have no natural meaning. For example, in the work by Baum the problem was character recognition and the active learning algorithm created images which contained no recognisable characters. Active learning could provide an elegant solution to the creation of new instances but current results in this area are limited with Freund providing the most theoretical results[26].

8.3.4 Fuzzy Logic

A potential way that tree size could be decreased and comprehensibility increased is by using fuzzy splits at the nodes. Fuzzy sets and fuzzy decision trees were briefly discussed in Chapter 3. Fuzzy trees, by relaxing the crisp decision boundaries, are more likely to represent the neural network using a smaller tree. If the fuzzy sets have meaningful linguistic terms (for example temp $\in$ {cold, cool, warm, hot}) then the comprehensibility of the tree will not be significantly degraded. Extracting fuzzy trees seems a logical next step and should lead to smaller, more comprehensible, trees. Combining decision trees with fuzzy trees has been previously examined[18][38]. However, creating optimal membership functions for the attributes remains problematic. It may be possible to use the sampling and querying approach of the neural network that has been successfully used in this thesis to augment the dataset to enhance the selection of the membership functions in a fuzzy decision tree.

8.3.5 Extraction from Recurrent Neural Networks

Previous research had mainly focused on feed-forward neural network classification problems. A significant contribution of this thesis was the application of decision tree extraction to regression neural networks, which had been a neglected area of research. Extraction from recurrent neural networks represents another neglected area of research. Recurrent neural networks are cyclic meaning the output of a layer can be the input of a preceding layer. This enables the neural network to maintain state. This type of neural network is applicable to pattern recognition problems that are temporal, for example speech recognition. The Backpropagation-through-time[93] algorithm is a popular training method for such neural networks and has been applied to real world problems. The recurrent connections makes analysis of what they have learnt even more difficult than for feed-forward neural networks. A significant problem in extracting from recurrent neural networks would be finding a suitable form to represent the extracted knowledge. Temporal logics should be considered but do not fulfil the criterion of being easy to comprehend. There has been some research into creating temporal decision trees[41] which could provide a way of representing the knowledge of a recurrent neural network in an easy to understand form similar to that used in this thesis.

8.3.6 Computational Time

The extraction algorithms developed in this thesis take considerably longer than direct decision tree induction. There are two elements that contribute to this increase in computational time: training the neural network and creating the new instances. Neural Network training as an iterative technique can be very time consuming. How long a neural network will take to train is very difficult to predict because there are no theoretical guarantees that back-propagation will find a solution so training may need to be restarted. Early stopping techniques such as stopping based on a validation set also complicate estimating the training time required by a neural network.

If the extraction algorithm is being used as a decision tree induction algorithm then it is correct to include the neural network training time as part of the computational cost of creating the decision tree. However, the algorithms were developed under the assumption that the algorithm would be extracting rules from a previously existing trained neural network.

The second increase in computational time is the result of new instance creation. The modeling of the dataset by kernel density estimation and the labelling by the neural network increases the computational time at each node. Furthermore, because the new instances reveal more complex relationships, as seen for example in the monk's datasets, this increases the number of nodes in the tree resulting in an increase in computational time that grows exponentially each time the tree size increases by a level.

Although the computational time of creating the tree is increased over C4.5, the computational time for using the resulting tree to make a decision is the same for trees of the same size. In many domains an increase in the computational time of producing a tree is worthwhile if it produces a more accurate classifier.

8.4 Final Comments

This thesis has proposed rule extraction algorithms for both classification and regression problems. Although extracting rules from classification based neural networks has been the subject of previous research, the algorithms presented in this thesis have extended that work by using new advances in decision tree induction. In contrast, rule extraction for regression problems is an area that has, so far, been neglected. The model tree extraction algorithms presented demonstrate it is possible to extend decision tree extraction to neural networks in regression domains and that it is beneficial.

The 'black-box' problem is far from solved but the methods show the potential for decision tree extraction from neural networks. The methods achieved higher classification accuracy than the decision tree induction algorithms which alone illustrates the utility of training a neural network and then extracting a decision tree, even if the neural network itself is not of interest and all that is desired is the most accurate decision tree possible. The fidelity results showed that much of the knowledge was being extracted but the results show there is still room for improvement in rule extraction. Although there is work to be done hopefully the results presented in this thesis represent a step towards opening the 'black box'.

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables.* Dover Publications, 1972.

[2] T. Allison and D. Cicchetti. Sleep in mammals: ecological and constitutional correlates. *Science*, 194(4266):732–734, 1976.

[3] H. C. Anderson, A. Lotfi, and L. C. Westphal. Comments on functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Trans. Neural Networks*, 9(6):1529–1531, 1998.

[4] R. Andrews, J. Diederich, and A. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

[5] A. S. Avila Garcez, K. Broda, and D. M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125(1-2):155–207, 2001.

[6] T. L. Bailey and C. Elkan. Estimating the accuracy of learned concepts. In *IJCAI-93, Vols 1 and 2 - Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 895–900, 1993. BA18B.

[7] E. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Trans. Neural Networks*, 2:5–19, 1991.

[8] D. A. Belsley, E. Kuh, and R. E. Welsch. *Regression diagnostics : identifying influential data and sources of collinearity.* Wiley, 1980.

[9] C. M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, Oxford, 1995.

[10] O. Boz. Converting a trained neural network to a decision tree. In *Electrical Engineering and Computer Science*, page 180. Lehigh University, Lehigh, 2000.

[11] L. Breiman. Bagging predictors. *Machine Learning 1*, 24:123–140, 1996.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Chapman & Hall, New York, 1984.

[13] J. L. Castro, C. J. Mantas, and J. M. Benítez. Neural networks with a continuous squashing function in the output are universal approximators. *Neural Netw.*, 13(6):561–563, 2000.

[14] B. Cestnik, I. Kononenko, and I. Bratko. ASSISTANT 86: A knowledge-elicitation tool for sophisticated users. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning*, pages 31–45. Sigma Press, Wilmslow, GB, 1987.

[15] M. W. Craven and J. W. Shavlik. Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 73–80, Amherst, MA, 1993. Morgan Kaufmann.

[16] M. W. Craven and J. W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *International Conference on Machine Learning*, pages 37–45, New Brunswick, NJ, 1994.

[17] M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems 8*, pages 24–30, Denver, CO, 1997. MIT Press.

[18] K. A. Crockett, Z. Bandar, D. McLean, and J. O'Shea. On constructing a fuzzy inference framework using crisp decision trees. *Fuzzy Sets and Systems*, 157(21):2809–2832, 2006.

[19] J. Curtis, G. Matthews, and D. Baxter. On the effective use of cyc in a question answering system. In *IJCAI Workshop on Knowledge and Reasoning for Answering Question*, Scotland, 2005.

[20] R. Davis, B. Buchanan, and E. Shortliffe. Production rules as a representation for a knowledge-based consultation program. *Artificial Intelligence*, 8(1):15–45, 1977.

[21] H. Demuth and M. Beale. *Neural Network Toolbox*. Mathsoft, 2002.

[22] C. M. Ennett, M. Frize, and E. Charette. Improvement and automation of artificial neural networks to estimate medical outcomes. *Medical Engineering and Physics*, 26(3):321–328, 2004.

[23] Fausett and L. V. *Fundamentals of neural networks : architectures, algorithms, and applications*. Prentice-Hall, Englewood Cliffs, NJ, 1994.

[24] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals Eugen.*, 7:179–188, 1936.

[25] F. Foresee and M. Hagan. Gauss-newton approximation to bayesian learning. In *Proceedings of International Conference on Neural Networks (ICNN'97)*, volume 3, pages 1930–5, 1997.

[26] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168, 1997.

[27] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 32(2):337–374, 2000.

[28] L. M. Fu. Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 373–389, 1995.

[29] R. P. Gorman and T. J. Sejnowski. Analysis of hidden units in a layered network to classify sonar targets. *Neural Network*, 1:75–89, 1988.

[30] A. Gower. a general coefficient of similarity and some of its properties. *Biometrics*, 27:857-872, 1971.

[31] S. Haykin. *Neural networks: a comprehensive foundation.* Prentice Hall, second edition, 1999.

[32] S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998.

[33] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[34] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551–560, 1990.

[35] R. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[36] J. P. Ignizio. *An Introduction To Expert Systems.* Mcgraw-Hill, 1991.

[37] J. Jang and C. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Trans. Neural Networks*, 4(1):156–159, 1993.

[38] C. Z. Janikow. Fuzzy decision trees: Issues and methods. *IEEE Trans. Syst., Man, Cybern. C*, 28(1):1–14, 1998.

[39] B. JS. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In S. F. H. J., editor, *Neurocomputing, Algorithms, Architectures and Applications. Proceedings of the NATO Advanced Research Workshop*, pages 227–36, 1990.

[40] N. Kandil, K. Khorasani, R. Patel, and V. Sood. Optimum learning rate for backpropagation neural networks. In *Proceedings of the Candaian Conference on Electrical and Computer Engineering*, pages 465–468, Vancouver,BC, Canada, 1993.

[41] K. Karimi and H. J. Hamilton. Temporal rules and temporal decision trees: A c4.5 approach. Technical report, University of Regina, 2001.

[42] N. K. Kasabov. Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems. *Fuzzy Sets Systems*, 82(2):135–149, 1996.

[43] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29:119–127, 1980.

[44] D. Kilpatrick and M. Cameron-Jones. Numeric prediction using instance-based learning with encoding length selection. In *Progress in Connectionist-Based Information Systems*, pages 984–987. Springer-Verlag, 1998.

[45] D. Klahr and R. S. Sieglar. Three aspects of cognitive development. *Cognitive Psychology*, 8:481–520, 1976.

[46] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, May 2005.

[47] D. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project.* Addison-Wesley, 1990.

[48] T. Masters. *Practical neural network recipes in C++.* Academic Press, Boston, 1993.

[49] T. Masters. *Advanced algorithms for neural networks : a C++ sourcebook.* Wiley, New York, 1995. Timothy Masters. ill. ; 24 cm. + 1 computer disk (3 1/2 in.) System requirements for accompanying computer disk: C++.

[50] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[51] C. McMillan, M. C. Mozer, and P. Smolensky. The connectionist scientist game: Rule extraction and refinement in a neural network. In *Thirteenth Annual Conference of the Cognitive Science Societ,*, pages 424–430, 1991.

[52] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, New York, 1994.

[53] M. L. Minsky and S. Papert. *Perceptrons : An Introduction To Computational Geometry.* MIT Press, Cambridge, MA, 1969.

[54] P. Murphy and M. Pazzani. Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 183–187, Evanston, IL., 1991.

[55] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[56] D. Nauck and R. Kruse. A neuro-fuzzy method to learn fuzzy classification rules from data. *Fuzzy Sets Syst.*, 89(3):277–288, 1997.

[57] K. M. Neaupane and S. H. Achet. Use of backpropagation neural network for landslide monitoring: a case study in the higher himalaya. *Engineering Geology*, 74(3-4):213–226, 2004.

[58] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *International Joint Conf on Neural Networks*, volume 3, pages 21–26, San Diego, 1990.

[59] P. C. Pendharkar. A threshold-varying artificial neural network approach for classification and its application to bankruptcy prediction problem. *Computers and Operations Research*, 32(10):2513–2522, 2005.

[60] D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments in learning by back propagation. Technical report, Carnegie-Mellon University, 1986.

[61] L. Prechelt. A study of experimental evaluation of neural network learning algorithms: Current research practise. Technical Report 19/94, Fakultat fur Informatik, Universtitat Karlsruhe, D-76128 Karlsruhe, Germany, Aug 1994.

[62] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge,UK, 1988.

[63] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[64] J. R. Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.

[65] J. R. Quinlan. *C4.5 : Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[66] J. R. Quinlan. Combining instance-based and model-based learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 236–243, Amherst, Massachusetts, 1993. Morgan Kaufmann.

[67] J. R. Quinlan. Comparing connectionist and symbolic learning methods. In *Proceedings of a workshop on Computational learning theory and natural learning systems (vol. 1) : constraints and prospects*, pages 445–456, Cambridge, MA, USA, 1994. MIT Press.

[68] M. Riedmiller and H. Braun. RPROP- A fast adaptive learning algorithm, 1992.

[69] B. D. Ripley. *Pattern recognition and neural networks.* Cambridge University Press, Cambridge, 1996.

[70] J. Rothwell, Z. Bandar, J. O&#x2019;Shea, and D. McLean. Charting the behavioural state of a person using a backpropagation neural network. *Neural Comput. Appl.*, 16(4):327–339, 2007.

[71] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1. MIT Press, Cambridge, 1986.

[72] K. Saito and S. Nakano. Medical diagnostic expert system based on pdp model. In *Proceedings of IEEE International Conference on Neural Networks*, volume 1, pages 255–262, 1998.

[73] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[74] G. P. Schmitz, C. Aldrich, and F. Gouws. Extracting decision trees from artificial neural networks. In *Proc. Minerals and Materials '96*, pages 250–257, South Africa, 1996.

[75] G. P. Schmitz, C. Aldrich, and F. S. Gouws. Ann-dt: An algorithm for extraction of decision trees from artificial neural networks. *IEEE Trans. Neural Networks*, 10(6):1392–1401, 1999.

[76] R. Setiono, W. K. Leow, and J. Y. L. Thong. Opening the neural network black box: an algorithm for extracting rules from function approximating artificial neural networks. In *ICIS '00: Proceedings of the twenty first international conference on Information systems*, pages 176–186, Atlanta, GA, USA, 2000. Association for Information Systems.

[77] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 37:379–423, 1948.

[78] E. Shortliffe. *Computer-based medical consultations, MYCIN.* Elsevier, New York, 1976.

[79] B. W. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall, London, 1986.

[80] J. S. Simonoff. *Smoothing methods in statistics.* Springer-Verlag, New York, 1996.

[81] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988.

[82] K. O. Stanley, B. D. Bryant, and R. Miikulainen. Evolving neural network agents in the nero video game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG'05)0*, Pistcataway, NJ, 2005. IEEE.

[83] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

[84] S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. D. Jong, S. Dzeroski, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. Michalski, T. Mitchell, P. Pachowicz, B. Roger, H. Vafaie, W. V. de Velde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1991.

[85] A. Tickle, F. Maire, G. Bologna, R. Andrews, and J. Diederich. Lessons from past, current issues, and future research directions in extracting knowledge embedded in artificial neural networks. In S. Wermter and R. Sun, editors, *Hybrid Neural Systems.* Springer Verlag, 2000.

[86] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich. The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Trans. Neural Networks*, 9(6):1057–1068, 1998.

[87] G. G. Towell and J. W. Shavlik. The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1993.

[88] A. M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[89] P. Utgoff and C. Bradley. An incremental method for finding multivariate splits for decision trees. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 58–65. Morgan Kaufmann, 1190.

[90] S. Vijayakumar and H. Ogawa. Improving generalization ability through active learning, 1999.

[91] Y. Wang and I. Witten. Induction of model trees for predicting continuous classes. In *Proc European Conference on Machine Learning Poster Papers*, pages 128–137, Prague, Czech Republic, 1997.

[92] I. Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems.* Morgan Kaufmann, 1997.

[93] P. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, Jan 1990.

[94] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences.* PhD thesis, Harvard University, 1974.

[95] H. White. Connectionist nonparametric regression - multilayer feedforward networks can learn arbitrary mappings. *Neural Networks*, 3(5):535–549, 1990.

[96] R. W. Williams and K. Herrup. The control of neuron number. *Annu Rev Neurosci*, 11:423–453, 1988.

[97] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–260, 1992.

[98] D. Yuret. The binding roots of symbolic ai: a brief review of the cyc project. Unpublished Paper, 1996.

[99] L. A. Zadeh. Fuzzy sets. *Information and Control*, pages 338–353, 1965.

[100] J. Zhao, J. Shawetaylor, and M. VanDaalen. Learning in stochastic bit stream neural networks. *Neural Networks*, 9(6):991–998, 1996.

APPENDIX A

Published Research

Darren Dancey, Z. A. Bandar, David McLean: Logistic Model Tree Extraction From Artificial Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics*, Part B 37(4): 794-802 (2007)

Darren Dancey, David McLean, Zuhair Bandar: Decision Tree Extraction from Trained Neural Networks. *FLAIRS Conference* 2004

Logistic Model Tree Extraction From Artificial Neural Networks

Darren Dancey, Zuhair A. Bandar, and David McLean

Abstract—**Artificial neural networks (ANNs) are a powerful and widely used pattern recognition technique. However, they remain "black boxes" giving no explanation for the decisions they make. This paper presents a new algorithm for extracting a logistic model tree (LMT) from a neural network, which gives a symbolic representation of the knowledge hidden within the ANN. Landwehr's LMTs are based on standard decision trees, but the terminal nodes are replaced with logistic regression functions. This paper reports the results of an empirical evaluation that compares the new decision tree extraction algorithm with Quinlan's C4.5 and ExTree. The evaluation used 12 standard benchmark datasets from the University of California, Irvine machine-learning repository. The results of this evaluation demonstrate that the new algorithm produces decision trees that have higher accuracy and higher fidelity than decision trees created by both C4.5 and ExTree.**

Index Terms—**Artificial intelligence, feedforward neural networks, multilayer perceptrons (MPLs), neural networks.**

I. Introduction

ARTIFICIAL neural networks (ANNs) are universal approximators and, therefore, can approximate any Borel measurable function to an arbitrary accuracy [1]. For classification, this means that neural networks can easily solve any practical classification problem [2] and have been successfully applied to a diverse range of problem domains. For example, recent applications have included problems from financial [3], engineering [4], and medical [5] domains.

However, despite their relative success, the further adoption of neural networks in some areas has been impeded due to their inability to explain, in a comprehensible form, how they have made a decision. This lack of transparency in the neural network's reasoning has been termed the Black Box problem. Andrews *et al.* [6] observed that ANNs must obtain the capability to explain their decisions in a human-comprehensible form before they can gain widespread user acceptance and to enhance their overall utility as learning and generalization tools.

Neural networks store their "knowledge" in a series of real-valued weight matrices representing a combination of nonlinear transforms from an input space to an output space. Rule extraction attempts to translate this numerically stored knowledge into a symbolic form that can be readily comprehended. The ability to extract symbolic knowledge has many potential advantages: the knowledge obtained from the neural network can lead to new insights into patterns and dependencies within the data; from symbolic knowledge, it is easier to see which features of the data are the most important; and the explanation of a decision is essential for many applications, such as safety-critical systems.

Manuscript received December 21, 2005; revised June 28, 2006. This paper was recommended by Associate Editor J. Basak.

The authors are with the Department of Computing and Mathematics, Manchester Metropolitan University, Manchester, U.K. (e-mail: d.dancey@mmu.ac.uk).

Digital Object Identifier 10.1109/TSMCB.2007.895334

Andrews *et al.* [6] and Tickle *et al.* [7], [8] summarize several proposed approaches to rule extraction. Many of the earlier approaches required specialized neural network architectures or training schemes. This limited their applicability; in particular, they cannot be applied to *in situ* neural networks. The other approach is to view the extraction process as a learning task. This approach does not examine the weight matrices directly but tries to approximate the neural network by learning its input–output mappings.

An example of this second approach has been to extract decision trees from the neural network [9]–[11]. Decision trees [12], [13] are a graphical representation of a decision process. The combination of symbolic information and graphical presentation make decision trees one of the most comprehensible representations of pattern recognition knowledge. However, decision trees are a more limited form of classifier than neural networks [14]. This paper presents a new rule extraction method that extracts a logistic model tree (LMT) from a trained neural network. LMTs [15] are a recent addition to decision trees that replace the terminal nodes of a decision tree with logistic regression functions. This has the advantage of producing decision trees that are more comprehensible, have higher accuracy, and have higher fidelity with the neural network than previous decision tree extraction algorithms.

This paper is organized as follows. The next section recaps the pattern classification problem and some relevant techniques used to solve it. Section III overviews rule extraction and important previous rule extraction methods. Section IV describes ExLMT, which is our new rule extraction method. In Section V, ExLMT is empirically evaluated on a number of standard benchmark datasets from the University of California, Irvine (UCI) machine-learning repository. Sections VI and VII provide the discussion and the conclusion, respectively.

II. Pattern Classification

The basic framework for classification [2] is that objects need to be classified as coming from a number of classes $C_1, \ldots, C_K$. A process called feature extraction takes a number of measurements p from the object. This produces a vector of features X commonly called an instance. X, therefore, belongs to an instance space $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_p$, where $\mathcal{X}_i$ is either the set of real numbers for continuous valued features or a finite set for nominal valued features. The task is then to build a classifier $\hat{c}$ that, given an instance, $X = x$, will classify it as belonging to one of the K classes, i.e.,

$$\hat{c} : \mathcal{X} \rightarrow \{1, 2, \ldots, K\}. \tag{1}$$

To estimate $\hat{c}$, a training set $\mathcal{T}$ must be available that consists of a series of instances augmented with a known classification. These example instances with known classification can then be used to estimate the parameters in $\hat{c}$.

Neural networks are one method of implementing such a classifier. ExLMT also makes use of decision trees and logistic regression to implement classifiers, and these three methods will be briefly described next.

1) Neural Networks: The field of neural networks consists of a large collection of models and techniques originally inspired by biological nervous systems, such as the human brain [16], [17]. The basic building block of neural networks is the artificial neuron [18]. These artificial neurons accept a number of weighted inputs then process these inputs to produce an output. It is the value of these weights that determine the function mapping of the neural network. Using the backpropagation algorithm [19], multiple layers of perceptrons organized into a network are able to learn nonlinear mappings such as the pattern recognition task of (1).
2) Decision Trees: Decision trees [12], [13] are one of the most widely used classifier models. They are directed acyclic graphs consisting of nodes and connections (edges) that illustrate decision rules. Each nonterminal node has a splitting test associated with it, which splits the data into mutually exclusive subsets. The terminal nodes called leaves represent a classification. This has the effect of partitioning the instance space $\mathcal{X}$ into a series of disjoint regions R separated by axis-parallel hyperplanes

$$\mathcal{X} = \bigcup_{r \in R} \mathcal{X}^r, \mathcal{X}^r \cap \mathcal{X}^{r'} = \emptyset, \qquad \text{where} \quad \mathcal{X}^r \neq \mathcal{X}^{r'}. \tag{2}$$

3) Logistic Regression: Logistic regression [20] is a statistical method used to predict posterior-class probabilities $P(C = k|X = x)$ for the K classes. Logistic regression for n variables fits a logistic function of the form

$$y = \frac{e^{\beta_0} + \sum_1^n \beta_i x_i}{1 + e^{\beta_0 + \sum_1^n \beta_i x_i}} \tag{3}$$

to the class probabilities, where β_i are the parameters to be most commonly estimated using maximum-likelihood estimation.

III. Rule Extraction

Rule extraction from neural networks aims to reduce the complexity of a neural network into a more easily understood symbolic form. These rules can then be analyzed for trustworthiness for safety-critical systems or used to provide insights into the relationships found by the neural network.

Andrews *et al.* [6] classifies rule extraction algorithms along the following five dimensions:

1) expressive power;
2) translucency;
3) specialized training regimes;
4) quality of the extracted rules;
5) algorithmic complexity.

The expressive power refers to the type of rules extracted from the neural network. Previous rule extraction techniques have extracted rules expressed in various form including Boolean logic [21], fuzzy logic [22], IF ... THEN ... rules [23], n-of-m rules [24], and decision trees [9], [10].

The Translucency means the level of granularity with which the neural network is examined. Craven and Shavlik [9] divided these into decompositional techniques, which examined the individual weights, and pedagogical techniques, which treated the neural network as a black box and learns the concept represented by the neural network by using it as an oracle.

Many of the rule extraction algorithms require the standard neural network training algorithms, such as backpropagation, to be modified. Although such techniques have been successful, they tend not to be portable across different neural network types. Andrews *et al.* [6] proposes four metrics for measuring the quality of the rules extracted from the neural network: accuracy, fidelity, consistency, and comprehensibility. Accuracy measures the ability of the rule set to correctly classify previously unseen instances from the problem domain, i.e.,

$$P\left(\hat{c}(X) = C\right). \tag{4}$$

Fidelity is how well the extracted classifier ($\hat{c}$) corresponds to the original neural network (nn). It can be stated as the probability

$$P\left(\hat{c}(X) = nn(X)\right). \tag{5}$$

Consistency, in this context, is whether the extracted rule set is the same under different training sessions of the neural network. Comprehensibility is a measure of the number of rules produced by the extraction algorithm. The final dimension is algorithmic complexity, which attempts to provide a measure of the efficiency of the technique, considering such aspects as whether the algorithm scales exponentially with the number of hidden nodes or inputs.

A. Existing Extraction Methods

The subset rule extraction method [23] is typical of the decompositional approach, and similar methods have been proposed by Saito and Nakano [25] and Fu [21]. The subset method extracts a series of rules from each node in the network. A rule is created for each combination (or subset) of inputs that could cause a node to activate. For example, given the node in Fig. 1, the following rules could be extracted: $a \wedge b \wedge c \Longrightarrow y$, $a \wedge b \wedge \neg d \Longrightarrow y$, $a \wedge c \wedge \neg d \Longrightarrow y$, $b \wedge c \wedge \neg d \Longrightarrow y$, $b \wedge c \wedge \neg d \Longrightarrow y$. This particular implementation requires the

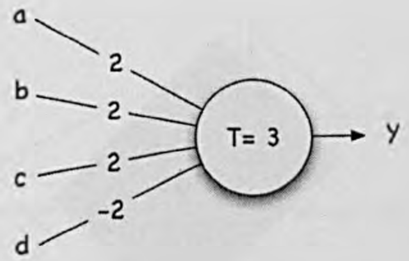


Fig. 1. Neural node with four inputs and a threshold value of three.

outputs of the nodes to be binary and, therefore, cannot be applied to preexisting multilayer perceptrons (MLPs) that normally have nodes with real valued outputs. Moreover, the number of rules increases exponentially with the number of nodes, making the algorithm intractable for large networks. This approach was extended by the n-of-m method, again by Towell and Shavlik [23]. A n-of-m rule contains a set m of tests, of which n must be satisfied for the rule to be evaluated as *true*. For example, the n-of-m rule 2-of-$\{r_1, r_2, r_3\}$ is equivalent to $(r_1 \wedge r_2) \vee (r_1 \wedge r_3) \vee (r_2 \wedge r_3)$. This style of rule is particularly appropriate for representing the activation of a neural node. For example, the rules of Fig. 1 can be represented as 3-of-$\{a, b, c, \neg d\}$. However, for a multiple-layered network to be represented in a concise number of n-of-m rules and overcome the exponential growth problem of the subset algorithm, the antecedents of the rules should be equivalent, i.e., it does not matter which n is *true*. Standard backpropagation has no predisposition to favor such an arrangement. Therefore, either the neural network needs to be initialized using a preexisting rule set and/or trained using a special training algorithm.

Algorithms [26], [27] have been proposed which extract fuzzy rule sets [28] from neural networks. These approaches usually require a domain expert to label the resulting fuzzy sets and/or require specialized neural network architectures and training algorithms. These approaches have generally been applied to rule refinement, where a preexisting set of fuzzy rules have their membership functions refined by the neural network. Jang and Sun [29] have noted a functional equivalence between radial basis function networks and fuzzy inference systems under some conditions. However, it has been shown that the equivalence conditions are more restrictive than was initially thought, resulting in special training algorithms again being required [30].

Trepan [31] follows the pedagogical approach to rule extraction. Trepan creates an n-of-m decision tree [32], which, in addition to the C4.5-style splitting rule, can make use of an n-of-m splitting rule at any of the nodes. The use of n-of-m splits can fit certain concepts more naturally than C4.5-style splits at the cost of a certain amount of comprehensibility. Another interesting feature of Trepan is its use of best first tree expansion in contrast to the more usual depth-first expansion. The next node to expand is the node that maximizes the function

$$n^* = \arg\max_n \left(\text{reach}(n)\left(1 - \text{fidelity}(n)\right)\right) \tag{6}$$

where reach(n) is the number of instances that have reached node n and fidelity(n) is the percentage of instances at node n that the decision tree and the neural network are classified as the same class. This has the effect of concentrating growth of the tree in the region that increases fidelity the most. However, after the tree is fully grown and pruned, the difference between the two methods is negligible. But, the real advantage of this approach is the ability to more precisely control the accuracy growth tradeoff. To decide which attribute to base the splitting test on, Trepan uses information gain. To extract the "knowledge" from the neural network, Trepan uses a sampling-and-querying approach. The neural network is used as an oracle, which can be queried for the class assignment of a sampled instance. To create a query instance, Trepan models the original dataset using an empirical distribution for nominal attributes and kernel density estimation [33] for the continuous attributes. The empirical distribution means that the nominal values are sampled with a probability based on their frequency in the original dataset. For the continuous attributes, a probability-density function (PDF), using a kernel density estimate with a Gaussian kernel, is sampled.

ExTree [10] creates a tree using the more comprehensible C4.5-style splitting rules. Unlike Trepan, ExTree uses a simple depth-first tree expansion scheme, resulting in large trees that overfit the data. A similar method to that employed in Quinlan's C4.5 algorithm is then used to prune the tree.

ExTree uses a slightly modified version of information-gain ratio, which is a modification to information gain that compensates for multiway splits. ExTree, like Trepan, samples from empirical distributions for nominal attributes and a kernel density estimate of the PDF for the continuous attributes.

A further advantage of the pedagogical approaches is that they can be applied to any "black-box" classifier, such as ensembles of neural networks [34].

IV. ExLMT

This section describes ExLMT, a new method of extracting an LMT from a neural network. Current decision tree extraction methods such as Trepan and ExTree have produced reasonable results on many datasets, but there remains a significant gap between the accuracy of the neural network and the extracted decision tree. This clearly indicates that more information remains to be extracted. Moreover, on many datasets, the extracted decision trees and the neural network disagree on the classification on a significant number of instances (low fidelity).

ExLMT extracts a form of LMT from the neural network. The LMT method is a recent contribution to the machine-learning field [15]. Decision trees produced by LMT are similar to standard decision trees but have the terminal nodes replaced by a polytomous logistic regression model. The replacement of the terminal nodes by logistic models has two significant effects: The decision tree now predicts class probabilities instead of giving a simple class assignment and the nodes are linear combinations of a subset of the attributes resulting in decision hyperplanes that are not axis-parallel. Fig. 2 shows how a two LMT tree may fit a 2-D instances space. Fig. 3 gives

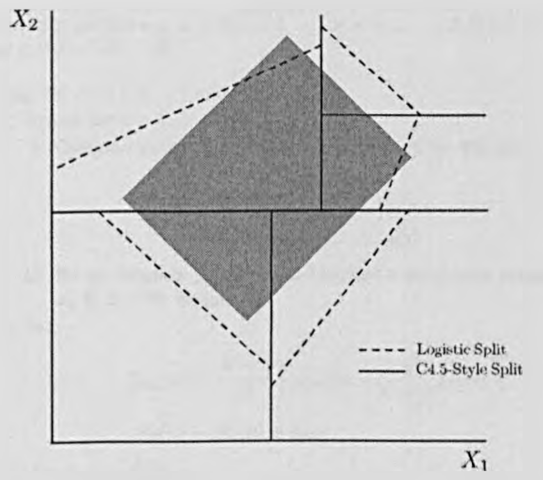


Fig. 2. Decision hyperplanes of an LMT tree. Instances inside the gray box are of class 1. Instances outside the box are of class 2. The solid axis-parallel lines show the C4.5-style splitting rules, and the dotted lines show the nonaxis parallel splits achievable by using the logistic regression nodes.

an overview of the ExLMT algorithm. Each of the component steps will now be expanded.

Step 1) **Acquire or train the neural network:** before the ExLMT can be used to build a decision tree, a trained neural network is required. The ExTree method, being a pedagogical type of rule extraction, is independent of the neural network architecture and training algorithm.

Step 2) **Relabel the dateset:** recall that the original training set $\mathcal{T}$ consisted of the instance X and a known classification C. A new relabeled dataset $\mathcal{R}$ is created replacing the known classification with the mapping of the neural network $\hat{c}$, such that

$$\mathcal{R} = \{X, \hat{c}(X)\}. \tag{7}$$

Step 3) **Generate new data:** ExLMT uses Craven's sampling-and-querying approach [31] to elicit knowledge from the neural network. ExLMT models the original dataset then samples this model to create new instances. These new instances are then used to query the neural network, obtaining class labels. This has the effect of expanding the original dataset. To model the nominal attributes, an empirical distribution is used. This means that nominal values in the new instances are sampled according to their frequency in the original dataset. For continuous attributes, a PDF is estimated using kernel density estimation with a Gaussian kernel

$$f(x) = \frac{1}{m}\sum_{i}^{m}\left(\frac{1}{\sqrt{2\pi\sigma}}e^{-\left(\frac{x-u_i}{2\sigma}\right)^2}\right) \tag{8}$$

where m is the number of original instances, u_i is the attribute value for the ith examples, and σ is the width for the Gaussian kernel. As illustrated in Fig. 4, the kernel density estimation can be thought of as creating a PDF by summing a series of Gaussian functions centered on the current data

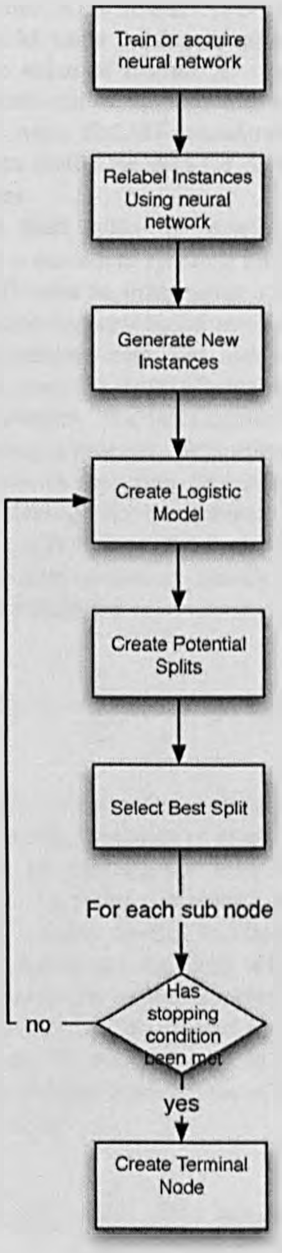


Fig. 3. Outline of the ExLMT algorithm.

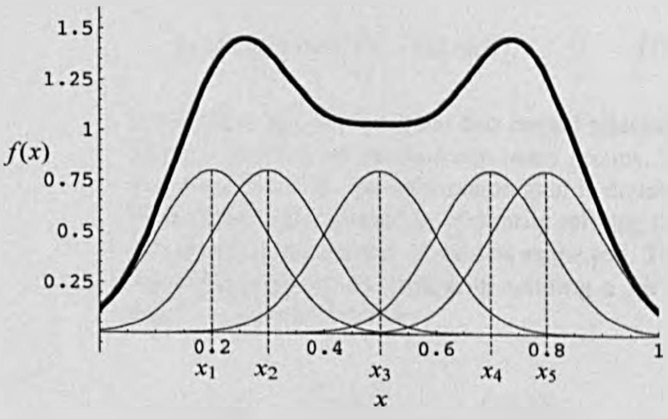


Fig. 4. Kernel density estimation using five Gaussian kernels.

```
1) Start with weights $w_{ik} = 1/N, i = 1, \ldots, N, k = 1, \ldots, J, F_k(x) = 0$
   and $p_k(x) = 1/K \quad \forall k$

2) Repeat for $m = 1, 2, \ldots, M$:
   a) Repeat for $k = 1, \ldots, K$:
      i) Compute working responses and weights in the kth class
            $z_{ik} = \frac{y^*_{ik} - p_k(x_i)}{p_k(x_i)(1 - p_k(x_i))}$
            $w_{ik} = p_k(x_i)(1 - p_k(x_i))$
      ii) Fit the function $f_m k(x)$ by a weighted least-squares regression of
          $z_{ik}$ to $x_i$ with weights $w_{ik}$
   b) Set
            $f_{mk}(x) \leftarrow \frac{K-1}{K}\left(f_{mk}(x) - \frac{1}{K}\sum_{j=1}^{K} f_{mj}(x)\right)$
            $F_k(x) \leftarrow F_k(x) + f_{mk}$
   c) $P_k(x) \leftarrow \frac{e^{F}_k(x)}{\sum_{j=1}^{K} e^{F_j(x)}}$,
3) Output Classifier
            $\arg\max_k F_k(x)$
```

Fig. 5. LogitBoost: an adaptive Newton algorithm.

points with a standard deviation, or bandwidth, determined by the size of the dataset. Based on some preliminary experiments, ExLMT uses a bandwidth of $1/\sqrt{m}$. The new instances' attribute values are then sampled from this PDF. The new instances $\mathcal{T}^*$ are then labeled by the neural network to produce a set of instances to be added to the relabeled dataset $\mathcal{R}$, such that

$$\mathcal{T}' = \mathcal{R} \cup \mathcal{T}^*. \tag{9}$$

The next three steps create the LMT tree following the procedure given by Landwehr [15]. Nodes continue to be split while they contain at least ten instances or all the instances belong to the same class.

Step 4) **Create initial logistic regression model:** an initial logistic regression model is built using all the data in $\mathcal{T}'$. The logistic regression model is then fitted using the LogitBoost method [35]. LogitBoost uses a ensemble of functions F_k to predict classes $1, \ldots, K$ using M "weak learners." Fig. 5 details the LogitBoost algorithm as originally given by Friedman.

$$F_k(x) = \sum_{m=1}^{K} f_{mk}(x) \tag{10}$$

Each of the "weak learners" f_{mk} can be any algorithm that fulfils (1). When f_{mk} are linear functions in x, then F_{mk} is equivalent to the logistic model. The LogitBoost algorithm can then be seen as an iterative Newton method of fitting the logistic regression function.

Step 5) **Create candidate splits:** when deciding which attribute to split on, ExLMT considers two types of splitting rules. For discrete features, ExLMT creates a branch for each possible value of the feature. For real valued features, a binary split is made with two outcomes $X_i \leq \alpha$ and $X_i > \alpha$. To determine the threshold value α, the set of instances are sorted on the value of feature X_i. An ordered set of m instances can be divided into two ordered subsets $m - 1$ ways. ExLMT considers each of these $m - 1$ ways to divide the data for each of the real-valued features.

Step 6) **Select best split:** the previous step resulted in set of z candidate splitting tests, $\{T_1, T_2, \ldots, T_z\}$; ExLMT uses an information-gain ratio [12], an information entropy-based method to choose among this candidate tests. The aim is to select the test, which gives the most information about the class of the instances. The information gained by an event occurring is inversely proportional to the probability of the event occurring. The information of an event E occurring can be defined as $\log_2(1/p(E)) = -\log_2(p(E))$ bits. Thus, the average amount of information needed to classify a pattern in a set S can be calculated as

$$\text{info}(S) = -\sum_{i=1}^{K} p(C_i) \log_2 (p(C_i)) \text{ bits} \tag{11}$$

with $P(C_i)$ being the probability of an instance in set S being a member of class C_i. The information gained by splitting the data according to test T can be found by calculating the average amount of information needed to classify an instance before splitting the data and subtracting the amount of information needed to classify an instance for each of the subsets created by the split. Therefore, for a test T which results in N subsets, the sum of the average information of the N subsets, $S = \bigcup_{i=1}^{N} S_i$, is

$$\text{info}_T(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \times \text{info}(S_i) \text{ bits}. \tag{12}$$

The total information gained by test T can be calculated as

$$\text{gain}(T) = \text{info}(S) - \text{info}_T(S). \tag{13}$$

Information gain has a natural bias toward selecting the test, which splits the data into many groups. To overcome this bias, the information gain is divided by the information gained by arbitrarily splitting the set into the same number of subsets as the test. The information gained by arbitrarily splitting a set S into N subsets is given by

$$\text{split info}(T) = \sum_{i=1}^{N} \frac{|S_i|}{|S|} \times \log_2 \left(\frac{|S_i|}{|S|}\right). \tag{14}$$

TABLE I
DATASET CHARACTERISTICS

Dataset	Instances	Features	Continuous	Classes
Balance Scale	625	4	4	3
Sonar	208	60	60	2
Diabetes	768	8	8	2
Heart	270	13	13	2
Hepatitis	155	19	6	2
Housing	506	13	13	2
Labor	57	16	8	2
Wine	178	13	0	3
Grub Damage	155	8	2	4
Zoo	101	16	0	7
Iris	150	4	0	3
Primary Tumor	339	16	0	21

The gain ratio of test T can, thus, be calculated as

$$\text{gain ratio}(T) = \frac{\text{gain}(T)}{\text{split info}(T)}. \quad (15)$$

ExLMT then uses the best split T^*, which is the split that maximizes the information-gain ratio

$$T^* = \arg\max_i\left(\text{gainratio}(T_i)\right). \quad (16)$$

Step 7) **Refine logistic model:** for each node resulting from the split created at the previous stage, the logistic regression function is refined based only on the subset of $\mathcal{T}$ that reached that node. This refinement means that, as the tree grows, the logistic regression models capture information local to the region of $\mathcal{X}$ that the tree structure above has partitioned. Because of the iterative and additive nature of the LogitBoost algorithm, the refinement is simply running more iterations of a copy of the LogitBoost model of the node above but using only the subset of $\mathcal{T}$ that reached this node.

V. EMPIRICAL EVALUATION

ExLMT was evaluated using the criteria outlined in Section III on 12 standard benchmarking datasets from the well-known UCI machine-learning repository [36]. The primary characteristics of the datasets are given in Table I.

The datasets represent a wide range of classification problem domains. Because all the datasets, with the exception of balance-scale, are based on measured or observed data, they are likely to contain noise. Four of the datasets have only continuous features, and four datasets are purely nominal. The remaining eight datasets have a mixture of continuous and nominal features. Five of the datasets have more than two classes. The remaining seven datasets have a dichotomous class variable. Missing values in the datasets were replaced with the mean value of that feature. Other than this replacement, no other modifications to the datasets were made. A stratified tenfold cross-validation [37] was carried out comparing ExLMT, ExTree, and C4.5. To further improve the reliability of the results, the cross-validation was repeated ten times. A paired t-test was used to test whether the difference between methods on each dataset was significant. Values where $P \leq 0.05$ were considered to be significant. To test whether the difference between the methods across the 12 datasets as a whole was significant, the Wilcoxon rank sign test was used. This test is similar to the well-known paired t-test but does not make the assumption that the data are normally distributed. The null hypothesis for this test is that the two samples were drawn from identical populations, or from symmetric populations with the same mean. It is calculated by finding the differences between each matched pair, then ranking these differences by magnitude. The ranks are then labeled as negative if the difference was negative. The test statistic is then found by taking the smaller of the W_+ or W_-, where

$$W_+ = \sum(\text{Positive Ranks})$$

$$W_- = \sum(\text{Negative Ranks}).$$

The W statistic is then evaluated against standard statistical tables to determine if it is significant.

A. Neural Network Parameters

To evaluate the rule extraction algorithms, neural networks with good predictive accuracy on the benchmark datasets were required. The neural networks were standard MLPs using backpropagation [19]. To avoid overfitting by the neural network, a momentum term was used [19]. The training algorithm minimized the summed squared error with a weight decay term added, again to reduce the chance of overfitting. Given that the output of neural networks is $y = f(\mathbf{x}; \mathbf{w})$. The training examples are a set $\{\mathbf{x}^p, \mathbf{t}^p\}$. The error, E being minimized by backpropagation, is then

$$E = \sum_p \left(t^p - f(\mathbf{x}^p; \mathbf{w})^p\right)^2 + \Phi. \quad (17)$$

Φ is the decay term and is defined as the sum of the weights $\mathbf{w}$

$$\Phi = \frac{1}{2}\sum_i w_i^2 \quad (18)$$

where the sum runs over all the weights and biases.

Table II details the parameters used for the backpropagation algorithm for each dataset. The best parameters were chosen from a small selection of preliminary experiments, but further optimization of the parameters is likely to be possible. Epochs was the maximum number of iterations of the backpropagation algorithm. Validation size was the percentage of $\mathcal{T}$ set aside to be used as a validation set. LR and MR are the learning and momentum rates, respectively. Decay refers to whether a decay term was used in the error function.

B. Results

Table III shows the average percentage accuracy of ExLMT compared to the original neural network, C4.5 and ExTree.

TABLE II
NEURAL NETWORK PARAMETERS

Dataset	Hidden Layers	Epochs	Val. Size	LR	MR	Decay
Balance Scale	10,5	2500	0	0.1	0.9	No
Sonar	20	500	0	0.2	0.3	Yes
Diabetes	20	500	0	0.2	0.3	Yes
Heart	20	500	0	0.2	0.3	Yes
Hepatitis	20	500	0	0.2	0.3	Yes
Housing	2	2000	0	0.2	0.3	No
Labor	25	2500	25	0.1	0.9	Yes
Wine	25	2500	25	0.1	0.9	Yes
Grub Damage	25	2500	25	0.1	0.9	Yes
Zoo	10	2000	0	0.3	0.2	No
Iris	10	2000	0	0.3	0.2	No
Primary Tumor	10	2000	25	0.3	0.2	No

TABLE III
CLASSIFICATION ACCURACY: MLP, C4.5, AND EXTREE AND EXLMT

Dataset	MLP	C4.5	ExTree	ExLMT
Balance Scale	96.00	77.82	81.05 *	93.33 *
Sonar	79.31	73.61	73.69	77.99 *
Diabetes	77.61	73.15	76.03 *	77.1 *
Heart Statlog	83.46	77.02	82.74 *	83.44 *
Hepatitis	85.82	77.02	82.81 *	85.06 *
Housing	88.93	83.12	83.96	85.57 *
Labor	91.57	80.70	85.83 *	90.67 *
Wine	93.65	89.34	90.78	92.32 *
Grub Damage	46	35.96	42.39 *	44.03 *
Zoo	94.69	92.61	92.70	93.7 *
Iris	95.8	94.73	95.00	95.8 *
Primary Tumor	44.28	41.01	40.95	42.48 *

TABLE IV
COMPARISON OF CLASSIFICATION ACCURACY: NUMBER OF TIMES ALGORITHM IN COLUMN OUTPERFORMED ALGORITHM IN ROW

-	MLP	C4.5	ExTree	ExLMT
MLP	-	0	0	0
C4.5	12	-	11	12
ExTree	12	1	-	12
ExLMT	11	0	0	-

TABLE V
PERCENTAGE FIDELITY: C4.5, EXTREE, AND LMT. *t-TEST SHOWED IMPROVEMENT AGAINST C4.5 WAS SIGNIFICANT

Dataset	C4.5	ExTree	ExLMT
Balance Scale	77.48	80.84 *	93.94 *
Sonar	73.44	77.28 *	84.53 *
Diabetes	83.91	92.38 *	98.28 *
Heart Statlog	85.81	92.26 *	98.44 *
Hepatitis	86.98	92.82 *	96.29 *
Housing	85.22	89.72 *	93.04 *
Labor	81.2	90.40 *	96.63 *
Wine	90.56	92.98 *	96.20 *
Grub Damage	53.14	79.25 *	88.18 *
Zoo	94.75	95.95 *	93.70
Iris	97.07	97.00	95.80
Primary Tumor	55.89	70.8 *	84.22 *

TABLE VI
TREE SIZE

Dataset	C4.5	ExTree	ExLMT
Balance Scale	77.82	193.4	18
Sonar	27.9	53.32	7.18
Diabetes	43.4	55.2	3.92
Heart Statlog	34.64	46.68	4.52
Hepatitis	20.12	22.5	2.18
Housing	38.04	93.08	35.76
Labor	7.92	15.7	1.81
Wine	17.35	28.09	3.37
Grub Damage	56.73	31.89	27.37
Zoo	15.7	36.48	1.68
Iris	8.28	22.16	5.56
Primary Tumor	89.9	136.19	21.24

Table IV shows how the different methods compare with each other. Both tree-extraction techniques, ExTree and ExLMT, which extracted trees with higher accuracy than the C4.5-induced decision tree. ExLMT managed to extract decision trees that outperformed both C4.5 and ExTree trees on all 12 datasets. On one dataset (Iris), ExLMT extracted a decision tree with the same classification accuracy as the neural network. On seven of the datasets, ExLMT was within a percentage point of the accuracy of the neural network. Wilcoxon ranks sign tests showed that improvement in accuracy by ExLMT over C4.5 on the 12 datasets overall was significant ($p = 0.0005$). A further Wilcoxon test showed that the improvement of ExLMT over ExTree was also significant ($p = 0.0005$).

Table V shows the average fidelity of ExLMT with the neural network. The fidelity of C4.5 with the neural network was also calculated to give a baseline fidelity measure. Fidelity was calculated as the percentage of the m instances that a classifier, and the original neural network classified the same giving

$$\text{fidelity}(\hat{c}_1, \hat{c}_2) = 1/m \sum_{i}^{m} \text{isEqual}\left(\hat{c}_1(x_i), \hat{c}_2(x_i)\right) \qquad (19)$$

where

$$\text{isEqual}(a, b) = \begin{cases} 1, & a = b \\ 0, & a \neq b. \end{cases}$$

The fidelity of the ExLMT-extracted trees was compared to C4.5-induced trees and trees extracted using ExTree. Wilcoxon rank sign tests showed that the overall difference between ExLMT and both C4.5 and ExTree to be significant ($p = 0.001, p = 0.002$). t-tests showed that the difference on all but two datasets to be significant.

Table VI shows the average size of the tree produced by the three decision-tree algorithms.

VI. DISCUSSION

The results of the empirical evaluation showed that ExLMT-extracted trees have higher accuracy and fidelity than the ExTree. This clearly demonstrates the advantage of the logistic models at the leaf nodes.

Interestingly, the fidelity on the Zoo and Iris datasets was lower with ExLMT than with the C4.5-produced decision tree.

```
plasma <= 127: negative
plasma > 127
|  mass <= 29.9: negative
|  mass > 29.9
|  |  pressure <= 61: positive
|  |  pressure > 61
|  |  |  plasma <= 157
|  |  |  |  age <= 30
|  |  |  |  |  preg <= 0
|  |  |  |  |  |  pressure <= 68: positive
|  |  |  |  |  |  pressure > 68
|  |  |  |  |  |  |  insu <= 135
|  |  |  |  |  |  |  |  mass <= 35.5: negative
|  |  |  |  |  |  |  |  mass > 35.5: positive
|  |  |  |  |  |  |  insu > 135: negative
|  |  |  |  |  pregnant > 0
|  |  |  |  |  |  pregnant <= 2: negative
|  |  |  |  |  |  pregnant > 2
|  |  |  |  |  |  |  pedigree <= 0.332: negative
|  |  |  |  |  |  |  pedigree > 0.332
|  |  |  |  |  |  |  |  plasma <= 144: positive
|  |  |  |  |  |  |  |  plasma > 144: negative
|  |  |  |  age > 30: positive
|  |  |  plasma > 157: positive
```

Fig. 6. C4.5-induced decision tree for the diabetes dataset.

```
plasma <= 139.014612
|  pregnant <= 8.02428:        Model 1
|  pregnant > 8.02428
|  |  pedigree <= 0.464:       Model 2
|  |  pedigree > 0.464:        Model 3
plasma > 139.014612:           Model 4

Model 1: Class 0 = 14.2 + -0.11pregnant +
 -0.05plasma + 0.02pressure
+ -0.14mass * + -3.54pedigree  + -0.02age

Model 2: Class 0 = 23.52 + -0.83pregnant +
-0.11plasma + 0.02pressure + -0.03skin + -0.15mass +
-6.76pedigree + 0.12age

Model 3: Class 0 = 6.24 + -1.52pregnant +
-0.04plasma + 0.24pressure + -0.17skin +
0.01insurance + -0.17mass + -4.31pedigree + 0.09age

Model 4: Class 0 = 15.49 + -0.13pregnant +
-0.08plasma + 0.04pressure + -0.02skin  +
-0.13mass + -2.49pedigree
```

Fig. 7. ExLMT-extracted tree for the diabetes dataset.

However, the ExLMT-extracted tree, for these datasets, was significantly smaller than either the C4.5- or ExTree-produced trees. A possible explanation of this is that the simple logistic regression was a more appropriate model than a tree structure for these simple datasets.

A drawback of the original ExTree algorithm was that it produced trees with a large number of nodes, which reduces the comprehensibility of the model. ExLMT's logistic model leaves are a much more compact representation but are less comprehensible than the simple class assignment at the leaf node as used by C4.5 and ExTree. Although ExLMT produces trees with significantly less nodes than ExTree or even C4.5, whether these trees are more comprehensible overall is subjective.

If we consider two decision trees for the diabetes dataset, one induced using C4.5 (Fig. 6) and the other extracted from the ANN using ExLMT (Fig. 7). The C4.5 tree has 25 nodes, of which 13 are leaf nodes. In contrast, the ExLMT tree had only seven nodes, of which four were leaf nodes with logistic models. As shown previously in Table III, the ExLMT trees outperformed the C4.5 trees on this dataset. Both trees begin with a split on the plasma-glucose concentration level. This indicates that plasma is the most significant attribute. However, C4.5 and ExLMT differ in where the split point is made. C4.5 split at 127, which resulted in two subsets with 437 and 255 instances. ExLMT split at 139, which resulted in two subsets with 515 and 177 instances. The second level of the trees differ significantly. C4.5 chose to label the larger less-than-127 subset as negative for diabetes. Then, preceded to further partition the remaining instances with a 23-node subtree. Conversely, ExLMT did not split the higher than-139 group of instances instead assigning a single logistic model, $\text{Class0} = 15.49 - 0.13\text{pregnant} - 0.08\text{plasma} + 0.04\text{pressure} + -0.02\text{skin} + -0.13\text{mass} + -2.49\text{pedigree}$. This model had slightly higher accuracy and higher fidelity than the subtree and, in the opinion of the authors, is as comprehsible if not more so.

VII. Conclusion

This paper has demonstrated a new method ExLMT, which is for extracting a decision tree from a trained neural network. An empirical evaluation was carried out comparing this new method to ExTree, a method that extracts traditional C4.5-style decision trees, and the C4.5 method that induced trees directly from the dataset. The evaluation was based on 12 well-known datasets from the UCI machine-learning repository. The evaluation showed that the extracted LMTs had significantly higher classification accuracy than either the corresponding C4.5 or ExTree trees. Additionally, on a majority of the datasets, the ExLMT-extracted trees had significantly higher fidelity with the neural networks than trees extracted using ExTree. ExLMT produced much smaller trees than either C4.5 or ExTree but, because of the additional logistic model at the leave nodes, it is unclear if these are easier to comprehend.

Acknowledgment

The authors would like to thank the anonymous referees for their insightful comments which helped to improve this paper.

References

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.
[2] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1996.
[3] P. C. Pendharkar, "A threshold-varying artificial neural network approach for classification and its application to bankruptcy prediction problem," *Comput. Oper. Res.*, vol. 32, no. 10, pp. 2513–2522, 2005.
[4] K. M. Neaupane and S. H. Achet, "Use of backpropagation neural network for landslide monitoring: A case study in the higher Himalaya," *Eng. Geol.*, vol. 74, no. 3/4, pp. 213–226, 2004.
[5] C. M. Ennett, M. Frize, and E. Charette, "Improvement and automation of artificial neural networks to estimate medical outcomes," *Med. Eng. Phys.*, vol. 26, no. 3, pp. 321–328, 2004.
[6] R. Andrews, J. Diederich, and A. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowl.-Based Syst.*, vol. 8, no. 6, pp. 373–389, 1995.

[7] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich, "The truth will come to light: Directions and challenges in extracting the knowledge embedded within trained artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 9, no. 6, pp. 1057–1068, Nov. 1998.

[8] A. Tickle, F. Maire, G. Bologna, R. Andrews, and J. Diederich, "Lessons from past, current issues, and future research directions in extracting knowledge embedded in artificial neural networks," in *Hybrid Neural Systems*, S. Wermter and R. Sun, Eds. New York: Springer-Verlag, 2000.

[9] M. W. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained networks," in *Advances in Neural Information Processing Systems 8*. Denver, CO: MIT Press, 1997, pp. 24–30.

[10] D. Dancey, D. Mclean, and Z. Bandar, "Extracting decision trees from trained artificial neural networks," in *Proc. 17th Int. FLAIRS*, 2004, pp. 515–519.

[11] A. Browne, B. D. Hudson, D. C. Whitley, M. G. Ford, and P. Picton, "Biological data mining with neural networks: Implementation and application of a flexible decision tree extraction algorithm to genomic problem domains," *Neurocomputing*, vol. 57, pp. 275–293, 2004.

[12] J. R. Quinlan, *C4.5 : Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[13] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. New York: Chapman & Hall, 1984.

[14] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*. New York: Ellis Horwood, 1994.

[15] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Mach. Learn.*, vol. 59, no. 1/2, pp. 161–205, May 2005.

[16] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[17] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995.

[18] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.

[19] D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, vol. 1. Cambridge, MA: MIT Press, 1986.

[20] D. W. Hosmer and S. Lemeshow, *Applied Logistic Regression*. New York: Wiley-Interscience, 1989.

[21] L. M. Fu, "Rule learning by searching on adapted nets," in *Proc. 9th Nat. Conf. Artif. Intell.*, 1995, pp. 373–389.

[22] S. H. Huang and H. Xing, "Extract intelligible and concise fuzzy rules from neural networks," *Fuzzy Sets Syst.*, vol. 132, no. 2, pp. 147–167, 2002.

[23] G. G. Towell and J. W. Shavlik, "The extraction of refined rules from knowledge-based neural networks," *Mach. Learn.*, vol. 13, no. 1, pp. 71–101, 1993.

[24] M. W. Craven and J. W. Shavlik, "Learning symbolic rules using artificial neural networks," in *Proc. 10th Nat. Conf. Artif. Intell.*, 1993, pp. 73–80.

[25] K. Saito and S. Nakano, "Medical diagnostic expert system based on pdp model," in *Proc. IEEE Int. Conf. Neural Netw.*, 1998, vol. 1, pp. 255–262.

[26] N. K. Kasabov, "Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems," *Fuzzy Sets Syst.*, vol. 82, no. 2, pp. 135–149, 1996.

[27] D. Nauck and R. Kruse, "A neuro-fuzzy method to learn fuzzy classification rules from data," *Fuzzy Sets Syst.*, vol. 89, no. 3, pp. 277–288, 1997.

[28] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-3, no. 1, pp. 28–44, Jan. 1973.

[29] J. Jang and C. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Trans. Neural Netw.*, vol. 4, no. 1, pp. 156–159, Jan. 1993.

[30] H. C. Anderson, A. Lotfi, and L. C. Westphal, "Comments on 'functional equivalence between radial basis function networks and fuzzy inference systems'," *IEEE Trans. Neural Netw.*, vol. 9, no. 6, pp. 1529–1532, Nov. 1998.

[31] M. W. Craven and J. W. Shavlik, "Using sampling and queries to extract rules from trained neural networks," in *Proc. Int. Conf. Mach. Learn.*, New Brunswick, NJ, 1994, pp. 37–45.

[32] P. Murphy and M. Pazzani, "Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees," in *Proc. 8th Int. Workshop Mach. Learn.*, Evanston, IL, 1991, pp. 183–187.

[33] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*. London, U.K.: Chapman & Hall, 1986.

[34] Z.-H. Zhou, Y. Jiang, and S.-F. Chen, "Extracting symbolic rules from trained neural network ensembles," *AI Commun.*, vol. 16, no. 1, pp. 2–15, 2003.

[35] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting," *Ann. Stat.*, vol. 32, no. 2, pp. 337–374, 2000.

[36] S. Hettich, C. Blake, and C. Merz. (1998). *UCI Repository of Machine Learning Databases*. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[37] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *J. R. Stat. Soc.*, vol. 36, no. 2, pp. 111–147, 1974.

Darren Dancey received the B.Sc. degree (with honors) in computing from Manchester Metropolitan University, Manchester, U.K., in 2001, where he is currently working toward the Ph.D. degree.

His research interests are currently focused on neural networks, rule extraction, and fuzzy systems.

Zuhair A. Bandar received the B.Sc. (Eng.) degree in electrical engineering from Mosul University, Mosul, Iraq, in 1972, the M.Sc. degree in electronics from the University of Kent, Canterbury, U.K., in 1974, and the Ph.D. degree in artificial intelligence and neural networks from Brunel University, Uxbridge, U.K., in 1981.

He is currently a Reader in intelligent systems in the Department of Computing and Mathematics, Manchester Metropolitan University, Manchester, U.K. He is also a founding member and the Managing Director of Convagent Ltd., a company which conducts research and develops applications in the field of conversational agents. His research interests also include the application of artificial-intelligence techniques to psychological profiling from nonverbal behavior. Aspects of this work have been patented.

David McLean received the B.Sc. degree (with honors) in computer science from the University of Leeds, Leeds, U.K., in 1989, and the Ph.D. degree "Generalization in Continuous Data Domains," in neural networks from Manchester Metropolitan University, Manchester, U.K., in 1996.

From 1996 to 1997, he worked for DERA (Malvern) and Thomson Marconi Sonar. In 1997, he took up his current position as Lecturer at Manchester Metropolitan University. He is a member of the Intelligent Systems Group and is a founding member of Convagent Ltd., a company which conducts research and develops applications in the field of conversational agents. His other main interest is in automatic psychological profiling from nonverbal behavior using artificial-intelligence techniques. He is a member of a research group that currently holds a patent for such a system.

Decision Tree Extraction from Trained Neural Networks

Darren Dancey and **Dave McLean** and **Zuhair Bandar**

Intelligent Systems Group
Department of Computing and Mathematics,
Manchester Metropolitan University,
Chester Street, Manchester, M1 5GD.
United Kingdom.
d.dancey@mmu.ac.uk

Abstract

Artificial Neural Networks (ANNs) have proved both a popular and powerful technique for pattern recognition tasks in a number of problem domains. However, the adoption of ANNs in many areas has been impeded, due to their inability to explain how they came to their conclusion, or show in a readily comprehendible form the knowledge they have obtained.

This paper presents an algorithm that addresses these problems. The algorithm achieves this by extracting a Decision Tree, a graphical and easily understood symbolic representation of a decision process, from a trained ANN. The algorithm does not make assumptions about the ANN's architecture or training algorithm; therefore, it can be applied to any type of ANN. The algorithm is empirically compared with Quinlan's C4.5 (a common Decision Tree induction algorithm) using standard benchmark datasets. For most of the datasets used in the evaluation, the new algorithm is shown to extract Decision Trees that have a higher predictive accuracy than those induced using C4.5 directly.

Introduction

The two main approaches to machine learning have been Artificial Neural Networks(ANNs) and symbolic learning algorithms. ANNs characteristically produce models that are capable of generalizing to previously unseen data (prediction). However, ANN's do not explicitly reveal the reasoning behind their decisions. Conversely, symbolic learning methods, do not generalize as well as ANNs, but present the explanation behind their reasoning explicitly. This paper presents a method that extracts a symbolic representation from the knowledge embedded within an ANN. Therefore combining the predictive accuracy of an ANN with the advantage of an explicit explanation provided by a symbolic model.

Artificial Neural Networks

The field of Artificial Neural Networks consists of a large collection of models and techniques originally inspired by biological nervous systems such as the human brain. ANNs are based around a number of individual models of neurons

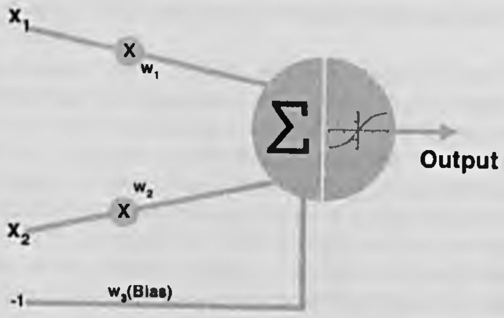


Figure 1: An Artificial Neuron

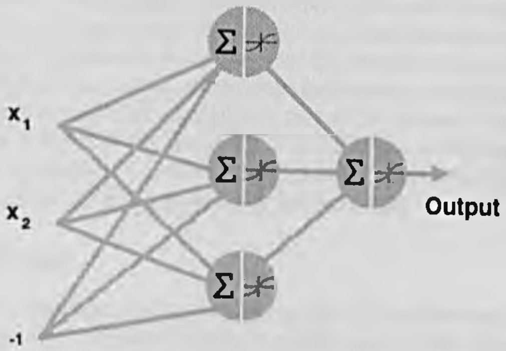


Figure 2: A two layer Multilayer Perceptron

(figure 1) arranged in a network. These artificial neurons accept a number of *weighted* inputs and process these inputs to produce an output. It is the value of these weights that determine the function of the ANN. Using the backpropagation algorithm (Rumelhart, Hinton, & Williams 1986), Multilayer Perceptrons (MLPs) are able to learn non-linear mappings. It is this type of model that will be used throughout this paper. A typical two layer MLP is shown in figure 2.

Decision Trees

Decision Trees are one of the most widely used classifier models(Michie, Spiegelhalter, & Taylor 1994). Decision

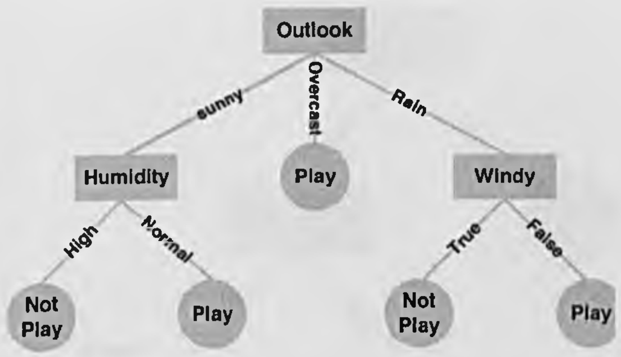


Figure 3: Decision for Quinlan's Play–Not Play Example

Trees are directed acyclic graphs consisting of nodes and connections (edges) that illustrate decision rules. Each non-terminal node has a splitting test associated with it, which splits the data into mutually exclusive subsets. The terminal nodes called leaves represent a classification. A Decision Tree for the Quinlan's classic 'play/not play tennis' example(Quinlan 1986) is shown in figure 3.

To make a decision using a Decision Tree start at the root node and follow the tree down the branches, according to the tests for the instance being classified, until a leaf node representing the class is reached. Although Decision Trees are very simple to understand, the method of creating a decision tree from examples is a nontrivial task, in fact, it has been shown to be NP complete(Hyafil & Rivest 1976).

Rule Extraction From Multilayer Perceptrons

Multilayer Perceptron's (MLP's) greatest weakness is their lack of transparency. Unlike decision trees, which show their reasoning explicitly, MLPs hide their knowledge in the complex interrelationships of their weights. This means that although MLPs often provide excellent models for prediction, they provide no insight into the relationships between input values and output values that the model may have found(Andrews, Diederich, & Tickle 1995). For example, Rothwell(2002) has created an ANN that can classify a persons responses as either deceptive or truthful, using clues in their nonverbal behaviour (eye moments, shrugs etc) but although the ANN has good predictive accuracy it does not reveal the relationships it has found between nonverbal behaviour and deception.

The aim of rule extraction is to reduce the complexity of an ANN into a more easily understood symbolic form. These rules can then be analyzed for trustworthiness for safety critical systems or used to provide insights into the relationships found by the ANN.

There have been two main approaches to extracting rules from trained ANNs decompositional and pedagogical(Craven & Shavlik 1994a). The decompositional approach examines the individual weights of the underlying ANN. This approach is typified by the KT algorithm(Fu 1995). The second approach to rule extraction is the pedagogical approach. This approach is typified by the Trepan algorithm(Craven & Shavlik 1994b). This approach treats the ANN like a 'black box', and uses a symbolic learning algorithm to 'learn' the rules which represent the mapping the ANN has found.

ExTree

ExTree is an algorithm(figure 5) for extracting Decision Trees from trained ANNs. ExTree is an example of the pedagogical approach to rule extraction. ExTree uses Craven's querying and sampling method (Craven & Shavlik 1995), but unlike Craven's Trepan, which uses MofN based splits(Murthy 1995), ExTree uses standard splitting tests like CART and C4.5.

The standard Decision Tree induction algorithms have the limitation that the selection of the splitting test is based on fewer and fewer instances as the tree grows downwards. Therefore, the splitting tests that are near the bottom of the tree are often poorly chosen because they are based on less data. ExTree alleviates this problem by generating new instances then querying the ANN (which acts as an oracle) with the newly created instances. ExTree can then select a splitting test based on the newly created instances as well as the original dataset.

ExTree requires a trained ANN to act as an oracle. In the next section ExTree is applied to trained MLPs but ExTree could be as easily applied to other ANN types such as trained Radial Basis Function networks or even other pattern recognition techniques which are opaque. ExTree does not require the ANN to use a special training algorithm or architecture only that it maps the input space to 1 of K classes. Once a trained ANN is available ExTree proceeds in a similar manner to Decision Tree induction algorithms recursively splitting the tree by finding the best feature to split on.

Split Types

ExTree considers two types of tests: for discrete features ExTree creates a branch for each possible value of the feature, for continuous numeric features a binary split is made with two outcomes $A \leq Z$ and $A > Z$. The threshold value Z is determined by first sorting the set of instances on the value of feature A. For a set with m unique values for feature A there will be $m - 1$ possible split points that could partition the set into two. ExTree chooses a split point halfway between the bounding values.

Split Selection Measure

To determine which one of the possible splits to use, ExTree uses a modification of Information Gain. Information Gain has a bias towards selecting tests with many outcomes. Quinlan(1999) proposed a modification to Information Gain giving Information Gain Ratio. Gain Ratio is determined by dividing the Information Gain by the Information gained solely by splitting the data into the number of outcomes resulting from the test. The information gained by arbitrarily

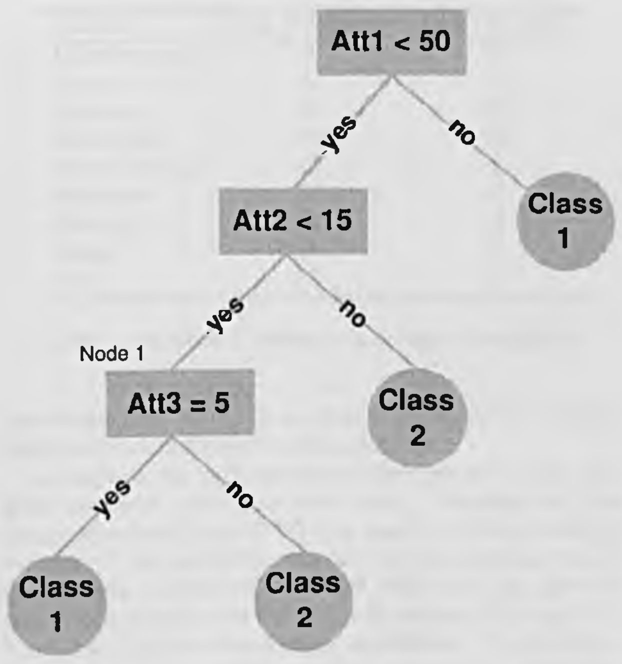


Figure 4: A Decision Tree demonstrating a constraint

```
Extree( dataset $S$, constraints $Const$ )
BEGIN
NewInstances :=
      Create $N$ new instances constrained
by $Const$;
FOR each instance in NewInstances
   Label instance using ANN
$S$ := $S$ + NewInstances;
IF all $S$ belongs to $C_k$ THEN
   label node as leaf $C_k$
9     RETURN
ELSE
   Find Best Split $S*$
   Split the $S$ into subsets $S_1..S_n$according
to $S*$
   FOR each subset$S_i$
   BEGIN
      IF the number of instances in
subset is 0
      THEN mark node as dominating class
of parent
ELSE IF node is a mixture of classes
   Create new Constraint $Const_i$ from $Const$,
   ExTree(oracle,$S_i$,$Const_i$)
END
```

Figure 5: ExTree Algorithm

splitting a set S into n subsets is given by

$$\text{split info}(X) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \times \log_2\left(\frac{|S_i|}{|S|}\right). \quad (1)$$

The gain ratio of test X can thus be calculated as

$$\text{gain ratio}(X) = \frac{\text{gain}(X)}{\text{split info}(X)}. \quad (2)$$

Oracle Querying

As previously stated the advantage of pedagogical approaches such as ExTree is that new instances can be created and classified by the ANN. ExTree is able to create these new instances by maintaining a set of constraints which flows down the tree with the training instances. These constraints specify what conditions an instance must have satisfied to have reached a node as determined by the splitting tests above. For example, new instances created at node 1 in figure 4 must satisfy the constraints: $Att1 < 50$ and $Att2 < 15$. Given these constraints, new instances can created by sampling linearly in the area of input space delimited by the constraints. Currently ExTree makes an extra 100 extra instances at ever split point but ideally the number of extra instances created would be adjusted to suite the dataset.

Pruning

ExTree only stops growing the tree when the set of instances reaching a node all belong to the same class or the instances can not be split any further. For the majority of datasets which contain noise this will lead to overfitting. ExTree uses a form of post-pruning to create smaller trees that should generalize better and be more comprehendible. Before training, 33% of the training data is set-aside as a validation set. ExTree uses the pruning method of subtree replacement. Starting at the leaves and working back towards the root, each subtree is tested using the validation set to determine whether the replacement would be beneficial. If the tree with the replacement has a lower error then the subtree is replaced.

Empirical Evaluation of Extree

ExTree was evaluated using benchmark machine learning datasets from the well known UCI machine learning repository(Blake & Merz 1998). The predictive performance of a trained MLPs and C4.5 induced Decision Trees were compared on number of datasets. Nine datasets from those which the MLP outperformed the C4.5 Decision Tree were randomly chosen to be used in this evaluation. The number of input features and number of classes for each dataset is given in table 1. The Balance scale dataset is an artificial dataset originally generated to model psychological experiments, all the others are real-world datasets originally collected in their respective fields and then donated to the UCI machine learning repository. The Hepatitis, Diabetes, Housing and Heart datasets consist of only numeric features. The Vote dataset consists of purely discrete data. The Labor and Colic datasets have a mixture of numeric and discrete features. The Housing dataset in its original form has a continuous output value, but for these experiments it has been

Dataset	num of features	num of classes
Balance-scale	4	3
Colic	8	2
Diabetes	24	2
Eucalyptus	19	5
Heart-statlog	13	2
Hepatitis	20	2
Housing	14	2
Labor	16	2
Vote	17	2

Table 1: Number of features and classes for datasets

DataSet	Neural(CE)	C4.5[2]	ExTree100
Balance-scale	89.60	77.92	78.60
Colic	82.61	81.52	81.79
Diabetes	75.91	72.53	76.04
Eucalyptus	62.09	60.60	57.20
Heart-statlog	83.33	71.11	78.15
Hepatitis	83.87	70.32	80.65
Housing	87.55	82.41	85.18
Labor	90.35	83.33	85.96
Vote	96.09	93.79	95.63
Mean	84.18	77.69	79.80

Table 2: Percentage of Instances Classified Correctly

transformed into a two class discrete problem of predicting whether the output value is above $20000.

To measure the performance of the algorithm, two standard statistical techniques were used: Stratified ten fold cross-validation(Stone 1974) was used to obtain a reliable measure of the predictive accuracy of the algorithm on the datasets and a Wilcoxon(Wilcoxon 1945) rank sign test was used to test whether the difference in accuracy between ExTree and C4.5 was statistically significant. In all the experiments the same ANN topology was used: a two-layer MLP, with five hidden nodes. All training was done using gradient descent with momentum to minimize a cross entropy (Van Ooyen & Nienhuis 1992) error function. The hidden nodes used the bipolar activation function. The nodes in the output layer used the softmax activation function[1]. Learning rate and momentum were set at 0.01 and 0.9 respectively. Performance of the ANNs could possibly be improved by optimizing the learning rate, momentum and architecture of each ANN to each of the individual datasets but because the purpose of this paper is to illustrate the validity of the ExTree approach to rule extraction this has not been done. To foster generalization, 33% of the training set was set aside to be used as an early stopping validation set. The input features were normalized to have a mean of 0, and a standard deviation of 1 for the ANN as is normal for MLP training(Demuth & Beale 2002; Haykin 1999). For purposes of comparison, predictive classification accuracy results were obtained for these datasets using an implementation of Quinlan's C4.5 algorithm[2]. The C4.5 implementation used the same validation set based pruning technique as ExTree to ensure that any differences in predictive accuracy were not due to the pruning techique used.

Table 2 shows the results obtained using 10-fold cross-validation. As expected the results confirm that ANNs do outperform C4.5. C4.5 does not make maximum use of the information present in the datasets. ExTree performed slightly better on average than C4.5 did. ExTree produced more accurate models on 8 of the 9 datasets. A Wilcoxon rank sum test showed that the difference between the C4.5 and Extree was significant ($p < 0.01$). ExTree appeared to do particularly well on numerically dominated datasets with the largest improvement over C4.5 made on the Heart and Hepatitis datasets which consist of purely numeric features. A likely explanation for this improvement is that if the region of $\mathcal{X}$ where the optimal splitting point lies is sparsely represented in the dataset then C4.5 will be unlikely to find it, whereas ExTree will have sampled extra points in the region and so will be able to produce a more accurate estimate of the optimal splitting point. There are still large differences between many of results obtained by the ANN and ExTree which suggests that there is still much knowledge to be extracted. The ANN outperformed both ExTree and C4.5 by around 10% on the Balance scale dataset. This is almost certainly due to Decision Trees not being able to represent the mapping required by the balance scale dataset[3] This indicates that there will be ANNs that ExTree will be unable to extract sufficiently comprehensible rules from because Decision Trees are simply not powerful enough to represent the function that the ANN has learnt.

Conclusion

A method for extracting Decision Trees from trained Artificial Neural Networks regardless of the ANNs architecture and independent of its learning algorithm has been presented. It was found that the trees produced had better predictive accuracy than trees produced using the C4.5 based learning algorithm for eight of the nine datasets. The results obtained using the ExTree algorithm indicate that querying and sampling the ANN to induce a C4.5 like decision tree is a workable approach for a wide range of problem domains. The results showed that there were still large differences between the predictive accuracy of the underlying ANN and ExTree on some datasets. This suggests there is further knowledge to be extracted from the ANN. An obvious next step to achieving this would be to modify the number of new instances generated at the nodes (currently 100).

[1]The combination of softmax activation functions and a cross entropy error function has the advantage of allowing a probabilistic interpretation of the ANNs output(Ripley 1996)(Bishop 1995).

[2]It should be noted that this was not the 'official' C4.5 released by Quinlan but a C++ work-a-like implementation which shares much of the codebase of the ExTree implementation to ensure a fair comparison.

[3]Because balance scale is an artificial dataset the concept function is actually known: (Feature1 $\times$ Feature2) is equal, greater than or less than (Feature2 $\times$ Feature3)?

Preliminary experiments using an increased number of generated instances on a subset of the datasets used in this paper have indicated an improvement in predictive accuracy.

The results report in the last section used ANNs had not been optimized for the individual datasets. Optimizing the ANN topology would most likely increase the accuracy of the ANN which would in turn increase the accuracy of the Decision Tree extracted by ExTree.

References

Andrews, R.; Diederich, J.; and Tickle, A. B. 1995. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems* 8(6):373–389.

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* Oxford: Oxford University Press.

Blake, C., and Merz, C. 1998. UCI repository of machine learning databases. *University of California, Irvine, Dept. of Information and Computer Sciences.*

Craven, M. W., and Shavlik, J. W. 1994a. Using sampling and queries to extract rules from trained neural networks. In *Eleventh International Conference of Machine Learning.* San Francisco: Morgan Kaufmann.

Craven, M., and Shavlik, J. W. 1994b. Using sampling and queries to extract rules from trained neural networks. In *International Conference on Machine Learning*, 37–45.

Craven, M., and Shavlik, J. 1995. Extracting tree-structured representations of trained networks. In Touretzky; S., D.; Mozer; C., M.; Hasselmo; and E., M., eds., *Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference.*, 24–30.

Demuth, H., and Beale, M. 2002. *Neural Network Toolbox.* Mathsoft.

Fu, L. 1995. Rule learning by searching on adapted nets. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 373–389.

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

Hyafil, L., and Rivest, R. 1976. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters* 5(1):15–17.

Michie, D.; Spiegelhalter, D.; and Taylor, C. 1994. *Machine learning, neural and statistical classification.* Ellis Horwood series in artificial intelligence. New York: Ellis Horwood.

Murthy, S. K. 1995. *On Growing Better Decision Trees from Data.* Ph.D. Dissertation, Johns Hopkins University.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1(1):81–106.

Quinlan, J. 1999. Simplifying decision trees. *International Journal of Human-Computer Studies* 51(2):497–510.

Ripley, B. D. 1996. *Pattern recognition and neural networks.* Cambridge ; New York: Cambridge University Press.

Rothwell, J. 2002. *Artificial Neural Networks For Psychological Profiling Using Multichannels Of Nonverbal Behaviour.* Ph.D. Dissertation, Manchester Metropolitan University.

Rumelhart, D.; Hinton, G.; and Williams, R. 1986. Learning internal representations by error propagation. In Rumelhart, D., and McCleland, J., eds., *Parallel Distributed Processing*, volume 1. Cambridge: MIT Press.

Stone, M. 1974. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society* 36:111–147.

Van Ooyen, A., and Nienhuis, B. 1992. Improving the convergence of the back-propagation algorithm. *Neural Networks* 5(3):465–71.

Wilcoxon, F. 1945. Individual comparisons by ranking methods. *Biometrics* 1:80–83.