Sumaiyah Kola

Optimising Rule Extraction for Deep Neural Networks

Computer Science Tripos – Part II

Clare College

November 10, 2020

Proforma

Candidate Number:	**2420F**
College:	**Clare College**
Project Title:	**Optimising Rule Extraction for Deep Neural Networks**
Examination:	**Computer Science Tripos – Part II June 2020**
Word Count:	**10,533**[1]
Project Originator:	Dr Zohreh Shams
Supervisors:	Dr Zohreh Shams & Botty Dimanov

Original Aims of the Project

The main aim of this project is to implement a decompositional rule extraction algorithm for deep neural networks (DNNs). The ruleset extracted should be comprehensible yet closely mimic the behaviour of the DNN. Rules should be extracted from a variety of classification tasks. The algorithm should include optimisations to ensure it can scale to larger, more complex problems. Further, a method to prune the extracted ruleset using hill-climbing heuristics should also be implemented.

Work Completed

A rule extraction algorithm was implemented that can successfully extract rules from DNNs trained on a wide range of classification tasks. The algorithm includes optimisations to reduce time and memory usage as well as decrease the overall size of the ruleset extracted, increasing the comprehensibility. The rulesets extracted appear to closely mimic the behaviour of the DNNs. A comparative evaluation is performed with two other rule extraction algorithms. Lastly, a successful rule ranking mechanism is implemented that incorporates hill-climbing heuristics to prune the extracted ruleset.

[1]This word count was computed using `texcount diss.tex`

Special Difficulties

Difficulties communicated directly to the department.

Declaration

I, Sumaiyah Kola of Clare College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed S. Y. Kola

Date November 10, 2020

Contents

List of Figures

Chapter 1

Introduction

Recent developments in Artificial Neural Networks (ANNs) have allowed us to progress rapidly in areas such as computer vision, natural language processing and speech recognition. Although ANNs can consistently attain a high classification accuracy, they fail to provide an accurate and comprehensible account of what has been learnt [16].

One method to mitigate this incomprehensibility is to extract a set of logical rules from the trained ANNs. The ruleset should be comprehensible to humans, yet closely mimic the internal behaviour of the model. The majority of existing rule extraction algorithms cannot handle the complexity of an ANN with more than one hidden layer ([26], [30]).

This dissertation will focus on the implementation of a rule extraction algorithm for Deep Neural Networks (DNNs). The aim of this algorithm is to translate the embedded internal knowledge of the complex model into a small set of comprehensible rules. One such deep learning model created uses the gene expression data from cancer patients to classify them into those likely or unlikely to relapse.

1.1 Motivation

Machine learning models have shown their strengths in solving complex artificial intelligence problems. Despite this, there is a hesitance to adopt them into widespread use; it is difficult to determine why a model made a particular decision. The knowledge acquired by an ANN during training is embedded within its architecture and weights which are, at a first glance, incomprehensible. Transparency is needed from these black-box systems if they are to be trusted.

In safety-critical domains, often dealing with high risk and strong regulations, clear and comprehensible decision models are required; dangerous consequences could result from a hidden malfunction. Extracting a comprehensible model from these black box ANNs can potentially enhance their safety and usability here.

Furthermore, the General Data Protection Regulation (GDPR), effective from May 2018, illustrates the need for explainable models [29]. The European Union's "Right to

Explanation" states that when a decision is made by an autonomous system on behalf of a person, it is essential that clarification is given. The rule extraction system must therefore extract rules that closely mimic the internal behaviour of the network.

Rule-based assistive tools have already been deployed and adopted in clinical settings ([23], [11]). The transparency of these models provides clinicians with understandable evidence to support their predictions. Deep learning models can be shown to outperform the rule-based systems in accuracy but fail to achieve the same level of accepted transparency [28].

Extracting rules from an ANN can also expose 'hidden features' learnt by the model during training. These can indicate salient features and patterns otherwise embedded in the data. Rulesets are known to be *simulatable*. A simulatable model is one in which the user can take the input data and step through the entire decision process in a reasonable time and produce a prediction.

Part of this work involves extracting rules from a DNN classifying gene expression data from cancer patients. It would then be possible for the clinicians to inspect the extracted rulesets and generate predictions for new patients. It is also easy to observe the effect of a slight change in a single input feature in the prediction made for the patient. Transparency in these models would prove useful to oncologists looking to justify decisions made by the model.

This project aims to implement, test and evaluate an optimised rule extraction algorithm suitable for DNNs. The dissertation begins by discussing the relevant theory and background material required. An introduction of various existing rule extraction algorithms is given. This is followed by an in-depth description of the rule extraction algorithm I have implemented here. The algorithm is then extensively analysed and evaluated on a variety of DNNs trained for classification tasks.

Chapter 2

Preparation

This chapter offers an overview of the work undertaken prior to project implementation. The project aims to extract rules from trained ANNs. The ANNs for this project are trained to solve a variety of classification tasks.

Firstly, a description of the classification problem is given. This is followed by an explanation of the key ideas and concepts behind two potential solutions; artificial neural networks and rule-based classification systems. Furthermore, an introduction to existing rule extraction algorithms is provided. The chapter concludes by describing the tools used throughout implementation and defining the starting point of the project.

2.1 The Classification Problem

Classification is a supervised learning task; a *model* is built on the basis of labelled observations and used to predict the *class* of new unseen observations.

Each observation in dataset D is represented as a vector of k *feature* values, $\vec{x_i} \in \mathbb{R}^k$ and the corresponding class value, $y_i \in \mathbb{Z}^l$. There are l possible classes. For *binary* classification $l = 2$ and for *multi-class* classification $l > 2$. The classification problem can be formulated as a mathematical function,

$$f : \mathbb{R}^k \to \mathbb{Z}^l$$

such that $f(\vec{x_i})$ returns the correct classification y_i.

Data is used to learn or *train* a classifier, $\hat{f}$, to approximate the classification function f, such that $\hat{f}(\vec{x_i})$ returns the class prediction $\hat{y_i}$ and $\hat{y_i} = y_i$ for the majority of observations. Training data takes the form of labelled observations,

$$D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_n, y_n)\}$$

The learning process should avoid *overfitting.* Overfitting can occur when $\hat{f}$ does not generalise well to new data, often a result of learning the details and idiosyncrasies present in the training data.

There are a wide range of methods that approximate f and tackle the classification problem. An overview of artificial neural networks and rule-based classification systems is given below.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are brain-inspired machine learning algorithms. Such algorithms learn to perform tasks by considering existing examples, instead of being explicitly programmed with rules and instructions.

Although ANNs can take many forms, they are generally built from a collection of interconnected nodes known as artificial neurons. This project only considers fully connected feedforward ANNs .

2.2.1 Artificial Neurons

Artificial Neurons, also known as *perceptrons*, are simple computational units that form the building blocks for ANNs (Figure 2.1). Each neuron can take several input values, $x_1, x_2, ..., x_n$, and produces a single output or activation, y.

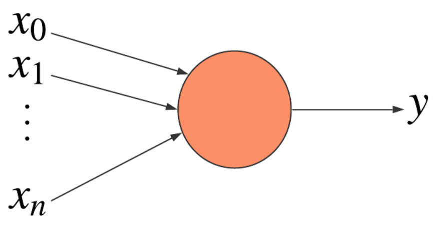


Figure 2.1: An Artificial Neuron

To compute y, each input is separately weighted and a linear transformation is applied to the sum. The result is then passed through a non-linear activation function $g : \mathbb{R} \to \mathbb{R}$.

$$y = g(b + \sum_i x_i w_i)$$

where w_i is the weight of x_i and b is the bias of a neuron.

The activation function governs the threshold at which the neuron is activated as well as the activation strength. The step function is the simplest activation function and activates the neuron if the result of the linear combination is above a threshold. The non-linearity allows additional complexity to be added when combining the inputs and can provide richer capabilities when modelling functions.

Popular activation functions include:

- *logistic function (sigmoid)*

$$g(z) = \frac{1}{1 + e^{-z}}$$

- *hyperbolic tangent (tanh)*
$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$
- *rectified linear unit (ReLU)*
$$g(z) = max(0, z)$$

The *sigmoid* and *tanh* functions are bounded and return results in the ranges $(0, 1)$ and $(-1, 1)$, respectively. The unbounded *ReLU* function returns results in the range $[0, \infty)$.

Neurons can be stacked to build larger structures; the output of one neuron forms the input for the next. This is the basis for *feedforward* neural networks.

2.2.2 Feedforward Neural Networks

Feedforward artificial neural networks are built from sequential layers of neurons, $h_0, h_1, ..., h_k, h_{k+1}$. Information in the network is only fed forward; the output of neurons in layer h_i are passed as input to neurons in layer h_{i+1}. In a *fully-connected* network, all neurons in layer h_i are connected to all neurons in layer h_{i+1}.

The input layer, h_0, receives information from the outside world and the output layer, h_{k+1}, indicates how the network has processed and responded to the information received. The layers in between, $\{h_1, h_2, ..., h_k\}$, are known as *hidden* layers. A network is said to be *deep* if it has more than one hidden layer.

A fully-connected feedforward DNN with two hidden layers is shown in Figure 2.2. The bias values, b, are depicted inside the neurons.

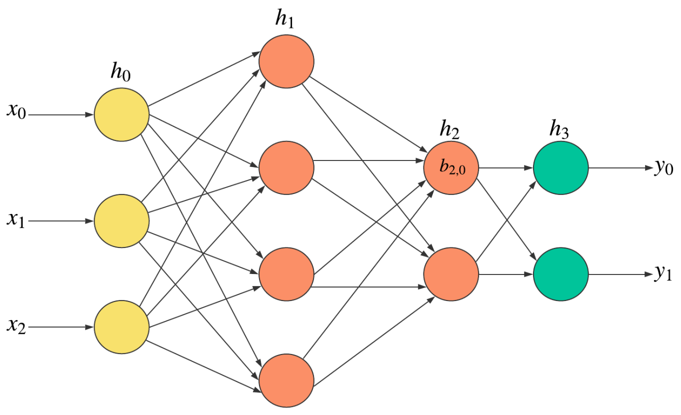


Figure 2.2: A fully-connected feedforward deep neural network with two hidden layers

ANNs can be used to solve classification problems. The input layer contains one neuron for each dimension (feature) of the vectors in the training dataset. The output layer contains a neuron for each possible predicted class i.e. $h_{k+1,i}$ represents the class y_i. The model classifies an instance to be in class y_i if $h_{k+1,i} = max(h_{k+1})$, i.e. the activation value of the neuron $h_{k+1,i}$ is the maximum in the layer.

The *softmax* function is often used to process the information from neurons in the output layer [14]. It converts the activation values into probabilities that can be used for prediction.

$$softmax(y_i) = \frac{e_{y_i}}{\sum_{j=1}^{l} e_{y_i}}$$

where l is the number neurons in the output layer. The function returns values in the range $[0, 1]$. All results sum to 1 to form a probability distribution. The larger the value for class y_i, the higher the probability. The class with the largest *softmax* value is chosen as the class prediction.

The *initialisation* of the network describes how the the initial set of weights is chosen. The learning process then finds the optimal weights w_i and biases b that minimise a loss function. The values are iteratively updated using gradient descent. The cross entropy loss function is often used for this [5].

2.3 Rule-Based Classification

ANNs are one potential solution to the classification problem. This section describes a more transparent alternative in the form of a rule-based classifier. A rule-based classifier learns a model represented as a *ruleset* $R = \{r_1, r_2, ..., r_n\}$, a collection of *IF-THEN* rules.

Rule-based classification systems are often chosen as rules are easy to understand; the logic used closely resembles how humans model logic. Explanations for predictions made by these systems are clear and traceable and new instances can be easily classified. Decisions made by these systems can be justified with ease.

Rule *induction* algorithms can be used to generate classification rules from a set of observations. Some algorithms first generate a decision tree from which rules can easily be extracted. Rule extraction algorithms generate rulesets from trained ANNs.

This section describes the form of classification rules and how they can be applied and evaluated on data. A description of rule induction can be found in Section 2.4.2.

2.3.1 Rules

Rule are expressed in the form,

IF antecedent THEN conclusion

where the rule *antecedent* or pre-condition is a *boolean expression* made up of *terms*, $t_1, t_2, ..., t_m$. Each term is a condition on an instance attribute, e.g. $x_3 > 0.4$.

A boolean expression is constructed from a set of terms connected by boolean operators such as disjunctions ($\vee$), conjunctions ($\wedge$), exclusive-or ($\oplus$) etc. The expression is said to be in *disjunctive normal form* (DNF) if it is made up of a disjunction of conjunctions of terms, e.g.

$$(t_1 \wedge t_3 \wedge t_5) \vee (t_1 \wedge t_2) \vee t_5$$

Rules in the ruleset are disjunctive. The ruleset is used to classify new instances and approximate the behaviour of the classification function f as close as possible. The rule conclusion is the class prediction, $\hat{y}$. The ruleset R is described as being *applied* to a new example $\vec{x_i}$ to classify it.

2.3.2 Ruleset Application

To classify a new example $\vec{x_i} \in \mathbb{R}^k$, each rule in the ruleset is applied. To apply a rule, each term in the rule antecedent is evaluated on $\vec{x_i}$. The rule is said to be *satisfied* by $\vec{x_i}$ if $t(\vec{x_i}) = \mathit{TRUE}$ for all terms in the rule premise i.e. the attributes of $\vec{x_i}$ satisfy all the rule pre-conditions.

A *default* rule, of the form ELSE $\hat{y} = y_j$, is often added to provide a robust solution. The conclusion of this rule is predicted in the case where $\vec{x_i}$ does not satisfy any rules in the ruleset.

Different strategies can be used to determine the overall class prediction. In a *majority voting* strategy, the class with the largest number of satisfied rules is predicted. A *rule confidence* strategy can be as be used. A total score for each output class is computed as a sum of the confidence scores of the satisfied rules for each class. The class with the largest total score is predicted.

2.3.3 Ruleset Evaluation

Evaluation metrics are described in order to assess the *quality* and *performance* of rules extracted. The accuracy of a ruleset refers to its ability to account for unseen examples. The comprehensibility refers to how easily the extracted model can be inspected and understood.

Rulesets with maximal accuracy are often very large as they aim to cover all possible cases and model the complex logic of the ANN. It is often the case that comprehensibility comes at the cost of accuracy. Rulesets can be shrunk to provide a more interpretable system. This is known as the *accuracy vs comprehensibility* trade-off [7].

Rule extraction algorithms aim to generate comprehensible rulesets that accurately mimic the behaviour of the high-accuracy opaque ANNs. A measure of fidelity can be introduced to determine how well the extracted rules mimic the behaviour of the ANN. A full description of the evaluation metrics can be found in Section 4.1.2.

2.4 Rule Extraction Algorithms

This section provides an introduction to some of the existing rule extraction approaches for ANNs. The knowledge acquired by an ANN during training is encoded in its architecture and weights. Rule extraction algorithms aim to translate the network encoding into a comprehensible set of logical rules that represent the embedded knowledge.

Different rule extraction approaches have been developed with differing priorities and *representation languages*. The term representation language was coined to describe the language used to explain the knowledge embedded in the network [8]. Representation languages include *IF-THEN* rules, fuzzy logic rules, decision trees and finite-state automata. Here I only consider algorithms extracting logical *IF-THEN* rules.

This section begins with a description of a taxonomy created to categorise rule extraction algorithms. This is followed by an introduction of three relevant existing rule extraction algorithms.

2.4.1 Taxonomy of Approaches

As new rule extraction algorithms were developed, a multi-dimensional classification taxonomy was proposed [3]. The primary classification dimension considers the expressive power and form of the extracted rules. Algorithms can extract rules using conventional *boolean* logic or alternative representations, including *fuzzy* logic ([15], [4]).

The second dimension considers the *transparency* with which the ANN is considered in the rule extraction process. *Pedagogical* methods view the ANN as a black box and focus on learning a function mapping the input features to the output class. The trained ANN generates examples and the algorithm learns the predictive behaviour of the model as a whole. *Decompositional* algorithms consider all hidden nodes and weights in the network. Rules are extracted at the neuron level. An *eclectic* approach considers elements of both.

In this project I implement a decompositional rule extraction algorithm for DNNs. Decompositional algorithms focus on extracting rules at the level of individual neurons in each layer of the trained ANN. This has the advantage of extracting knowledge from the internal structure of the ANN. There are few existing implementations of decompositional rule extraction algorithms for deep ANNs.

2.4.2 Existing Approaches

This section introduces three existing rule extraction algorithms. Each algorithm, at some point, extracts rules from decision trees. As such, all three approaches are based on decision tree induction, described below.

Decision Tree Induction

C4.5 was developed by Ross Quinlan and is a decision tree induction algorithm [24]. It can also be used directly as a pedagogical rule extraction algorithm. Given the training data and ANN predictions as input, C4.5 returns rules that describe the output of the ANN in terms of only the inputs.

Decision Trees (DTs) are recursive structures made up of a *root* and internal *nodes*, *branches* and *leaves*. DTs are easy to interpret and implement. DTs can be used for classification. Each node describes a test on an attribute of the example. The possible outcomes are described on branches to new nodes. The root is the topmost node and leaf nodes hold labels for possible classes.

To classify an example, begin at the root. Repeatedly, test the attribute specified at the node and traverse along the corresponding output branch until a leaf node is reached. The leaf node contains the class prediction. The process is deterministic; there is one possible path for each example.

The C4.5 algorithm can be used to *induce* DTs from training data that can be used for classification. It is a *greedy* algorithm; trees are constructed in a top-down recursive divide-and-conquer approach. DTs are built starting from the root and the C4.5 algorithm is applied recursively at each node.

Let S be the set of observations that have reached a node t. At each node C4.5 picks an attribute that splits the data into disjoint subsets $S_1, S_2, ..., S_n$. Ideally the test should split S such that the smallest tree is built overall. However, finding the optimal tree is NP-complete and often infeasible. Instead, a greedy-heuristic based on *information entropy* is used to find the 'purest' subsets.

The entropy or 'impurity' of a set S containing observations from C different classes is measured,

$$H(S) = -\sum_{i=1}^{C} p_i \log_2 p_i$$

where p_i is the fraction of observations for class i in S. $H(S) = 0$ when all observations belong to the same class. Of all the tests that can split S, C4.5 chooses the test that maximises the information gain. Information gain is the difference in entropy between the original and the partitioned sets,

$$Gain(S,t) = H(S) - H(S|T) = H(S) - \sum_{i=1}^{n} \frac{|S_i|}{|S|} H(S)$$

where $|S_i|$ denotes the number of observations in S. Once the tree is created, a pruning step is introduced in which leaf nodes are recursively merged until the best possible tree is achieved.

Rules can easily be extracted from a DT. Each path from root to a leaf generates a rule. The antecedent is a conjunction of all the conditions along the path and the conclusion is the contents of the leaf node. In general, rules have the form,

$$IF\ T_1 \wedge T_2 \wedge ... \wedge T_n\ \ THEN\ L_j$$

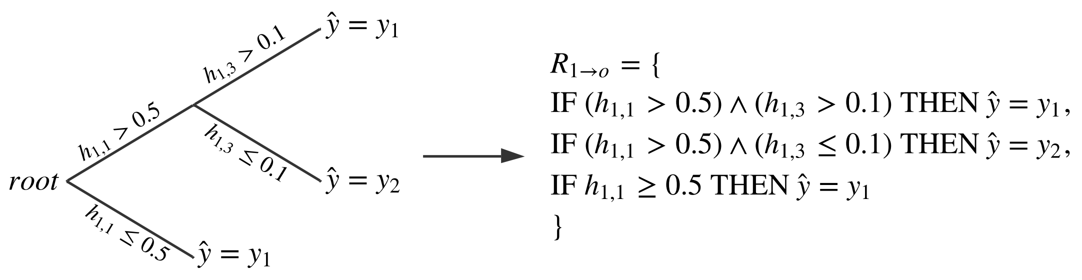
where $T_1, T_2, ..., T_n$ are the conditions of the nodes from root and L_j is the class of the leaf node.

C4.5 works well with both discrete and continuous data. Quinlan has since created C5.0 to improve upon C4.5 mainly in terms of speed and memory usage [25]. Details about the improvements in C5.0 can be found in Section 2.6.1.

CRED

CRED (Continuous/discrete) Rule Extractor via Decision Tree Induction is a decompositional rule extraction algorithm for ANNs [26]. CRED uses C4.5 to generate rules from DTs and only works with ANNs with one hidden layer.

First, the C4.5 algorithm is used to convert each neuron in the output layer into a DT using activation values sampled from neurons in the hidden layer. The nodes in the tree are tests on the activation values of neurons in the hidden layer. Leaves indicate class predictions. The DTs are converted into rulesets; rules are simplified and redundant and unsatisfiable rules and terms are deleted (Figure 2.3).



$R_{1\to o} = \{$
IF $(h_{1,1} > 0.5) \wedge (h_{1,3} > 0.1)$ THEN $\hat{y} = y_1$,
IF $(h_{1,1} > 0.5) \wedge (h_{1,3} \leq 0.1)$ THEN $\hat{y} = y_2$,
IF $h_{1,1} \geq 0.5$ THEN $\hat{y} = y_1$
$\}$

Figure 2.3: Result from step 1 of CRED: ruleset $R_{1\to o}$

$R_{1\to o}$ is an *intermediate* ruleset and describes the behaviour of the ANN output in terms of the hidden layer. For each term in $R_{1\to o}$, a DT is generated with C4.5 using values in the input layer. The intermediate ruleset $R_{i\to 1}$ is extracted. $R_{i\to 1}$ could include rules such as, IF $x_i, 1 > 0.2$ THEN $h_{1,1} > 0.5$.

The intermediate rulesets, $R_{i\to 1}$ and $R_{1\to o}$, are substituted and merged to build $R_{i\to o}$, a ruleset describing the ANN outputs in terms of the inputs. CRED cannot work with ANNs with more than one hidden layer.

DeepRED

DeepRED (Deep Neural Network Rule Extraction via Decision Tree Induction) is one of the first decompositional rule extraction algorithms for DNNs [30]. It is an extension of the CRED algorithm.

Rules are extracted in a step-wise process. DTs are first generated between adjacent layers in the network using the C4.5 algorithm; sampled activation values from layer h_{j-1} are used as attributes for layer h_j. Rulesets are extracted from the DTs and simplified. Redundant and unsatisfiable rules and terms are deleted.

The layer-wise rulesets are then substituted and merged to achieve a ruleset that describes the network output only in terms of values in the input. The algorithm can extract accurate rulesets from DNNs and outperformed a pedagogical baseline. However, a large number of the experiments were not successful and exceeded time and memory constraints. The algorithm did not scale well to larger datasets in which the merging process often proved difficult. The algorithm also did not scale well to multi-class classification process.

2.5 Requirements Analysis

This section narrows down the scope of my project through an analysis of the requirements. The core aims of the project are described below:

- Build, train and evaluate DNNs for a wide range of of classification tasks.
- Use the C5.0 algorithm to extract DTs and subsequently rules between adjacent layers in each DNN. To improve upon existing decompositional approaches, a more efficient algorithm is chosen to induce rules between adjacent layers of the ANN.
- Build an algorithm to recursively merge the layer-wise rulesets into a ruleset describing the output of the DNN in terms of only the input. The more efficient C5.0 algorithm should return fewer rules thus easing the rule merging process.
- Implement a method that uses hill climbing heuristics to rank and prune the extracted rules. Smaller rulesets with shorter rules are easier for humans to comprehend.
- Perform a comparative evaluation between my rule extraction algorithm and a pedagogical baseline.

I split the project requirements into manageable weekly tasks and devised a timeline for the implementation and write-up of the dissertation. Multiple 'slack time' slots were included in the timeline to account for unexpected delays. The tools required for this project are described below.

2.6 Choice of Tools

2.6.1 Libraries

The **Keras** library was used to build, train and evaluate the DNNs for this project. **Keras** is an open-sourced neural network library. It is highly optimised and widely used in ANN research.

The **NumPy** and **Pandas** libraries were also used throughout the project for efficient data processing and storage. Although difficult, a working implementation of the **C5.0** algorithm was found. The full details of this are discussed below.

C5.0

As mentioned C5.0 is an extension of the C4.5 DT induction algorithm described in Section 2.4.2. C5.0 shares much of its core algorithm with its predecessor, the algorithm first grows a DT that is collapsed into rules.

C5.0 has been shown to produce DTs with similar predictive accuracies to C4.5 but the DTs are much smaller and in turn yield shorter rulesets. One reason for this is that C5.0 introduces an extra global pruning step of the DT. Sub-trees are removed until the error rate of the tree exceeds a threshold. The rules extracted from the DT are also pruned.

The full multi-threaded implementation of C5.0 is only available with a commercial licence. The C source code for a single-threaded version is available under the GNU General Public licence and has been ported to R as the `C50` library. There is currently no stable version of the C5.0 algorithm available for Python.

To overcome this I wrote the wrapper implementation required to call the R `C50` library and parse the results back to my Python program. This required an understanding of both R and Python. This process was a lot more difficult than anticipated due to a lack of documentation. Understanding specifically how the C5.0 algorithm worked required parsing the C source code. Rules are returned from `C50` as strings that needed to each be parsed into corresponding objects in Python.

The `C50` implementation of C5.0 was used with the default parameters.

2.6.2 Programming Language

The majority of the project was implemented in **Python** using Python 3.6. It is the language of choice for the majority of research in machine learning and made the most sense for this project. Python syntax is concise, readable and easy to pick up.

My IDE of choice was **PyCharm** as it provided a useful integration with version control as well as useful tools for debugging and reformatting the code.

2.6.3 Computational Resources

The majority of this project was developed and tested on my personal laptop, running Windows on 2.5 GHz Intel Core i7 with 16GB RAM. The High Performance Computing Service (HPC) was used to generate data in the form of trained DNNs for rule extraction (see Section 3.4.2). Unfortunately, the `C50` implementation is not compatible with the HPC so the rule extraction algorithm was run only on the laptop.

2.6.4 Version Control

For version control, I chose **git**. This allowed me to track key changes in the project and rollback to previous implementations if necessary. The code in the git repository was pushed regularly to **GitHub**.

2.6.5 Backup Strategy

All written work was carried out on **Overleaf**, an online LaTeX editor. Both code and written work were continuously backed up to **Microsoft One Drive**. Weekly backups of all relevant project files were made to an external HDD.

2.7 Starting Point

Prior to this project I had very little experience with machine learning and no practical experience programming in Python.

Previous Tripos courses proved extremely useful for the theoretical background of the project. In particular, the 1B Logic and Proof course contained the majority of information needed to simplify and manipulate propositional logical rules. The Part II Natural Language Processing Unit of Assessment provided experience with using Python and popular machine learning libraries as well as building and evaluating accurate models without overfitting. Additionally, the 1B Artificial Intelligence course provided useful background knowledge for understanding ANNs and how they work.

In terms of technical skills, I had no prior experience with R or Python. I had never written a substantial piece of code or worked with any of the machine learning libraries before. I also had no experience using facilities similar to the HPC which I used substantially toward the end of the project.

Chapter 3

Implementation

This chapter focuses on the implementation phase of the project, highlighting the motivations for design choices and focusing on the high-level structure of the project. This project involves the implementation and evaluation of a rule extraction algorithm for DNNs.

Firstly, I explain and justify my implementation of logical rules. An in-depth explanation of the rule extraction algorithm I implement follows this. Rules in the extracted ruleset are unordered and disjunctive. A rule ranking procedure is introduced to score and subsequently rank the rules extracted from the ANNs.

Rules are extracted from trained ANNs. A description of the process used to generate and train suitable ANNs is then given. The chapter concludes with an overview of the project repository.

3.1 Rule Representation

This section describes the form of the rulesets extracted by the rule extraction algorithm. I begin by describing the form each rule in the ruleset takes. I then discuss how this representation is implemented.

A *ruleset* is an unordered set of disjunctive rules, $R = \{r_1, r_2, ..., r_n\} = r_1 \vee r_2 \vee ... \vee r_n$. A rule is of the form,

IF antecedent THEN conclusion

where the rule antecedent is a conjunction of terms.

Each term is a *split value* and is determined by a neuron and its activation value threshold. A neuron, $h_{l,p}$, is identified by the network layer l and position p at which it appears. A term takes the form, $x_p > t$ or $x_p \leq t$ for input attributes and $h_{l,p} > t$ or $h_{l,p} \leq t$ for hidden units.

A conjunction of terms is referred to as a *conjunctive clause* and is represented as a set, $t_1 \wedge t_2 \wedge ... \wedge t_m = \{t_1, t_2, ..., t_m\}$. The clause is *true* if all the terms are *true*. The rule premise is a conjunctive clause.

3.1.1 Implementation

This section describes the implementation of logical *IF-THEN* rules and rulesets and their features. Figure 3.1 provides the implementation for `Rule` and its associated classes.

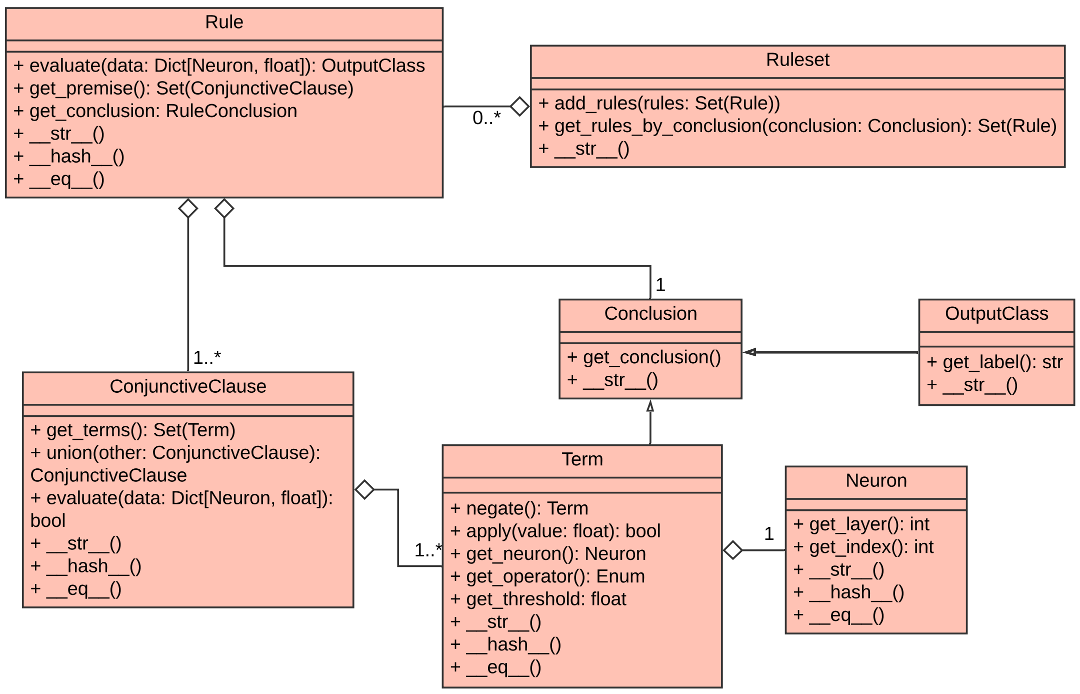


Figure 3.1: UML diagram for the `Rule` object and its associated classes

A rule is represented with a `Rule` object that has premise and conclusion. The `Rule` premise is implemented as a set of `ConjunctiveClause` objects. The conclusion is a `Term` for intermediate rules and an `OutputClass` for the final rules.

A `ConjunctiveClause` is made up of a set of `Term` objects. Each `Term` is made up of a `Neuron`, an operator and a threshold value. A `Term` can be applied to a value and can also be negated, $\overline{h_{1,2} > 0.3} = h_{1,2} \leq 0.3$. A `ConjunctiveClause` of terms can also be applied to data in the form of a dictionary of neurons mapped to values,

$$\{(h_{1,0}, 0.4), (h_{1,1}, 0.1), ...\}$$

The `Ruleset` object stores a set of disjunctive rules. To reduce memory usage, rules with identical conclusions are stored in DNF form e.g.,

$$IF\ (h_{1,3} > 0.4 \wedge h_{1,1} \leq 0.1) \vee (h_{1,1} > 0.5) \vee (h_{1,1} \leq 0.1 \wedge h_{1,2} < 0.3)\ THEN\ \hat{y} = y_2$$

is a ruleset of three rules with the same conclusion. The `Rule` premise now stores a set of `ConjunctiveClause` objects. A simple *IF-THEN* rule is a `Rule` whose premise is a singular `ConjunctiveClause`.

In order to minimise memory usage, it is important that duplicates of terms and rules are not stored. Unordered sets are used throughout the implementation to handle this automatically at each step of the algorithm. An ordering of the sets is not required as both $\wedge$ and $\vee$ operations are communicative, idempotent and associative.

The Python set implementation and its built-in operations was used. This allowed rulesets and clauses to have very efficient $O(1)$ access and $O(1)$ insertion costs on average. Relevant methods concerning object equality and hash values were overwritten to allow the custom objects to be stored in a set. Furthermore, the hashable objects were made immutable to ensure the hash was not altered once the object was in the set. The string representations of all objects were also overwritten to allow rules to easily be inspected and read.

3.2 Rule Extraction

The rule extraction algorithm implemented is inspired by the decompositional nature of DeepRED described in Section 2.4.2. This section provides an in-depth explanation of the rule extraction algorithm I have implemented.

3.2.1 Algorithm

This rule extraction algorithm considers a multi-class classification problem with n instances, $\vec{x_1}, \vec{x_2}, ..., \vec{x_n}$, each with an associated class, $y_i \in \{y_1, y_2, ..., y_u\}$ and an ANN with k hidden layers, $\{h_0, h_1, ..., h_k, h_{k+1}\}$. Input and output layers, h_0 and h_{k+1}, are also referred to as i and o, respectively.

Each layer h_i is made up of H_i neurons and so $H_0 = |\vec{x_i}|$. It is assumed that there is one output neuron for each class, $H_{k+1} = u$. Activation values sampled at layer h_i for observation $\vec{x_j}$ are denoted $h_i(\vec{x_j})$.

Pseudocode for the rule extraction algorithm is detailed in Algorithm 1. The subroutines called in the algorithm are described in Section 3.2.2.

Rules are extracted per class, as such even binary classification problems are treated as multi-class. The algorithm iterates over each class y_v separately. The ruleset R^v is first initialised with a rule indicating the network prediction class R^v (line 2). Each hidden layer is then processed in descending order and an empty intermediate ruleset, I, is initialised.

The C5.0 algorithm then iterates over each unique term present in the rule premises of R^v (line 9), e.g. if $R^1_{h_2 \to o} = \{\mathit{IF}\ h_{2,1} > 0.1 \wedge h_{2,2} \leq 0.3\ \mathit{THEN}\ \hat{y} = y_1, \mathit{IF}\ h_{2,1} > 0.1\ \mathit{THEN}\ \hat{y} = y_1\}$, $T = \{h_{2,1} > 0.1, h_{2,2} \leq 0.3\}$. The data passed to C5.0 is composed of sampled activation values from layer h_j (line 6). The target is made up of the term applied to activation values sampled from layer h_{j-1} (line 8), e.g. if $t = h_{2,1} > 0.1$ and $h_1(\vec{x_1}) = [0.05, 0.3, 0.0]^T$, $y_1{}' = \mathit{TRUE}$.

C5.0 returns rules that describe layer h_j by means of layer h_{j-1}. These rules form the intermediate ruleset $I_{h_{j-1} \to h_j}$. $I_{h_{j-1} \to h_j}$ is substituted into R^v as soon as it is extracted to achieve $R^v_{h_{j-1} \to o}$ (line 11). Details of the substitution steps can be found in Section 3.2.2.

Algorithm 1 Rule Extraction Algorithm

Input: Artificial neural network $\{h_0, h_1, ..., h_{k+1}\}$, training data $\{(\vec{x_1}, y_1), ..., (\vec{x_n}, y_n)\}$

Output: Ruleset approximating the neural network

```
1:  for class $y_v \in y_1, y_2, ..., y_u$ do
2:      $R^v_{h_k \to o} \leftarrow \{IF\ h_{k+1,v} > 0.5\ THEN\ \hat{y} = y_v\}$
3:      for hidden layer $j = k, k-1, ..., 1$ do
4:          $I_{h_{j-1} \to h_j} \leftarrow \emptyset$
5:          $T = Set(getTermsFromRulePremises(R^v_{h_j \to h_{j+1}}))$
6:          $\vec{x_1}', \vec{x_2}', ..., \vec{x_n}' \leftarrow h_j(\vec{x_1}), h_j(\vec{x_2}), ..., h_j(\vec{x_n})$
7:          for $t \in T$ do
8:              $y_1', y_2', ..., y_n' \leftarrow t(h_{j+1}(\vec{x_1})), t(h_{j+1}(\vec{x_2})), ..., t(h_{j+1}(\vec{x_n}))$
9:              $I_{h_{j-1} \to h_j} \leftarrow I_{h_{j-1} \to h_j} \cup \texttt{C5.0}((\vec{x_1}', y_1'), (\vec{x_2}', y_2'), ..., (\vec{x_n}', y_n'))$
10:         end for
11:         $R^v_{h_{j-1} \to o} \leftarrow substitute(I_{h_{j-1} \to h_j}, R^v_{h_j \to o})$
12:         $R^v_{h_{j-1} \to o} \leftarrow deleteUnsatisfiableRules(R^v_{h_{j-1} \to o})$
13:         $R^v_{h_{j-1} \to o} \leftarrow deleteRedundantTerms(R^v_{h_{j-1} \to o})$
14:     end for
15: end for
16: $R_{i \to o} \leftarrow R^1_{i \to o} \cup R^2_{i \to o} \cup ... \cup R^u_{i \to o}$
17: returns: $R_{i \to o}$
```

Unsatisfiable rules and redundant terms are eliminated (lines 12 and 13), details of which are found in Section 3.2.2.

At each substitution step, $R^v_{h_j \to o}$ describes the network output in terms of layer h_j. All rules in R_v have the same conclusion, namely $\hat{y} = y_v$. As such, R_v can be represented with a single rule in disjunctive normal form (DNF), referred to as a *total rule*. The ruleset $R^1_{h_2 \to o} = \{IF\ h_{2,1} > 0.1 \wedge h_{2,2} \leq 0.3\ THEN\ \hat{y} = y_1, IF\ h_{2,1} > 0.1\ THEN\ \hat{y} = y_1\}$ can be represented as a total rule, $IF\ (h_{2,1} > 0.1 \wedge h_{2,2} \leq 0.3) \vee h_{2,1} > 0.1\ THEN\ \hat{y} = y_1$. The premise of a total rule is a set of conjunctive clauses, each clause representing an *IF-THEN* rule. The size of the total rule indicates the number of rules. A total rule is extracted for each output class.

Once all hidden layers are processed, $R^v_{i \to o}$ consists of only rules that describe the behaviour of the neural network output in terms of its input features.

3.2.2 Rule Manipulation

This section describes the methodology used to substitute rulesets in the rule extraction algorithm (Algorithm 1). It goes on to define and describe how redundant terms and unsatisfiable rules are detected.

Substituting Rulesets

Here it is assumed that $R^v_{h_j \to o}$ is stored as a DNF total rule. An intermediate ruleset $I_{h_{j-1} \to h_j}$ is substituted into the total rule $R^v_{h_j \to o}$ to achieve $R^v_{h_{j-1},o}$.

Pseudocode for the substitution method is described in algorithm 2.

Algorithm 2 Rule Substitution Algorithm

```
    Input: Intermediate ruleset $I_{h_{j-1} \to h_j}$, total rule $R^v_{h_j \to o}$
    Output: Total rule $R^v_{h_{j-1} \to o}$
 1: procedure SUBSTITUTE($I_{h_{j-1} \to h_j}, R^v_{h_j \to o}$)
 2:     $premise(R^v_{h_{j-1} \to o}) \leftarrow \emptyset$
 3:     for $clause \in premise(R^v_{h_j \to o})$ do
 4:         $C \leftarrow []$
 5:         for $term \in clause$ do
 6:             $C \leftarrow C + getRulePremisesByConclusion(I_{h_{j-1} \to h_j}, term)$
 7:         end for
 8:         for $new\_clause \in CartesianProduct(C)$ do
 9:             $premise(R^v_{h_{j-1} \to o}) \leftarrow premise(R^v_{h_{j-1} \to o}) \cup new\_clause$
10:         end for
11:     end for
12: end procedure
```

Each clause in $R^v_{h_j \to o}$ is a conjunction of terms, $\{t_1, t_2, ..., t_m\}$. Each term t_i can be substituted with a rule premise from $I_{h_{j-1} \to h_j}$ with conclusion t_i. However, there may be multiple rules with conclusion t_i.

The function *getRulePremisesByConclusion* takes the intermediate ruleset $I_{h_{j-1} \to h_j}$ and term t_i and returns the set of rule premises that have the conclusion t_i (line 6). List C is composed of these sets, one for each term t_i in the original conjunctive clause, e.g. $C = [\{c_1, c_4\}, \{c_3\}, \{c_2, c_3\}]$.

The terms from each combination of clauses in C are used to construct new clauses. These clauses form the premise of the total rule $R^v_{h_{j-1},o}$. The Cartesian product is used to compute the clause combinations (line 8).

The Cartesian product between two sets, X and Y, is the set of all ordered pairs with one element each from X and Y. This can be extended for multiple sets. For n sets, $X_1, X_2, ..., X_n$ the Cartesian product is defined,

$$X_1 \times X_2 \times ... \times X_n = \{(x_1, x_2, ..., x_n) | \forall_{i \in [1..n]} x_i \in X_i\}$$

The cardinality of the output is the product of the cardinality of the input sets, $|X_1| \cdot |X_2| \cdot ... \cdot |X_n|$. As such, the substitution phase is often the longest and most memory intensive step, the number of clauses in R_v can grow exponentially with each iteration over a hidden layer.

The Python `itertools` module is used to compute the Cartesian product. The module provides fast and memory efficient tools. The implementation uses a generator to compute the combinations one at a time instead of building up the entire output in memory.

Unsatisfiable Rules

A rule is said to be *unsatisfiable* if it contains a pair of contradicting or mutually exclusive terms such as,

$$IF\ x_1 < 0.1 \wedge x_1 \geq 0.3\ \ THEN\ \hat{y} = y_3$$

where the conditions on x_1 can't all be satisfied. The entire rule is unsatisfiable and is subsequently eliminated.

Unsatisfiable rules are detected if they include a conjunctive clause in which at least two terms are mutually exclusive. The entire rule is removed from its subsequent ruleset.

Redundant Terms

Within a rule, a term is redundant if a more general term is also present. In the rule, e.g.

$$IF\ x_1 < 0.1 \wedge x_3 \geq 0.5 \wedge x_1 < 0.5 \wedge x_3 \geq 0.7\ \ THEN\ \hat{y} = y_3$$

the terms $x_1 < 0.1$ and $x_3 \geq 0.7$ are redundant and can be deleted.

The algorithm builds up a dictionary mapping each attribute or neuron with its most general threshold value in the clause. The dictionary is used to build a new clause to replace the old. The removal of duplicate terms is handed implicitly as terms are stored in an ordered set.

3.2.3 Optimisations

Algorithm 1 uses a more optimised DT induction algorithm and other optimisations to improve upon current decompositional approaches. DeepRED suffers issues of scalability due to time and memory constraints. To account for this, the control flow of the algorithm is also updated to reduce memory usage. Further details on all optimisations introduced are described below.

The C4.5 algorithm is commonly used to generate DTs between adjacent layers of the network from which rules are inferred. Algorithm 1 uses the newer C5.0 algorithm. C5.0 is faster, more memory efficient and builds smaller simpler DTs than its predecessor C4.5. I am yet to come across a decompositional rule extraction algorithm that utilises C5.0. Further details about the C5.0 algorithm can be found in Section 2.6.1.

There are many other opportunities to optimise, for example an existing algorithm, DeepRED, iterates over the hidden layers of the ANN twice. During the first iteration an intermediate ruleset for each pair of adjacent layers is generated and stored, $[I_{i\rightarrow h_1}, I_{h_1\rightarrow h_2}, ..., I_{h_k\rightarrow o}]$. During the second iteration, rulesets for pairs of adjacent layers are substituted. Only after each substitution are unsatisfiable rules and redundant terms deleted.

As a result, in DeepRED T (line 5) contains terms that are redundant or unsatisfiable and extra computation is required to iterate over them and induce DTs. To overcome this, this algorithm performs only one iteration over the hidden layers of the ANN. An intermediate ruleset is extracted and immediately substituted into R^v (line 11). Redundant terms and unsatisfiable rules are eliminated after this step prior to the next iteration. This can reduce the size of T and decrease computation time. Memory usage is also decreased drastically as only one intermediate ruleset is stored at a time.

In the DeepRED algorithm, data for the DT induction algorithm was computed during each iteration over the terms $t \in T$. Here, the predictors are computed once prior to the iteration to save on computation (line 6).

3.3 Rule Ranking

The task of finding the optimal ruleset is NP-hard and existing rule extraction algorithms often generate large rulesets [12]. To increase comprehensibility and shrink the ruleset, rules can be deleted. This section describes a mechanism to rank the rules in the ruleset. This ranking can be used to decide which rules to delete.

There are various metrics that can be used to rank rules in the extracted ruleset. It is common to use the accuracy score of the rule to determine which to delete. However, a rule that is tailored to a specific example case can have a very high accuracy but a very low generalisability.

Mashayekhi and Gras suggest computing a score for each rule to create a ranking [19]. This score considers the accuracy of the rule, the rule coverage and the comprehensibility of the rule simultaneously. Muller et al. successfully use this approach to rank rules extracted from a random forest [22].

For this project, I implement the scoring function introduced by Mashayekhi and Gras,

$$rankScore = \frac{cc - ic}{cc + ic} + \frac{cc}{ic + k} + \frac{cc}{rl}$$

where cc (correctly classified) is the number of instances classified correctly by the rule. The variable ic (incorrectly classified) is the number of incorrectly classified training samples covered by the rule and rl indicates the length (number of terms) in the rule premise. Lastly, k is a predefined constant and $k = 4$, as seen in *ranking*. Mashayekhi and Gras report no significant change in their results from modifying k. Rules with a higher accuracy are ranked higher. The formula also prefers shorter rules as they are more comprehensible for humans.

A greedy hill-climbing approach is then applied to shrink the ruleset. Iteratively, a percentage of the lowest ranked rules in the ruleset are removed and the accuracy of the smaller, more accessible ruleset is recorded. Rules can be deleted until the accuracy of the ruleset falls below a threshold. Muller et al. discover that removing 50% of the lowest ranked rules decreases the accuracy of the ruleset by less than 3%.

3.4 Data Generation

Data, in the form of trained DNNs, is required to evaluate the performance of the rule extraction algorithm. This section describes the process of generating trained DNNs to evaluate the performance of the rule extraction algorithm.

Preliminary investigations revealed that the ANN training configuration, the initial weights and training data all had a significant impact of the performance of subsequent rule extraction. To overcome this, a data generation pipeline was devised (Figure 3.2).

3.4.1 Pipeline

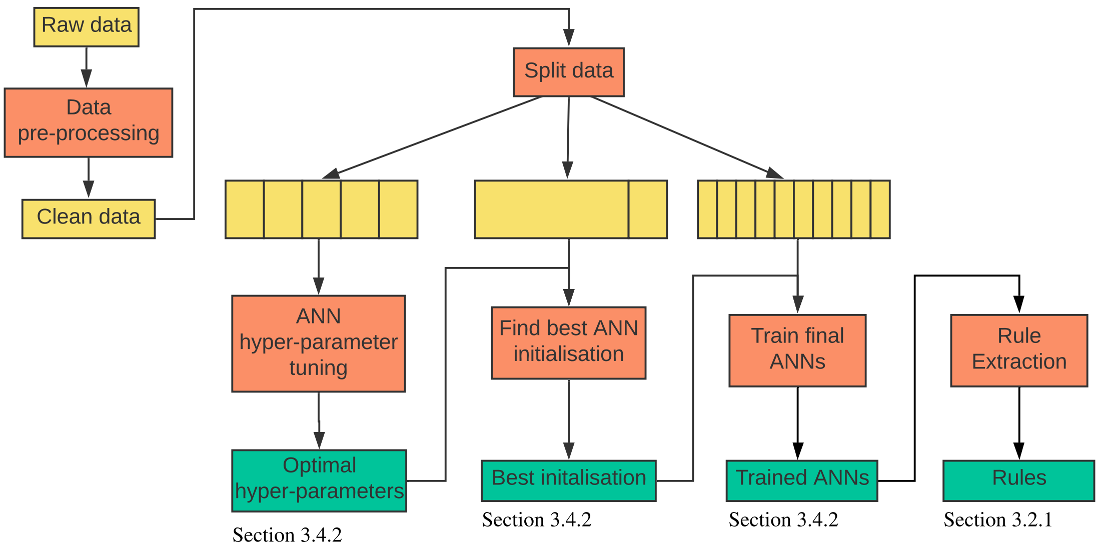


Figure 3.2: Pipeline illustrating the data generation process

The raw data is pre-processed. Prior to model training, data is cleaned and transformed. Data pre-processing is an important step to maximise the quality of information that can be extracted.

The data is standardised. All the categorical features are encoded numerically to be understood by the model. All numerical features are scaled to be in the range $[0, 1]$ so that attributes have similar magnitudes. This prevents features with larger values from carrying more significance in the network. Rows with missing data are eliminated. If the dataset for a multi-class classification task is transformed to a binary classification task, the dataset is reduced to allow for a roughly equal class distribution.

The clean data is first split in five stratified folds. A cross-validated search over the ANN hyper-parameter subspace returns the optimal ANN training configuration (see Section 3.4.2). This training configuration is fixed hereafter.

To control for the ANN initialisation, the data is then fixed. The clean data is split into a single 80/20 fold so that the same data is used as different initialisations are tested (see Section 3.4.2). The best initialisation is then imposed as the final ANNs are trained.

The early tests revealed that there was large variance in the number of rules extracted from each fold. As such a larger 10-fold cross-validation is performed for which the data is split into ten stratified folds. Rules are extracted for each fold.

The `StratifiedKFold` and `ShuffleSplit` implementations in scikit-learn are used to split the data. The ordering of the indices is fixed to ensure reproducible results.

3.4.2 Neural Network Generation

Hyper-parameter Tuning

The grid search algorithm exhaustively searches through a specific subset of the hyper-parameter space and selects those that yield the DNN with the highest classification accuracy [6]. Grid search is a well-studied technique. As such the `GridSearchCV` implementation by scikit-learn is selected as it is optimised and widely used.

The algorithm trains a DNN with each possible tuple of the hyper-parameters selected in their Cartesian product. The tuple that achieves the DNN with the highest accuracy is chosen. The *batch size*, *epochs* and the *number of neurons* in each hidden layer of the DNN are tuned with *batch size* $\in [10, 20, 50, 100]$ and *epochs* $\in [50, 100, 200, 400]$. The number of neurons in the first hidden layer of the DNN is one of $[10, 15, 50, 100]$ and $[2, 5, 10, 50]$ for the second hidden layer. These values give rise to 256 possible candidate DNNs.

A 5-fold cross-validated search is performed to avoid overfitting the hyper-parameters. This process is very computationally expensive as it requires training 5 data folds for each of the 256 candidate models, totalling 1280 trained models. The High Performance Computing Service was required for this.

Network Initialisation

Initially DNNs were initialised with weights randomly selected from a uniform distribution. In a 10-fold cross-validation, each fold generates a DNN initialised with different weights and trained on a different subset of the data. The training process for DNNs aims to minimise a loss function. The loss function has many local minima and the initialisation of weights determines the minima the network is likely to fall into. Early experimentation revealed that although different minima had comparable accuracies, the number of rules extracted could differ significantly. The results of the early experimentation can be seen in Section 4.2.1.

To account for this, I introduce an intermediate step that selects an initialisation of weights to impose on all the networks. To control for the initialisation, the data is fixed. Five DNNs with different initialisations are trained on this data and rules are then extracted from each. The best initialisation is the one that gives rise to the smallest ruleset. This initialisation is then imposed.

Training

The performance of the rule extraction algorithm is evaluated using 10-fold cross validation to account for the variations in the data that affect the rule extraction performance.

For each fold of each dataset, a DNN is trained with the hyper-parameters specified in Section 3.4.2. The network weights are initialised with weights found in Section 3.4.2 and then trained. The trained DNNs are evaluated and form the input for the rule extraction algorithm.

3.4.3 Datasets

In this project, DNNs are trained for both artificial and real-world classification tasks. The datasets and corresponding trained DNNs are referred to by the same *italicised* name.

METABRIC

The METABRIC (Molecular Taxonomy of Breast Cancer International Consortium) dataset is specific to this dissertation and is made up of data from cancer patients [10].

This dataset is chosen to illustrate the utility of rule extraction in a medical context. Clinicians often hesitate to trust a 'black-box' ANN model that provides little justification for the potentially life-changing decisions it makes. Establishing the trust of clinicians through explainable machine learning models is key to their adoption in medical applications.

The *MB-GE-ER* classification problem considers Estrogen Receptors (ER). ER are considered biomarkers for survival and indicate better survival and recovery rates in patients. Oncologists are familiar with this metric as a trusted indicator of potential relapse.

The data used for the *MB-GE-ER* classification task is a subset of the original METABRIC dataset shown to have biological relevance for this particular task [20]. The data is made up of 1000 real-valued attributes. Each attribute describes the gene expression level of a specific gene. The gene expression level corresponds to the amount of RNA produced from the gene in a given time. The target, ER, is either *positive* or *negative*. *MB-GE-ER* is a binary classification task.

Artificial Datasets 1 and 2

Artif-1 and *Artif-2* are artificial datasets created by the authors of DeepRED [30]. *Artif-1* and *Artif-2* are binary classification tasks.

Examples in the dataset are tuples of five attributes and a class label, $(x_1, x_2, x_3, x_4, x_5, y)$. Attributes $x_1 \in \{0, 0.5, 1\}$ and $x_2 \in \{0, 0.25, 0.5, 0.75, 1\}$ are discrete while $x_3, x_4, x_5 \in [0, 1]$ are continuous. The class labels for *Artif-1* and *Artif-2* are computed with an *IF-THEN* rule in DNF.

For *Artif-1*, *IF* $(x_1 = x_2) \vee (x_1 > x_2 \wedge x_3 > 0.4) \vee (x_3 > x_4 \wedge x_4 > x_5 \wedge x_2 > 0)$ *THEN* $y = y_2$ *ELSE* $y = y_1$. The greater-than relations between real-valued attributes are difficult for DTs to model.

For *Artif-2*, *IF* $(x_1 = x_2) \lor (x_1 > x_2 \land x_3 > 04) \lor x_5 > 0.8$ *THEN* $y = y_1$ *ELSE* $y = y_2$. *Artif-2* contains an attribute, x_4, that has no effect on y. It could be observed whether the rule extraction algorithm can pick this up.

The artificial datasets allow direct comparisons to be made between the true models, the logical rules above, and the extracted rules. It can provide unique insight into what the ANN has learnt as the true models of most real-world tasks remain unknown.

Wisconsin Diagnostic Breast Cancer

The *BreastCancer* dataset is made up of 30 real-valued attributes describing digitised images of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei that are present in the image e.g. the height and width of the image.

The images are classified as *benign* and *malignant*. The dataset is fairly small with only 569 examples overall. *BreastCancer* is a binary classification task.

MNIST

The MNIST dataset is made up of images of handwritten digits [18] and provides two classification tasks here. Each image is classified as an integer digit in the range $[0, 9]$. The digits are described with 784 grey-scale attributes, i.e. values in the range $[0, 255]$.

This multi-class classification task is transformed into a binary classification task, *MNIST-binary*. As in the DeepRED paper, an ANN is first trained with the multi-class data. The problem is then reduced to distinguish only between class '1' and all other classes (1 vs all).

Letter Recognition

The Letter Recognition dataset can be found found in the UCI Machine Learning Repository and provides two classification tasks here. It contains the 26 letters of the English alphabet as classes and is a multi-classification task. Each instance describes an image of a handwritten letter.

The pixelated images are converted into 16 attributes that describe it. These include, the width and height of the image, among others. Each feature is described with an integer in the range $[0, 15]$.

As before, this dataset is also transformed into a binary classification task *LetterRecognition-binary*. The ANN is first trained with the multi-class data and the data is then reduced to distinguish only between class 'A' and all other classes.

3.5 Repository Overview

This section describes a high-level overview of the structure of the project source code as found in the repository. The implementation of the data generation pipeline and the rule extraction systems were all written from scratch.

- `Project`
  - `Write-up` - Academic literature and LaTeX files for the project proposal and dissertation
  - `Implementation`
    - `data` - Pre-processed datasets for classification tasks
      - `<dataset-name>` - Each dataset followed the same structure
        - `data.csv`
        - `<n-fold>` - Contains the indices used to split that data into n folds
    - `src` - Python source code for the project
      - `Rules` - Implementation of `Ruleset`, `Rule`, `Term` etc
      - `evaluate_rules` - Evaluation metrics
      - `rank_rules` - Rule ranking mechanism
      - `logic_manipulator` - Manipulate logical rules as seen in Section 3.2.1, contains the rule substitution implementation
      - `extract_rules` - Implementations of various rule extraction algorithms
        - `RuleEx` - Pseudocode for this algorithm can be found in Algorithm 1
        - `DeepRED-C5` - An implementation of DeepRED
        - `Ped-C5` - A pedagogical C5.0 baseline

Chapter 4

Evaluation

This chapter presents an analysis and evaluation of the rule extraction algorithm implemented. It begins with a description of the framework used to evaluate the system. This includes alternative rule extraction algorithms and multiple evaluation metrics to perform a comparative evaluation. I then discuss the results of this. The chapter concludes with an analysis of the rule ranking procedure.

4.1 Evaluation Framework

4.1.1 Rule Extraction Systems

For the purpose of evaluation, the optimised decompositional rule extraction algorithm implemented in Section 3.2.1 is referred to as *RuleEx* hereafter. *RuleEx* is compared with two alternative systems: (1) A pedagogical C5.0 rule extraction algorithm, *Ped-C5* (2) An implementation of DeepRED that uses C5.0, *DeepRED-C5*.

Ped-C5

I implement a pedagogical rule extraction algorithm using the C5.0 algorithm [25]. A pedagogical approach extracts rules by creating a mapping directly between the ANN input and output. C5.0 is called directly on the training data and the labels predicted by the ANN.

The algorithm does not consider the inner structure of the network. As a result, pedagogical approaches are very fast and easily scale to larger networks. C5.0 uses the predictions of the ANN as input to directly generate rules for classification.

DeepRED-C5

DeepRED is one of the first rule extraction algorithms for DNNs. It would have been ideal to evaluate *RuleEx* against the original DeepRED algorithm. Unfortunately there is very little existing documentation regarding the available DeepRED implementation. The results presented in the original paper do not include specific results for the evaluation metrics I have specified in Section 4.1.2.

The current codebase for DeepRED is very large, poorly documented and relies on old implementations of libraries. In particular, the algorithm relies on the C4.5 DT induction algorithm described in Section 2.4.2. It was difficult to locate a freely available, trusted, working version of C4.5. As such it was not feasible to reimplement the original DeepRED algorithm.

Instead I have attempted to reimplement DeepRED with current, usable resources. I used the pseudocode from the original paper to mimic a version of DeepRED. I replace C4.5 with C5.0. This substitution will cause a huge improvement over the original DeepRED due to the inherent differences between C4.5 and C5.0 (see Section 2.6.1).

4.1.2 Evaluation Metrics

Evaluation metrics are defined to assess the quality of rulesets extracted in terms of their *accuracy*, *fidelity* and *comprehensibility*. The algorithmic complexity of each rule extraction algorithm is also measured. In total, six measurements are taken: accuracy, fidelity, number of rules, average rule length, rule extraction time and rule extraction memory usage.

The rules extracted from each ANN are evaluated on n observations, $x_1, x_2, ..., x_n$. Each observation has a true class label, y_i. The class predicted by the ANN for x_i is denoted $\hat{y_{n_i}}$. The extracted rules are applied to each observation in turn and yield a class prediction, $\hat{y_{r_i}}$.

Accuracy

The accuracy of the ruleset measures the extent to which the extracted ruleset can be used to classify new, unseen examples.

$$accuracy = \frac{|\{x_i | \hat{y_{r_i}} = \hat{y}_i \forall i \in [1, n]\}|}{n}$$

Fidelity

The fidelity of the ruleset quantifies how well it mimics the internal behaviour of the ANN from which it was extracted.

$$fidelity = \frac{|\{x_i | \hat{y_{r_i}} = \hat{y_{n_i}} \forall i \in [1, n]\}|}{n}$$

The measure is maximal when the ANN classifications and ruleset classifications are identical for all data instances.

Comprehensibility

The comprehensibility of a ruleset is measured with two metrics: the number of rules in the ruleset and the average length of a rule.

Dataset	Attributes	Examples	Network Topology	Network Accuracy (%)
Artif-1	5	30000	5-10-5-2	99.8
Artif-2	5	5000	5-10-5-2	100
BreastCancer	30	569	30-100-10-2	97.9
LetterRecognition-binary	16	1578	16-10-50-2	99.2
MNIST-binary	784	2703	784-100-50-2	99.45
MB-GE-ER	1000	1980	1000-10-50-2	96.1

Table 4.1: Overview of the datasets and neural networks used for evaluation

The number of terms in the antecedents of each rule are averaged to retrieve the average length of a rule. In general, the more compact the ruleset, the more human-comprehensible.

Complexity

The complexity of the algorithm is measured with two metrics: the time and memory usage. The time, measured in seconds (s), taken to extract rules from the trained ANNs is recorded. The memory usage, measured in megabytes (Mb), during rule extraction is recorded.

4.2 Hyper-Parameter Optimisation

This section describes the experiments undertaken during the data generation pipeline. Data, in the form of trained DNNs, is required as input for the rule extraction algorithms. The full data generation pipeline can be found in Section 3.4.

This section begins with an overview of the final trained DNNs from which rules are extracted. Table 4.1 describes the characteristics of the datasets and the final DNNs trained. The network 'topology' describes the number of neurons in each layer. The accuracy is computed as an average of the accuracies of the DNNs trained on each of the ten folds.

An analysis of the effects of the ANN initialisation of weights on the performance of the rule extraction algorithm is described below.

4.2.1 Neural Network Initialisation

At first, the initial weights of the DNNs were selected randomly from a uniform distribution. The effect of the initialisation on the performance of rule extraction can be seen in Figure 4.1.

Each graph illustrates the rule extraction performance of five DNNs trained on the same data. Each DNN had a different set of initial weights prior to training. Rules are then extracted from the trained DNNs using *RuleEx*.

It is evident from the graph that the initialisation of the DNN plays a big role. Given that the data fed to the DNNs is fixed, all of the variation comes from the initialisation of weights. In particular, for the *MB-GE-ER* data set, three initialisations yield very similar accuracies of 93.1%, 93.4% and 93.2%. Although two extract a similar number of rules, the third extracts almost 3000 more rules.

When the data is fixed, there is a large variation in the number of rules extracted. This motivated the structure of the data generation pipeline in which a single initialisation is imposed on all the DNNs trained on a dataset (see Section 3.4.2). The initialisation chosen is the one that yields the smallest ruleset. This initialisation is imposed for the 10-fold cross validation, where the only measure that changes is the data.

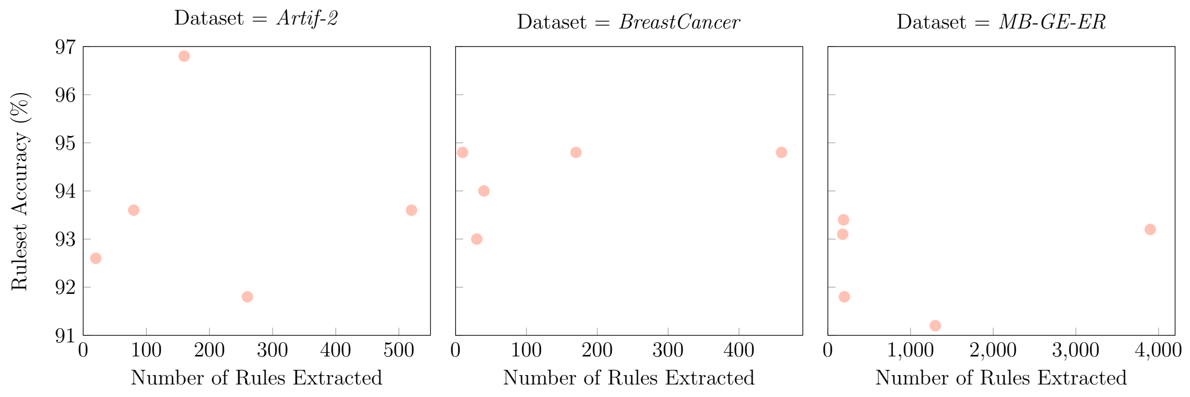


Figure 4.1: The effect of network initialisation on rule extraction of *Artif-2*, *BreastCancer* and *MB-GE-ER*

4.3 Rule Extraction Results

This section evaluates the performance of the rule extraction algorithm. It begins with an analysis of *RuleEx*. This is followed by a comparative evaluation of *RuleEx* with *DeepRED-C5* and *Ped-C5*.

A 10-fold cross-validated evaluation of rule extraction was carried out using *RuleEx*, *DeepRED-C5* and *Ped-C5* on each of the datasets. An overview of the results can be seen in Tables 4.2, 4.3 and 4.4, respectively.

4.3.1 Performance of RuleEx

The aim of this project was to implement a rule extraction algorithm to extract logical rules from DNNs for a variety of classification tasks. *RuleEx* could successfully extract rulesets for the selected datasets (Table 4.2).

The DNNs trained for classification all had high accuracies above 96%. The fidelity values for rules extracted using *RuleEx* are all above 92%. Fidelity is a measure of how well an extracted model can mimic the behaviour of the original model. The high fideltiy values indicate that behaviour of the extracted rulesets is very similar to the original

DNNs, as desired. In turn, the accuracy values of the extracted rulesets are also all above 92%.

There is a large variety in the number of rules extracted for the different classification tasks. Notably, the *MB-GE-ER* task generated an average of 2955.2 rules. On the other hand, the seemingly complex image classification task *MNIST-binary* only extracted an average of 11.3 rules. Although the number of rules can differ significantly between datasets, the average rule length is small and fairly consistent. This is a success of *RuleEx* as shorter rules are more comprehensible to humans.

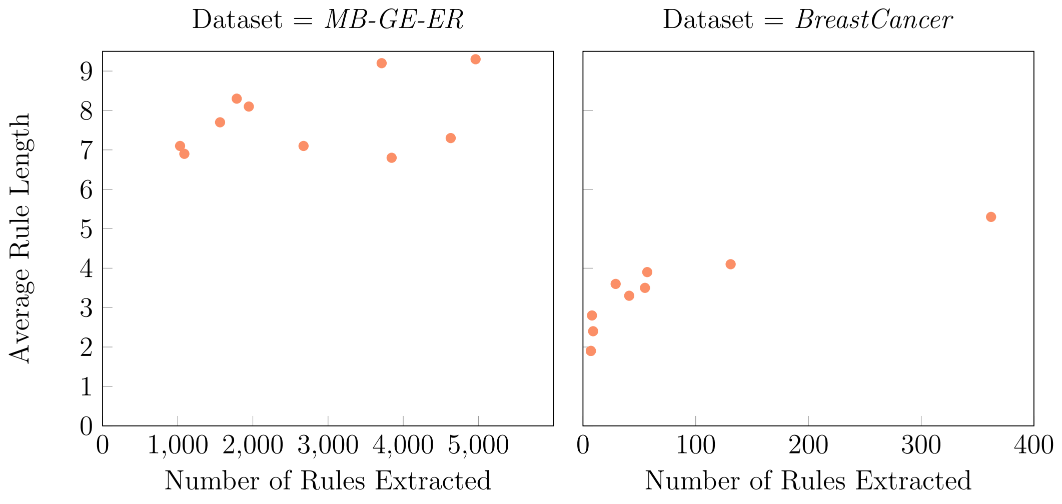


Figure 4.2: Results of *RuleEx* comprehensibility analysis on each fold of *MB-GE-ER* and *BreastCancer*

The variation in rule extraction performance between folds of the same dataset can be seen in Figure 4.2. It illustrates the measures of comprehensibility of the rulesets extracted from each of the 10 folds for the *MB-GE-ER* and *BreastCancer* classification tasks. The *MB-GE-ER* graph shows a large variation in the number of rules extracted from each data fold. Despite this, the rule length remains consistent between 6.9 and 9.5 terms per rule, on average. The *BreastCancer* dataset also displays a similar consistency in the average rule length.

RuleEx could successfully extract rules from the DNNs with feasible time and memory usage. It can be seen that larger rulesets required more time and memory for extraction. The *MB-GE-ER* dataset extracted an average of 2955.2 rules and took an average of 148.6s for each fold. *MB-GE-ER* also had the largest memory usage for *RuleEx* as 112.4Mb. It should be noted that the larger the size of the intermediate rulesets extracted from each pair of adjacent layers, the larger the memory consumption of the algorithm. This is due to the rule subsitution step, in which the intermediate rulesets are recursively substituted and merged into a final ruleset (see Section 3.2.1).

4.3.2 Comparative Evaluation of Rule Extraction Algorithms

This section describes a comparative evaluation of *RuleEx* with alternative rule extraction algorithms. Firstly, I begin with an overview of the performance of the existing DeepRED

Dataset	Number of Rules Extracted	Average Rule Length	Accuracy (%)	Fidelity (%)	Time (s)	Memory (Mb)
Artif-1	112.3	3.1	94.9	94.9	28.3	86.4
Artif-2	23.1	6.2	93.2	97.3	4.2	12.5
BreastCancer	184.2	5.2	93.9	94.2	3.5	7.8
LetterRecognition-binary	83.1	12.2	96.6	97.1	34.3	44.3
MNIST-binary	11.3	7.2	98.2	99.1	95.6	22.5
MB-GE-ER	2955.2	7.8	92.8	93.6	148.6	112.4

Table 4.2: 10-fold cross-validation results of the *RuleEx* algorithm

Dataset	Number of Rules Extracted	Average Rule Length	Accuracy (%)	Fidelity (%)	Time (s)	Memory (Mb)
Artif-1	110.2	2.9	94.9	94.9	34.3	91.2
Artif-2	21.1	6.2	93.2	97.3	9.4	19.4
BreastCancer	201.2	4.3	94.6	95.1	6.2	10.6
LetterRecognition-binary	77.5	12.3	96.6	97.1	45.2	55.4
MNIST-binary	10.3	6.1	98.2	99.2	112.3	60.5
MB-GE-ER	3124.3	7.1	95.4	93.2	321.4	140.3

Table 4.3: 10-fold cross-validation results of the *DeepRED-C5* algorithm

Dataset	Number of Rules Extracted	Average Rule Length	Accuracy (%)	Fidelity (%)	Time (s)	Memory (Mb)
Artif-1	45.3	2.5	95.3	96.6	7.6	37.8
Artif-2	10.2	5.4	92.1	93.4	4.1	12.3
BreastCancer	201.2	4.3	94.1	95.3	3.2	10.6
LetterRecognition-binary	32.1	11.2	95.4	96.4	6.5	17.9
MNIST-binary	4.5	5.3	97.9	98.2	17.4	21.7
MB-GE-ER	34.2	4.5	95.1	96.3	13.4	32.3

Table 4.4: 10-fold cross-validation results of the *Ped-C5* algorithm

algorithm. As mentioned, it was not possible to reimplement the original DeepRED algorithm. The results reported by the authors are described below [30].

Among other datasets, the authors of DeepRED perform rule extraction on *Artif-1*, *Artif-2*, *LetterRecognition-binary* and *MNIST-binary*. Each iteration of DeepRED is run with 10,000Mb of allocated RAM and a time limit of 24 hours for its execution. An experiment is said to 'abort' if it exceeds these time or memory limits. The authors of DeepRED report a high abortion rate of $\approx 46\%$ of the experiments. It should be noted that a large proportion of the aborted experiments exceeded memory limits while merging. A large number of intermediate rules generated by the deeper layers can lead to this. *RuleEx* can successfully and consistently extract rules for the *Artif-1*, *Artif-2*, *LetterRecognition-binary* and *MNIST-binary* datasets with reasonable time and memory usage.

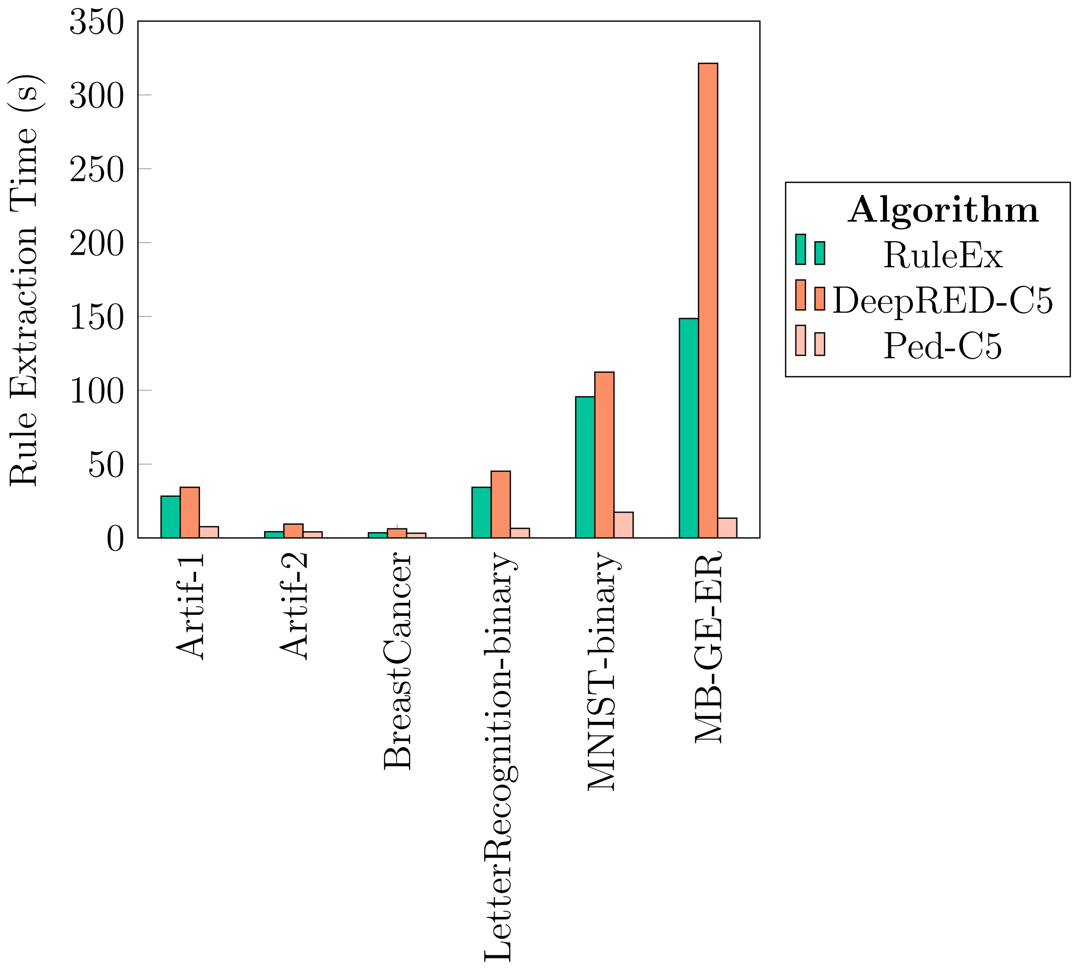


Figure 4.3: A comparison of the time usage of the various rule extraction algorithms

DeepRED-C5 is my own implementation of DeepRED that substitutes C4.5 with C5.0. *DeepRED-C5* can also successfully extract rulesets with high accuracies for all of the datasets mentioned. Surprisingly, *Ped-C5* can also extract rules with a high accuracy and fidelity from all of the datasets. *Ped-C5* is a pedagogical rule extraction algorithm that does not use any information about the internal structure of the ANN to extract

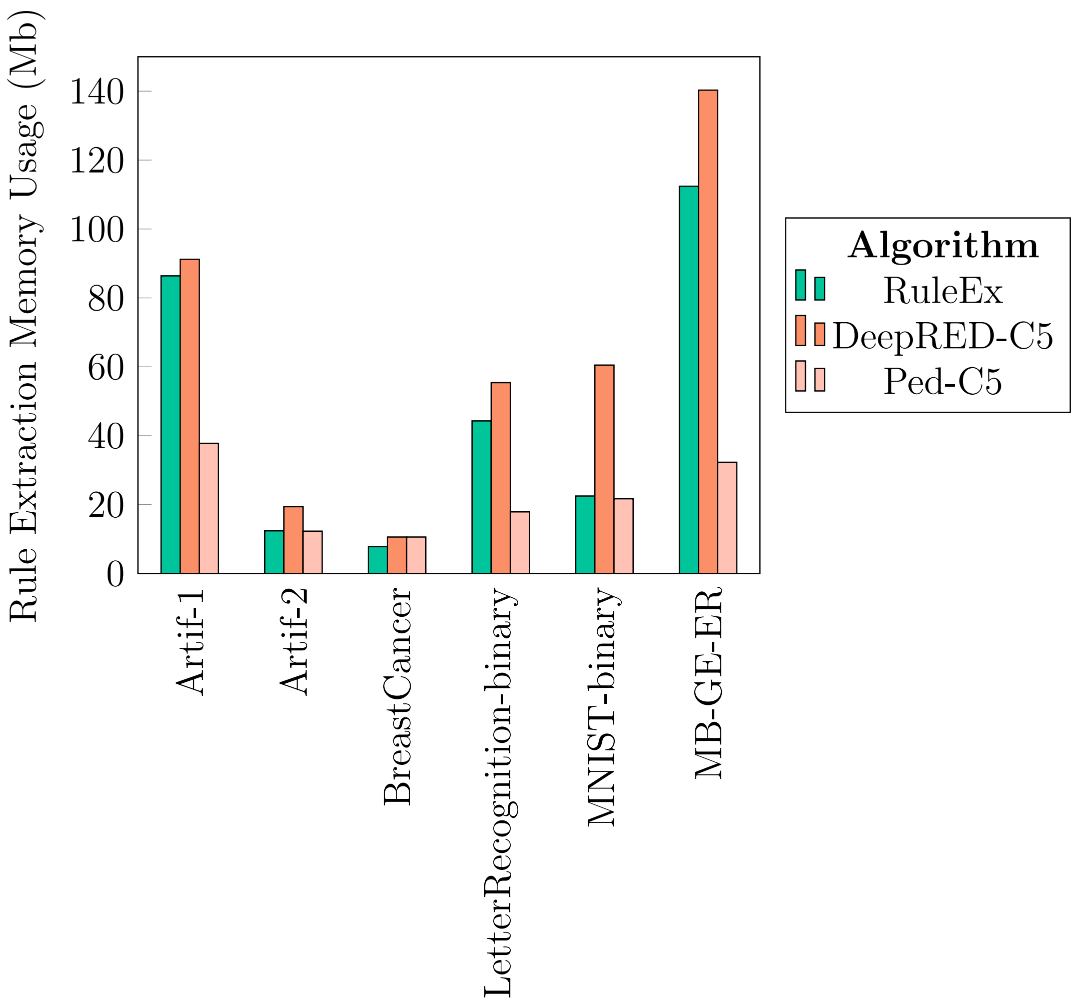


Figure 4.4: A comparison of the memory usage of the various rule extraction algorithms

rules.

Figures 4.3 and 4.4 illustrate the average time and memory usage of the algorithms, respectively. As expected, *Ped-C5* consistently has the lowest time and memory usage for each dataset. Qualitatively, the rules extracted using *Ped-C5* can be very different as the pedagogical algorithm does not use any information about the internal structure of the network. One way of verifying that would be to compare the features that appear in rules extracted decompositionally and pedagogically. Due to time constraints, this is left for future work. The pedagogical algorithm also does not perform any rule substitution or merging steps. The ANN input data and predictions are passed into the C5.0 algorithm and the rules extracted are returned.

In most cases *RuleEx* outperforms *DeepRED-C5* in both time and memory usage. A rule extraction algorithm may be used as an online approximator for an ANN. The extracted model could be used to explain the results of the network to a user. Given these circumstances, the time and memory usage of the algorithm are important factors. The *RuleEx* algorithm contains specific optimisations to reduce memory usage. A full description of the optimisations used can be found in Section 3.2.3.

4.4 Rule Ranking Results

This section describes the rule ranking procedure. This is applied to the rules extracted using RuleEx. The motivation behind rule extraction algorithm lies in extracting an accurate, comprehensible model from an existing high-performing opaque model.

The extracted rulesets can be shrunk to increase comprehensibility. Smaller models are easier for humans to interpret. However, the deletion of rules is known to come with a cost in terms of the accuracy. The trade-off between accuracy and comprehensibility is explored here.

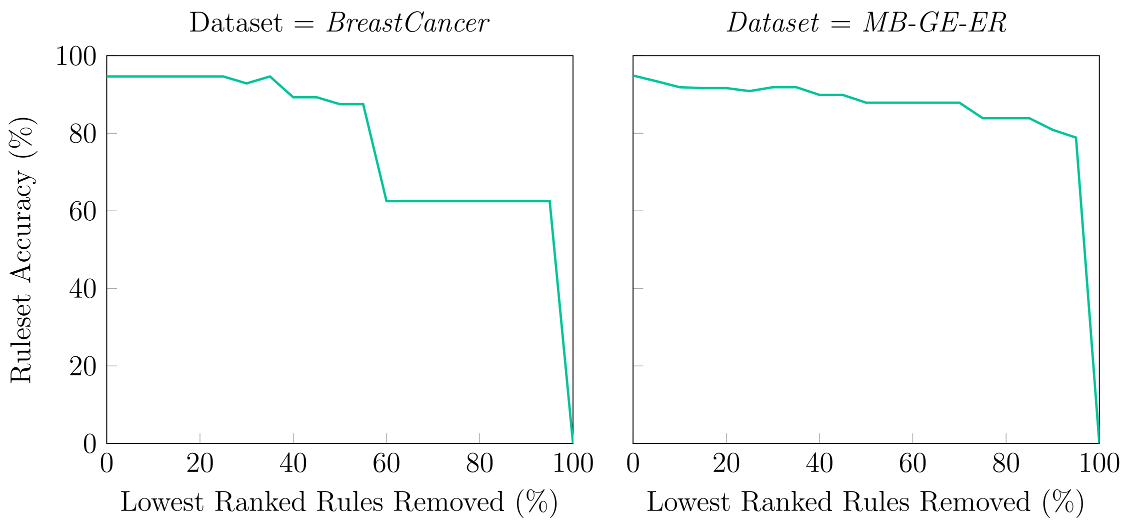


Figure 4.5: Rule ranking procedure applied to *BreastCancer* and *MB-GE-ER*

Each rule in the ruleset is assigned a *rankScore*. Details about how this is computed can be found in Section 3.3. The rules in the ruleset are then ordered according to their *rankScore*. This can be used as the foundation to decide which rules to eliminated to shrink the ruleset. The lowest ranked rules are eliminated.

For this analysis 5% of the rules with the lowest scores are iteratively deleted and the remaining ruleset is the applied to the data. The accuracy of the smaller rulesets can be seen in Figure 4.5.

The accuracy of the *BreastCancer* dataset decreases slowly at first from 94.6% to 87.5% after 55% of the lost ranked rules are removed. There is a sharp decrease in accuracy as the next portion of rules are eliminated, after which the accuracy remains stable. On the other hand, the accuracy of the *MB-GE-ER* ruleset decreases at a steady rate throughout. The accuracy of the full ruleset is 94.9% and the accuracy of the top 5% of the ruleset is 78.8%. Surprisingly, removing the bottom 50% of the rules decreases the accuracy by around 4%.

This procedure illustrates that when the size of the extracted ruleset is a problem, removing a large percentage of the lowest ranked rules can have little to no affect on the accuracy of the ruleset. The smaller ruleset is now however much more comprehensible for humans.

Chapter 5

Conclusion

In this dissertation I presented an optimised decompositional rule extraction algorithm that can be applied to DNNs to translate the knowledge embedded in the opaque model into a set of comprehensible logical rules.

5.1 Summary of Results

In this report I have successfully implemented an algorithm to extract logical rules from trained DNNs. Suitable methods were described to train the DNNs over a variety of classification tasks. I evaluate the algorithm alongside two alternative approaches.

I also discuss the implementation of a procedure to rank the extracted rules. Interestingly, this demonstrated the trade-off between accuracy and comprehensibility as rules were iteratively deleted from the ruleset. If the size of the ruleset is a problem, this rule ranking procedure provides an appropriate solution.

5.2 Success Criteria

This section illustrates how each of the success criterion, described in the Project Proposal, was met.

All of the other core elements of the project are achieved. Extension 1 involved the implementation of a pruning algorithm on the DNN prior to rule extraction. It was decided that this was no longer needed as RuleEx successfully extracted a small enough ruleset from the original DNNs. The work undertaken to achieve the core success criteria is described below.

- *Building and evaluating a DNN classifying the gene expressions of cancer patients* - The METABRIC dataset was successfully used to build, train and evaluate DNNs to classify the biomedical data from cancer patients. This process is described in Section 3.4.
- *Using the C5.0 algorithm to extract DTs from the DNN and extracting rules from*

the DTs generated - I found a library implementing C5.0 that generated the DTs and extracted the rules. Significant engineering effort was required to utilise this implementation as mentioned in Section 2.6.1.

- *Implementing a recursive algorithm to return a rule set describing the output of the DNN in terms of the input* - The full pseudocode and details of the rule extraction algorithm implemented can be found in Section 3.2.1.
- *Implementing a perturbational pruning step on the input layer of the DNN and a rule shortening step on the generated output* - As mentioned, the perturbational pruning step was no longer required. The C5.0 algorithm itself performs the rule pruning mentioned (see Section 2.6.1).
- *Implementing a method using hill climbing heuristics to prune the rules generated from the model* - The rule ranking mechanism used is described in Section 3.3 and evaluated in Section 4.4.
- *A comparative evaluation of the rule extraction algorithm* - RuleEx is evaluated against two alternative systems described in Section 4.3.

5.3 Lessons Learnt

This dissertation is by far the largest and longest project I have undertaken thus far. It has also been one of the most rewarding. Learning how to translate all of my ideas into one cohesive piece of prose was challenging but ultimately beneficial to any career path I choose.

Narrowing down the scope of the project early saved time in the long run. The work could easily be split into chunks that were delivered and discussed weekly during meetings with my supervisors. I found scheduling this time each week was incredibly useful as any and all issues were discovered and discussed immediately.

In the initial timeline of my Project Proposal I scheduled in ‘slack time’ slots every few weeks. No tasks were scheduled here and the time was available if, for any reason, prior tasks took longer than expected. Even with a detailed plan, all of these time slots were fully utilised.

Due to personal circumstances, I took a long break in between the implementation phase and writing the dissertation. This taught me the importance of keeping thorough clear records as my previous notes and documentation was now invaluable. Writing clear, modular, readable code meant that it took a fraction of the time to return to work. I now understand why good software engineering practices are imperative to to all projects, regardless of team size.

5.4 Future Research

During this project I discovered that the conditions under which a DNN was trained significantly impacted the performance of subsequent rule extraction. Time constraints meant that I could not explore this further and instead the data generation pipeline was altered to accommodate for this (see Section 3.4).

It was the case that particular training configurations and ANN initialisations lead to more 'explainable' models; rule extraction from these models was easier and the extracted rulesets were smaller. Further research into this area would investigate and potentially explain this. If achieved, this could lead to the development of a method to systematically train an ANN to be more interpretable.

Bibliography

[1] Intro to tensorflow for deep learning. https://www.udacity.com/course/intro-to-tensorflow-for-deep-learning–ud187. Accessed: 24-10-2019.

[2] Robert Andrews, Joachim Diederich, and Alan Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 6:373–389, 12 1995.

[3] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373 – 389, 1995. Knowledge-based neural networks.

[4] José Manuel Benítez, Juan Luis Castro, and Ignacio Requena. Are artificial neural networks black boxes? *IEEE transactions on neural networks*, 8 5:1156–64, 1997.

[5] Pieter-Tjerk Boer, Dirk Kroese, Shie Mannor, and Reuven Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134:19–67, 02 2005.

[6] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. 02 2015.

[7] Mark W. Craven and J. W. Shavlik. Extracting comprehensible models from trained neural networks. 1996.

[8] Mark W. Craven and Jude W. Shavlik. Using neural networks for data mining. *Future Generation Computer Systems*, 13(2):211 – 229, 1997. Data Mining.

[9] Christina Curtis, Sohrab P Shah, Suet-Feung Chin, Gulisa Turashvili, Oscar M Rueda, Mark J Dunning, Doug Speed, Andy G Lynch, Shamith Samarajiwa, Yinyin Yuan, and et al. The genomic and transcriptomic architecture of 2,000 breast tumours reveals novel subgroups, Apr 2012.

[10] Aparicio Lab Research Department. Metabric dataset.

[11] Heather Duncan, James Hutchison, and Christopher S. Parshuram. The pediatric early warning system score: A severity of illness score to predict urgent medical need in hospitalized children. *Journal of Critical Care*, 21(3):271 – 278, 2006.

[12] Jerome Friedman and Nicholas Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9, 04 1999.

[13] Stephan I Gallant. Connectionist expert systems. *Communications of the ACM*,

31(2):152–170, 1988.

[14] M.W Gardner and S.R Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14):2627 – 2636, 1998.

[15] Samuel H. Huang and Hao Xing. Extract intelligible and concise fuzzy rules from neural networks. *Fuzzy Sets and Systems*, 132(2):233 – 243, 2002.

[16] Ulf Johansson, Tuve Löfström, Rikard König, Cecilia Sönströd, and Lars Niklasson. Rule extraction from opaque models– a slightly different perspective. pages 22–27, 12 2006.

[17] Sebastian Lapuschkin, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. The lrp toolbox for artificial neural networks. *Journal of Machine Learning Research*, 17(114):1–5, 2016.

[18] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[19] Morteza Mashayekhi and Robin Gras. Rule extraction from random forest: the rf+hc methods. 06 2015.

[20] A. Mukherjee, R. Russell, Suet-Feung Chin, and B. Liu. Associations between genomic stratification of breast cancer and centrally reviewed tumour pathology in the metabric cohort. *npj Breast Cancer*, 4(1):5, 2018.

[21] Tamara Müller and Pietro Lio'. Peclides neuro a personalisable clinical decision support system for neurological diseases. *bioRxiv*, 2019.

[22] Tamara Müller and Pietro Lio. Peclides neuro: A personalisable clinical decision support system for neurological diseases. *Frontiers in Artificial Intelligence*, 3, 04 2020.

[23] David R. Prytherch, Gary B. Smith, Paul E. Schmidt, and Peter I. Featherstone. Views—towards a national early warning score for detecting adult inpatient deterioration. *Resuscitation*, 81(8):932 – 937, 2010.

[24] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[25] Ross Quinlan. Is see5/c5.0 better than c4.5?

[26] Makoto Sato and Hiroshi Tsukimoto. Rule extraction from neural networks via decision tree induction. volume 3, pages 1870 – 1875 vol.3, 02 2001.

[27] Hai Shu and Hongtu Zhu. Sensitivity analysis of deep neural networks, 2019.

[28] Effy Vayena, Alessandro Blasimme, and I. Glenn Cohen. Machine learning in medicine: Addressing ethical challenges. *PLOS Medicine*, 15(11):1–4, 11 2018.

[29] Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR): A Practical Guide.* Springer Publishing Company, Incorporated, 1st edition, 2017.

[30] Jan Zilke, Eneldo Mencía, and Frederik Janssen. Deepred – rule extraction from deep neural networks. pages 457–473, 10 2016.

Appendix A

Project Proposal

Introduction and Description of the Work

Although Deep Neural Networks (DNN) consistently attain high classification accuracy, the lack of transparency reduces their usability in domains prioritising comprehensibility. This project aims to remedy this by translating the embedded internal knowledge of the complex model into a set of comprehensible symbolic rules.

Recent developments in DNN have allowed us to progress rapidly in areas such as computer vision, natural language processing and speech recognition. Despite their ability to outperform traditional systems in these tasks, NN fail to provide an accurate and comprehensible explanation of what has been learnt. Transparency is needed from these black-box systems if they are to be trusted. Furthermore, when a decision is made by an autonomous system on behalf of a person, it is essential that clarification is given, as evidenced by recent regulations such as the European Union's "Right to Explanation". However, the knowledge acquired by a NN during training is embedded within its architecture and parameters which are, at a first glance, incomprehensible. Methods to mitigate the incomprehensibility date back to the work of Gallant [13] in the 1990s and include Feature Importance [17] and Sensitivity Analysis [27] techniques. This project focuses on rule extraction (RE) approaches.

Rule based decisions systems, such as Decision Trees and IF-THEN rules, tackle classification problems in a manner similar to humans, making them more understandable. Methods of RE can be characterised according to their translucency using a taxonomy offered by Andrews et al [2]. Decompositional algorithms consider all hidden nodes and weights in a network whilst pedagogical methods view the NN as a black box, learning a function mapping the input features to the correct class. An eclectic approach considers elements of both.

Although the process of decompositional RE is more tedious, pedagogical approaches are less likely to accurately capture all the valid rules describing the inner behaviour of the network. Before Zilke [30], most RE approaches could not handle the complexity of a NN with more than one hidden layer. He proposed DeepRED, a decompositional algorithm that extracts layer-wise rules from a DNN that are merged in a final step. The work is an extension of CRED [26], using the well-known C4.5 [24] algorithm to extract decision trees from the network.

The aim of this project is to implement a decompositional RE algorithm for DNN that perform binary classification tasks on a subset of the gene expression data available from METABRIC; Biological relevance of which has been reported [9]. The biomedical data from cancer patients can be labelled oestrogen receptor (ER)-positive/negative or Distant-Relapse (DR)-positive/negative. Transparency in these models would prove useful to oncologists looking to justify the decisions made. The algorithm will be evaluated in terms of the accuracy and comprehensibility of the rules generated.

Resources Required

I plan to use my personal laptop to develop this project: Dell XPS - 8th gen Intel Core i7, 16GB Ram, 500GB SSD, running Windows. If I experience a failure with my laptop, I will use the MCS computers in the Computer Laboratory. I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.

I will use git for revision control, with my git repository hosted on GitHub. This will serve as a cloud backup of my code. I will also store additional resources, as well as a local copy of my code, in a Dropbox folder that will sync automatically. Additionally, I will perform weekly backups to an external hard disk in case these other backup techniques fail.

The dataset used for this project is publicly available from METABRIC [10]. I will keep a copy in a Dropbox folder and on my local machine in case the availability of this external dataset is unexpectedly compromised.

Starting Point

The project is based on the DeepRED algorithm performing RE on DNN. I had no knowledge of RE techniques before beginning the research for this project. Since then I have read a number of academic papers in the area of Explainable Artificial Intelligence and existing algorithmic approaches. The part I Logic and Proof and the part II Information Theory course will be useful when reducing the rule set generated, improving comprehensibility but maintaining the accuracy of the network.

Prior to this project I had limited practical experience with Machine Learning. I have some programming knowledge from the CST Tripos, including Java, with little experience in Python. Hence, I intend to spend some time in the initial stages building my experience in both areas using material from large open online courses [1], research papers and existing open-source implementations.

Substance and Structure of the Project

The purpose of this project is to implement a rule extraction algorithm on a pre-trained DNN that generates human comprehensible rules. A network needs to be selected and trained until it meets accuracy requirement and is to be used as input to the RE algorithm.

Decompositional rule extraction is a stepwise process:

1. The updated C5.0 [25] algorithm, based on C4.5, will be used to generate a decision tree. The split points will come from the activation values of the last hidden layer's neurons with the leaves representing the resulting classification. It uses a heuristic function based on the concept of information entropy;
2. Logical rules can be extracted from these decision trees;
3. The process is repeated using C5.0 on each hidden layer in the network working backwards. This results in intermediate rule sets that describe each layer of the DNN in terms of the outputs of the previous layer;
4. The rule sets are merged using substitution layer-wise until a rule set is formed describing the behaviour of the network output in terms of features in the input;

The original work of DeepRED includes a perturbational approach to neuron pruning in which individual features, unnecessary to produce an acceptable classification, are ignored. This approach, as well as the rule shortening mentioned, will be implemented here.

An alternative method using a hill climbing approach [21] to prune the rules generated in the network will also be implemented.

The core elements of this project are split into phases below:

Research: The research phase consists of preparatory background reading into existing rule extraction methods with a specific focus on taking elements of existing algorithms, for single layer networks and extending them to apply to deeper models.

Preparation: To assess the success of a rule extraction algorithm, input data in the form of a trained neural network is needed. This will be produced in this phase. Relevant prepatory courses will be covered here.

Implementation: The bulk of the implementation will focus on the rule extraction algorithm. A baseline method will be produced first, generating the decision trees and intermediate rules from the unpruned network. A neuron pruning and rule shortening step will be added before rule extraction. The performance of the algorithm will then be evaluated according to metrics defined below.

Evaluation: A comparative analysis of the rules extracted from the algorithm before and after pruning will be performed according to the following metrics:

Accuracy: Describes whether the extracted rules can make correct predictions for previously unseen test cases

Fidelity: Related to accuracy and measures how closely the predictions made by the rules and DNN agree i.e. how closely do the rules mimic the model

Comprehensibility: Quality of the rules extracted is measured; the number of rules and the average number of terms can be used as indicators for this

Success Criteria

- Building and evaluating a deep neural network classifying the gene expressions of cancer patients into relevant biological categories. The dataset will be split into a training and test set, with the test set used for evaluation of accuracy
- Using the C5.0 algorithm to extract decision trees from the output layer and extracting logical rules from the decision trees generated
- Implementing a recursive algorithm that performs decision tree induction and rule generation on the intermediate layers of the network
- Merging the generated rule sets into a rule set describing the output of the network in terms of the input
- Implementing a perturbational pruning step on the input layer of the network and a rule shortening step on the generated output
- Implementing a method using the hill climbing heuristics to prune the rules generated from the model
- A comparative evaluation of the baseline RE method on the trained DNN, with and without different pruning algorithms, based on the metrics defined

Possible Extensions

If I meet my core success criteria with time to spare, there are a number of interesting paths I could explore further:

1. **An alternative network pruning approach:** The existing perturbational approach used in DeepRED could be compared with a gradient based method of pruning
2. **Evaluating the consistency of the algorithm:** An algorithm is considered consistent if it extracts similar rules under different training instances. A cross-validation method could be used, as well as a metric to compare rule sets with each other
3. **Producing a visualisation of the decision tree generated:** A visual representation of the extracted model, in the form of a decision tree, may prove to be useful to the clinicians making the diagnoses to clearly see how the DNN came to a conclusion

Timetable and Milestones

The intention is to finalise the core project implementation and evaluation by the end of December. Lastly, the dissertation should be finished at least a week before the deadline. In order to track progress, I will work in two-week sprints with a clearly stated milestone at the end of each period. Slots marked slack time are intentionally empty and to be used as buffer slots to accommodate any unexpected delays

28.10.19 - 10.11.19 [Michaelmas term]

Learn and gain more practical experience of Machine Learning with Python. Continue reading relevant research papers. Setting up backup arrangements as described in the Resources Required section.

Milestone: Familiarity with TensorFlow and biomedical data. Begin building initial DNN. Introduction drafted.

11.11.19 - 24.11.19 [Michaelmas term]

Train and test the implementation of the network on data given. Explore the C5.0 algorithm and begin its implementation

Milestone: A trained NN that can be used as input to the rule extraction algorithms. A thorough understanding of how the C5.0 algorithm works. Preparation drafted.

25.11.19 - 08.12.19 [Michaelmas term/Christmas vacation]

Finish implementation of C5.0. Implementation of method to extract rules from the decision trees

Milestone: Rules extracted explaining the behaviour of the output in terms of the values found in the last hidden layer of neurons

09.12.19 - 22.12.19 [Christmas vacation]

Write algorithm to merge all the intermediate rules. Implement initial pruning algorithm on the network prior to rule extraction.

Milestone: Ruleset generated that describes the behaviour of the pruned/unpruned network output in terms of only the input features

23.12.19 - 05.01.20 [Christmas vacation]

Implement additional rule pruning approach using the hill climbing heuristics. Evaluate core elements of the project. Write a draft of the progress report for supervisor.

Milestone: Second pruning algorithm implemented. First progress report draft complete

06.01.20 - 19.01.20 [Christmas vacation/Lent term]

Slack time

Milestone: All core success criteria met, begin Implementation draft

20.01.20 - 02.02.20 [Lent term]

Finalise and submit the progress report. Continue Implementation draft. Begin preparatory reading for extensions.

Milestone: Submit progress report

03.02.20 - 16.02.20 [Lent term]

Begin work on extensions. Prepare to give presentation

Milestone: Progress report presentation

17.02.20 - 01.03.20 [Lent term]

Continue work on extensions.

02.03.20 - 15.03.20 [Lent term/Easter vacation]

Finish any extensions

Milestone: Finished implementation of extensions

16.03.20 - 29.03.20 [Easter vacation]

Draft remaining chapters of write up

Milestone: Evaluation and Conclusion drafted

30.03.20 - 12.04.20 [Easter vacation]

Complete first draft of dissertation. Complete any unfinished tasks.

Milestone: Hand in draft dissertation

13.04.20 - 26.04.20 [Easter vacation/Easter term]

Improve dissertation incorporating changes based on supervisor feedback

Milestone: Second dissertation draft completed

27.04.20 - 08.05.20 [Easter term]

Proofread dissertation and finalise any changes. Hand in final copy by start of May.

Milestone: Dissertation completed