UNIVERSITY OF GRONINGEN

MASTER'S THESIS

Explainable AI: On the Reasoning of Symbolic and Connectionist Machine Learning Techniques

Author:
Cor STEGING

Supervisors:
Prof. Dr. Bart VERHEIJ
Prof. Dr. Lambert SCHOMAKER

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Artificial Intelligence

in the

Department of Artificial Intelligence

July 11, 2018

"We all want progress, but if you're on the wrong road, progress means doing an about-turn and walking back to the right road; in that case, the man who turns back soonest is the most progressive."

C. S. Lewis

UNIVERSITY OF GRONINGEN

Abstract

Faculty of Science and Engineering
Department of Artificial Intelligence

Master of Science in Artificial Intelligence

Explainable AI: On the Reasoning of Symbolic and Connectionist Machine Learning Techniques

by Cor STEGING

Modern connectionist machine learning approaches outperform classical rule-based systems in problems such as classification tasks. The major downside of the connectionist approach, however, is the lack of an explanation for the decisions that it makes, which is a quality that the rule-based systems are known for. The new sub-field of Explainable Artificial Intelligence (XAI) therefore aims to combine the performance of the connectionist approach with the understandability of knowledge systems. Examples of such hybrid systems extract rules from neural networks, which can explain how a network comes to a particular decision. However, this is built upon the premise that the reasoning of a neural network that performs well is sound. Previous research has suggested that this is not necessarily true; a high performance does not guarantee a sound rationale. This thesis therefore aims to investigate how well machine learning systems are able to correctly internalize the underlying structure or rules of a dataset. This is accomplished with the use of artificial datasets, whose rules and structure are known beforehand. The results confirm that a high classification accuracy can be obtained without having to have learned all of the rules that define the dataset. Furthermore, the systems are able to internalize certain rules better than others. When multiple rules define the dataset, interaction effects occur that cause the systems to be less proficient in internalizing the rules.

Contents

Abstract

1 Introduction 1
1.1 Current state of AI . . . 1
1.2 Explainable AI . . . 1
1.3 Goal of the Study . . . 3

2 Theoretical Background 5
2.1 Machine Learning . . . 5
2.2 Neural Networks . . . 5
2.2.1 Backpropagation . . . 7
2.2.2 Gradient descent variations . . . 8
2.2.3 Activation Functions . . . 8
Sigmoid . . . 8
Hyperbolic Tangent . . . 9
ReLU . . . 9
2.3 Symbolic Learning Techniques . . . 9
2.3.1 Association Rules . . . 10
Support and Confidence . . . 10
Lift and Conviction . . . 10
Apriori algorithm . . . 11
Classification Association Rules . . . 11
2.3.2 Decision Trees . . . 12
Gini Impurity and Information Gain . . . 12
Bagging, Boosting and random Forest . . . 13
2.4 Soundness of the Rationales . . . 14
2.4.1 Analysing Neural Networks . . . 14
2.4.2 Analysing Symbolic Learning Techniques . . . 15

3 Bench-Capon Replication Study 17
3.1 Neural Networks and Open Texture . . . 17
3.2 Replicating the study . . . 20
3.2.1 Datasets . . . 21
3.2.2 Neural networks . . . 22
3.2.3 Experiments and results . . . 23
Performance of the networks . . . 24
Age and Distance Experiment . . . 24
Examining the weights . . . 26
3.3 The Influence of Noise on Neural Networks . . . 28
3.4 Discussion and conclusion . . . 30

4 Symbolic Learning Methods **33**
4.1 Datasets 33
4.2 Decision Trees 33
4.2.1 The Algorithm 34
4.2.2 Results 35
4.2.3 The Influence of Noise on Decision Trees 38
4.2.4 Conclusion regarding Decision Trees 39
4.3 Association Rules 40
4.3.1 The Algorithm 41
4.3.2 Results 41
The AR System trained on Training set A 42
The AR System trained on Training set B 43
4.3.3 The Influence of Noise on Association Rules 44
4.3.4 Conclusion regarding Association Rules 45
4.4 Conclusion 46

5 Learnability of Symmetrical Boolean Functions **47**
5.1 Previous Research 47
5.2 Symmetrical Boolean Functions 48
5.3 Datasets 49
5.4 Machine Learning Algorithms 49
5.5 Single Condition Datasets 50
5.6 Interaction Results 53
Parity - Exact interaction 54
Parity - M-out-of-N interaction 58
M-out-of-N - Exact interaction 59
Discussion on interactions 59
5.7 The effect of training data 60
5.8 The Influence of Noise in learning Symmetrical Boolean Functions . . 66
5.9 Conclusion 69

6 Learning Tort Law using Neural Networks **71**
6.1 Dutch Tort Law 71
6.2 Experimental set-up 73
6.2.1 Datasets 73
6.2.2 Neural Networks 74
6.3 Results 74
6.4 The Influence of Noise in the Tort Law problem 79
6.5 Conclusion 80

7 Discussion and Conclusion **81**
7.1 The Importance of an Understandable Rationale 81
7.2 Results 82
7.3 Discussion 83
7.4 Future Research 86
7.5 Conclusion 87

A Best and Worst Node Configurations **89**

B Tort Law Decision Tree **95**

Bibliography **97**

Chapter 1

Introduction

1.1 Current state of AI

In recent years, the notion of artificial intelligence has reached the masses with the introduction of smart devices, digital assistants, self-driving cars and many other inventions. This booming interest in AI can partially be attributed to a shift in focus from using rule-based reasoning to the use of statistic-based reasoning. The former, and older 'wave' of AI uses the knowledge of experts and models this knowledge into a set of rules that its system uses to reason with. These systems are well defined and their reasoning is easy to understand by lay people. Their major drawback, however, is that the rigorous rule set is not flexible enough to accurately perform tasks that carry along some uncertainty; tasks such as character recognition are nearly impossible to define through rules alone. This is where the second 'wave' of AI comes in, which uses statistical learning rather than expert knowledge. These systems are not completely modelled by hand, but are instead trained using large sets of data. Statistical learning has opened the door for great achievements in fields such as pattern recognition and reinforcement learning that would have been impossible with knowledge systems. The issue that underlies this popular form of AI, however, lies in its inherent lack of transparency; the systems effectively function as a black box that provides an output to a given input, whose reasoning is entirely dependent on the data that it trains on. Moreover, successful machine learning techniques such as deep learning or random forests provide no reasoning or explanation of the decisions that it makes at all. More transparent machine learning techniques, such as decision trees or association rules, create a form of rule set from training data, which it uses to base its decisions on. Though more transparent in their reasoning, these systems often perform worse than their 'black box' counterparts. Another issue regarding statistical learning, is that the reasoning that is generated though the data-driven systems cannot be controlled and can therefore in some cases be undesirable; their reasoning can be dubbed as unethical or as a product of sheer coincidence and thus fallacious. Given these premises, people tend to distrust machine learning algorithms when they are not provided with an explanation (Edwards and Veale, 2017).

1.2 Explainable AI

The response to the lack of transparency, often called the third wave of AI, is Explainable Artificial Intelligence (XAI) (Gunning, 2017). XAI is supposed to combine the performance of statistical learning with the transparency of knowledge systems. These XAI systems would therefore be able to provide an explanation as why it has made a particular decision. By examining the reasoning of a system, the user is able to judge whether or not he or she should trust the decision that the system has made. This is also in compliance with the European General Data Protection Regulations (GDPR), which demands that a right to an explanation be given to users whose personal data is used by an algorithm (Voigt and Bussche, 2017). Even though the details surrounding the GDPR with regards to AI and machine learning are still a subject of debate (Wachter, Mittelstadt, and Floridi, 2017), XAI appears to be the most logical step forward. Attempts have been made to create so-called ’glass-box’ systems that offer explanations to their users without sacrificing much in terms of performance (Holzinger et al., 2017). However, there is generally a trade-off between transparency and performance, which is especially true for complex machine learning tasks. One of the type of techniques developed to combat the issue of the lack of explanations, has dealt with the extraction of rules or decision trees from (deep) neural networks (Hailesilassie, 2016). These systems should therefore maintain the performance of the connectionist approach (neural networks), yet achieve the same high level of transparency as the symbolic approach (e.g. decision trees).

Whether or not the explanation that such an XAI system gives would make sense, however, is not certain. The internal rationale that a black-box system uses may yield high performance results, even though that rationale is not sound (Bench-Capon, 1993). A great example of this phenomenon is the use of adversarial examples in image classification, in which an input image is altered slightly using a perturbation, such as in Figure 1.1 (Yuan et al., 2017). To the human eye, there is hardly any difference between the original image of a panda on the left and the perturbated image on the right. The neural network, however, reasons quite differently and mistakenly classifies the perturbated image as a ”gibbon”, with 99.3% confidence.

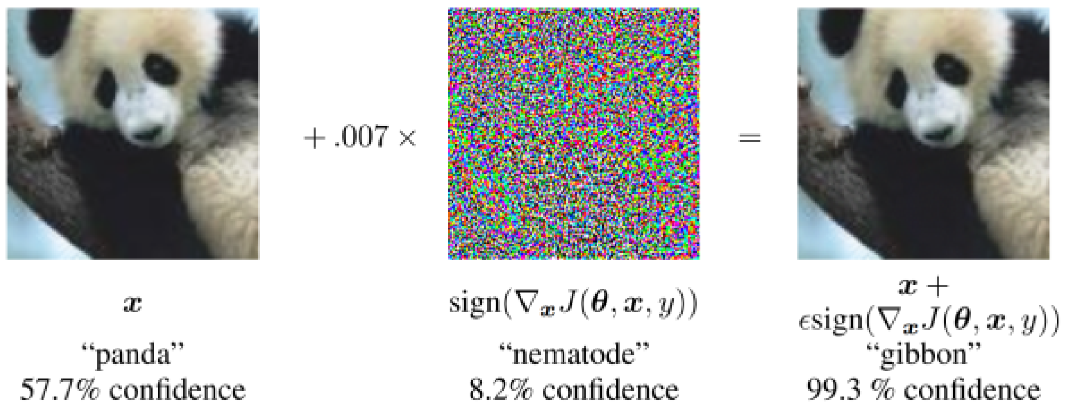

+ .007 ×
=
x
"panda"
57.7% confidence
$\text{sign}(\nabla_x J(\theta, x, y))$
"nematode"
8.2% confidence
$x + \epsilon\text{sign}(\nabla_x J(\theta, x, y))$
"gibbon"
99.3 % confidence

FIGURE 1.1: The classification on a clean example (left) and a perturbated adversarial image (right) (Yuan et al., 2017)

It is clear from this example, that the network may have been able to classify pandas relatively well, but was not able to accurately learn what a panda is, otherwise it would not have misclassified the adversarial example. Even if this network could provide us with an explanation as to why it classifies something as a panda, it would not give an understandable explanation (e.g. it is black and white, it has two eyes, etc.), but instead it would argue in a way that would seem incoherent and absurd to a regular human being. An explanation from an irrational system is therefore not sufficient; their reasoning should be sound as well.

1.3 Goal of the Study

It has been shown that machine learning algorithms often do not internalize the underlying structure of the data correctly, despite yielding a high classification accuracy (Bench-Capon, 1993) (Yuan et al., 2017). With the rise of explainable AI, a correct, understandable explanation of the decisions that the machine learning system makes is desired. This requires a systems that learns the rules and structures that define the data, instead of simply learning the correct input-output combinations. The aim of this study is therefore to investigate machine learning techniques in terms of how well they are able to learn rules, rather than in terms of a general performance accuracy. By creating artificial datasets that are generated from a set of predetermined rules, it becomes possible to determine how well machine learning systems are able to internalize these rules after training on the data. Both connectionist (black-box) techniques and symbolic learning techniques will be examined in a number of different learning scenarios.

In the upcoming chapter, the theoretical background will be explored, which will act as the foundation of the study. It will focus on explaining the machine learning techniques that are used in the proceeding experiments. In Chapter 3, the experiment performed by Bench-Capon in 1993 will be repeated in order to investigate the limitations of the neural network in terms of learning rules. Then, in Chapter 4, the same experiment will be performed, but with symbolic learning techniques instead of neural networks. Both association rules and decision trees will be used in this experiment. Chapter 5 describes a more formal experiment, in which symmetrical Boolean functions are used to generate various datasets. Both decision trees and neural networks are trained and tested on the datasets to investigate how well the systems are able to internalize these functions. Additionally, the effects that the amount of training data has on how well each function is learned is explored. Chapter 6 sets out to investigate how well neural networks are able to learn the rules of Dutch tort law. Specifically, the trade-off between the number of hidden layers and the total number of nodes will be examined. The thesis closes with Chapter 7, in which the study will be discussed and concluded.

Chapter 2

Theoretical Background

2.1 Machine Learning

A machine learning system is often described as a computer program that can increase its performance of completing a specific task by learning from experiences of similar tasks that it has performed in the past (Carbonell, Michalski, and Mitchell, 1983). This abstract notion of machine learning encompasses a large variety of applications that can broadly be discerned into three categories: Supervised learning, unsupervised learning and reinforcement learning (Ayodele, 2010). In supervised learning, a system will learn to predict the value of a dependent output variable using a number independent input variables. The system learns to predict the correct output values by training on a set of input-output pairs from the past. Unsupervised learning does not use such training examples, and is instead used to examine unknown structures within the data. For example, data can be clustered such that similar data points are grouped together. Lastly, reinforcement learning is used to teach systems to make correct decisions using a trial and error strategy. Good decisions are rewarded, thus encouraging the system to make similar decisions in the future.

This study focusses mainly on supervised machine learning. To be even more specific, the study deals with classification rather than regression. In classification, the input data will be classified into one of multiple classes, whereas in regression, the input is used to determine a continuous output value. Generally speaking, supervised learning classification attempt to learn rules from a labelled training data set that describe the relations between the attributes of the data and its label. These rules can then be used to provide labels to new unlabelled data. Supervised learning is famously used in tasks such as image recognition, wherein systems are able to infer what the object in a picture is with accuracies of up to 97% (He et al., 2016). Examples of supervised learning algorithms therefore include commonly used algorithms such as (deep) neural networks, support vector machines, naive Bayes and decision trees. The two broad approaches in creating classification systems are the connectionist approach and the symbolic approach. Both approaches will be discussed in more detail in the next sections

2.2 Neural Networks

Connectionist systems, or artificial neural networks are systems that are based on the network of neurons of the human brain. These artificial neurons, referred to as nodes, are organized in interconnected layers to form a network as seen in Figure

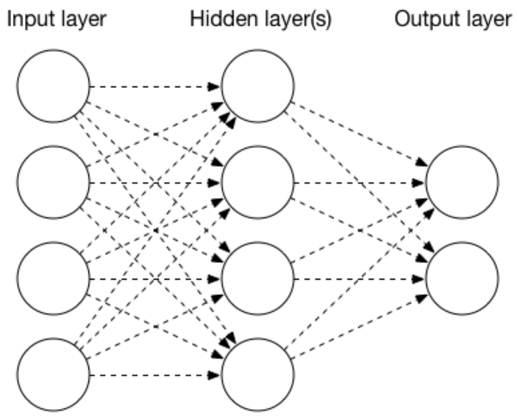


FIGURE 2.1: Diagram of a basic artificial neural network.

2.1. Each node in the network is able to receive signals from nodes in the previous layer and transmit signals to nodes in the next layer. The output of each node is determined by providing the sum of all of its inputs to an activation function. Additionally, each connection between two nodes has a weight that is associated with it, which causes the input of the receiving node to be multiplied by the value of the weight. Most neural networks also contain a bias for each node, which can be seen as an additional static input value. Neural networks always consist of at least an input layer and an output layer, where the input layer represents the independent variables and the output layer the dependent variables. The nodes of the input layer receive an initial signal from an input source that is then propagated forward throughout the network, eventually reaching the output layer. Most networks also contain one or more hidden layers of neurons between the input and output layers. Formally, the elements of a neural network can be expressed as follows:

- χ_j^l: input of node j in layer l
- W_{ij}^l: The weight from node i in the previous layer $l-1$ to node j in layer l.
- θ_j^l: The bias of node j in layer l
- O_j^l: The output of node j in layer l
- $\sigma(x)$: The activation function

The output of node j in layer l can then be expressed as follows:

$$O_j^l = \sigma\Big(\theta_j^l + \sum_{i \in (l-1)} \chi_i^l * W_{ij}\Big) \tag{2.1}$$

The key element in an artificial neural network is therefore finding the best values for the bias and the weights of the connections, such that the input values of the network generate the correct output values. These values are optimized by presenting numerous examples to the network and comparing the output that the network provides to the expected output of the examples and updating the weights accordingly. The way that the weights are updated is through a process called backpropagation.

2.2.1 Backpropagation

Backwards propagation, or backpropagation, calculates the changes that need to be made to each individual weights in the network (Rumelhart, Hinton, and Williams, 1988). In its most basic form, a round in the training phase of a neural network consists of three parts. First, a labelled example is forwarded propagated through the network. Then, the error of the system is calculated by comparing the output that the network produced with the expected output (the target T_j). Lastly, based on the error, the weights are adjusted throughout the network.

The forward propagation used during the training phase is the same as described earlier. To calculate the error of the system, an error function $E(O_j, T_j)$ is used that translates the difference between the output of the network O_j and the target output T_j into a real number. There are a number of different error functions that can be used, the default being the the following:

$$E(O_j, T_j) = \frac{1}{2} \left\| O_j - T_j \right\|^2 \tag{2.2}$$

This is the error of node j in the output layer, however, each of the weight in the network has contributed to this value in some way. Therefore, the gradient of the error of each output node is calculated and propagated backwards through all of the nodes in the network. This is done in order to update the weights and biases based on how much influence they have had on the final error. In other words, the rate of change of the error with respect to the value of a given weight needs to be calculated. This is effectively a gradient descent in which the global minima holds the optimal values for the weights to provide the minimal error. For the weights in output layer k, the partial derivative can be calculated as follows:

$$\frac{\partial E}{\partial W_i j^k} = \delta_j^k O_i^{k-1} \tag{2.3}$$

Where δ_j^k is the error term of the final layer:

$$\delta_j^k = \sigma'(O_j^k) E(O_j^k, T_j) \tag{2.4}$$

In this equation, σ' represent the derivative of the activation function. The partial derivative of weights in a hidden layer l can be calculated as follows:

$$\frac{\partial E}{\partial W_i j^l} = \delta_j^l O_i^{l-1} \tag{2.5}$$

Where the error term δ_j^l in hidden layer l is calculated as shown below, where n represents a node in layer $l + 1$:

$$\delta_j^l = \sigma'(O_j^l) \sum_{n \in l+1} \delta_n^{l+1} W_{jn}^{l+1} \tag{2.6}$$

The weights of the system can then be updated by adding the partial derivatives to the current weights. Often, this change in weights is controlled for by some learning rate η, causing the weights to be updated as follows:

$$W_i j^l = W_i j^l + \eta \frac{\partial E}{\partial W_i j^l} \tag{2.7}$$

The biases of the nodes within the network are updated based solely on the error term of that node:

$$\theta_j^l = \theta_j^l + \eta\delta_j^l \tag{2.8}$$

2.2.2 Gradient descent variations

There are three main variations of the gradient descent algorithm: stochastic gradient descent, batch gradient descent and mini-batch gradient descent. In stochastic gradient descent, the weights and biases are updated every time a new instance from the training dataset is presented to the network during the training phase. In batch gradient descent, on the other hand, all instances of the training dataset are presented to the network and their errors are stored and accumulated. In this variation, the weights and biases of the network are only updated after all of the instances have been presented to the network. Batch gradient descent converges faster and more steadily than stochastic gradient descent and is guaranteed to converge to at least a local minimum. Stochastic gradient descent, however, is more likely to converge to a global minimum, since variation between the individual data points introduces a noise effect. In order to achieve the best of both worlds, the mini-batch approach attempts to combine the advantages of the stochastic- and batch gradient descent algorithms. In a mini-batch algorithm the training data is split up into smaller batches which are presented to the system in the same way as in a regular batch gradient descent algorithm. These batches have less variance than a stochastic descent, thus converging more steadily, but more than a full batch approach, which increases the probability of converging to the global minima.

2.2.3 Activation Functions

As described in equation 2.1, the output of a node is calculated using an activation function $\sigma(x)$. This activation function defines the output of a node based on its input. There are a number of different activation functions that can be used in a neural network, each using a different method for calculating the output.

Sigmoid

The original activation function used in the first neural networks is the logistic- or sigmoid function. This function maps the output of the neuron to a number between 0 and 1 using the following formula:

$$\sigma(x) = \frac{1}{1+e^{-1}} \tag{2.9}$$

A plot of this function can be seen in Figure 2.2A This function was initially used, as its values can be linked to the firing of a biological neuron, where 1 represents a fully firing neuron and 0 represents no firing at all. However, it comes with a number of problems. First of all, the function is always positive and not centred around 0. This means that the gradients of the weights during back propagation are either all positive or all negative, which can cause the change in gradients to become too large in a particular direction. Secondly, there is almost no change in output value if x is larger than around 3 or when x is smaller than around -3. This means that the gradient of values in that range will be extremely small, causing the network to barely learn or not at all. This is referred to as the vanishing gradient problem (Hochreiter, 1998).

Hyperbolic Tangent

Another activation function is the hyperbolic tangent function, or tanh, which is expressed as follows:

$$\sigma(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.10}$$

One obvious improvement of this function over the sigmoid function is that it is zero centred, which can be seen in Figure 2.2B. The range of the function is from -1 to 1, thus solving the first issue of the sigmoid function. The vanishing gradient problem, however, is still present with this function as well.

ReLU

Currently, the most widely used activation function is the Rectified Linear Unit function (ReLU) (Nair and Hinton, 2010). This function is defined as follows:

$$\sigma(x) = max(0, x) \tag{2.11}$$

This creates quite a different graph than the previous two activation functions as seen in Figure 2.2C. The value is simply 0 if x is less than 0, and equal to x if x is more than 0. This means that not all neurons are always activated, because if the input is negative, the output will be zero. This creates a sparser network, which is more efficient and can be computed more quickly. One of the major issues of the ReLU functions is the problem of 'dead neurons'; if all of the inputs of a neuron are less than zero, the gradient will be zero and the weights will no longer be updated, thus causing a 'dead' neuron. To fix this issue, a variant of the ReLU function was created, called a Leaky ReLU (Maas, Hannun, and Ng, 2013). Instead of setting the output of negative values of x to zero, a leaky ReLU sets the output to αx, where α is a small positive number. This creates a graph as seen in Figure 2.2D. This variation of ReLU does not create the zero gradients that occur in regular ReLU networks. The value of α can be a fixed number, or regarded as a parameter that is optimised during the training phase.

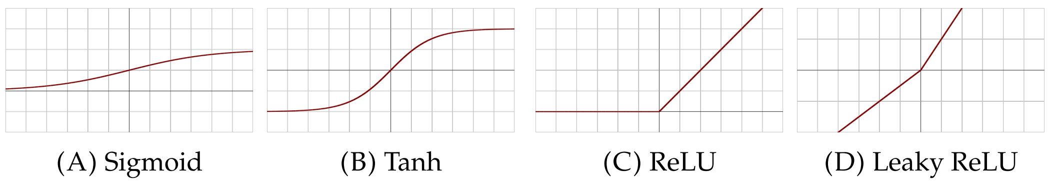

(A) Sigmoid (B) Tanh (C) ReLU (D) Leaky ReLU

FIGURE 2.2: Graphs of various activation functions.

There are more variations on each of these activation functions, however, variations of ReLU are currently the most widely used activation functions due to their small computation time.

2.3 Symbolic Learning Techniques

The other approach to classification systems is the symbolic approach. The main difference between the symbolic and the connectionist approach is that the models of symbolic learning techniques are more comprehensible to humans. This is unlike neural networks, whose models are not easily interpreted. Generally, symbolic learning techniques attempt to find patterns in data using data mining, and translate these patterns into classification rules. These classification rules can then be used to classify new, unseen instances. Two common symbolic learning techniques will be discussed in the next sections.

2.3.1 Association Rules

One symbolic data mining technique is association rule learning (Agrawal, Imieliński, and Swami, 1993). This algorithm identifies patterns between variables in a dataset which it uses to generate rules. It was originally used to analyse the buying behaviour of consumers, where rules were generated that could predict the products that consumers would buy based on their previous purchases. An example of such a rule would be as follows:

$$[A] \rightarrow [B](c, s) \tag{2.12}$$

Such a rule is read as *'if A is true, then B is likely to be true as well'*. Variables c and s represent the confidence and support of the rule respectively, both of which indicate the significance of the rule.

Support and Confidence

In a dataset, the support of an item denotes how often that item occurs in the dataset. For instance, if item A occurs 5 times in a dataset with 10 items, $Support(A)$ will be 50%. The confidence of a rule displays how often the consequent is true if the antecedent is true in the dataset. It is expressed as follows:

$$Confidence(A \rightarrow B) = \frac{Support(A, B)}{Support(A)} \tag{2.13}$$

This value is used to determine how good or interesting a rule is, where a higher value indicates a better rule. One of the major issues with confidence is that the measure does not control for the amount of times that the consequent B occurs in the datasets. There are therefore some other measures of performance that can be used instead of the confidence measure.

Lift and Conviction

Lift is one of those variations on the confidence measure (Brin et al., 1997). It also describes how likely it is that the consequent is true assuming that the antecedent is true, but it controls for the amount of times that the conclusion occurs in the dataset as well:

$$Lift(A \rightarrow B) = \frac{Support(A, B)}{Support(A) * Support(B)} \tag{2.14}$$

If the lift value is larger than 1, it is likely that the consequent is true if the antecedent is true. If it is smaller than 1 it is unlikely that the consequent is true if the antecedent is true. Conviction is another measure of interest. It represents the expected amount of times that that the antecedent is true, but the consequence is not. It is expressed as follows:

$$Conviction(A \rightarrow B) = \frac{1 - Support(B)}{1 - Confidence(A \rightarrow B)} \tag{2.15}$$

A conviction rating of 1.3, for example, would signal that the rule would be incorrect 30% more often if there was no relation between the antecedent and consequent. Using these concepts, rules from a dataset can be evaluated and good rules can be used to classify new instances. However, generating and evaluating all possible rules from a dataset is too computationally expensive, so algorithms to reduce the number of possible rules have been developed.

Apriori algorithm

The Apriori algorithm is the most commonly used algorithm for association rule mining (Agrawal and Srikant, 1994). It reduces the number of item combinations based on the apriori principle: if a set of items does not occur often in the data set, all possible sets of items that include that set do not occur often either. Whether or not an item or item set occurs 'often enough' is determined by a predetermined support threshold. The algorithms starts off by examining all of the individual items and evaluating their support value. If this value is below the threshold, the item is removed. For the remaining items, all possible sets of two items are made and the support value of each set is evaluated. If these values are below the threshold, the pairs are removed and new combinations of three items are made. This is repeated until a single item set remains, out of which the association rules can be generated. For each possible rule, the performance measure (confidence, lift, or conviction) can be calculated and based on a performance threshold, the 'good' rules will be saved. During the creation of these rules, the apriori principle can be used again, to reduce the number of rules that will need to be examined: If a rule such as 2.12 has a low confidence, for example, rules such as $[A] \rightarrow [B, C]$ can be excluded, as it will have a low confidence as well.

One of the main issues with the apriori algorithm, however, is that the support threshold needs to be lowered in datasets with a large variety of items. This, in turn, increases the probability of spurious associations to be included in the final set of items. Additionally, even though the apriori algorithm skips over a large number of items and rules, it is still quite computationally expensive.

Classification Association Rules

In order to use association rules for the classification problem, the Classification Based on Association algorithm can be used (Ma and Liu, 1998). This algorithms consists of two parts; finding the Classification Association Rules (CARs), and building the classifier. To find the CARs, a modified version of the apriori algorithm is used. The main difference is that, with CARs, the consequent of each rule is always the class attribute of an instance. To build the classifier using the CARs, the rules are ranked based on their confidence and support. Starting with the rule with the highest confidence, each rule will be used to classify all instances from a training set. If an instance was correctly classified, the rule will be marked and the instance that it classified correctly will be removed from the dataset. If the rule is marked, it will be appended to the end of the classifier. The default class of the classifier will then be changed to the majority class of the remaining dataset, and the new classifier will be used to classify the entire dataset. The error of this classification will be stored on the index of the current rule. This is repeated for every CAR that was generated. Once every rule has been presented to the system, the first rule in the classifier with the lowest total number of errors will be found. Every rule that was appended to the classifier after this rule will be discarded, as they only caused more errors. Finally, a new default class will be computed for the final classifier.

One general limitation of association rule mining, is that it cannot handle continuous variables. Therefore, continuous attributes of datasets need to be discretized in order for the association rule algorithm to work. In cases where discretization is not an option, it is best to use a different symbolic learning algorithm, such as decision trees.

2.3.2 Decision Trees

Decision tree algorithms are another type of symbolic supervised learning technique (Rokach and Maimon, 2014). The goal of these algorithms is to generate a decision tree that can be used to predict the the output of a given input. A decision tree is a type of directed graph that displays decisions and their consequences, where each node represents a decision and leaf nodes represent the consequence. Just as with association rules, the models that decision tree algorithms generate (the decision trees) are easily comprehensible by humans. A key difference with association rules is that decision trees can handle both discrete and continuous variables. It can therefore also be used for both regression and classification tasks. However, for the intends of this study, only the latter type of decision tree algorithms will be discussed.

To generate a decision tree, a decision tree learning algorithm trains on a training data set. There are are a number of different variations on the tree building algorithms, but most of them share the general structure of the following algorithm. A decision tree is created from top to bottom, starting with the root node that encompasses the entire data set. The aim is to split this set into two subsets based on the value of a particular attribute, such that the maximum level of homogeneity in terms of classes within the subsets is achieved. This homogeneity is based on a performance metric, which will be discussed in the next section. Each subset will be a new node in the tree, and these sets will be split up recursively in the same way into more subsets. If a subset consists of instances of only one class, or if splitting the set will not increase the homogeneity level, it will become a leaf node. This creates a decision tree in which the leaf nodes represent a class of the data.

Gini Impurity and Information Gain

One of the more commonly used performance metrics to decide on how to split a node in a decision tree are the Gini Impurity, as used in the CART algorithm (Breiman et al., 1984), and the Information Gain, as used in the C4.5 and C5.0 algorithms (Kuhn and Johnson, 2013). First off, the Gini impurity of a set of instances denotes the chance of incorrectly selecting a class for a random instance based on the distribution of class labels in the set. For each instance i in the set, its probability of randomly being chosen p_1 is multiplied by its chance of being mislabelled. The sum of these values will be the Gini Impurity. For set E with j amount of classes, the Gini impurity can thus be calculated as follows:

$$G(E) = \sum_{i=0}^{j} 1 - p_i^2 \tag{2.16}$$

When deciding on how to split a node into two subsets, the option that results in the largest decrease in the the Gini impurity will be selected.

Information gain is based on the idea of entropy. The entropy of a set denotes the homogeneity of the set, rather than the impurity. The entropy of set E with a total number of j classes can be calculated like this:

$$H(E) = -\sum_{i=0}^{j} p_i log_2 p_i \tag{2.17}$$

The information gain is simply the difference in entropy between two nodes. In a tree learning algorithm, the splitting of a node that creates the largest increase in information gain will be chosen. Despite their slightly different formulae, there is no significant difference in performance between trees generated using the Gini Impurity measure and trees generated using the information gain measure (Raileanu and Stoffel, 2004).

Bagging, Boosting and random Forest

A major issue in decision tree algorithms is that the models map the entire training data, rather than generalizing. This can create a huge, overly complex tree that is overfitted to the data. Most decision tree algorithms will therefore prune their final tree models, meaning that they remove certain nodes in order to keep the model more general. The goal of pruning is to remove nodes without impacting the performance of a system, but even then, overfitting can still be a problem. In order to solve this issue, an ensemble technique called bootstrapped aggregated, or bagged decision trees is often used (Breiman, 1996). Bagged decision tree algorithms create a number of 'bags' that contain random instances from the training data set. These bags are used to each generate a decision tree. All of the resulting decision trees are then used as an ensemble to classify a new unseen instance. The mode of the output of all of the trees is used as the final predicted class. This method prevents overfitting and can often increases the accuracy of the classifier.

Another way to improve the classification accuracy of decision trees is through a method called boosting (Schapire and Singer, 1999). This is an ensemble method, similar to bagging, where multiple decision trees are generated that vote on an output for new instances. In boosting, however, the training of the decision trees is different. The first bag is created using random instances from the training data set, after which a decision tree is generated from that bag. The entire training set is then used to measure the performance of the current system, which at that point consists of only the single tree. Each instance in the training set is then given a weight, depending on how well the current system was able to classify said instance. When the next bag is created, it is filled with random instances, but such that instances that were previously classified incorrectly have a higher chance of being present in this new bag. The system then generates a new decision tree from that bag and classifies the entire training set again on the system, which now includes two decision trees. This repeats itself until a predetermined amount of bags have been created.

Random forest is another ensemble technique based on bagging, and is currently still one of the most widely used machine learning techniques (Ho, 1995). The random forest algorithm will also create a number of bags of data, but during the creation of a decision tree from a bag, only a random subset of the attributes of the instances are used. The reason for this is that in classical bagging methods the same strong predicting attributes will be used in almost every decision tree, thus creation correlations between the trees of the ensemble. By giving each tree in the ensemble both a different training set and different attributes, this issue is largely avoided and a higher classification accuracy is obtained (Ho, 2002).

2.4 Soundness of the Rationales

Both the connectionist approach and the symbolic approach can yield high accuracies in classification tasks, but deep neural networks currently dominate in tasks such as image and speech recognition. Neural networks, however, are generally regarded as black box systems, where only the input and output are considered important. There is no easily understandable meaning to the set of weights that a neural network produces, making it difficult to explain why a neural network makes a certain classification. An explanation for a classification is often desired and in fields such as law or medicine, an explanation is even required. A 1993 study set out to investigate how well neural networks could train on open texture problems in law, and whether or not the rationale that the network learns is acceptable (Bench-Capon, 1993). In this study, a fictional dataset was generated using a set of predefined rules. Three different neural networks were trained on this dataset, and their classification accuracy was around 99%, which is a very acceptable performance. Investigating the network, however, showed that the rationales behind the classification, or why the neural network made certain decisions, were not always sound; the 'rules' that the network learned did not match the initial rules that were used to generate the dataset. Most of the simple boolean rules were easily discovered by the network, but rules based on spurious correlation with noise attributes were wrongly discovered by the network as well. Especially complex rules based on a combination of two or more attributes proofed to be impossible for the network to learn. The study therefore concluded that a high performance of a classifier does not prove that the rationale of the classifier is valid.

2.4.1 Analysing Neural Networks

Extracting the classification rationale from a neural network is not a straightforward tasks. The model that a network produces consists of a layered graph of weights which needs to be analysed in some fashion in order to discover the rationale. This rationale should be a set of rules that describe the classification process and is understandable to humans. The study by Bench-Capon that was discussed earlier analyses the network by training it in an inverted manner, such that the input attributes are treated as the output and the output class is treated as the input. The new output is then a list of weights of the attributes of the instances, ranging from 0 to 1. A neutral attribute that does not contribute anything in the classification would have a weight of 0.5. Based on this idea, it is possible to infer which attributes contribute to the classification, however, not a lot more can be determined through this method. More advanced techniques have been created since then, which can broadly be categorised into three approaches: the decompositional approach, the pedagogical approach and the eclectic approach (Hailesilassie, 2016).

The decompositional approach first creates rules based on individual nodes in the network, which are then aggregated later on. Using the decompositional approach, the output Y of node j, is expressed as a rule as follows:

$$\text{IF } O_j \geq \alpha \text{ THEN } Y = 1 \text{ ELSE } Y = 0 \tag{2.18}$$

Where O_j is output of the node as calculated using Equation 2.1 and α is a threshold value. Pedagogical approaches still view the system as a black box and do not analyse the internals of system. Instead, it tries to find rules that encompass the classification system as a whole rather than examining its individual parts. Eclectic approaches attempt to combine the two former approaches, for example by clustering the hidden nodes and generating rules based on the input and output of the cluster (Hruschka and Ebecken, 2006). A variety of algorithms have emerged from each approach. These algorithms can model the rules that they discover as a set of basic IF-THEN rules or create a decision tree. Modern rule extraction algorithms are able to accurately extract the right rules from a network, such that using the rules adequately reflect the internal working of the network. However, studying the rules often shows that the rationale behind the classifications are not completely valid, which is especially true for complex rules (Lu, Setiono, and Liu, 1996).

2.4.2 Analysing Symbolic Learning Techniques

By design, models generated using symbolic learning techniques are much easier to analyse and interpret. The rule mining techniques that were discussed produce IF-THEN rules such as 2.12, which can be understood without any difficulty. Similarly, decision trees are intuitive structures that make it easy for humans to follow the decision making processes. The main disadvantage of symbolic learning techniques is that the current connectionist techniques simply outperform it in terms of classification accuracy. Besides that, no association rule algorithm has yet been developed that can effectively deal with data with continuous attributes. Decision trees can handle continuous data, but they are prone to overfitting. The proposed solutions, ensemble learning techniques, decrease overfitting and increase the classification accuracy, but at the cost of making the rationale behind the classifications more complex. Through analysing symbolic learning techniques, it has been discovered that in these technique spurious correlations can cause the system to learn invalid rules as well (Webb, 2007).

Chapter 3

Bench-Capon Replication Study

In this chapter, the study that was performed by Bench-Capon in 1993 will be discussed in more detail (Bench-Capon, 1993). Additionally, the study was replicated as accurately as possible and the results of that repeated study are reported here as well. The results of both studies will be compared and discussed.

3.1 Neural Networks and Open Texture

In his paper 'Neural Networks and Open Texture', Bench-Capon aims to investigate three aspects of neural networks with regards to classification problems in law. First of all, he wants to know whether neural networks can achieve a high performance on open texture law problems. The second aspect is to discover whether or not the rationale that the neural network uses in its classification is acceptable. Lastly, Bench-Capon investigates whether it is possible to derive rules form the neural network by examining its weights.

To accomplish this goal, he creates a fictional database that is used to train the network on. The main reason for choosing a fictional dataset is that there is no empirical method of testing the rationale of a neural network trained on a real database. The rules of a fictional dataset, on the other hand, are known beforehand, thus making it possible to compare it to the rationale that the network produces. The database in question contains a number of cases, where each case represents a person who may or may not be eligible to receive a welfare benefit. It is generated from a LISP script based on 6 conditions that determine whether a person receives the benefits, as shown below:

1. The person should be of pensionable age (60 for a woman, 65 for a man).
2. The person should have paid contributions in four out of the last five relevant contribution years.
3. The person should be a spouse of the patient.
4. The person should not be absent from the UK.
5. The person should have capital resources not amounting to more than £3,000.
6. If the relative is an in-patient the hospital should be within a certain distance: if an out-patient, beyond that distance.

Only if all of the conditions apply to a person will he or she receives benefits; if one of these conditions does not apply to the individual, he or she is not eligible for a welfare benefit. Whether a person is eligible for a benefit or not, is therefore dependent on 8 variables: gender, age, how many contributions were paid, whether he or she is the spouse of the patient, whether he or she resides in the UK, his or her capital resources, what type of patient the relative is, and how far away he or she lives from the patient.

Using these conditions, 2400 cases are generated, where half of them are eligible and the other half are ineligible. When a case is eligible, the values of the variables are distributed randomly across the range of values that the conditions describe as satisfactory. When the case was ineligible, the cases are generated such that they fail on each condition equally. The rest of the variable of unsatisfactory cases are then generated entirely randomly. Additionally, to investigate the effects of noise, 52 noise variables are included in the dataset that did not influence whether the case was satisfactory or not. There are a total of 64 input features, and one output feature (eligible or ineligible). This welfare dataset was also used later on in different studies, in agent dialogue settings wherein two agents debate with each other in order to solve a classification task (Wardeh, Bench-Capon, and Coenen, 2009a) (Wardeh, Bench-Capon, and Coenen, 2009b).

Three neural networks are used in the study, with 1, 2 and 3 hidden layers. The one hidden layer network had 12 nodes in its hidden layer, the two hidden layer network had 24 and 6 hidden nodes, and the three hidden layer network had 24, 10 and 3 nodes in its hidden layers. The networks thus follow the default triangular shape, where each succeeding hidden layer has less nodes that its preceding layer. These networks are implemented in Aspirin (Leighton, **aspirin**), which is a software package for creating neural networks.

After training each network on the databases, 2000 new test cases are generated and used to determine the performance of each classifier. The resulting accuracies are quite high, reaching 99.25%, 98.90% and 98.75% accuracy for the one, two and three layer network respectively. To analyse the rationale, a new test set is generated in which all of the features are satisfied except for age. The age feature is distributed over this dataset from 0 to 100 in steps of 5. This makes it possible to create Figure 3.1, where the age is plotted against the output, where 1 represents eligible and 0 represents ineligible. According to the first condition, the expected output of this graph would show a 1 for men after 65 and for women after 60, and a 0 for all other ages. However, the networks with one and two hidden layers almost always provide a 1 as their output, and the outputs of the network with three hidden layers are completely off: the difference between men and women seems to be almost 15 years, and both are provided a satisfied output at an age that is much lower than it should be. A similar graph could be made for the last condition (the patient type/distance condition), but the networks always yields a satisfied output regardless of the distance or patient type.

Bench-Capon argues that the high accuracies can be achieved with a classifier that completely ignores some features due to probabilities inherent to the dataset. The probability of a condition being unsatisfied is statistically higher if another condition is also unsatisfied. For instance, if only four conditions were used to determine the output, they would accurately classify the cases that were explicitly designed for fail on those conditions. For 1200 failing cases, that means that there are $4 * 200 = 800$ cases that are classified correctly with just four conditions. Additionally, half of the

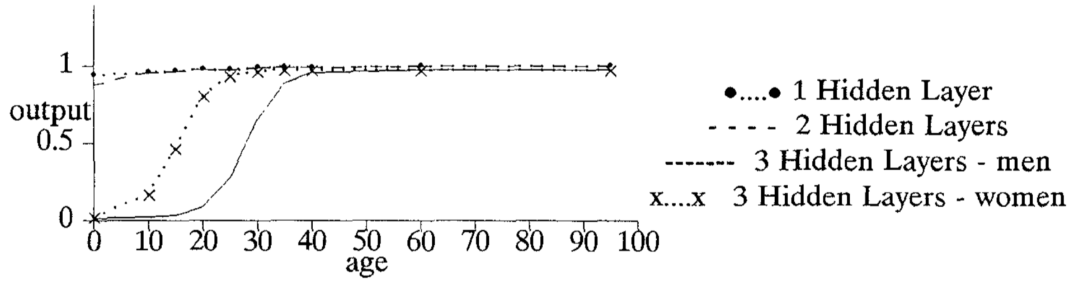


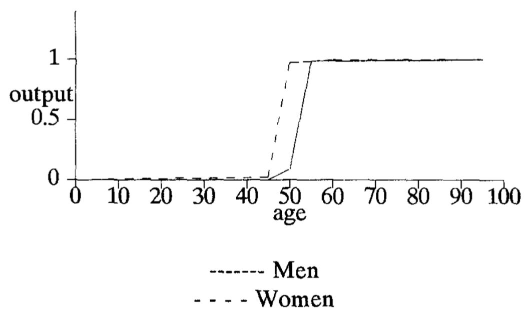
FIGURE 3.1: The output of the networks for cases where every case is satisfied except for age, versus the age (Bench-Capon, 1993).

remaining dataset will be guessed correctly as well, since the problem is binary. That means that $800 + 200 + 100 + 50 + 25 = 1175$ of the 1200 (97.91%) incorrect cases will be classified correctly with only four conditions. When the eligible cases are correctly classified as well, this results in a total accuracy of 98.95%, which is close to the achieved performance. Testing this network using a database in which each ineligible case failed on exactly one condition proved this, as performance decreased to 72.25-74.33%.

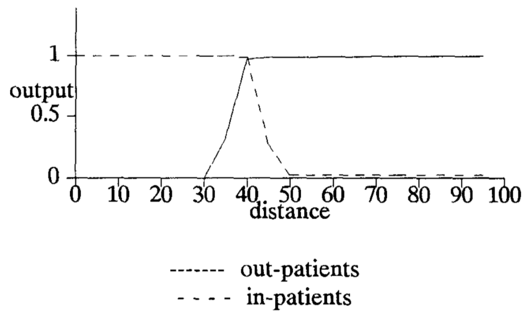
The network is retrained using the a training data set in which the ineligible cases failed on exactly one condition. When testing these networks using the same test dataset as before, similar results are obtained for the one and two layered neural network. The network with three layers never reaches accuracies over 80% when training on this new training set, and is thus discarded from the rest of the study. Testing the other two networks on a the new test set where ineligible cases fail on a single condition only yields accuracies around 98%, which is significantly higher than the 74% that is obtained when training on the initial data set. The same age versus output experiment was performed on these new networks, providing the graph as seen in Figure 3.2A. In this graph, the correct age difference of 5 years can be seen between men and women, though the output is still considered as eligible too soon (at around 45 for females and 50 for men). A much steeper line at the point at which the output changes can be observed as well. The graph of distance versus output for both in- and out-patients can be seen in Figure 3.2B. The difference between the output for the type of patients should occur at the 50 miles point, but the network predicts it around 40 miles. There is also some overlap between the patient types, but this is a significant improvement on the networks trained on the initial database.

The performance accuracy between networks trained on the first and second database are almost identical, but in terms of rationale, the second dataset performs lot better. A performance metric like the classification accuracy does therefore not guarantee a sound rationale. Furthermore, the right distribution of data is key in obtaining an acceptable rationale. Creating such a dataset, however, is only possible if the rationale is already known beforehand. Bench-Capon therefore argues that we must understand the domain of the problem before placing confidence in the rationale of a neural network.

In order to determine whether rules can be derived from the net, Bench-Capon posed that it must be possible to determine which features are significant and what the significance of that feature is. To investigate the significance, the network is inverted,



(A) Age vs. output



(B) Distance vs. output

FIGURE 3.2: The results of the experiment using networks trained on the dataset where the unsatisfied cases failed on only one feature (Bench-Capon, 1993).

TABLE 3.1: The output of the reverse network of the features that deviated most from 0.5 (Bench-Capon, 1993).

Feature	Output
Spouse	0.995
Residence	0.984
Contribution 5	0.920
Capital	0.118
Contributions 1	0.125
Contributions 4	0.819
Contributions 2	0.191
Contributions 3	0.203
Age	0.779
Noise 8	0.776
Noise 16	0.720
Gender	0.354

such that the output (eligible or not) becomes the input, and the input features become the output. The new outputs, one for each variable, were then examined based on their deviation from the mean (0.5). The result is shown in Table 3.1. The boolean variables, such as the spouse from conditions 3 or the residence variable from condition 4 showed up clearly in these outputs, meaning that it was clear what their significance and contribution to the classification was. Mixed conditions, such as condition 1 and 6 cannot be represented accurately in this form. Patient type and distance, for instance, both yield outputs around 0.5. Bench-Capon therefore concluded that conditions that use a combination of variables are difficult to discover and that an analysis of the net is in most cases only useful in trying to confirm a predefined hypothesis.

3.2 Replicating the study

The study by Bench-Capon raises a few interesting points, which is why the study was repeated in this current study. It will be replicated as closely as possible, with the aim of yielding similar results. Not all of the specific information needed to replicate the study exactly were present in the paper, therefore some design choices

TABLE 3.2: The features of the dataset

Feature	Range of values
Gender	*male* or *female*
Age	0-100
Paid contributions	1-5
Spouse	*True* or *False*
Residence	*True* or *False*
Capital	0-10,000
Patient type	*in* or *out*
Distance	0-100

had to be made for this study. Whenever possible, a number of alternatives and their results will be presented.

3.2.1 Datasets

The datasets in the original study were generated using a LISP program using the six condition as described in the previous section. These conditions consists out of 12 variables, which can be seen in Table 3.2 along with the possible range of values that each variable can have.

With these features, each of the individual conditions that determine whether a person is eligible for a welfare benefit can be expressed logically as follows:

C1. (Gender = *'male'* $\wedge$ Age >= 65) $\vee$ (Gender = *'female'* $\wedge$ Age >= 60)

C2. Paid_contributions >= 4

C3. Spouse = True

C4. Residence = True

C5. Capital < 3000

C6. (Patient_type = *'in'* $\wedge$ Distance < 50) $\vee$ (Patient_type = *'out'* $\wedge$ Distance >= 50)

Since all of the conditions need to be applicable in order for the person to be eligible for a benefit, the eligibility can be expressed as an implication of the conjunction of C1-C6:

$$(\text{C1} \wedge \text{C2} \wedge \text{C3} \wedge \text{C4} \wedge \text{C5} \wedge \text{C6}) \rightarrow \text{Eligible}.$$

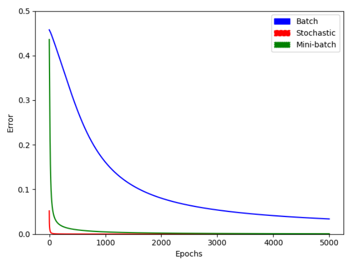
This means that even if only one of the conditions is not applicable, while the other conditions are applicable, the person will still not be eligible for a welfare benefit. Using a Python script, a number of different versions of the dataset are generated. Every instance in a dataset represents a single person and contains all of their features (as seen in Table 3.2 as well as whether the person is eligible for the welfare benefit based on these features and conditions C1-C6. Additionally, 52 noise features were added to each instance, with random values ranging from 0 to 100. Every instance therefore has 60 features. In total, 6 different datasets are created:

- Training set A: A set of 2400 instances, where half of the instances are eligible for welfare benefits and the other half are not eligible. Instances that are ineligible are given values such that at least one of the six conditions C1-C6 in particular is not applicable. The other features of these instances are given a random value across its range as shown in Table 3.2.
- Training set B: A set of 2400 instances, where half of the instances are eligible for welfare benefits and the other half are not eligible. Instances that are ineligible are given feature such that exactly one of the six conditions C1-C6 is not applicable. The other features of these instance are given values such that the rest of the conditions are applicable.
- Test set A: A set of 2000 instances, generated in the same fashion as Training set A.
- Test set B: A set of 2000 instances, generated in the same fashion as Training set B.
- Age experiment set: a set of 2000 cases, where the Gender feature of half of the cases is 'male' and the other half is 'female'. The values of the Age feature are varied between 0 and 100 in steps of 5. The other features of the instances are given values such that the remaining conditions are applicable.
- Distance experiment set: a set of 2000 cases, where the Patient type feature of half of the cases is in and the other half is out. The values of the Distance feature are varied between 0 and 100 in steps of 5. The other features of the instances are given values such that the remaining conditions are applicable.

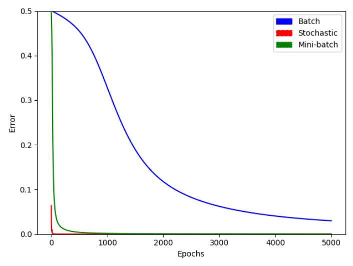
3.2.2 Neural networks

The neural networks used in the study by Bench-Capon were created in the simulation environment of Aspirin, so not much is known about the details of each network. However, the three networks that were made in this study share all of the characteristics that were specified in the original paper. The three networks therefore had one, two, and three hidden layers respectively. Network 1 has 12 nodes in its hidden layer, network 2 has 24 and 6 nodes, and network 3 had 24, 10 and 3 nodes in its hidden layers. It is unknown which activation function was used by the networks in the original study, but the most common activation function at that time was the sigmoid function (Ramachandran, Zoph, and Le, 2017), which is therefore what is used in this study as well. The type of gradient descent in the back-propagation phase is also unknown, so the use of batch, stochastic and mini-batch descent will be examined. The learning rate of the networks is set at a constant value of 0.05 throughout this section. When training the networks on training set A, the error rate over time graphs were generated for each type of gradient descent as seen in Figure 3.3.

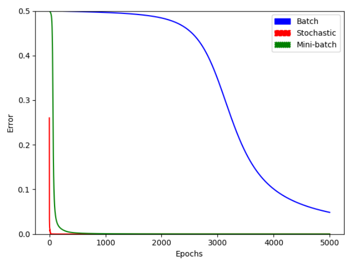
One epoch has passed when the entire training set has been shown to the network once. The error of one epoch is the average of the errors that are calculated once every time back propagation is executed in that epoch. What is clear to see, is that overall the error rate decreases at a much slower pace when using the batch gradient descent method, and for network 3 the system only starts learning at around 2000 epochs for this method of descent. The error rate for both the stochastic and mini-batch approach decreases much faster. The error rate of the stochastic approaches



(A) Network 1



(B) Network 2

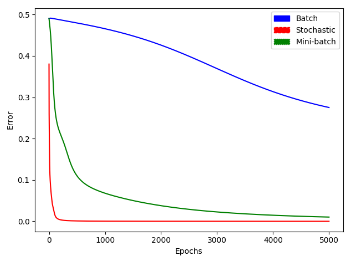


(C) Network 3

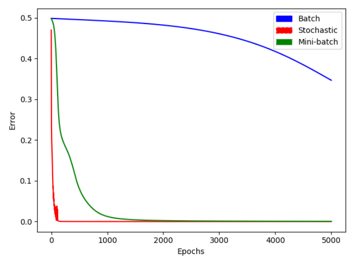
FIGURE 3.3: The errors per training epoch for batch, stochastic and mini-batch descent for each network while training on training set A.

decreases very steeply and reaches its minimum value after only a few epochs. The mini-batch approach produces error rates that decrease a bit more slowly than the stochastic approach, but still much more quickly than the default batch approach. It should be noted that epochs do not indicate how fast a network is trained in terms of time. Batch training, for example, only uses backpropagation once every epoch (at the end), whereas backpropagation occurs after every instance of the training set for a stochastic approach. A stochastic epoch will therefore take significantly longer than a batch epoch. This is also the reason as to why the networks that use a stochastic approach start off with a lower error rate.

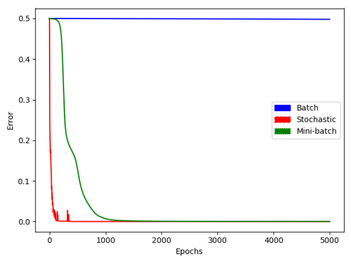
The networks trained on training set B showed similar results for each form of gradient descent, as seen in Figure 3.4 but took a lot more epochs to converge, if they converged at all within the epoch range. The plots in Figure 3.4 show the error rate over time for 5,000 epochs, the same amount as in Figure 3.3. The batch descent approach converged at around 20 thousand epochs.



(A) Network 1



(B) Network 2



(C) Network 3

FIGURE 3.4: The errors per training epoch for batch, stochastic and mini-batch descent for each network while training on training set B.

3.2.3 Experiments and results

Three different experiments are reported by Bench-Capon in his study. The performance of networks on several test sets, the age-gender and distance-patient type conditions, and the weights of the network are examined. These experiments will be repeated, and their results will be discussed and compared to the original paper in the upcoming sections.

Performance of the networks

After training the networks on training set A and training set b, their performance can be measured by letting it classify the four different test ests. To create an overview of the performances of the networks, Table 3.3 and Table 3.4 are created. There is hardly any difference in classification accuracy between the different types of gradient descent, therefore all of the results shown here are derived from networks trained using the mini-batch approach. In Table 3.3, the accuracies of networks trained on training set A are shown for each of the test sets. The networks are trained with a batch size of 50 for 1500 epochs. Training the networks for 5,000 epochs showed the same level of performance for each test set, which means that 1500 epochs are sufficient. The performance of the networks on test set A are very close to the performances yielded by the networks in the original study. Testing on test set B, however, yields accuracies that are around 5% higher. The performance of the networks on the age and distance experiment sets were not mentioned in the original paper. Testing on these sets show a reasonable performance for the age set, and lower accuracies for the distance set.

TABLE 3.3: The accuracy of each network trained on training set A for each test set.

	Test set A	Test set B	Age set	Distance set
1 layer	98.60	75.70	99.98	85.11
2 layers	98.70	77.25	99.96	85.51
3 layers	98.9	74.10	99.95	85.19

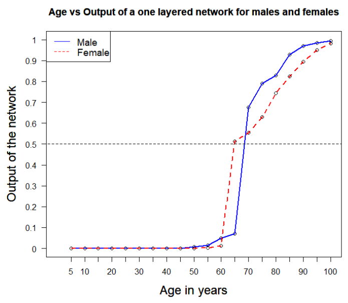
In Table 3.4 the performance of the networks trained on training set B are shown after training on 150 epochs. Initially, the networks were trained for 5,000 epochs, but this provides very poor results, especially on test set B. When increasing the amount of epochs to, 20-, 50- or even 100 thousand, the accuracy on test A remained stable at around 96%, whereas the accuracy on test set B decreased to almost 50%. Interestingly, when training the system for only 150 epochs, much better results were found, which are the results shown in 3.4. The accuracies on test set A are similar to the networks trained on training set A and the results found by Bench-Capon. The performances on test set B are higher than those of the networks trained on training set A, but not as high as the 98% of the initial study. Performance on the age set is also much lower than the ones yielded by the training set A networks, and the accuracies of the distance set are quite a bit lower as well.

TABLE 3.4: The accuracy of each network trained on training set B for each test set.

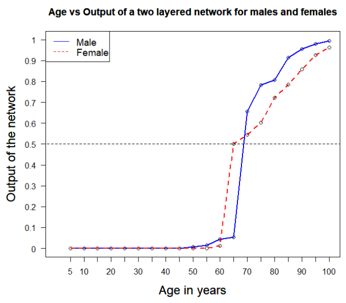
	Test set A	Test set B	Age set	Distance set
1 layer	97.45	81.80	78.7	70.95
2 layers	97.80	81.15	79.3	71.9
3 layers	98.5	82.65	80.45	72.75

Age and Distance Experiment

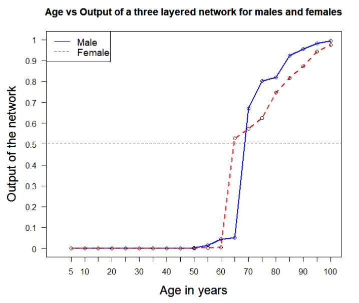
The outputs of the three networks trained on training set A on the age experiment set were used to produce the graphs as seen in Figure 3.5. These graphs plot the age of the person against the average output of the network for that person. An output above 0.5 is considered eligible, whereas an output below 0.5 is considered ineligible. For each of the three networks, both the female and male conditions are learned correctly: males below the age of 65 and females below 60 are given an output lower than 0.5, and males of 65 and above, and females of 60 and above are given an output above 0.5. The outputs for females of the age of 65 is very close but just above 0.5. This is in contrast with the original study, as the initial networks of the original study were not able to learn these conditions, as seen in Figure 3.1. Another difference is that the output is not 1.0 for all eligible age/gender combinations, but slowly increases from 0.5 to 1.0 as the age increases. This, however, has no effect on the classification, as it is a binary classification problem.



(A) Network 1



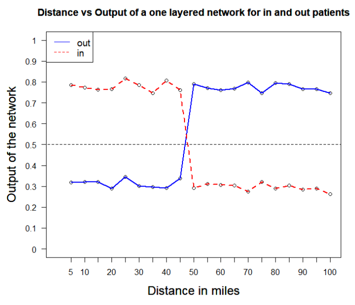
(B) Network 2



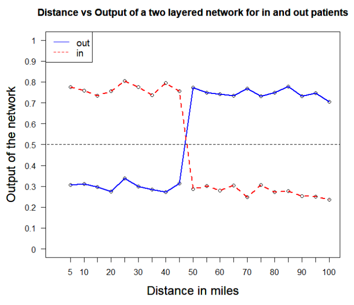
(C) Network 3

FIGURE 3.5: The age in years versus the output of the network trained on training set A for males (blue) and females (red).

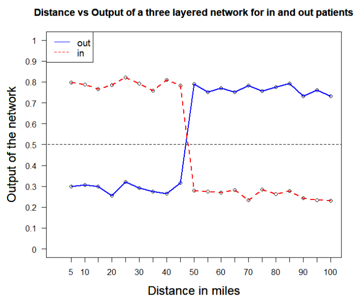
For the distance experiment, the output of the three networks trained on training set A on the distance experiment set are shown in Figure 3.6. Here, the distance is plotted in miles against the average output of the networks. Again, an output above 0.5 denotes an eligible case and an output below 0.5 denotes an ineligible case. Just as with the age experiment, all three of the networks seem to be able to accurately learn the conditions about distance and patient type: in-patients closer than 50 miles, and out-patients equal or further than 50 miles satisfy the condition, and in-patients equal or further than 50 miles and out-patients that are closer than 50 miles do not satisfy the condition. This, again, is in contrast with the original study, considering the fact that the networks in the original study were unable to learn this condition at all. Instead, those networks simply yield an eligible output regardless of the distance value and the patient type.



(A) Network 1



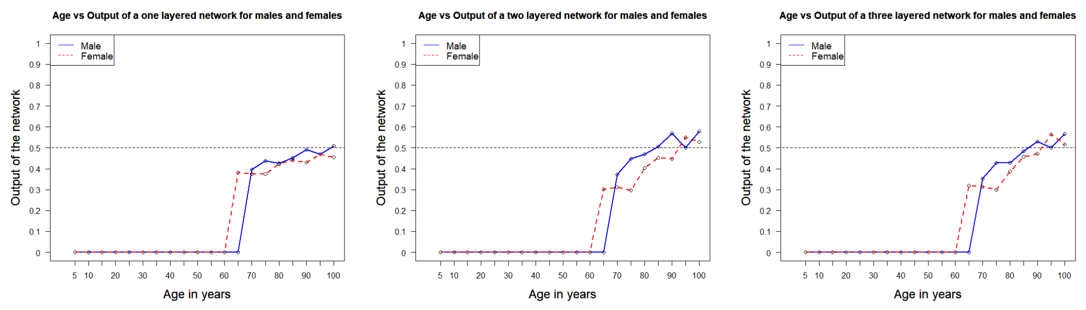
(B) Network 2



(C) Network 3

FIGURE 3.6: The distance in miles versus the output of the network trained on training set A for in patients (blue) and out patients (red).

In a similar fashion, the age and distance experiment are performed on the networks trained on training set B, which can be seen in Figure 3.7. At a first glance, the shape of the graphs appear to be quite similar to the ones in Figure 3.5, with the key difference being that these outputs never seem to go much higher than 0.5. This means that, according to these networks, a person is not always eligible if the correct gender/age values are provided. Interestingly, the networks does seem to discover the 5 year difference between male and females, and there is a steep increase in the outputs at 60 and 65 years. This increase in outputs, however, is not sufficient and thus an output level of 0.5 is only reached at around 85 years, dependent on the number of layers in the network. This means that only people of around 85 years or older will be classified as eligible according to these networks. This explains the lower accuracy that was yielded from the networks trained on training set B for the age experiment set.



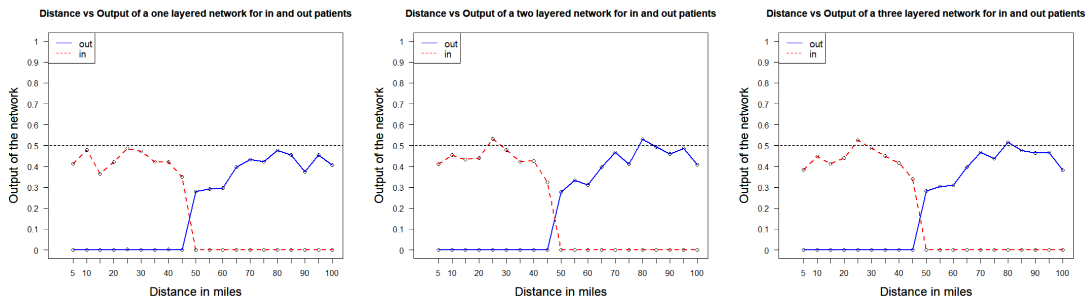
(A) Network 1 (B) Network 2 (C) Network 3

FIGURE 3.7: The age in years versus the output of the network trained on training set B for males (blue) and females (red).

In Figure 3.8 the graphs of the distance experiment are shown for the networks trained on training set B. Similarly to the previous age experiment, the shape of these graphs resemble the ones in Figure 3.8, but the outputs are much lower. This causes the same effects as discussed in the previous age experiment, where the conditions are considered unsatisfied, even when the distance/patient type combination is correct. This was not expected, as the networks trained on training set A were able to learn the correct condition, and the network in the original paper that was trained on the same type of output was able to learn it as well (as seen in Figure 3.2B). Since the output does not reach above 0.5 at points where it should, these graphs do explain the fact that these networks performed worse on the distance test set than the networks trained on training set A.

Examining the weights

To examine the weights, the three layered neural network was inverted, such that the output of the network is the 64 features, and the input is the single 'satisfied' attribute. This inverted network was trained on both training set A and training set B. When these trained networks are presented with a '1' as the input, the weights of each feature in the dataset will be the output. If a feature contributed nothing to the system, it is expected to have a weight of 0.5, since this would imply that it has a weight of 0.5 when presented with a '0' as well. If the feature had a positive impact on being satisfied, it will be higher than 0.5, and it will be lower than 0.5 if the feature had a negative impact. The features of both networks were sorted based on how much their output deviated from 0.5. The top 10 features and their outputs can be seen in Table 3.5.



(A) Network 1 (B) Network 2 (C) Network 3

FIGURE 3.8: The distance in miles versus the output of the network trained on training set B for in patients (blue) and out patients (red).

TABLE 3.5: The outputs of the reverse network trained on training set A (left) and training set B (right)

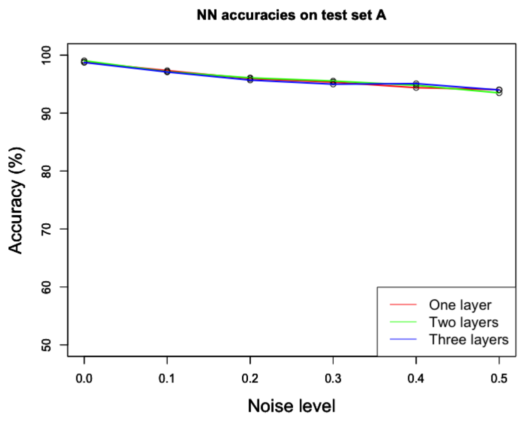
Feature	Output
Spouse	0.9998
Residence	0.9998
Contributions 2	0.0001
Contributions 3	0.0002
Contributions 1	0.0002
Capital	0.148
Age	0.809
Noise 41	0.475
Noise 36	0.476
Noise 24	0.478
gender	0.518

Feature	Output
Contributions 2	0.046
Contributions 1	0.055
Contributions 3	0.065
Spouse	0.832
Residence	0.832
Capital	0.240
Age	0.729
Contributions 4	0.403
Contributions 5	0.430
Noise 21	0.480
Noise 6	0.481

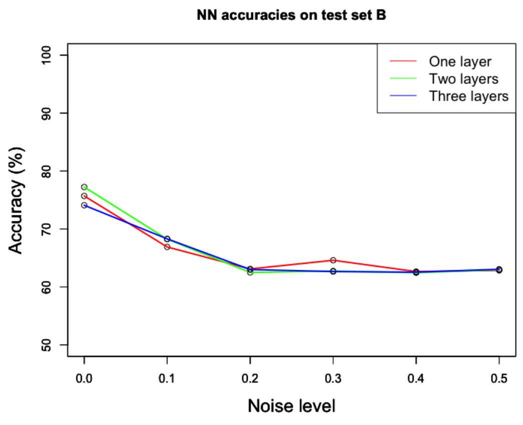
By looking at just the features themselves, we can see that the boolean values such as 'Spouse' and 'Residence' are spotted by both networks, just as in the original paper. For network A, the outputs are quite similar too, but they are much lower for network B. The first three contribution variables show a clear negative impact on network A and B with very low outputs, even lower than in the original study. This means that if one of the first three contribution variables is present, the output will most likely be 0, or ineligible. The effect of contribution 4 and contribution 5 can only be seen in network B, however, with lower output values than expected. Just as in the original paper, capital also has a clear negative impact on the output of both networks, with similar output values. Age has a positive influence in both networks and the output value is similar to that of the original experiment. The gender feature can also be seen in the table of network A, but with an output value that is very close to 0.5. Just as in the original paper, it is difficult to make sense of the more complex conditions of the dataset from just looking at the weights of the features alone. This method can therefore only be used with simple Boolean variables to determine what the influence of the variable is; a value higher than 0.5 means that the output is usually true if the variable is true, and when the value is lower than 0.5, the output is usually false if the variable is true .

3.3 The Influence of Noise on Neural Networks

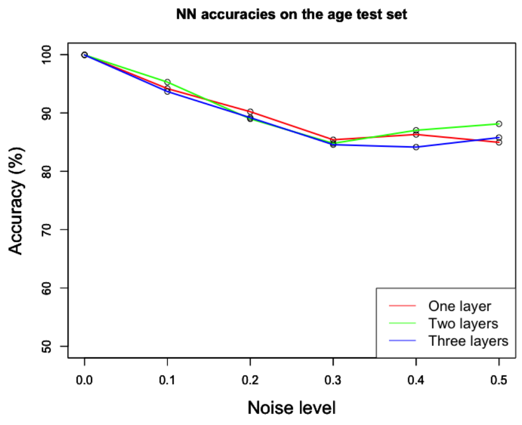
In his paper, Bench-Capon explains how he adds 52 noise variables to the dataset to determine whether or not the neural networks are able to learn rules in noisy environments. These noise variables do not contribute to the rules in any way, meaning that the networks should simply ignore them. In both the original study and this repeated study, the networks appear to be relatively successful in ignoring the noise variables. These additional noise variables, however, did not affect the rules that define the dataset. In a regular, non-artificial dataset, the underlying structure of the data is usually more noisy; there are slight deviations from the rules that define the output. For example, an input image can have a few black pixels from dirt on the camera, or someone can mistakenly enter the wrong number on a registration form. This sort of noise is very common in most real life datasets. To explore the effects that noise would have on the learning of the rules of the welfare dataset, varying degrees of noise are added to training dataset. The networks will then train on the noisy training data and afterwards classify the test sets.



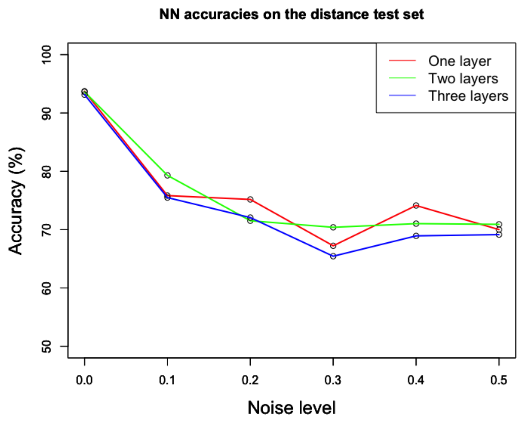
(A)



(B)



(C)



(D)

FIGURE 3.9: The effect of varying levels of noise on the accuracies on test set A (A), test set B (B), the age test set (C) and the distance test set (D) for networks with one, two and three hidden layers.

In this experiment, a certain percentage of the instances in the training dataset will receive a noisy variable, meaning that the value of that variable is inverted. The value of this variable will be adjusted to mimic the noise of a regular dataset. The variable is chosen randomly per instance, and its value will be set to the maximum value for that variable, minus the current value of the variable. For instance, a Boolean variable, such as residency, that is set to 0 will be set to 1 ($1 - 0 = 1$). A numerical variable, such as age, that is set to 75 will be set to 25, as the maximum age is 100 ($100 - 75 = 25$). The noise will be applied to training set A, test set A and test set B. The percentage of instances that receive a noisy variable is varied from 0 to 50%. This does not mean, however, that all of the instances with a noisy variable have an incorrect label. For example, a person who is ineligible for a welfare benefit due to the fact that they live outside of the UK, can be given a noisy variable such that his or her age is below the threshold of the age/gender condition. This would not influence whether or not the person receives a benefit, since he or she was ineligible already. The networks with one, two and three hidden layers are used, just as in the previous section. The accuracies on test set A, test set B, the age test set and the distance test set are shown in Figure 3.9.

First of all, there seems to be no significant differences in the performances of the networks with one, two or three hidden layers. They all follow the same trends and have minimal differences in accuracy. In Figure 3.9A, the accuracies on test set A drop steadily as the amount of noise in the training data increases, but never drops below 90%. The accuracies on test set B (3.9B) drop more rapidly with more noise, plateauing at 50% accuracy with 20% noise or more. This is because the instances in test set B that are not eligible for a welfare benefit fail on only one of the six conditions, whereas the instances of test set A that are ineligible can fail on multiple conditions at the same time. Therefore, if one of those instances in test set A contains a noisy variable of a particular condition that it should fail on, causing the condition not to fail, the label might still be correct because there are other conditions that can fail as well. In test B this is not the case, as each instance only fails on one condition. The accuracies on both the distance and age test sets are also lower as more noise is used in the training data, indicating that the networks have more difficulties in learning the conditions if the training data set is noisy. These results are to be expected, as the underlying structure of the training set is noisy and therefore more difficult to learn.

As stated earlier, real datasets almost always contain noise and their underlying structure is, more often than not, unknown. Incorporating noise into the artificial dataset therefore makes it more like a real life problem and thus allows the results to be more easily applied to real life scenarios as well. The results show that when only a few of the instances have a noisy variable, the accuracies on the age and distance test set drop quite a lot, indicating that the network has more trouble with learning the conditions. Still, the accuracy on test set A remains above 90%, even when half of the instances in the training set contain noise. It can therefore be concluded that the noise slightly influences the overall performance of a network and makes it so that the networks are able to learn the conditions less successfully.

3.4 Discussion and conclusion

When placing the results of Bench-Capon and the results of this study side by side, some clear differences and similarities can be seen. When it comes to the networks that are trained on training set A, the performance in terms of classification accuracy is quite similar to the findings of the initial study. The weights of these network as seen in Table 3.5 also roughly match the weights of the network of the original study. This is, however, where the similarities end. Based on the results shown in Figure 3.5 and 3.7, the networks of this study are able to learn the age and distance conditions almost perfectly, whereas their counterparts in the original study were less successful (see Figure 3.1). in the original paper, it was not speculated as to why the initial networks were unable to learn the conditions as accurately as desired, therefore it is difficult to understand why the networks in this study were able to learn the correct conditions.

The networks trained on training set B, however, showed very different results than in the original study. First of all, the optimal number of epochs required to yield the optimal performance was very low, at 150 epochs. This is quite strange, as the graphs in Figure 3.4 show that the errors of a mini-batch approach do not yet reach a plateau at 150 epochs. When training the networks on more epochs, up until 100,000, the network yielded lower classification accuracies of only 50% on test set B. The most likely explanation for this behaviour is that training set B is more prone to overfit the network to the training data, thus decreasing the performance on the B test set. By design, every ineligible case in training set B contains features that fail only one condition, such that the rest of the features do satisfy their respective conditions. The network is therefore presented with much more satisfied conditions than unsatisfied conditions, especially since half of the cases in the training data set are eligible and thus containing only satisfied conditions. One hypothesis could be that due to this fact, the network can more easily pick up on when cases are ineligible, because the values of the features in the unsatisfied range (values that do not satisfy their condition) appear only in ineligible cases. The eligible cases are more difficult to learn, as the values of features in the satisfied range (values that satisfy the condition) appear in the eligible cases, and 5 out of 6 times in the unsatisfied cases as well (only 1 of the 6 conditions is unsatisfied in training set B, thus causing the other 5 conditions to be satisfied).

The graphs in Figure 3.7 and Figure 3.8 support this hypothesis, as the ineligible cases provide an output of 0.0, and the eligible cases produce an output around 0.5. The average output of network B when testing on test set B for 10,000 epochs is only 0.173, even though half of the cases were eligible, and thus an average output of 0.5 would be expected. This confirms that the network has a clear bias towards unsatisfied cases on this test set. This accounts for the low classification accuracy (around 50%) on test set B when training the network for more epochs. When testing this network on test set A, accuracies of around 97% are achieved, which are close to the performances of network A. The average output when testing on test set A is 0.458, which is significantly higher than the 0.173 average when testing on test set B. Still, a bias towards the unsatisfied cases exists, which could be explained by the hypothesis. The results on test set A are, however, a lot better than the results on test set B, as seen in Table 3.4. A possible explanation for this performance is that the B networks learned to classify the eligible cases using only a few simple boolean conditions. According to Bench-Capon, those would be sufficient to achieve a high classification accuracy on test set A. In test set B, however, these conditions often have the same value ranges in both the eligible and ineligible cases, thus making it impossible to achieve a classification rate as high as on test set A.

This repeated experiment has shown some interesting results that do not completely concur with the original experiment. What can be said ,is that the way that the dataset is set up or generated has a large impact on both the performance and the rationale of the network. This study supports Bench-Capon's conclusion that a network can achieve a high performance on these type of databases and that it is difficult to determine the rationale of a network by examining its weights. Contrary to his conclusion, however, within the scope of the experiment the rationales of the networks appear to be sound.

Chapter 4

Symbolic Learning Methods

In the following sections, the study by Bench-Capon (Bench-Capon, 1993) as repeated in Chapter 2 is performed using symbolic learning methods. Specifically, decision trees and association rules are examined. Just as in the previous chapter, these methods are evaluated based on their classification accuracy and on how well they are able to learn the rules that define the dataset.

4.1 Datasets

The datasets used in this study are based on the fictional welfare benefit dataset of Bench-Capon (Bench-Capon, 1993) as described in detail in section 3.2.1. These dataset contain the personal information of a large number of individuals and whether or not these individuals are eligible for a welfare benefit. Whether someone is eligible is formally be defined as the following function:

$$(\text{C1} \wedge \text{C2} \wedge \text{C3} \wedge \text{C4} \wedge \text{C5} \wedge \text{C6}) \rightarrow \text{Eligible}.$$

Where C1-C6 are defined as follows:

C1. (Gender = *'male'* $\wedge$ Age >= 65) $\vee$ (Gender = *'female'* $\wedge$ Age >= 60)

C2. Paid_contributions >= 4

C3. Spouse = True

C4. Residence = True

C5. Capital < 3000

C6. (Patient_type = *'in'* $\wedge$ Distance < 50) $\vee$ (Patient_type = *'out'* $\wedge$ Distance >= 50)

The machine learning algorithms are trained on either training set A or training set B (see section 3.2.1), while the other datasets are used to test the performance of the trained systems.

4.2 Decision Trees

The first symbolic learning technique used in this experiment are decision trees. Decision trees are a type of directed graph, where each node has two child nodes. Nodes without any children are referred to as leaf nodes. Each non-leaf node has an associated question and its two child nodes represent each of the possible answers to that question (either yes or no). Decision trees classify new instances by simply answering the questions in each node and progression through the tree until a leaf node has been found. Each leaf node contains a class label, which is the label that the new instance will get. The decision trees are generated using a decision tree algorithm, which will be discussed in the next section. The main advantage of a decision tree classifier over a neural network is that the rationale that a decision tree uses can easily be interpreted by humans.

For the welfare benefit dataset, the expected tree that the algorithm generates would look something like the tree in Figure 4.1. In this Figure, an output of 1 indicates that the person is eligible for welfare benefits, whereas an output of 0 indicates that the person is ineligible for welfare benefits. This tree incorporates all of the six conditions C1-C6 that are used to determine whether the person is eligible or not, and would therefore create an accuracy of 100% on each of the test sets. It should be noted that in this particular tree, the conditions are represented in order, such that the first question is part of C1 and the last question deals with C6. However, since the order of a conjunction is not important for its truth value, those type of variations on this tree would not be unexpected or wrong.

4.2.1 The Algorithm

The decision tree algorithm used in this experiment is a simple CART algorithm that uses both the gini impurity and information gain (Kuhn and Johnson, 2013). The decision trees are built from top to bottom, where the top node of the tree represents the entire dataset. The goal is to split up the dataset at each node to achieve the highest possible homogeneity in the child nodes in terms of the class labels, such that the final leaf nodes of the tree contain instances with only one class type each. To split a dataset, a question needs to be asked. An example question would be 'Is Age greater or equal to 65?', which would separate the instances based on whether their age is greater or equal to 65. In order to find the best question for the dataset at each node, questions are generated using all of the possible feature-value combinations that exist within the dataset at the current node. This dataset is split using each of the generated questions and the information gain is calculated for each question as well. The question with the highest information gain is chosen as the best question and is used in the final tree at that node. Two new nodes are then generated from the previous node using the question in the previous node. This process repeats itself recursively until a node is formed with either a completely homogeneous class distribution or if it is no longer possible to split the node any further (all further questions do not increase the information gain). These nodes are the leafs in the tree. Additionally, a stopping criterion is implemented as a form of pre-pruning. This criterion can prevent a node from splitting into child nodes if the information gain is lower than the stopping criterion.

For each question, the information gain is calculated as follows, where $current\_node$ represents all of the instances in the current node, $true\_branch$ represents the subset of the data that answered 'yes' to the question, and $false\_branch$ represents the subset of data that answered 'no' to the question.

$$InfoGain = Gini(current\_node) - p * Gini(true\_branch) - (1-p) * Gini(false\_branch) \tag{4.1}$$

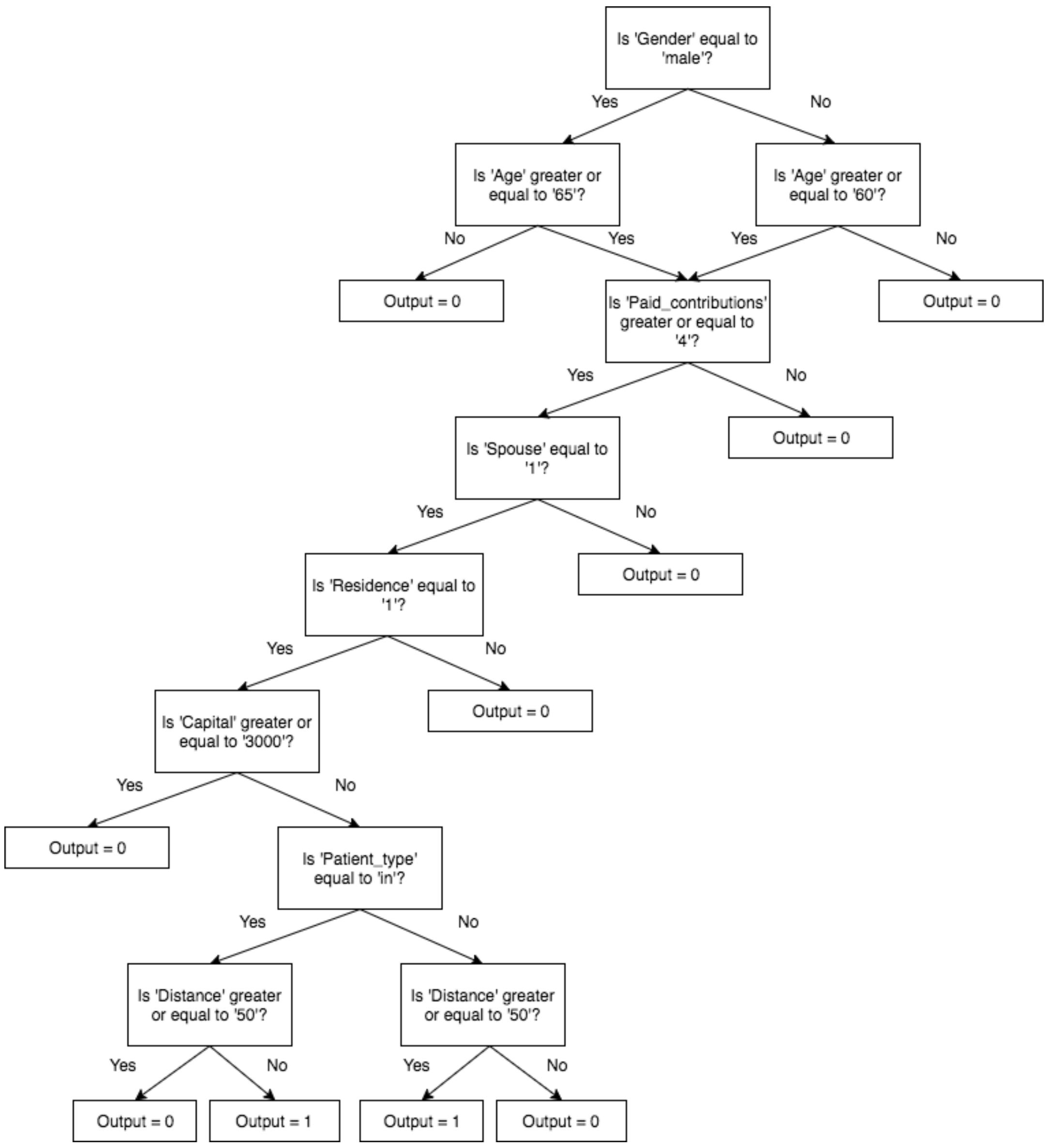

Is 'Gender' equal to 'male'?
Yes
No
Is 'Age' greater or equal to '65'?
Is 'Age' greater or equal to '60'?
No
Yes
Yes
No
Output = 0
Is 'Paid_contributions' greater or equal to '4'?
Output = 0
Yes
No
Is 'Spouse' equal to '1'?
Output = 0
Yes
No
Is 'Residence' equal to '1'?
Output = 0
Yes
No
Is 'Capital' greater or equal to '3000'?
Output = 0
Yes
No
Output = 0
Is 'Patient_type' equal to 'in'?
Yes
No
Is 'Distance' greater or equal to '50'?
Is 'Distance' greater or equal to '50'?
Yes
No
Yes
No
Output = 0
Output = 1
Output = 1
Output = 0

FIGURE 4.1: The expected decision tree that the algorithm will produce, based on the rules that generated the dataset.

In this formula, p is the probability of selecting an instance from the *true_branch* from the *current_node*, which is calculated by simply dividing the number of instances from the *true_branch* by the number of instances in the *current_node*. The Gini impurity of a set X with j labels is calculated as follows, where p_i is the chance of an instance of class i being randomly selected from the set :

$$G(X) = \sum_{i=0}^{j} 1 - p_i^2 \tag{4.2}$$

4.2.2 Results

To start off, the decision tree algorithm was trained on both Training set A and Training set B separately without a stopping criteria. The trees were then generated by training on the same training sets but with a stopping criterion (SC) of 0.003. This, however, only affected the tree generated from Training set B. The trained systems were used to classify the four test sets, the results of which can be seen in Table 4.1 below.

TABLE 4.1: The performance of the decision trees trained on Training set A & B for all test sets.

	Test set A	Test set B	Age set	Distance set
Training set A	99.90	91.40	97.50	50.00
Training set B	97.70	89.05	96.12	50.16
Training set A (SC)	99.90	91.40	97.50	50.00
Training set B (SC)	99.90	91.40	97.49	50.00

The decision tree generated by training on Training set A performs really well on both Test A (99.9%) and on Test set B (91.4%). On the age set the accuracy is quite high as well (97.5%), but it has significant issues with the distance set (50%). One of the main advantages of decision trees, as mentioned earlier, is that the rationale that a decision tree uses to classify new instances is easy to interpret. The decision tree that was generated by training on Training set A can be see in Figure 4.2. From this figure it is clear that the systems is able to accurately learn C2 (paid contributions), C3 (spouse), and C4 (residence) and almost learned C5 (Capital) perfectly as well (it is off by 3). It seems that it was unable to accurately learn C1 (Age/Gender), which the system incorrectly simplified to 'Is the age greater or equal to '60'. It also completely neglected rule R6 (patient type/distance). This tree makes it possible to interpret the results of Table 4.1 in quite an intuitive matter.

First of all, in the previous chapter and in Bench-Capon's original study, it was shown that even if the system learns only a few of the conditions, it can still have quite a high performance in classification (Bench-Capon, 1993). This explains the high performance of the system on Test set A. When we look at the performance on the Age set, the system misclassified 2.5% of all the cases. From Figure 4.2 we know that the system simply looks at whether someone is above 60 years of age or not. This means that a male of 60 years of age (who should not be eligible for a welfare benefit) is eligible for a welfare benefit according to this system. The age feature in the Age set is varied from 0 to 100 in steps of 5 for both genders, which means that there are 20 age steps for both men and women. Therefore, 1 in every 40 instances is a male with the age of 60, which equals 2.5% of the dataset. Similarly, on test set B, all of the cases that are supposed to fail because rule C6 (patient type/distance) is not applicable will not be recognized correctly by the system. That is 1 out of every 6 instances, or around 16.67%. Half of these cases are classified correctly by the system by chance, which means that the system will always classify around 8.3% of Test set B incorrectly. The remaining misclassified instance (around 0.4%) can be explained by the issues regarding C1 (Gender/Age) as discussed earlier and perhaps the imprecise deduction of C5 (Capital). Since the system is unable to learn C6 (patient type/distance) whatsoever, it makes sense that the performance on the Distance set is equal to 50%.

The decision tree generated from Training set B has a total of 32 non-leaf nodes. The first five nodes (from the top down) are the same as the tree generated from Training set A, as seen in Figure 4.2, but in a different order. The remaining nodes almost exclusively use the noise attributes as questions. The system is therefore overfitted to the training data, as it is supposed to completely ignore the noise attributes. Because

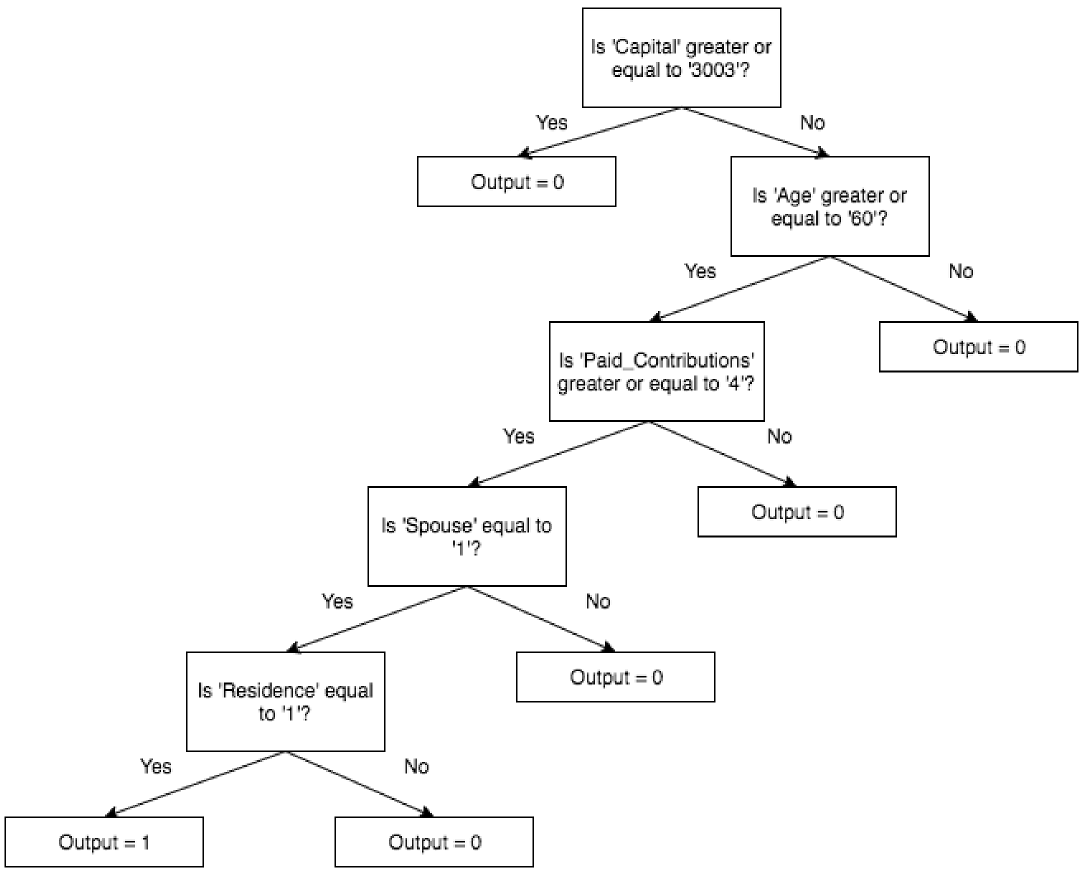
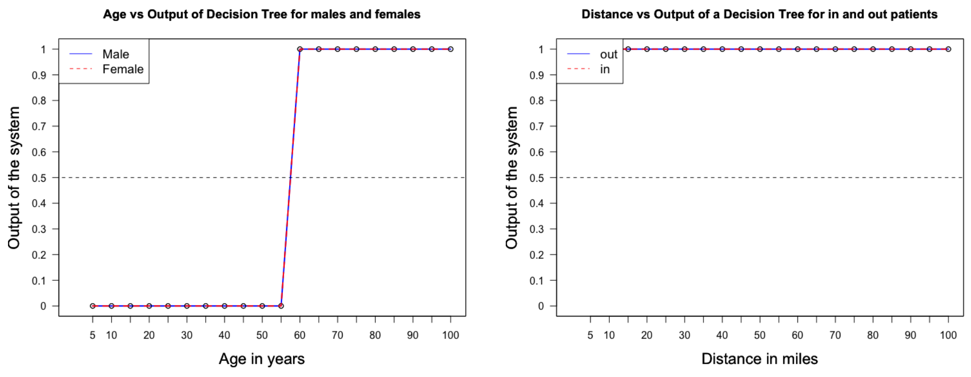

Is 'Capital' greater or equal to '3003'?
Yes
No
Output = 0
Is 'Age' greater or equal to '60'?
Yes
No
Is 'Paid_Contributions' greater or equal to '4'?
Output = 0
Yes
No
Is 'Spouse' equal to '1'?
Output = 0
Yes
No
Is 'Residence' equal to '1'?
Output = 0
Yes
No
Output = 1
Output = 0

FIGURE 4.2: The decision tree created by training on Training set A.

of these extra, incorrect nodes, the performance decreases on Test set A, B and the age set when compared to the tree generated from Training set A. The performance on the distance set is slightly higher (0.16% higher), which is most likely due to a spurious correlation with the noise attributes, as the tree does not include any questions regarding distance or patient type.

Because of the overfitting that occurs while training on Training set B, the trees are generated again using the same training sets, but with a stop criterion of 0.003. For the tree generated from Training set A, this does not make any difference, as it generates the exact same tree as seen in Figure 4.2, and thus performed exactly the same as well (as seen in Table 4.1). For the tree Trained on Training set B, however, the tree is shortened significantly, such that only the first five nodes remain. This means that the tree is almost the same as the tree seen in Figure 4.2, but with a different order of the nodes and a slightly lower value for C5 (Capital). In Table 4.1 we can also see that the performance of this tree is almost exactly the same as the performance of the trees trained on Training set A. Variations in the stopping criterion do not yield interesting results; if it is lower than 0.003, the tree includes all of the nodes with the noise attributes, and if the criterion is 0.003 or higher, it changes it to a variation of the tree seen in Figure 4.2.

Just as in the previous chapter, the graphs for the age experiment and distance experiment are shown in Figure 4.3 below. Even though these results may seem obvious, they are included for the sake of completeness. These results are produced by the tree that is generated using Training set A, but all of the other trees produce the same graphs. In Figure 4.3A the output of the system is plotted against the age for both males and females. There is a clear change from 0 to 1 when the age changes from 55 to 60. This is due to the second node of the tree, as seen in Figure 4.2, which asks the question of whether the age is greater or equal to 60. In Figure 4.3B, the output of the system is plotted against the distance for both in and out patients. Since there is no node that uses the patient type or distance, the graph is a flat line at 1, meaning that the system classifies every instance as eligible for welfare benefits.



(A) Gender and Age vs Output (B) Patient type and Distance vs Output

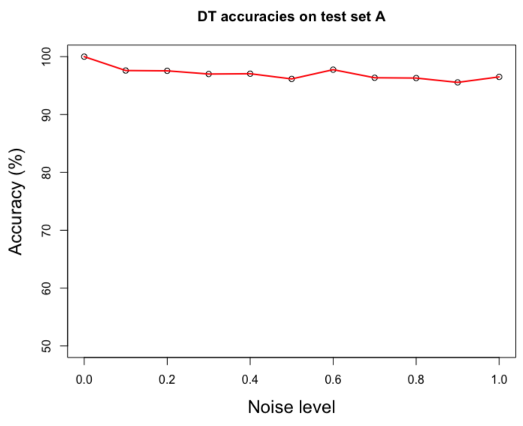
FIGURE 4.3: a. The age in years versus the output of the decision tree for males (blue) and females (red). b. The distance in miles versus the output of the decision tree for out patients (blue) and in patients (red).

4.2.3 The Influence of Noise on Decision Trees

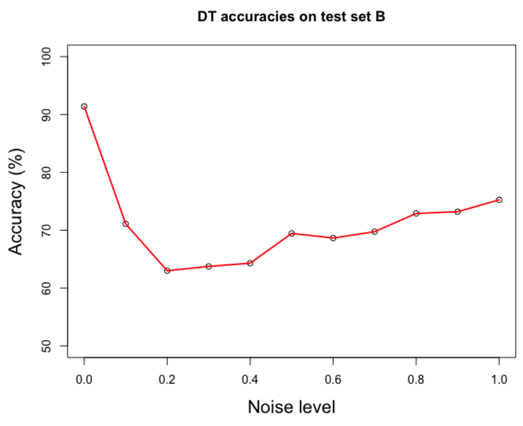
As in the previous chapter, it is examined how well the decision tree algorithm is able to learn the dataset in a noisy environment. To this end, the same type of noise is used, where a certain percentage of the instances of a dataset has a noisy variable, whose value is the opposite of what it should be. The noise level is varied between 0 to 100% and is applied to training set A, test set A and test set B. The decision tree algorithm is trained on training set A, and its performance on test set A, test set B, the age test set and the distance test set can be seen in Figure 4.4.

The accuracies in Figure 4.4A decrease slightly once more noise is applied, but they remain quite high. The noise has a larger effect on the performance on test set B, and drops quite significantly with only 10% noise. Both of these results are similar to the results of the neural networks in the previous chapter, where the accuracy on test set A decreased very slowly and the accuracy on test set B decreased more rapidly. The accuracies on the age test set are lower with noise as well, but they fluctuate quite a lot after adding more than 20% noise. This seemingly unstable accuracy pattern is due to the fact that this test set practically only uses two variables to get the right output: age and gender. At some noise levels, the questions that the decision tree generates are unable accurately split the dataset on the age/gender condition, thus causing a low accuracy. The accuracies on the distance test set remain around 50% regardless of the amount of noise in the training data set.

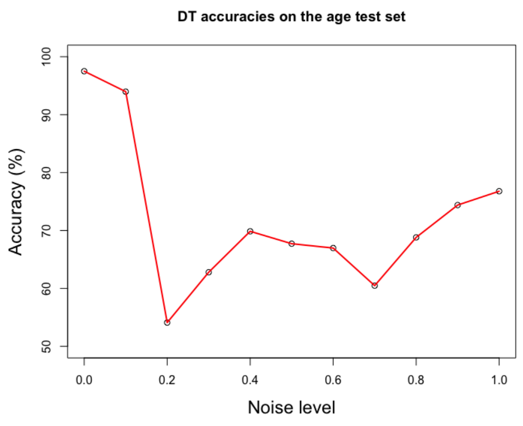
These results display the performances of the decision trees in a more realistic manner, as real life, non-artificial datasets usually contain noise. Just as with neural



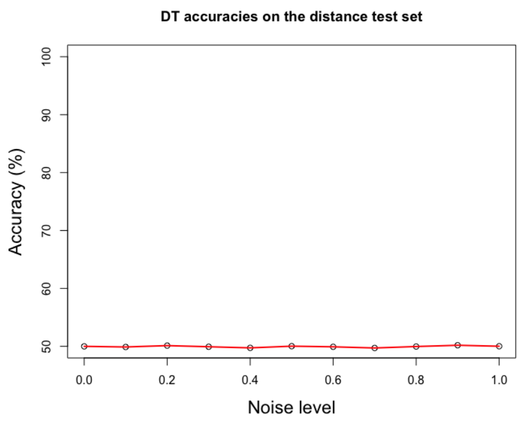
(A) Test set A



(B) Test set B



(C) Age test set



(D) Distance test set

FIGURE 4.4: The effect of varying levels of noise on the accuracies of the decision tree algorithm on test set A (A), test set B (B), the age test set (C) and the distance test set (D).

networks, the overall performance of the decision tree only decreases slightly with more noise, as seen in 4.4A. Its ability to learn the correct conditions, however, plummets as soon as 20% noise is introduced into the training set. Regardless of the noise, it cannot deduce the distance/patient type condition.

4.2.4 Conclusion regarding Decision Trees

The decision trees in this experiment are able to perform really well in terms of classification accuracy, with accuracies reaching up to 99.9%. Training on Training set A yields results that are slightly better than if the system trains on Training set B (without stopping criterion). When examining the rationale, however, it seemed that the system was only able to learn the easier conditions with only one variable. Rule C1 (Gender/Age) and C6 (Patient/Distance) were not learned by the system, even though it approximated rule C1. Interestingly, if a decision tree is generated using the Age set or the Distance set as a training set, it is able to perfectly produce rules C1 (Gender/age) and C6 (Patient type/Distance) respectively. This shows that a decision tree is able to learn these more complex rules. An initial hypothesis is that the training sets are too small for the decision tree algorithm to pick up on these complex rules. Training on training sets with 10.000 or even 100.000 instances, however, provided the same results.

4.3 Association Rules

The second symbolic learning technique that will be examined is association rule learning (Agrawal, Imieliński, and Swami, 1993). This technique finds associations between the features within the data and makes rules that reflect this association. An example of such an association rule is $[A] \rightarrow [B]$, which states that B must be true if A is true. In order to use these rules in classification tasks, Classification Association Rules (CARs) need to be used, in which the consequent of each rule is the class feature with its label (Ma and Liu, 1998). For the welfare benefit dataset, this means that the consequent will always be $[Eligible, 1]$ or $[Eligible, 0]$, where the former represents that the person is eligible for a welfare benefit and the former that the person is not eligible for a benefit. Such a combination of a feature and its value is referred to as an item. For example, one of the CARs that the system is expected to produce is $[Residence, 0] \rightarrow [Eligible, 0]$, which states that if the person is not a residence of the UK, he or she is not not eligible for a benefit. This is the CAR representation of condition C4 (Residence). Using the CARs that the system finds, it is able to classify new instances. Association rule learning is not possible on continuous data, therefore some of the features (age, capital resources, distance and the noise features) need to be discretized. The values of each of these continuous features are divided into 20 bins based on the maximum and minimum values of that feature. The age feature, for instance, was discretized into steps of 5 years, since the maximum value is 100 years and the minimum value is 0 years. Each instance in the dataset is then converted to a set of items. Based on conditions C1-C6, the expected CARs that the system produces are as follows:

- $[[Age, X], [Gender, male] \rightarrow [Eligible, 0]$, for all values of $X < 60$
- $[[Age, X], [Gender, female] \rightarrow [Eligible, 0]$, for all values of $X < 65$
- $[Paid\_contributions, X] \rightarrow [Eligible, 0]$, for all values of $X < 4$
- $[Spouse, 0] \rightarrow [Eligible, 0]$
- $[Residence, 0] \rightarrow [Eligible, 0]$
- $[Capital, X] \rightarrow [Eligible, 0]$, for all values of $X < 3000$
- $[[Distance, X], [Patient\_type, in] \rightarrow [Eligible, 0]$, for all values of $X > 45$
- $[[Distance, X], [Patient\_type, out] \rightarrow [Eligible, 0]$, for all values of $X < 50$

For each of the rules above that contains an item with X, a rule is created for each unique value that X can have in that situation. For example, in the first rule there are 11 possible values for X, one for each possible value that the age variable can have: [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55]. Therefore, 11 rules could be generated to completely satisfy the condition. When none of the rules in a CAR classifier apply to a new instance, it is given the default class. The expected default class for the welfare benefit dataset is that the person is eligible for a welfare benefit: $[Eligible, 1]$. In this way, if all of the conditions C1 to C6 apply to the instance, none of the CARs above apply, causing the system to correctly classify the new instance as eligible using the default class. Other possible valid rules would include all of the features with values such that rules C1 to C6 are satisfied, which would lead to $[Eligible, 1]$.

4.3.1 The Algorithm

The algorithm used to find the CARs and build a classifier consist of two parts: the Apriori algorithm that generates 'good' CARs from the data and the CBA-CB (Classification Based on Association - Classifier Builder) algorithm that builds the classifier (Ma and Liu, 1998).

CARs can be generated by simply going through all possible sets of items and adding an output item. These CARs can then be evaluated based on their support and confidence. The support of a rule indicates how often a set of items occur in the dataset. The confidence indicates how often the consequent is true if the antecedent is true. Based on a minimum support threshold, CARs that have a support that is too low will be discarded. Generating all of the possible CARs of a dataset, however, is extremely computationally intensive. Therefore, the Apriori principle is used to decrease the amount of CARs that need to be evaluated (Agrawal and Srikant, 1994). The Apriori principle states that if a set of items does not occur often in the dataset, then all possible sets that include that set do not occur often either. This means that if a CAR has a low support, any CAR that includes the same items in its antecedent will also have a low support. For example, if the rule $[A, 1] \rightarrow [C, 1]$ has a low support, rule $[A, 1] \wedge [B, 1] \rightarrow [C, 1]$ will have a low support as well. This allows the algorithm to skip over a large number of potential CARs, as their support would have been too low anyway. Once all of the good CARs have been generated, they are sorted on their precedence. A rule r_i precedes rule r_j if r_i has a greater confidence, if the confidences are the same but r_i has a higher support, or if both the support and the confidence are the same but r_i was generated earlier.

The sorted list of CARs is used to create a classifier. Starting with the rule with the highest precedence, each rule will be used to classify all instances from a training set. If an instance was correctly classified, the rule will be marked and the instance that it classified correctly will be removed from the dataset. If the rule is marked, it will be appended to the end of the classifier. The default class of the classifier will then be changed to the majority class of the remaining dataset, and the new classifier will be used to classify the entire dataset. The error of this classification will be stored on the index of the current rule. This is repeated for every CAR that was generated. Once every rule has been presented to the system, the first rule in the classifier with the lowest total number of errors will be found. Every rule that was appended to the classifier after this rule will be discarded, as they only caused more errors or did not reduce the error and are thus redundant. Finally, a new default class will be computed for the final classifier. New instances are classified by going through all of the rules in classifier in order. The first rule that applies to the instance is used to determine the class of the instance. If none of the rules apply, the instance is classified as the default class.

4.3.2 Results

The system is trained on both Training set A and Training set B separately and is tested using Test set A, Test set B, the Age set and the Distance set. The performance of the system can be seen in Table 4.2. The minimum support threshold is set at 0.02, which, through trial and error, was discovered to yield the best results in terms of performance. When training on Training set A, 2182 rules are generated by the Apriori algorithm and the final classifier consists of 39 rules. When training on Training set B, the Apriori algorithm generates 2047 rules and the final classifier consists of 91 rules. The default class of both of the systems is 1 (eligible).

TABLE 4.2: The performance of the association rule system trained on Training set A & B for all test sets.

	Test set A	Test set B	Age set	Distance set
Training set A	99.50	89.50	63.30	50.42
Training set B	76.05	63.60	54.17	49.80

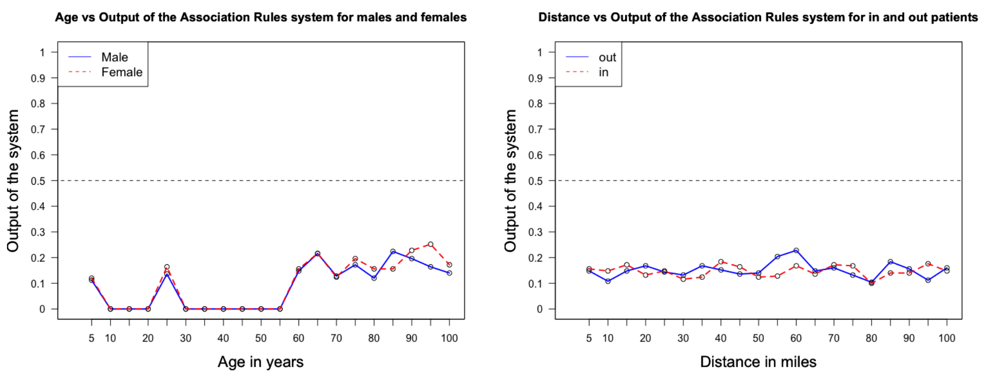
The AR System trained on Training set A

From Table 4.2 it is clear that the system trained on Training set A yields great results when classifying Test set A (99.5%). The performance on Test set B is quite a bit lower (89.5%), and the results from the Age set is even lower (63.3%). The accuracy on the Distance set is roughly 50%, which is the classification accuracy that the system would have if it would label every instance as either eligible or ineligible regardless of its features. The rules of the classifier of this system can explain these performances. The first five rules are shown below:

- $[Spouse, 0] \rightarrow [Eligible, 0]$
- $[Residence, 0] \rightarrow [Eligible, 0]$
- $[Paid\_contributions, 3] \rightarrow [Eligible, 0]$
- $[Paid\_contributions, 2] \rightarrow [Eligible, 0]$
- $[Paid\_contributions, 1] \rightarrow [Eligible, 0]$

All five of these rules were expected. Out of the remaining 34 rules, 18 are single item rules with the capital feature, where a capital value higher than 3000 leads to $[Eligible, 0]$ and a capital value lower than 3000 leads to $[Eligible, 1]$. The former part is correct, however, the latter part is not. Just because the capital value is low enough for rule C5 (Capital) to be applicable, this does not automatically imply that the person is eligible for a benefit, as all of the rules C1-C6 need to be applicable for that to be true. Similarly, 16 of the rules are single item rules with the age feature, which also include both the correct and incorrect rules; some of the rules incorrectly state that a person is eligible if he or she is older than 65, regardless of other conditions. The last two rules are $[[Paid_contributions, 4], [age, 75]] \rightarrow [Eligible, 1]$ and $[[Spouse, 1], [age, 75]] \rightarrow [Eligible, 1]$. Both of these rules are incorrect, as they imply that the person is eligible without knowing if the remaining conditions are satisfied.

As discussed in the previous chapter, with only a few rules, a high classification accuracy can be gained from Test set A, which explains the high performance (99.5%). The system is only able to learn C2 (Paid contributions), C3 (Spouse) and C4 (Residence) correctly, and partly learn C1 (Age/Gender) and C5 (Capital). This explains why the performance on Test set B is lower. Since C1 (Age/Gender) is only approximated and not learned correctly, the system has a low performance on the Age test set. The system does not learn C6(Distance/Patient type) at all, which is why the performance on the Distance set is roughly 50%. The plots of the age experiment and the distance experiment are shown in Figure 4.5.



(A) Gender and Age vs Output (B) Patient type and Distance vs Output

FIGURE 4.5: a. The age in years versus the output of the association rule system for males (blue) and females (red). b. The distance in miles versus the output of the association rule system for out patients (blue) and in patients (red).

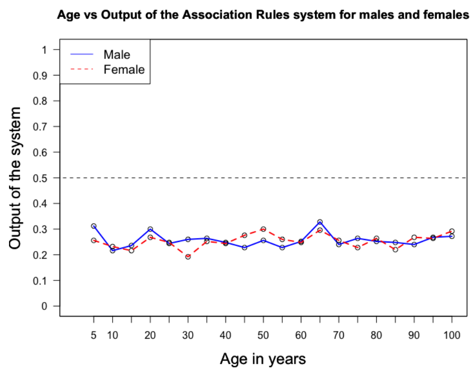
The output of the system in both graphs is the averaged output over all of the instances in the datasets. The actual classifier can only provide a 0 (ineligible) or a 1 (eligible). In the age plot (Figure 4.5A) most of the outputs for instances below 60 are equal to 0. When looking at the rules, these outputs make sense, as there are 9 rules in the form of $[Age, X] \rightarrow [Eligible, 0]$, where X is lower than 60. Notably, the rules in which X is 5 and X is 25 are not learned by the system, which is reflected in the graph in Figure 4.5A. There is an increase in the output after 55 years, which is due to the 5 rules of the form $[Age, X] \rightarrow [Eligible, 0]$, where X is 65, 75, 85 and 95. These rules, however, have quite a low precedence in the classifier, so often an earlier rule related to a different feature is used to classify these instances. This explains why the average output between 60 and 100 is still relatively low. The difference between male and female was not found, but some slight variations between the two genders can be seen among instances with higher ages. However, this is most likely due to noise from other features as the gender feature is not present in the classifier. In Figure 4.5B the distance is plotted against the output for in and out patients. There does not seem to be any relation between these two features, which makes sense, as none of the rules in the classifier contain the distance or patient type feature.

The AR System trained on Training set B

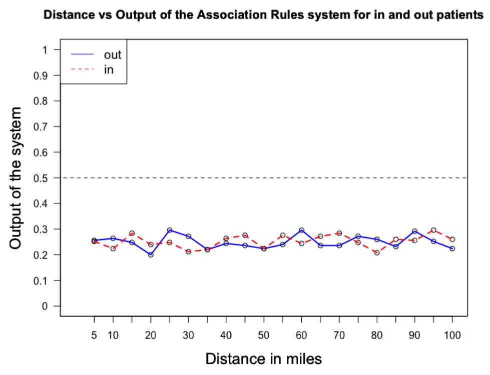
The system performs worse when trained on Training set B, as evident from the performances in Table 4.2. The accuracy on Test set A is only 76.05% and it only predicts 63.6% of the instances correctly of Test set B. This is lower than the system trained on Training set A, which scores 99.5% and 89.5% in these cases respectively. On the age set, the system only scores a bit over the guessing chance for binary problems with a 54.17% accuracy. The Distance set is classified with an accuracy of 49.8%, which would also have been achieved by a classifier that always predicts the same outcome.

This system consists of a classifier with a lot more rules than the previous system; the classifier generated from Training set B has 91 rules, whereas the classifier generated from Training set A only has 39 rules. The first five rules of this classifier, however, are the exact same as the one that the classifier trained on Training set A produced. Out of the remaining 86 rules, 81 rules include items based on the noise attributes. These are clearly incorrect, as the noise attributes do not contribute to the eligibility of the instances whatsoever. The other five rules that do not include the noise attributes are incorrect as well, in the sense that they are in the form of 'If one of the conditions C1-C6 is applicable, the person is eligible'. As discussed in the previous section, this is incorrect, as all of the rules need to apply in order for someone to be eligible. With less correct rules than the classifier trained on Training set A, it makes sense that all of the overall performance of this system is lower, as seen in Table 4.2.

In Figure 4.6 the plots of the age and distance experiment are shown. As expected, both plots do not show any relation between the two variables and the output. This is because there are no rules regarding gender, age, distance or patient type in the classifier.



(A) Gender and Age vs Output



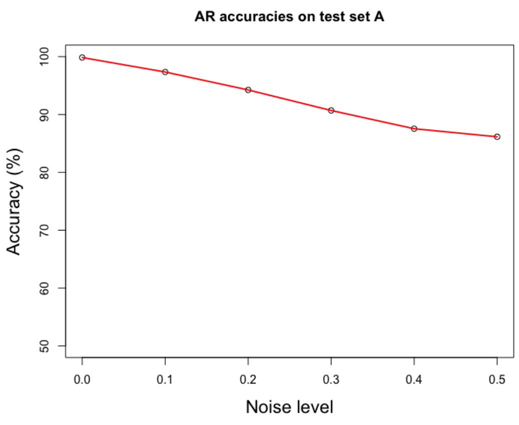
(B) Patient type and Distance vs Output

FIGURE 4.6: a. The age in years versus the output of the association rule system for males (blue) and females (red). b. The distance in miles versus the output of the association rule system for out patients (blue) and in patients (red).

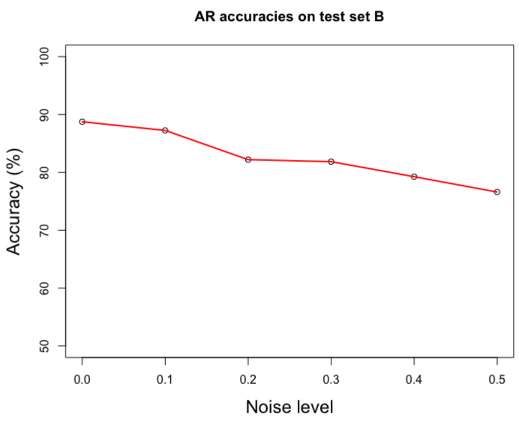
4.3.3 The Influence of Noise on Association Rules

Just as with the neural networks and the decision trees, the association rule system is trained and tested on data sets with noise as well. A percentage of the instances in the dataset are provided with noise, which changes the value of one random variable into its opposite value. This type of noise is applied in various levels to training set A, test set A and test set B. The association rule classifier will train on this noisy training set A, and be tested on test set A, test set B, the age test set and the distance test set. The resulting accuracies can be seen in Figure 4.7.

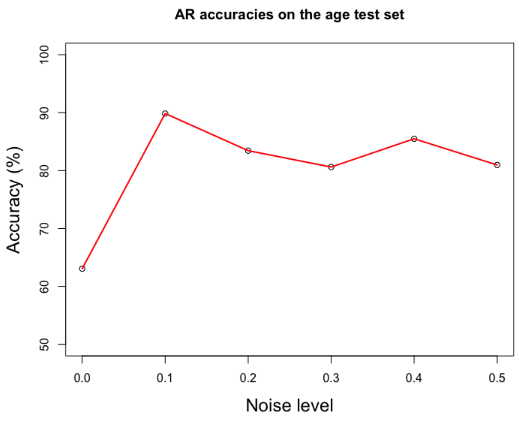
The accuracies on test set A and test set B seem to drop somewhat at the same rate as more noise is introduced, but the accuracies on test set A are higher. This is different from the accuracies on the neural network and decision tree, where the accuracies on test set B dropped more significantly than the accuracies on test set A. Interestingly,



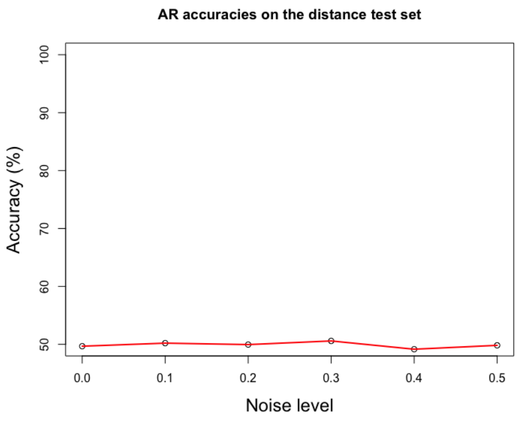
(A)



(B)



(C)



(D)

FIGURE 4.7: The effect of varying levels of noise on the accuracies of the association rule algorithm on test set A (A), test set B (B), the age test set (C) and the distance test set (D).

the accuracies on the age set actually increases with noise rather than decrease. It is still not able to connect the gender variable to the age variable in order to successfully learn the condition, but it is able to generate more rules using age. A possible explanation deals with the fact that the age feature is spread out over 20 variables and only one of these variables is changed when noise is added to an instance. Most of the other features are represented by only a single variable and are therefore affected more by the noise than the age feature. It could therefore be that the support and confidence of the rules of the other conditions drop more significantly than the rules of the age condition. This means that the confidence and support of the age rules are relatively higher, and thus more age rules are included, which in turn leads to a higher classification accuracy. The accuracy on the distance set remains at 50%, regardless of the noise level, just as the decision tree.

4.3.4 Conclusion regarding Association Rules

When training on Training set A, the association rules classifier is only able to learn the simple conditions of the dataset: C2 (Paid contributions), C3 (Spouse), C4 (Residence). It is able to approximate rules C1 (Gender/Age) and C5(Capital) as well, but cannot learn it completely. This is an inherent issue with association rules, as they cannot easily internalize a 'greater than' relationship. Instead, all of the possible values within the range of the 'greater than' relationship have to be defined and learned as individual rules. This makes it possible for the system to miss some of these rules, creating unwanted results as seen in Figure 4.6A when the age is 5 or 25. Even with this limited set of rules, however, the classifier is able to yield an accuracy of 99.5%. The classifier trained on Training set B performs more poorly and was only able to learn the simple conditions. Unlike with the decision tree system, training the association rule system on the Age test set does not cause it to learn condition C1 (Gender/Age). Instead it produces similar rules as when trained on Training set A.

4.4 Conclusion

Both decision trees and association rules have been explored in order to determine what rules they can and cannot learn. Both type of the techniques are able to easily recognize and learn simple boolean conditions, e.g. C3 (Spouse), C4 (Residence), or categorical conditions with a limited amount of unique values, like C2 (Paid contributions). The decision tree algorithm has no real difficulties with simple 'greater than' relationships, e.g. C5 (capital), even though the values that it learns may not match exactly. The disjunction of two conjunctions, C1 (Age/Gender) and C6 (Patient type/ Distance), were not learned by either of the two techniques. It was shown that decision trees are able to learn these type of rules under relaxed circumstances. Regardless of the rules that it learned, both techniques were able to classify new instances with an accuracy of over 99%. Both association rules and decision trees performed worse when training on Training set B. In all circumstances, the decision tree system outperformed the association rule system.

Compared to neural networks in the previous section, both systems roughly preform equally well in terms of performance when trained on Training set A and testing on Test set A (NN: 98.9%, DT: 99.9%, AR: 99.5%) . When training on Training set A and testing on Test set B, these systems perform much better than the neural network (NN: 65.3%, DT: 91.4%, AR: 89.5%). On the Age set, the decision tree system preformed better, but the association rule system performed worse than the neural network (NN: 93.1%, DT: 97.5%, AR: 63.3%). Both of the symbolic systems performed really poorly on the Distance set, whereas the neural network performed decently (NN: 75.4%, DT: 50.0%, AR: 50.4%). When training on Training set B, performance of of each of the classifiers goes down for almost all test sets. The only exception is that the neural network scores higher on Test set B when training on Training set B. In terms of the rules that the classifiers are able to learn, the neural network is the clear winner. It is the only system out of the three that is able to correctly internalize the disjunction of conjunction conditions of both C1 (Gender/Age) and C6 (Patient type/Distance).

Chapter 5

Learnability of Symmetrical Boolean Functions

Besides a high performance in terms of accuracy, it is idealistically desirable to have machine learning algorithms that are able to correctly internalize the structure of the data. Combined with rise of of explainable AI, in which systems need to provide a reason for their decision, the desire to learn the right rules has become even more prominent; we want our systems to provide an explanation to their decisions that seems logical to a human user. The internal rationale of a system, however, does not necessarily need to be sound in order for the system to achieve a high performance accuracy, as was shown in the previous chapters. This chapter will aim to provide a more formal overview of what structures machine learning systems are able to internalize correctly, and where their limitations lay.

5.1 Previous Research

Previous studies have shown that achieving high classification accuracies with a machine learning system does not guarantee that the rules that this system learns are sound (Bench-Capon, 1993). Using a fictional dataset that is generated from a set of conditions, it is possible to investigate how well a machine learning algorithm is able to learn these specific conditions. In earlier research, a comparison was made between the performance of a neural network (Bench-Capon, 1993), defeasible logic (Johnston and Governatori, 2003) and an adaptation of the CN2 algorithm using argumentation (Možina et al., 2005). All three systems were tasked with training on and classifying the same fictional dataset. This dataset consisted of six conditions that determined the output, wherein the output is true if and only if all of the conditions are true. All systems produced a high classification accuracy, but none of them were able to exactly reproduce all of the six conditions of the dataset.

The study managed to draw a few interesting conclusions about the conditions (Možina et al., 2005). First of all, it was shown that simple boolean conditions that have to be either true or false are easily identified by all of the systems. Similarly, the systems seem to have no difficulty in identifying conditions in which a specific threshold value needs to be exceeded or not. Exclusive OR (XOR) type conditions on the other hand, in which only one of two variables needs to be true but not both, are difficult for the systems to learn. Additionally, so-called M-out-of-N conditions, in which at least M out of the N variables need to be true, are not easily identified by the system either.

In the previous chapters, the performance of neural networks, association rules and decision trees were explored in terms of how well they are able to learn specific conditions using a fictional dataset. Just as in previous research, Boolean and threshold conditions were easily identified by all systems. Both association rules and decision trees proved to have difficulties with the XOR type conditions, which conforms to the results found in previous research. In the repeated study of Bench-Capon, 1993, however, it was shown that neural networks are able to learn the XOR conditions without any apparent difficulties. This could indicate that there are certain types of conditions that some machine learning algorithm are able learn, whereas others are unable to do so. Interestingly, the decision trees were able to learn the XOR conditions in relaxed circumstances; without any other conditions. This suggests that interactions between conditions exist that can have an effect on what the machine learning algorithm is able to learn. In this chapter, a number of conditions will be examined to determine how well certain machine learning algorithms are able to learn them. Furthermore, the interaction between these conditions will be explored in order to investigate their effects. Lastly, the effects of the size of the training data on the learning of the conditions will explored.

5.2 Symmetrical Boolean Functions

The type of functions that neural networks and other machine learning algorithms have shown to struggle with in previous research can be generalized to symmetrical Boolean functions. The output of these functions is 1 based on the amount of 1's in the input vector. The location of the 1's in the input vector is therefore irrelevant. A famous example of a symmetrical Boolean function with two variables is the XOR function as discussed before, which provides an output of 1 if and only if the number of 1's in the input vector is equal to 1. With more than two variables, the XOR function can be generalized into the parity function, which is a function that returns an output of 1 if and only if there is an odd number of 1's in the input vector.

Symmetrical Boolean functions can be sub-categorized into the following three categories (Wegener, 1987):

- M-out-of-N function: $f^n_m(x) = 1 \leftrightarrow |x| \geq m$
- Exact value function: $f^n_m(x) = 1 \leftrightarrow |x| = m$
- Counting function: $f^n_{mk}(x) = 1 \leftrightarrow |x| \equiv m \bmod k$

In this notation, a Boolean function $f^n(x)$, has a Boolean input vector x with n values, where $|x|$ denotes the number of 1's in vector x. Variables m and k are fixed values. These three types of symmetrical Boolean functions form the basis of conditions used in the experiment. To decrease the overall complexity in terms of variables, however, the parity function will be used instead of the counting function. The parity function is a specific form of the counting function, where m is equal to 1 and k is equal to 2, which can therefore be expressed as follows:

- Parity function: $f^n(x) = 1 \leftrightarrow |x| \equiv 1 \bmod 2$

5.3 Datasets

A large number of datasets are generated based on a set of conditions Each instance in the dataset has an output value, which is true if and only if all of its conditions are true as well, similar to the datasets in previous research (Bench-Capon, 1993). This set of conditions can include the M-out-of-N function, the exact value function and the parity function. Each condition also has a set of parameters. As described earlier, parameter n denotes the number of variables of the condition and parameter m is used in M-out-of-N functions and exact value functions to determine the output. The number of variables of the instances in the datasets are therefore based on the amount of variables of the conditions.

For any set of conditions, the following datasets are generated: a training set, a general test set and a specific test set for each condition. First of all, a training set consists of 150,000 instances where half of the instances are given random values such that their output value is true, and the other half are given random values such that their output value is false. In the latter half, the instances are generated such that they fail on each condition equally. With two conditions for example, this means that the latter half (the half where the output is false) would consists for 50% out of instances where the first condition is false and for 50% out of instances where the second condition is false. If a condition is not specifically set up to fail, it is provided with random values. The general test set consists of 150,000 instances as well, and is generated in the exact same fashion.

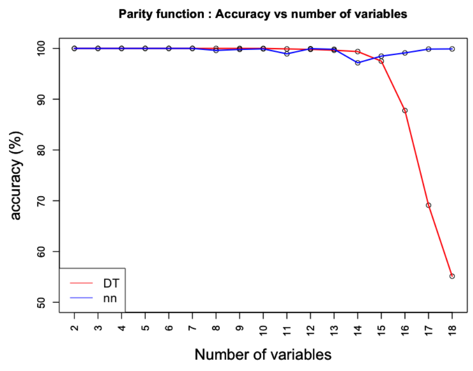
For each condition, an additional specific test set is generated. This test set is used to determine how well a system is able learn a specific condition. Given such a condition, its test set is constructed by generating all of the possible input values that the condition can have. For example, the XOR condition (or parity function with n=2) has $2^2 = 4$ possible values and would therefore have a specific test set with 4 instances. The variables of additional conditions are given random values such that their output is true. This ensures that the output is dependent on only the one condition for which the test set is generated.

5.4 Machine Learning Algorithms

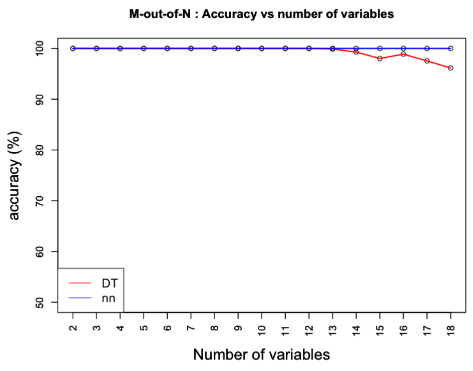
Both the neural network and the decision tree algorithm from the previous chapters will be used in this experiment. The decision tree system uses the CART algorithm with the Gini impurity and it does not make use of early stopping or any other form of pruning. The neural networks used in this experiment consist of three hidden layers, with 25, 10 and 3 hidden nodes respectively. The learning rate is set to 0.05 and the sigmoid is used as the activation function. The networks use a mini-batch approach with a batch size of 50. The maximum number of times that the entire training dataset will be presented to the network during the training phase is set to 2000 rounds. Training stops early if there is hardly any change in the training errors in the past 10 rounds.

5.5 Single Condition Datasets

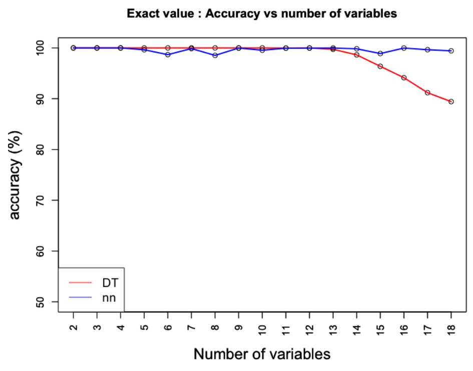
First of all, datasets are generated with only one condition with different numbers of variables n; from 2 to 18. For the M-out-of-N and exact value condition, the value of m will be varied as well: between 1 and n. For each condition, both the decision tree algorithm and the neural network are trained on the training dataset and are then tested on the general test set and the specific condition test set. In Figure 5.1 the accuracy of the decision tree and the neural network on the general test set is shown versus the number of variables for all three conditions. For the M-out-of-N and exact value condition, the mean accuracies of all possible values values of m are displayed.



(A) Parity condition



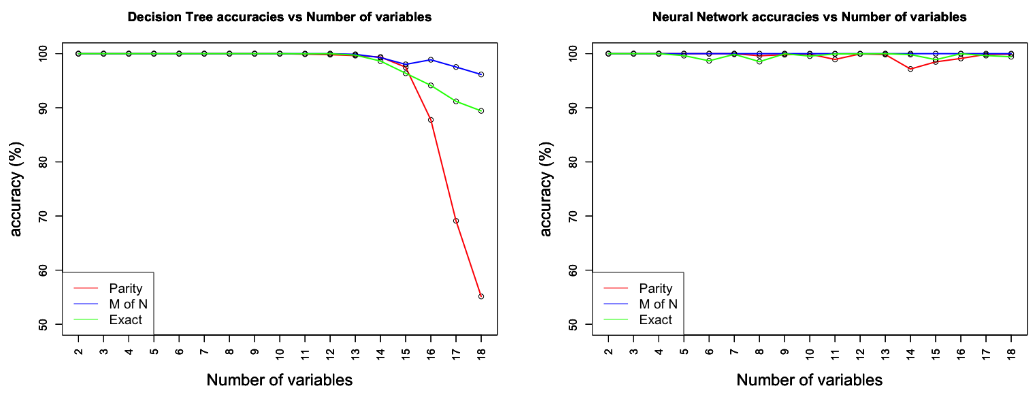
(B) M-out-of-N condition



(C) Exact value condition

FIGURE 5.1: The accuracy of the decision trees (red) and neural network (blue) on the general test set after training on the parity condition (A), M-out-of-N condition (B) and exact value condition (C) with different numbers of variables.

The accuracies of these systems on the specific test sets were almost identical, and a repeated measures anova showed no significant difference between the performance on the general test set and on the specific test set for all of the conditions. Figure 5.1 shows that the accuracy of the decision tree system decreases as the number of variables of the condition increases. The neural network does not appear to have difficulties with any of the conditions, as the mean accuracy for each number of variables never reaches lower than 98%. Earlier research has shown that with sufficient training and enough hidden nodes, a neural network should theoretically be able to learn any parity function (Wilamowski, Hunter, and Malinowski, 2003). If we assume that the same rule applies to the other types of symmetrical Boolean functions, the 150,000 training instances are sufficient for a network of this format to learn the conditions. Clearly, this is not sufficient for the decision tree algorithm. In Figure 5.2 the mean accuracies of both the neural network and decision tree algorithm for the different conditions are shown for varying number of variables. The average accuracy of the neural network is relatively stable near 100% for each condition and each number of variables. The accuracy of the decision tree decreases at around 14 variables for all three conditions. The accuracy on the parity condition drops rapidly to 55% with 18 variables, whereas the accuracies of the exact value and M-out-of-N conditions decrease to 89% and 96% respectively at 18 variables.



(A) Decision tree accuracies (B) Neural network accuracies

FIGURE 5.2: The accuracies of the decision tree (A) and neural network (B) on the parity condition (red), M-out-of-N condition (blue) and exact value condition (green) for varying number of variables.

As stated earlier, the accuracies of Figure 5.1 and Figure 5.2 are aggregated over all possible values of m. When keeping the number of variables n constant and varying the values for m, some interesting patterns emerge. In Figure 5.3 the systems were trained and tested on datasets with only the exact value condition, where the number of variables n was set to 20. The training set of these systems was reduced to only 500 training instances, as the larger training set introduced a ceiling effect in the neural network as seen in Figure 5.2B The accuracies of the neural network and the decision tree are shown on both the general and specific test set.

The graphs of Figure 5.3 are generated from datasets where n is equal to 20, but systems trained on datasets with other values of n generated graphs with similar shapes. Interestingly, the accuracies of the decision tree on the general test and on the exact test set form a parabola; the accuracy decreases until m is equal to around half of the value of the total number of variables n, after which it increases again. The accuracies of the neural network see a similar dip at this point, on both the general and exact test set. This parabola shape makes sense, as the ratio between the number of possible values that return true and the number of possible values that return false is highest when m is equal to half of n. For instance, if m is equal to 1, the number of input vectors that will return true is equal to only 1, whereas the number of input vectors that will return false is equal to $2^n - 1$. Therefore the system only needs to recognize only a single input vector that returns true, and return false otherwise. Conditions with ratio of true to false outputs that deviate far from 0.5 are

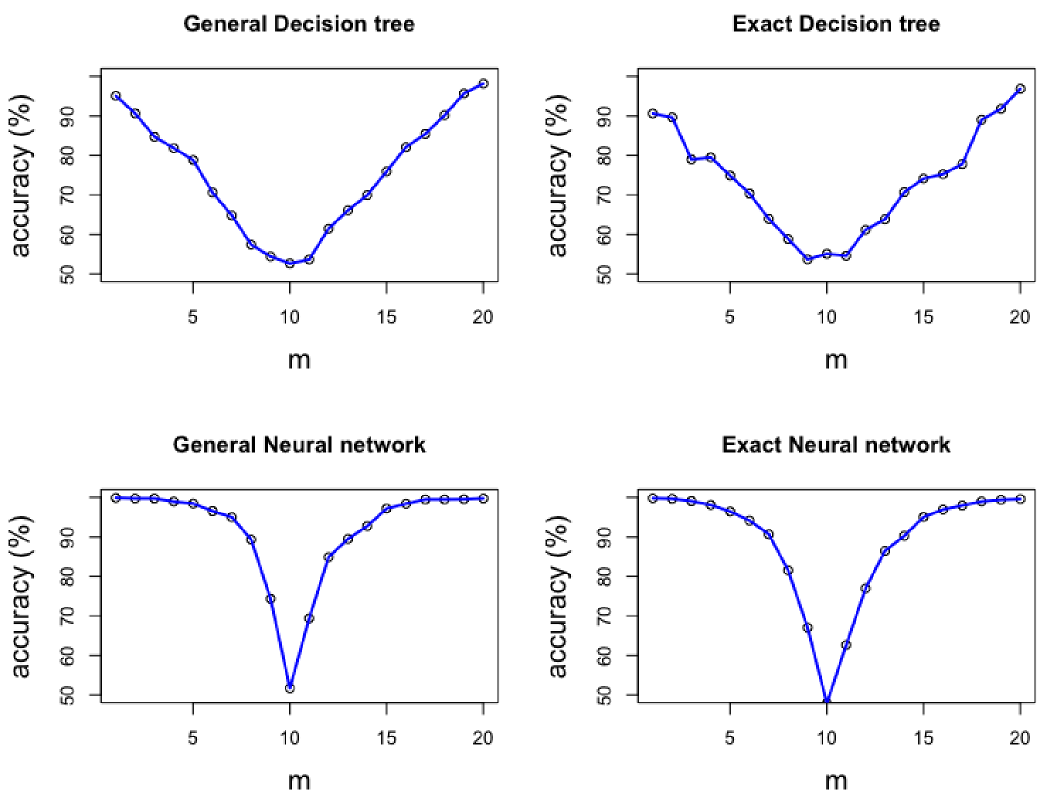


FIGURE 5.3: The accuracy of the decision trees and neural networks on the general and specific exact test set after training on the exact condition with 20 variables and varying values of m.

simple, and therefore easier for the systems to learn. The decrease and increase in accuracy of the neural networks, however, is much more sudden then the accuracies of the decision tree. The accuracies of the neural network are therefore significantly higher than the ones of the decision tree (Decision tree: 75.49% on general, 73.54% on exact; Neural network: 91.71% on general, 88.936% on exact).

A set of graphs similar to Figure 5.3 was generated for the M-of-N condition, and can be seen in Figure 5.4. In the figure, the same parabola shape can be seen in the accuracies of the decision tree algorithm on the general test set. The accuracy trend of the neural network on the general test set somewhat resembles the parabola shape as well, but the values all remain above 90% accuracy. The accuracies on the M-out-of-N test set in Figure 5.4, however, do not show a similar shape; they display a quick decrease in accuracy, followed up by an increase in accuracy after m=4 as m increases. The accuracies of the decision tree on the M-out-of-N test set dip down to 50%, whereas the neural network only decreases to around 70%. Furthermore, the accuracy of the neural network increases more rapidly, plateauing at 8 variables, whereas the decision tree only reaches a 100% accuracy at 13 variables. Just as with the exact value function, the accuracies of the neural network are significantly higher than those of the decision tree (Decision tree: 82.1075% on general, 79.088% on M-out-of-N; Neural network: 98.72% on general, 98.00% on M-out-of-N). Compared to the accuracies when trained on the exact function (Figure 5.3), however, the accuracies when trained on the M-out-of-N function are around 6-10% higher.

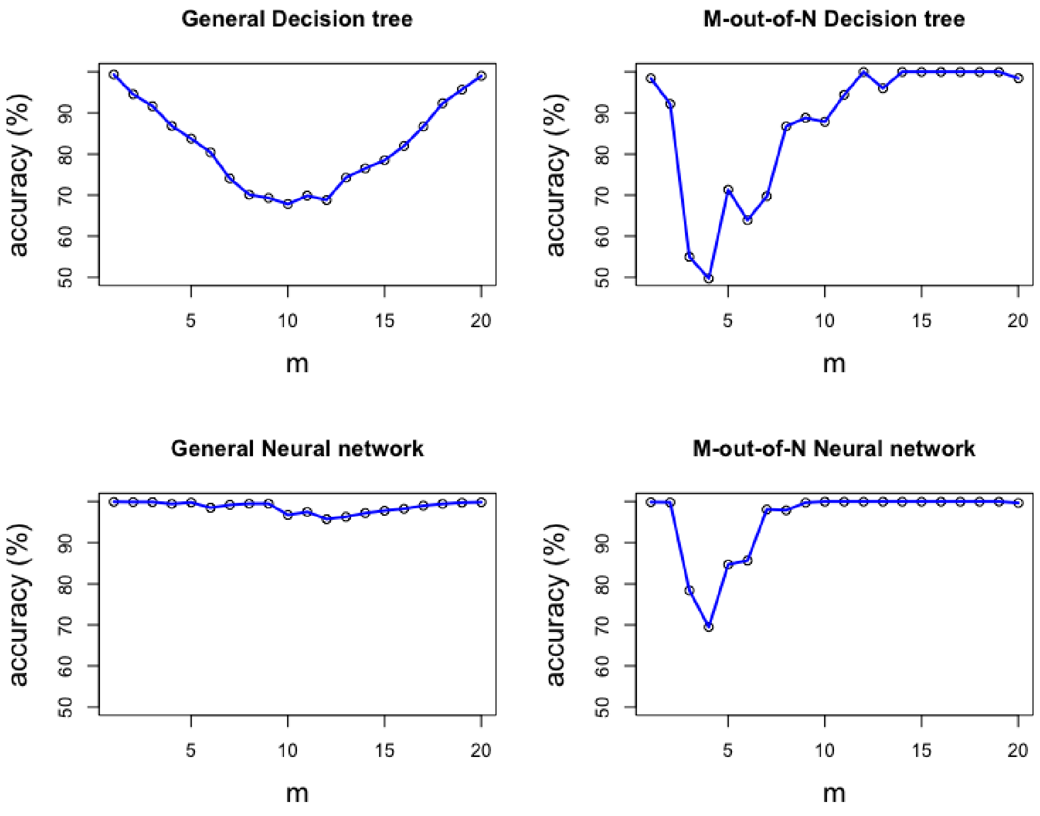


FIGURE 5.4: The accuracy of the decision trees and neural networks on the general and specific M-of-N test set after training on the M-out-of-N condition with 20 variables and varying values of *m*.

5.6 Interaction Results

As results from previous chapters have shown, there can be interactions between the results. For instance, a machine learning algorithm may be able to easily learn two conditions separately, but have difficulty when attempting to learn both at the same time. To investigate this, neural networks and decision trees were trained and tested on datasets that include two of the three symmetrical Boolean functions. First of all, the interaction between the parity function and the exact value function is examined, followed up by the interaction between the parity function and the M-out-of-N function, and lastly the interaction between the M-out-of-N function and the exact function will be explored. For all three combinations, the number of variables for each function will be varied between 2 and 10. Just as in the results of the single conditions, the accuracies of the M-out-of-N function and the exact function are averaged over all possible values of *m*. For each interaction, three accuracies will be examined per system: the accuracy on the general test set, the accuracy on a specific test set of the first condition and a specific test set of the second condition.

An overview of the average accuracies of the decision trees and neural networks is given in Table 5.1 and Table 5.2 respectively. These show the mean accuracies for each interaction on each of the different test sets: the general test set, and one test specific test set for each of the two functions of the interaction. What is clear from these tables, is that the mean accuracy of both the neural network and the decision

TABLE 5.1: The mean accuracies of the decision tree on each test set after training on each interaction.

	General accuracy	Function 1 accuracy	Function 2 accuracy
Parity, Exact	98.50	17.48	78.75
Parity, M-out-of-N	99.99	30.36	81.75
M-out-of-N, Exact	99.99	96.00	32.53

TABLE 5.2: The mean accuracies of the neural network on each test set after training on each interaction.

	General accuracy	Function 1 accuracy	Function 2 accuracy
Parity, Exact	99.65	14.59	78.31
Parity, M-out-of-N	100.00	30.59	81.65
M-out-of-N, Exact	99.99	99.88	26.14

tree on the general test set is quite high for each interaction. The accuracies on the specific test sets, however, are almost always a lot lower. With the parity-exact interaction, for instance, the decision tree algorithm preforms with an accuracy of 98.5% on the general test set, but only 17.5% on the parity test set and 78.8% on the exact test set. This is additional proof to back up the claim that a system can perform with a high accuracy, even if it has not learned all of the rules.

More detailed results of the interactions are shown in Figure 5.5, Figure 5.6 and Figure 5.7, one for each interaction. Each figure consists of six heatmaps, one for every test set and system combination. The heatmaps show the accuracy of a system on a test set for different numbers of variables for both conditions. Because there are two conditions in the datasets, each with a number of variables ranging from 2 to 10, the heatmaps are a 9-by-9 grid. The accuracies are color-coded to more easily interpret the results, where a blue color indicates the maximum accuracy, white indicates the average accuracy and red indicates the lowest accuracy on that particular test set. The colors do therefore not always indicate the same accuracy level, but are instead relative.

Parity - Exact interaction

In Figure 5.5, the results of the interaction between the parity function and the exact value function is shown. The graphs are based on 405 data points, as there are 9 possible combinations for the parity function when using 2 to 10 variables, and $9! = 45$ possible combinations in which the exact value function can be set (because of all of the possible values of m). This generates $9 * 45 = 405$ data points.

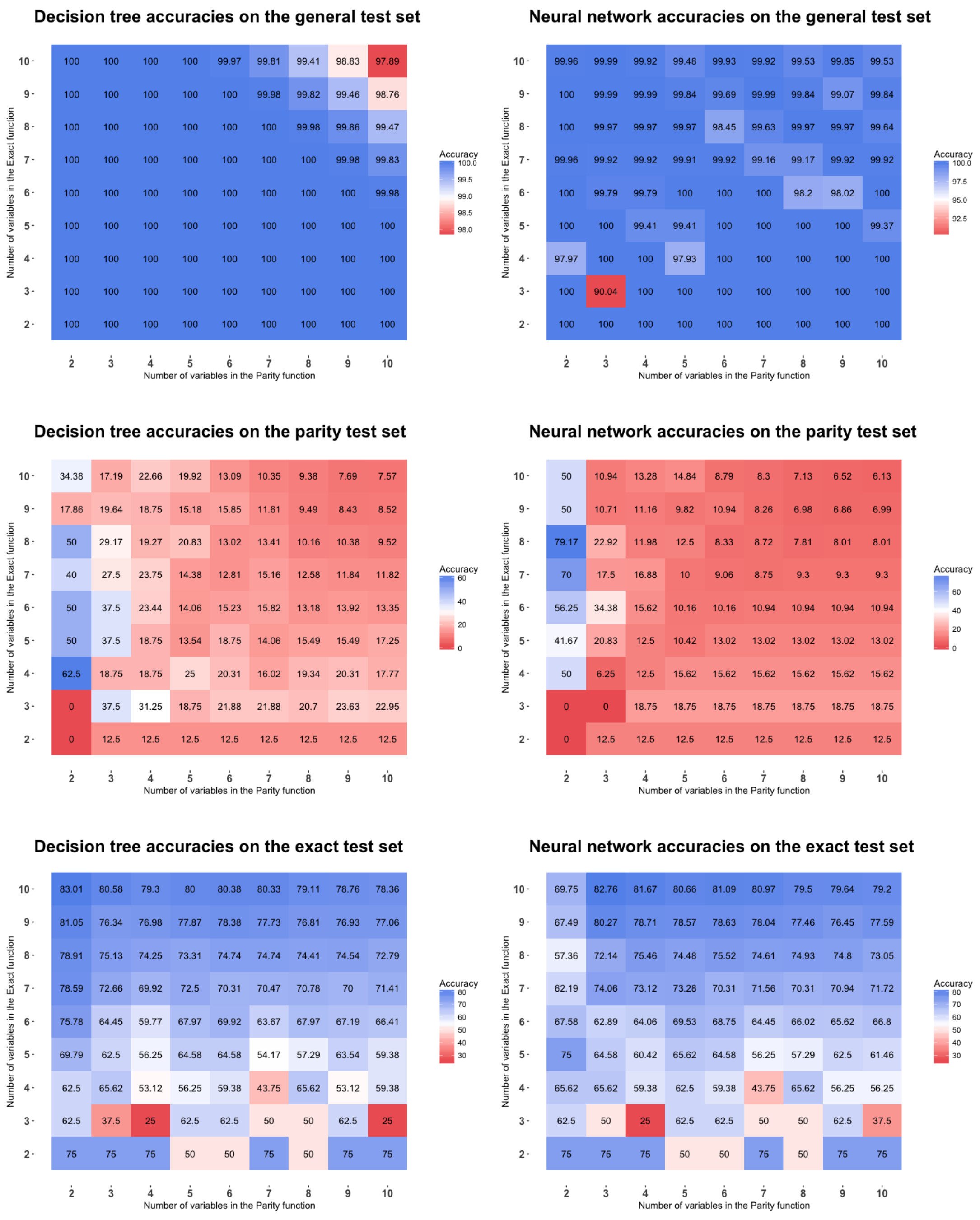


FIGURE 5.5: The mean accuracies of the decision trees (left) and neural networks (right) on different test sets after training on training sets with a parity function (x-axis) and an exact function (y-axis) with varying numbers of variables in each function.

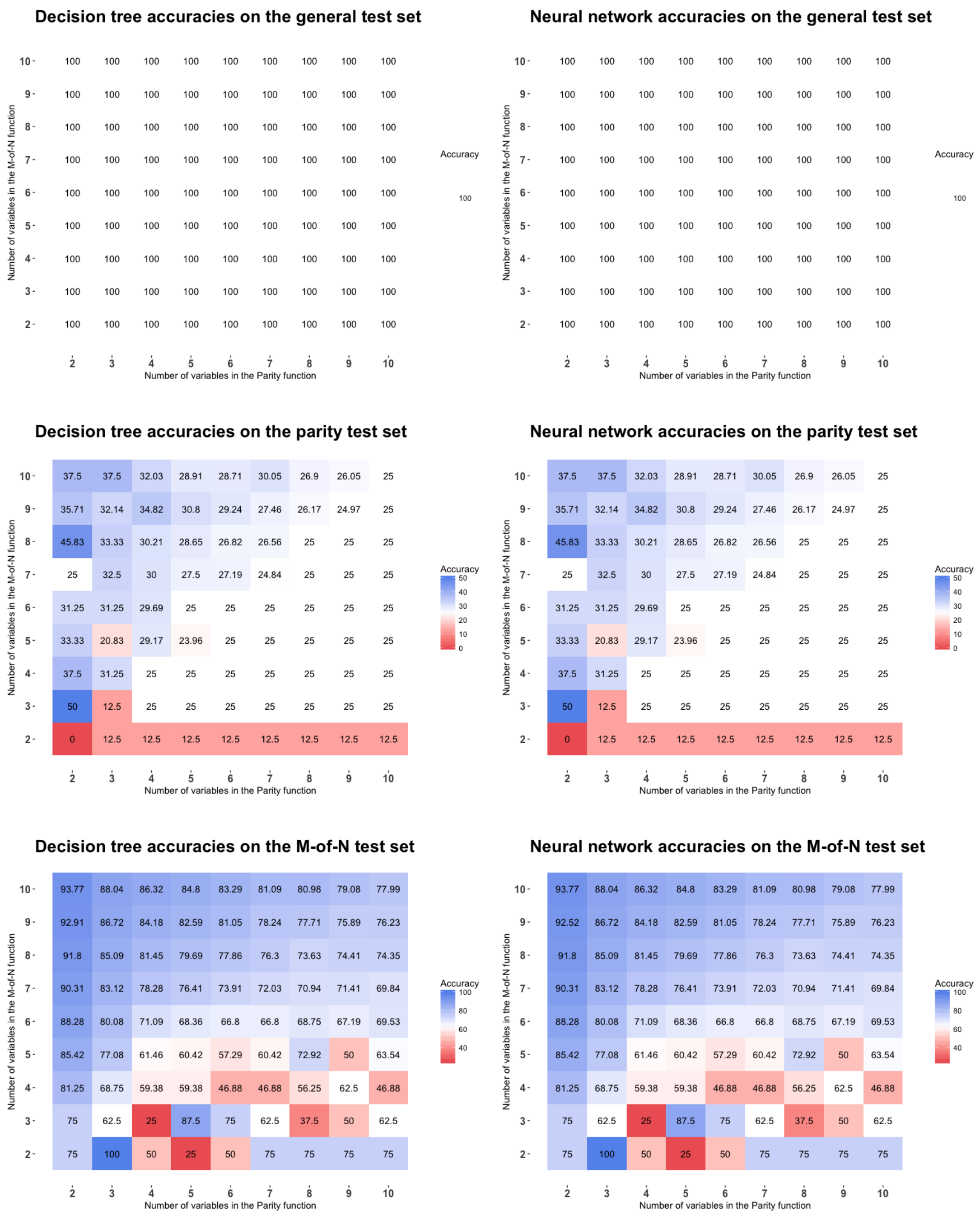


FIGURE 5.6: The mean accuracies of the decision trees (left) and neural networks (right) on different test sets after training on training sets with a parity function (x-axis) and a M-of-N (y-axis) function with varying numbers of variables in each function.

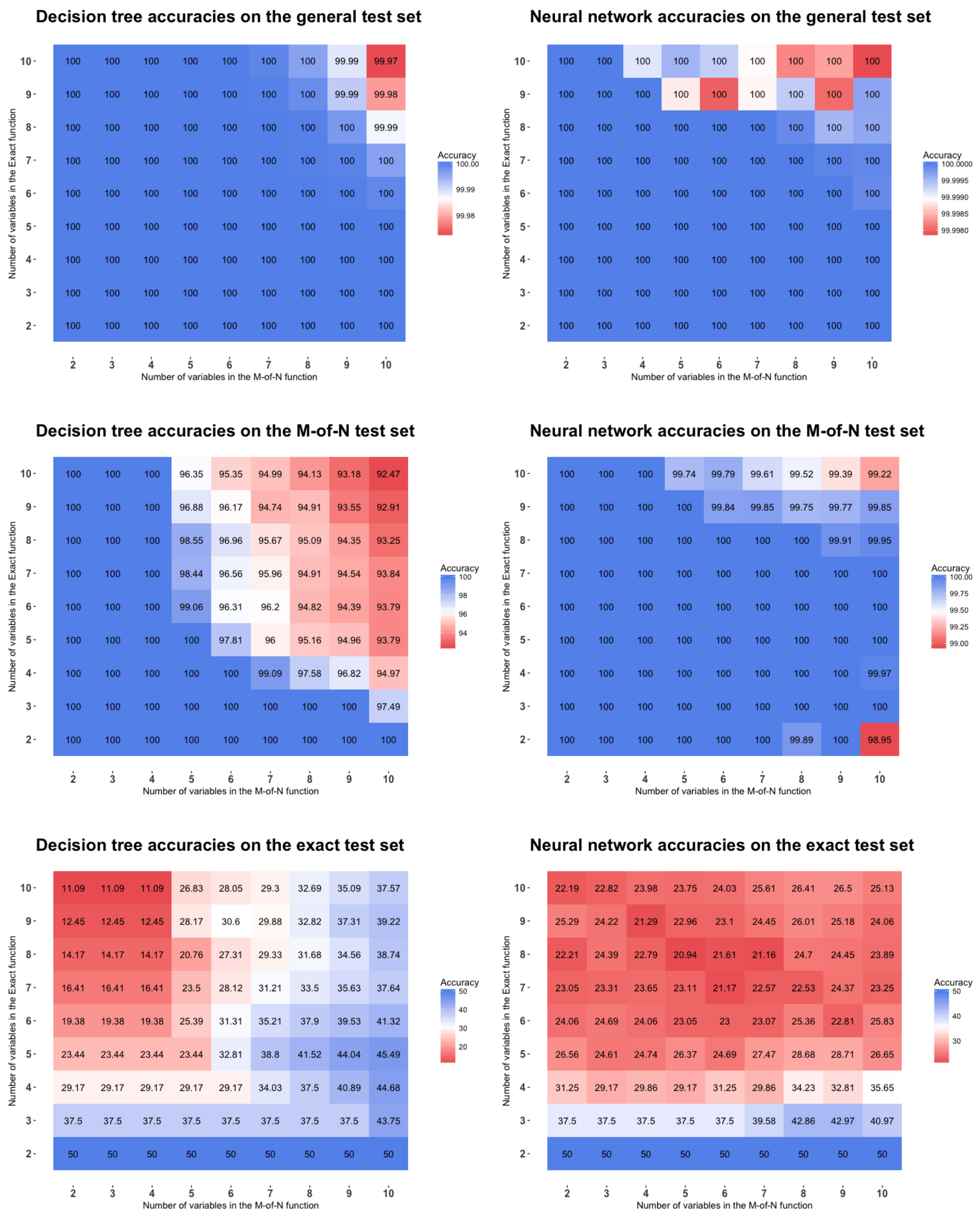


FIGURE 5.7: The mean accuracies of the decision trees (left) and neural networks (right) on different test sets after training on training sets with a M-of-N function (x-axis) and an exact function (y-axis) with varying numbers of variables in each function.

The graph displaying the decision tree accuracy on the general test set (top left) show the expected results; when the number of variables of both functions increase, the accuracy decreases. Furthermore, as long as the total number of variables remains below 15, the decision tree does not appear to have any difficulties with the general test set. The trend of the accuracies of the neural network on the general test set (top right), however, is less clear and there does not seem to be an apparent reason for the lower or higher accuracies at certain positions. The accuracies of the neural network and the decision tree on the parity test set (middle graphs) both show the same pattern: the more parity variables are introduced, the lower the accuracy. The number of exact value variables do not seem to have an impact on how well the parity function is learned by either the decision tree or the neural network. Strangely enough, both systems appeared to have trouble with learning the parity function when both of the functions had only 2 variables. The graphs that display the accuracies on the exact test set (bottom graphs) also show a similar pattern between the decision tree and neural network. It is interesting to see how both systems have difficulty with the exact same combination of variables. For example, both systems have trouble with learning the exact value function if the parity function has 4 variables and the exact value function has 3 variables, or when the parity function has 7 variables and the exact value function has 4 variables. Generally, the systems have a higher accuracy on the exact test set with more exact value variables and fewer parity variables, but this trend is quite noisy.

These are interesting result, as the effect that the number of variables of a function has on how well that function is learned seems to be dependent on the function itself: the parity function is generally learned better by the systems with less parity variables, whereas the exact value function is learned better by the systems with more exact variables. An explanation can be given for both cases. The case of the parity function could be explained by the fact that a function with a larger number of variables is more complex, and thus more difficult to learn. The case of the exact value function can be explained by the idea that with more exact value variables, the effect of the other variables (the parity variables) is reduced, thus making it easier for the system to learn the exact value function. This relies heavily on the assumption that the interference of the parity function has a greater negative impact on how well the exact value function can be learned than the number of variables of the exact value function.

Parity - M-out-of-N interaction

The results of the interaction between the parity function and the M-out-of-N function are shown in Figure 5.6. These graphs are also based on 405 distinct data points, one for each possible combination of variables for the two functions. What is immediately obvious, is that both the neural network and the decision tree scored almost perfectly on the general test set (the accuracies of the decision tree are rounded off to 100%, but are actually slightly lower as seen in Table 5.1). Even though the accuracies on the general test set are high, it appears that the systems are not able to accurately learn the right rules, as evident by the accuracies on the specific test sets. The accuracies on the parity test set (middle), show that the systems achieve higher accuracies in learning the parity function if the number of parity variables is low, just as in the interaction with the exact value function. Interestingly, however, it also appears that a high number of M-out-of-N variables increases the accuracy with which the systems learn the parity function. On the M-out-of-N test set, the same trend can be found: a high number of M-out-of-N variables and a low number of parity variables causes the highest accuracy in learning the M-out-of-N rule. This is the same effect as found in the parity-exact interaction, where a high number of parity variables has a negative influence on learning the parity function, but a high number of exact/M-out-of-N variables has a positive influence on learning the exact/M-out-of-N function. The overall accuracy trends of the neural network are very similar to those of the decision tree on all of the test sets.

M-out-of-N - Exact interaction

In Figure 5.7, the results of the interaction between the M-out-of-N function and the exact value function are shown. These graphs were made from a total of 2025 data points, as both functions have $9! = 45$ possible configurations with 2 to 10 variables ($45 * 45 = 2025$). The accuracies on the general test set (top graphs) show the expected results: more variables in both of the functions creates a lower overall accuracy. The accuracies of the neural network are rounded up, but are actually slightly lower than 100%, as evident from the mean accuracy in Table 5.2. The accuracies on the M-out-of-N test set (middle graphs) show a similar trend, where the accuracy decreases as the number of variables of each function increases. This relationship is clearly seen in the accuracies of the decision tree, and less so in the accuracies of the neural network. In the latter results, the accuracy tend to decrease mostly as the number of variables in the exact value function increases, rather than when the number of M-out-of-N variables increases. The results of the decision tree on the exact test set (bottom left) show a increase in accuracy when the number of exact value variables decreases and the number of M-out-of-N variables increases. A similar trend can be observed in the neural network on the same test set (bottom right), but the positive effect that the number of M-out-of-N variables has on the accuracy is not present.

Discussion on interactions

TABLE 5.3: This table shows the the effect that a high number of variables of a function has on the performance of a system (in terms of learning the function) for all functions that it interacts with.

Performance on ↓	Parity	Exact	M-out-of-N
Parity	-	negative	negative
Exact	positive	-	negative
M-out-of-N	positive	negative	-

There are some interesting findings in these results. First of all, it shown once again that a high performance in terms of classification accuracy does not guarantee that the system has learned the rules that define the dataset, as evident by Table 5.1 and Table 5.2. Secondly, when dealing with interactions, the number of variables of a function can either increase or decrease how well a system learns the function, depending on the other function that is in the dataset. The accuracy on the parity test set increases with a low number of parity variables in both the interaction with the exact value function and the M-out-of-N function. The accuracy on the exact value test set increases with a high number of exact variables in the interaction with the parity function and with a low number of exact variables in the interaction with the M-out-of-N function. Similarly, the accuracy on the M-out-of-N test set increases with a high number of M-out-of-N variables in the interaction with the parity function, and with a low number of M-out-of-N variables in the interaction with the exact value function. The effects of a high number of variables in a function on how well the system is able to learn the function shown in Table 5.3. The table shows that higher number of parity variables always have a negative effect on the how well the parity function is learned, regardless of other the functions that define the dataset. This is also true for the other two functions, except when interacting with the parity function; when interacting with the parity function, a high number of exact/M-out-of-N variables has a positive effect. Earlier results already showed that the parity function is more difficult to learn than the other two functions. This, combined with the finding of Table 5.3 support the idea that was made earlier: the interference of the parity function has greater negative impact on the how well the the exact and M-out-of-N function are learned than the number of variables of said function. This is could explain why the increase in variables creates a positive change in learning the exact and M-out-of-N functions in combination with the parity function. Lastly, in these interactions there are certain combinations of numbers of variables that cause distinct increases or decreases in performance,regardless of the system used and without any apparent reason. A great example of this phenomena can seen in the accuracies on the M-of-N test set in Figure 5.6 (bottom). These heatmaps are almost identical in 'shape', and there are certain combinations that stand out in both graphs in terms of accuracy, without any particular reason. Because there are multiple points and they exist in the graphs of both systems, it is unlikely that they are occurred by chance.

5.7 The effect of training data

Previous research showed that a neural network should theoretically be able to learn any parity function, given that it has sufficient nodes and training data (Wilamowski, Hunter, and Malinowski, 2003). The results in Figure 5.2B shows that the neural network used in this experiment is able to learn the parity function, and the M-out-of-N and Exact value function, quite well. These network consist out of 3 hidden layers, with 25, 10 and 3 hidden nodes respectively, and are trained until they either converge or are presented with the training data for 2000 times. Training the network with less training instances, only 500, results in a poorer performance, as seen in Figure 5.3 and Figure 5.4. Therefore, it has already been established that machine learning systems perform worse on these functions with less training data. The exact relation between the performance and the number of training data, however, is not yet known. This experiment aims to investigate precisely that; the accuracy of neural networks and decision trees on symmetrical Boolean functions for different training dataset sizes.

Datasets will be generated for each of the three symmetrical Boolean functions, with a number of variables varying between 2 and 14. For the M-out-of-N function and the exact value function, the value of m will be varied as well, to include all of the possible combinations of m and n. Just as in previous experiments, the accuracies of these two functions will be aggregated, such that the average accuracy of a given n is taken across all possible values of m. The size of the training dataset will be varied between 100 and 3000 training instances, with steps of 100 instances.

In Figure 5.8 the average accuracies on the general test set across all possible number of variables versus the training set size is shown for each of the three functions and both systems. No significant difference was found between the accuracies on the general test set and the specific function test sets, therefore the latter is excluded from the results. Unsurprisingly, the accuracies of both systems on all three functions increase with more training data. The increase in accuracy versus the training set size somewhat resembles a type of root function relationship. Most of the graphs display accuracy curves that are quite smooth, whereas the accuracies of neural network on the parity function creates a more jagged line. An initial explanation for this might be that the each data point of the M-out-of-N function and exact function that is shown in the graph is averaged across all possible values of m, whereas the data points in the graphs of the the parity function are based on a single value. However, the accuracies of the decision tree in the parity function are subjected to the same constraints, but do form a smooth curve. When comparing the performance of the decision tree and neural network on the parity function, it is clear that the decision tree performs quite a lot better than the neural network. On the other two functions, however, the neural network performs slightly better than the decision tree in terms of accuracy.

Using non-linear regression, models were created for each of the graphs in Figure 5.8. As stated earlier, the curves of the graphs seemed to represent a root function, therefore models were created using the root function: $acc = a * dbs^{\frac{1}{b}}$, where acc is the accuracy on the test set, dbs is the training database size (number of instances) and a and b are constants. The equations of these models, along with their correlations with the original data (Pearson), can be found in Table 5.4. Judging by the correlations, most of the models fit the data quite well. This is especially true for the models of the decision tree data, where all correlations were higher than 0.98. The neural network data appears to be more difficult to predict, as least with these models, especially the accuracies on the M-out-of-N and Exact value test sets.

TABLE 5.4: The root functions founds using non-linear regression and their correlation with the data.

Function	System	Model Equation	Correlation
Parity	Neural Network	$acc = 32.21 * dbs^{\frac{1}{11.16}} + 48.93$	0.92
Parity	Decision Tree	$acc = 49.08 * dbs^{\frac{1}{13.98}} + 67.33077$	0.99
M-out-of-N	Neural Network	$acc = 96.08 * dbs^{\frac{1}{187.46}} + 96.44396$	0.70
M-out-of-N	Decision Tree	$acc = 74.23 * dbs^{\frac{1}{29.73}} + 85.20275$	0.99
Exact	Neural Network	$acc = 70.41 * dbs^{\frac{1}{21.90}} + 80.54066$	0.89
Exact	Decision Tree	$acc = 62.23 * dbs^{\frac{1}{19.10}} + 77.79121$	0.99

In order to extract models with a higher correlation with the data, a few other equations were used to generate models and evaluate their correlation. By far the best results were gained when using the Michaelis-Menten equation, which is an equation that was originally used to explain enzymatic reactions in biology (Michaelis and Menten, 1913). The resulting equations of the models are in the form of $acc = \frac{a*dbs}{b+dbs}$, where acc is the accuracy on the test set, dbs is the training database size (number of instances) and a and b are constants. The exact equations and their correlations

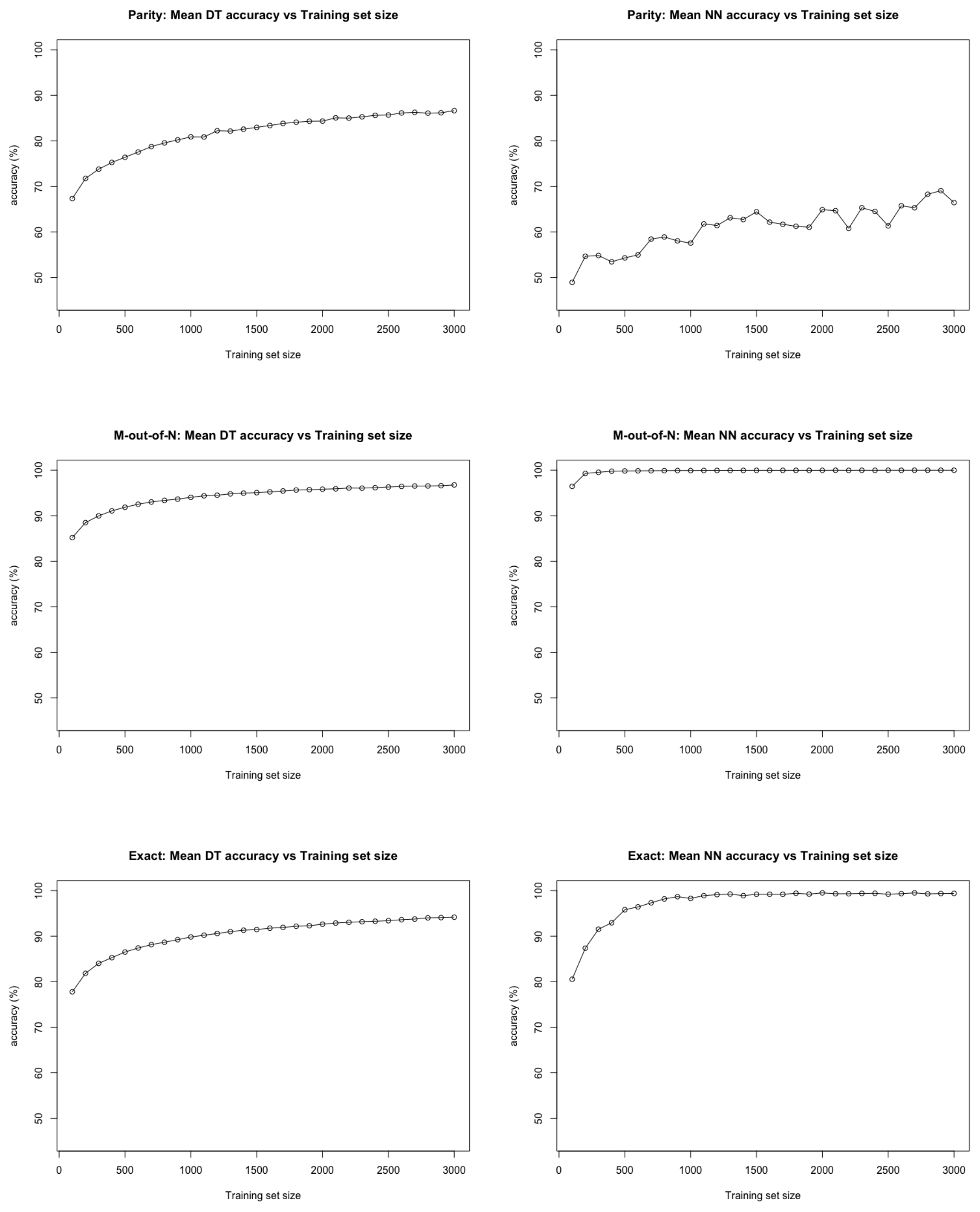


FIGURE 5.8: The mean accuracies of the decision trees (left) and neural networks (right) on the general test sets versus the size of training dataset, for each function.

with the original data are given in Table 5.5. When examining the models of the decision tree data, the correlations between the Michaelis-Menten models and the original data is roughly the same as the correlations between the root function models and the original data. The correlations with the neural network data, however, are much higher with the Michaelis-Menten function than with the root function. The data of the neural network on the M-out-of-N test sets proves to be the most difficult to model, which is most likely due to the ceiling effect in the accuracy as seen in Figure 5.8: the graph reaches 100% accuracy after relatively few training instances and remains at that accuracy value with increased training set sizes. The Michaelis-Menten models are plotted against the original data in Figure 5.9. These graphs quite neatly show where the models fail to predict the right accuracies. For the decision tree data, the models are not able to predict the sharp increase at the low training set sizes, instead it predicts an accuracy that is too high for a system that is trained on only 100 instances. With regards to the neural network data, it is clear that a high correlation with the parity data cannot reasonably be achieved due to the slightly scattered accuracy values. In the graph of the M-out-of-N set, the ceiling effect can be observed. The model is also not able to accurately capture the curve of the Exact value function data either, yet manages to capture a number of data points, thus gaining a high correlation.

TABLE 5.5: The Michaelis-Menten functions founds using non-linear regression and their correlation with the data.

Function	System	Model Equation	Correlation
Parity	Neural Network	$acc = \frac{25.39*dbs}{1411.64+dbs} + 48.93$	0.93
Parity	Decision Tree	$acc = \frac{24.88*dbs}{874.91+dbs} + 67.33$	0.99
M-out-of-N	Neural Network	$acc = \frac{3.84*dbs}{131.74+dbs} + 96.44$	0.84
M-out-of-N	Decision Tree	$acc = \frac{13.74*dbs}{575.22+dbs} + 85.20$	0.99
Exact	Neural Network	$acc = \frac{21.8*dbs}{291.8+dbs} + 80.54$	0.95
Exact	Decision Tree	$acc = \frac{20.18*dbs}{704.91+dbs} + 77.79$	0.99

In order to further investigate the relation between the training dataset size, the number of nodes and the accuracy, the heatmaps of Figure 5.10 are made. These display the accuracies for every combination of dataset size and number of variables in the function. Blue indicates a high accuracy, whereas red indicates a low accuracy. The dataset size is displayed on the x-axis and the number of variables is displayed on the y-axis.

The general trend of the accuracies, is that the systems perform better with less variables and more training data, which is not surprising. This indicates that more complex functions (functions with more variables) require more training data to be learned successfully. The accuracies on the decision tree create a smooth gradient in terms of accuracy, whereas the differences in the accuracies of the neural network are more distinct and pronounced. For instance, in the accuracies of the neural network on the parity function (top right), it is clear that the neural network is able to either learn the function (accuracy $\approx$ 100%) or not at all (accuracy $\approx$ 50%). Just as in Figure 5.8, we can see that the decision tree is able to adequately learn the parity functions with more variables and less training data than the neural network can. The opposite is true for the M-out-of-N and Exact value function, where the neural network is able to learn the functions with a higher accuracy than the decision tree in almost every single combination of training set size and number of variables. When taking into account the accuracy for a given combination of the number of variables and training dataset size, both the decision tree and the neural network seem to be able to learn the M-out-of-N function most easily, followed up by the exact value function. Both systems appear to have the most difficulties with learning the parity function.

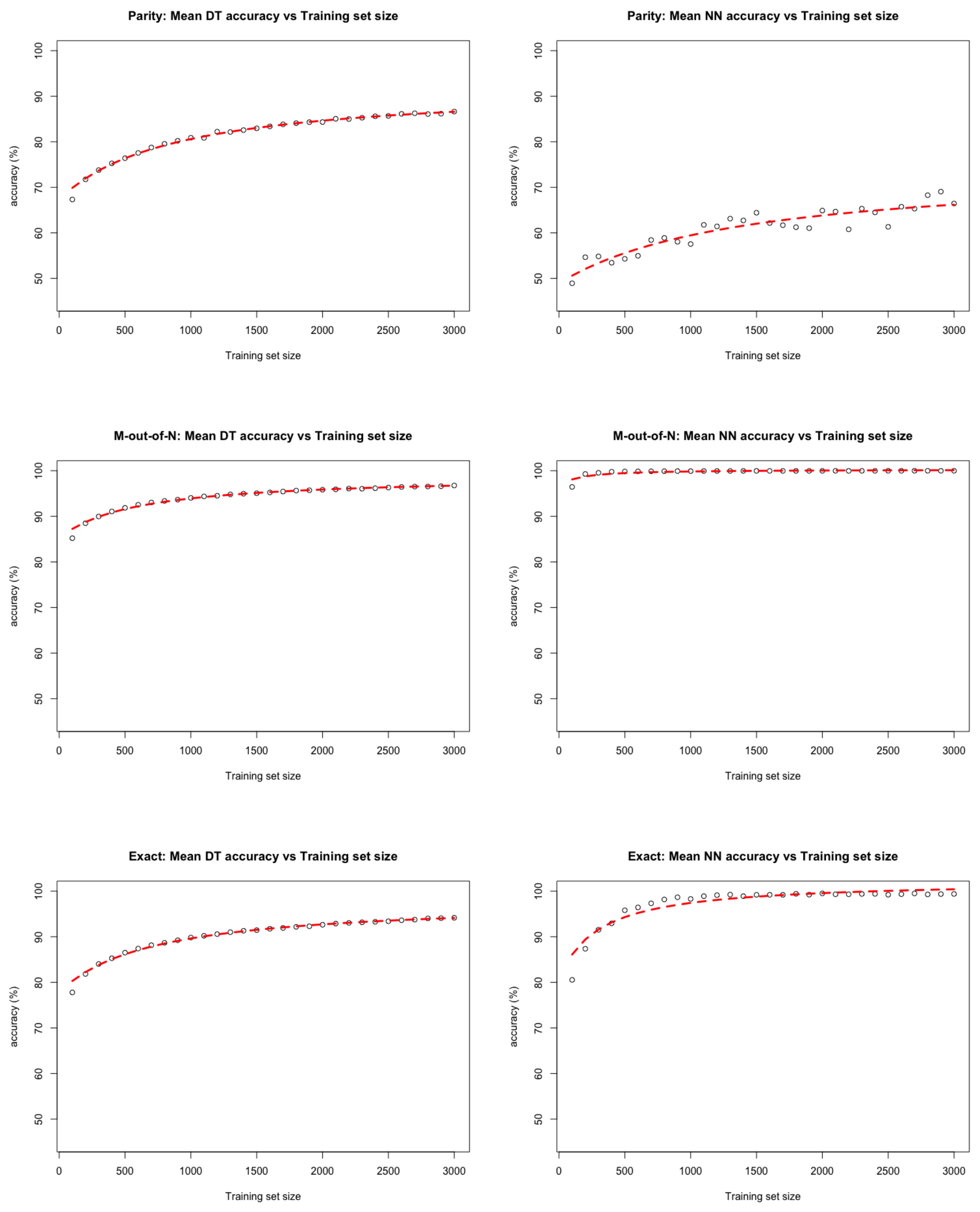


FIGURE 5.9: The mean accuracies of the decision trees (left) and neural networks (right) on the general test sets versus the size of training dataset, for each function. The red dashed line indicate the predicted accuracies of the models.

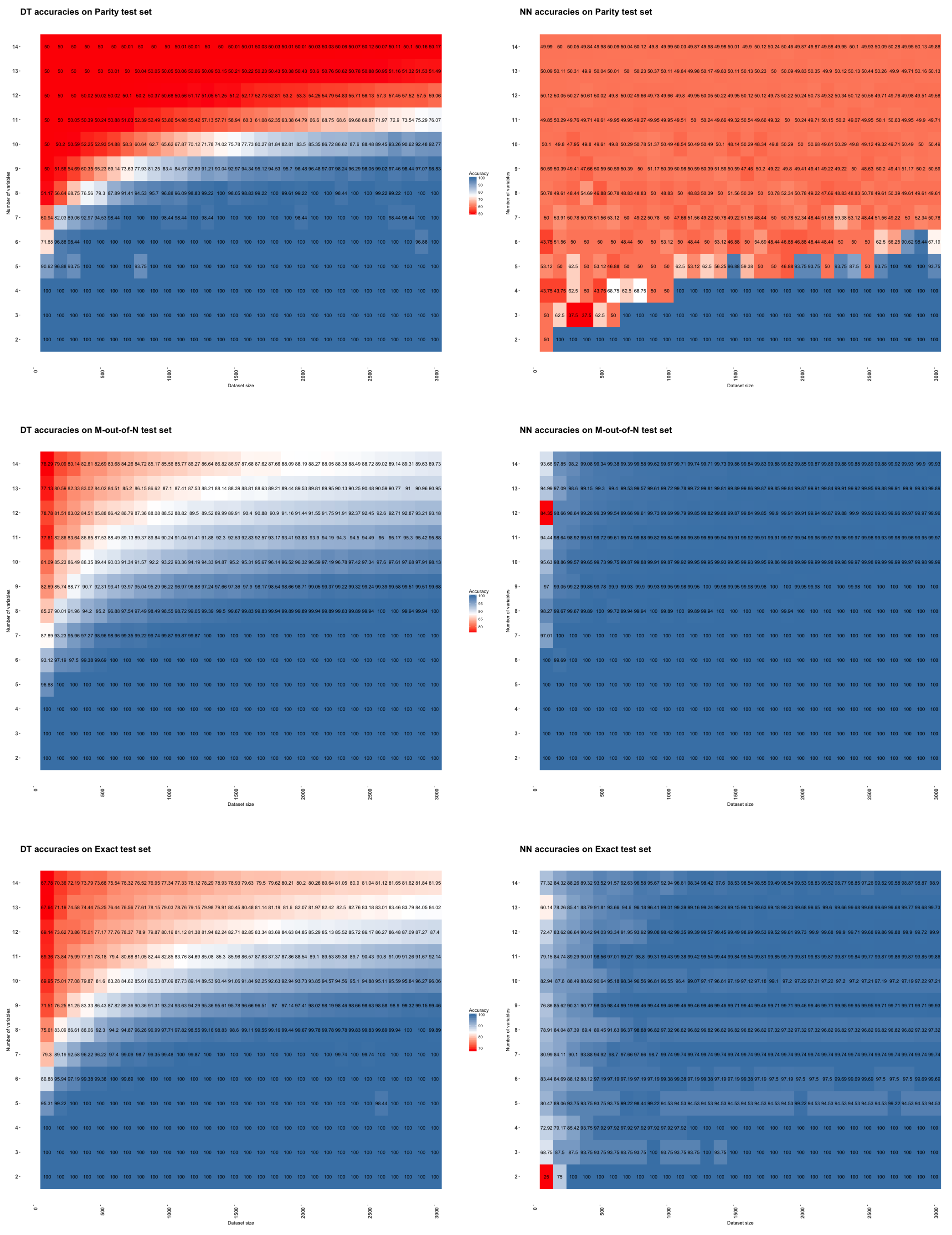


FIGURE 5.10: The mean accuracies of the decision trees (left) and neural networks (right) on the general test set versus the size of training dataset and the number of variables of the function, for each function.

Two interesting points can be seen in the heatmaps of the neural networks. First of all, in the accuracies on the exact test set, the most bottom-left square is significantly lower than its surrounding squares (only 25% accuracy). This is the result of the neural network trying to learn the exact value function with two variables, where exactly one out of the two must be true, but not both, with only 100 training instances. This is also the definition of the parity function, which neural networks are known to struggle with when provided with insufficient training data and can thus explain the low performance. The second interesting observation is the low performance of the neural network on the M-out-of-N function with 12 variables (accuracy of 84.35%), after training on 100 training instances. This is interesting as the performance on the M-out-of-N function with 13 and 14 variables is higher. Upon closer inspection, there appear to be two outlier in the data that have caused this mean accuracy. An accuracy of only 58% was scored by the network on the function with 12 variables out of which 7 or more had to be true, and on the same function where 8 or more had to be true. Retraining and testing the network on these two particular cases yielded results varying from 45% to 80%, which indicates that the network is only able to learn the function sometimes with such a small amount of training instances.

This illustrates another points where decision trees and neural networks differ: with a given amount of training instances, a decision tree will either always learn the function or it will always not be able learn the function, whereas the neural network can sometimes learn the function, and cannot learn the function at another time with the exact same settings. This is due to the different ways in which neural networks and decision trees learn. The CART algorithm that the decision tree system uses is therefore more reliable in the results that it yields. The gradient descent that the neural network uses, on the other hand, is less predictable as it can get stuck in local minima. When it does not get stuck, however, it can often learn rules and achieve higher accuracies with less training data than the decision tree, as evident from the M-out-of-N results and exact value results in Figure 5.10.

5.8 The Influence of Noise in learning Symmetrical Boolean Functions

The datasets used in these experiments were artificially created such that the label of every instance can be calculated based on the input variables. There are no instances that do not follow the strict conditions that define the dataset. In most real life datasets, however, this is usually not the case; noise can cause the underlying structure of the dataset to become less clear, making it more difficult for machine learning techniques to extrapolate them. In order to create a more practical and realistic grasp of the learnability of symmetrical Boolean functions, noise will be applied to the data sets. This is the same type of noise as in the previous two chapters, where a percentage of the instances of the dataset will be altered, such that the value of one of its variables is the opposite of what it initially was. This type of noise was applied to the training sets and the general tests in varying levels, ranging from 0% to 100%. Both the neural networks and the decision tree algorithm were trained on training sets and tested on test sets featuring a single condition. The parity function, M-out-of-N function and exact value function were explored, with a number of variables N ranging from 2 to 10 and all possible values of M. The mean accuracies of all possible combinations of N and M were used to create the graphs in Figure **??**, which display the mean accuracies on multiple test sets for both decision trees and neural networks versus the level of noise.

What is immediately clear from Figure 5.11, is that the accuracies of the decision tree and neural network appear to be quite similar and they always follow a similar trend. The only exception are the accuracies on the general test set of the parity function and the accuracies on the parity test set. On all of the function specific test sets (right side of Figure 5.11) the accuracies drop as the level of noise increases. This indicates that the systems have more difficulties with learning the functions as more noise is introduced. Comparatively,learning the M-out-of-N function is most prone to the influence of noise as it decreases the slowest with more noise. It is followed up by the exact function, which drops until 50%, and then the parity test which drops to almost 0% with a decision tree at 100% noise. This difference in accuracy change can be explained through the fact that instances of some functions are more prone to have an incorrect label after the introduction of a noise variable then instance from another function. For example, an instance from the M-out-of-N function that yields a true label, can have more than M amount of 1's in its input vector. This means that if one of its 1's is turned into a 0 through noise, the output is still true. When the output of an M-out-of-N function is false and the number of ones is lower than $M - 1$ and one of the 0's in its input vector is changed into a 0 through noise, the output is still false as well. Therefore noise does not always alter the output of an instance. This is more often true for M-out-of-N instances than for exact value, or parity functions. An instance of the exact value function can still yield a false output if one of its input variables is changed from a 0 to a 1 or vice versa, but an instance of the exact value function that yielded a true output cannot do so any more once one of its input variables is changed from a 0 to a 1 or vice versa. This is because the number of 1's in the input vector of an exact value function needs to be equal to M and once it yields a true output, it can only yield a false output if one of its variables is changed. Lastly, the output of a parity function is determined by whether the number of 1's in the input vector is odd. With 100% noise, however, each instance has a variable that is changed from 0 to 1, or form 1 to 0. This means that an instance with an odd number of 1's (which would yield a true output) now has an even number of 1's, and an instance with an even number of 1's (which would yield a false output) now has an odd number of 1's. Therefore, at 100% noise, the systems will learn that an even number of 1's yields a true output and an odd number of 1's yields a false output. This can cause an accuracy of 0%.

The phenomena that was just described can also explain the graphs on the left hand side of Figure 5.11, which at a first glance may seem quite strange; after a certain point the accuracies on the general test sets seem to increase as more noise is added to the system. This is because the general test sets also contain noise, even though they contain instances are different from the training set. With enough noise, a new type of structure or rules can be created in the data. For example, at 100% noise, the parity function effectively changes into a function wherein an even number of 1's in the input vector returns true rather than false. The systems simply learn this new structure, and are thus able to perform quite well on the general test sets. This explains the parabola shape in Figure 5.11A. The same effects occur to a lesser extend in the accuracies on the general test sets of the M-out-of-N and exact value functions, because they are more prone to the noise and do therefore not create a completely new structure.

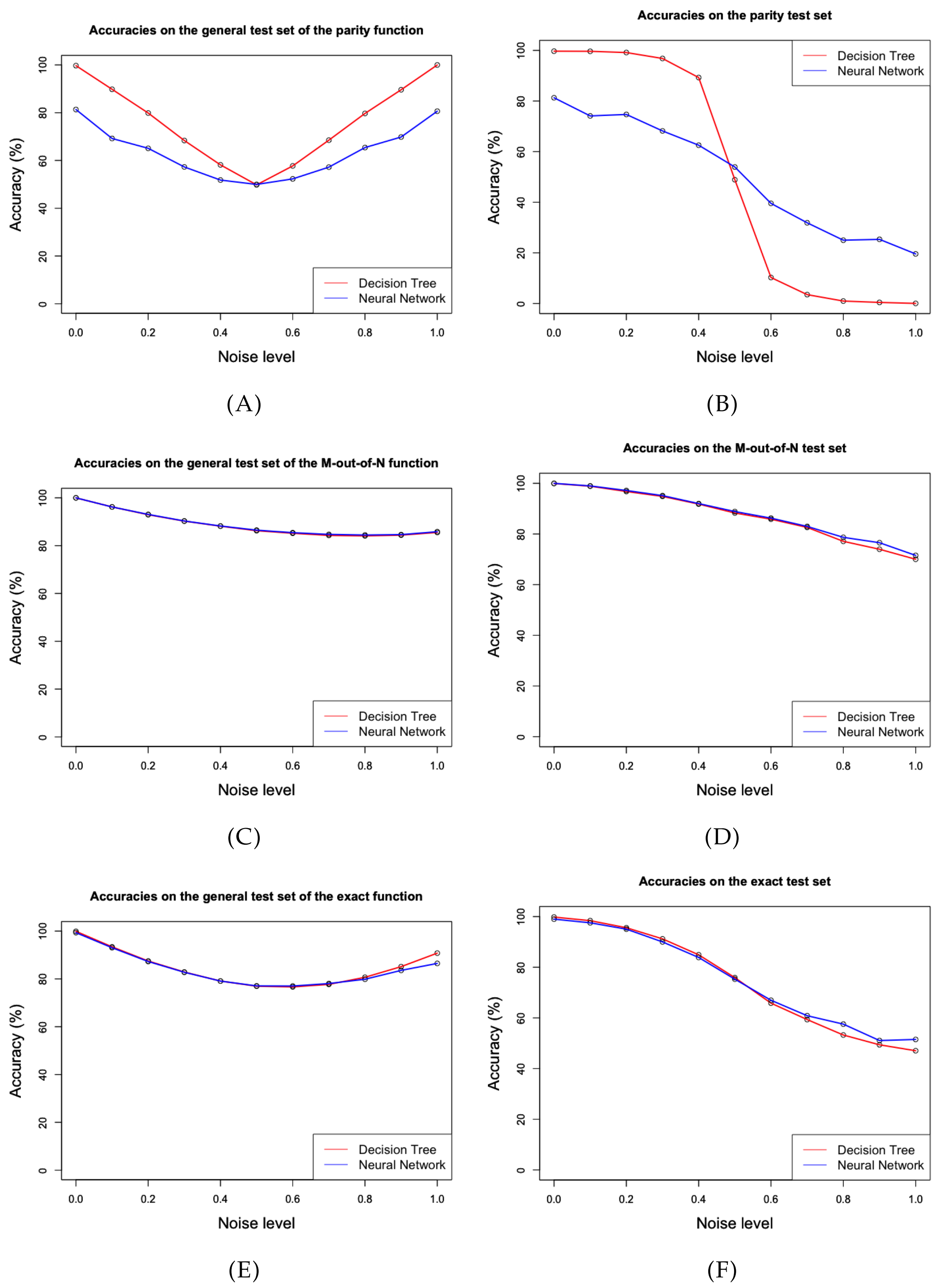


FIGURE 5.11: The effect of varying levels of noise on the accuracies of the decision tree algorithm (red) and the neural network (blue) after training on the parity function (A and B), the M-out-of-N function (C and D) and the Exact value function (E and F). The graphs on the left show the accuracies on the general test set, whereas the graphs on the right display the accuracies on the function specific test sets.

5.9 Conclusion

This section examined the learnability of three symmetrical Boolean functions: the parity function, the M-out-of-N function and the exact value function. With sufficient training data, both decision trees and neural networks are able to learn these functions. Functions with more variables are more difficult to learn by both machine learning systems than functions with less variables. The easiest of the three function to learn is the M-out-of-N function, followed up by the exact value function and then the parity function. Given a set amount of training data, a decision tree is able to learn more complex parity functions than the neural network. The opposite is true for the M-out-of-N and exact value function, where the neural network is able to learn more complex functions for a given number of training instances. The most difficult exact value rule to learn for a given number of variables N, is when $M = \frac{N}{2}$ for both systems. Similarly, the most difficult M-out-of-N rule is when M is equal to roughly $\frac{N}{4}$.

When combining two functions, interaction effects can occur that cause the systems to be unable to learn functions that they are able to learn individually. With datasets with two functions, the accuracies on the general test sets are generally quite high, but testing on test sets that are specifically created to determine how well the systems are able to learn the rules that make up the dataset, show significantly worse results. This supports the claim that a high classification accuracy is no guarantee that the right rationale is used in the classification process (Bench-Capon, 1993). A higher number of variables of a functions in an interaction scenario can either increase or decrease how well that function is learned, depending on the type of function that it is, and the type of function that it interacts with.

Training the systems with various training set sizes showed that the functions are not all learned equally well for a given training set size. The accuracy versus training set size trends of the decision trees can be modelled relatively well using the Michaelis-Menten function. The data of the neural network is more difficult to model, but can be approximated using similar equations. Due to the different ways in which they learn, decision trees and neural networks respond differently to changes in the training set size. Decision trees respond more predictably and will improve slightly with more training data. Neural networks either learn the function very well, or not at all, depending on whether or not it gets stuck in a local minima. For a given training set size, neural networks will therefore not always produce the same results. Despite this unpredictability, however, neural networks are often able to learn rules at small training set sizes, where decision trees are unable to do so.

Chapter 6

Learning Tort Law using Neural Networks

Human beings are able interpret the world at different levels of abstraction. For instance, we are able to mentally represent a cat as a animal with four legs, a tail and whiskers. This allows us to correctly recognize an object as a cat, even if we have never seen that cat before. In other words; by creating an abstract notion of an object, we are able to recognize instances of those objects without having to have seen all of its variations. Being able to reason at different levels of abstraction is therefore a useful skill to have.

In machine learning, the generalization of concepts has been achieved most successfully by deep neural networks in image recognition tasks (He et al., 2016). In handwriting recognition, for instance, a system should be able to identify new characters based on their general shapes, rather than through a pixel-by-pixel comparison with characters that it has seen before. To achieve this, multiple layers of hidden nodes are used, based on the idea that each hidden layer represents a more abstract version of the image. For the handwriting recognition task, this could mean that the first hidden layers represents pixel clusters and edges, whereas the next hidden layers represent basic shapes and the last hidden layer represents the abstract notion of a complete character. This is, of course, an idealistic and simplified example, as most deep learning handwriting recognition systems consist of more than three hidden layers. However, it does introduce a correlation between the depth of a network and its level of abstraction; the more hidden layers the network has, the higher its supposed level of abstraction and thus its performance. In this chapter, that notion will be put to the test, by training a large number of networks with various numbers of hidden layers on a dataset with different levels of abstraction.

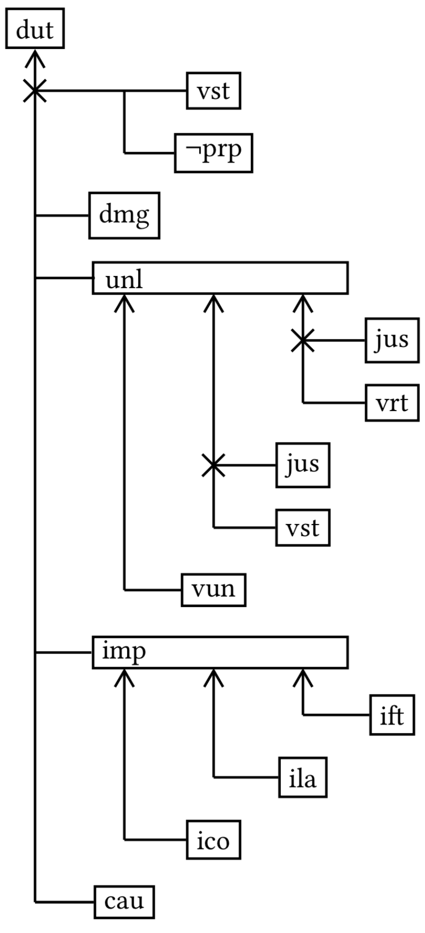
6.1 Dutch Tort Law

The data that this experiment uses is based on Dutch tort law, following the paper of Verheij, 2017. In particular, the wrongful act will be examined. Article 6:162 and article 6:163 of the Dutch civil code describe when a wrongful act is committed and whether the damages that the act caused must be repaired:

Art. 6:162 BW. 1. A person who commits an unlawful act toward another which can be imputed to him, must repair the damage which the other person suffers as a consequence thereof.
2. Except where there is a ground of justification, the following acts are deemed to be unlawful: the violation of a right, an act or omission violating a statutory duty or a rule of unwritten law pertaining to proper social conduct.
3. An unlawful act can be imputed to its author if it results from his fault or from a cause for which he is answerable according to law or common opinion.

Art. 6:163 BW. There is no obligation to repair damage when the violated norm does not have as its purpose the protection from damage such as that suffered by the victim.

These laws were summarized in the argumentative model in Figure 6.1A, alongside its elementary propositions as seen in Figure 6.1B (Verheij, 2017).



(A) Arguments and their attacks in the domain of Dutch tort law.

dut	There is a duty to repair someone's damages.
dmg	Someone has suffered damages by someone else's act.
unl	The act was unlawful.
imp	The act can be imputed to the person that committed the act.
cau	The act caused the suffered damages.
vrt	The act is a violation of someone's right.
vst	The act is a violation of a statutory duty.
vun	The act is a violation of unwritten law against proper social conduct.
jus	There exist grounds of justification.
ift	The act is imputable to someone because of the person's fault.
ila	The act is imputable to someone because of law.
ico	The act is imputable to someone because of common opinion.
prp	The violated statutory duty does not have the purpose to prevent the damages.

(B) Elementary propositions in the domain of Dutch tort law.

FIGURE 6.1: Arguments and their attacks (A) and their elementary propositions (B) in the domain of Dutch tort law (Verheij, 2017).

Figure 6.1 illustrates that there is a duty repair someone's damages (*dut*) if there were damages (*dmg*), if the act was unlawful (*unl*), if the act can be imputed to the person that committed the act (*imp*) and if the act caused the suffered damages (*cau*). This relates to article 6:162.1 and can be written as $dmg \wedge unl \wedge \text{imp} \wedge \text{cau} \rightarrow \text{dut}$. This argument can be defeated, however, if the act is a violation of a statutory duty (*vst*) and the violated norm has as its purpose the protection from damages ($\neg$ *prp*), as written in article 6:163.

An act can be unlawful (*unl*) and imputed to a person (*imp*) through a number of different arguments, as written in article 6:162.2 and 6:162.3 respectively. Unless justified (*jus*), the violation of a right (*vrt*) or a violation of a statutory duty (*vst*) is unlawful (*unl*). If the act is a violation of unwritten law against proper social conduct(*vun*), it is always unlawful (*unl*). An act can be imputed to the person that committed the act (*imp*) if it is because of common opinion (*ico*), because of the law (*ila*) or because it is the person's fault (*ift*). From a purely logical point of view, this would render the notion of unlawfulness (*unl*) and the notion of imputability (*imp*) redundant, as they can be defined by their arguments. Through these notions, however, a level of abstraction is introduced that makes it easier to represent and understand the law.

6.2 Experimental set-up

In the experiment, neural networks are trained on a set of ort law cases, and are tasked with predicting whether or not there is a duty to repair someone's damages in new cases. In particular, the experiment focuses on how well the networks can learn the rules of tort law as described in Figure 6.1. When higher levels of abstraction are present in the datasets, a deeper neural network is presumed to be more effective (He et al., 2016). This concept will be put to the test, by training networks on the tort law dataset with varying number of hidden layers and a varying number of nodes. The hypothesis states that the performance of networks with more hidden layers will be higher for any given total amount of nodes in the network than networks with less hidden layers, and that networks with more hidden layers require less nodes to perform perfectly.

6.2.1 Datasets

The datasets used in the experiment are generated according to the arguments as shown in 6.1. The instances of these datasets consist of the 11 Boolean variables as shown in the figure, including the variable *dut*, which determines whether or not there is a duty to repair someone’s damages, but excluding the abstract notions of unlawfulness (*unl*) and imputability (*imp*). The *dut* variable is provided with a true or false value based on the values of the other ten variables and the rules from Figure 6.1.

In total, three databases are generated. First of all, a training set of 3000 instances is generated, where the variables are given values such that the value of *dut* is true for half of the instances and false for the other half. The second dataset is a general test set of 2000 instances, where variables are given values such that half of the values of *dut* are true the other half are false as well, just as in the training set. Lastly, a unique dataset is constructed which consists of all of the possible value combinations for the other ten variables. This unique dataset consists of 1024 instances (2^{10}), out of which there are 912 instances where *dut* is false and 112 instances where *dut* is true. Only around 11% of the instances therefore have a *dut* value that is true, as opposed to the 50% of the training set and general test set. This unique dataset will be used to measure how well the neural network is able to learn all of the rules.

6.2.2 Neural Networks

A variety of different networks are used in the experiment. All of the networks are trained on the training set using a mini-batch approach with a batch size of 50 and a constant learning rate of 0.05. The networks are trained for at most 2000 rounds (where the entire training set is presented to the network in each round) or until the networks have converged. A network has converged when the error remains constant over 10 rounds. Networks have either one, two or three hidden layers with varying amount of nodes in each hidden layer.

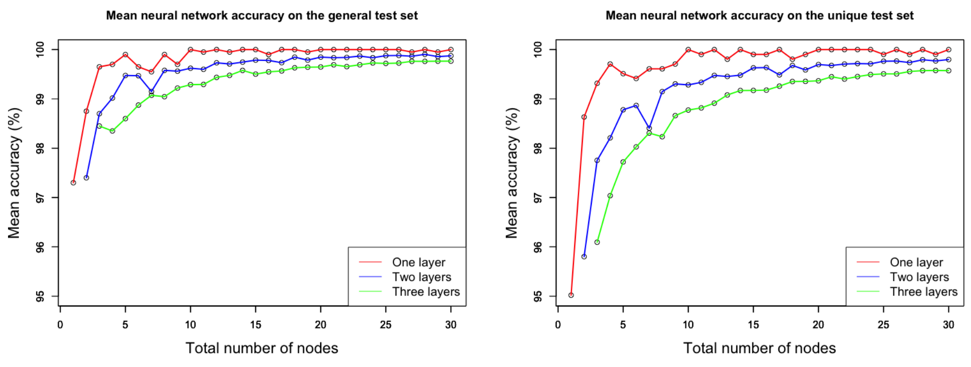
In this experiment, the total number of nodes of a network is varied between 1 and 30 across one, two or three hidden layers. All possible distributions of the nodes across the hidden layers will be evaluated; e.g. with 3 nodes and 2 hidden layers, both the network with 1 node in its first hidden layer and 2 nodes in the second, and the network with 2 nodes in its first hidden layer and 1 node in the second will be evaluated.

6.3 Results

The performance of the networks are tested on both the general test set and the unique test set. Thirty networks are trained with one hidden layer, with a total number of nodes varying between 0 and 30. With two hidden layers, there are 435 possible networks where the total number of nodes in the network is 30 or less. Similarly, 4060 networks with three lhidden ayers were trained, which includes all of the possible combinations with 30 or less total nodes. The average accuracies versus the total number of nodes in the one, two are three hidden layered networks are shown in Figure 6.2. Figure 6.2A displays the accuracies on the general test set, whereas Figure 6.2B displays the accuracies on the unique test set.

First of all, it is clear that the accuracies on the general test set (Figure 6.2A) are higher than the accuracies on the unique test set (Figure 6.2B). In the general dataset, 50% of the instances have a *dut* value that is true, and the other 50% of the instances have *dut* values that is false. The general dataset therefore contains duplicates and a fixed distribution, which is how datasets are usually constructed. In previous chapters and research, it was shown that with these type of datasets, it is possible for a system to reach a high accuracy without having learned all of the rules (Bench-Capon, 1993). The unique dataset, however, contains all possible unique instances, meaning that the performance on the unique test set reflects how well the system is able to learn all of the rules that define the dataset. This difference in performance between the general test set and the unique test set are therefore expected.

In both of the graphs in Figure 6.2, there is a difference in accuracy across the total number of nodes between the networks with one, two and three hidden layers. The mean accuracies of the network with one hidden layer is always higher than the



(A) General test set

(B) Unique test set

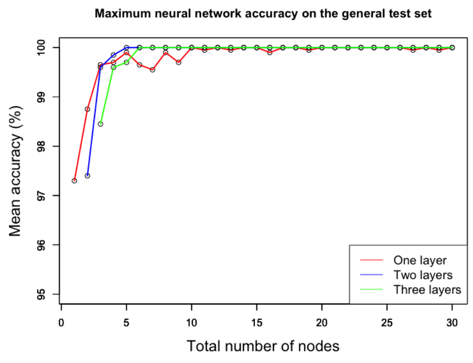
FIGURE 6.2: The mean accuracy of the neural networks on the general test set (A) and the unique test set (B) versus the total amount of nodes in the network, for networks with one (red), two (blue) and three (green) hidden layers.

mean accuracies of the network with two hidden layers, which in turn are higher than the mean accuracies of the network with three hidden layers. This appears to be in contrast with the hypothesis, which proposed that the performance of networks with more hidden layers will be higher for any number of nodes. These results seem to suggest the opposite; networks with more hidden layers have a lower average accuracy for any number of nodes.

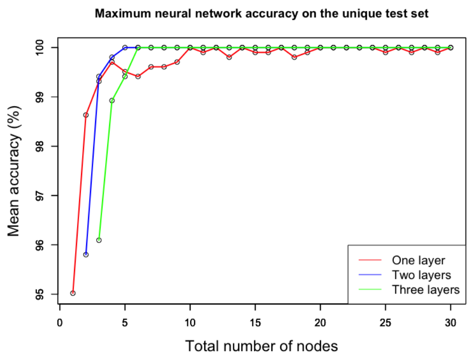
Because the number of possible node configurations over the hidden layers increases as the number of hidden layers increases, the comparisons in Figure 6.2 might not be completely fair; some node configurations might perform quite poorly, thus causing a causing a drop in the average accuracy for that number of nodes. Perhaps a better assessment could be made by comparing the the maximum accuracies for each number of nodes. This would display the performance of the networks by the number of nodes for the optimal node configuration across the hidden layers. The graphs displaying the maximum accuracy for each number of nodes on both the general and unique test set are shown in Figure 6.3

The first thing that should be noticed in Figure 6.3, is that the accuracies of the one hidden layer network are the same as in Figure 6.2. This is because there is only one way to configure the nodes in a network with one hidden layer for any total number of nodes. Once again, the accuracies in Figure 6.3A are higher than in Figure 6.3B, for the same reason as described earlier. In both of the graphs, a clear ceiling effect can be observed after six nodes. Additionally, it appears that with more than three nodes, the optimal configuration of a two hidden layer network yields a higher accuracy than the one hidden layer network. The network with three hidden layers reaches the ceiling (100% accuracy) faster than the network with one hidden layer on both test sets, but not as fast as the network with two hidden layers. When configuring the network in its optimal fashion, the optimal number of hidden layers seems to be dependent on the total amount of nodes in the network; with 1 or 2 nodes, the one hidden layer network performs best, with 3, 4 or 5 nodes, the two hidden layer network performs best. It cannot be confirmed that a network with 6 or more nodes performs better with a three hidden layer network (which would confirm a trend), as both the two hidden layer and three hidden layer network hit a ceiling of 100%

after 6 nodes.



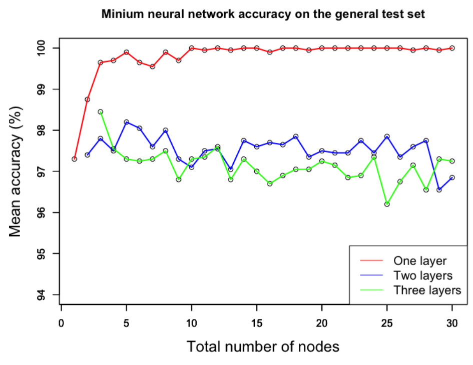
(A) General test set



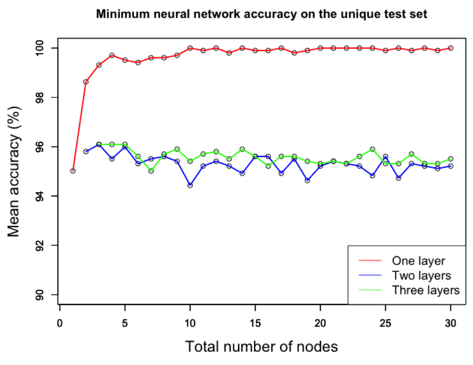
(B) Unique test set

FIGURE 6.3: The maximum accuracy of the neural networks on the general test set (A) and the unique test set (B) versus the total amount of nodes in the network, for networks with one (red), two (blue) and three (green) hidden layers.

For the sake of completeness, the minimum accuracy for each number of nodes on both the general and unique test set are shown in Figure 6.4.



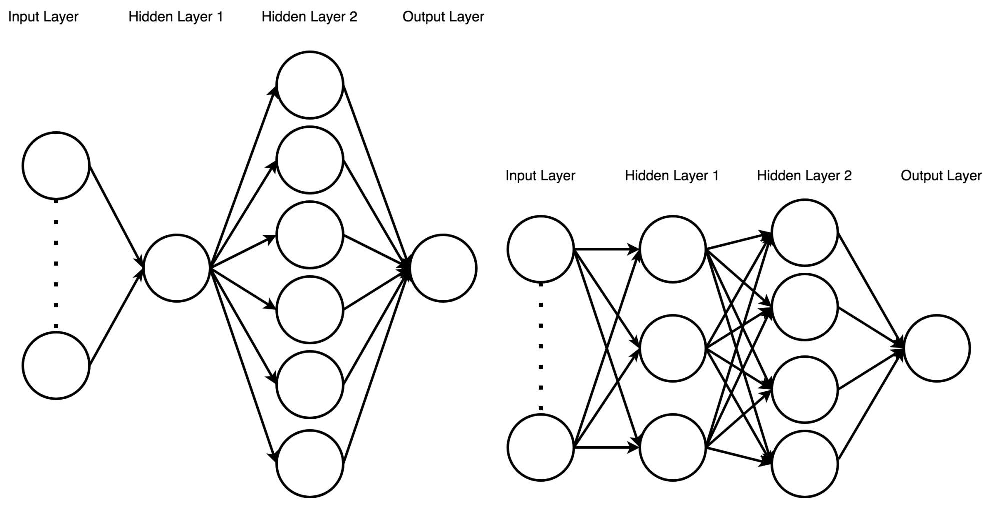
(A) General test set



(B) Unique test set

FIGURE 6.4: The minimum accuracy of the neural networks on the general test set (A) and the unique test set (B) versus the total amount of nodes in the network, for networks with one (red), two (blue) and three (green) hidden layers.

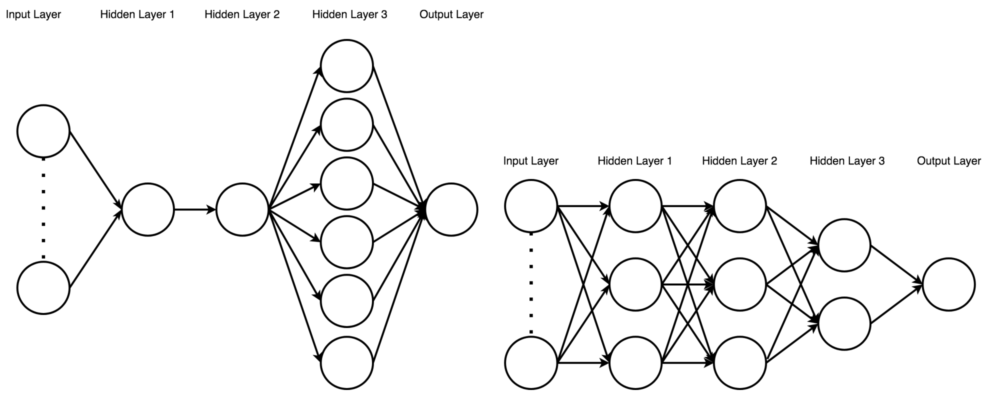
Once again, the accuracies for the one hidden layer network in these graphs are the same as in Figure 6.2 and Figure 6.3. The low accuracies for the two and three hidden layer network illustrate the importance of the node configuration of a network. Upon closer inspection, the worst performance is achieved through node configurations wherein the first hidden layer only has one node. For instance, for a two hidden layer network with a total number of 7 nodes, the worst node configuration is to have 1 node in the first hidden layer and 6 nodes in the second hidden layer, as seen in Figure 6.5A. The optimal node configuration for 7 nodes over two hidden layers is to have 3 nodes in the first hidden layer and 4 nodes in the second hidden layer, as shown in Figure 6.5B.



(A) The worst node configuration (B) The best node configuration

FIGURE 6.5: The best and worst node configurations for a network with seven nodes distributed over two hidden layers.

The same principle applies to networks with more layers as well. For example, the worst configuration of a network with three hidden layers and 8 nodes is to have one node in each of its first two hidden layers, and the remaining six nodes in its third hidden layer, as seen in Figure 6.6A. The optimal configuration with 8 nodes, on the other hand, is to have 3 nodes in each of its first two hidden layers and two nodes in its third hidden layer, as seen in Figure 6.6B.



(A) The worst node configuration (B) The best node configuration

FIGURE 6.6: The best and worst node configurations for a network with eight nodes distributed over three hidden layers.

The best and worst node configurations, for a total number of nodes ranging from 2 to 7 is shown in Table 6.1, alongside the accuracies for each of these configurations. In this table, a [2,1] configuration represents a network with 2 nodes in its first hidden layer and 1 node in its second hidden layer. The same data, but for networks with three hidden layer is shown in Table 6.2, with a total amount of nodes ranging from 3 to 8. In this table we see that in some cases there are multiple configurations that create an optimal or worst configuration. As the total amount of nodes increases, the number of configurations that provide optimal results increases drastically as well, which is why not all of the data is shown in the tables. The complete

TABLE 6.1: The best and worst node configurations for a two layered neural network and their accuracies for a given number of total nodes.

Total number of nodes	Best configuration	Best accuracy	Worst configuration	Worst accuracy
2	[1,1]	95.8	[1,1]	95.8
3	[2,1]	99.4	[1,2]	96.1
4	[3,1]	99.8	[1,3]	95.5
5	[3,2]	100	[1,4]	96.0
6	[4,2]	100	[1,5]	95.3
7	[3,4]	100	[1,6]	95.5

TABLE 6.2: The best and worst node configurations for a three layered neural network and their accuracies for a given number of total nodes.

Total number of nodes	Best configuration	Best accuracy	Worst configuration	Worst accuracy
3	[1,1,1]	96.1	[1,1,1]	96.1
4	[2,1,1]	98.9	[1,1,2], [,1,2,1]	96.1
5	[2,2,1]	99.4	[1,1,3],[1,2,2],[1,3,1]	96.1
6	[4,1,1]	100	[1,4,1]	95.6
7	[4,1,2],[4,2,1]	100	[1,4,2]	95.0
8	[3,3,2]	100	[1,1,6]	95.7

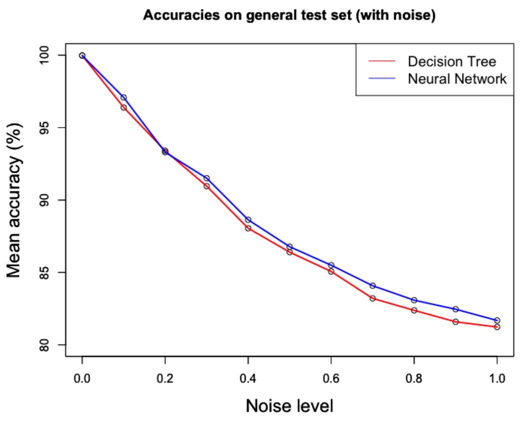
tables for both the two and three hidden layer network for all of the nodes can be found in appendix A. It cannot be said with certainty, however, that the number of optimal configurations will always increase when the total number of nodes increases, because the ceiling effect in terms of classification accuracy most likely has an influence on this correlation as well.

In both Table 6.1 and Table 6.2 the worst configurations all have a single node in its first hidden layer. This effectively summarizes an entire data instance into a single node, causing a loss of information. When applied correctly, going from a large number of nodes to a smaller number of nodes is expected create a higher level of abstraction, which is what is hoped to be achieved. This is why neural networks often employ a triangular shape, where each hidden layer contains less nodes than the hidden layer that preceded it. Most of the best configurations in the tables fit that triangular description, while some, however do not. Interestingly, in Table 6.1 the best configuration with 7 nodes is [3,4], rather than [4,3]. And similarly, in Table 6.2, the best configuration for 8 nodes is [3,3,2], instead of [4,3,1], for instance.

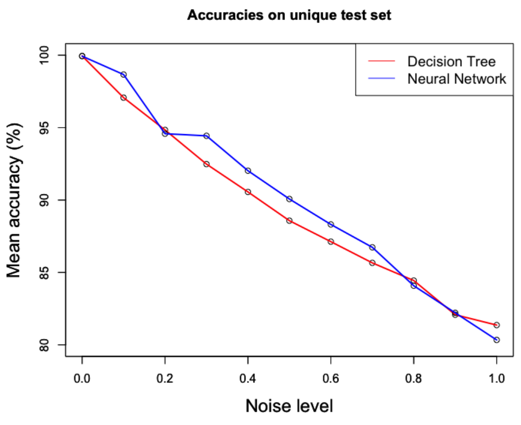
When applying the decision tree algorithm from the previous chapters to the training data, a decision tree is generated that accurately reflects all of the arguments as seen in Figure 6.1. The tree that it generates can be found in Appendix B. Because it is able to learn all of the rules, its accuracy on both the unique and general test set is 100%. Even though the aim of this experiment is to investigate the learning behaviour of neural networks, it should be noted that a decision tree is able to learn these rules more effectively and more quickly without the need to optimize variables.

6.4 The Influence of Noise in the Tort Law problem

As discussed in the previous chapters, real life, non-artificial datasets often contain noise. This noise makes it more difficult for machine learning algorithm to learn the structure of the data. The tort law dataset generated in this experiment is an artificial dataset without any form of noise; the rules that were defined in Article 6:162 and Article 6:163 are used to generate the data, thus leaving no room for mistakes or noise. This is quite unrealistic, however, as in practice human errors can be made. Therefore noise will be included in the training dataset and the general test set, in order to investigate the effects that the noise has on how well systems are able to learn tort law. The same type of noise is used as in previous chapters, wherein a percentage of the instances of the data are changed, such that the value of one of its variables is reversed. In this case, since the variables are all Boolean, noise means changing the value of variable from true to false or from false to true. When a particular percentage of instances are provided with noise, it does not necessarily mean that all of these instances now have the wrong output label. For example, when 10% of the instances are given a noisy variable, only 4% of the instances will have a wrong output label. Both the neural network and the decision tree algorithm are trained on the training dataset with noise, and are tested on the general test set, which also includes noise, and the unique test set, without noise. The resulting accuracies can be seen in Figure 6.7.



(A) Network 1



(B) Network 2

FIGURE 6.7: The effect of varying levels of noise on the accuracies of the decision tree (red) and neural network (blue) on the general test set (A) and unique test set (B).

Both graphs show similar drops in accuracy for both the neural network and the decision tree algorithm. When 100% of all of the instances contain noise, the accuracy drops to around 80% on both the general and unique test set. Therefore, even with a lot of noise, the systems still perform adequately on the tort law problem.

6.5 Conclusion

From these results, we can conclude that neural networks are able to learn the arguments of Dutch Tort law and their attacks, as seen in Figure 6.1. Whether or not the networks actually learn these arguments is dependent on the number of hidden layers in the network, the total amount of nodes in the network, and how these nodes are distributed across the hidden layer. In their optimal node configuration, networks with two or more hidden layers outperform networks with only one hidden layer, granted that there are a sufficient amount of nodes in the network. However, with a small amount of nodes, networks with less hidden layers have shown to perform better than networks with more hidden layers, even in their most optimal node configuration. By far, the worst node configurations are those in which the first, or the first two hidden layers only contain a single node. The best node configurations are generally those with a triangular shape, but exceptions to this rule are present.

Chapter 7

Discussion and Conclusion

7.1 The Importance of an Understandable Rationale

In recent years, machine learning has become a widely applied method in autonomous decision making. This is largely due to the success of statistical based machine learning, which relies on on large sets of data from the past rather than on preprogrammed expert knowledge. These statistical methods, like deep neural networks, have been extremely successful in tasks such as image and speech recognition, in which correct decisions are made over 97% of the time (He et al., 2016). Despite its success in terms of making correct decisions, however, the statistical based machine learning systems are often distrusted by human users, as they do not provide an explanation for the decisions that they make (Edwards and Veale, 2017). The counter movement to these successful, but opaque 'black-box' systems is explainable AI (XAI), which aims to create machine learning systems that are able to explain why they made their decisions (Gunning, 2017).

Examples of XAI systems include rule extraction algorithms, which extract rules or decision trees from trained (deep) neural networks (Hailesilassie, 2016) (Zilke, Mencía, and Janssen, 2016). This makes it so that the reasoning of the machine learning systems can easily be understood, without sacrificing much of the high performances that deep neural networks yield. The premise that this solution is build upon, however, is that the inherent reasoning of the machine learning systems makes sense; the user of the system should be able to understand and make sense of its reasoning. In previous research, it was shown that slightly altering an image with use of perturbations can drastically change the decisions of a machine learning system with regards to that image, even though the differences are unnoticeable by humans (Yuan et al., 2017). This suggests that machine learning systems do not always internalize the structure of the data in the same way as humans would. Instead, it finds different correlations in the data that lead to the correct decisions. This might lead to a high performance, but it is difficult to explain why. This creates a problem for XAI systems; for what is the use of an incomprehensible explanation?

The explanations of the machine learning systems therefore need to be sound, such that a user is able to follow along with the reasoning of the system. Earlier research had already suggested that a high performance accuracy does not guarantee a sound rationale (Bench-Capon, 1993). This means that that a different performance method is required to examine and test the reasoning of the systems. This study aimed to investigate the rationales of machine learning techniques with the use of artificial datasets. These datasets are generated from a set of conditions, which define the structure of the data. The advantage of this approach, is that the structure of the data is known beforehand. This method therefore makes it possible to determine whether the machine learning systems that train on the data learn the 'correct' rules.

7.2 Results

In a replication study of the paper by Bench-Capon (1993), neural networks were trained on fictional datasets that were generated using a set of conditions. The original study shows that the networks are unable to learn some of the conditions, in particular disjunctions of conjunctions (XOR condition), even though the classification accuracy on the test set is quite high. The replication study shows similar performances on the test set, but the networks in this study are also able to learn the XOR conditions that the network in the original study failed to learn. Training on an alternative training set with a different data distribution, which improved the performance of the networks in the original study in terms of how well the conditions were learned, actually has the opposite effect in the current study; the conditions are learned less successfully when training on the alternative dataset. By adding noise to the data, the networks has more issues with learning the conditions. This experiment shows that it is possible for neural networks to learn certain conditions, but it is dependent on the way in which the dataset is set up. Additionally, it reaffirms one of the ideas of Bench-Capon, which stated that a high performance accuracy is no guarantee for a sound rationale.

The same type of experiment was performed with symbolic learning techniques; decision trees and association rules. These systems score just as high as the neural networks in terms of classification accuracy, and are able to easily identify and learn simple Boolean and categorical conditions. The decision tree system also has no problems with the greater-than conditions. They are both unable, however, to extrapolate the disjunction of conjunction (XOR) conditions, which the neural networks are able to learn. When training and testing on only a single disjunction of conjunction condition, the decision tree algorithm is able to learn the condition. This indicates that there might be an interaction between the conditions that makes it more difficult for a system to learn it. Additionally, noise has a strong negative impact on how well the conditions are learned by the decision tree, and less so on the association rule classifier.

In order to investigate what conditions the connectionist and symbolic approaches can learn and how the conditions might interact with each other, another experiment was performed. In that experiment, three symmetrical Boolean functions were examined with varying numbers of variables. Both types of systems are theoretically able to learn each of the three functions individually, if a sufficient amount of training data and a large enough network is used. The most difficult function to learn is the parity function, which is the generalized form of a disjunction of conjunctions (XOR) condition as investigated in the first two experiments. Without any interference of other conditions, decision trees are better suited to deal with the parity functions than neural networks. When training on datasets with multiple functions, the systems are not able to learn each of the functions as well, even though the overall classification accuracy is still quite high. Furthermore, how much the decrease in how well the functions are learned is, is dependent on the type of the functions that interact with each other. The effects of the training data set size on the accuracy and how well the systems can learn the functions were investigated as well. This showed that the accuracy of the decision tree system for various training data set sizes can be modelled quite effectively, as its change in performance is predictable. Predicting the same performance for the neural network system, however, proved to be more difficult, since neural networks can get stuck in local optima. In turn, however, neural networks can often learn functions at smaller training set sizes than decision trees. When adding noise to the training data, the conditions become more difficult to learn. Especially the parity function becomes almost impossible to learn after sufficient noise is added.

The last experiment focused on the different levels of abstraction that a machine learning system should be able to reason at, with the use of a Dutch tort law case. The different hidden layers of a neural network are often seen as different abstract representations of a problem, wherein each hidden layer is a more abstract version of its preceding layer. If that were true, a network with more hidden layers (but with the same amount of nodes) should therefore be more effective in learning issues with higher levels of abstraction. The experiment that was performed showed that, indeed, networks with more hidden layers generally perform better on these problems than networks with less hidden layers with the same total amount of nodes. This is only true, however, if the nodes are distributed over the hidden layers in the most optimal configuration. The optimal distribution of the nodes across the hidden layers generally follows a triangular shape, wherein each hidden layer contains less nodes than its preceding hidden layer. With a very small amount of nodes, networks with less hidden layers also perform better (e.g. with only two nodes, a network with 1 hidden layer will outperform a network with 2 hidden layers).

7.3 Discussion

The first interesting result gained in this study is the neural networks ability to successfully learn the XOR conditions in the welfare dataset from Bench-Capon (1993), even though the networks in the original study are unable to do so. It cannot be stated with certainty as to where these differences in results came from. The original study uses the Aspirin software to create its neural networks, but the details surrounding the exact parameters and set-up of the networks are not mentioned. The networks in this study therefore use a number of different activation functions, types of gradient descent and learning rates, but all of the networks are able to learn the conditions correctly. An even more surprising finding is the differences in accuracy between the original study and the replication study after training on the alternative dataset. Training on this alternative dataset, wherein each instance that was ineligible for a welfare benefit had exactly one condition which was false, yields significantly worse results in terms of how well the XOR conditions were learned. Surely, using such a dataset as a test set creates a better indication of how well each condition is learned. There does not seem to be any obvious reason, however, as to why training on such a dataset should increase the performance of how well a condition is learned. The results after training on the alternative dataset in the replication experiment are explained and discussed in detail in section 3.4. Based on this explanation, the results that are yielded in the replication study seem to make sense. No similar explanation is given for the results in the original study, thus making it difficult to argue for the use of this alternative dataset as a training set.

The symbolic learning techniques are unable to learn the conditions that the neural networks could; both association rules and decision trees are unable to learn the XOR conditions of the welfare dataset. In general, the performance of the association rule classifier is quite low compared to the performances of the decision tree and neural network. Initially, it was hypothesised that perhaps the association rule classifier may create easily comprehensible rules and a relatively high performance, which would have been a successful step forward for XAI. However, the major downside of association rules is the need for discretization; all of the continuous variables of a dataset must first be split up into categorical variables before they can be used by the association rule system. Additionally, association rule systems are unable to easily represent greater-than rules, and instead need to create a rule for every possible value of a particular variable. These facts, combined with the low performance makes associations rules inferior to decision trees in these scenarios.

Unlike association rules, the decision trees are able to learn XOR conditions from the welfare benefit if no other condition is present. This indicates that the other conditions have an effect on how well the XOR condition is learned. This seems sensible, as more conditions create a more complex structure. The way that the CART algorithm of the decision tree works, is to recursively split the dataset based on the values of a variable, such that the gain in information is highest; the classes are separated as much as possible. In a dataset with a number of conditions, splitting on the simple Boolean conditions with only a single variable would therefore separate the classes most effectively. Splitting a dataset on one variable of an XOR condition hardly separates the classes at all; it is only the combination of the two variables of the XOR condition that correctly separates them. With only a single XOR condition in the training and test set, the information gain of splitting on a single XOR variable is quite low as well, but there is no alternative and thus the split occurs.

The experiment with the symmetrical Boolean functions has confirmed that the neural network and the decision tree algorithm can learn the XOR condition and its generalized form, the parity function, without interference from other variables. Compared to the M-out-of-N and exact value function, the parity function seems to be the most difficult function for machine learning systems to learn. Both neural networks and decision trees, however, can learn them with sufficient training. Once multiple conditions are presented to the systems at the same time, however, the systems are unable to learn the conditions as well. The accuracies on the general test sets are still quite high for all possible interactions between the functions, but the systems do not learn the functions themselves as well. The accuracies on the general test set and on each of the functions individually can be seen in Table 7.1 and Table 7.2 (originally Table 5.1 and Table 5.2). It is strange to see how systems can perform with an accuracy of almost 100% on the general test set, yet fail dramatically on the test sets of the individual functions. This indicates that they do not necessarily learn these functions, but rather that they learn a completely different function that somehow accurately maps the input to the output. There must therefore be a confounding structure within the data that the systems choose to learn over the original, intended structure.

In most of the experiments, the performances of the neural network and the decision trees were quite similar. When considering the fact that decision trees are transparent and much more easy to comprehend, they would appear to be superior to the "black-box" neural network. However, the experiments in this study used

TABLE 7.1: The mean accuracies of the decision tree on each test set after training on each interaction.

	General accuracy	Function 1 accuracy	Function 2 accuracy
Parity, Exact	98.50	17.48	78.75
Parity, M-out-of-N	99.99	30.36	81.75
M-out-of-N, Exact	99.99	96.00	32.53

TABLE 7.2: The mean accuracies of the neural network on each test set after training on each interaction.

	General accuracy	Function 1 accuracy	Function 2 accuracy
Parity, Exact	99.65	14.59	78.31
Parity, M-out-of-N	100.00	30.59	81.65
M-out-of-N, Exact	99.99	99.88	26.14

relatively simple functions and conditions, which both connectionist and symbolic machine learning techniques can learn. When dealing with image or speech data, for instance, the (deep) neural networks will almost always outperform the symbolic learning techniques. This is because neural networks are better at generalizing, whereas decision trees attempt to map out the entire training set. Due to these inherent differences, they are generally used for different purposes.

By investigating the influence of the amount of training data, it was shown that the accuracy for any given amount of the training data is largely dependent on the structure of the data and the machine learning system that is used. However, the same general trend seems to occur in all possible combinations of systems and data structure. The fact that the accuracy can be modelled quite effectively using the size of the training dataset could provide insights into how much training should be used in the future. Of course, this is a relatively simple dataset and the models and equations that are generated in this experiment would not be of much use in large deep neural network tasks, but it might be possible to generate the same type of models for more complex datasets.

The main conclusion of the experiment regarding tort law is that the distribution of nodes across the different hidden layers is vital in obtaining the best possible performance in terms of learning the correct rules. Generally, a triangle shape network seems to be the most efficient way of distributing the nodes, in which each hidden layer has less nodes than the proceeding hidden layer. The worst node configurations are those with a single node in the first hidden layer (as seen in Figure 6.5A and Figure 6.6A). These networks effectively reduce the entire input layer into a single node, causing a large reduction in the amount of information that it can use. A triangular shape, on the other hand, reduces the amount of information slowly, until eventually the output layer is reached. This should create a form of abstraction, wherein each layer contains a more abstract representation of the problem than the layer that preceded it. The number of optimal configurations appear to increase as more nodes and hidden layers are available to the system, but due to the ceiling effect in the experiment, this cannot be confirmed.

The datasets that were used in this study were artificially generated using a set of conditions. Without any form of noise, this would result in datasets that do not deviate from the rules at all, thus creating very clean dataset. In the welfare benefit dataset that was used in chapter 3 and 4, a large number of noise variables were added. These were used in an attempt to "throw off" the systems, but all systems largely ignored them. These additional noise variables do not change the rules either, as none of the rules included the noise variables. When applying noise to the variables of the rules of the training sets, however, the effects were more significant. Neural networks, decision trees and the association rule classifier had more difficulty in classifying the instances of the welfare benefit dataset, and they were all less successful in learning the rules when more nose was applied to the training dataset. This is not surprising, but it does provide a more realistic view on the issue, since most real-life datasets contain noise as well. Judging from Figure 5.11, it would seem that noise generally has the same effect on both the decision tree and the neural network, but the effect is different depending on the conditions of the dataset.

7.4 Future Research

These experiments focussed on particular conditions using specific machine learning techniques in a relatively simple environment. As a result, however, more questions can be raised with regards to how well machine learning can learn the underlying structure of the data that they are trained on. Obvious follow-up experiments could deal with different tasks, such as image recognition using adversarial images (Yuan et al., 2017), in order to investigate the rationales used by the systems in these tasks. Working with real datasets, however, makes it difficult to quantify how well their rules or structures are learned by the system, as these structures are often not known beforehand. Future research could also focus on other forms of machine learning, such as random forests or deep learning with more than three hidden layers. With the use of rule extraction algorithms such as deepRED (Zilke, Mencía, and Janssen, 2016) or NeuroRule (Lu, Setiono, and Liu, 1996), the rationales of black-box systems can be studied more closely in order to find out if their reasoning is sound (assuming that the extracted rules accurately reflect the reasoning of the system).

Future research will also require the use of new metrics to measure how well a system is able to learn the structure of the data, rather than relying solely on a performance metric such as the classification accuracy. Because the underlying structure of real life data is often unknown, it is difficult to determine whether the internal rationale that a system creates can be considered correct. With the use of artificial datasets the issue is avoided, as the structure of an artificial dataset is known beforehand. Through this method, it is possible to measure how well a system is able to learn the structure of the data. This is the method used in this study, and for the purposes of the experiments it was sufficient. In future research, however, this may not be the case for every experiment, as the method is limited to simple rule based artificial conditions. When dealing with datasets with more complex structures, such as images, it will be difficult to monitor how well the system has learned the structure, and thus a better performance metric is required.

7.5 Conclusion

This study has shown that machine learning algorithms do not always internalize the structure of their training data as we would expect. In the future, machine learning algorithms that simply execute a task with a high performance will not suffice; an explanation of their decision making will be required (Gunning, 2017). Yet, currently in most statistical machine learning approaches, systems are considered adequate once they reach an accuracy above a certain performance threshold. This is seen as sufficient, despite the fact that their unsound rationales are often based on confounding correlations in the data, rather than on the underlying rules that define the data. Can it then be claimed that explainable AI has been achieved, if it only explains the inner workings of the black-box, regardless of whether or not that explanation is comprehensible? The answer is two-fold. First of all, such a system could seen as useful, because an explanation of an irrational system shows the user that its rationale is unsound, thus allowing the user to determine what to do with the system based on that fact. In other words, it eliminates the Schrödinger-esque uncertainty that currently surrounds the black-box systems and shows us what is inside, even if opening the box can turn out to be disappointing. On the other hand, we want systems to be intelligent, and not just to perform well in arbitrary performance tests. In that case, is not satisfactory for a system to perform well based solely on statistics through correlations in the data, rather than by extrapolating the underlying rules or structures that define the data. In either case, a new age of AI has begun, wherein the performance of the systems are both excellent and understandable.

Appendix A

Best and Worst Node Configurations

TABLE A.1: The best and worst node configurations for a two layered neural network and their accuracies for a given number of total nodes.

Total number of nodes	Best configuration	Best accuracy	Worst configuration	Worst accuracy
2	[1, 1]	95.80078125	[1, 1]	95.80078125
3	[2, 1]	99.4140625	[1, 2]	96.09375
4	[3, 1]	99.8046875	[1, 3]	95.5078125
5	[3, 2]	100	[1, 4]	95.99609375
6	[4, 2]	100	[1, 5]	95.3125
7	[3, 4]	100	[1, 6]	95.5078125
8	[3, 5], [4, 4], [5, 3], [6, 2], [7, 1]	100	[1, 7]	95.60546875
9	[3, 6], [5, 4], [7, 2], [8, 1]	100	[1, 8]	95.41015625
10	[3, 7], [4, 6], [5, 5], [6, 4], [8, 2], [9, 1]	100	[1, 9]	94.43359375
11	[3, 8], [5, 6], [7, 4], [9, 2]	100	[1, 10]	95.21484375
12	[3, 9], [5, 7], [6, 6], [8, 4], [9, 3], [10, 2], [11, 1]	100	[1, 11]	95.41015625
13	[3, 10], [7, 6], [9, 4], [11, 2], [12, 1]	100	[1, 12]	95.21484375
14	[3, 11], [6, 8], [8, 6], [10, 4], [11, 3], [12, 2]	100	[1, 13]	94.921875
15	[3, 12], [4, 11], [5, 10], [6, 9], [7, 8], [8, 7], [9, 6], [10, 5], [11, 4], [12, 3], [13, 2]	100	[1, 14]	95.60546875
16	[4, 12], [5, 11], [6, 10], [7, 9], [8, 8], [11, 5], [12, 4], [14, 2], [15, 1]	100	[1, 15]	95.60546875
17	[3, 14], [7, 10], [8, 9], [11, 6], [12, 5], [13, 4], [15, 2], [16, 1]	100	[1, 16]	94.921875
18	[4, 14], [5, 13], [6, 12], [7, 11], [8, 10], [9, 9], [11, 7], [12, 6], [13, 5], [14, 4], [15, 3], [17, 1]	100	[1, 17]	95.5078125
19	[4, 15], [5, 14], [6, 13], [8, 11], [9, 10], [10, 9], [11, 8], [12, 7], [13, 6], [16, 3], [17, 2], [18, 1]	100	[1, 18]	94.62890625
20	[3, 17], [4, 16], [5, 15], [6, 14], [7, 13], [8, 12], [9, 11], [10, 10], [11, 9], [12, 8], [14, 6], [15, 5], [16, 4], [17, 3], [18, 2], [19, 1]	100	[1, 19]	95.21484375
21	[3, 18], [4, 17], [5, 16], [6, 15], [7, 14], [8, 13], [10, 11], [11, 10], [12, 9], [14, 7], [18, 3], [19, 2], [20, 1]	100	[1, 20]	95.41015625
22	[3, 19], [4, 18], [5, 17], [7, 15], [10, 12], [12, 10], [13, 9], [15, 7], [17, 5], [18, 4], [19, 3], [20, 2], [21, 1]	100	[1, 21]	95.3125
23	[4, 19], [5, 18], [6, 17], [7, 16], [8, 15], [9, 14], [10, 13], [11, 12], [12, 11], [13, 10], [14, 9], [16, 7], [18, 5], [20, 3], [21, 2]	100	[1, 22]	95.21484375
24	[3, 21], [7, 17], [8, 16], [9, 15], [10, 14], [11, 13], [12, 12], [13, 11], [14, 10], [15, 9], [16, 8], [18, 6], [19, 5], [20, 4], [21, 3], [23, 1]	100	[1, 23]	94.82421875
25	[3, 22], [4, 21], [5, 20], [6, 19], [7, 18], [9, 16], [10, 15], [11, 14], [12, 13], [15, 10], [16, 9], [17, 8], [18, 7], [19, 6], [20, 5], [21, 4], [22, 3], [23, 2], [24, 1]	100	[1, 24]	95.60546875
26	[3, 23], [4, 22], [5, 21], [6, 20], [8, 18], [9, 17], [10, 16], [11, 15], [12, 14], [13, 13], [14, 12], [15, 11], [16, 10], [17, 9], [18, 8], [19, 7], [20, 6], [21, 5], [22, 4], [23, 3], [24, 2], [25, 1]	100	[1, 25]	94.7265625
27	[7, 20], [8, 19], [9, 18], [10, 17], [11, 16], [12, 15], [14, 13], [15, 12], [16, 11], [17, 10], [18, 9], [19, 8], [20, 7], [21, 6], [22, 5], [23, 4], [24, 3], [25, 2], [26, 1]	100	[1, 26]	95.3125
28	[3, 25], [4, 24], [5, 23], [6, 22], [8, 20], [9, 19], [10, 18], [11, 17], [12, 16], [13, 15], [14, 14], [15, 13], [16, 12], [17, 11], [18, 10], [19, 9], [21, 7], [22, 6], [23, 5], [25, 3], [26, 2], [27, 1]	100	[1, 27]	95.21484375
29	[4, 25], [5, 24], [6, 23], [7, 22], [8, 21], [9, 20], [10, 19], [11, 18], [12, 17], [13, 16], [14, 15], [15, 14], [16, 13], [17, 12], [19, 10], [20, 9], [21, 8], [24, 5], [25, 4], [26, 3], [27, 2], [28, 1]	100	[1, 28]	95.1171875
30	[3, 27], [4, 26], [5, 25], [6, 24], [8, 22], [9, 21], [11, 19], [12, 18], [13, 17], [15, 15], [16, 14], [17, 13], [18, 12], [19, 11], [20, 10], [21, 9], [22, 8], [23, 7], [24, 6], [25, 5], [26, 4], [28, 2], [29, 1]	100	[1, 29]	95.21484375

TABLE A.2: The best and worst node configurations for a three layered neural network and their accuracies for a given number of total nodes.

Total number of nodes	Best configuration	Best accuracy	Worst configuration	Worst accuracy
3	[1, 1, 1]	96.09375	[1, 1, 1]	96.09375
4	[2, 1, 1]	98.92578125	[1, 1, 2], [1, 2, 1]	96.09375
5	[2, 2, 1]	99.4140625	[1, 1, 3], [1, 2, 2], [1, 3, 1]	96.09375
6	[4, 1, 1]	100	[1, 4, 1]	95.60546875
7	[4, 1, 2], [4, 2, 1]	100	[1, 4, 2]	95.01953125
8	[3, 3, 2]	100	[1, 1, 6]	95.703125
9	[3, 4, 2], [4, 1, 4], [4, 2, 3], [4, 4, 1], [5, 1, 3], [5, 2, 2], [6, 1, 2], [6, 2, 1], [7, 1, 1]	100	[1, 2, 6]	95.8984375
10	[3, 2, 5], [4, 1, 5], [4, 2, 4], [5, 2, 3], [6, 1, 3], [6, 2, 2], [7, 1, 2], [8, 1, 1]	100	[1, 1, 8]	95.41015625
11	[4, 5, 2], [5, 2, 4], [5, 4, 2], [6, 1, 4], [6, 2, 3], [6, 3, 2], [7, 1, 3], [7, 3, 1], [8, 2, 1]	100	[1, 1, 9]	95.703125
12	[3, 3, 6], [3, 5, 4], [4, 1, 7], [4, 2, 6], [5, 3, 4], [5, 4, 3], [5, 6, 1], [6, 2, 4], [7, 2, 3], [7, 3, 2], [7, 4, 1], [8, 3, 1], [10, 1, 1]	100	[1, 6, 5], [1, 8, 3]	95.80078125
13	[3, 1, 9], [4, 7, 2], [5, 1, 7], [5, 4, 4], [5, 7, 1], [6, 1, 6], [6, 4, 3], [6, 5, 2], [7, 2, 4], [7, 3, 3], [8, 3, 2], [8, 4, 1], [9, 3, 1], [10, 1, 2], [11, 1, 1]	100	[1, 2, 10]	95.5078125
14	[3, 1, 10], [3, 4, 7], [4, 2, 8], [4, 6, 4], [5, 3, 6], [5, 4, 5], [5, 5, 4], [5, 7, 2], [6, 1, 7], [6, 3, 5], [6, 5, 3], [7, 1, 6], [7, 5, 2], [8, 4, 2], [8, 5, 1], [9, 1, 4], [9, 3, 2], [9, 4, 1], [10, 1, 3], [10, 2, 2], [10, 3, 1], [11, 1, 2]	100	[1, 7, 6]	95.8984375
15	[3, 1, 11], [3, 2, 10], [4, 1, 10], [4, 2, 9], [4, 6, 5], [4, 8, 3], [4, 9, 2], [5, 2, 8], [5, 3, 7], [5, 4, 6], [5, 5, 5], [5, 6, 4], [6, 1, 8], [6, 3, 6], [6, 4, 5], [6, 6, 3], [6, 7, 2], [7, 2, 6], [7, 3, 5], [7, 5, 3], [7, 6, 2], [7, 7, 1], [8, 1, 6], [8, 2, 5], [8, 4, 3], [8, 5, 2], [9, 2, 4], [9, 3, 3], [9, 4, 2], [9, 5, 1], [10, 1, 4], [10, 2, 3], [10, 3, 2]	100	[1, 2, 12]	95.60546875
16	[3, 1, 12], [3, 10, 3], [4, 2, 10], [4, 4, 8], [4, 6, 6], [4, 8, 4], [4, 10, 2], [4, 11, 1], [5, 1, 10], [5, 3, 8], [6, 1, 9], [6, 2, 8], [6, 6, 4], [6, 7, 3], [6, 8, 2], [7, 3, 6], [7, 4, 5], [7, 5, 4], [7, 7, 2], [7, 8, 1], [8, 3, 5], [8, 4, 4], [8, 6, 2], [9, 1, 6], [9, 2, 5], [9, 3, 4], [10, 1, 5], [10, 3, 3], [10, 5, 1], [11, 3, 2], [11, 4, 1], [12, 1, 3]	100	[1, 2, 13]	95.21484375
17	[3, 6, 8], [4, 2, 11], [4, 6, 7], [4, 9, 4], [4, 11, 2], [5, 1, 11], [5, 4, 8], [5, 8, 4], [5, 9, 3], [5, 11, 1], [6, 2, 9], [6, 7, 4], [6, 8, 3], [6, 9, 2], [7, 1, 9], [7, 7, 3], [7, 8, 2], [8, 1, 8], [8, 3, 6], [8, 6, 3], [9, 1, 7], [9, 2, 6], [9, 4, 4], [9, 5, 3], [10, 1, 6], [10, 2, 5], [10, 5, 2], [10, 6, 1], [11, 1, 5], [11, 2, 4], [11, 3, 3], [11, 4, 2], [11, 5, 1], [12, 1, 4], [12, 2, 3], [12, 4, 1], [13, 2, 2], [13, 3, 1], [14, 1, 2], [14, 2, 1], [15, 1, 1]	100	[1, 2, 14]	95.60546875
18	[3, 1, 14], [3, 2, 13], [3, 8, 7], [3, 10, 5], [3, 12, 3], [3, 13, 2], [4, 4, 10], [4, 5, 9], [4, 6, 8], [4, 10, 4], [4, 11, 3], [4, 12, 2], [5, 3, 10], [5, 4, 9], [5, 5, 8], [5, 7, 6], [5, 8, 5], [6, 2, 10], [6, 4, 8], [6, 8, 4], [7, 3, 8], [7, 5, 6], [7, 8, 3], [7, 9, 2], [7, 10, 1], [8, 2, 8], [8, 4, 6], [8, 5, 5], [8, 7, 3], [9, 1, 8], [9, 4, 5], [9, 6, 3], [9, 7, 2], [9, 8, 1], [10, 2, 6], [10, 3, 5], [10, 4, 4], [10, 5, 3], [10, 7, 1], [11, 1, 6], [11, 2, 5], [11, 3, 4], [11, 5, 2], [12, 1, 5], [12, 3, 3], [13, 1, 4], [13, 2, 3], [13, 4, 1], [14, 1, 3], [14, 2, 2], [15, 1, 2], [15, 2, 1], [16, 1, 1]	100	[1, 7, 10], [1, 8, 9]	95.60546875
19	[3, 5, 11], [3, 6, 10], [3, 9, 7], [3, 10, 6], [3, 11, 5], [3, 12, 4], [4, 1, 14], [4, 4, 11], [4, 6, 9], [4, 8, 7], [4, 11, 4], [4, 12, 3], [4, 13, 2], [5, 5, 9], [5, 9, 5], [5, 12, 2], [5, 13, 1], [6, 1, 12], [6, 2, 11], [6, 3, 10], [6, 4, 9], [6, 8, 5], [6, 10, 3], [7, 1, 11], [7, 3, 9], [7, 4, 8], [7, 10, 2], [8, 3, 8], [8, 9, 2], [9, 1, 9], [9, 3, 7], [9, 5, 5], [9, 6, 4], [9, 8, 2], [9, 9, 1], [10, 1, 8], [10, 2, 7], [10, 4, 5], [10, 5, 4], [10, 6, 3], [11, 2, 6], [11, 3, 5], [11, 4, 4], [11, 5, 3], [11, 6, 2], [12, 1, 6], [12, 2, 5], [12, 3, 4], [12, 4, 3], [12, 6, 1], [13, 1, 5], [13, 3, 3], [13, 4, 2], [13, 5, 1], [14, 2, 3], [14, 3, 2], [14, 4, 1], [15, 1, 3], [15, 2, 2], [15, 3, 1], [16, 1, 2]	100	[1, 2, 16]	95.41015625
20	[3, 15, 2], [4, 1, 15], [4, 5, 11], [4, 7, 9], [4, 8, 8], [4, 12, 4], [4, 13, 3], [4, 15, 1], [5, 3, 12], [5, 5, 10], [5, 6, 9], [5, 7, 8], [5, 9, 6], [5, 10, 5], [5, 12, 3], [5, 13, 2], [6, 1, 13], [6, 5, 9], [6, 8, 6], [6, 9, 5], [6, 10, 4], [6, 11, 3], [7, 2, 11], [7, 3, 10], [7, 4, 9], [7, 5, 8], [7, 8, 5], [7, 9, 4], [7, 10, 3], [7, 11, 2], [8, 2, 10], [8, 5, 7], [8, 8, 4], [8, 11, 1], [9, 1, 10], [9, 5, 6], [9, 6, 5], [9, 7, 4], [9, 10, 1], [10, 1, 9], [10, 3, 7], [10, 6, 4], [10, 7, 3], [10, 8, 2], [10, 9, 1], [11, 1, 8], [11, 3, 6], [12, 2, 6], [12, 3, 5], [12, 6, 2], [12, 7, 1], [13, 1, 6], [13, 2, 5], [13, 3, 4], [13, 6, 1], [14, 1, 5], [14, 3, 3], [14, 4, 2], [14, 5, 1], [15, 1, 4], [15, 2, 3], [15, 3, 2], [16, 3, 1], [17, 2, 1]	100	[1, 6, 13]	95.3125

TABLE A.3: The best and worst node configurations for a three layered neural network and their accuracies for a given number of total nodes.

Total number of nodes	Best configuration	Best accuracy	Worst configuration	Worst accuracy
21	[3, 1, 17], [3, 5, 13], [3, 6, 12], [3, 7, 11], [3, 11, 7], [3, 12, 6], [3, 13, 5], [3, 15, 3], [4, 2, 15], [4, 5, 12], [4, 8, 9], [4, 11, 6], [4, 13, 4], [4, 14, 3], [4, 16, 1], [5, 2, 14], [5, 5, 11], [5, 6, 10], [5, 8, 8], [5, 10, 6], [5, 11, 5], [5, 12, 4], [5, 13, 3], [5, 15, 1], [6, 1, 14], [6, 2, 13], [6, 5, 10], [6, 7, 8], [6, 8, 7], [6, 9, 6], [6, 13, 2], [7, 1, 13], [7, 2, 12], [7, 4, 10], [7, 5, 9], [7, 6, 8], [7, 7, 7], [7, 10, 4], [7, 12, 2], [7, 13, 1], [8, 5, 8], [8, 6, 7], [8, 7, 6], [8, 8, 5], [8, 9, 4], [8, 10, 3], [9, 2, 10], [9, 3, 9], [9, 4, 8], [9, 6, 6], [9, 9, 3], [9, 10, 2], [10, 1, 10], [10, 3, 8], [10, 4, 7], [10, 5, 6], [10, 7, 4], [10, 8, 3], [10, 9, 2], [11, 1, 9], [11, 3, 7], [11, 4, 6], [11, 5, 5], [11, 7, 3], [11, 8, 2], [12, 1, 8], [12, 2, 7], [12, 3, 6], [12, 5, 4], [12, 6, 3], [13, 2, 6], [13, 4, 4], [13, 6, 2], [13, 7, 1], [14, 2, 5], [14, 3, 4], [14, 4, 3], [14, 6, 1], [15, 1, 5], [15, 2, 4], [15, 3, 3], [15, 5, 1], [16, 1, 4], [16, 3, 2], [17, 1, 3], [17, 2, 2], [17, 3, 1]	100	[1, 3, 17]	95.41015625
22	[3, 1, 18], [3, 3, 16], [3, 7, 12], [3, 13, 6], [3, 15, 4], [3, 17, 2], [4, 2, 16], [4, 5, 13], [4, 6, 12], [4, 9, 9], [5, 4, 13], [5, 5, 12], [5, 6, 11], [5, 8, 9], [5, 9, 8], [5, 10, 7], [5, 11, 6], [5, 14, 3], [6, 1, 15], [6, 2, 14], [6, 3, 13], [6, 4, 12], [6, 10, 6], [6, 13, 3], [6, 14, 2], [6, 15, 1], [7, 1, 14], [7, 2, 13], [7, 4, 11], [7, 6, 9], [7, 12, 3], [8, 4, 10], [8, 5, 9], [8, 7, 7], [8, 9, 5], [8, 11, 3], [8, 12, 2], [9, 2, 11], [9, 4, 9], [9, 10, 3], [9, 12, 1], [10, 1, 11], [10, 2, 10], [10, 3, 9], [10, 4, 8], [10, 5, 7], [10, 7, 5], [10, 9, 3], [10, 10, 2], [10, 11, 1], [11, 1, 10], [11, 2, 9], [11, 3, 8], [11, 4, 7], [11, 5, 6], [11, 6, 5], [11, 7, 4], [11, 8, 3], [11, 9, 2], [12, 1, 9], [12, 2, 8], [12, 3, 7], [12, 4, 6], [12, 5, 5], [12, 7, 3], [12, 9, 1], [13, 2, 7], [13, 3, 6], [13, 4, 5], [13, 5, 4], [13, 7, 2], [13, 8, 1], [14, 1, 7], [14, 2, 6], [15, 1, 6], [15, 6, 1], [16, 1, 5], [16, 2, 4], [16, 4, 2], [16, 5, 1], [17, 2, 3], [17, 4, 1], [18, 1, 3]	100	[1, 2, 19], [1, 6, 15]	95.3125
23	[3, 7, 13], [3, 10, 10], [3, 11, 9], [3, 12, 8], [3, 14, 6], [3, 16, 4], [4, 2, 17], [4, 4, 15], [4, 10, 9], [4, 12, 7], [4, 13, 6], [5, 2, 16], [5, 5, 13], [5, 6, 12], [5, 7, 11], [5, 11, 7], [5, 12, 6], [5, 14, 4], [5, 15, 3], [5, 16, 2], [6, 1, 16], [6, 2, 15], [6, 3, 14], [6, 7, 10], [6, 8, 9], [6, 10, 7], [6, 11, 6], [6, 13, 4], [6, 14, 3], [7, 1, 15], [7, 2, 14], [7, 4, 12], [7, 5, 11], [7, 7, 9], [7, 8, 8], [7, 9, 7], [7, 10, 6], [7, 11, 5], [7, 12, 4], [7, 13, 3], [7, 15, 1], [8, 1, 14], [8, 3, 12], [8, 4, 11], [8, 5, 10], [8, 6, 9], [8, 7, 8], [8, 8, 7], [8, 11, 4], [9, 2, 12], [9, 3, 11], [9, 4, 10], [9, 5, 9], [9, 9, 5], [9, 10, 4], [9, 11, 3], [9, 12, 2], [9, 13, 1], [10, 2, 11], [10, 5, 8], [10, 6, 7], [10, 9, 4], [10, 12, 1], [11, 1, 11], [11, 3, 9], [11, 4, 8], [11, 5, 7], [11, 6, 6], [11, 11, 1], [12, 1, 10], [12, 4, 7], [12, 5, 6], [12, 6, 5], [12, 7, 4], [12, 8, 3], [12, 9, 2], [12, 10, 1], [13, 1, 9], [13, 2, 8], [13, 3, 7], [13, 4, 6], [13, 5, 5], [13, 6, 4], [13, 7, 3], [13, 8, 2], [13, 9, 1], [14, 1, 8], [14, 3, 6], [14, 4, 5], [14, 7, 2], [14, 8, 1], [15, 1, 7], [15, 2, 6], [15, 3, 5], [15, 5, 3], [15, 6, 2], [15, 7, 1], [16, 1, 6], [16, 2, 5], [16, 4, 3], [16, 5, 2], [16, 6, 1], [17, 1, 5], [17, 2, 4], [17, 3, 3], [17, 4, 2], [17, 5, 1], [18, 1, 4], [18, 2, 3], [18, 3, 2], [19, 1, 3], [19, 2, 2], [19, 3, 1], [20, 1, 2], [20, 2, 1]	100	[1, 5, 17]	95.60546875
24	[3, 2, 19], [3, 7, 14], [3, 8, 13], [3, 12, 9], [3, 14, 7], [3, 15, 6], [3, 19, 2], [4, 1, 19], [4, 5, 15], [4, 8, 12], [4, 9, 11], [4, 10, 10], [4, 11, 9], [4, 12, 8], [4, 14, 6], [4, 16, 4], [4, 18, 2], [5, 2, 17], [5, 4, 15], [5, 7, 12], [5, 9, 10], [5, 10, 9], [5, 13, 6], [5, 14, 5], [5, 17, 2], [6, 1, 17], [6, 2, 16], [6, 3, 15], [6, 6, 12], [6, 7, 11], [6, 8, 10], [6, 9, 9], [6, 11, 7], [6, 13, 5], [7, 1, 16], [7, 2, 15], [7, 5, 12], [7, 9, 8], [7, 10, 7], [7, 13, 4], [7, 16, 1], [8, 2, 14], [8, 4, 12], [8, 5, 11], [8, 6, 10], [8, 8, 8], [8, 9, 7], [8, 11, 5], [8, 13, 3], [8, 15, 1], [9, 1, 14], [9, 2, 13], [9, 6, 9], [9, 7, 8], [9, 8, 7], [9, 10, 5], [9, 11, 4], [9, 12, 3], [9, 14, 1], [10, 1, 13], [10, 2, 12], [10, 4, 10], [10, 5, 9], [10, 7, 7], [10, 9, 5], [10, 10, 4], [10, 11, 3], [10, 12, 2], [11, 4, 9], [11, 6, 7], [11, 7, 6], [11, 9, 4], [11, 11, 2], [12, 2, 10], [12, 6, 6], [12, 8, 4], [12, 9, 3], [12, 10, 2], [12, 11, 1], [13, 1, 10], [13, 2, 9], [13, 3, 8], [13, 5, 6], [13, 7, 4], [14, 2, 8], [14, 3, 7], [14, 4, 6], [14, 5, 5], [14, 6, 4], [15, 1, 8], [15, 3, 6], [15, 5, 4], [15, 6, 3], [16, 1, 7], [16, 3, 5], [16, 4, 4], [16, 5, 3], [16, 7, 1], [17, 1, 6], [17, 2, 5], [17, 4, 3], [17, 5, 2], [17, 6, 1], [18, 1, 5], [18, 3, 3], [18, 5, 1], [19, 1, 4], [19, 2, 3], [19, 3, 2], [20, 1, 3], [20, 2, 2], [20, 3, 1], [21, 1, 2], [21, 2, 1]	100	[1, 3, 20], [1, 6, 17]	95.8984375
25	[3, 5, 17], [3, 13, 9], [3, 14, 8], [3, 16, 6], [3, 18, 4], [3, 20, 2], [3, 21, 1], [4, 1, 20], [4, 2, 19], [4, 11, 10], [4, 14, 7], [4, 16, 5], [5, 1, 19], [5, 5, 15], [5, 6, 14], [5, 8, 12], [5, 9, 11], [5, 10, 10], [5, 11, 9], [5, 14, 6], [5, 16, 4], [5, 17, 3], [6, 1, 18], [6, 2, 17], [6, 4, 15], [6, 7, 12], [6, 8, 11], [6, 9, 10], [6, 10, 9], [6, 12, 7], [6, 13, 6], [6, 14, 5], [6, 17, 2], [6, 18, 1], [7, 2, 16], [7, 3, 15], [7, 4, 14], [7, 6, 12], [7, 7, 11], [7, 9, 9], [7, 11, 7], [7, 13, 5], [7, 14, 4], [7, 16, 2], [7, 17, 1], [8, 3, 14], [8, 5, 12], [8, 6, 11], [8, 7, 10], [8, 8, 9], [8, 11, 6], [8, 13, 4], [8, 14, 3], [8, 15, 2], [9, 1, 15], [9, 5, 11], [9, 6, 10], [9, 7, 9], [9, 9, 7], [9, 11, 5], [9, 13, 3], [9, 15, 1], [10, 1, 14], [10, 2, 13], [10, 3, 12], [10, 4, 11], [10, 7, 8], [10, 9, 6], [10, 10, 5], [10, 11, 4], [10, 12, 3], [10, 14, 1], [11, 1, 13], [11, 2, 12], [11, 4, 10], [11, 5, 9], [11, 6, 8], [11, 8, 6], [11, 9, 5], [11, 10, 4], [11, 12, 2], [11, 13, 1], [12, 2, 11], [12, 3, 10], [12, 4, 9], [12, 5, 8], [12, 8, 5], [12, 9, 4], [12, 10, 3], [12, 11, 2], [13, 1, 11], [13, 2, 10], [13, 4, 8], [13, 8, 4], [13, 9, 3], [13, 10, 2], [14, 1, 10], [14, 4, 7], [14, 5, 6], [14, 7, 4], [14, 9, 2], [14, 10, 1], [15, 2, 8], [15, 3, 7], [15, 4, 6], [15, 5, 5], [15, 6, 4], [15, 8, 2], [16, 1, 8], [16, 5, 4], [16, 7, 2], [17, 1, 7], [17, 3, 5], [17, 4, 4], [17, 5, 3], [17, 6, 2], [17, 7, 1], [18, 1, 6], [18, 2, 5], [18, 5, 2], [18, 6, 1], [19, 1, 5], [19, 2, 4], [20, 1, 4], [20, 2, 3], [20, 3, 2], [21, 2, 2], [22, 2, 1], [23, 1, 1]	100	[1, 3, 21]	95.3125

TABLE A.4: The best and worst node configurations for a three layered neural network and their accuracies for a given number of total nodes.

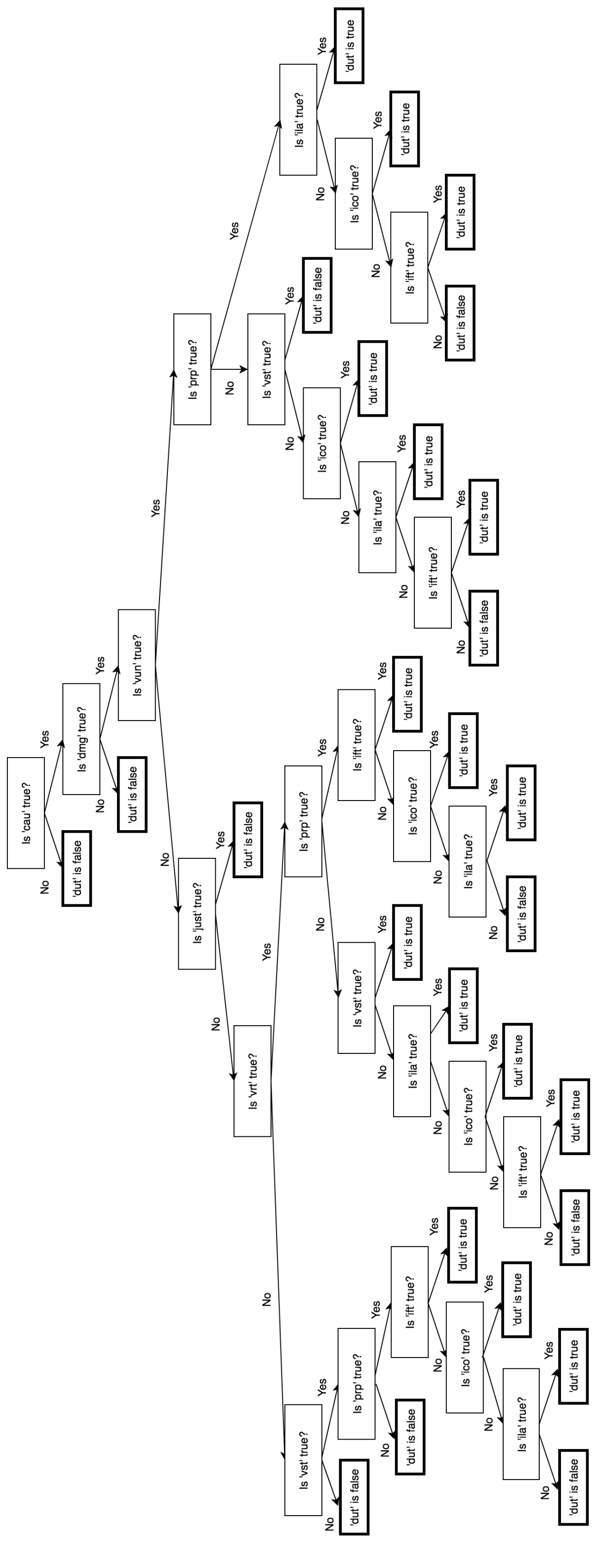
Total number of nodes	Best configuration	Best accuracy	Worst configuration	Worst accuracy
26	[3, 1, 22], [3, 2, 21], [3, 8, 15], [3, 12, 11], [3, 14, 9], [3, 15, 8], [3, 17, 6], [4, 1, 21], [4, 5, 17], [4, 6, 16], [4, 7, 15], [4, 10, 12], [4, 11, 11], [4, 18, 4], [4, 19, 3], [4, 20, 2], [5, 4, 17], [5, 7, 14], [5, 9, 12], [5, 11, 10], [5, 13, 8], [5, 14, 7], [5, 15, 6], [5, 19, 2], [6, 1, 19], [6, 2, 18], [6, 3, 17], [6, 4, 16], [6, 5, 15], [6, 7, 13], [6, 8, 12], [6, 10, 10], [6, 11, 9], [6, 12, 8], [6, 13, 7], [6, 16, 4], [6, 17, 3], [6, 19, 1], [7, 1, 18], [7, 2, 17], [7, 3, 16], [7, 4, 15], [7, 7, 12], [7, 9, 10], [7, 10, 9], [7, 11, 8], [7, 13, 6], [7, 14, 5], [7, 15, 4], [7, 17, 2], [7, 18, 1], [8, 1, 17], [8, 3, 15], [8, 4, 14], [8, 5, 13], [8, 6, 12], [8, 7, 11], [8, 9, 9], [8, 10, 8], [8, 11, 7], [8, 12, 6], [8, 13, 5], [8, 14, 4], [8, 15, 3], [8, 17, 1], [9, 3, 14], [9, 5, 12], [9, 7, 10], [9, 8, 9], [9, 9, 8], [9, 10, 7], [9, 11, 6], [9, 12, 5], [9, 13, 4], [9, 14, 3], [9, 15, 2], [9, 16, 1], [10, 2, 14], [10, 3, 13], [10, 4, 12], [10, 6, 10], [10, 7, 9], [10, 9, 7], [10, 11, 5], [10, 12, 4], [10, 13, 3], [10, 14, 2], [10, 15, 1], [11, 2, 13], [11, 5, 10], [11, 7, 8], [11, 9, 6], [11, 10, 5], [12, 2, 12], [12, 5, 9], [12, 6, 8], [12, 7, 7], [12, 9, 5], [12, 10, 4], [12, 11, 3], [12, 13, 1], [13, 1, 12], [13, 2, 11], [13, 3, 10], [13, 4, 9], [13, 5, 8], [13, 6, 7], [13, 7, 6], [13, 8, 5], [13, 10, 3], [13, 12, 1], [14, 3, 9], [14, 4, 8], [14, 5, 7], [14, 6, 6], [14, 8, 4], [14, 9, 3], [14, 10, 2], [15, 2, 9], [15, 3, 8], [15, 4, 7], [15, 5, 6], [15, 6, 5], [16, 1, 9], [16, 2, 8], [16, 5, 5], [16, 6, 4], [16, 7, 3], [16, 8, 2], [17, 1, 8], [17, 2, 7], [17, 4, 5], [17, 6, 3], [17, 7, 2], [17, 8, 1], [18, 2, 6], [18, 3, 5], [18, 4, 4], [18, 5, 3], [18, 6, 2], [19, 1, 6], [19, 2, 5], [19, 4, 3], [20, 1, 5], [20, 3, 3], [20, 5, 1], [21, 1, 4], [21, 3, 2], [23, 2, 1], [24, 1, 1]	100	[1, 9, 16]	95.3125
27	[3, 1, 23], [3, 5, 19], [3, 9, 15], [3, 10, 14], [3, 15, 9], [3, 19, 5], [3, 21, 3], [4, 1, 22], [4, 2, 21], [4, 3, 20], [4, 4, 19], [4, 5, 18], [4, 6, 17], [4, 7, 16], [4, 8, 15], [4, 9, 14], [4, 11, 12], [4, 14, 9], [4, 18, 5], [4, 21, 2], [5, 4, 18], [5, 5, 17], [5, 13, 9], [5, 14, 8], [5, 15, 7], [5, 16, 6], [5, 17, 5], [5, 18, 4], [5, 20, 2], [6, 1, 20], [6, 2, 19], [6, 3, 18], [6, 4, 17], [6, 5, 16], [6, 7, 14], [6, 8, 13], [6, 9, 12], [6, 12, 9], [6, 13, 8], [6, 15, 6], [6, 17, 4], [6, 18, 3], [6, 19, 2], [7, 1, 19], [7, 2, 18], [7, 3, 17], [7, 4, 16], [7, 5, 15], [7, 6, 14], [7, 7, 13], [7, 9, 11], [7, 10, 10], [7, 11, 9], [7, 13, 7], [7, 14, 6], [7, 15, 5], [7, 17, 3], [7, 18, 2], [7, 19, 1], [8, 1, 18], [8, 3, 16], [8, 4, 15], [8, 5, 14], [8, 6, 13], [8, 8, 11], [8, 9, 10], [8, 11, 8], [8, 17, 2], [8, 18, 1], [9, 2, 16], [9, 3, 15], [9, 4, 14], [9, 5, 13], [9, 6, 12], [9, 7, 11], [9, 10, 8], [9, 11, 7], [9, 12, 6], [9, 13, 5], [9, 14, 4], [9, 15, 3], [9, 16, 2], [9, 17, 1], [10, 1, 16], [10, 2, 15], [10, 4, 13], [10, 5, 12], [10, 6, 11], [10, 7, 10], [10, 8, 9], [10, 9, 8], [10, 10, 7], [10, 12, 5], [10, 14, 3], [10, 16, 1], [11, 1, 15], [11, 3, 13], [11, 4, 12], [11, 5, 11], [11, 6, 10], [11, 8, 8], [11, 11, 5], [11, 13, 3], [11, 14, 2], [11, 15, 1], [12, 1, 14], [12, 2, 13], [12, 3, 12], [12, 6, 9], [12, 7, 8], [12, 8, 7], [12, 9, 6], [12, 10, 5], [12, 11, 4], [12, 12, 3], [12, 13, 2], [13, 1, 13], [13, 3, 11], [13, 4, 10], [13, 6, 8], [13, 7, 7], [13, 8, 6], [13, 9, 5], [13, 10, 4], [14, 1, 12], [14, 3, 10], [14, 4, 9], [14, 5, 8], [14, 6, 7], [14, 8, 5], [14, 9, 4], [14, 10, 3], [14, 12, 1], [15, 1, 11], [15, 5, 7], [15, 11, 1], [16, 1, 10], [16, 4, 7], [16, 5, 6], [16, 6, 5], [16, 7, 4], [16, 10, 1], [17, 1, 9], [17, 3, 7], [17, 4, 6], [17, 5, 5], [17, 6, 4], [17, 7, 3], [17, 8, 2], [17, 9, 1], [18, 1, 8], [18, 3, 6], [18, 4, 5], [18, 5, 4], [18, 6, 3], [18, 7, 2], [18, 8, 1], [19, 2, 6], [19, 3, 5], [19, 5, 3], [19, 7, 1], [20, 1, 6], [20, 5, 2], [20, 6, 1], [21, 3, 3], [21, 4, 2], [21, 5, 1], [22, 1, 4], [22, 3, 2], [22, 4, 1], [23, 3, 1], [24, 1, 2], [24, 2, 1], [25, 1, 1]	100	[1, 8, 18], [1, 9, 17], [1, 13, 13]	95.703125
28	[3, 1, 24], [3, 2, 23], [3, 6, 19], [3, 7, 18], [3, 14, 11], [3, 17, 8], [3, 18, 7], [3, 19, 6], [3, 23, 2], [4, 1, 23], [4, 4, 20], [4, 6, 18], [4, 8, 16], [4, 9, 15], [4, 10, 14], [4, 11, 13], [4, 12, 12], [4, 14, 10], [4, 20, 4], [4, 22, 2], [4, 23, 1], [5, 2, 21], [5, 5, 18], [5, 6, 17], [5, 8, 15], [5, 11, 12], [5, 12, 11], [5, 13, 10], [5, 14, 9], [5, 16, 7], [5, 22, 1], [6, 1, 21], [6, 2, 20], [6, 5, 17], [6, 6, 16], [6, 7, 15], [6, 8, 14], [6, 9, 13], [6, 10, 12], [6, 12, 10], [6, 14, 8], [6, 15, 7], [6, 16, 6], [6, 19, 3], [7, 1, 20], [7, 2, 19], [7, 4, 17], [7, 5, 16], [7, 6, 15], [7, 7, 14], [7, 9, 12], [7, 10, 11], [7, 12, 9], [7, 13, 8], [7, 14, 7], [7, 15, 6], [7, 19, 2], [7, 20, 1], [8, 5, 15], [8, 8, 12], [8, 10, 10], [8, 12, 8], [8, 14, 6], [8, 16, 4], [9, 1, 18], [9, 2, 17], [9, 3, 16], [9, 4, 15], [9, 7, 12], [9, 8, 11], [9, 9, 10], [9, 10, 9], [9, 12, 7], [9, 13, 6], [9, 14, 5], [9, 16, 3], [9, 17, 2], [9, 18, 1], [10, 1, 17], [10, 7, 11], [10, 8, 10], [10, 9, 9], [10, 10, 8], [10, 11, 7], [10, 13, 5], [10, 14, 4], [10, 15, 3], [10, 17, 1], [11, 1, 16], [11, 2, 15], [11, 3, 14], [11, 4, 13], [11, 5, 12], [11, 7, 10], [11, 9, 8], [11, 10, 7], [11, 11, 6], [11, 12, 5], [11, 15, 2], [11, 16, 1], [12, 1, 15], [12, 3, 13], [12, 4, 12], [12, 5, 11], [12, 6, 10], [12, 7, 9], [12, 8, 8], [12, 9, 7], [12, 10, 6], [12, 13, 3], [12, 14, 2], [12, 15, 1], [13, 1, 14], [13, 3, 12], [13, 7, 8], [13, 12, 3], [13, 14, 1], [14, 1, 13], [14, 4, 10], [14, 5, 9], [14, 6, 8], [14, 8, 6], [14, 9, 5], [14, 10, 4], [14, 12, 2], [14, 13, 1], [15, 4, 9], [15, 5, 8], [15, 7, 6], [15, 8, 5], [15, 9, 4], [15, 10, 3], [15, 12, 1], [16, 2, 10], [16, 3, 9], [16, 4, 8], [16, 5, 7], [16, 7, 5], [16, 8, 4], [16, 9, 3], [16, 10, 2], [16, 11, 1], [17, 3, 8], [17, 4, 7], [17, 5, 6], [17, 6, 5], [17, 7, 4], [17, 9, 2], [17, 10, 1], [18, 1, 9], [18, 2, 8], [18, 3, 7], [18, 7, 3], [18, 9, 1], [19, 1, 8], [19, 2, 7], [19, 3, 6], [19, 5, 4], [19, 6, 3], [19, 7, 2], [19, 8, 1], [20, 2, 6], [20, 4, 4], [20, 5, 3], [20, 7, 1], [21, 2, 5], [21, 3, 4], [21, 4, 3], [21, 5, 2], [21, 6, 1], [22, 2, 4], [22, 3, 3], [22, 4, 2], [22, 5, 1], [23, 1, 4], [23, 2, 3], [24, 1, 3], [24, 2, 2], [24, 3, 1], [25, 1, 2], [25, 2, 1], [26, 1, 1]	100	[1, 8, 19]	95.3125

TABLE A.5: The best and worst node configurations for a three layered neural network and their accuracies for a given number of total nodes.

Total number of nodes	Best configuration	Best accuracy	Worst configuration	Worst accuracy
29	[3, 1, 25], [3, 3, 23], [3, 8, 18], [3, 9, 17], [3, 12, 14], [3, 22, 4], [3, 24, 2], [4, 3, 22], [4, 4, 21], [4, 6, 19], [4, 8, 17], [4, 9, 16], [4, 11, 14], [4, 12, 13], [4, 18, 7], [4, 20, 5], [4, 22, 3], [5, 1, 23], [5, 2, 22], [5, 8, 16], [5, 9, 15], [5, 12, 12], [5, 14, 10], [5, 15, 9], [5, 16, 8], [5, 17, 7], [5, 18, 6], [5, 20, 4], [5, 22, 2], [6, 1, 22], [6, 2, 21], [6, 3, 20], [6, 5, 18], [6, 7, 16], [6, 9, 14], [6, 11, 12], [6, 13, 10], [6, 15, 8], [6, 18, 5], [6, 19, 4], [6, 20, 3], [6, 22, 1], [7, 1, 21], [7, 4, 18], [7, 8, 14], [7, 9, 13], [7, 10, 12], [7, 11, 11], [7, 15, 7], [7, 17, 5], [7, 18, 4], [7, 19, 3], [7, 21, 1], [8, 3, 18], [8, 5, 16], [8, 7, 14], [8, 8, 13], [8, 11, 10], [8, 16, 5], [8, 18, 3], [9, 2, 18], [9, 3, 17], [9, 4, 16], [9, 6, 14], [9, 7, 13], [9, 8, 12], [9, 9, 11], [9, 11, 9], [9, 12, 8], [9, 13, 7], [9, 16, 4], [9, 17, 3], [9, 18, 2], [10, 1, 18], [10, 3, 16], [10, 4, 15], [10, 5, 14], [10, 6, 13], [10, 9, 10], [10, 12, 7], [10, 13, 6], [10, 14, 5], [10, 15, 4], [10, 16, 3], [10, 17, 2], [11, 1, 17], [11, 2, 16], [11, 3, 15], [11, 4, 14], [11, 5, 13], [11, 6, 12], [11, 8, 10], [11, 9, 9], [11, 11, 7], [11, 12, 6], [11, 13, 5], [11, 17, 1], [12, 1, 16], [12, 2, 15], [12, 3, 14], [12, 4, 13], [12, 5, 12], [12, 6, 11], [12, 7, 10], [12, 8, 9], [12, 9, 8], [12, 10, 7], [12, 11, 6], [12, 12, 5], [12, 13, 4], [12, 15, 2], [12, 16, 1], [13, 1, 15], [13, 2, 14], [13, 4, 12], [13, 5, 11], [13, 9, 7], [13, 13, 3], [14, 1, 14], [14, 2, 13], [14, 3, 12], [14, 4, 11], [14, 5, 10], [14, 6, 9], [14, 7, 8], [14, 8, 7], [14, 10, 5], [14, 11, 4], [14, 13, 2], [14, 14, 1], [15, 1, 13], [15, 2, 12], [15, 3, 11], [15, 6, 8], [15, 8, 6], [15, 9, 5], [15, 11, 3], [16, 1, 12], [16, 2, 11], [16, 4, 9], [16, 6, 7], [16, 8, 5], [16, 11, 2], [17, 1, 11], [17, 3, 9], [17, 4, 8], [17, 5, 7], [17, 6, 6], [17, 8, 4], [17, 9, 3], [17, 10, 2], [17, 11, 1], [18, 1, 10], [18, 3, 8], [18, 4, 7], [18, 5, 6], [18, 6, 5], [18, 7, 4], [18, 9, 2], [18, 10, 1], [19, 1, 9], [19, 3, 7], [19, 6, 4], [19, 7, 3], [19, 9, 1], [20, 1, 8], [20, 2, 7], [20, 3, 6], [20, 6, 3], [20, 7, 2], [21, 1, 7], [21, 2, 6], [21, 3, 5], [21, 4, 4], [21, 6, 2], [21, 7, 1], [22, 2, 5], [22, 3, 4], [22, 5, 2], [22, 6, 1], [23, 1, 5], [23, 2, 4], [23, 3, 3], [23, 4, 2], [23, 5, 1], [24, 1, 4], [24, 3, 2], [25, 1, 3], [25, 2, 2], [25, 3, 1], [26, 1, 2], [26, 2, 1], [27, 1, 1]	100	[1, 6, 22]	95.3125
30	[3, 6, 21], [3, 17, 10], [3, 21, 6], [3, 22, 5], [3, 25, 2], [4, 4, 22], [4, 5, 21], [4, 7, 19], [4, 8, 18], [4, 9, 17], [4, 12, 14], [4, 13, 13], [4, 14, 12], [4, 17, 9], [4, 20, 6], [4, 22, 4], [4, 24, 2], [5, 3, 22], [5, 5, 20], [5, 6, 19], [5, 9, 16], [5, 11, 14], [5, 12, 13], [5, 14, 11], [5, 15, 10], [5, 17, 8], [5, 18, 7], [5, 19, 6], [5, 24, 1], [6, 1, 23], [6, 2, 22], [6, 4, 20], [6, 5, 19], [6, 8, 16], [6, 11, 13], [6, 14, 10], [6, 16, 8], [6, 21, 3], [6, 22, 2], [7, 1, 22], [7, 2, 21], [7, 3, 20], [7, 4, 19], [7, 5, 18], [7, 6, 17], [7, 10, 13], [7, 11, 12], [7, 12, 11], [7, 15, 8], [7, 18, 5], [7, 19, 4], [7, 21, 2], [7, 22, 1], [8, 2, 20], [8, 3, 19], [8, 4, 18], [8, 7, 15], [8, 8, 14], [8, 9, 13], [8, 13, 9], [8, 18, 4], [8, 19, 3], [8, 21, 1], [9, 1, 20], [9, 2, 19], [9, 5, 16], [9, 8, 13], [9, 9, 12], [9, 11, 10], [9, 12, 9], [9, 13, 8], [9, 14, 7], [9, 19, 2], [9, 20, 1], [10, 1, 19], [10, 4, 16], [10, 7, 13], [10, 8, 12], [10, 9, 11], [10, 11, 9], [10, 12, 8], [10, 13, 7], [10, 15, 5], [10, 16, 4], [10, 17, 3], [11, 2, 17], [11, 3, 16], [11, 5, 14], [11, 6, 13], [11, 7, 12], [11, 8, 11], [11, 9, 10], [11, 12, 7], [11, 14, 5], [11, 16, 3], [11, 18, 1], [12, 1, 17], [12, 2, 16], [12, 4, 14], [12, 5, 13], [12, 6, 12], [12, 7, 11], [12, 9, 9], [12, 10, 8], [12, 11, 7], [12, 13, 5], [12, 14, 4], [12, 15, 3], [12, 17, 1], [13, 10, 7], [13, 11, 6], [13, 12, 5], [13, 13, 4], [13, 14, 3], [13, 15, 2], [13, 16, 1], [14, 1, 15], [14, 4, 12], [14, 5, 11], [14, 8, 8], [14, 9, 7], [14, 10, 6], [14, 11, 5], [14, 12, 4], [14, 13, 3], [14, 14, 2], [14, 15, 1], [15, 2, 13], [15, 3, 12], [15, 5, 10], [15, 7, 8], [15, 8, 7], [15, 14, 1], [16, 1, 13], [16, 2, 12], [16, 3, 11], [16, 4, 10], [16, 6, 8], [16, 7, 7], [16, 8, 6], [16, 10, 4], [16, 11, 3], [16, 13, 1], [17, 1, 12], [17, 2, 11], [17, 3, 10], [17, 4, 9], [17, 5, 8], [17, 6, 7], [17, 7, 6], [17, 11, 2], [18, 2, 10], [18, 3, 9], [18, 4, 8], [18, 5, 7], [18, 7, 5], [18, 9, 3], [18, 11, 1], [19, 1, 10], [19, 3, 8], [19, 4, 7], [19, 7, 4], [19, 8, 3], [19, 9, 2], [19, 10, 1], [20, 1, 9], [20, 2, 8], [20, 3, 7], [20, 4, 6], [20, 5, 5], [20, 6, 4], [20, 7, 3], [20, 8, 2], [20, 9, 1], [21, 1, 8], [21, 2, 7], [21, 4, 5], [21, 7, 2], [22, 2, 6], [22, 4, 4], [22, 5, 3], [22, 6, 2], [22, 7, 1], [23, 3, 4], [23, 6, 1], [24, 1, 5], [24, 4, 2], [24, 5, 1], [25, 3, 2], [25, 4, 1], [26, 1, 3], [26, 3, 1], [27, 1, 2], [27, 2, 1], [28, 1, 1]	100	[1, 5, 24]	95.5078125

Appendix B

Tort Law Decision Tree



Bibliography

Agrawal, Rakesh, Tomasz Imieliński, and Arun Swami (1993). "Mining Association Rules Between Sets of Items in Large Databases". In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. SIGMOD '93. Washington, D.C., USA: ACM, New York, pp. 207–216. ISBN: 0-89791-592-5.

Agrawal, Rakesh, Ramakrishnan Srikant, et al. (1994). "Fast algorithms for mining association rules". In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215, pp. 487–499.

Ayodele, Taiwo Oladipupo (2010). "Types of machine learning algorithms". In: *New advances in machine learning*. InTech.

Bench-Capon, Trevor (1993). "Neural Networks and Open Texture". In: *Proceedings of the 4th International Conference on Artificial Intelligence and Law*. ICAIL '93. Amsterdam, The Netherlands: ACM, New York, pp. 292–297. ISBN: 0-89791-606-9.

Breiman, Leo (1996). "Bagging predictors". In: *Machine learning* 24.2, pp. 123–140.

Breiman, Leo et al. (1984). *Classification and regression trees*. CRC press, Boca Raton.

Brin, Sergey et al. (1997). "Dynamic itemset counting and implication rules for market basket data". In: *ACM SIGMOD Record*. Vol. 26. 2. ACM, New York, pp. 255–264.

Carbonell, Jaime G., Ryszard S. Michalski, and Tom M. Mitchell (1983). "An Overview of Machine Learning". In: *Machine Learning: An Artificial Intelligence Approach*. Ed. by Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell. Springer, Berlin Heidelberg, pp. 3–23. ISBN: 978-3-662-12405-5.

Edwards, Lilian and Michael Veale (2017). "Slave to the Algorithm? Why a Right to Explanationn is Probably Not the Remedy You are Looking for". In:

Gunning, David (2017). *Explainable artificial intelligence (XAI)*. URL: `https://www.darpa.mil/attachments/XAIProgramUpdate.pdf` (visited on 06/15/2018).

Hailesilassie, Tameru (2016). "Rule Extraction Algorithm for Deep Neural Networks: A Review". In: *CoRR* abs/1610.05267.

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Ho, Tin Kam (1995). "Random decision forests". In: *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*. Vol. 1. IEEE, Piscataway, pp. 278–282.

— (2002). "A data complexity analysis of comparative advantages of decision forest constructors". In: *Pattern Analysis & Applications* 5.2, pp. 102–112.

Hochreiter, Sepp (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116.

Holzinger, Andreas et al. (2017). "A glass-box interactive machine learning approach for solving NP-hard problems with the human-in-the-loop". In:

Hruschka, Eduardo R and Nelson FF Ebecken (2006). "Extracting rules from multilayer perceptrons in classification problems: A clustering-based approach". In: *Neurocomputing* 70.1, pp. 384–397.

Johnston, Benjamin and Guido Governatori (2003). "Induction of defeasible logic theories in the legal domain". In: *Proceedings of the 9th international conference on Artificial intelligence and law*. ACM, New York, pp. 204–213.

Kuhn, Max and Kjell Johnson (2013). *Applied predictive modeling*. Vol. 810. Springer, New york.

Leighton, R.L. *The Aspirin/MIGRAINES software tools user's manual*. Version V5. The MITRE Corporation. 1991.

Lu, Hongjun, Rudy Setiono, and Huan Liu (1996). "NeuroRule: A Connectionist Approach to Data Mining". In:

Ma, Bing Liu Wynne Hsu Yiming and Bing Liu (1998). "Integrating classification and association rule mining". In: *Proceedings of the fourth international conference on knowledge discovery and data mining*.

Maas, Andrew L, Awni Y Hannun, and Andrew Y Ng (2013). "Rectifier nonlinearities improve neural network acoustic models". In: *Proceedings of the International Conference on Machine Learning*. Vol. 30. 1.

Michaelis, Leonor and Maud Leonora Menten (1913). *Die kinetik der invertinwirkung*. Universitätsbibliothek Johann Christian Senckenberg.

Možina, Martin et al. (2005). "Argument based machine learning applied to law". In: *Artificial Intelligence and Law* 13.1, pp. 53–73.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Raileanu, Laura Elena and Kilian Stoffel (2004). "Theoretical comparison between the gini index and information gain criteria". In: *Annals of Mathematics and Artificial Intelligence* 41.1, pp. 77–93.

Ramachandran, Prajit, Barret Zoph, and Quoc Le (2017). "Searching for activation functions". In:

Rokach, Lior and Oded Maimon (2014). *Data Mining With Decision Trees: Theory and Applications*. 2nd. River Edge, NJ, USA: World Scientific Publishing Co., Inc., River Edge. ISBN: 9789814590075, 981459007X.

Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams, et al. (1988). "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3, p. 1.

Schapire, Robert E and Yoram Singer (1999). "Improved boosting algorithms using confidence-rated predictions". In: *Machine learning* 37.3, pp. 297–336.

Verheij, Bart (2017). "Formalizing Arguments, Rules and Cases". In: *Proceedings of the 16th edition of the International Conference on Articial Intelligence and Law*. ICAIL '17. ACM, New York, pp. 199–208. ISBN: 978-1-4503-4891-1.

Voigt, Paul and Axel von dem Bussche (2017). *The EU General Data Protection Regulation (GDPR): A Practical Guide*. 1st. Springer, New York.

Wachter, Sandra, Brent Mittelstadt, and Luciano Floridi (2017). "Why a Right to Explanation of Automated Decision-Making Does Not Exist in the General Data Protection Regulation". In: *International Data Privacy Law* 7.2, pp. 76–99.

Wardeh, Maya, Trevor Bench-Capon, and Frans Coenen (2009a). "Padua: a protocol for argumentation dialogue using association rules". In: *Artificial Intelligence and Law* 17.3, pp. 183–215.

— (2009b). "PISA—pooling information from several agents: multiplayer argumentation from experience". In: *Research and Development in Intelligent Systems XXV*. Springer, pp. 133–146.

Webb, Geoffrey I (2007). "Discovering significant patterns". In: *Machine Learning* 68.1, pp. 1–33.

Wegener, Ingo (1987). "The complexity of symmetric boolean functions". In: *Computation theory and logic*. Springer, New York, pp. 433–442.

Wilamowski, Bodgan M, David Hunter, and Aleksander Malinowski (2003). "Solving parity-N problems with feedforward neural networks". In: *Neural Networks, 2003. Proceedings of the International Joint Conference on*. Vol. 4. IEEE, Piscataway, pp. 2546–2551.

Yuan, Xiaoyong et al. (2017). "Adversarial Examples: Attacks and Defenses for Deep Learning". In: *CoRR* abs/1712.07107.

Zilke, Jan Ruben, Eneldo Loza Mencía, and Frederik Janssen (2016). "DeepRED–Rule Extraction from Deep Neural Networks". In: *International Conference on Discovery Science*. Springer, New York, pp. 457–473.