



Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

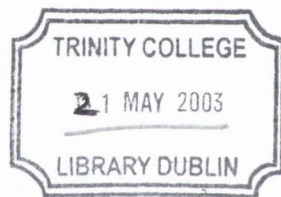
I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Explaining The Output Of Ensembles On A Case By Case Basis

Robert Wall

A thesis submitted to the University of Dublin
for the degree of Doctor in Philosophy

April 2003



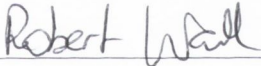
THESIS
7411
7322

Declaration

The work described in this thesis is, except where otherwise stated, entirely that of the author and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Robert Wall



April 2003

Permission to Lend or Copy

I agree that Trinity College Library may lend or copy this thesis upon request.

Signed:

Robert Wall

A handwritten signature in cursive script that reads "Robert Wall". The signature is written in dark ink and is positioned above a horizontal line.

April 2003

Acknowledgements

I would like to acknowledge the help and support of numerous people during the research and writing of this thesis:

- My friends inside and outside of college, in particular, Gabriele, Conor and my girlfriend Deborah.
- My mum, dad, sister and brother for their encouragement.
- Doctors Paul Walsh and Stephen Byrne for the essential work of analysing my results without which this thesis would not have been possible.
- Department of Computer Science, Trinity College for its financial support over the last three years.

Lastly, and most important of all, I would like to thank my supervisor Professor Pádraig Cunningham for his excellent advice and support during my time as a postgraduate — mo mhíle buíochas duit.

Summary

This thesis introduces a novel method for explaining the predictions of ensembles of artificial neural networks on a case by case basis. Current research is primarily directed towards building global model, that is, models that fully describe all possible input conditions and their associated outputs. The alternative case by case approach is referred to as local explanation. *This thesis demonstrates a process for performing local explanation.*

The current global approach is considered ineffective due to an implicit trade off that must take place during its creation. The trade off is between the comprehensibility of the rules and their fidelity to the original ensemble predictions. In a domain with poor coverage, this trade off might be particularly detrimental.

The local explanation approach is accomplished by modelling each of the networks as a rule-set and computing the resulting coverage statistics for each rule, given the data used to train the network. Later, the coverage statistics are used to choose the rule or rules that best describe a previously unseen case under investigation. This approach is based on the premise that ensembles perform an implicit problem space decomposition, with ensemble members specialising in different regions of the problem space. Thus, the explanation of an ensemble prediction involves explaining the ensemble members that best fit the case. *A new metric is introduced, in this thesis, to assess this fit and hence rank the rules in order of importance.*

In order to test the performance and feasibility of the system, the local explanation process and rule ranking techniques were implemented in code. Ensembles with backpropagation neural networks [50] as members were used as the black box to be explained. The explanatory rules were generated using the `c4.5rules` package [47]. Backpropagation ensembles and `c4.5rules` are not the only possibilities, and other methods are also presented in the background chapters.

Two datasets were used during testing and an expert in each domain evaluated the results. Both datasets were from the medical domain. The first dataset involved the prediction of which children displaying signs of bronchiolitis should be admitted overnight to hospital. The second dataset involved the prediction of the Warfarin dosage to be administered to patients based on their previous history of taking the drug and their current symptoms. The bronchiolitis dataset represented poorer coverage of its domain than the Warfarin dataset.

The evaluation demonstrated that a subset of the local explanation's top ranked rules formed a concise and easily understood explanation. Furthermore, in line with expectations, the evaluation demonstrated that the local explanation approach is of particular use in the more poorly covered domain.

Contents

1	Introduction	12
1.1	Contributions of this Thesis	16
1.2	Structure of Thesis	16
2	Neural Networks	19
2.1	Backpropagation Neural Networks	20
2.1.1	Structure	20
2.1.2	Training	23
2.1.3	Execution — Steps 3–5	28
2.1.4	Training — Steps 3–9	28
2.2	Considerations when Training Neural Networks	29
2.2.1	Overfitting	29
2.2.2	Bias & Variance in Neural Networks	31
3	Ensembles	33
3.1	Training Multiple Diverse Learners	34
3.1.1	Bagging	36
3.1.2	Boosting	36
3.1.3	Cross Validation Ensembles	38
3.1.4	Feature Subsets	38
3.2	Combining results	39

3.2.1	Averaging	40
3.2.2	Linear Regression	40
3.2.3	Principal Components Regression	41
3.3	Summary	43
4	Rule Learning Algorithms	44
4.1	Decision Trees	45
4.1.1	C4.5	46
4.1.2	Classification and Regression Trees(CART)	50
4.1.3	Rule Extraction from Decision Trees	55
4.2	Rule Inducing Algorithms	59
4.2.1	CN2	60
4.2.2	FOIL	62
4.2.3	RIPPER	63
4.2.4	SLIPPER	67
4.3	Summary	69
5	Explaining Neural Networks	70
5.1	Strategies	71
5.1.1	Network Decomposition	71
5.1.2	Black Box	75
5.2	Evaluating Rule Quality	81
5.3	Global \vee Local Explanation	82
5.4	Rule Extractions from Ensembles	84
5.5	Summary	86
6	Solution	88
6.1	Building an Ensemble of Rules from an Ensemble of Neural Networks	89

6.2	Rule Coverage Statistics	90
6.2.1	Advanced Rule Coverage Statistics	91
6.2.2	Rule Fit and Ranking	91
6.2.3	Worked Example of Calculating Rule Fit Using Iris Dataset	94
6.3	Rule Simplification	97
7	Implementation	99
7.1	Introduction	99
7.2	Practical Implementation Issues	99
7.2.1	Programming	99
7.2.2	Distributing Work	101
8	Evaluation	104
8.1	Evaluation Process	105
8.2	Bronchiolitis	108
8.2.1	Data	108
8.2.2	Explanations	109
8.3	Warfarin	111
8.3.1	Data	111
8.3.2	Explanations	112
8.4	Summary	114
9	Conclusions & Future Work	122
9.1	Future Work	124
A	Dataset Features	134

List of Figures

2.1	Single layer neural network	21
2.2	Graph of logical XOR function	22
2.3	Multilayer backpropagation neural network	26
2.4	Graph of training and generalisation error	30
4.1	Example decision tree using Iris data	46
4.2	Data that is ill suited for decision tree learning.	47
4.3	Example rules extracted from the decision tree in Figure 4.1.	59
4.4	CN2 algorithm	61
5.1	Shrinking the dimension of a rectagle in rectangular basis function networks	74
5.2	Extracting a rule from the hyperrectangle in a hidden unit	75
6.1	Number line showing unbounded rule	93
6.2	Graph of Iris data in two dimensions	96
7.1	Master/slave architecture	102

List of Tables

2.1	XOR truth table	21
4.1	Errors before and after rule pruning in C4.5	57
8.1	Bronchiolitis dataset structure	108
8.2	Results of 5-fold cross validation performed on bronchiolitis data	109
8.3	Bronchiolitis example evaluated by expert	110
8.4	Rules produced for the example in Table 8.3	117
8.5	Accuracies on test data	118
8.6	Wins, losses and draws for the rules computed by the local explanation method	118
8.7	Analysis of rules generated for bronchiolitis data	118
8.8	Warfarin dataset structure	118
8.9	Results of 5-fold cross validation performed on Warfarin data .	118
8.10	Warfarin example evaluated by expert	119
8.11	Rules produced for the example in Table 8.10	120
8.12	Accuracies on test data	121
8.13	Wins, losses and draws for the rules computed by the local explanation method	121
8.14	Analysis of rules generated for the Warfarin data	121
A.1	Bronchiolitis data features	134

A.2 Warfarin data features 135

Chapter 1

Introduction

The prediction accuracy of neural networks and in particular neural network ensembles has improved, as a result of recent research, to the point that they frequently outperform many traditional systems. Despite this improvement, their adoption as a useful prediction tool in many areas has been slow to non-existent.

The reasons for this poor utilisation in the field of medical diagnosis, although the reasons are similar for other fields, is summarised in this introduction and further expanded throughout the thesis. This introduction also provides an overview of how the research described in this thesis can overcome these difficulties.

Medical datasets provide one of the richest sources of prediction problems ideally suited to prediction techniques. Medical staff could benefit enormously from systems that could assist them in diagnosing and understanding medical problems.

Theoretically, neural networks could be used extensively in assisting in diagnosing a patient's symptoms. Realistically, however, the black box nature of neural networks precludes them from providing this assistance. Doctors are wary of relying on the unqualified diagnoses returned by a computer just

as people in general are wary of trusting any prediction (either from people or computers) without an explanation. In addition, the presentation of a diagnosis in such a definitive form by neural networks could lead the doctor to feel that his/her role is being undermined or even usurped. Providing an explanation of the output might improve confidence in the predictive capabilities of the system thus ensuring greater user acceptance.

In a more general context, the problem of lack of explanation may be even more critical. For instance, use of a neural network in automated safety critical tasks may be impossible, if operation of the network cannot be verified.

To achieve the goal of using neural networks in medical research it is therefore necessary to:

- Take advantage of ensembles of neural networks to provide predictions that are as accurate as possible.
- Provide comprehensible explanations for the user of the output of the ensemble.
- Present explanations to the user, such as a doctor or other professional user, in such a way that the information presented may be used to complement his/her professional experience and judgement and not to replace it.

This thesis addresses each of these issues in turn and provides possible solutions.

This thesis also views the goal of providing coherent explanations for ensemble operations from a somewhat different angle, than most current literature in the area.

Most researchers have focused on producing models of an entire phenomenon. These models will be referred to here as “global models”. The aim of these global models is to produce a comprehensible form that provides appropriate outputs for all possible variations of inputs. This type of model is useful for explaining many types of problems.

For example, a doctor involved in providing “In Vitro Fertilisation” (IVF) is more likely to be a specialist in this area. A global model can aid in the doctor’s understanding of the domain to the fullest extent by summarising all of the conditions under which IVF will be successful or unsuccessful. The global model may also help provide new insights into the domain. Furthermore, the global model may help doctors allocate scarce hospital resources to those cases where they will be of most benefit.

In producing these models, there is an implicit trade-off between comprehensibility and fidelity:

- *Comprehensibility* is an estimation of the understandability of the model.
- *Fidelity* is a measure of how closely the derived model predicts the same outputs as the the original model.

Simplifying a complex model (e.g. by pruning a decision tree) to make it more comprehensible may result in a loss of fidelity, i.e. the derived model’s capacity to explain the original network diminishes.

Global models must balance carefully these two important characteristics.

The approach taken by this thesis is that these global models are not always appropriate. The inherent comprehensibility/fidelity trade-off may result in the wasting of important information. Furthermore, a global model is wasteful in situations where the users do not have the luxury of time to study the model and become experts in the particular domain.

An example of a setting like this would be the busy accident and emergency ward of a hospital. Doctors here are concerned with the quick diagnosis of patient symptoms and less with the minutiae of a problem domain. In this situation, alternatives to a global model may be more useful.

The alternative approach will be referred to as *local explanation*. Local explanation can be seen as on-demand explanation. For each individual prediction made by the ensemble a tailored explanation is produced that best explains it in terms of the input features. Delaying the production of an explanation like this allows the system to use all available data for every prediction. Tailoring the explanation according to the symptoms displayed ensures that the most appropriate explanation is output.

This thesis takes the approach of displaying a number of possible explanations in order to ensure that these local explanations act to complement the doctor's reasoning.

A global model can only provide a single explanation. This explanation may fail to capture all of the details of the prediction. This could be due to the comprehensibility/fidelity trade off encountered in its production. If there is more than one regularity in the data that explains this prediction the global model may also fail to show this.

The local explanation approach of displaying several rules at once, overcomes these difficulties. Because the rules explaining the prediction are not chosen until the last moment no details are lost as a result of comprehensibility/fidelity trade-offs. Also, the approach of displaying several rules at once means that different regularities explaining the prediction that were captured from the diverse ensemble members (that correctly predicted the result) can also be displayed.

The local approach may actually produce many more possible explana-

tions than are to be displayed. To overcome this, the rules are ranked using a novel ranking technique developed as part of this thesis. This technique allows rules to be selected as predictive with increased confidence even if the coverage of that rule on the training data is poor (this problem is often known as the small disjunct problem [35]).

The doctor can now decide on the validity behind the logic in each rule and thus the overall validity of the ensemble prediction itself.

1.1 Contributions of this Thesis

The principal contributions of this thesis to an understanding of explaining ensembles of neural networks are:

- Demonstrates a process for explaining outputs on a case by case basis.
- Demonstrates an evaluation of the case by case basis to explanation that shows that local explanation is of particular use when the data coverage is poor.
- Demonstrates and introduces a new measure for determining the fit of an example to a rule.
- Demonstrates that a subset of rules ranked using the calculated rule fit forms a concise and easily understood explanation.

1.2 Structure of Thesis

The thesis begins with an overview of many of the current machine learning algorithms that are relevant to the goal of explaining neural network ensembles.

Chapter 2 explains backpropagation neural networks which are the networks used in the Implementation section of this thesis due to their proven track record [58, 53] (although other network types could also be used). Chapter 3 presents methods for both creating ensemble predictions and combining them to obtain the best results.

The first half of Chapter 4 covers decision tree algorithms, while the second half concentrates on algorithms that can learn rules directly. The purpose of this chapter is twofold. *Firstly*, the method chosen to explain individual neural networks is to build a more comprehensible learner, e.g. a decision tree, to model the neural network by using data that has been labelled by the network. Any of the methods presented in that chapter can be used to do this. *Secondly*, the method proposed for the explanation of ensembles of neural networks can in fact be generalised to explain an ensemble of rules. The choice of which method to use is left entirely to the modeller. This choice could be guided by personal preference, performance on particular data or availability of existing code or time to implement a method (a modular system could swap one rule learner with another with little trouble later if required).

Chapter 5 looks at existing strategies for explaining individual neural networks. This chapter concludes with a look at what little research has been done to date on the problem of explaining ensembles.

Chapter 6 presents a solution to the problem of explanation, focusing on neural networks but including a note on using pure rule based ensembles.

Chapter 7 outlines a brief description of the solution implementation.

Chapter 8 includes an evaluation of the method in two domains by experts in each domain.

Chapter 9 concludes the thesis, draws conclusions and presents sugges-

tions for future work.

Chapter 2

Neural Networks

Artificial neural networks are developing rapidly in the field of machine learning. Already they have demonstrated [58, 53] that they generalise well for a broad array of both classification and regression problems. The fundamental idea driving the development of neural networks is to model the operation of the neurons in the brain.

Neural network units are interconnected by weights (similar in function to the axon and dendrites in the brain). Firstly, the total signal received by a unit is scaled and propagated to all connected units. Secondly, the signal reaches some output units that trigger a physical reaction. The output from a simpler artificial neural network could similarly be used to control some reaction, e.g. in a robot, but more often the output is simply outputted for use by the user.

Stepping up from their most basic structure, the overall function of these units is to partition the input space into separate regions. The output strength varies across regions and is either directly interpretable in the case of regression problems or can be mapped to a class for classification problems.

This representation of the search space is very powerful. With the addition of more units in the hidden layer of a typical back propagation neural

network, the network can be trained to approximate any continuous function to any degree of accuracy [36]. In practice, however, this is rarely feasible. The data available for training frequently represents only a subset of the entire function. Introducing many more hidden units for training involves tuning many more parameters in the network and these parameters are likely to overfit the available data. By this it is meant that the network will lose its ability to generalise to new instances.

The power of neural networks comes with a heavy cost, however, their operation is quite opaque. It is impossible for even an experienced user to visualise the regions (hyperplanes in the case of backpropagation neural networks) separating the different outputs. Neural network operation has attracted the black box moniker for this opaque behaviour. Chapter 5 presents an overview of research that tries to explain the predictions of neural networks.

Section 2.1 of this chapter will look at backpropagation neural networks. Some other issues that must be taken into account in neural networks are discussed in Section 2.2.

2.1 Backpropagation Neural Networks

2.1.1 Structure

Single Layer Networks

For simple learning tasks, it may be sufficient to use a single layer neural network. That is where input units are connected directly to a layer of output units. Every input neuron is connected to every output neuron. A diagram of such a network is given in figure 2.1

Although sufficient for simple learning tasks, few real world problems can be modeled satisfactorily with a network like this. This problem was

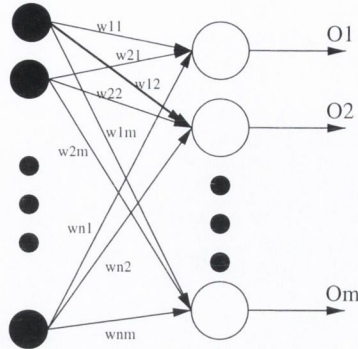


Figure 2.1: Single layer neural network

Table 2.1: XOR truth table

x_1	x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

highlighted dramatically by Minsky & Papert in their 1969 book *Perceptrons* [42]. In this book, they demonstrated that a single layer neural network was incapable of learning even the simple XOR logical function. The problem is that the class outputs of this function are not linearly separable. The truth table for this function is set out in table 2.1 and the problem of separability is easily seen in the diagram in figure 2.2. No single line can be drawn to separate the output classes.

In mathematical terms, this problem can be seen as follows. The response of the output of a single layer neural network is y_{in} . This response is determined by the inputs and the weights connecting these inputs to the outputs.

$$y_{in} = b + \sum_i x_i w_i$$

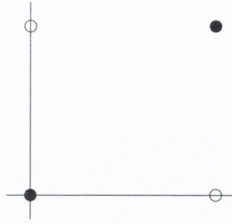


Figure 2.2: Graph of logical XOR function

The decision boundary for this input is determined by the relation:

$$0 = b + \sum_i x_i w_i$$

Depending on the number of inputs in the network, this equation represents a line, plane or hyperplane. In the case of the XOR problem, there are two inputs and the region of positive classes is separated by the region of negative classes by the line:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

For two input problems such as logical AND and OR functions there are many values of b , w_1 and w_2 that will separate these classes. For XOR however, this is not possible.

The answer to this problem was known and lay in using more than a single layer in the network. The problem now was how to update the inter-connecting weights in a multilayer network.

After the initial hype surrounding neural networks, this discovery led to the stagnation of the field for many years.

Multi-Layered Networks

Werbos [66] in 1974 was the first to suggest a solution to the problem of updating weights in a multilayer neural network. This solution was not

highly publicised, however, and as a result neural network research slowed down throughout the 1970's. It wasn't until the mid 1980's when Le Cun [38] independently solved the problem followed closely by Rummelhart et al. [50], who refined and further publicised LeCun's work that backpropagation networks came of age.

The solution to the problem was, that when backpropagating the error in order to update the weights, the first derivative of this activation function should be used to find the direction of the minimum error. This is the direction in which weights should be updated.

Good candidates for activation functions include the sigmoid, bipolar sigmoid and hyperbolic tangent functions. These functions all have the common traits of being continuous along their operating range. A useful trait of these functions is that their first derivative has a simple relationship to the original function output thus decreasing the computational burden during training. In general, any differential function that has an appropriate range for the target values should be acceptable for use in backpropagation training.

Thresholding functions are only useful for categorical outputs.

2.1.2 Training

Certain conditions must be met with regard to the initial setup of the network and the data to be used for training, before training of a neural network can begin.

To train a neural network a number of parameters should be set, these are:

- Number of hidden units
- Learning Rate

- Momentum Rate
- Initial weight values
- Stopping criterion

There are no rules for automatically setting these values to the optimum values and hence tuning these values is somewhat of a black art based on rules of thumb and user experience.

The number of hidden units will determine the complexity of the function that the neural network will learn. The number of units actually used must be carefully controlled. Too few units and the network will be unable to fit the learning data and the bias will be high; too many units and the bias may be low, the training is likely to take significantly longer and the network may overfit the training data.

The learning rate determines the proportion of the weight change as calculated by the learning algorithm that should be added to the original weights.

If the data has many outliers, a lot of noise or even wrong feature values/class outputs, it is preferable not to make dramatic changes of direction in the weight values. Momentum takes care of this by adding a proportion of the previous weight change(s) in addition to the usual proportion specified by the learning rate. Training can proceed reasonably quickly as long as patterns are in the same direction, while still using a smaller learning rate to prevent a large response from any single training pattern.

When initialising a backpropagation neural network, it is preferable to initialise the weights to small random values. In this way, the activation functions are unlikely to reach saturation and cause small weight updates initially that will decrease the speed of learning.

The data should also be adequately prepared before starting to train

a backpropagation neural network. There are two particularly important points here.

Firstly the data should be normalised, this helps even out the effect of data points having different ranges in the activation functions.

Secondly, any symbolic features in the data set should be replaced by a number of units corresponding to the number of possible feature values, with the constraint that only one unit may be active in an example. Alternatively, if the number of possible values of the symbolic variable is large, a gray code may be used to encode the values of the symbolic feature. An appropriate number of units ($\log_2 N$, where N is the number of feature values) should then be added to the network to receive the code.

Finally, if there is a skewed class distribution, the minority class should be copied to make up the difference in numbers and/or the majority class should be reduced in size. This will avoid the network being biased toward any class that may have been seen more often during training.

The backpropagation neural network training algorithm(as described in [25]), is given below. The variables in this algorithm correspond to those marked in Diagram 2.3. The variables z_{in} and y_{in} not marked on the diagram correspond to the unscaled inputs to the hidden and output units respectively. The function $f(\cdot)$ is the activation function, used for scaling the units outputs. α is the learning rate being used.

Step 0: Initialise weights. (Set to small random values).

Step 1: While stopping condition is false, do Steps 2–9.

Step 2: For each training pair do Steps 3–8.

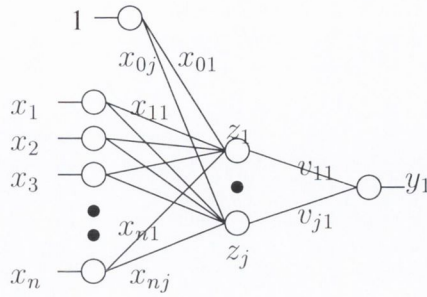


Figure 2.3: Multilayer backpropagation neural network

Feedforward

Step 3: Each input unit ($X_i, i = 1, \dots, n$) receives input signal to all units in the layer above (the hidden units).

Step 4: Each hidden unit ($Z_j, j = 1, \dots, p$) sums its weighted input signals,

$$z_in_j = v_{0j} + \sum_{i=1}^n x_i v_{ij}$$

applies its activation function to compute its output signal,

$$z_j = f(z_in_j)$$

and sends this signal to all units in the layer above (output units).

Step 5: Each output unit ($Y_k, k = 1, \dots, m$) sums its weighted input signals,

$$y_in_k = w_{0k} + \sum_{j=1}^p z_j w_{jk}$$

and applies its activation function to compute its output signal,

$$y_k = f(y_{in_k}).$$

Backpropagation of error

Step 6: Each output unit($Y_k, k = 1, \dots, m$) receives a target pattern corresponding to the input training pattern, computes its error information term,

$$\delta_k = (t_k - y_k)f'(y_{in_k}),$$

calculates its weight correction term(used to update w_{jk} later),

$$\Delta w_{jk} = \alpha \delta_k z_j,$$

calculates its bias correction term(used to update w_{0k} later),

$$\Delta w_{0k} = \alpha \delta_k,$$

and sends δ_k to units in the layer below.

Step 7: Each hidden unit($Z_j, j = 1, \dots, p$) sums its delta inputs(from units in the layer above),

$$\delta_{in_j} = \sum_m^{k=1} \delta_k w_{jk},$$

multiplies by the derivative of its activation function to calculate its error information term,

$$\delta_j = \delta_{in_j} f'(z_{in_j}),$$

calculates its weight corrections term(used to update v_{ij} later),

$$\Delta v_{ij} = \alpha \delta_j x_i,$$

and calculates its bias correction term(use to update v_{0j} later),

$$\Delta v_{0j} = \alpha \delta_j.$$

Update weights and biases:

Step 8: Each output unit($Y_k, k = 1, \dots, m$) updates its bias and weights($j = 0, \dots, p$):

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$$

Each hidden unit($Z_j, j = 1, \dots, p$) updates its bias and weights ($i = 0, \dots, n$):

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}.$$

Step 9: Test stopping condition.

2.1.3 Execution — Steps 3–5

Execution of the network is very fast. It comprises the feedforward section of the training algorithm only. The initial values of the example to be tested are passed to the input units(Step 3). These values are propagated to the first hidden layer and these units apply an activation function(Step 4). Next these hidden outputs are passed to the output layer. The output units also apply an activation function to the outputs(Step 5). Finally, the result can be read by the user.

In the case of a backpropagation neural network having more than a single hidden layer, the outputs of the first hidden layer(Step 3) are passed into further hidden units and are again dealt with like Step 3, until the output units are reached and Step 4 is executed.

2.1.4 Training — Steps 3–9

The training of a backpropagation neural network comprises the execution(Steps 3–5), backpropagation of error(Steps 6–7) and updating of weights(Steps 8–9) and finally a stopping condition is checked(Step 9).

The network first executes the training data. This allows the network to assess the training error. This error is typically measured using the squared difference between the predicted value of the network and the true function value. With an error calculated, the network can begin the process of backpropagating this error in order to adjust the value of the weights in the network.

Adjusting the value of the weights allows the formation of hyperplanes used to divide the input space into regions that predict different output classes.

Two changes often made by practitioners to the basic backpropagation algorithm described above are that, firstly, weight updates are often done in batches, this has the property of smoothing the updates and means the weights make more precise jumps and do not vary greatly during training. The Second change is the inclusion of a momentum parameter. The effect of this parameter has been described already. The revised weight updates now are:

$$\Delta w_{jk}(t+1) = \alpha \delta_k z_j + \mu [w_{jk}(t) - w_{jk}(t-1)]$$

$$\Delta v_{ij}(t+1) = \alpha \delta_j x_i + \mu [v_{ij}(t) - v_{ij}(t-1)]$$

2.2 Considerations when Training Neural Networks

2.2.1 Overfitting

When training the data, it is also necessary to ensure that training is stopped when the network reaches the minimum generalisation error. That is, network training should be terminated at the point where it has reached the

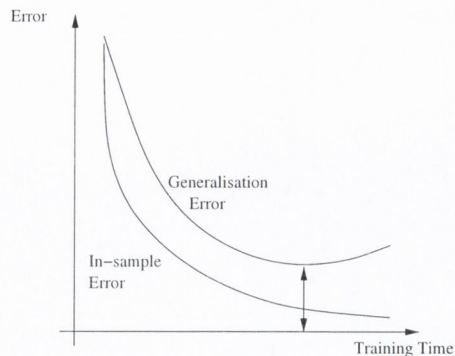


Figure 2.4: Graph of training and generalisation error

optimum point of learning. The optimum point of learning is where the network has reached a trade-off between learning the training examples and retaining the ability to output appropriate values for unseen examples. The point where training should be stopped is shown graphically in Figure 2.4

In the graph in Figure 2.4, it is clear that the ‘in sample’ error (i.e. the training error) continues to fall while the generalisation error falls for a time, until the network reaches a point where it begins to overfit the training data and hence gradually loses its ability to correctly predict the outputs for unseen cases. Checking these errors is straightforward during training, a validation set can be used as an estimate for this generalisation error. Before training commences, the data should be split into two sets, a training set and a validation set. Data that appears in the training set should not appear in the validation set. The network is trained using the training set and the error on this data is computed by executing that network with the data after every epoch (or a preset number of epochs), this is the training error. After computing the error on this training data, the network is then executed on the validation set. The error on the outputs predicted by the network is taken as the generalisation error. Every time this error falls to a new mini-

mum the network should be saved as the point of maximum generalisation. Once this error rises for a preset number of epochs or the training reaches a preset maximum number of epochs, training should be stopped and the saved network should be returned as the “best” network.

2.2.2 Bias & Variance in Neural Networks

The final consideration when training neural networks is to balance the errors due to bias and variance. These two errors are not independent, reducing one will cause an increase in the other. In short, a network fitting the training data closely will have a low bias but a higher variance, while a network with a lower variance will lead to a decrease in the fit of the training data. For optimal learning it is necessary to balance both of these factors.

The bias/variance dilemma was studied in some detail by Geman et al. [30]. In this paper, the authors show in detail the bias/variance decomposition of mean-squared error. This is of particular interest for backpropagation neural networks as this is the most used error function for these networks.

Equation 2.1 shows the breakdown derived by Geman et al. for the mean squared error.

$$E_{\mathcal{D}} \left[(f(\mathbf{x}; \mathcal{D}) - E[y|\mathbf{x}])^2 \right] = (E_{\mathcal{D}}[f(\mathbf{x}; \mathcal{D})] - E[y|\mathbf{x}])^2 + E_{\mathcal{D}} \left[(f(\mathbf{x}; \mathcal{D}) - E_{\mathcal{D}}[f(\mathbf{x}; \mathcal{D})])^2 \right] \quad (2.1)$$

The bias and variance of this equation are averaged over the possible training sets \mathcal{D} . The function $f(\mathbf{x}; \mathcal{D})$ is the prediction of the network on an example \mathbf{x} given the network trained on the set \mathcal{D} . The desired response is y .

The left hand side of this equation is the mean squared error formula, measuring the squared distance from the function f (the neural network)

to the regression $E[y|\mathbf{x}]$. This value is then averaged for the set of possible training sets \mathcal{D} .

On the right the first part of this equation measures the bias. The bias can be thought of as the average distance of a network function f trained on a set of data \mathcal{D} from the true regression for the same input \mathbf{x} . If on average there is a big difference, the bias is said to be large. In general, this will depend on the probability distribution P of the data and how \mathcal{D} reflects this distribution. The same network may be biased in some cases but not in others.

The second part of this equation on the right hand side measures the variance. This measures the average distance of a network f trained on a set of data \mathcal{D} from the average distance of other networks trained on different sets of data.

Variance for a single network can be controlled by combining examples that are nearby in the input space. However, this will typically increase the bias of that network, as details of the regression are lost, e.g. peaks and valleys are blurred. Bias for a single network can be controlled by introducing more hidden units into the network. This has the effect of increasing the complexity of the function that the neural network can learn. It is, however, likely to increase the variance significantly.

Therefore, to achieve a low error, it is necessary to reduce both the bias and the variance components. Typically, reducing one of these will cause an increase in the other. This is commonly known as the bias/variance trade-off.

For more functions displaying the same desirable properties of the mean squared error studied by Geman et al., see Hansen & Heskes [34].

Chapter 3

Ensembles

Recent research in machine learning and, in particular, neural networks has begun to exploit the power of training multiple learners to approximate the same function. These multiple learners, collectively known as an *ensemble*, were first introduced by Hansen & Salamon [32]. By combining the predictions from these learners, it is possible to increase the accuracy of the predictions and in the process reduce the instability of predictions. Instability refers to the phenomenon whereby two neural networks trained to approximate the same function may actually output very different results for new examples, depending on the initial conditions and the training parameters used.

It is interesting to note that although the idea of combining multiple machine learners is relatively recent, the increased accuracy obtainable from a committee of experts is not. As long ago as 1784, the Marquis of Condorcet put forward the theorem, now known as the Condorcet Jury Theorem [18]:

“If each voter has a probability p of being correct and the probability of a majority of voters being correct is M , then $p > 0.5$ implies $M > p$. In the limit M approaches 1, for all $p > 0.5$ as the number of voters approaches infinity.”

A more accessible modern reference for this theorem is Nitzan and Paroush [44]. The first part of this theorem is not controversial, it is easy to show that if a new committee member makes correct decisions more than half of the time and makes different mistakes to the rest of the committee then the performance of the committee will improve with the addition of this new member. However, in practice the second claim is unlikely to be true. A very large committee will not, in practice, be right all of the time. It will not be possible to find new members that will increase the diversity of the committee; instead their voting behaviour will be collinear with some existing members of the committee. Typically the diversity of the ensemble will plateau as will the accuracy of the ensemble at some size between 10 and 50 members.

In order to get the best possible results from an ensemble, it is preferable that a large degree of diversity exists among the members of that ensemble. That is, the members should all be experts in localised areas of the input space. The reason for this is quite simple. If all of the members either predict the same answers or are all experts in roughly the same area of the input space, then the existence of more than one such learner does not supply any more information than a single network alone. Methods of introducing diversity into these learners are outlined in section 3.1.

There are several methods available for combining the results. A few of these have been chosen and are outlined in section 3.2.

3.1 Training Multiple Diverse Learners

When training an ensemble of networks, it is necessary to train each of the networks with the goals of an ensemble in mind. In particular, the bias variance trade-off described in Section 2.2.2 is important. It may make more

sense to think of this trade-off in terms of the error/ambiguity model described first by Krogh & Vedelsby [37].

Krogh & Vedelsby's formula for describing the error/ambiguity of an ensemble is derived in full by Zenobi [67]. In their decomposition they express the bias and variance components of the ensemble error as the weighted ensemble error and the ensemble ambiguity (diversity). Their equation relating these variables is given in Equation (3.1) where E is the ensemble error, \bar{E} is the weighted ensemble error and \bar{A} is the weighted ambiguity measure.

$$E = \bar{E} - \bar{A} \quad (3.1)$$

Instead of expressing the averages for error and ambiguity over different training sets, Krogh & Vedelsby use the weighted averages over the ensemble. If the ensemble is strongly biased the ambiguity will be small, because the networks implement very similar functions and thus agree on inputs even outside the training set. A larger variance between the networks will make the ambiguity higher and in this case the generalisation error will be smaller than the average generalisation error.

There are several methods commonly used to introduce this ambiguity into ensembles. All of these methods work to some degree by skewing the number or type of examples being presented to the individual networks during training. The methods presented below include:

- Section 3.1.1 - Bagging
- Section 3.1.2 - Boosting
- Section 3.1.3 - Cross validation
- Section 3.1.4 - Feature Subsets

By skewing the distribution of examples being presented to each of the networks using one of these methods, the networks training should be concentrated on different examples to other networks in the ensemble. In this way, the ambiguity can be increased between networks as they will make mistakes in different areas of the input space. This is equivalent to adding more members to the Marquis de Condorcet’s committee who have different opinions and hence make different mistakes thus increasing the overall predictive accuracy of the committee.

3.1.1 Bagging

Bagging, short for “bootstrap aggregating”, was introduced by Breiman [10]. The first part of bagging is the process of bootstrapping the input examples. Bootstrapping is a popular statistical technique of sampling a dataset with replacement [10]. When sampling N times from a dataset of size N , approximately 63% of the examples will be chosen at least once. This set of data is then used as the training data for the chosen machine learning prediction algorithm. In the case of neural networks, the remaining data can be used to prevent overfitting during training. In bagging, Breiman suggests using an average as the method for combining the results. Averaging is covered in more detail in section 3.2.1.

3.1.2 Boosting

The original work on boosting was performed by Schapire [51]. The basic idea behind this work is to build a weak learner using the available data and using an equal probability for the selection of each example in the data. Once this learner has been built the probabilities of the examples in the dataset are adjusted so that the more difficult examples are more likely to be chosen.

One of the most popular implementations of this method is that used by Freund & Schapire [26]. This is outlined in detail below:

The initial weights of each example in the training are set as uniform, i.e. $D_1(i) = \frac{1}{N}$, where N is the total number of training examples. The objective now is to minimise the weighted error:

$$\epsilon_t = \sum_i D_t(i) I(h_t(x_i) \neq g_i) \quad (3.2)$$

where I is the indicator function, h_t is the current hypothesis and g_i is the true goal class.

If $\epsilon_t \geq \frac{1}{2}$: goto output with $T = t - 1$.

Otherwise set:

$$\alpha_i = \log \frac{1 - \epsilon_i}{\epsilon_i} \quad (3.3)$$

and finally update the distribution of weights on the training set:

$$D_{t+1}(i) = D_t(i) e^{\frac{-\alpha_t I(h_t(x_i) = g_i)}{Z_t}} \quad (3.4)$$

where Z_t is a normalisation factor (chosen so that D_{t+1} is a distribution).

The final output classifier $H(x)$ is:

$$H(x) = \arg \max_{g \in G} f(x, g) = \arg \max_{g \in G} \left(\sum_{t=1}^T \alpha_t I(h_t(x) = g) \right) \quad (3.5)$$

Diversity is thus built into the models during construction by virtue of the fact that each model focuses its training on different examples.

Boosting does raise an overfitting problem. Particularly noisy data could train some of the models on bad data. These models would provide very inaccurate predictions leading to an overall reduction in the accuracy of the

ensemble. The problem of overfitting using boosting and in particular the AdaBoost method is raised in Maclin & Opitz [39].

3.1.3 Cross Validation Ensembles

K-fold cross validation relies on splitting the available data, D , for training into a total of K sets, D_1, D_2, \dots, D_k . This approach is used by Krogh & Vedelsby in their paper analysing the bias and variance components of neural networks in terms of error and ambiguity [37].

A total of K networks are then trained on these sets, each time using all but one of the sets ($D - D_k$) as training data and using the remaining set (D_k) for testing the generalisation error of the network during training and thus overfitting.

K-fold validation makes good use of the available data and introduces reasonable diversity as long as all of the sets are a fair representation of the data distribution.

3.1.4 Feature Subsets

A recent method used to introduce diversity into ensemble members involves training each member using a different feature mask [68]. Each mask is a boolean string with a length equal to the number of features in the training data. In this string 1's correspond to features that should be used in the training of a network and 0's correspond to features that should be omitted.

The masks are produced using a wrapper method. The wrapper method approach involves estimating the "goodness" of each mask with respect to the bias of the individual network type. A summary of the mask production algorithm as described in [21] is shown below:

1. Generate a random mask and estimate its generalisation error using

cross validation.

2. Start iterating through the mask
3. Flip the current bit of the mask and estimate the generalisation error of the new mask using cross validation
4. If the new mask has a lower error than the previous mask, then accept this bit flip, otherwise reverse the flip and retain the original mask
5. If the end of the mask has not been reached then continue from Step 3
6. If no bit flips have been accepted then output the current mask as optimum, otherwise continue from Step 2

A more complex variation on this algorithm is described by Zenobi [68]. In this variation, Zenobi describes how feature subsets can be found that maximise the total ambiguity in the ensemble.

The alternative to the wrapper approach described above is to simply use random masks. Random masks do help to introduce diversity, but at the cost of higher error. A good wrapper technique should on average outperform random masks.

3.2 Combining results

Once an ensemble of networks is trained, the results from each network must be combined so as to present a single result to the user.

For classification tasks, the simplest method is to simply vote among the networks, with the majority class declared as the predicted class.

The problem is somewhat more difficult for regression tasks. There are a large variety of methods to combine regression results, each with particular

strengths. Three of these methods, averaging, linear regression and principal components regression are detailed below. A brief description of the problems solved by these methods is included for clarity.

3.2.1 Averaging

Averaging results is the method used by Breiman in his paper on bagging [10]. Perrone & Cooper [45] also make reference to this technique which they call the Basic Ensemble Method (“BEM”). Averaging works by assigning equal weights($1/N$, where N is the total number of networks in the ensemble) to the predictions of each neural network in the ensemble.

$$O_j = \sum_{i=0}^N \frac{1}{N} o_i(x_j) \quad (3.6)$$

3.2.2 Linear Regression

Linear regression has been independently studied by several researchers, [45, 33].

Perrone & Cooper refer to their method as the Generalised Ensemble Method(GEM). In this method they minimise the mean squared error in order to set the weights, α_i , with respect to the target function $f(x)$. The formula they suggest for calculating these weights is shown in Equation 3.7.

$$\alpha_i = \frac{\sum_j C_{ij}^{-1}}{\sum_k \sum_j C_{kj}^{-1}} \quad (3.7)$$

In this formula, the C_{ij} defines the correlation matrix:

$$C_{ij} = E[m_i(x)m_j(x)] \quad (3.8)$$

The $m_i(x)$ above are defined as the difference between the true value of the function and the value predicted by network i , i.e. $f(x) - f_i(x)$.

It is important to note that the columns in the C_{ij} matrix should be uncorrelated. Correlation between columns will lead to the matrix being unstable when inverted. To avoid this problem they suggest dropping all but one of any correlated group of columns. This should not result in a great loss of accuracy. The problem of correlated columns is dealt with again in Section 3.2.3.

The weights produced by Perrone & Cooper will be subject to the constraint $\sum_{i=1}^N \alpha_i = 1$. In the more general case of linear regression, this constraint is not applicable.

3.2.3 Principal Components Regression

Principal Components Regression (“PCR*”), was developed by Merz & Pazzani [40]. PCR* was developed with the goal of eliminating the problem of collinearity of networks while still predicting weights that provide a high level of accuracy. Collinearity can lead to very unstable matrices when inverting matrices, an unavoidable step when using any linear regression based method.

Merz & Pazzani identify three methods for reducing the problem of collinearity. They are:

- Train models to have uncorrelated errors by adjusting the bias of the learning algorithm.
- Use a gradient descent technique for setting the weights.
- Use a linear regression method with constraints on the possible weights produced.

None of these solutions provide a full answer to the problem. Models naturally have a certain level of collinearity so even explicit training may not always eliminate this collinearity. Gradient descent techniques are prone to getting stuck in local minima and not finding optimal solutions. Finally, constrained linear regression may also lead to sub optimal weighting solutions.

The basic algorithm of PCR* is set out below:

Input: A^F , the matrix of predictions of the models in F

1. $\mathbf{C} = cov(A^F)$
2. $\mathbf{PC} = PCA(\mathbf{C})$
3. $K = Choose_Cutoff(\mathbf{PC})$
4. $\hat{f}^\beta = \beta_1 \mathbf{PC}_1 + \dots + \beta_K \mathbf{PC}_K$ where $\beta = (\mathbf{PC}_K^T \mathbf{PC}_K)^{-1} \mathbf{y}$
5. $\alpha_i = \sum_{k=1}^K \beta_k \gamma_{k,i}$
6. Return α

In the above algorithm, \mathbf{C} is the covariance matrix for the predictions A^F and \mathbf{PC} is the set of principal components based on the matrix \mathbf{C} .

The search aspect of PCR* is in step 3, where the number of principal components that are going to be used in the determination of the weights is found. The authors of PCR* show how cross validation is one technique that may be used to judge the error on different subsets of the principal components. The optimal number of components to use is taken at the point of minimum error.

In Step 4, linear least squares regression is used to derive an estimate of f using only the K most important principal components that were found in the search stage. Finally Step 5 computes the weights to be used for

combining future predictions from the ensemble of networks by expanding the equation in Step 4 to $PC_K = \gamma_{K,0}\hat{f}_0 + \dots + \gamma_{K,N}\hat{f}_N$ and setting each of the weights α_i to be the coefficients of the original networks(\hat{f}_j).

Although Merz & Pazzani developed PCR* to use all of the networks, stating that “correlation could be handled without eliminating any of the learned models”, it is only fair to refer to other work in the area of eliminating correlation. One such piece of work has been done by Zhou [70] in which he does drop models in order to reduce the correlation and hence instability in assigning weights to ensemble members.

3.3 Summary

The ensembles used in the Evaluation chapter of this thesis were built using bagging to obtain maximum diversity. Bagging is a flexible method for building ensembles providing good, stable performance over a wide variety of datasets. It makes good use of all of the data in building the ensemble and avoids problems of learning noise in the dataset sometimes associated with boosting.

The datasets evaluated were both classification problems and hence a simple majority voting scheme was used to combine the results.

Chapter 4

Rule Learning Algorithms

Rules are arguably one of the simplest representations of knowledge in a machine learning system. Their simple, directly interpretable form has won them a strong following throughout the machine learning fraternity. Decision trees represent a specialised set of rules organised in branches and leaves. When followed in an order determined by an example case, the branches will lead to a single leaf node. This node will have a class associated with it and this is used as the prediction output. Decision trees are readily decomposable to propositional rule sets.

Each rule is typically written in the form of an IF clause which contains one or more terms, the conditions of which must be met in order to “fire” that rule. When a rule is fired, the class associated with the rule, usually written as a THEN clause is either counted as a vote toward an overall class prediction or it is presented directly to the user as the predicted class. An example rule is shown below:

```
IF Sa_02_2 > 91.89  
AND Dehydration=None  
AND Retractions=0  
AND Age_in_Months > 1.87
```

THEN DISCHARGE

Rules such as in the example above, may be generated by a variety of methods. Rule extraction from neural networks is covered in Chapter 5. An introduction to decision trees is covered in section 4.1 and rule extraction from these is covered in section 4.1.3. Algorithms for generating rules directly are covered in section 4.2, these include CN2, FOIL and FOCL.

Tom Mitchell's book Machine Learning [43] is an excellent general introduction to the areas of decision trees and rules.

4.1 Decision Trees

Decision trees comprise a very popular set of machine learning methods. Their popularity is due to their proven accuracy in modelling a wide range of problems [58, 53]. In addition to their good performance, they are easily interpretable by experts involved in the field of study.

Decision trees operate by partitioning input features on axis-parallel boundaries; each such partition is known as a decision node. Each decision node may have one or more child nodes. The child node(s) may be either a decision node or a leaf node. Leaf nodes have a class associated with them and can not have any children. Once a leaf node has been reached when processing a decision tree, processing stops and the class associated with that child is returned as a prediction to the user.

An example decision tree is shown in Figure 4.1. This tree is built using Fishers Iris data from the UCI repository. The Iris-setosa class is linearly separable from the other two, this is reflected by the first split in the tree. This split fully separates this class from the other classes. The remaining two classes are not as easily separated and require several branches.


```

Petal Length <= 1.9 : Iris-setosa (50.0)
Petal Length > 1.9 :
|   Petal Width > 1.7 : Iris-virginica (46.0/1.0)
|   Petal Width <= 1.7 :
|   |   Petal Length > 5.3 : Iris-virginica (2.0)
|   |   Petal Length <= 5.3 :
|   |   |   Petal Length <= 4.9 : Iris-versicolor (48.0/1.0)
|   |   |   Petal Length > 4.9 :
|   |   |   |   Petal Width <= 1.5 : Iris-virginica (2.0)
|   |   |   |   Petal Width > 1.5 : Iris-versicolor (2.0)

```

Figure 4.1: Example decision tree using Iris data

One major disadvantage of trees is in the way that they can only partition features on axis parallel boundaries. If a class is naturally partitioned by a hyperplane that does not lie parallel to axis boundaries, then many decision nodes on several features may be required to accurately represent this decision boundary. This problem can be seen in Figure 4.2. In this figure, the splits made by the decision tree are represented by the broken line. A neural network would have little trouble finding a compact solution to this problem, however, a human user of a system would have great trouble visualising the mathematical solution presented by the network.

4.1.1 C4.5

One of the most popular algorithms used for building decision trees is Quinlan's C4.5. The popularity of this program stems from its freely available implementation (with accompanying source code) and its proven performance over a wide variety of domains.

Building a Tree

Building a tree in C4.5 involves searching each of the features to find the one which provides the most information in predicting one of the classes. Each



Figure 4.2: Data that is ill suited for decision tree learning.

split of a feature is crucial. If the most discriminating features are chosen at each stage in building a decision tree, the tree will tend to be small. A small tree represents a concise concept description for the hypothesis, thus satisfying Occam's razor (i.e. where two or more descriptions exist, the simplest of these should be preferred).

To understand the C4.5 measure of information, it is useful to look at ID3, an algorithm for building decision trees also proposed by Quinlan [46]. In this algorithm, Quinlan used a gain criterion to assess the information content of splitting a set of data. Quinlan himself sums up this criterion with the statement: "The information conveyed by a message depends on its probability and can be measured in bits as minus the logarithm to base 2 of that probability."

The probability of selecting a class, C_j from a set S is

$$\frac{\text{freq}(C_j, S)}{|S|} \quad (4.1)$$

and so the information conveyed by this is

$$-\log_2 \left(\frac{\text{freq}(C_j, S)}{|S|} \right) \text{bits} \quad (4.2)$$

To find the expected information for a message with a class C_j with respect to class membership, sum over all the classes in proportion to their frequencies in S :

$$\text{info}(S) = - \sum_{j=1}^k \frac{\text{freq}(C_j, S)}{|S|} \times \log_2 \left(\frac{\text{freq}(C_j, S)}{|S|} \right) \text{bits} \quad (4.3)$$

When applied to a set of training cases(T), $\text{info}(T)$ measures the average amount of information needed to identify the class of a case in T (also known as the entropy of the set S).

The expected information requirement of the training set T when split according to a criterion X can now be expressed as:

$$\text{info}_X(T) = \sum_{i=1}^n \frac{|T_i|}{|T|} \times \text{info}(T_i) \quad (4.4)$$

Finally the quantity,

$$\text{gain}(X) = \text{info}(T) - \text{info}_X(T) \quad (4.5)$$

measures the information that is gained by partitioning T according to the test X . The *gain criterion* then selects the test that maximises this information gain.

This gain criterion worked quite well, however it had one serious flaw that Quinlan corrected in C4.5. The gain criterion is strongly biased in favour of tests with many outcomes. A worst case scenario would be a feature that comprises only unique values(i.e. every subset of this feature would contain only a single case). In this case, information gain would be maximal as $\text{info}_X(T) = 0$. This was “corrected” in C4.5, by using a gain ratio criterion.

Consider the information content of a message that indicates the information content of a test. By analogy with the definition of $info(S)$, we have:

$$split\ info(X) = - \sum_{i=1}^n \frac{|T_i|}{|T|} \times \log_2 \left(\frac{|T_i|}{|T|} \right) \quad (4.6)$$

This now represents the potential information of this test. By contrast the information gain measures the information relevant to classification. By combining the two using the formula below, it is possible to measure the proportion of useful information generated by the split.

$$gain\ ratio(X) = gain(X)/split\ info(X) \quad (4.7)$$

Pruning a Tree

C4.5 continues to subdivide the data as described in the previous section, selecting the best splits of the data until either a partition consists only of a single class or no test offers an improvement. The problem with this, however, is that the tree may now “overfit” the data.

In order to remedy this situation it is important to prune the generated tree. C4.5 uses post pruning to prune extra structure from the tree. This can take place in two different ways. These are:

- Discard one or more subtrees and replace them with a leaf
- Replace a subtree by one of its branches

Quinlan uses a pessimistic estimate of the tree branch. The error is computed using the resubstitution error (the error of the tree using the training data). This technique allows C4.5 to build a tree using all of the available data. In contrast, cross validation techniques can only build a tree using a

portion of the data and must use the remaining out of sample data for error estimation.

The resubstitution error can be viewed as the number of cases E covered incorrectly from a total N cases covered by a leaf. The probability of the same error being made by the entire population cannot be determined exactly from the resubstitution error, but this probability of error has itself a (posterior) probability distribution that is usually summarised by a pair of confidence limits. For a given confidence level CF therefore, the upper limit of this probability distribution can be found from the confidence limits for the binomial distribution; the upper limit is referred to here as $U_{CF}(E, N)$. C4.5 simply uses this upper limit as the predicted error at a leaf. C4.5 then computes error estimates for all leaves and subtrees by assuming that they were computed from a population with the same size as the training set. A leaf covering N cases during training therefore, would be expected to have at most $N \times U_{CF}(E, N)$ errors. Similarly, the number of predicted errors of a subtree is the the sum of the errors of its branches.

C4.5 traverses the tree backwards. At each subtree it tests if a lower error rate is achievable if the subtree was replaced by either a leaf or one of its branches. A replacement is made if an error reduction is possible. This continues until no further replacements are possible without increasing the estimated error of the tree.

4.1.2 Classification and Regression Trees(CART)

Classification and Regression Trees, better known as CART, described in Breiman et al's book of the same name [11] is one of the first implementations of decision trees. Together with C4.5, CART is one of the most important references on the subject of decision trees.

Building a Tree

Building a tree with CART begins with the generation of a set of questions, Q . These questions will form the basis for the possible splits of nodes. For symbolic features, these questions will be of the form $q_i \in [b_1, \dots, b_n]$, where i denotes the feature. For numerical or ordered features, the possible questions are of the form $q_i \leq c$. Each of these questions defines a possible split in the data, i.e. all examples in the data will fall on one side or the other of a question. Only one split is made at every node in the decision tree.

In order to decide the best split at any node in the tree, an impurity criterion is used. Impurity refers to the proportion of examples that fall inside a node on the tree. The criterion for an impurity function, $i(t)$, in CART is that ϕ is a non negative function of the probabilities $p(1-t), \dots, p(n-t)$ with the following properties:

$$\phi\left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}\right) = \text{maximum} \quad (4.8)$$

$$\phi(1, 0, \dots, 0) = \phi(0, 1, \dots, 0) = \dots = \phi(0, 0, \dots, 1) = 0 \quad (4.9)$$

To actually decide on the best split it is necessary to choose the split that most reduces an impurity measure(i.e. brings the tree closer to the point where the node almost entirely comprises a single class). Specific impurity measures used by CART are shown later.

This difference in impurity at any node can be written as:

$$\Delta i(s, t) = i(t) - p_R i(t_R) - p_L i(t_L) \quad (4.10)$$

where $i(t)$ is the parent node impurity and p_R and p_L are the new probability estimates of the number of examples that will be classified into the new right and left hand nodes respectively. $i(t_R)$ and $i(t_L)$ are the new impurities at

the right and left hand nodes respectively. The split that maximises the difference in impurity will be chosen as the question in the next decision node of the tree.

In their book, Breiman et al. describe several possible functions that could be used to measure the “goodness” of a split. The first of these is an entropy based impurity measure. This is a simple and well understood function that exhibits the desirable properties for measuring impurity.

$$i(t) = - \sum_j p(j|t) \log p(j|t) \quad (4.11)$$

Two other functions described for determining the best split include the Gini function for measuring node impurity and the Twoing rule.

The Gini impurity measure, assesses node impurity, not using the plurality rule (the most abundant class) but instead assigns an object to the class i with a probability $p(i|t)$. The probability of this object actually being class j is therefore $p(j|t)$. The estimated probability of misclassification under this rule is therefore:

$$i(t) = \sum_{i \neq j} p(i|t)p(j|t) \quad (4.12)$$

Unlike the Gini criterion for determining the best split of a node, the Twoing rule does not operate on an overall measure of impurity $i(t)$, and hence finding an overall tree impurity $I(t)$, is not possible. This is not considered a problem as a splitting criterion should be judged primarily on how it performs during tree construction. The p_L and p_R are the proportion of examples from the parent node t reaching the left, t_L , and right, t_R respectively.

$$\Phi(s, t) = \frac{p_L p_R}{4} \left[\sum_j \left| p(j|t_L) - p(j|t_R) \right| \right]^2 \quad (4.13)$$

The stopping criteria used by CART is a simple threshold value:

$$\max_{s \in S} \Delta I(s, t) < \beta \quad (4.14)$$

When the change in impurity for a node fails to exceed a threshold β , that node is no longer split. When this condition is reached for all terminal nodes, the tree growing phase is completed.

Estimating Error

The simplest method used to estimate the error of a CART tree is to calculate the resubstitution estimate using the probability of misclassification. To understand this, it is first important to note how CART assigns a class label to a leaf node (i.e. any example reaching this leaf node will be assigned this class as a prediction). The class assigned is simply the class that appeared most often from the original training data in that leaf node, i.e. the class j for which $p(j|t)$ is greatest.

The resubstitution error is then the error produced when running the training data through the tree. This error may be easily calculated for a particular node by summing up all the probabilities of finding each of the remaining classes not assigned by the assignment rule $j(t)$:

$$\sum_{j \neq j(t)} p(j|t) \quad (4.15)$$

or more simply:

$$r(t) = 1 - \max_j p(j|t) \quad (4.16)$$

However, this is not a completely satisfactory metric for estimating the error. It tends to be overly optimistic when computing error, in particular if the tree has overfitted to the data. A more precise method of calculating error is to separate the available data into two sets, a training set and a test

set. Once the tree has been built using the training set, the error on the tree is estimated by filtering the test set through the tree and calculating probabilities of misclassification at each terminal node. These probabilities are then summed as in the simple resubstitution case described above.

Where insufficient data is available to sacrifice some data as a test set, k -fold cross validation may be used. K -fold cross validation involves splitting the data L into k sets, $\{L_1, \dots, L_k\}$, and training k trees leaving out each one of the sets of data each time to use for estimating the tree error. Like the test set case, the set of data omitted from training is used to calculate error estimates for each terminal tree leaf. An overall error estimate is then found by finding the average of these k error estimates.

All of the above error estimates can be modified easily to include a measure of the cost of misclassification. For the simple resubstitution error case, the misclassification cost is:

$$\sum_j C(i|j)p(j|t) \tag{4.17}$$

where $C(i|j)$ is some function that measures the cost of classifying an example with true class label j as i . Using this costing analysis, different weights can be assigned to different misclassifications, thus perhaps biasing a tree towards making fewer expensive mistakes.

Pruning a Tree

The simple threshold stopping the tree described in the previous section proved unsatisfactory. A small value of β resulted in overly complex trees. Although they had small error estimates, this was due to overfitting of the training data, i.e. they had a low bias, but a high variance giving poor performance on unseen test data. As with neural networks, an optimum

generalisation performance must be found. Increasing the value of β failed to generate trees that were substantially better in performance.

Instead of trying to stop tree growth at an optimum point, Breiman et al. implemented a post pruning strategy in CART. The tree was initially grown to be very complex and then nodes and branches of nodes were removed until an optimum tree structure remained (relative to the original tree).

The basic form of pruning used by CART uses minimal cost-complexity as a measure of pruned tree performance.

In this pruning, the cost-complexity measure $R_\alpha(T)$ is defined as:

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}| \quad (4.18)$$

where \tilde{T} is defined as the number of leaf nodes in the tree and $\alpha \geq 0$ is a real value called the cost complexity parameter.

For each value of α it is possible to find a subtree $T(\alpha) \leq T_{max}$ which minimises $R_\alpha(T)$, i.e.,

$$R_\alpha(T(\alpha)) = \min_{T \leq T_{max}} R_\alpha(T) \quad (4.19)$$

It is now possible to find different measures of α that will give more pruned subtrees, $T_1 > T_2 > \dots > t_1$. The problem now is to choose the best of these subtrees. This is done by estimating the error on the sub trees using one of the methods of assessing error described in the previous section, e.g. resubstitution error, cross validation, etc.

4.1.3 Rule Extraction from Decision Trees

The method of rule extraction from decision trees described here is that described by Quinlan [47] for C4.5. It could however be used for any decision tree.

Extracting Rules from a Tree

The process of rule extraction is very simple due to the nature of the decision tree. Individual rules are extracted from an unpruned decision tree by following the edges of the decision tree from the root node to each leaf node. Every decision node becomes another term in the rule clause while the leaf node becomes the predicted class for that rule (i.e. the THEN clause of the rule).

Pruning Extracted Rules

There are two methods by which the extracted rules may be pruned:

The first method is that the number of terms in the rule may be reduced. This may be done when removing a term in a rule does not significantly increase the number of errors made by that rule on the training set. In C4.5 a greedy search is performed on the terms of a rule clause. The cost associated with removing each one of the rule terms is calculated and if this cost does not exceed the original upper limit of the rule then it is removed. Using the same notation as Quinlan, the upper error rate of the original rule is expressed as $U_{CF}(E, N)$ where E is the number of cases covered erroneously by the rule and N is the total number of cases covered.

So for a rule R , before removing condition X , R covers Y_1 cases correctly and E_1 cases incorrectly. After removing condition X it now covers not only the original cases, but also a number of extra cases. These extra cases covered may include those of the same class as the original rule and those of incorrect classes. These extra cases are known as Y_2 and E_2 respectively. These errors are set out in table 4.1.

The original pessimistic error rate of this rule is therefore $U_{CF}(Y_1, Y_1 + E_1)$. After removing condition X of this rule, however, the error rate may be

Table 4.1: Errors before and after rule pruning in C4.5

	Class C	Other Classes
Satisfies condition X	Y_1	E_1
Does not satisfy condition X	Y_2	E_2

rewritten as $U_{CF}(Y_1 + Y_2, Y_1 + Y_2 + E_1 + E_2)$, taking into account the extra cases covered both correctly and incorrectly.

Conditions are then removed in a greedy fashion (i.e. the condition with the least error below the original rule error rate is removed first) and these pessimistic error rates are recomputed after every removal. This continues until as many conditions as possible have been removed.

The second method of pruning the extracted rules is to actually drop entire rules from the ruleset. In C4.5 rule utility is measured using a minimum description length (“MDL”)[49] approach. In MDL, the hypothesis which requires the minimum number of bits to transfer its encoded message and any exceptions is preferred above the others.

In sending the hypothesis, all terms in the rule clause must be sent, but since they may be sent in any order, the number of bits required to send this information is reduced by $\log_2(x!)$, where x is the number of terms in the rules.

Exceptions are then encoded by specifying which of those examples that are covered are false positives and which of those examples not covered are false negatives. Thus the number of bits required for this encoding is simply:

$$\log_2 \left(\binom{r}{fp} \right) + \log_2 \left(\binom{n-r}{fn} \right) \quad (4.20)$$

The first term in equation 4.20 is the bits needed to transfer the false positives(fp) while the second term indicates the number of bits required to transfer the false negatives(fn) from the total number of bits n .

The total number of bits required to encode this theory is therefore the sum of the bits to encode the theory (i.e. the rule terms) plus the number of bits required to encode the exceptions. In practice however, Quinlan reduced this amount slightly after experiments demonstrated that in practice the number of bits was frequently overestimated. Therefore the true number of bits computed is as set out in equation 4.21.

$$\text{ExceptionBits} + W \times \text{TheoryBits} \quad (4.21)$$

where W is a constant value between 0 and 1.

Unlike the pruning of rule terms, pruning of entire rules does not proceed using a greedy hill climbing search. Instead, if the number of rules is small all possible subsets are considered and with larger numbers of rules, a simulated annealing approach is used. In the case of simulated annealing, the system repeatedly picks a rule and adds it to the subset(S) if it is not already there and removes it otherwise. If, as a result of the action, the change in bits(ΔB) is positive then the change to S is accepted with probability $e^{-\frac{\Delta B}{K}}$. K is a synthetic temperature whose value is reduced during the course of execution and hence the probability of the change being accepted is also reduced.

A consequence that is important to be aware of, after rule pruning, is the possibility that more than one rule may match a new test example. It is important to have a strategy in place to deal with this situation. The simple strategy used by C4.5 to resolve conflicts is to order the rules by the number of examples that they cover in the training set. The first rule to match an unseen example is therefore taken to be the prediction for that case.

The second consequence of rule pruning is that no rules may match an unseen example. C4.5 approaches this problem by setting aside a default class. The default class is the class that covers the most uncovered training

examples after rule pruning. An unclassified example is predicted to be the default class, if no rule matches.

Figure 4.3 shows an example of pruned rules extracted from the decision tree shown in Figure 4.1.

```
Rule 1:
    Petal Length <= 1.9
    -> class Iris-setosa [97.3%]

Rule 4:
    Petal Length > 1.9
    Petal Length <= 5.3
    Petal Width <= 1.7
    -> class Iris-versicolor [90.4%]

Rule 6:
    Petal Width > 1.7
    -> class Iris-virginica [94.4%]

Rule 3:
    Petal Length > 4.9
    -> class Iris-virginica [91.8%]

Default class: Iris-setosa
```

Figure 4.3: Example rules extracted from the decision tree in Figure 4.1.

4.2 Rule Inducing Algorithms

The following sections describe common rule induction algorithms. Unlike rule extraction from neural networks or even decision trees, these algorithms are designed to output rules directly. The algorithms described here represent some of the most prominent in the area.

4.2.1 CN2

The CN2 algorithm for rule induction was introduced by Clark & Niblett [13]. This algorithm builds closely on the previous work by Michalski's AQ algorithm [41].

CN2 works by performing a beam search across the possible attributes. A beam search can be thought of as a number of parallel hill climbing searches. Or alternatively, may be thought of as a breadth first search where only the most promising subsequent nodes are expanded. Once a search has reached a point where it cannot expand any more nodes, the algorithm returns the best complex (rule clause) found. The CN2 algorithm is outlined in Figure 4.4.

There are two important heuristics used in the search for rules. These are:

- Assess the quality of the current complex
- Assess the significance of the current complex

To assess the quality of the current complex, the CN2 algorithm uses an entropy based measure. The set of examples E' that are covered by the complex (i.e. those examples that are satisfied by the complex selectors) are found and the probability distribution $P = (p_1, \dots, p_n)$ of the classes of these examples is then computed. The entropy of these examples can then be computed using the formula in equation 4.22.

$$Entropy = - \sum_i p_i \log_2(p_i) \quad (4.22)$$

Entropy is the favoured measurement of rule quality as it distinguishes probability distributions that are more easily specialised. For instance, given

1. Search for the best complex using the current training set E .
 - (a) While the possible set of complexes is not empty:
 - i. Create a new set of possible complexes by intersecting the current best complexes with the set of all possible selectors, removing any redundant and unchanged complexes.
 - ii. Test the quality of every new complex using Equation 4.22 with respect to the set of training examples E .
 - iii. Each complex that passes the quality test should be tested using Equation 4.23 to find the best complex found.
 - iv. Remove the worst complexes from the total set and continue from i.
 - (b) Return the best complex found.
2. If a complex is found:
 - (a) Remove the examples E' from the set of training examples E that are covered by the complex.
 - (b) Assign the most common class C in the set E' as the output for this complex.
 - (c) Add this rule to rule list.
 - (d) Continue from Step 1.
3. Return the completed rule list to the user.

Figure 4.4: CN2 algorithm

the two distributions $P_1 = (0.7, 0.1, 0.1, 0.1)$ and $P_2 = (0.7, 0.3, 0, 0)$, an entropy measurement will select the latter whereas a simpler maximum correct may not. This is desirable because if the majority class is removed, the distributions will become $P_1 = (0, 0.33, 0.33, 0.33)$ and $P_2 = (0, 1, 0, 0)$ demonstrating how much simpler it is to specialise the second distribution to a definition describing a single class only.

The second heuristic used in the search for rules involves testing the significance of the current complex. This is done to ensure that the rule

complex under consideration is a genuine regularity in the data and not merely one that has occurred as a result of noise in the data. The formula used for computing this significance is the likelihood ratio statistic:

$$2 \sum_{i=1}^n f_i \log \left(\frac{f_i}{e_i} \right) \quad (4.23)$$

where $F = \{f_1, \dots, f_n\}$ is the observed frequency distribution satisfying a given complex and $E = \{e_1, \dots, e_n\}$ is the expected distribution of the same number of examples under the assumption that the complex selects examples randomly. The lower the score the more likely that this complex was formed by chance.

4.2.2 FOIL

FOIL is a first order rule learner proposed by Quinlan [48]. First order rules are commonly known as Horn clauses. First order rules differ from the propositional rules created by algorithms such as CN2(see Section 4.2.1), in that they may include variables. Variables are properties of features that may be attached to any example containing that feature. Propositional rules on the other hand must have precise values for every feature of every example.

The advantage of learning rules comprising Horn clauses is that these rules may be inputted directly into rule based languages such as PROLOG.

The FOIL algorithm is actually very similar in structure to the CN2 algorithm described in Section 4.2.1. The outer loop of the CN2 algorithm is very similar to the outer loop of the FOIL algorithm, training continues until the performance of the next rule learned is below some threshold value. In FOIL the inner loop of CN2 is effectively extended to deal with the production of first order rules.

The main differences between FOIL and the previous algorithm lie in the

method of generating candidate specialisations and in the gain criteria used to assess the goodness of new hypotheses.

To generate possible specialisations of a rule, FOIL employs one of two methods. Firstly, it may add any of the possible predicates, so long as the variables in the predicates already exist in the rule. The second method is to check for equality between the values of two variables already existing in the rule.

In FOIL, the objective when adding new variables to literals is to cover as many positive examples as possible. To maximise this goal, the information theory method of minimum description length is used. The number of bits required to encode the original rule and the augmented rule to be tested are computed and if the new rule reduces the number of bits significantly then the change is accepted. The precise formula used is in Equation 4.24.

$$Foil_Gain(L, R) = t \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right) \quad (4.24)$$

In this equation, p_0 is the number of positive examples covered by the original rule and n_0 is the number of negative examples covered by the original rule. Similarly, p_1 and n_1 are the number of positive and negative examples covered by the new rule respectively. Finally, t is the number of positive examples that are still covered by the new rule.

4.2.3 RIPPER

RIPPER [16] is a rule learning algorithm that is based on the work of Quinlan's FOIL [48], Brunk & Pazzani's "Reduced Error Pruning" (REP) [12] and Fürnkranz & Widmer's "Incremental Reduced Error Pruning" (IREP) [29].

The basic algorithm of RIPPER is similar to that of FOIL. In particular

the GrowRule procedure is a propositional version of the FOIL algorithm. It works by adding conditions of the form $A_n = v$ or $A_c \leq \theta$ or $A_c \geq \theta$, where A_n is a symbolic attribute and v is a legal value and A_c is a numeric attribute and θ is a value for A_c that occurs in the data. GrowRule continues to add propositions that maximise FOIL's information gain criterion until no negative examples in the growing set are covered by the rule.

In REP, the training data is split into two sets, a growing set and a pruning set. A ruleset is grown to overfit the training data. These rules are post pruned using the pruning set by applying one or more pruning operators to any single rule. A hill climbing technique is used to select the next operator to apply. Simplification is complete when applying any operator would increase the error on the pruning set. REP's major shortcoming is its complexity. Cohen [15] showed that given sufficiently noisy data, REP required $O(n^4)$ time. Even the initial overfitting of rules required $O(n^2)$ time to complete.

The most successful response to the inefficiency of REP was the algorithm IREP. IREP is competitive with REP in terms of error rates and was significantly faster than REP. IREP builds a ruleset in a greedy fashion. Like REP the full training set is split into a training set and a pruning set. However, after a rule is found and pruned, it removes all positive and negative examples from the full training set before splitting it again. This continues until either there are no positive examples remaining or the rule found by IREP has an unacceptably large error rate. It is on IREP that RIPPER bases its error pruning technique.

In IREP pruning, the deletion that maximises the function

$$v(\text{Rule}, \text{PrunePos}, \text{PruneNeg}) \equiv \frac{P + (N - n)}{P + N} \quad (4.25)$$

is chosen, where P and N are the total number of positive examples in

PrunePos and PruneNeg respectively and p and n are the total number of positive and negative examples covered by the rule in PrunePos and PruneNeg respectively. This process is repeated until no deletion improves the value of v .

The ability to handle multi class problems is included in the RIPPER implementation of IREP. This is accomplished by ordering the examples of each class in increasing order of prevalence, i.e. C_1, \dots, C_k , where C_1 is the least prevalent class and C_k is the most prevalent class. Repeated calls to GrowRule are now made using the current least prevalent class with remaining examples uncovered as the positive examples and all other classes are considered to be negative. This continues until only the most prevalent class C_k remains. This class becomes the default class.

RIPPER also extended the IREP algorithm to handle missing attributes. Any rule involving a test on an attribute A are deemed to have failed if the value for that attribute is missing in a given example. This behaviour was introduced to separate the positive examples using only tests that were known to succeed.

Three improvements were also made on the IREP algorithm:

1. An alternative metric for assessing the value of rules in the pruning phase of IREP.
2. A new heuristic for determining when to stop adding rules to a rule set.
3. A postpass that “optimises” a rule set in an attempt to more closely approximate conventional (i.e. non incremental) reduced error pruning.

Cohen found that occasional failures of IREP to converge as the number of examples increased could be traced to the metric used to guide pruning

shown in equation 4.25. The original pruning metric would prefer a rule R_1 that covered $p_1 = 1000$ positive examples and $n_1 = 1000$ negative examples to a rule R_2 that covered $p_2 = 1000$ positive examples and $n_2 = 1$ negative examples even though the rule R_2 is significantly more predictive. Instead of the original pruning metric, Cohen replaced it with:

$$v^*(Rule, PrunePos, PruneNeg) \equiv \frac{p - n}{p + n} \quad (4.26)$$

where p and n are the number of positive and negative examples of the pruning set covered by the rule.

Cohen reports [16] that IREP seems to be particularly sensitive to the small disjuncts problem [35]. Small rules that cover few examples may have high error rates causing IREP to stop prematurely. To overcome this problem RIPPER uses Minimum Description Length(MDL) theory to assess the length of the ruleset and the examples. No rules are added once this description becomes a constant “ d ” bits longer than the smallest description length.

The final improvement made to the IREP algorithm in RIPPER involves an optimisation of the global ruleset. Each rule is optimised in the order which the rules were constructed. Two alternative rules are constructed. The first of these is known as the replacement rule. A rule is grown and then pruned with the objective of minimising the error of the entire rule set. The second rule constructed is formed by greedily adding conditions to the original rule, this is known as the revised rule. Finally a decision is made whether to retain the original rule, or replace it with the replacement or the revised rules. MDL is used to make this decision. Each of the alternatives is inserted into the ruleset and rules that are increasing the total length of the ruleset and examples are removed. Once a final decision has been made, the modified IREP algorithm is reapplied to learn new rules for any positive

examples that may be left uncovered after being removed during pruning.

4.2.4 SLIPPER

SLIPPER is a rule learning algorithm introduced by Cohen & Singer [17]. Unlike the many other rule learners including RIPPER and CN2, where covered examples are removed from the set of training examples, SLIPPER uses a boosting like approach to change the distribution of the examples, so more emphasis is placed on those examples misclassified in earlier rounds. Boosting is covered in the context of ensembles in Section 3.1.2.

Every rule in SLIPPER has a fixed confidence value associated with it. The sign of these confidence values determines the class of an example classified by a rule. Rules not covering an example, output a confidence of zero. Therefore, to classify an example using the strong hypothesis, it is only necessary to sum the confidence values of covering rules and return the sign.

The confidence values are computed using the formula:

$$\hat{C}_R = \frac{1}{2} \ln \left(\frac{W_+ + \frac{1}{2n}}{W_- + \frac{1}{2n}} \right) \quad (4.27)$$

where $W_+ = \sum_{x_i \in R: y_i = +1} = +1$ and $W_- = \sum_{x_i \in R: y_i = -1} = -1$. To prevent rules covering few examples and having $W_- = 0$ leading to impractically large values, the confidence is “smoothed” by adding $\frac{1}{2n}$ to both W_+ and W_- .

When growing a rule, SLIPPER restricts itself to positively correlated rules, hence the objective function that is attempted to be maximised is:

$$\tilde{Z} = \sqrt{W_+} - \sqrt{W_-} \quad (4.28)$$

Once a rule is grown, i.e. no negative examples in the `GrowSet` remain uncovered, there is a danger of that rule overfitting the data. The rule is therefore pruned immediately after training by minimising the equation:

$$(1 - V_- - V_+) + V_+ \exp^{-\hat{C}_{R'}} + V_- \exp^{+\hat{C}_{R'}} \quad (4.29)$$

The full algorithm for slipper is shown below:

1. Train the weak learner using the current distribution D
 - (a) Split data into **GrowSet** and **PruneSet**
 - (b) **GrowRule**: starting with the empty rule, greedily add conditions to maximise equation 4.28
 - (c) **PruneRule**: starting with the output of **GrowRule**, delete some final sequence of conditions to minimise equation 4.29 where $\hat{C}_{R'}$ is computed using equation 4.27 and **GrowSet**.
 - (d) Return as R_t either the output of **PruneRule**, or the default rule, whichever minimises the equation $Z = 1 - (\sqrt{W_+} - \sqrt{W_-})^2$.

2. Construct $h_t : \chi \rightarrow \mathbb{R}$:

Let \hat{C}_{R_t} be given by equation 4.27 (evaluated on the entire dataset).

Then

$$h_t(x) = \begin{cases} \hat{C}_{R_t} & \text{if } x \in R_t \\ 0 & \text{otherwise} \end{cases}$$

3. Update:

(a) For each $x_i \in R_t$, set $D(i) \leftarrow \frac{D(i)}{e^{y_i \cdot \hat{C}_{R_t}}}$

(a) Let $Z_t = \sum_{i=1}^m D(i)$

(a) For each x_i , set $D(i) \leftarrow \frac{D(i)}{Z_t}$

Output the final hypothesis: $H(x) = \text{sign}(\sum_{R_t: x \in R_t} \hat{C}_{R_t})$

4.3 Summary

This chapter presented a selection of methods that may be used for building rules that model a domain. In the implementation of this thesis, only C4.5 was used for building rules. C4.5 was chosen because of its proven performance over a wide variety of data and readily available implementation.

The system presented in this thesis could easily be used with any of the other methods described here with virtually no changes required.

Chapter 5

Explaining Neural Networks

Neural networks have proved themselves as good predictors for a large variety of problems. Despite their successes, their use is frequently ruled out for many problems that could benefit the most from their predictive accuracy. The reason for this is very simple. The domains in which they are not used are typically where explanation is considered as important as prediction. These include safety critical or medical domains where reliance on unsupported predictions is simply not an option. The consequences of a bad prediction may be costly or even life threatening. Depending on the requirements of the domain, the ability to explain neural networks could be of use in several different ways:

- The explanations could be used to verify the networks operation.
- Failures that may occur can be understood by looking at the explanation of the neural network operation and steps can be taken to avoid similar failures in the future, e.g. by retraining with new examples.
- The network may be replaced by the explainable model (e.g. a decision tree or set of rules), so that the operation can be guaranteed at all times.

The final point in the above list may not seem sensible. If the network is to be replaced by a set of rules or a decision tree why not build such a structure from the start and skip the intermediate step of building a neural network? The answer to this is quite simple, neural networks are good at generalisation. Given a limited number of training examples, neural networks can make excellent approximations to the true function being studied and therefore perform well on future unseen examples. This good performance can be used to tag a larger set of generated data. A more comprehensible learner can then use this larger collection of data to generate a structure with similar characteristics to the original network.

This chapter provides a brief outline of some of the areas of rule extraction relevant to this thesis. For a more complete review of the area see [63, 5]. This chapter begins by outlining in Section 5.1 the two high level strategies that may be adopted for network explanation. These include network decomposition in Section 5.1.1 and black box methods in Section 5.1.2. The issue of evaluating the quality of extracted explanation rules is addressed in Section 5.2. Two explanation approaches are then covered in Section 5.3. Finally Section 5.4 concludes the chapter with a look at explanation of neural network ensembles.

5.1 Strategies

5.1.1 Network Decomposition

Decompositional methods translate networks structure directly to rules. In the case of backpropagation networks, therefore, the aim would be to identify the hyperplanes partitioning the input space. At first this may appear to be a powerful method of explaining neural networks, however, they have the limitation of being architecture dependant. Many legacy networks are

excluded as they were not trained with explanation in mind.

One of the first methods proposed for the explanation of neural networks involved direct decomposition of the network. The method KT was proposed by Fu [27]. The core idea behind the KT algorithm is perhaps the most obvious approach to the decomposition of most networks with weighted interconnections between units. When presented with an input example, the KT algorithm searches for the smallest possible combination of inputs whose values will trigger the desired output. A set of rules explaining the network can be accumulated in this way.

A good introduction to decompositional methods involves an understanding of local function networks. For an example of these networks and rule extraction see Andrews [6]. The idea behind these networks is that they have boundaries in each dimension and these boundaries are adjusted as new examples are misclassified. The final boundaries in each dimension form the boundaries of the rule terms.

Another local function network approach is by Berthold [8] and involves the use of Rectangular Basis Function Networks. The training algorithm for these networks is based on Berthold's previous work on Dynamic Decay Adjustment (DDA) [7] for training Radial Basis Function Networks.

Rectangular Basis Function networks work by creating hyperrectangles that encompass areas of the hyperspace defined by the input features dimensions.

Each hidden unit p_i^c of class c and index $i(1 \leq i \leq m_c)$, m_c being the number of hidden units of that class) has a number of parameters associated with it. These are:

- An activation $R_i^c(\cdot)$
- A reference vector (centre): $\vec{r}_i^c = (r_{i,1}^c, \dots, r_{i,n}^c)$

- An amplitude(weight): A_i^c
- Two sets of “radii”:
 - Set of axes along which the rectangle is spread out towards infinity K^∞
 - Set of axes K^e along which the rectangle is restricted with a radius of σ_j

The activation of a unit is 1 if a new training example of the same class is correctly classified by that unit and zero otherwise. The first unit to correctly classify the example has its weight increased by a constant amount. This weight will be used later during classification of unseen examples. The centre of a hidden unit is the first example that causes a misclassification in another unit. The radii are the dimensions of the hyperrectangle around this centre.

Training begins with no hidden units. Hidden units are added only when existing units misclassify a new example. When a new hidden unit is added, the dimensions of that unit must be shrunk so that no conflicts exist between that unit and all units of other classes and vice versa (it is not necessary to shrink that units dimensions with respect to units of the same class).

A simple example of shrinking the dimensions of a rectangle is shown graphically in two dimensions in Figure 5.1. The example in *a* has been misclassified and it is necessary to shorten one of the dimensions of the enclosing rectangle. To do this there are three choices. In *b* the left dimension has been shrunk, in *c* the top dimension has been shrunk and in *d* both dimensions have been shrunk. It can be seen clearly from this diagram that to avoid the misclassification it is only necessary to shrink a single dimension so the solution presented in *d* can be discounted. To maximise the size of hyper-

rectangles, it is recommended to shrink the dimension that will lead to the least reduction in area of the remaining rectangle. Clearly, this means that b is the correct dimension to shrink.

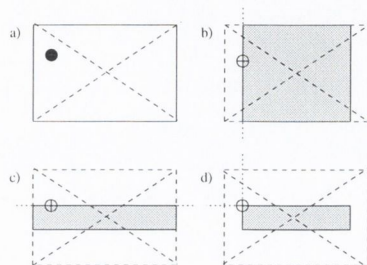


Figure 5.1: Shrinking the dimension of a rectangle in rectangular basis function networks

Execution of the network proceeds by testing each hidden unit with the unseen example. If the example falls within the dimensions of that unit's hyperrectangle, then the output unit adds that unit's weight to the total score for that class. The class with the highest score is outputted as the prediction.

The extraction of rules from this network is very straightforward. Each hidden unit can be mapped directly to a single rule. This is easiest to visualise from Figure 5.2. In this figure, the centre of the rectangle is marked by the unit with the cross through it. The rectangle is unbounded on the top side. In the other dimensions, there are examples of another class that caused those dimensions to be shrunk when they were misclassified. The rule that is extracted from this rectangle is:

IF $x_1 < V_X < x_2$ AND $y_1 < V_Y$ THEN TRUE

where V_X and V_Y are the values of the features X and Y to be tested by the rule.

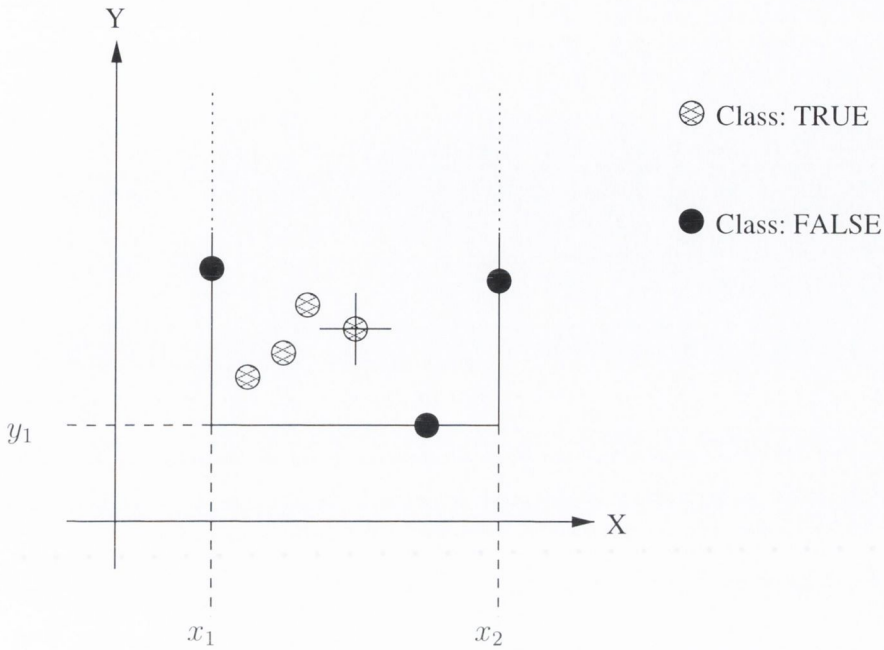


Figure 5.2: Extracting a rule from the hyperrectangle in a hidden unit

5.1.2 Black Box

In contrast to direct decomposition methods, black box methods require no knowledge of the internal network structure. They operate solely by analysing the predicted output(s) on input vector(s). To analyse this relationship black box methods typically use traditional rule learning algorithms to model the network.

A popular strategy adopted by researchers into black box methods is to use a second machine learning approach that models the input/output behaviour of the network. The second learning algorithm does not learn the target function directly, instead it learns the response of the neural network to the training inputs. In modelling the network, it is hoped that strong patterns that are being used internally by the network for prediction will be made clear and the second learning algorithm will output rules based on

these patterns. Prominent examples of this type of learning are:

- Thrun: Validity Interval Analysis(“VIA”) [59]
- Craven & Shavlik - TREPAN [19]
- Schmitz, Aldrich and Gouws - ANN-DT [52]

In his paper describing his technique for rule extraction from neural networks, Thrun outlines four criteria for successful explanation of a network. These are:

- No architectural requirements – the proposed method should work with all types of networks.
- No training requirements – special provisions during training should not be required, their presence would likely prevent the method being used with legacy networks.
- Correctness – generated rules should reflect the knowledge contained in the network as accurately as possible and not merely be approximations to the network operation.
- High expressive power – powerful languages for expressing the rules extracted from the network should be used. Compact rulesets are more easily understood.

Thrun presents his method for network explanation using these criteria as goals.

Thrun analyses backpropagation-like networks by propagating entire activations intervals of units. These activation intervals comprise upper and lower bounds that, when satisfied, lead to a provably correct activation space.

In the context of rule extraction, these intervals are used to prove or disprove conjectured rules. Initial intervals constraints are set using a linear programming method (Thrun uses the Simplex method). Intervals are refined by propagating them both forwards and backwards through the network. It should be noted that these propagations are independent of network training (i.e. they have no relationship to the gradient updates in back propagation networks).

Craven & Shavlik use a more conventional approach to the extraction of rules from a network for the purpose of explaining a network's operation. Using the network as an oracle, a large number of generated examples are labelled. These generated examples in addition to the training set, define precisely the network response. Using these examples, a decision tree is built to model the dependencies captured in these examples. Decision trees are easily decomposed to rules and hence are good structures for explaining networks.

The decision tree algorithm used by Craven & Shavlik, grows the tree in a best first manner as opposed to the more traditional depth first approach of C4.5 and CART (both C4.5 and CART are described in full in Chapter 4. At each node in the tree, TREPAN stores:

- A subset of training instances
- A set of query instances
- A set of constraints

The subset of training instances are simply those training instances that reached that node. The query instances are used in conjunction with the training instances to either determine the next split of an internal node or

alternatively set the class of a leaf node. Finally the set of constraints define criteria that instances must possess in order to reach this node. These constraints are used when generating a set of query instances for the node.

As mentioned previously, the TREPAN decision tree algorithm grows the tree in a best-first manner. To determine the next best node to grow, Craven & Shavlik attempt to estimate which node when grown will give the greatest increase in fidelity. This is justifiable because, the idea is to model the network as faithfully as possible. The equation for selecting this node is:

$$f(N) = reach(N) \times (1 - fidelity(N)) \quad (5.1)$$

where $reach(N)$ is an estimate of the fraction of instances that reach N when passed through the tree and $fidelity(N)$ is the fidelity of those instances reaching N with respect to the original network.

The reasoning behind a best first growth of the decision tree is both practical and commendable. The size and complexity of the tree can be finely controlled and at any stage the tree can be verified by a user as an increasingly accurate global model of the network.

The last major difference in the decision tree in TREPAN is the stopping criterion. Three criteria are used, one local and two global. The local criterion is simply a probability measure that the instances reaching the potential leaf node are all of a single class. When this probability reaches a preset constant value the current node is marked as a leaf. The first of the global criteria is a limit on the number of possible internal nodes. The second global criterion uses a validation set to evaluate the fidelity of the increasingly more accurate trees for modelling the network. The tree with the lowest error is considered to be the best.

The final method for explaining neural networks that is examined here is

by Schmitz, Aldrich & Gouws and is named ANN-DT. Like Craven & Shavlik, Schmitz et al. also aim to produce a decision tree as their final output. Also like Craven & Shavlik, Schmitz et al. use the network to label a collection of generated data to be used by the decision tree builder. Unlike the previous methods, though, ANN-DT focuses on explaining problems with a continuous numeric output. The basic steps followed in building the decision tree are similar to CART (which is described in Section 4.1.2 and hence only the differences are covered here).

The selection of attribute and threshold for splitting is done in two different ways. The first is by minimisation of the weighted variance:

$$V_w = \sum_{k=1}^2 \frac{n_k}{n} \text{Var}(O_k) \quad (5.2)$$

This is the same procedure as used in the CART algorithm when forming a regression tree.

The second method is an analysis of attribute significance. This method focuses on inter-relationships that occur inside the network function. If attributes can change their value independently of one another then the absolute value of the directional derivative integrated in a straight line between two points can be used as a measure of the significance of a single attribute. However, if in the more likely case, there is a dependence between variables, this is not appropriate.

The absolute variation between two points x_i and x_j in the dataset is:

$$v_{ij} = \int_{x_i}^{x_j} |\Delta f(x) \cdot u| dx \quad (5.3)$$

where u is the unity vector in the direction $x_i - x_j$.

The variation between attributes having a large effect on the output of the neural network, $f(x)$ and variations in the neural network output response

will be highly correlated. Thus, a measure of the significance of a variable a over a data set S would be the correlation between the absolute variance of the function and the absolute variation of that attribute taken between all possible pairs of points in S :

$$\begin{aligned} \sigma(f)_a &= \text{correlation}(v_{ij}(f), v_{ij}(a)) \\ &= \frac{\sum_i^N \sum_{j>i}^N v_{ij}(f) - \bar{v}(f)v_{ij}(a) - \bar{v}(a)}{\sqrt{\sum_i^N \sum_{j>i}^N v_{ij}(f) - \bar{v}(f)} \sqrt{\sum_i^N \sum_{j>i}^N v_{ij}(a) - \bar{v}(a)}} \quad (5.4) \end{aligned}$$

Those attributes with the highest correlation between changes in the neural network output and changes in the attribute value are the most significant and should be used for splits higher in the tree.

Schmitz et al note that where the number of computations is excessive, the result can be approximated by selecting random pairs.

The data is recursively split in this way until either the standard deviation is zero or when some stopping criterion is reached. This criterion would prevent a split occurring where the outcome of one of the sub branches would not be statistically different from the outcome of the other branch. This prepruning is designed to help prevent overfitting of the decision tree and to improve overall comprehensibility of the presented rules.

One such test that can be used to determine if two branches are statistically different from each other is the F-test (it tests if the standard deviation of two populations are equal). This test is only applied to branches formed below a preset level in the tree. This helps ensure that tree growth is not stopped prematurely. In addition, branches containing only a single data point are also deemed to have failed. Finally, the maximum depth of trees is capped at a preset maximum to prevent overly large and incomprehensible

trees.

The final stage of the ANN-DT algorithm is to prune the trees. The authors use a simple fast greedy pruning technique. They note however that the more sophisticated CART algorithm for pruning decision trees could also be used.

5.2 Evaluating Rule Quality

The process of extracting rules from neural networks is a trade-off. The following measures were proposed by Towell & Shavlik [61]:

- *Accuracy*: The accuracy of the rule set is simply a measure of the rule sets ability to accurately predict unseen cases.
- *Fidelity*: The fidelity of the rule set measures how well the rule set models the behaviour of the neural network. In cases where the rules are being used to verify the operation of the network, the rules should exhibit a high degree of fidelity.
- *Comprehensibility*: The comprehensibility of the rule set is a measure of the ‘understandability’ of the rules. This may be measured in two different ways. The first measures the global comprehensibility, i.e. the total rule set size. A bigger rule set is likely to be more difficult to understand. Once it has been determined that the extracted rules are potentially comprehensible, the second measure looks at the individual rules. If the number of terms in each rule is not too large, the rules may be easily assessed. This assessment may lead to new insight into the data being studied and may help prove or disprove theories, by indicating previously unnoticed trends or confirming suspected trends that exist within the data.

Obviously it is important that the rules exhibit an accuracy as good as the original network. However, it is also important that this accuracy reflects a good fidelity between the rules outputs and the network outputs. A rule set with a similar accuracy to a neural network but that makes mistakes on different examples to the neural network is not a particularly good description of that network. This is the reason why it is inadequate to train a neural network and a more comprehensible learner, such as a decision tree, separately, and conclude that the tree represents the knowledge in the neural network. A more common approach is that the decision tree is built to model the behaviour of the network by using the network to label a set of data.

An equally important consideration is the trade-off between comprehensibility and fidelity. A learner built to model the network with perfect fidelity may be totally incomprehensible. A decision tree may contain many bushy subtrees that are no easier to understand than the original neural network. However, pruning that tree will lead to the decision tree classifying examples differently to the network thus reducing the fidelity.

Finding a good balance of these quality measures is essential in any algorithm that attempts to explain a neural network.

5.3 Global \vee Local Explanation

Most researchers have focused on producing *global model explanations*. These models aim to fully describe all situations in which a particular event will occur. In a global model, there is an implicit trade-off between the complexity of the model and its fidelity. This trade-off can be seen in terms of the fidelity and comprehensibility evaluation criterion proposed by Towell & Shavlik [61] that are listed in section 5.2. A model built with perfect fidelity may be very complex and the comprehensibility will therefore be reduced. A

comprehensible model however may be useless for verifying the operation of the network because its fidelity is too low.

Although a global model may be useful in many situations, it is argued here that it is not always appropriate. For example, it may be useful in the problem of predicting success in IVF (in-vitro fertilisation) research, studied by Cunningham et al.[22], to produce a global model of the phenomenon. Such a model would allow practitioners to spend time understanding the conditions leading to success and to focus their research on improving their techniques. Also, a global model would allow the targeting of potential recipients of the treatment who have a higher probability of success. This would lead to a monetary saving for the health service and would avoid great disappointment for couples for whom the treatment would most likely fail. A global model might also allow doctors to suggest changes a couple might make in order to improve their chances of success with the treatment.

In the accident and emergency department of a busy hospital, the explanation requirement would be quite different. Here the need is for decision support rather than knowledge discovery. What is needed is an explanation of a decision in terms of the symptoms presented by individual patients. This explanation task is described here as *local explanation*.

In the context of ensembles (see Chapter 3), the decision to use a global or local approach becomes an even bigger issue. Ensembles built for maximum diversity may have many individual networks that are experts in particular areas of the input space. Building a global model from an ensemble may result in a trade off where many of the finer details covered only by a small number of networks are dropped. The global model may fail to give the most precise rules as opposed to if a local approach had been used and the production of explanations had been delayed until the explanation is actually

required.

Other researchers who have also approached the problem of local explanation include Šíma [55] and his approach is reviewed in [14]. Local explanations of time series predictions have also been explored by Das et al [23].

Although both Šíma and Das use local explanation in that they both provide explanations on a case by case basis, neither is directly comparable to this work. Das' method focuses on finding repetitive patterns in time series (and does not rely on neural networks). Šíma's method does rely on a backpropagation neural network but instead of providing rules as explanations, outputs percentage importance values for each of the inputs based on a decomposition of the neural network weights.

5.4 Rule Extractions from Ensembles

Despite the advantages of ensembles, little work has been done to provide explanations for predictions made by ensembles, although the importance of finding a method for rule extraction from ensembles was highlighted by Craven [20].

Although little work has been done in rule extraction from ensembles, it is still possible to use any of the black box methods introduced in section 5.1.2. However, this approach to the explanation of ensembles may not be optimal. A well built ensemble should comprise diverse members, each of which are experts in different areas of the input space. Modelling an ensemble as a black box ignores this diversity and looks only at the bigger picture. An algorithm that attempts to harness this diversity to produce optimum rules may outperform black box methods.

The two methods that are presented here for explaining neural network ensembles are:

- Domingos - Combined Multiple Models(“CMM”) [24]
- Zhou - Rule Extraction from Neural network Ensembles(“REFNE”) [69]

In his algorithm CMM, Domingos [24] creates an ensemble of neural networks using bagging. The ensemble is then used to assign labels both to the original training examples and to a set of randomly generated instances of fixed size. The `c4.5rules` package [47] is then used to create a set of production rules that model the behaviour of the ensemble. C4.5 is described in more detail in Section 4.1.1. Domingos reports reasonable fidelity and accuracy using this approach.

A more recent article specifically addressing the extraction of rules from neural network ensembles has been published by Zhou [69]. The method proposed is called Rule Extraction From Neural network Ensembles(REFNE).

The trained ensemble is used to generate additional instances that are used in the subsequent rule extraction algorithm.

A rule is formed when a subset of attributes are found to classify a set of examples that fall into a single class.

The search for the subset of attributes begins with the selection of a single symbolic attribute and testing all of the possible values of this attribute. If no value of this attribute classifies all examples it appears in to a single class, then all other single symbolic attributes are similarly tested. If no single attribute can be found to fulfill the necessary criterion, then all subsets of two or more subsets of symbolic attributes are considered. The process of adding symbolic attributes and searching all subsets of a particular size for a rule continues until no more symbolic attributes are left. At this stage a continuous attribute is discretised and the search continues.

In order to optimise the speed of this search, Zhou uses the “experience” of failed searches to guide later searches. An example of this is that if a set of symbolic attributes $\{a_i\}$ fails to find a rule and a continuous attribute b is discretised, then future rule searches should only examine subsets also containing b because all the other subsets of $\{a_i\} \cup b$ have already been examined.

In order to avoid suspect or poor rules, REFNE implements a number of optimisations. These include dropping any instances for which a tie exists. That is, if an equal number of networks in the ensemble predict different classes for an instance, it is not clear which label should be assigned to that instance. Also, REFNE may be tuned to ensure that any rules to be added to the output rule set increase the fidelity by at least a constant value.

Zhou reports good results using REFNE when compared to the popular C4.5 rules package.

5.5 Summary

This chapter described some of the many methods that have been proposed for explaining both individual neural networks and ensembles of networks.

The work presented in this thesis complements the methods presented for explaining a neural network as rules by introducing a ranking system for these rules that focus the user on the most important variables influencing the prediction.

The decision to pursue local explanation rules out any method targetted at the output of a single global model of the ensemble, e.g. Domingos [24] and Zhou’s [69] algorithms or using a black box model to model the entire ensemble behaviour. Instead, one of the individual network explanation methods is required to produce rules for each of the networks in the ensemble.

Both decomposition and black box methods are available to explain individual networks. It was decided to use a black box method for the extraction of rules from a network. Black box methods showed much promise for both comprehensibility and fidelity measures in many of the papers reviewed. Also, the selection of a black box method meant that the choice of network was not restricted in any way.

The precise method used was that each network labelled a set of generated data. The `c4.5rules` [47] package was then used to build a set of production rules that modelled the networks behaviour. C4.5 was chosen as the rule builder because of its proven ability to generate comprehensible and accurate (increased accuracy when modelling network behaviour is equivalent to increased fidelity) trees and rule sets. Maximising both of these variables is crucial in explaining neural networks.

Chapter 6

Solution

The solution presented in this chapter to the problem of explaining the outputs of neural networks is in fact more flexible and can be applied to any machine learning ensemble where the individual members can be expressed as rules. For this reason the description of the process of translating the networks to rules and the rule selection process have been decoupled in this chapter.

The idea behind this solution is very simple. Section 6.1 defines how the solution involves building a ruleset explaining each network. Section 6.2 covers the process of testing each rule with every one of the training data to find the coverage for each rule. Section 6.2.1 then shows how this simple coverage information can be augmented with a more useful and precise description of how the rule covers the training data. Section 6.2.2 then describes the online process of using this calculated coverage information for calculating a fitness score and ranking the rules using this fitness score. A worked example of this process is presented in Section 6.2.3. Finally Section 6.3 describes how extra diversity was added to the original networks by training them on feature subsets to solve a problem involving extraneous terms in rule clauses.

6.1 Building an Ensemble of Rules from an Ensemble of Neural Networks

The neural networks chosen for use in this system were the standard back-propagation [50] type. These neural networks have been shown in the past [58, 53]), to have excellent generalisation for a wide variety of prediction tasks. Furthermore, it has also been shown [22] that for a large number of these prediction tasks, ensembles have the effect of increasing both prediction accuracy and stability.

An ensemble of backpropagation networks was built by training individual networks on a bootstrapped set of data. Bootstrapping, described in the context of machine learning by Breiman [10], randomly selects training examples with replacement from a set of examples. In this way, approximately two thirds of examples will be selected at least once if the total number of examples selected is the same as the number of examples in the complete set. The remaining examples that have not been selected at all are used for preventing overfitting during training of the network.

For problems involving data with a skewed class distribution, the minority class was duplicated in the data. This prevented the network being biased towards the majority class.

The black box approach was chosen in this work to generate rules from these neural networks (see Section 5.1.2). The specific black box method used was the C4.5 decision tree algorithm and the associated `c4.5rules` program was used to generate production rules for use in explanations. C4.5 was used for generating both the rules for individual networks used in local explanations and the single global decision tree/rules. Using C4.5 in this way is similar to the way in which Domingos [24] uses it.

Although C4.5 was chosen to build the rules in this particular case, *any*

rule learner could be used. The choice of which rule learner to apply is the choice of the modeller and the solution presented here is *not* dependent on this choice.

Once trained, each network was used as an oracle on the training data and the ensemble of networks also acted as an oracle to label the data used in the production of the global rules. Decision trees were then built to model these targets(i.e. model the networks/ensemble). Finally, a ruleset was extracted from these decision trees.

To compensate for a lack of data in some of the datasets studied, extra examples were generated. These examples were generated using a very simple algorithm, namely:

- For every example in the training data
 - For every feature in an example
 - * If the feature is continuous add a small amount($\pm 5\%$) of noise to its value

With these extra data, the decision boundaries between classes should be clear and lead to a well defined tree.

6.2 Rule Coverage Statistics

The concept of rule coverage is pivotal to the operation of this system. It is by estimating how well a rule covers the training data that it is possible to estimate how well it will cover a future unseen example. The simplest measure of rule fit is the fraction of training examples that fire a rule. A rule is considered fired by a single example if the values of each feature in the example fit inside the boundaries of any term in the rule clause testing that feature. The coverage score of the rule is increased by a constant amount

each time the rule is fired. This coverage proportion alone gives a reasonable indication of the generality of a rule. Due to incomplete datasets, however, many rules may classify areas of the input space incorrectly. Identifying the areas of the input space covered by the rule and in which we have the most confidence in the rule is the subject of the next section.

6.2.1 Advanced Rule Coverage Statistics

To improve the coverage information, it is useful to know what areas of the input space are well covered by the terms in the rule clause. This can be accomplished by calculating some extra statistics.

When computing the simple proportion of examples that fire each of the rules, it is necessary to save these examples in a list associated with that rule. When all of the training examples have been tested with all of the rules, the mean and standard deviation of each of the features with continuous values that appear in the rule clause are calculated. This is not possible with symbolic features as these have an implicitly perfect fit.

6.2.2 Rule Fit and Ranking

Once the off line process of calculating rule coverage statistics is complete, it is possible to calculate an on-line rule fitness score for new examples with respect to the rules in the system. However, not every rule is checked for every new example.

When a new example is introduced into the system, either the rulesets or the original neural networks vote on the outcome. *Only* the ensemble members contributing to the majority prediction are used in the ranking of predictive rules. Each of the rules in each of these “correct” rulesets (or the rulesets corresponding to the correct networks) are considered with this new

example. All of the rules fired by this example are collected together.

At this stage, the system has identified a group of rules that could potentially be used to explain the example being tested. Using the rule coverage statistics calculated earlier, it is now possible to go one step further and rank these rules in order of our confidence in the predictiveness of each rule. This is done by calculating a fitness score for each rule. This fitness score is calculated by testing how similar the new example is to the training examples that also fired this rule.

For every term in each rule clause, a score is calculated using equation 6.1. The mean(μ) in this equation is the mean of the feature values for each term that fired the rule and the standard deviation(σ) is the standard deviation of the feature values for each term that fired the rule.

$$\text{Fitness}_X = \max_i \left| \frac{x_i - \mu_i}{\sigma_i} \right| \quad (6.1)$$

Once a fitness score for each of the terms in each rule has been calculated, each rule must be assigned an overall fitness score. The term with the maximum (i.e. poorest) fitness score is then selected as the fit for the rule as a whole. This is similar to the approach taken in Mycin [54] when comparing the conjunction of two hypotheses where the weakest measure of belief is also taken as the overall measure of belief.

There are two exceptions in the calculation of this fitness score:

- Rules whose terms are duplicates of others
- One sided rules, where the value of a feature lies on the unbounded side

The first of these exceptions arises frequently for rules where a common pattern exists in the data and several rulesets predict the output class using

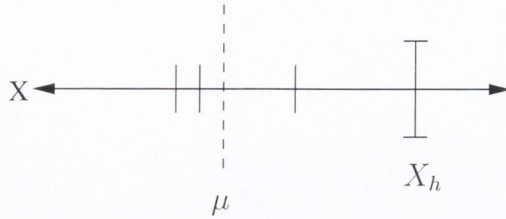


Figure 6.1: Number line showing unbounded rule

this rule. In order that this rule is not presented more than once to the user, duplicate rules are removed. The fitness of the final remaining rule is then boosted by a small constant to reflect our increased confidence in the predictiveness of the rule.

The second exception is for one sided rules, these are rules that are bounded on a single side only. If the value of the feature being tested is on the unbounded side of the mean, that term is automatically given a perfect fitness score. This situation can be seen graphically in Figure 6.1. This diagram shows a number line representing the rule:

IF $X < X_h$ THEN TRUE

The training examples that fitted this rule are marked along with their mean value. Any value of X that fits this rule and is less than the mean μ is automatically given a perfect fitness score.

This fitness measure gives us our main criteria for ranking rules. However, it is possible for ties to occur when a group of rules all have maximum fitness. Ties can be resolved in these situations by considering rule specificity, i.e. the number of terms in the rule. In situations where simple explanations are preferred, rules with few terms are preferred. In situations where elaborate explanations might be interesting rules with more terms in the left-hand-side can be ranked higher.

The doctor examining the results of the Bronchiolitis data (one of the

datasets used in the evaluation of this research) suggested that, in practice, simple explanations might be appropriate for holding a patient overnight whereas more elaborate explanations might be necessary for discharge. The logic behind this is that a single symptom might be enough to cause concern about a child whereas to discharge a child no adverse symptoms should be observable.

So in selecting and ranking rules to explain the Bronchiolitis data the main criterion was the ranking based on the rule fit. Ties were then resolved by selecting the simplest rules for admissions and the most complex rules for discharges. This produced very satisfactory results.

In general therefore, a policy for resolving ties should be agreed with a domain expert on a class by class basis.

6.2.3 Worked Example of Calculating Rule Fit Using Iris Dataset

To demonstrate how the fitness metric works, a simple example of analysing extracted rules is included in this section. The dataset used is Fishers Iris dataset from the UCI repository [9].

This dataset comprises three classes with 50 examples of four features each. One of these classes is linearly separable from the other two. For a back propagation network this is a straightforward task. In order to increase the difficulty of the problem, the number of training examples in each class has been reduced to 17. Using bootstrapped sets, the number of examples from each class seen during training of individual networks will be varied thus giving better diversity.

Nine unseen examples, three from each class were used to test the system. For each of these examples, predictions were made and five ranked rules were

output as explanations of these predictions.

These rules were then ranked by confidence of their fit to the unseen test example. A sample test example appears below along with two rules that were selected as predictive of the class.

```
sepal_length      == 5.7
sepal_width       == 2.6
petal_length      == 3.5
petal_width       == 1.0
```

```
[0.548107]
```

```
IF 0.497102 < petal_width <= 1.178020
```

```
THEN Iris-versicolor
```

```
[1.552252]
```

```
IF 2.089680 < petal_length <= 4.198180
```

```
THEN Iris-versicolor
```

The boundaries of these rules are shown graphically in figure 6.2.

From this figure, it can be seen that the Iris-versicolor test point is significantly closer to the mean of the training points in the petal_width dimension than it is to the mean of the points in the petal_length dimension. This closeness increases our confidence in recommending this rule as an explanation for the prediction of the network for that example, i.e. it is ‘like’ the examples on which this rule is based.

In the results on the Iris dataset several examples of rule duplication arise. For example, the following rule was ranked as one of the five most predictive

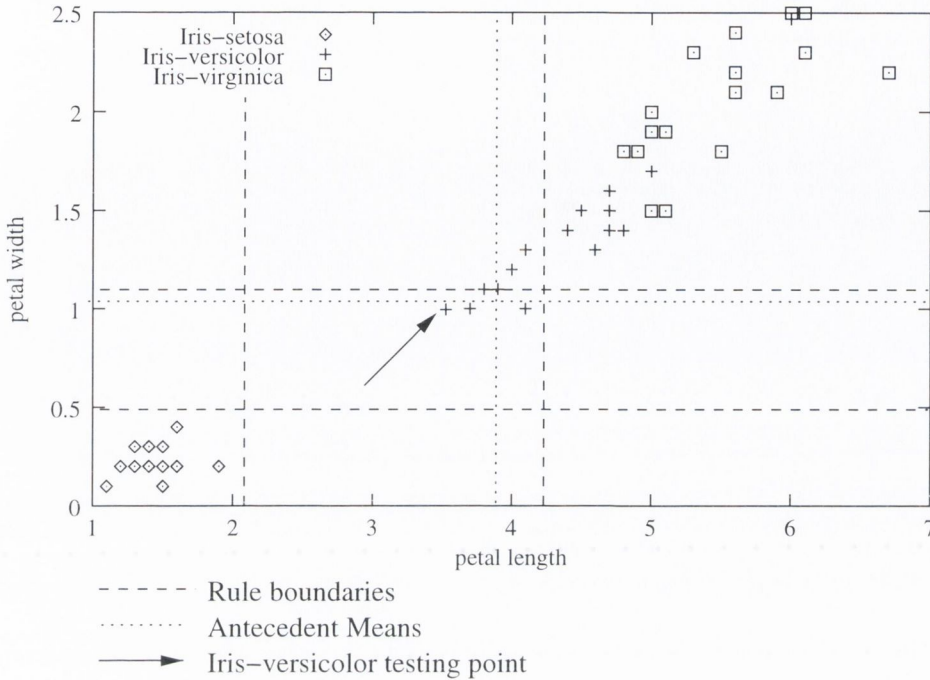


Figure 6.2: Graph of Iris data in two dimensions

rules(the fitness for this rule is reported in square brackets at the top of the rule):

```
[0.940733]
IF sepal_width > 2.294270
AND 2.089680 < petal_length <= 4.771990
AND petal_width <= 1.711730
THEN 1
```

In fact this rule appeared four times in the set of predictive rules. The other occurrences of this rule may have had slightly different limits, but for the purposes of duplicate boosting, it is important only that the example being tested fits each of the rules. To reflect this increased confidence in this rule, its fitness value was ‘boosted’ by dividing its original fitness by 1.2 for every duplicate occurrence. The original fitness of this rule before duplicates

were taken into account was 1.95. Without the duplicate rules this rule would not have been as good as the next ranked rule whose fitness was 1.86.

6.3 Rule Simplification

A major problem encountered with many of the rules selected using the above method was that, although, the rules contained many excellent terms in their clauses, there were frequently conditions which provided little extra information or were contradictory to the proposed class output. In order to try to remove these extraneous terms, each of the networks was trained using a subset of the available training features. This approach has the useful effect of increasing diversity in the ensemble, which should give an overall decrease in the ensemble error assuming the error in the individual networks does not increase substantially. In the case of this work, feature subsets were selected according to the wrapper based algorithm described in [68] and described below:

- Generate a random feature mask (i.e. a feature subset) and estimate the generalisation error for that mask using cross validation
- Cycle through the mask flipping each bit in turn and if the estimation of generalisation error on this new mask is less than before accept the flip, otherwise reject it and reset the bit
- Repeat from step 1 until no improvements are found (i.e. no bit flips accepted) on a full traversal of the mask

Once the required number of masks has been found, each of the networks in the ensemble is trained using separate masks. Rule extraction then continues as described in Section 6.1.

The rules extracted using this method are more focused on specific local patterns inside the data and fewer extraneous terms appear inside rule clauses.

Chapter 7

Implementation

7.1 Introduction

The implementation of the ideas described in this thesis was for testing the feasibility and performance of the concepts described. The implementation comprised command line tools under Linux. Only the results of the implementation were presented to the experts evaluating the results. Writing a graphical user interface (“GUI”) for interacting with the system was not included in the scope of the thesis and so this system is not necessarily representative of how this work might be integrated into existing hospital databases.

The descriptions provided in this chapter therefore concentrate on the tools that were used to implement the system and why they were chosen.

7.2 Practical Implementation Issues

7.2.1 Programming

The implementation to test this system was written using only free software [56] on Linux [60]. In total three languages were used in the development of the system software. These were:

- C++
- Python
- Bash shell script

C++

For reasons of efficiency, the neural networks were implemented using C++ [57] with the g++[2] compiler. C++ is a good choice for this type of problem as it is a flexible language that compiles directly to machine code.

With the advent and adoption of the ANSI C++[1] specification, C++ has become a significantly more portable language. In particular, the increased availability of the STL(Standard Template Library) allows developers to focus even more on solving problems rather than tackling low level implementation details such as allocating and freeing memory. Good use was made of the STL's collection and stream classes for reading the database of examples.

Python

Python [62] is a flexible high level scripting language and is well suited to the manipulation of text files. With the exception of the neural network implementation where C++ was used, Python was the main language used.

Python has many attractive features:

- Perl-like regular expressions
- Object oriented
- Flexible data types(e.g. lists and associative arrays)
- Functional programming tools(map, filter, reduce and lambda)

The regular expressions were used to good effect while parsing the output of c4.5rules. The text output of this program included error information and headings that were not required for this system and these were easily excluded when searching for rules with regular expressions.

The object oriented nature of Python allowed for a good abstraction of the various parts of the programs, e.g. reading the data format file (in C4.5 “names” format), reading the data, reading the rules and separating each term in each rule clause.

Lists and associative arrays are natural data types for holding rules and examples and associating information with them. The functional programming like functions help speed up the time consuming task of iterating over structures in an interpreted language by performing the loop in the faster compiled code of the interpreter.

Bash shell script

The final part of the implementation was written using Bash [31] shell script. This was used to tie the individual python scripts and the neural network programs together.

Shell scripts can be used not only for starting programs, but also to dynamically set and adjust the values of environment variables and even to loop over groups of commands. The return values of programs can also be read to check for and report any runtime errors.

7.2.2 Distributing Work

The process of creating the masks (see Section 6.3) is very intensive, particularly if the training set is large. For this reason, this work was distributed across a cluster of computers each running Linux.

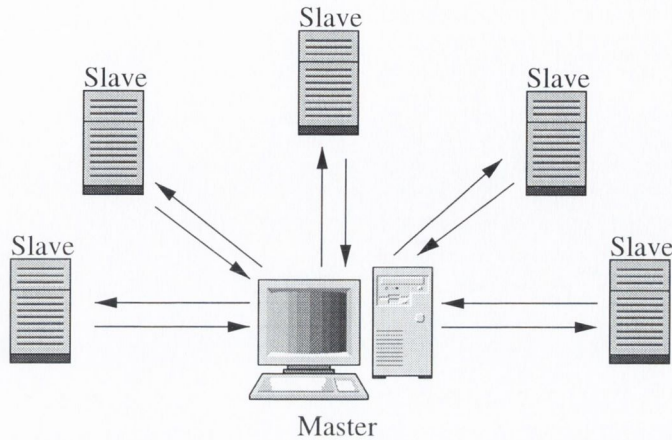


Figure 7.1: Master/slave architecture

The cluster is configured such that users' home directories are shared between all computers using NFS. All computers participating in the job therefore have access to the same pool of data. Each computer has local disk space, so intermediate results can be saved and accessed quickly on demand.

The distribution of work was carried out using the master/slave paradigm. In this scenario, one computer acts as a master, while all other computers are considered slaves. The master coordinates the work to be done and is responsible for collating results. This distribution architecture can be seen graphically in figure 7.1.

To facilitate communication, the program was written using the MPI(Message Passing Interface) [4]. This interface defines a flexible array of functions for sending and receiving messages. The implementation used here was MPIch from the Argonne National Laboratory [3], which is also available under a free software license.

The code for distributing the work of training the neural networks for testing mask performance was written using the C++ MPI bindings. This facilitated simple integration with the neural network code.

Summary and future of parallel computing

The methods being used by machine learning researchers are becoming ever more computationally demanding and the problems being tackled are growing ever more complex. Even the rapid advances in processing speed often cannot keep pace, in this environment, to provide real time results and interactivity.

The distribution of processing work across clusters of computers therefore holds great promise for researchers.

The nature of the problem described in this thesis is often known as “Embarrassingly Parallel” due to its obvious parallel solution. There is no need for communication between different learners and the training time decreases in direct proportion to the number of processors available.

For more complex problems however, there may be large communications overhead between processes and it is in this environment that a cluster will be of most benefit. The improvement over a single computer is likely to be several orders of magnitude greater as processes do not need to be swapped in and out of memory and expensive kernel inter-process communication (“IPC”) calls can be avoided. Gigabit and faster networking speeds and the zero-copy implementation in the modern Linux make networks a viable transport mechanism for most machine learning tasks involving extensive communication. Furthermore, clusters can comprise off the shelf components and when combined with the Linux operating system and other free software they make a cost effective yet easily upgradeable and scalable alternative to expensive proprietary supercomputers.

Chapter 8

Evaluation

The evaluation of the work undertaken in this thesis was not a straightforward task. While accuracy of predictions can be assessed from a dataset, the quality of associated explanations can only be assessed by experts working in the area of the prediction. For this particular reason, two domains, for which experts were readily available, were assessed. These domains were:

- Predicting whether or not very young children showing signs of bronchiolitis should be admitted to hospital including explaining the reason behind admitting or discharging a child.
- Predicting the quantity of the blood thinning drug Warfarin that should be administered to patients based on their previous history of taking the drug and their current symptoms.

The same datasets were also used in evaluations of earlier work in this research, [64, 65].

The iris dataset used in Section 6.2.3 to demonstrate how the ideas of local explanation and rule ranking are implemented is a useful introduction to the evaluation, although not part of the formal evaluation. The iris dataset demonstrates that the proposed system makes a reasonable attempt at finding the explanations in which we have the most confidence.

As stated above, the two domains chosen for evaluation by this thesis were medical. The principal reason for this selection is that medical data provides a potentially rich source of data for machine learning practitioners. Accuracy and explanation are both very important in terms of user acceptance of a machine learning based system. Work that focuses only on accuracy may not be accepted in a live implementation, this would rule out many novel neural network based approaches.

Each of the selected domains are discussed separately, Bronchiolitis in Section 8.2 and Warfarin in Section 8.3.

It should be stressed, that the selection of these medical domains for study is somewhat arbitrary but also driven by availability of experts in the area. The work is in no way restricted to these domains and the primary goal is to show that local explanation is a viable approach to the explanation requirements when compared with a global approach using a similar rule extraction method.

8.1 Evaluation Process

The process of evaluating the results was the same for both datasets. This process consisted of the generation of rules from each of the networks in an ensemble along with a global set of rules modelling the ensemble operation. This process is described in more detail in Chapter 6.

As noted in Chapter 7, the work undertaken and the programs developed for the purposes of testing this thesis were not representative of how this system would actually be integrated into current medical systems. For this reason, the experts evaluating the results did not interact with the system and were simply presented with formatted results.

A total of ten examples were selected randomly from each of the datasets

being studied. These examples were not used at any point during training of the networks or building of the subsequent rules, their use is confined to future tests of the ideas embodied by the programs. Predictions for, and explanations of, these examples were given to the experts for evaluation using both global and local approaches. A maximum of five ranked rules were produced by each example for each method.

The domain experts evaluated the results by scoring each of the explanations based on the predicted output. The scores given ranged from 1 – 5. These scores translated into assessments of rule quality, with a higher score indicating a better rule.

1. Wrong
2. Poor
3. Fair
4. Good
5. Very good

For each set of scored rules (one set per dataset), a number of overall scores were calculated to determine how well the local and global rules performed.

These scores were as follows:

1. Average Top Rules
2. Average Top Correct Rules
3. All Predictions
4. All Predictions (Minimum Rules)

5. Correct Predictions

6. Correct Predictions (Minimum Rules)

Scores 1 and 2 are concerned only with the top ranked rule from each method for each of the datasets. The top ranked rule is the rule in which the program has the most confidence and hence hopefully the one with the most accurate information. If the system was to display only a single rule it would be this one and hence, this is an important score.

Score 3, takes into account the scores for all rules displayed to the user of the system. Again, if the rule ranking technique has worked well in the local approach these scores should not be too low relative to the global approach or the other scores. Score 4 is similar to this, but uses only the minimum number of rules produced by the methods for calculating the score for each test case. For example, if the global approach uses two rules to describe the test case and the local approach uses four rules to describe the test case then only two rules from each approach are used in calculating the score.

Score 5 considers only the rules in those test cases correctly predicted by the system. This score is important as it shows the performance of the system when it has almost certainly fully understood the case under investigation. Finally, score 6 uses only the minimum number of rules in each of the correctly predicted test cases as described above.

There is no comparison made between these results and the results of other methods of rule extraction from ensembles, e.g. Zhou's REFNE [69] (see Section 5.4). This is for two (related) reasons:

- The *focus* of this thesis is on demonstrating that the local explanation approach is a viable approach to the problem of explaining the outputs of an ensemble. There is no sensible application of methods generating

global rulesets explaining ensembles for a single network.

- Comparing the results obtained below with the results obtained from a global model from a different method is an unfair comparison between two different rule learners - either one could perform better on a particular dataset.

For these reasons, to make a fair comparison between local and global rules, it is necessary to use a similar method for the generation of both sets of rules. The method chosen was to use the C4.5 package and this is described in more detail in Chapter 6.

8.2 Bronchiolitis

8.2.1 Data

The bronchiolitis dataset has the structure shown in Table 8.1.

Table 8.1: Bronchiolitis dataset structure

Total examples	118
Continuous Features	10
Symbolic Features	12
Missing Values	Yes

The bronchiolitis dataset represents a somewhat poor coverage of the overall domain. This was confirmed in 5-fold cross validation tests done in the domain. For each cross validation test, an ensemble comprising 5 networks was built from the training data using bagging to select the data. Average accuracies were computed for each of the 25 networks along with average accuracies for the 5 ensembles. For each network trained, a ruleset was also extracted to model its behaviour. The accuracies of these rulesets were recorded and likewise the accuracy of these rulesets used as an ensemble

was also recorded. This provided an insight into how well the rules performed compared to the original networks. These accuracies are shown in Table 8.2.

Table 8.2: Results of 5-fold cross validation performed on bronchiolitis data

	Av. \pm S.D.
Average Ensemble Accuracy	72.5% \pm 2.4
Average Rules Ensemble Accuracy	70.4% \pm 2.8
Average Network Accuracy	68.8% \pm 5.2
Average Rules Accuracy	66% \pm 5.7
Average Network/Rules Fidelity	82% \pm 7.6

This table clearly demonstrates the fact that not only did the ensemble outperform the individual networks but the networks were also quite unstable. This instability is reflected in the standard deviation figures reported next to the accuracies. In the case of the individual networks the standard deviation is more than double that for the ensembles. This feature of an increase in accuracy and stability is one of the positive features of using ensembles.

8.2.2 Explanations

The explanations associated with the predictions from both the local and global approaches were evaluated by Dr. Paul Walsh, an expert in the area of bronchiolitis.

An example of a rule produced using both the local explanation method and a global explanation method for the example is shown in Tables 8.3 and 8.4, respectively.

Before analysing these scores, however, it is useful to first see the accuracies of the two methods using the ruleset ensembles for predictions on the test data. This is shown in Table 8.5.

Table 8.3: Example evaluated by expert ('?' indicates a missing value)

Feature	= Value
Age in Months	= 3.17
Anorexia	= 0
Decreased Activity	= 0
Smoking ANY	= 1.22
Smoking MOTHER	= ?
Entry Temperature	= 37.00
HR	= 162.00
HR gt 98%	= 0
RR1	= 38.00
Sa O2	= 97.00
HR2	= 110.00
HR2 gt 98%	= 0
RR2	= 28.00
Sa O2 2	= 95.00
Dehydration	= None
LOC	= Alert
Retractions	= 0
Grunting	= 0
BS	= 0.00
DecBil	= 0
Crac and Whez	= 0
Whez only	= 1
Decision	= DISCHARGE

The first of the analyses performed on the scores, involved taking the average of the scores for each example. This was performed twice, once using all the rules and the second time using only the minimum number of rules produced by the two methods, e.g. TWO for the example shown in Table 8.4. In this way the same number of rules was used in the comparison. The number of wins, losses and draws for each method was then computed. This is given in Table 8.6.

Table 8.6 shows that the local explanation approach performs well. Taking all of the rules into account for each example in the test set, the local

approach is a clear winner. The probability of getting draws is much higher when using the minimum set of rules. If there is only a single rule produced using one of the methods, then, a draw results if the scores for these rules are the same. When averaging over all the rules, draws are much less likely.

Table 8.7 contains the overall scores calculated for the bronchiolitis results. The descriptions for these scores are set out in Section 8.1 at the start of this chapter.

The results from this table clearly show that the local explanation approach outperforms the globally extracted rules. The average scores are higher in all cases.

The final statistic that was performed was a pooled t-test. The average scores for all rules in both local and global approaches were averaged and from these a pooled standard deviation was calculated. The t-test was found to be significant at a 90% confidence level. For the score of all the rules using only the minimum number in either method, this confidence level was 60%.

T-tests were also carried out for the other scores. For the average top ranked rule score, the confidence level was found to be 70%, while for the average top ranked rule in correctly identified cases, it was 60%. Lastly, for the correct predictions, the confidence level found was 70% and for the correct predictions using only the minimum number of rules from both methods, it was 90%.

It is expected that given a larger test, these scores would further improve.

8.3 Warfarin

8.3.1 Data

The Warfarin dataset has the structure shown in Table 8.8.

The evaluation of the Warfarin data is less straightforward. The data supplied for this domain represented excellent coverage of the domain. Therefore, the explanations extracted using the global approach could be expected to be reasonably accurate as the global model would miss few of the details in its construction. This was indeed confirmed to be the case.

A 5-fold cross validation of the dataset was also performed, similarly to the bronchiolitis data. This involved the construction of 5 ensembles comprising 5 networks each. Each of the errors for the ensembles were averaged and the average error of the 25 networks was also recorded. These results are shown in Table 8.9.

It is clear from this table that the ensemble failed to provide a significant boost in accuracy above that of the individual networks. This is symptomatic of a well covered domain.

8.3.2 Explanations

The evaluation of the Warfarin results were carried out by Dr. Stephan Byrne, an expert in the area of administering the Warfarin drug.

One of the ten examples used in the final test set is shown in Table 8.10 and the rules produced for this example are shown in Table 8.11.

The accuracies of both methods using the derived rulesets for predictions on the ten test points can be seen in Table 8.12.

The first evaluation of this data simply takes the average of each of the scores for each example and calculates how well the local explanation approach performs against the alternative rules built to model the ensemble. The results of this can be seen in Table 8.13.

The results in Table 8.13 show that although the rules built to model the ensemble do outperform the locally extracted rules, the gap between them is

not very wide. The ‘Minimum Rules’ row of this table, is where the average scores of the minimum number of rules produced by both methods for a particular example is calculated. So for the example shown in Figure 8.10, this minimum number would be ONE rule.

The detailed analyses of the rules produced for each example is given in Table 8.14. This table show that the globally extracted rules perform better than the locally extracted rules (though both methods have relatively high scores for every category, neither fails dramatically on any analysis). A description of these scores is contained in Section 8.1 at the start of this chapter.

In favour of the local explanation, but not visible from these results, is the fact that only the local explanation produced rules that were marked as excellent by the expert in the area (half of the examples contained rules marked as excellent). Also weighting the scores somewhat in favour of the ensemble modelled global rules is the fact that these rules failed to produce any explanation for one of the test examples. Thus the effective explanation quality for this rule was ZERO, but this is not reflected by the averages.

As in the case of the bronchiolitis data a pooled t-test was performed by averaging all of the rules for both the local and global approaches. Together with a pooled standard deviation, a t-statistic was found. This statistic was found to be significant at the 95% confidence level. High confidence level were maintained for the other scores. This aids in confirming the belief that the global rules had captured much of the most important information very succinctly.

8.4 Summary

The evaluation of this work has shown that where the coverage of data is poor, ensembles can be used to increase accuracy and stability over a single model. Where there is good coverage in the data, using an ensemble leads to little, if any, improvement in the predictive accuracy.

Furthermore, explanation of predictions in a poorly covered domain are greatly improved by the use of local explanation techniques. The local approach delays the production of explanations until the last possible moment, thus maximising the information available and producing a better explanation. The bronchiolitis data demonstrates this phenomenon. The rules produced by the global model lacked sufficient detail and/or feature values were incorrect.

Although the scores given by the expert for the rules may seem low, there are a number of reasons for this. There are general reasons that apply to both datasets and more specific reasons for each dataset.

The quality of rules outputted by this system is highly dependent on the underlying rule generation technique. For this implementation C4.5 was used. A different rule inducer may produce better results.

Medical data is also inherently noisy. This noise may come from several places, but two important factors are:

- Symptoms are recorded at time of entry
- There are many extraneous factors not captured by the data available

The bronchiolitis dataset is very prone to the time symptoms are recorded. When presented with a child displaying symptoms of bronchiolitis, a doctor may use his/her experience and senses (e.g. touch and sight of the child)

to admit that child before the symptoms become severe. In addition, when any doubt whatsoever exists, the child is more likely to be admitted. The symptoms presented to the machine learner, however, are those of the child at time of entry and these may not yet have progressed to a level mandating entry.

The problem of extraneous factors in the dataset is also clearly visible in the bronchiolitis dataset. The expert in this area posed the example of a child who was otherwise healthy but whose mother abused drugs and hence the child would most likely be admitted to hospital.

In the case of the bronchiolitis data, the criteria used by the doctor in evaluating the explanations was to compare the explanations to published criteria to be used when evaluating children presenting symptoms of bronchiolitis. To exactly model these criteria is an extremely difficult proposition for any machine learning algorithm, particularly in the presence of the noise described above.

The results in the case of the Warfarin dataset, could have been greatly improved by using the entire dataset. Only a subset of the data was used in order to increase the difficulty of the problem.

In contrast, global models must make a trade-off between fidelity and comprehensibility as they try to explain an entire domain in a single model. As a consequence of this trade-off, important traits and characteristics in the individual models may be lost. In a well covered domain, for example Warfarin, the most important characteristics of the data are well represented and the global model represents a good explanation of the domain. Even in this well covered domain, however, the global model lost some of the finest details. The best rules from the point of view of the expert, therefore, were produced by the local explanation approach.

It could be argued that statistically it is more likely that excellent rules will appear in the local approach as more rules are outputted. However, this ignores the fact that the local approach must rank a potentially large number of rules and these excellent rules were consistently ranked highly and therefore output to the user. Also if the global model had truly covered all details of the domain, it too would have been graded as excellent. This was not the case and it is therefore clear that the comprehensibility/fidelity trade off was taking place and important details were being dropped. Also, the local approach does not require that a large number of rules are outputted and it still displays good performance when only the top ranked rule is considered. This shows that the rule ranking technique works well.

Table 8.4: Rules produced for the example in Table 8.3

Local	Global
<p>[0.00] IF Entry Temperature <= 44.96 AND Sa O2 > 93.18 AND LOC = Alert AND Crac and Whez = 0 AND BS <= 0.27 THEN DISCHARGE</p>	<p>[0.00] IF Sa O2 > 94.52 AND HR2 <= 131.23 AND Crac and Whez = 0 AND RR2 <= 29.37 THEN DISCHARGE</p>
<p>[0.00] IF HR > 141.00 AND Dehydration = None AND Retractions = 0 AND Age Months > 1.87 THEN DISCHARGE</p>	<p>[0.00] IF Sa O2 > 95.55 AND RR2 <= 31.89 AND BS <= 0.10 AND Whez only = 1 THEN DISCHARGE</p>
<p>[0.00] IF Sa O2 > 93.50 AND LOC = Alert AND Crac and Whez = 0 AND BS <= 0.27 THEN DISCHARGE</p>	
<p>[0.00] IF Sa O2 2 > 91.89 AND Dehydration = None AND Retractions = 0 AND Age in Months > 1.87 THEN DISCHARGE</p>	
<p>[0.00] IF Age in Months > 1.87 AND Sa O2 > 95.30 AND Dehydration = None AND HR <= 166.00 THEN DISCHARGE</p>	

Table 8.5: Accuracies on test data

	Accuracy
Local explanation	90%
Ensemble Model Rules	70%

Table 8.6: Wins, losses and draws for the rules computed by the local explanation method

	Wins	Losses	Draws
All Rules	7	3	0
Minimum Rules	4	3	3

Table 8.7: Analysis of rules generated for bronchiolitis data

	Locally Extracted Rules	Global Rules
Average Top Rules	2.8	2.5
Average Top Correct Rule	2.89	2.71
All Predictions	2.84	2.42
All Predictions (Minimum Rules)	2.53	2.42
Correct Predictions	2.76	2.5
Correct Predictions (Minimum Rules)	3.04	2.5

Table 8.8: Warfarin dataset structure

Total Examples	323
Continuous Features	8
Symbolic Features	5
Missing Values	0

Table 8.9: Results of 5-fold cross validation performed on Warfarin data

	Av. \pm S.D.
Average Ensemble Accuracy	70.1% \pm 7.9
Average Rules Ensemble Accuracy	70.6% \pm 6.0
Average Network Accuracy	70.7% \pm 6.9
Average Rules Accuracy	71.1% \pm 6.9
Average Network/Rules Fidelity	89.7% \pm 4.9

Table 8.10: Example evaluated by expert

Feature	= Value
Age	= 75.00
Weight	= 62.70
INRMasurement	= 2.30
PreviousDose	= 3.29
TherapyDuration	= 127.00
TargetINR	= 3.75
NoADR	= NoAdverse
Gender	= Female
CurrentMedicines	= None
OTC	= None
Alcohol	= 0.00
Compliance	= TooMuch
INR Delta	= 1.45
Dosage	= $2 \leq \text{SubsequentDose} < 5$

Table 8.11: Rules produced for the example in Table 8.10

Local	Global
<p>[0.00] IF Age > 73.39 AND Alcohol <= 3.94 AND INR Delta <= 1.84 THEN 2 <= SubsequentDose < 5</p>	<p>[0.71] IF 1.53 < INRMeasurement <= 3.47 AND 0.90 < PreviousDose <= 3.58 AND Alcohol <= 14.22 AND INR Delta <= 2.31 THEN 2 <= SubsequentDose < 5</p>
<p>[0.08] IF Age > 59.25 AND 2.07 < PreviousDose <= 4.05 AND INR Delta > -1.02 THEN 2 <= SubsequentDose < 5</p>	
<p>[0.31] IF Age > 51.33 AND 2.07 < PreviousDose <= 4.51 AND Alcohol <= 19.70 THEN 2 <= SubsequentDose < 5</p>	
<p>[0.36] IF 1.58 < INRMeasurement <= 3.53 AND 2.82 < PreviousDose <= 3.53 AND Alcohol <= 16.38 THEN 2 <= SubsequentDose < 5</p>	
<p>[0.56] IF Age > 63.27 AND 1.63 < PreviousDose <= 5.28 THEN 2 <= SubsequentDose < 5</p>	

Table 8.12: Accuracies on test data

	Accuracy
Local Explanation	80%
Ensemble Model Rules	70%

Table 8.13: Wins, losses and draws for the rules computed by the local explanation method

	Wins	Losses	Draws
All Rules	3	6	1
Minimum Rules	4	5	1

Table 8.14: Analysis of rules generated for the Warfarin data

	Locally Extracted Rules	Global Rules
Average Top Rules	3.3	3.78
Average Top Correct Rules	3.38	4
All Predictions	3.24	3.79
All Predictions (Minimum Rules)	3.21	3.79
Correct Predictions	3.38	3.92
Correct Predictions (Minimum Rules)	3.24	3.92

Chapter 9

Conclusions & Future Work

The research and factual data used as part of this thesis clearly demonstrates that explanation on a case by case basis, also known as local explanation, is a viable approach for solving certain problems. Included among these problems are those where the prediction being explained must be acted upon in a timely fashion and where there is no need to fully analyse the domain.

Local explanation is of particular value in poorly covered domains. The bronchiolitis data studied in this thesis is an excellent example of such a domain. When producing a single global model of this domain many details, that were included in the rules presented on a case by case basis, were omitted from the final model .

In a domain with better coverage, such as the Warfarin domain, this thesis demonstrated that the rules extracted from the global model performed equally well or outperformed those extracted locally. This is because important traits in the data were well represented in the individual models and little information was lost in preparing the full global model.

The rule ranking criteria proposed in this thesis performed well in selecting some of the better rules to be displayed to the user. Furthermore, this rule ranking criteria intuitively selects rules that are also likely to be selected

by a non-expert user.

Medical data sets have long been an important source of data for machine learning practitioners. Frequently, however, more emphasis has been placed on making accurate predictions with little or even no importance placed on explanation of the results. A principal aim of this thesis was to redress this imbalance by providing a general framework for explanation. With more work in the area of explanation, we may see greater user acceptance of machine learning software by the medical community and other users. When the user can decide for himself the correctness of the prediction, it will be perceived as less of a threat and more of an aid to the user.

Medical datasets also have problems, which although perhaps not unique to them, are very apparent. For instance, different output classes may have different explanation requirements. The complexity of the solution may vary according to the class being predicted. For example, a child showing just a single symptom of bronchiolitis should be admitted to hospital, whereas a child to be discharged must meet more stringent criteria. The explanation presented to the user for a prediction must therefore to the greatest extent possible follow these conditions.

There is perhaps an even greater problem when studying medical datasets. Often the examples provided for training represent as much information as possible. However, external factors relative to the patient's lifestyle and even doctor experience to admit a patient before the patient's symptoms become serious can mean that the symptoms recorded may not reflect the true seriousness or otherwise of a patient's illness. Many of the features that might be expected to be very predictive of the output do not perform that well in practice. This has a knock on effect on the quality of the rules output. This limitation may need to be overcome by using a more select number of

training examples that include less overall noise.

9.1 Future Work

Future work in the areas covered by this thesis could include:

- Problems involving regression outputs
- Improved feature selectors - possibly making use of Fürnkranz's round robin technique [28]
- Improved data capture
- Introducing standard measures of comprehensibility

There are many interesting *regression problems* in both the medical and financial fields. This may not be too difficult to model. In the same way that a fit can be found for examples to the rule antecedent, a similar fit could also be found for the rule output.

An interesting problem that became clear during the research conducted for this thesis was the need for *good feature selection*. One approach to this problem could include performing feature selection on a class by class basis. This would entail finding the most predictive features of each of the classes and only using those features for predicting that class. It is not entirely clear how this could be done.

One possible solution may be to use *round robin learning* [28] and learn the best set of features for a class when trained with one other class. Training could also proceed using round robin learning and using the best subset of features for "learning" each class. The explanations are most likely to contain the best features for those classifications when explaining the outputs.

Although this thesis is not strictly focused at the medical world, the need for *improved data capture* from medical systems is an important requirement that became apparent as the research progressed. From a machine learning perspective, this data capture could help improve on the current ad hoc methods of extracting the data for later analysis. With careful consideration during the building of such a system, data could be more easily filtered to exclude possible outliers not representative of the problem being studied (e.g. patients whose diagnosis is not necessarily reflective of the symptoms first presented).

A final area of future work is also multi disciplinary. Current machine learning research focuses almost exclusively on accuracy as a means of identifying the most useful methods. More work is required to introduce *standard measures of comprehensibility* that can be used to assess the usability of methods.

Bibliography

- [1] C++ standard. <http://www.ansi.org/>.
- [2] Gnu compiler collection(gcc). <http://gcc.gnu.org/>.
- [3] Mpich. <http://www-unix.mcs.anl.gov/mpi/mpich/>.
- [4] Mpi forum. <http://www.mpi-forum.org/>.
- [5] R. Andrews, J. Diederich, and A. B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6), 1995.
- [6] R. Andrews and S. Geva. *RULEX & CEBP Networks as the Basis for a Rule Refinement System*, pages 1–12. IOS Press, 1995.
- [7] Michael R. Berthold and J. Diamond. Boosting the performance of RBF networks with dynamic decay adjustment. In *Advances in Neural Information Processing Systems*. MIT Press, 1995.
- [8] Michael R. Berthold and Klaus-Peter Huber. From radial to rectangular basis functions: A new approach for rule learning from large datasets. Technical report, University of Karlsruhe, 1995. 15-95.
- [9] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

- [10] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1994.
- [11] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
- [12] C.A. Brunk and M.J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms. In L. Birnbaum and G. Collins, editors, *Proceedings of the 8th International Workshop on Machine Learning*, pages 389–393. Morgan Kaufmann, 1991.
- [13] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.
- [14] I. Cloete and J.M. Zurada, editors. *Knowledge Based Neurocomputing*. The MIT Press, 2000.
- [15] William W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*. Chambery, 1993.
- [16] William W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning*, pages 115–123, 1995.
- [17] William W. Cohen and Yoram Singer. A simple, fast, and effective rule learner. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence, Eleventh Conference on Innovative Applications of Artificial Intelligence*. American Association of Artificial Intelligence, 1999.
- [18] Marquis J.A. Condorcet. *Sur les Elections par Scrutiny*. Histoire de l'Academie Royale de Sciences, 1784.

- [19] M.W. Craven and J.W. Shavlik. Extracting tree-structured representations of trained networks. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 24–30. The MIT Press, 1996.
- [20] M.W. Craven and J.W. Shavlik. Rule extraction: Where do we go from here? Technical report, Department of Computer Sciences, University of Wisconsin, 1999. Machine Learning Research Group Working Paper 99-1.
- [21] Pádraig Cunningham and John Carney. Diversity versus quality in classification ensembles based on feature selection. In *11th European Conference on Machine Learning (ECML 2000)*. Springer-Verlag, 2000.
- [22] Pádraig Cunningham, John Carney, and Saji Jacob. Stability problems and the ensemble solution. *Artificial Intelligence in Medicine*, 20(3):217–225, 2000.
- [23] Gautum Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. In *Knowledge Discovery and Data Mining*, pages 16–22, 1998.
- [24] P. Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2(3), 1998.
- [25] L. Fausett. *Fundamentals of Neural Networks: architectures, algorithms, and applications*. Prentice-Hall, Inc., 1994.
- [26] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

- [27] L. Fu. Rule learning by searching on adapted nets. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 590–595, 1991.
- [28] Fürnkranz, Johannes. Pairwise classification as an ensemble technique. In *13th European Conference on Machine Learning (ECML 2002)*, pages 97–110. Springer-Verlag, 2002.
- [29] Fürnkranz, Johannes and Widmer, Gerhard. Incremental reduced error pruning. In *International Conference on Machine Learning*, pages 70–77, 1994.
- [30] S. Geman, Bienenstock E., and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4, 1992.
- [31] GNU. Bash shell. <http://www.gnu.org/software/bash/bash.html>.
- [32] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Patterns and Machine Intelligence*, pages 993–1001, 1990.
- [33] Sherif Hashem. Optimal Linear Combinations of Neural Networks. *Neural Networks*, 10(4):599–614, August 1997.
- [34] Tom Heskes. Bias/variance decompositions for likelihood-based estimators. *Neural Computation*, 10(6):1425–1433, 1998.
- [35] R. C. Holte, L. Acker, and B. W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 813–818. Morgan-Kaufmann, 1989.
- [36] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

- [37] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.
- [38] Y. Le Cun. Learning processes in an asymmetric threshold network, 1986.
- [39] Richard Maclin and David Opitz. An empirical evaluation of bagging and boosting. In *AAAI/IAAI*, pages 546–551, 1997.
- [40] Christopher J. Merz and Michael J. Pazzani. A principal components approach to combining regression estimates. *Machine Learning*, 36(1-2):9–32, 1999.
- [41] R.S. Michalski, I. Mozetic, J. Hong, and H. Lavrac. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the 5th National Conference on AI*, pages 1041–1045. Morgan Kaufmann, 1986.
- [42] M. Minsky and S. Papert. *Perceptrons : an introduction to computational geometry*. MIT Press, expanded edition, 1998.
- [43] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [44] S.I. Nitzan and J. Paroush. *Collective Decision Making*. Cambridge University Press, 1985.
- [45] M. P. Perrone and L.N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone, editor, *Neural Networks for Speech and Image Processing*, pages 126–142. Chapman-Hall, 1993.

- [46] Ross J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [47] Ross J. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann Publishers Ltd, 1988.
- [48] Ross J. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
- [49] J. Rissanen. Universal coding. *IEEE Transactions on Information Theory*, 4:629–636, 1984.
- [50] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representation by error propagation, 1986.
- [51] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
- [52] Gregor P.J. Schmitz, Chris Aldrich, and Francois S. Gouws. *ANN-DT: An algorithm for Extraction of Decision Trees from Artificial Neural Networks*, pages 369–401. In Cloete and Zurada [14], 2000.
- [53] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An experimental comparison (revised). Technical Report TR 955, University of Wisconsin-Madison, 1990.
- [54] E.H. Shortliffe. *Computer Based Medical Consultations: MYCIN*. Elsevier, 1976.
- [55] Jiří Šíma and Jiří Červenka. *Neural Knowledge Processing in Expert Systems*, pages 419–466. In Cloete and Zurada [14], 2000.

- [56] R. M. Stallman. The free software definition. <http://www.gnu.org/philosophy/free-sw.html>.
- [57] B. Stroustrup. C++. <http://www.research.att.com/~bs/homepage.html>.
- [58] S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, B. Roger, H. Vafaie, W. Van 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, 1991.
- [59] Sebastian B. Thrun. Extracting provably correct rules from artificial neural networks. Technical Report IAI-TR-93-5, Carnegie Mellon University, Dept. of Computer Science, 1, 1993.
- [60] L. Torvalds. Linux. <http://www.linux.org/>.
- [61] Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.
- [62] G. Van Rossum. Python. <http://www.python.org/>.
- [63] R. Wall and P. Cunningham. Exploring the potential for rule extraction from ensembles of neural networks. In J. Griffith and C. O'Riordan, editors, *11th Irish Conference on Artificial Intelligence and Cognitive Science(AICS 2000)*, 2000. (Also available as Trinity College Dublin Computer Science Technical Report TCD-2000-24).
- [64] R. Wall, P. Cunningham, and P. Walsh. Explaining the predictions of ensembles of neural networks on a case by case basis. In T. Elomaa,

H. Mannila, and H. Toivonen, editors, *Principles of Data Mining and Knowledge Discovery – 6th European Conference, PKDD 2002, Helsinki, Finland, August 19-23, 2002, Proceedings*, 2002.

- [65] R. Wall, P. Cunningham, P. Walsh, and S. Byrne. Explaining the Output of Ensembles in Medical Decision Support on a Case by Case Basis. In Cloete I. and Rohr K., editors, *Artificial Intelligence in Medicine: Knowledge Based Neurocomputing Methods*, 2003.
- [66] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard U., 1974.
- [67] G. Zenobi. A detailed derivation of the relationship between generalisation error and ambiguity in regression ensembles. Technical Report TR-CS-1999-76, Computer Science Department, Trinity College Dublin, December 1999.
- [68] G. Zenobi and P. Cunningham. Using ambiguity in preparing ensembles of classifiers based on different feature subsets to minimise generalisation error. In *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001. (12th European Conference on Machine Learning).
- [69] Z.-H. Zhou, Y. Jiang, and S.-F. Chen. Extracting symbolic rules from trained neural network ensembles. *AI Communications*, 2002. in press.
- [70] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: many could be better than all. *Artificial Intelligence*, pages 239–263, 2002.

Appendix A

Dataset Features

Table A.1: Bronchiolitis data features

Age in Months	Age of a child in months
Anorexia	Indicates if a child is suffering from anorexia
Decreased Activity	Indicates decreased activity of the child
Smoking ANY	Number of smokers in a household
Smoking MOTHER	Smoking Mother?
Entry Temperature	Temperature of child
HR	Heart Rate
HR gt 98%	Heart Rate greater than 98th percentile
RR1	Resting Rate
Sa O2	Oxygen blood saturation level
HR2	Heart rate after treatment
HR2 gt 98%	Heart Rate greater than 98th percentile after treatment
RR2	Resting Rate
Sa O2 2	Oxygen blood saturation level after treatment
Dehydration	Dehydration
LOC	Level of Consciousness
Retractions	Retractions
Grunting	Grunting
BS	Breath Sounds
DecBil	Decreased Billirubin
Crac and Whez	Crackles & Wheezes
Whez only	Wheezing Only

Table A.2: Warfarin data features

Age	Age of patient
Weight	Weight of patient
INRMMeasurement	INR Measurement
PreviousDose	Previous Dose of Warfarin administered
TherapyDuration	Duration of therapy
TargetINR	Target INR
NoADR	Number of ADR
Gender	Male/Female
CurrentMedicines	Taking current medicine
OTC	OTC
Alcohol	Units of alcohol consumption in units
Compliance	Compliance with drug regime
INR Delta	Change of INR