Universiteit Leiden

Master Computer Science

R2N-Tab: a hybrid model for binary tabular data

Name:	Michael van der Zwart
Student ID:	2299925
Date:	16-01-2024
Specialisation:	Artificial Intelligence
Daily supervisor:	Qi Huang
1st supervisor:	Matthijs van Leeuwen
2nd supervisor:	Niki van Stein

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this thesis project, we propose an extension to DR-Net, which is an approach for learning rule sets from neural networks applicable to binary classification tasks. We reuse its two-layer architecture but apply a deep learning layer for feature selection in front of it. The layer learns to label features that do not affect the predictive performance of the classifier with a negative weight, whereafter they are deactivated and not considered in the rule set. DR-Net provides better accuracy-simplicity trade-offs compared to state-of-the-art rule learners. However, a large number of epochs is required to obtain small rule sets. In this thesis project, we show that our method R2N-Tab, which combines DR-Net and the feature selection layer, constructs sparser rule sets with fewer epochs than the original DR-Net. Still, the constructed rule sets are accurate and have comparable predictive performance. We also compare our method to well-known feature selection and rule learning approaches. Overall, we show that our method finds a better fit for the combination of predictive performance and model complexity on large datasets.

Contents

Chapter 1

Introduction

Deep learning is a promising subfield of machine learning that has been applied to various domains for solving complex problems [28]. With its ability to automatically learn representations of data, this technique has shown breakthrough performance in, e.g., natural language processing (NLP) [22], computer vision [31], health care [29] and autonomous driving [13]. Deep learning proves effective when dealing with large datasets, allowing to learn complex patterns that are impossible for humans to discover. As a result, deep learning has the potential to transform many aspects of modern life and drive major advances in, e.g., science, medicine, and technology. These are just a few examples of the many fields in which deep learning has been successfully applied, and is still making rapid progress.

However, where the application of deep learning in various fields has been very promising, it is lagging performance in the domain of tabular data, as raised by Grinsztajn, Oyallon, and Varoquaux [14]. In the paper a benchmark for tabular data is presented, evaluating different deep learning approaches. Their research focuses on why tree-based models still outperform deep learning models in the domain of tabular data. One interesting observation they present is the appearance of a multitude of uninformative features within tabular datasets. In one experiment, uninformative features were removed from the data, and different models were evaluated, which tended to reduce the performance gap between tree-based models and deep learning models. Thus, deep learning models are less robust to uninformative features than tree-based models, which harms their performance. As a result, feature selection is required to remove irrelevant features. Interesting use cases of tabular datasets include sports statistics, financial data, or patient records. As the data grows proportionally to the number of features, the data is likely to contain more uninformative features. This affects the predictive performance of deep learning models, which impedes the application of artificial neural networks in those fields.

Another limitation of deep learning models is that they are mostly hard to interpret, which is known as the black-box problem [35]. This complicates the application of deep learning in more serious domains, where decisions impact patient health or other important concerns, and output generated by deep learning models that can not be interpreted by humans would be useless. Qiao et al. proposed a method called DR-Net (Decision Rules Network) where rules can be derived from a neural network architecture, describing the decisions it makes to increase interpretability [24]. DR-Net consists of neurons that map to logical IF-THEN rules, which describe how the model formulates its output.

In this work, we propose an extension to DR-Net. As described above, [14] mentions the problem of deep learning struggling to achieve breakthrough results in the domain of tabular data. In our work, one challenge is addressed in particular: the problem of deep learning models not being robust to uninformative features. Over the past years, feature selection methods have shown promising results [18]. In DR-Net, some feature masking is used, training mask variables that remove features from rule clauses. This should result in a minimal performance decrease, depending on a trade-off between model performance and rule set sparsity controlled by a λ constraint. However, a large number of epochs is required to achieve a sparse rule set with good predictive performance. The masking strategy from DR-Net is explained in detail in Chapter 3. This work contributes by adding a feature extraction component to DR-Net, which is fully focused on identifying uninformative features and removing them from the rule set. Note that since we make an addition to DR-Net, the structure of this network itself remains unchanged. The cancel layer that we add in front should result in more sparsity in the rule set, while the predictive performance is minimally harmed. Informative features that are not canceled out by our feature extraction component are considered to contribute to the predictive performance of the model. However, this does not necessarily mean that these features need to be present in all rules. This is why the masking strategy from DR-Net is still useful to achieve more sparsity.

1.1 Research question

Following the challenges presented earlier, we can summarize our research into one main research question. The deep learning layer should indicate uninformative features earlier, contributing to DR-Net by achieving more sparsity with less budget. Besides, we want to obtain a rule set with good predictive performance. This can be formulated in the following question:

> How can we learn a deep learning model for rule learning that best fits a trade-off between predictive performance and model complexity with as few epochs as possible?

We extend DR-Net by adding a deep learning layer in front, such that the feature set is diluted to retain only the relevant features, that contribute to the predictive performance of the model. The formulated research question includes figuring out how we indicate irrelevant features on the fly, and how we remove them from the rule set. The second part of the research question includes the fact that we want to achieve a model that is a good classifier and has decent predictive performance compared to other feature selection and rule learning approaches. Besides, we want to build an interpretable classifier, describing its output with rules. As sparser rule sets are easier to interpret, we have to deal with a trade-off, maximizing the predictive performance of the model while reducing the model complexity. This research question summarizes our main goal in this thesis project, improving on DR-Net and performing similarly compared to other related approaches introduced later on. In Chapter 4, our model is introduced, which should provide a solution to the described challenges.

1.2 Overview

The structure of this thesis project is as follows: in Chapter 2, some interesting related works are introduced regarding rule learning and feature selection, which can be useful for us to later compare our method against; in Chapter 3, detailed background information of concepts used in the thesis is provided; in Chapter 4, our method R2N-Tab is proposed and in-depth descriptions of its architecture are given; consequently, in Chapter 5 our method is compared against the other related methods, and reproducible experiments and results are discussed; finally, Chapter 6 concludes the thesis project and provides ideas about possible future work.

Chapter 2

Related work

In this chapter, we provide an overview of the related research area. We discuss several approaches to rule learning and feature selection that are interesting to compare our model against and highlight our contribution.

2.1 Rule learning

2.1.1 Rule lists

Rule lists consist of a series of IF-THEN-ELSE statements and they have been fundamental in interpretable classification tasks for decades. Some of the best-known classic rule learning algorithms include CART [5], ordered-CN2 [8], RIPPER [9] and C4.5 [25]. RIPPER (Repeated Incremental Pruning to Produce Error Reduction) constructs rules using a separate-and-conquer strategy, creating a rule that covers a subset of examples and then removing those from the training set. It employs a combination of heuristic pruning and error reduction techniques to generate a compact set of rules. Where RIPPER starts with a single rule that includes all instances, CN2 begins with an empty rule and incrementally adds conditions based on information gain. CART (Classification And Regression Trees) and C4.5 are both decision tree algorithms, allowing for the derivation of a rule list afterward based on the splits at each node. They differ in their splitting criteria: CART uses Gini impurity, while C4.5 uses information gain. All these classic rule learning algorithms lack a global optimization criterion and provide non-probabilistic rule lists.

More recent work includes methods that offer probabilistic rules, such as CLASSY [23] and Scalable Bayesian Rule Lists (SBRL) [33]. Classy is a greedy algorithm based on the minimum description length (MDL) principle, designed to find small and probabilistic rule lists that outperform state-of-the-art classifiers in terms of predictive performance and interpretability, including SBRL. SBRL utilizes Bayesian techniques and probabilistic reasoning to create rule lists that not only describe predictions but also express the associated uncertainty. As CLASSY and SBRL both utilize models based on Bayesian statistics, we exclusively incorporate the superior CLASSY into our experiments. It is worth noting that the aforementioned methods are all designed for multiclass classification, while R2N-Tab is a binary classifier. Although R2N-Tab can be extended to handle multiclass classification, our primary focus in this research is binary classification.

2.1.2 Rule sets

While rule lists have been commonly used, some research papers argue that they are difficult to interpret due to imposed orders among the rules they provide. The activation of a specific rule requires the understanding of all preceding rules. In rule sets, one chain of rules is constructed, separated by logical OR operations. Several rule learning methods construct rule sets instead of rule lists to increase interpretability. Binary methods that learn rules for a single class label include Column Generation (CG) [10] and Bayesian Rule Sets (BRS) [32]. Other methods learn a set of rules for each class against all other classes, including non-probabilistic separate-and-conquer methods unordered-CN2 [7] and FURIA [15]. As a result, these methods are capable of performing multiclass classification, albeit indirectly. Unordered-CN2 is an extension of the original CN2 algorithm, which shows how unordered rules as well as ordered rules can be generated. FURIA is a rule-based classifier that learns fuzzy rules instead of conventional rules such that decision boundaries can be modeled in a more flexible way. As a result, FURIA can not provide a single rule to describe a decision, which complicates direct comparison to our method.

For directly learning rules for multiclass targets, only two methods existed until not long ago, including Interpretable Decision Sets (IDS) [16] and Diverse Rule Sets (DRS) [36]. To address conflicts arising from overlaps of rules, IDS prioritizes the rule with the highest F1-score, while DRS opts for the most accurate rule. DRS shows some advantages over IDS, including superior performance. However, both are non-probabilistic, and neither truly unordered. Recently, a method for constructing probabilistic truly unordered rule sets was proposed named TURS [34]. TURS shows that it can find better accuracy-simplicity trade-offs across datasets compared to CN2, DRS, and BRS. As a result, the binary rule set method CG and multiclass approach TURS are most relevant for comparison in our experimental section. Just like the aforementioned approaches, DR-Net [24] also constructs rule sets. As we keep the original structure of DR-Net in our approach R2N-Tab, we construct rule sets as well. This complicates the direct comparison of our method to rule learners constructing rule lists, as they can not be trivially translated to rule sets. However, we still report the results for the reference.

2.2 Feature set reduction

An important aspect of machine learning is selecting a subset of relevant features from the original feature set by removing irrelevant features. Feature selection is used for several reasons, including better model performance and increased model interpretability. In our context of supervised learning, several techniques of feature selection are applicable. As we contribute with a layer specifically for feature selection, it is interesting to compare our method with well-known feature selection algorithms. Closely related to feature selection is dimensionality reduction. Where in feature selection a subset of relevant features is selected while maintaining the original feature space, dimensionality reduction transforms the feature space into a lower-dimensional space based on selected relevant features. Both are discussed in this section.

2.2.1 Feature selection

For feature selection, a survey aimed at reviewing the state-of-the-art techniques is presented by Miao et al. [6]. Feature selection can be divided into three categories of methods, namely filter methods, wrapper methods, and embedded methods. Filter methods are used as a pre-processing step and are only used for selecting relevant features, without incorporating a machine learning model. Wrapper methods select relevant features by evaluating a specific machine learning algorithm. Different combinations of features are used for training to empirically compare results. More interesting for our approach are embedded methods, which are closely related to our method R2N-Tab. In these kinds of models, feature selection is applied during the model training phase, which is generally used to reduce overfitting.

For the task of classification, a couple of embedded methods for feature selection exist. One type of embedded method is tree-based models, including best-known techniques gradient boosting [11] and random forest [20]. These models assign importance scores to features based on how much they contribute to the model performance. Features with higher scores are considered more relevant and are selected for use in the final model. Another relevant model is the Support Vector Machine (SVM), which can be used to select features using a linear kernel [21]. SVM applies weight values to features based on their contribution to drawing the decision boundary, where higher values mean a larger impact on the classification performance. Gradient boosting and SVM provide relevant features with a positive weight, and irrelevant features with a negative weight. Thus, we can obtain feature relevances by comparing them with a threshold of zero. However, random forest only provides positive weight values to features. A threshold on feature importance has to be provided by the user. In our experimental section, we can still use random forest by selecting the k features with the highest assigned weight values, using different values for k.

2.2.2 Dimensionality reduction

Closely related to feature selection, is dimensionality reduction. Relevant features are selected and the dimensionality of datasets is reduced by removing all irrelevant features, which decreases the complexity of machine learning models using the data. Whereas R2N-Tab applies feature selection without dimensionality reduction, these techniques are still relevant to compare against as they also reduce the feature set. The domain of dimensionality reduction can be roughly classified into two types of methods.

Linear methods. One of the best-known dimensionality reduction techniques for linear data is Principal Component Analysis (PCA) [1]. PCA is aimed at finding the minimum subset of features while maximizing variance in the data. However, PCA is an unsupervised method, making its projection unrelated to class labels. Another method is Linear Discriminant Analysis (LDA) [30], which is focused on finding the smallest subset of features that best separates the classes in the data. As a result, LDA is a supervised method, making it more effective than PCA for classification tasks.

Non-linear methods. For dealing with complex, non-linear datasets, several dimensionality reduction techniques exist. These methods include Multidimensional Scaling (MDS) and Isometric mapping (Isomap), which can better deal with complex patterns in high-dimensional data than linear methods. However, these methods are mainly focused on reducing the feature set, and not classification tasks.

2.2.3 CancelOut Layer

An approach very related to our approach was proposed by Borisov et al. [4]. They designed a deep learning layer for feature selection, that learns to assign weights to features that indicate their relevance. Features that are considered uninformative are 'canceled out' with a negative weight, whereas more relevant features are represented with a positive weight. We use a similar layer inspired by Borisov et al. in our approach, however with some adaptions. We want to find a relevant set of features on the fly, and filter them from the rule sets that are constructed. As we develop a hybrid approach that combines this feature selection layer with the Decision Rules Network, we need additional functionality to correctly handle the inputs and outputs of the feature selection layer. Features that are labeled with a negative weight by the feature selection layer should be filtered, such that they can not reach the Decision Rules Network. A detailed description of our implementation of the CancelOut Layer is provided in Chapter 4.

Chapter 3

Background

In this chapter, we provide information on concepts used in this thesis. Deep Neural Networks and the process of learning rules from data are explained in detail and examples of usage are provided. Besides, the architecture of Decision Rules Network (DR-Net) that we provide an extension to is explained, including descriptions of the data that is used and the training of the network.

3.1 Deep Neural Networks (DNNs)

A concrete type of machine learning model is Deep Neural Networks [17], DNNs, which are a class of artificial neural networks. DNNs are networks of multiple layers consisting of multiple nodes, inspired by the biological neural networks existing in the human brain. DNNs can learn complex relationships between input features and a prediction target, which makes them good classifiers. In Figure 1, a simple example is shown.

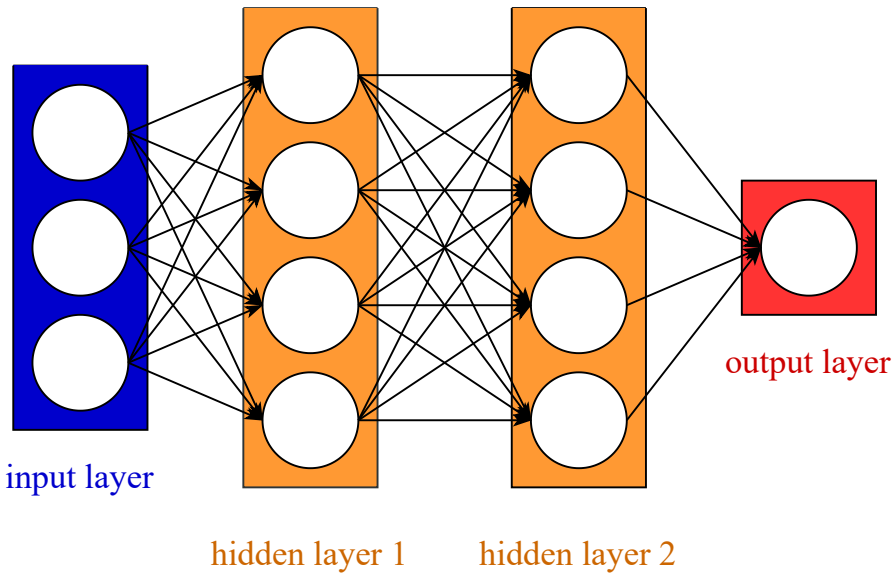


Figure 1: An example of a DNN with one input layer, two hidden layers, and one output layer. Training examples are fed into the input layer, and propagated through the hidden layers with matrix multiplication to finally come across an output in the output layer.

3.1.1 Formal definition

The connections between different layers are represented with numerical values indicating the strength of the connection, which are called the weights of the network. At the start, weights are initialized following a specific procedure which can be, e.g., randomly. Inputs of the network can be also numerical, but in our context they are binary, corresponding with active or inactive features. The output of the network is formulated by propagating the inputs through the network, performed by matrix multiplication with the weights in each layer. Consider inputs $x \in \{0, 1\}$ and weights $w \in \mathbb{R}$, then the output of a specific neuron y of the neural network is formulated by summing the product of all incoming inputs with the corresponding weights, shown in Equation 1.

$$y = \sum_{i=1}^{N} x_i w_i \tag{1}$$

In the process of learning rules from a neural network, each neuron represents an association of a feature or an association of one rule. The output of each neuron indicates a positive or negative association. In our context of binary classification, the output of the neural network is formulated by the output of the last neuron.

3.1.2 Learning strategy

During the training phase of the network, the desired weight values are learned to minimize a certain loss function. The loss is computed by comparing the predicted target values of the network to the actual target values and applying a defined function to it. At the end of each iteration of the algorithm, the computed loss is backpropagated through the network, layer by layer. Backpropagation starts at the output node, and in each layer, the partial derivatives are computed w.r.t. the weights in the layer. These partial derivatives indicate how much the loss will change by adapting each weight with a small amount. Once backpropagation has been done, the partial derivatives are used to update the weights with gradient descent. How fast weights are updating depends on the learning rate which is a user-specified parameter. In our approach, DNNs are useful for the task of classification. In the rule learning model, we propose an extension to use neural networks with a specific structure, which we explain in section 3.3.

3.2 Learning rules from data

Although machine learning algorithms have a large impact on life nowadays [27], they often suffer from the black-box problem, as described earlier. This complicates the understanding of the output they provide. In order to tackle this problem, and make machine learning algorithms explainable such that their decisions can be easily interpreted by humans, explainable AI (XAI) has made its entrance. Recently, a paper by Sahakyan et al. was proposed [26], summarizing XAI approaches in the domain of tabular data, which is also our main focus. Concretely, XAI should increase trust in machine learning models, to make them applicable in a larger range of fields. In the context of making machine learning models interpretable, XAI connects closely to the field of rule learning [12]. Rule learning is an XAI approach with the purpose of developing rules that describe the data and the output that is generated based on these rules. The learned rules provide an abstract representation of the data, increasing interpretability.

3.2.1 Illustrating the rule learning process

An example of a tabular dataset is shown in Table 1. A classifier can be trained on it to predict whether a customer is satisfied or not with the purchase that was made, based on some features including information about the customer and the purchase. The classifier should have good predictive performance, and besides, be explainable such that its decisions can be interpreted by humans.

CustomerID	Age	Gender	Category	Price	Target
1	45	Male	Electronics	395	Satisfied
2	51	Female	Fashion	140	Unsatisfied
3	28	Male	Electronics	280	Satisfied
4	39	Female	Beauty	50	Satisfied
5	70	Male	Garden	120	Unsatisfied
...	...	...	...	...	...

Table 1: A sample dataset of customers making purchases. On the dataset, a classifier can be trained to predict whether the customer is satisfied or unsatisfied with the purchase that was made.

Rule learning algorithms can be applied to the dataset to learn patterns, and provide an abstract representation of the dataset. Predictions are now descriptive and can be understood by human beings. An example of a rule set is shown in Table 2.

Rule		clause	output
1	`IF`	$\{\text{price} < 100\}$	
2	`OR`	$\{\text{category} \neq \text{fashion}\}$	
3	`OR`	$\{\text{gender} = \text{male}\}$	
	`THEN`		predict target = satisfied

Table 2: An example of a rule set for the dataset shown in Table 1. The rule learner provides a descriptive explanation of the dataset. If the price is below 100, or the category is not fashion, or the gender is male, then the machine learning model predicts that the customer was satisfied with the purchase. Otherwise, the model predicts that the customer was unsatisfied with the purchase. As the decisions of the machine learning model are defined by these rules, the model is explainable and its output can be interpreted by humans.

In Table 2 an example of a rule set is shown, which is specific to our approach. Rule learners constructing rule lists would construct multiple rules, separated by IF-ELSE statements. In the rule sets that we construct we have a disjunction of different rules, which does not implicate orders among rules and is thus better interpretable. Note that we provided an example of learning rules from data. The recently proposed DR-Net [24] that we build upon learns rules by training a neural network in a specific way. Afterward, rules can be derived from the network, which we discuss in detail in the next section.

3.3 DR-Net

Sections 3.1 and 3.2 include two concrete machine learning approaches, respectively DNNs and rule learning. The paper proposing DR-Net [24] actually brings them together in one network, learning rules from neural nets. DR-Net is an interpretable machine learning model for binary tabular data, and it has a specific two-layer neural network structure. After training, rule sets can be derived from the network that explain its decisions. With the approach, an interpretable DNN is constructed, allowing for its application in a broader range of fields. DR-Net exists of the Rules Layer followed by the OR Layer, and finally one output neuron. In Figure 2 an example of a trained DR-Net is shown, to predict whether the price of a car is below a certain threshold or not. Features of the data are consequently fed into the model and propagated through the network by matrix multiplication. In this section, we explain the structure of the network in detail.

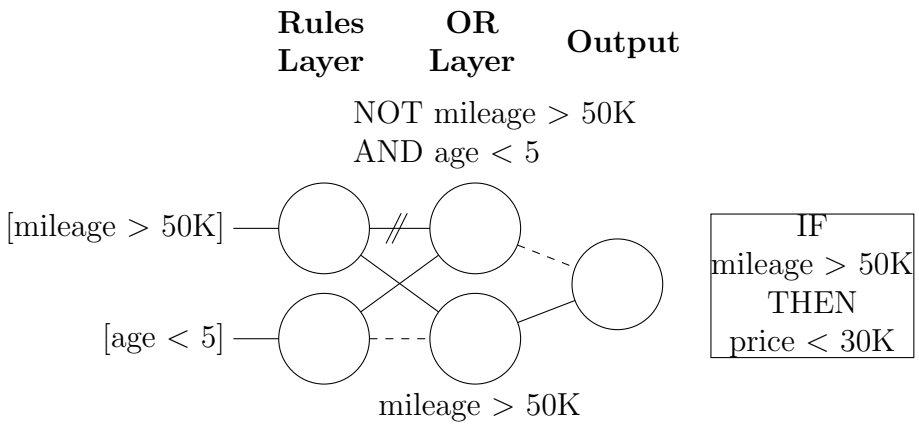


Figure 2: DR-Net [24]: a simple example of a trained model on a dataset with two features mileage and age to predict the price of a car. Solid lines represent positive associations of features, whereas crossed-out lines represent negative associations of features. Dashed lines in the Rules Layer correspond with filtered features from rules, whereas dashed lines in the OR layer correspond with filtered rules from the rule set.

3.3.1 Data preprocessing

The main goal of DR-Net is to train a classifier in the form of a Boolean logic function in disjunctive normal form (DNF), which exists of ORs and ANDs. Consequently, DR-Net consists of a Rules Layer containing neurons that perform a logical AND operation, and an OR Layer that has only one output neuron to produce the disjunction of the logical rules. To ensure that the logical AND operations are performed correctly in the Rules Layer, input features are expected to be binarized. To achieve this, a data preprocessing procedure has to be applied before feeding inputs into the model, as datasets often include numerical and categorical features that are non-binary. A sample dataset is shown in Table 3.
Before being fed to the Rules Layer of DR-Net, this dataset needs to be preprocessed to obtain binarized features. For numerical features, quantile discretization is used to obtain a set of thresholds for each feature. The original feature values are encoded in binary features by comparing them with the thresholds. The feature is encoded as 1 if the original feature value is less than the threshold, and encoded as 0 if not. Categorical variables are encoded to vectors

Age	Gender	Height
45	Male	76in
51	Female	79in
28	Male	72in
39	Female	73in
70	Male	72in
...	...	...

Table 3: Data before preprocessing. This sample dataset includes numerical and categorical features, that require preprocessing to make them applicable in our model.

of binary values. For example, consider an instance [car: BMW, mileage: 20K]. The categorical feature car can contain, e.g., categories BMW, Audi, Mercedes. Besides, for the numerical feature mileage, e.g., thresholds 10K, 20K, 30K can be obtained by quantile discretization. The instance [car: BMW, mileage: 20K] with categorical feature car and numerical feature mileage is encoded following the described procedure as [BMW, Audi, Mercedes, mileage $<$ 10K, mileage $<$ 20K, mileage $<$ 30K], which results in a binary input vector [1, 0, 0, 0, 0, 1]. Afterward, the data is in the expected binarized format and can be fed to the Rules Layer where a proper logical AND operation can be performed.
The dataset from Table 3 can be preprocessed as described in the example, which results in the dataset shown in Table 4.

Age<30	Age<40	Age<50	Gender	Height<75	Height<76	Height<77
0	0	1	0	0	0	1
0	0	0	1	0	0	0
1	1	1	0	1	1	1
0	1	1	1	1	1	1
0	0	0	0	1	1	1
...	...	...	...	...	...	...

Table 4: Data after preprocessing. For numerical features, original values are discretized by comparing them to different thresholds that they satisfy or not. For categorical variables, a new feature for each category is added to the dataset, ones indicating this example belongs to this category, and zeroes indicating it does not. Note that since we have only two attributes for the "gender" feature, only one feature with zeros and ones is needed here.

3.3.2 Rules Layer

Neurons in the Rules Layer are responsible for constructing a set of positive and negative associations of input features. The connections between input features and neurons in the Rules Layer determine how the feature is encoded in the corresponding rule. Positive weights (the solid lines in the Rules Layer in Figure 2) encode positive associations of a feature, whereas negative weights (the crossed-out lines in the Rules Layer in Figure 2) encode negative associations of a feature. The forward function of the Rules Layer is defined by Equation 2, which performs the dot product of weights and inputs and adds a dynamic bias based on the

weights of the neuron. The forward function ensures that an output of 1 can only be achieved if features with a positive weight obtain an input of 1, and features with a negative weight obtain an input of 0. Otherwise, the neuron is not activated and the corresponding rule will be filtered out in the OR Layer. Features with zero weight will not have any effect on the output (the dashed lines in the Rules Layer in Figure 2).

$$y = \sum_{i=0}^{D} w_i x_i - \sum_{w_i > 0} w_i + 1. \tag{2}$$

In the equation, x_i represents an input feature with $x \in \{0,1\}^D$ for D binarized features, whereas w_i represents the weight value of the connection between the input feature and the neuron. As the output of the Rules Layer needs to be binary to simulate the logical AND operation, an activation function is applied afterward:

$$f(x) = \begin{cases} 1, & \text{if } x = 1 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Equation 3 ensures that a neuron is only turned on when the output of Equation 2 equals 1. As a discrete activation function is applied after the Rules Layer that is not naturally differentiable, we need a way for gradients to still be correctly backpropagated through the network to learn the weights and minimize a loss function. For this, a straight-through estimator is used with the gradient clipping technique, as discussed by Bengio et al. [3]. The OR Layer receives inputs $x \in \{0,1\}$, which can not be properly derived as they are discrete. Thus, a default backward function is needed for adapting gradients, defined as:

$$g_{\hat{y}_i} = \begin{cases} 0, & \text{if } y_i < 0 \\ & \text{or } y_i > 1 \frac{\delta L}{\delta y_i} < 0 \\ g_{y_i}, & \text{otherwise} \end{cases} \tag{4}$$

allowing gradients to still be properly "flowing" through the network. In the equation $\frac{\delta L}{\delta y_i}$ represents the gradients of the loss function that needs to be minimized. Together, both conditions $y_i < 0$ and $y_i > 1 \frac{\delta L}{\delta y_i} < 0$ result in the performance of DR-Net to be empirically improved.

3.3.3 OR Layer

The OR Layer receives the activated values from neurons in the Rules Layer, which can be 1 or 0. To simulate the disjunction of the logical rules learned in the Rules Layer, the OR Layer contains one output neuron that can be activated or not, which determines the prediction of the network. First, an activation function is applied to the weights of the OR layer:

$$\hat{w}_i = \begin{cases} 0, & \text{if } w_i \leq 0 \\ 1, & \text{otherwise} \end{cases} \tag{5}$$

This equation ensures that neurons from the Rules Layer that have a corresponding weight in the OR Layer equal to or below zero are filtered from the rule set.

Consequently, the output of the OR Layer is constructed as:

$$y = \sum_{i=1}^{D} \hat{w}_i x_i - \epsilon. \tag{6}$$

With ϵ a small value such that the OR operation is 0 by default, but turned on when at least one input x_i equals 1. In the implementation of DR-Net, $\epsilon = 0.5$ is used.

3.3.4 Regularization

To update the weights of the network, a loss function is designed that includes a trade-off between model performance and rule set sparsity. For model performance, Binary Cross Entropy Loss is used (BCELoss). Let $\hat{y}$ define a batch of predicted values by the model, and y define a batch of actual target values, then BCELoss is defined as

$$\text{BCELoss}(\hat{y}, y) = \text{mean}(\text{L}) = \text{mean}(\{l_1, \ldots, l_N\}) \tag{7}$$

with l_n defined as

$$l_n = -w_n[y_n \cdot \log(\hat{y}_n) + (1 - y_n) \cdot \log(1 - \hat{y}_n)] \tag{8}$$

which stimulates the model towards correctly classifying more training examples. The other part of the loss function focuses on sparsity in the rule set. A positive weight in the Rules Layer encodes a positive association of an input feature, whereas a negative weight encodes a negated association of an input feature. Moreover, a zero weight ensures that an input feature is filtered out of the rule formulated by a specific neuron in the Rules Layer. As a weight of exactly zero is hard to achieve in a neural network, mask variables $z_i \in \{0, 1\}$ are trained to disable certain weights resulting in the exclusion of the corresponding feature. As a result, weights are replaced by its product with the mask variables:

$$w_i = w_i \cdot z_i \tag{9}$$

which disables the weight if $z_i = 0$. This results in more sparsity as the feature is not present in the rule anymore. Otherwise, the original weight remains and the feature influences the outcome of the model. The mask variables are trained with gradient descent. Thus, it is clear that we want a rule set as sparse as possible to make it more interpretable, and besides the ability to make good predictions on new data. As a result, in learning the model a trade-off has to be made between classification performance and sparsity. For this, a sparsity term is added to the loss function, besides only the BCELoss from Equation 7. The complete loss function can be formulated as

$$\mathcal{L} = \mathcal{L}_{\text{BCE}} + \lambda \mathcal{L}_{\text{R}} \tag{10}$$

where $\mathcal{L}_R$ is the loss corresponding with the sparsity of the rule set achieved with the mask variables. $\mathcal{L}_R$ is defined as the number of conditions unmasked in the rule set, normalized by the total number of conditions at the start. The strength of the mask variables enforcing sparsity is controlled by a λ constant, which allows for a trade-off between model performance and sparsity in the rule set.

The weights of the Rules Layer and the OR Layer are updated in an alternating manner, depending on a user-specified parameter, say i. Training starts with updating weights in the Rules Layer, which ensures that each neuron in the Rules Layer representing a decision rule will get sparser by filtering out features with the mask variables. At this stadium, weights in the OR Layer are fixed and we obtain the maximum number of rules. After i iterations, the OR Layer starts updating and the weights in the Rules Layer are temporarily fixed. Thus, repeatedly features are filtered out from neurons in the Rules Layer, whereafter neurons representing the rules are activated or not in the OR Layer. In our experiments, i is fixed to 500, as this was the best value as found in [24]. Equation 10 shows the general loss function optimized during the training phase. However, regarding the different λ values depending on what phase we are in, the concrete loss function is separated into two parts:

$$\begin{aligned} \mathcal{L}_1 &= \mathcal{L}_{\mathrm{BCE}} + \lambda_1 \mathcal{L}_{\mathrm{R}} \\ \mathcal{L}_2 &= \mathcal{L}_{\mathrm{BCE}} + \lambda_2 \mathcal{L}_{\mathrm{R}} \end{aligned} \tag{11}$$

λ_1 defines the coefficient used during the optimization of the Rules Layer, whereas λ_2 defines the coefficient used during the optimization of the OR Layer. The trade-off between model complexity and its predictive performance can be regulated by adapting those coefficients. In our experiments, we copy the fixed values of the coefficients from the original rule network, as we do not make adaptions here. Consequently, λ_1 is fixed to 1^{-2}, whereas λ_2 is fixed to 1^{-5}.

3.3.5 Extracting rules from DR-Net

After the training process, we want to translate DR-Net into rules for the interpretability of the decision-making. The rule set is represented by the active neurons in the OR Layer, which can be recognized by a positive weight value. For all the active neurons that represent the rules, the corresponding weight values in the Rules Layer determine which features are present in these neurons and how they are associated (positive or negative). The method for constructing the rule set is described in Algorithm 1. For each non-filtered neuron in the OR Layer, the corresponding weights in the Rules Layer are traversed. For weights below zero, a negative feature association is added to the newly constructed rule, whereas for weights above zero, a positive feature association is added to the newly constructed rule. Neurons with a zero weight are ignored. As an argument to the translate function, the headers are provided corresponding with the names of the features.

Algorithm 1 Translate DR-Net into rules

```
1: procedure GetRules(Headers)
2:     RuleSet ← [ ]                                   ▷ Initialize empty rule set
3:     Indices ← OR Layer ≠ 0                          ▷ Get active neurons
4:     for Index ∈ Indices do                          ▷ Traverse active neurons in OR Layer
5:         Rule ← [ ], Weights ← Rules Layer[Index]
6:         for Weight ∈ Weights, Header ∈ Headers do    ▷ Traverse all features
7:             if Weight < 0 then                      ▷ Negative association
8:                 Rule.extend('NOT' + Header)
9:             else if Weight > 0 then                 ▷ Positive association
10:                Rule.extend(Header)                 ▷ Add feature to rule
11:            end if
12:        end for
13:        RuleSet.extend(Rule)                        ▷ Add rule to rule set
14:    end for
15:    return RuleSet
16: end procedure
```

Chapter 4

Our proposed method: R2N-Tab

This chapter contains a detailed description of our new proposed method R2N-Tab, Rules from Neural Nets for Tabular datasets. An example of a trained model is provided in Figure 3. Note that the two-layer architecture from DR-Net is expanded to a three-layer architecture that includes a layer explicitly for feature selection upfront.

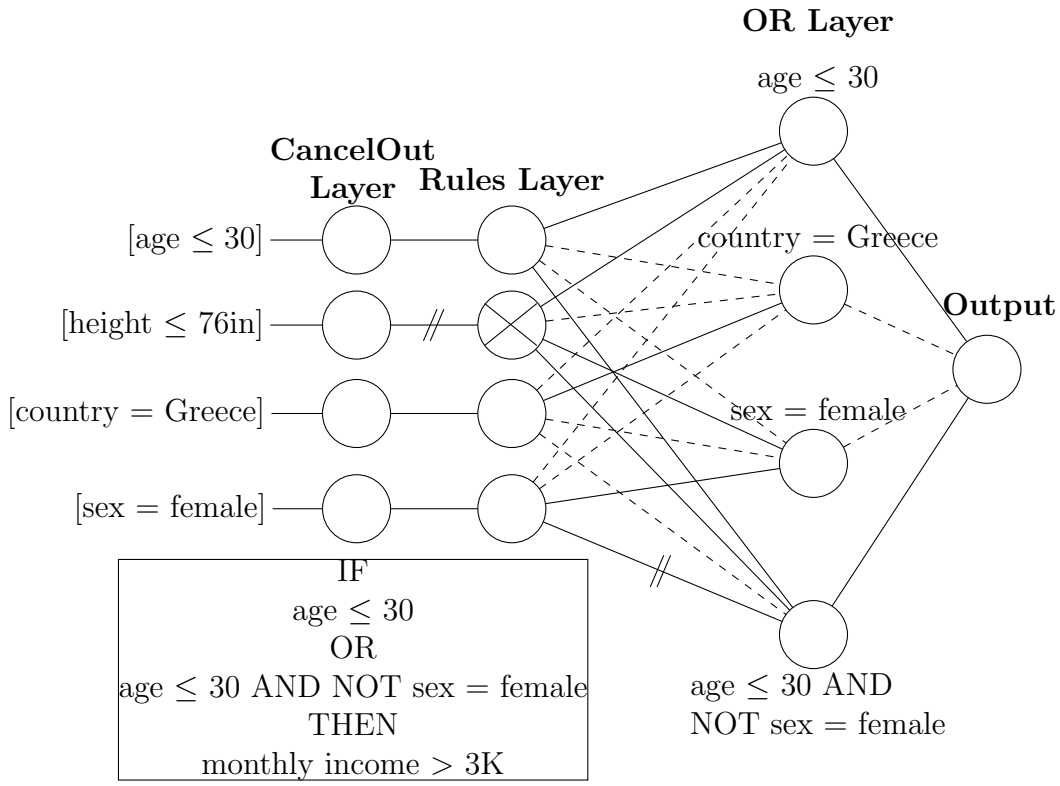


Figure 3: R2N-Tab: an example model. The solid lines represent positive weight, whereas the crossed-out lines represent negative weight. The dotted lines represent zero weight. In the beginning, we have a maximum of four rules with each four features. In the CancelOut Layer, the feature "height" is considered uninformative and has a negative weight, which results in the feature being filtered out of the rule set. The corresponding neuron in the Rules Layer is deactivated (cross). The Rules Layer and the OR Layer add more sparsity by masking weights. Together this results in a rule set of two rules and a total of three features, provided in the box below the network.

With our method R2N-Tab we propose an extension to the existing two-layer neural network architecture DR-Net. We stick to the domain of binary classification with tabular binarized datasets. The main goal is still to train a classifier in the form of a Boolean logic function in disjunctive normal form (consisting of ANDs and ORs) which serves as a rule set. However, we want to obtain a sparser rule set with fewer iterations of the algorithm, but with good predictive performance. For the development of our model R2N-Tab, we reuse the two-layer neural network architecture from DR-Net, but we add a layer for feature selection up front called the CancelOut Layer. The feature selection layer has a neuron for each feature in the dataset, which should decide whether the feature is informative or not. Each neuron in the CancelOut Layer has only one connection to the corresponding neuron in the Rules Layer, which belongs to the same feature in the dataset (as can be observed from Figure 3). For the design of the feature selection layer, we were inspired by the CancelOut Layer from [4]. While the idea of both layers is similar, our CancelOut Layer is constructed differently, as we want to apply feature selection on the fly. In this chapter, we explain the architecture of our method R2N-Tab in detail.

4.1 CancelOut Layer

The CancelOut Layer takes features x_i as input with $x_i \in \{0,1\}^D$ for D binarized features. The number of neurons in the CancelOut Layer is equal to the number of features in the dataset, and each neuron has one connection to the corresponding neuron in the Rules Layer, which indicates the importance of the feature. Features that are considered informative are represented with a positive weight, while less informative features obtain a negative weight. As a result, features with a negative weight should be filtered out in the Decision Rules Network that comes after the CancelOut Layer, as they do not contribute to better predictive performance of the model. Consequently, the CancelOut Layer is responsible for filtering out as many features as possible, as the goal of the layer is to obtain a sparser rule set. However, features should not be filtered out at the cost of predictive performance. To ensure that features with negative weight are filtered out in the Decision Rule Network, a ReLU activation function is applied to inputs of the CancelOut Layer based on the weights as shown in Equation 12.

$$\mathrm{CancelOut(X)} = \mathrm{X} \odot \mathrm{ReLU(W)} \tag{12}$$

where X represents the input features $\in \{0,1\}^D$ and W the weight vector, where $\odot$ means element-wise multiplication of the input features with their corresponding weight. The ReLU activation function is applied to the weights of the CancelOut Layer, and it ensures that negative weights are set to zero before multiplication with the inputs.

E.g., weight vector $\mathrm{W} = \begin{bmatrix} \mathrm{w} \\ \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \end{bmatrix}$ activates as $\mathrm{ReLU(W)} = \mathrm{ReLU}\left(\begin{bmatrix} \mathrm{w} \\ \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \end{bmatrix}\right) = \left(\begin{bmatrix} \mathrm{ReLU(w)} \\ \mathrm{ReLU(x)} \\ \mathrm{ReLU(y)} \\ \mathrm{ReLU(z)} \end{bmatrix}\right)$.

Note that the ReLU function activates inputs as in Equation 13.

$$\mathrm{ReLU(X)} = \mathrm{Max(0,X)} \tag{13}$$

Thus, the ReLU activation function ensures that weights greater than zero will remain their original values, while weights below zero are zeroed out.

Note that the CancelOut Layer developed in our model differs from the CancelOut Layer from [4], as we use ReLU activation in the forward function instead of sigmoid. In our approach, ReLU is more useful as it zeroes out negative inputs which is required to filter features with negative weight in the Decision Rules Network. After the CancelOut Layer, an activation function is applied to obtain binarized feature values $\in \{0, 1\}$, as shown in Equation 14.

$$f(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

As a result, neurons in the CancelOut Layer that have zero weight will input a value of zero to the Rules Layer of the Decision Rules Network for the feature they represent. Due to Equation 2, inputs to the Rules Layer that have a value of zero do not influence the output of the model. Thus, features can be involved in the Decision Rule Network iff their original values as inputted to the CancelOut Layer are equal to one, and besides they are not filtered by the CancelOut Layer. Otherwise, the features can not affect the output of the model, either because their original feature value was zero, or they are filtered by the CancelOut Layer. As we use a discrete activation function that is not naturally differentiable, a variant of the straight-through estimator from Equation 4 is used for correctly backpropagating gradients, as shown in Equation 15

$$g_{\hat{y}_i} = \begin{cases} 0, & \text{if } y_i \leq 0 \\ g_{y_i}, & \text{otherwise} \end{cases} \tag{15}$$

Note that the straight-through estimator from Equation 15 is slightly different than the straight-through estimator from Equation 4. We removed the condition $y_i > 1\frac{\delta L}{\delta y_i} < 0$ and changed $y_i < 0$ to $y_i \leq 0$, as this resulted in better model performance.

In theory, the CancelOut Layer is active from the start. However, we want to prevent the feature selection layer from canceling features too early in the training process. As the predictive performance is still weak, canceling more features can not result in large performance drops yet. To establish this, the weights in the CancelOut Layer are initialized at 4. This number is also used in [4]. This is a small positive number such that features are biased to be informative from the start, but can be still canceled after some epochs with a negative weight. With this initialization of the weights, the model has some time to update its weights to achieve better predictive performance without the concern of filtering possibly informative features. We also choose a uniform initialization of the weights by providing them with the same initial weight value, not to bias some features to be informative or not informative.

4.2 Decision Rules Network

In this thesis project, we extend DR-Net [24] with a feature selection layer in front of it. For DR-Net, we keep its original two-layer architecture. DR-Net trains mask variables to filter features from rules in the Rules Layer and filter rules in the OR Layer. The CancelOut Layer contributes by directly labeling uninformative features with a negative weight, and filtering them before rule sets are constructed in the next layers. In DR-Net, the Rules Layer directly receives the binarized inputs from the dataset. However, in our approach, the inputs are first passed through the CancelOut Layer, which works as a filter mechanism. The layer only allows features contributing to a better predictive performance of the model to pass through. The filtered input vector is inputted to the Decision Rules Network, where the interpretable rule sets are constructed. In the Decision Rules Network, we have to choose values for the hyperparameters controlling the learning process in the Rules Layer and in the OR Layer. For this, we use the best results as found in DR-Net [24], as we stick to the original structure. This means that the λ coefficient for optimization of the Rules layer is set to 0.01, and the λ coefficient for optimization of the OR Layer is set to 0.00001. Besides, the learning rate of weights in the Decision Rules Network is set to 0.01. The number of neurons in the OR Layer corresponding with the maximum number of rules is also a hyperparameter, but this depends on the dataset that is used. We vary this hyperparameter in the experiment chapter to empirically obtain the best results.

4.3 Regularization

As we provide an extension to DR-Net by adding a layer for feature selection in front, we partly reuse the loss function that is used in this model, as shown in Equation 10. However, we add one constraint controlling the weights in the CancelOut Layer. We refer to the loss of the CancelOut Layer with $\mathcal{L}_{\mathrm{C}}$, which is defined in Equation 16.

$$\mathcal{L}_{\mathrm{C}} = \frac{1}{N} \cdot \sum_{i=0}^{N} \mathrm{ReLU}(\mathrm{W}_i) \tag{16}$$

The penalty of the CancelOut Layer is constructed by summing all ReLU-activated weights for all neurons in the layer, normalized by the total number of neurons, which is equal to the number of features. In the defined loss function for the CancelOut Layer, filtered features get a zero penalty, while the penalty for unfiltered features is equal to their non-zero weights. The loss function penalizes the weights in the CancelOut Layer in a way to filter more features, as this would result in a lower loss because of their zero penalties. However, achieving more sparsity in the rule set by filtering features is not the only concern of the method, as it is still required to obtain a rule set that has good predictive performance. As a result, the global optimization criterion of the model is formulated as follows:

$$\begin{aligned} \mathcal{L}_1 &= \mathcal{L}_{\mathrm{BCE}} + \lambda_1 \mathcal{L}_{\mathrm{R}} + \lambda_3 \mathcal{L}_{\mathrm{C}} \\ \mathcal{L}_2 &= \mathcal{L}_{\mathrm{BCE}} + \lambda_2 \mathcal{L}_{\mathrm{R}} + \lambda_3 \mathcal{L}_{\mathrm{C}} \end{aligned} \tag{17}$$

Note that this is similar to the loss function used in DR-Net, as defined in Equation 11. However, the additional regularization term is included controlling the weights in the CancelOut Layer, defined in Equation 16.

4.4 Optimization strategy

For the optimization of our model, we use a similar approach as in the Decision Rules Network, where the weights in the Rules Layer and the OR Layer are optimized in an alternating manner. The phase starts at zero, which means that we optimize the parameters in the Rules Layer. After a specified number of epochs, the phase changes to one, which means that we optimize the parameters in the OR Layer. This alternating process is continued until the algorithm terminates.

For the CancelOut Layer, we develop a dynamic strategy to measure the desired number of iterations that are required for optimizing its parameters. We develop a mechanism that keeps track of the AUC values as measured on the training set in each of the i iterations of the training process. After each ith iteration, the average AUC of the past i iterations is evaluated to see how it has changed compared to before. In case the average AUC has decreased, and the CancelOut Layer has canceled more features in these i iterations, we can say that the optimization of its parameters had an undesired effect. In this case, we revert the optimization changes that were applied in the past i iterations w.r.t. the parameters in the CancelOut layer. Further optimization of its parameters is halted from now on. In case the CancelOut Layer has still potential, and the average AUC in the last i iterations has not decreased, another tracking phase of i is performed. An example of the dynamic mechanism to measure the CancelOut Layer potential on the fly is shown in Figure 4.

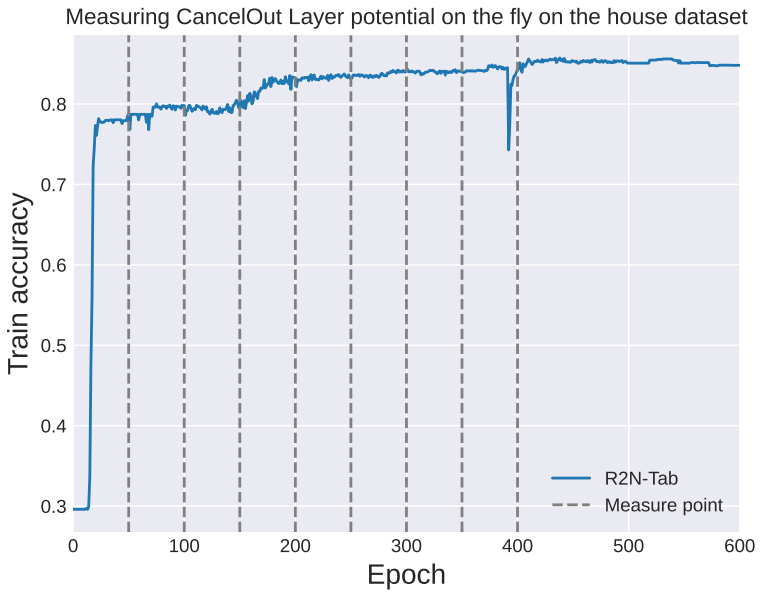


Figure 4: An example of how the CancelOut Layer potential is measured on the fly using the house dataset. In each ith epoch (i=50), the predictive performance of the model is measured. After 400 epochs, a decrease in predictive performance is measured, and the CancelOut Layer is fixed for the remaining number of epochs.

In our experiments, we fix i to 50 epochs, which is a small number of epochs allowing for the best determination of the desired number of iterations of the CancelOut Layer. Besides, 50 epochs are sufficient to obtain a substantial number of AUC measures.

Measuring the desired number of iterations of the CancelOut Layer has the advantage of sparing out a hyperparameter that defines the number of optimizations of the parameters. Besides, we can cancel a maximum number of features while maintaining good predictive performance.

In Figure 5 an overview of a complete training phase of 2000 epochs is shown. Training starts with phase zero which results in optimization of the parameters in the Rules Layer. Every 500 epochs the phase changes, which results in the optimization of the Rules Layer and the OR Layer in an alternating manner. The CancelOut Layer is optimizing its parameters from the start. After a dynamically determined number of epochs (400 in this example), the CancelOut Layer will be fixed as the best fit between predictive performance and sparsity has been found.

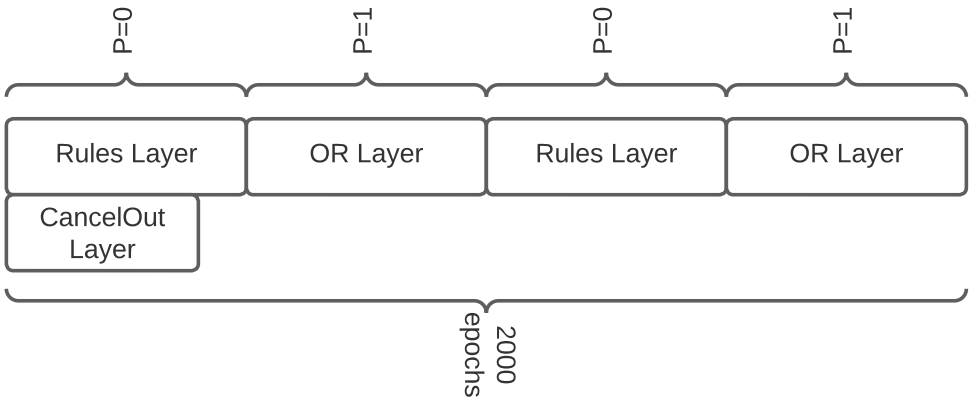


Figure 5: Global diagram of our model running 2000 epochs, where phase ($\mathrm{P} \in \{0, 1\}$) changes every 500 epochs. The CancelOut Layer is only updating its parameters for 400 epochs, as this resulted in the best sparsity and predictive performance. After the 400 epochs, the CancelOut Layer is fixed.

Chapter 5

Experiments and results

In this chapter, our implementation [1] of DR-Net together with the CancelOut Layer is evaluated. We use 8 varied datasets (shown in Table 5), that were selected to be diverse in number of features, instances, and features types. The datasets are ordered by descending number of features after preprocessing (D_2).

dataset	D_1	D_2	|instances|	class imbalance	features type
heloc	23	154	10459	0.52	numerical
house	16	132	22784	0.70	numerical
adult	14	128	32561	0.76	categorical and numerical
magic	10	90	19020	0.65	categorical and numerical
diabetes	9	67	768	0.65	numerical
chess	36	38	3195	0.52	categorical
banknote	4	36	1371	0.56	numerical
tictactoe	9	27	957	0.65	categorical

Table 5: Properties of the datasets used in our experiments: D denoting the dimensionality of the datasets, with D_1 the number of features in the dataset before preprocessing has been applied, and D_2 the number of features in the dataset after preprocessing; |Instances| denotes the number of examples that are available in each dataset; class imbalances denotes the fraction of instances belonging to the majority class; features type denotes whether the dataset includes categorical or numerical features or both. The provided selection includes datasets of different sizes in the number of features and instances, and besides different types of features, resulting in the best evaluation of our model.

[1] Implementation available on `https://github.com/mrvanderzwart/R2N-Tab`

The first four datasets are also used for the evaluation of DR-Net in [24]. The first two datasets are recent (2021), FICO HELOC (heloc), and a dataset on home price prediction (house). The adult and the magic datasets are from the UCI Machine Learning Repository [19]: Adult census (adult) and MAGIC gamma telescope, which is the magic dataset. For thorough evaluation, we add four more datasets which are UCI benchmark datasets, also used in [34]. All selected datasets are commonly used in classification papers. All datasets have a binary target variable that can be predicted with binary classification.

As datasets grow in features, they are likely to contain more irrelevant features, that do not contribute to better predictive performance of deep learning models. Table 5 includes datasets that differ in dimensionality and the number of instances. For evaluation, we compare our model based on predictive performance, rule set complexity, and runtime. From Table 5 can be observed that some datasets suffer from class imbalance. Thus, to measure the predictive performance we report the ROC-AUC scores, as this metric is less sensitive in dealing with imbalanced datasets compared to, e.g., accuracy scores. The rule set complexity is expressed in the number of rules in the rule set, and besides the total number of conditions in the rule set obtained by summing all conditions in all rules. In the experiments, plots are used for the visualization of the results, and abstract overviews are provided in the form of tables. For a significant evaluation, all results are averaged using 5 times repeated 5-fold cross-validation. Each section has a setup part, where the corresponding experiment is described. Besides, we provide a results and analysis part where the outcomes of the experiments are presented and interpreted. For the exact hyperparameter sets used during the experiments, we refer to Chapter 7 (Appendix A). For performing our experiments, we made use of the Distributed Ascii Supercomputer 5 (DAS5) [2]. We used the LU cluster, which is available for students at Leiden University on request. It consists of 24 nodes of 2.4 GHz CPUs and 64GB of RAM, which is ideal for running our experiments in parallel.

5.1 Comparison with Decision Rules Network

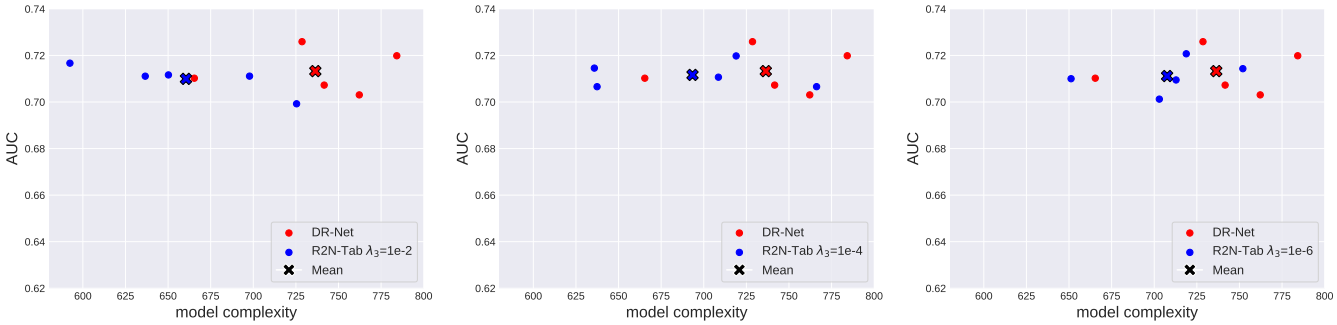
In this experiment, we compare our new proposed method R2N-Tab to the original Decision Rules Network, DR-Net. With our new approach, we want to develop a method that can construct small rule sets with a small number of epochs. Besides, the rule set should have good predictive performance in terms of AUC scores.

5.1.1 Experimental setup

A hyperparameter R2N-Tab has in addition to DR-Net, is the λ_3 coefficient defining the trade-off between model performance and sparsity in the CancelOut Layer. In this experiment, we compare DR-Net to R2N-Tab, varying the λ_3 coefficient. For this experiment, we used λ_3=1e-2, λ_3=1e-4 and λ_3=1e-6. These values are a good range for the λ_3 coefficient, as higher values resulted in aggressively removing features at the cost of predictive performance, whereas decreasing the λ_3 coefficient further than 1e-6 had no effect. For DR-Net, a large number of epochs is required to obtain a sparse rule set. As we want to develop a method that obtains sparser rule sets with fewer epochs, and besides comparable accuracy values, we run both models using a small number of 1000 epochs. In combination with the phase change of 500, 1000 epochs is enough to perform one full Rules Layer optimization phase and one full OR Layer optimization phase. Thus, 1000 is a small number of epochs but sufficient for one Rules Layer optimization phase and one OR Layer optimization phase. To explicitly show the relation between the predictive performance of both models and the model complexity, we use scatter plots. For the predictive performance of both methods, we measure the final AUC values after the training phase, which is performed on the test set. For the model complexity, we take the total number of features in the rule set as derived from the networks after the 1000 epochs.

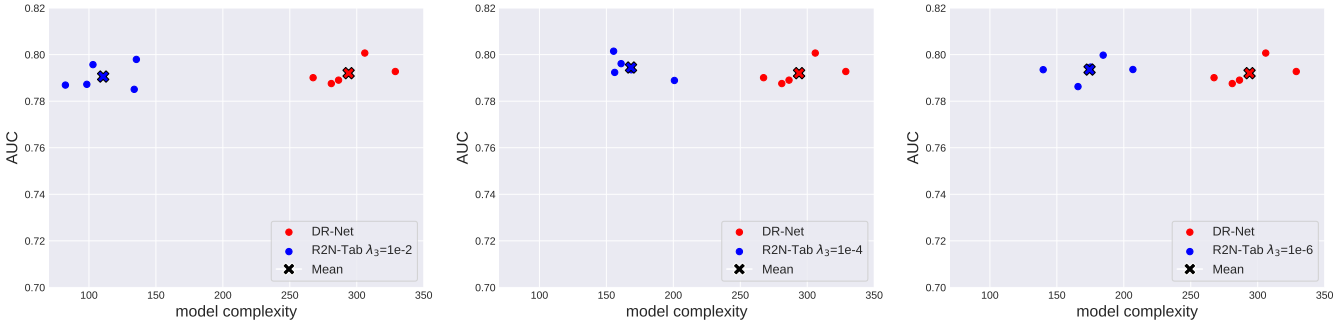
5.1.2 Results and analysis

In Figure 8, scatter plots are shown for three datasets visualizing the relation between predictive performance and model complexity for R2N-Tab (blue dots) against DR-Net (red dots). For both methods, 5 dots are shown representing each fold of our 5 times repeated 5-fold cross-validation experiment. Besides, the means of both methods are shown with a cross. The model complexity in terms of the total number of conditions in the rule set is shown on the x-axis, whereas the predictive performance in terms of AUC score is shown on the y-axis. A global observation that can be noticed is that the blue dots representing R2N-Tab are mostly located on the left side of the x-axis, while the red dots representing DR-Net are mostly located on the right side. This is a promising observation to make, as this corresponds with R2N-Tab overall constructing a sparser rule set compared to DR-Net. Across datasets, different values for the λ coefficient result in different model complexities and AUC scores. In particular the heloc and adult seem to benefit from a large λ coefficient for better model complexity. However, we also have to take into consideration the predictive performance of both methods, shown on the y-axis. Overall, DR-Net has slightly better AUC scores compared to R2N-Tab. However, differences are very small and dependent on the λ coefficient values. On all datasets, we have to deal with a clear trade-off between predictive performance and model complexity. Overall, this experiment proves that our method can construct sparser rule sets compared to DR-Net for a small number of epochs, while the rule sets have comparable predictive performance.



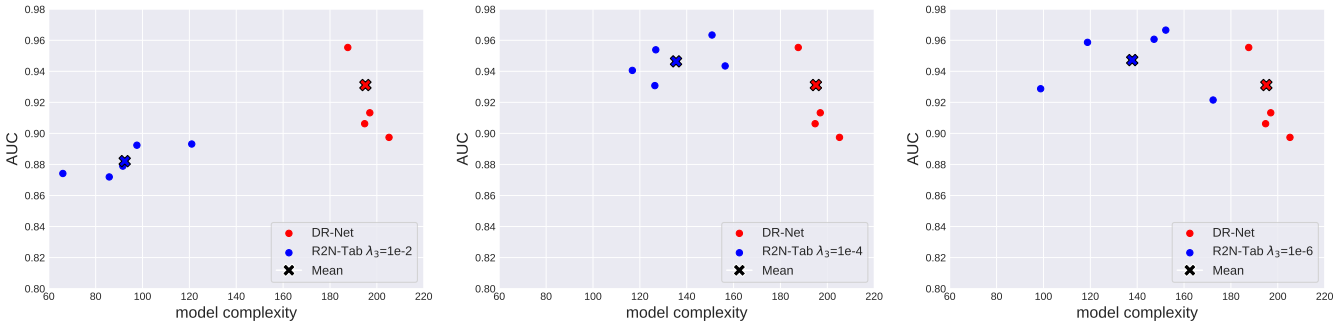
λ_3 =1e-2 λ_3 =1e-4 λ_3 =1e-6

(a) heloc dataset



λ_3 =1e-2 λ_3 =1e-4 λ_3 =1e-6

(b) adult dataset



λ_3 =1e-2 λ_3 =1e-4 λ_3 =1e-6

(c) chess dataset

Table 8: Scatter plots explicitly showing model complexity in terms of conditions in the rule sets compared to the predictive performance in terms of AUC scores. For clarity reasons, we visualized 3 out of 8 datasets. For our method R2N-Tab, we varied the λ coefficient defining the trade-off between canceling features and predictive performance. Overall, R2N-Tab constructs sparser rule lists than DR-Net, but maintains comparable AUC values.

	AUC				|RULES|			|CONDITIONS|		
DATASET	DR-NET	R2N-TAB λ=1E-2	R2N-TAB λ=1E-4	R2N-TAB λ=1E-6	DR-NET	R2N-TAB λ=1E-2	R2N-TAB λ=1E-4	R2N-TAB λ=1E-6	DR-NET	R2N-TAB λ=1E-2	R2N-TAB λ=1E-4	R2N-TAB λ=1E-6
HELOC	**0.71**	**0.71**	**0.71**	**0.71**	36	**33**	34	36	736	**660**	693	708
HOUSE	**0.83**	0.81	0.82	0.82	36	**25**	26	27	428	**244**	245	260
ADULT	**0.79**	**0.79**	**0.79**	**0.79**	18	11	**10**	**10**	294	**111**	168	174
MAGIC	**0.85**	0.84	0.84	0.84	34	**29**	31	31	362	**256**	315	304
DIABETES	**0.73**	**0.73**	**0.73**	**0.73**	36	37	37	**35**	545	**530**	552	539
CHESS	0.93	0.88	**0.95**	**0.95**	40	**25**	30	30	195	**92**	135	138
BANKNOTE	**0.99**	**0.99**	**0.99**	**0.99**	**43**	**43**	**43**	**43**	433	**431**	439	437
TICTACTOE	**0.85**	0.84	0.82	0.84	44	44	**41**	43	245	238	**221**	229

Table 9: AUC scores and model complexity values of R2N-Tab and DR-Net obtained in this experiment, visualized by the scatter plots in Figure 8. Across datasets, the best scores per column are shown in bold. A closer look at the AUC values reveals that R2N-Tab performs equally well on all datasets. Besides, R2N-Tab has better model complexity in terms of the number of rules and conditions in the rule sets for all λ coefficients.

In Table 9, the results as found in this experiment for all datasets are shown. As can be observed, R2N-Tab constructs rule sets consisting of fewer rules and fewer conditions, which together contribute to better model complexity. For the remainder of our experiments, we need fixed values for the λ_3 coefficient that best design the trade-off between interpretability and predictive performance. We select the best values for λ_3 coefficients based on the training phase, choosing the one that results in the Pareto optimal solution. The results are shown in Table 10.

DATASET	λ_3
heloc	1e-2
house	1e-4
adult	1e-2
magic	1e-2
diabetes	1e-2
chess	1e-4
banknote	1e-2
tictactoe	1e-6

Table 10: λ_3 coefficients per dataset that result in the Pareto optimal solution, measured on training data.

These values for the λ_3 coefficient result in the Pareto optimal solution, which is the best trade-off between predictive performance and model complexity. To conclude the comparison of R2N-Tab with DR-Net, we report the running times in Figure 7. As can be observed, R2N-Tab has a slightly larger running time per fold than DR-Net due to the additional feature selection layer. However, DR-Net takes more time to construct rule sets that are as sparse as the rule sets constructed by our method R2N-Tab.

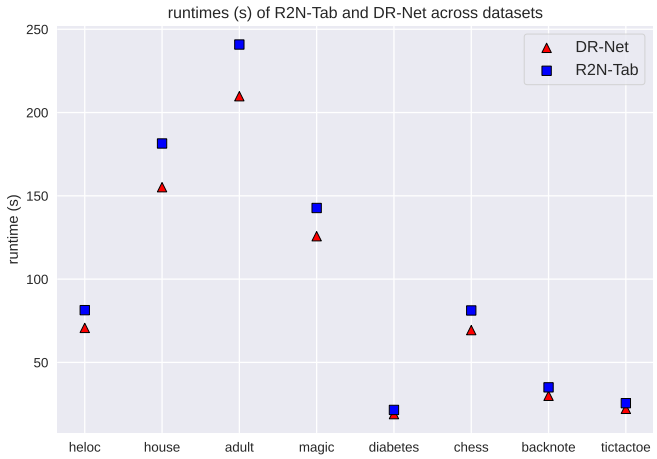


Figure 7: Running times per fold in seconds per dataset, comparing DR-Net (red triangles) to R2N-Tab (blue squares). Due to the deep learning layer in front for feature selection that R2N-Tab has in addition to DR-Net, our method has a slightly larger running time. However, R2N-Tab constructs sparser rule sets compared to DR-Net with comparable predictive performance.

5.2 Comparison with feature selection algorithms

As we propose a model that performs feature selection, comparing our model to other feature selection methods is interesting. For comparison, we use the following methods: 1) Gradient Boosting (GB), the three-based feature selector; 2) Linear Discriminant Analysis (LDA), the dimensionality reduction technique; 3) Support Vector Machine (SVM), the linear classifier; and 4) Random Forest (RF), the multitude of decision trees. All are methods that apply feature selection and can be used for classification tasks at the same time. However, our method has the benefit of constructing rule sets, that provide interpretability in the decision-making. In this experiment, we compare the predictive performance of all methods, in combination with the reduction of the feature set.

5.2.1 Experimental setup

In essence, our method R2N-Tab works just like the other mentioned feature selection methods. We run the algorithms, and feature importance scores are applied to all features. In most cases, a positive score indicates that the feature is considered relevant, whereas a negative score means that the feature is considered irrelevant. However, this is not the case in random forest, which only provides a ranking among features. To obtain a set of relevant features, we select the top k percent based on the feature importance scores, using $k = 25\%$, $k = 50\%$, and $k = 75\%$.

In this experiment, we measure the sparsity in the feature sets obtained by the different feature selection algorithms. Let x_i be the feature importance score of a feature x in a feature set $\{x_0, ..., x_N\}$, then we first apply Equation 18.

$$f_i = \begin{cases} 1, & \text{if } x_i < 0 \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

f_i will be 1 if the feature importance score x_i is below zero, which indicates that the feature is considered uninformative. Otherwise, f_i will be 0. Next, to obtain the reduction score for the corresponding feature selection algorithm, we sum all f_i values and normalize them by the total number of features, shown in Equation 19.

$$\text{reduction} = \frac{\sum_i^N f_i}{N} \tag{19}$$

A reduction score of x means that the feature set is reduced by $x\%$. Finally, we measure the predictive performance by applying a random forest classifier, using only the features that obtained a positive feature importance score. To obtain the feature importance scores of the feature selection algorithms, we can simply run the algorithms and extract the scores. Note that this is the procedure for all feature selection methods except random forest. For our method R2N-Tab, we run the algorithm until the CancelOut Layer is fixed. Afterward, we can derive the feature importance scores by using its learned parameters.

5.2.2 Results and analysis

Table 11 shows the results of our feature selection experiment. For all feature selection methods, the table reports the reduction score which indicates the percentage of features removed from the feature sets. Besides, the AUC scores are reported, obtained by applying a random forest classifier, using only the features in the reduced feature sets.

	AUC							REDUCTION						
DATASET	R2N-TAB	GB	LDA	SVM	RF* 0.25	RF* 0.50	RF* 0.75	R2N-TAB	GB	LDA	SVM	RF* 0.25	RF* 0.50	RF* 0.75
HELOC	0.71	**0.72**	**0.72**	**0.72**	0.72	0.72	0.69	0.41	0.16	**0.53**	0.51	0.25	0.50	0.75
HOUSE	0.83	**0.87**	0.82	0.82	0.87	0.86	0.83	0.33	0.19	**0.53**	0.52	0.25	0.50	0.75
ADULT	0.78	**0.79**	0.77	0.74	0.76	0.76	0.75	**0.76**	0.28	0.66	0.52	0.25	0.50	0.75
MAGIC	0.84	**0.87**	0.71	0.73	0.86	0.84	0.84	**0.70**	0.09	0.64	0.63	0.25	0.50	0.75
DIABETES	**0.76**	0.67	0.69	0.69	0.70	0.67	0.67	0.18	0.0	0.52	**0.54**	0.25	0.50	0.75
CHESS	0.94	**0.99**	0.75	0.76	0.98	0.98	0.98	**0.54**	0.12	0.55	0.55	0.25	0.50	0.75
BANKNOTE	0.99	**1.00**	0.99	0.99	1.00	0.97	0.93	**0.43**	0.03	0.14	0.22	0.25	0.50	0.75
TICTACTOE	**0.97**	**0.97**	0.80	0.82	0.99	0.91	0.77	0.23	0.19	**0.56**	0.48	0.25	0.50	0.75

Table 11: AUC and reduction scores for different feature selection methods across datasets. Gradient boosting obtains high AUC scores, but applies little reduction to the feature sets. LDA and SVM obtain high reduction scores, however, they are very behind gradient boosting concerning AUC scores on most datasets. Overall, our method R2N-Tab finds the Pareto optimal solution, maximizing the predictive performance while minimizing the feature set. We achieve similar AUC scores compared to gradient boosting but with good reduction scores.

Interpreting the results, we observe a couple of things. Overall, R2N-Tab and LDA obtain the best reduction scores, whereas gradient boosting is more careful in reducing the feature sets. Including the corresponding AUC scores, we observe that gradient boosting excels across all datasets. Thus, gradient boosting is a feature selection algorithm focusing on maintaining predictive performance, not filtering too many features at the cost of AUC values. LDA and SVM are more aggressive feature selection algorithms, resulting in them being outperformed by gradient boosting when it comes to predictive performance using AUC values. Including our method in the evaluation, we observe that we obtain similar AUC values compared to gradient boosting across datasets. However, our feature selection layer CancelOut can decrease the feature sets by large margins, even outperforming aggressive feature selection algorithms LDA and SVM across some datasets. All in all, we developed a method that has a feature selection layer that minimizes a feature set in a way to only maintain features contributing to the predictive performance of the method. Compared to the other feature selection algorithms, R2N-Tab provides the best fit between predictive performance and model complexity.

In Figure 8, the running times per fold in seconds for the different feature selection methods are shown across datasets. All feature selection algorithms we compare our method R2N-Tab against have shallow running times. Our proposed method needs about one to two orders of magnitudes longer to converge. However, R2N-Tab performs feature selection on the fly, and rule lists are constructed at the same time. Besides, the method is finding its parameters to reach optimal predictive performance. For the other feature selection methods, features have to be selected beforehand. Afterward, the reduced feature sets can be used for a classification task. Our method combines reducing the feature set by the CancelOut Layer with constructing rule lists and classification, which results in longer running times.

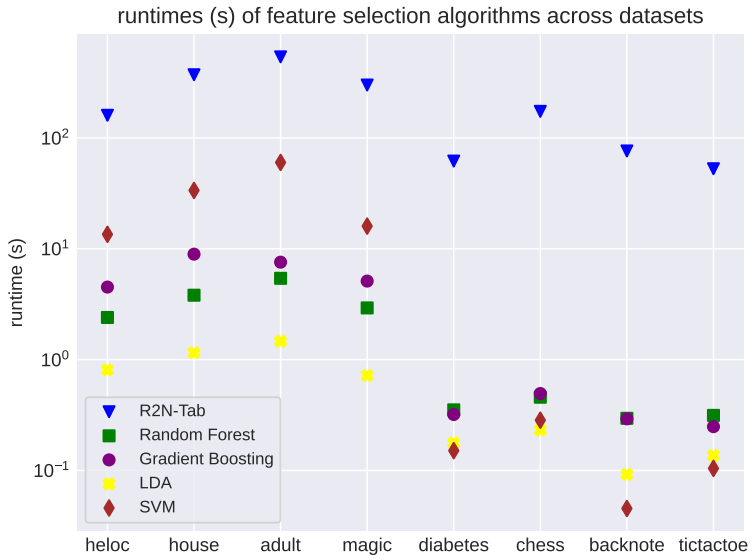


Figure 8: Running times per fold in seconds per dataset, comparing different feature selection algorithms. Our method R2N-Tab takes about one to two orders of magnitudes longer for execution compared to the other feature selection algorithms but provides additional functionality: constructing rule sets that provide interpretability in the decision-making process.

5.3 Comparison with rule learning algorithms

In this thesis project, we develop an interpretable method for classification. In the domain of interpretable machine learning, several approaches exist that develop rules explaining the decision of a model, as discussed in Section 2.1. In this section, we compare the predictive performance of our model against the mentioned rule learners. Besides, the interpretability in terms of model complexity is discussed.

5.3.1 Experimental setup

In defining the sparsest rule lists with the best predictive performance among datasets, both metrics are strongly correlated. A filtered rule list mostly comes with some decrease in predictive AUC values. To design the trade-off between both metrics, threshold values are built into the rule learners. CART, C4.5, and Classy include a maximum depth constraint, specifying that rule lists are only constructed up to a provided point. Lower values will result in sparser rule lists but with possibly lower predictive performance compared to higher constraint values. RIPPER includes a threshold value defining the maximum number of total conditions in the constructed rule list, which we also built into our method R2N-Tab. Varying the threshold value, we can design the trade-off for all the rule learners. In Figure 9, an example of the process on the adult (left) and house (right) datasets is shown.

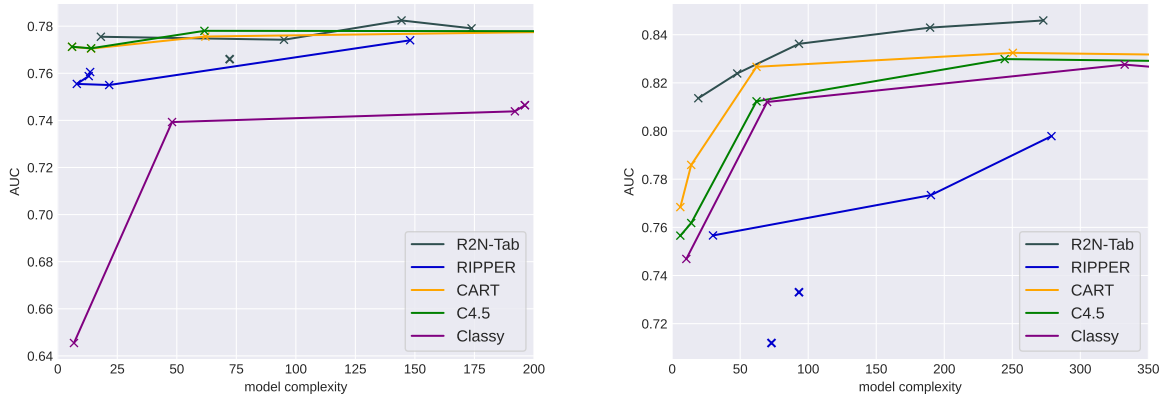


Figure 9: The predictive performance (AUC) and model complexity trade-off for different rule learners. On the left, the adult dataset is shown, whereas on the right, the house dataset is shown. The trade-off values are obtained by varying the maximum depth constraint among the methods.

For all rule learners, a sparser rule list is combined with decreased AUC values. For the experiment, we have to choose from the Pareto front. The best case scenario is a maximum AUC score of 1.0, combined with a minimum rule set complexity of 1 condition, which is known as the utopian point. For selecting the optimal value from the Pareto front, we take the point that has the shortest distance to the utopian point, which maximizes the predictive performance while minimizing the model complexity.

5.3.2 Results and analysis

In Table 12, the Pareto optimal solutions for all discussed rule learning methods are shown. A first look at the results reveals that TURS outperforms all rule learning methods in the predictive performance of the constructed rule sets, expressed in AUC scores. However, TURS constructs very dense rule sets, consisting of a lot of rules and conditions. Especially on the large datasets heloc, house, adult, and magic, rule sets explode in size. As a result, these rule sets are hard to interpret. Apart from our model, we include the Column Generation (CG) model, which also constructs rule sets. In terms of model complexity, CG constructs rule sets consisting of substantially fewer rules and conditions compared to TURS. However, AUC scores are quite low. Still, CG provides better solutions on the Pareto front, maximizing both objectives. Considering the classic rule learning algorithms that construct rule lists, similar AUC values are obtained compared to CG and our method. However, the rule lists contain a substantial number of rules and conditions. Further decreasing the model complexity results in a huge drop in AUC scores, as can be observed from Figure 9. This also holds for the CLASSY method.

Interpreting the Pareto optimal results of our method R2N-Tab, we observe that we can construct very sparse, yet accurate rule sets for the large datasets heloc, house, adult, and magic. For these datasets, the model complexities are comparable with the CG algorithm, whereas the trade-off between rule set complexity and predictive performance is superior to the other rule learning algorithms. However, R2N-Tab struggles to find a good fit between predictive performance and rule set complexity on the smaller datasets. In the smaller datasets, we are limited to a small number of examples. Our method benefits from large datasets, that provide lots of training examples, required for deep learning to work well.

In Table 10, the execution times per fold in seconds are shown for all rule learners across datasets. On the larger datasets, TURS has a tremendous execution time, which is 2 to 3 orders of magnitude longer than most other rule learners. On the other hand, decision tree methods CART and C4.5 both have a very small execution time across datasets. On the larger datasets, all remaining rule learners perform similarly in execution times. However, our method R2N-Tab takes longer on the smaller datasets, as it struggles to find a good fit between predictive performance and model complexity due to the limited number of examples available in these datasets.

	AUC							|rules|							|conditions|						
dataset	R2N-Tab	RIPPER	CART	C4.5	Classy	TURS	CG	R2N-Tab	RIPPER	CART	C4.5	Classy	TURS	CG	R2N-Tab	RIPPER	CART	C4.5	Classy	TURS	CG
HELOC	0.69	0.71	0.71	0.71	0.70	**0.73**	0.69	**1**	2	4	4	10	75	**1**	4	7	6	6	10	646	**3**
HOUSE	0.82	0.80	0.83	0.81	0.81	**0.91**	0.80	5	40	32	32	35	209	**2**	19	271	62	62	70	1648	**5**
ADULT	0.78	0.78	0.78	0.78	0.75	**0.89**	0.77	3	24	35	34	52	102	**2**	18	164	62	62	196	778	**6**
MAGIC	0.82	0.85	0.83	0.83	0.81	**0.89**	0.84	7	14	32	32	53	163	**5**	37	75	62	62	104	1222	**13**
DIABETES	0.74	0.74	0.73	0.72	0.67	**0.76**	0.72	18	**4**	8	8	**4**	10	7	79	13	14	14	**6**	55	25
CHESS	0.94	0.99	0.98	0.99	0.96	**1.00**	0.93	7	10	28	23	10	36	**3**	15	42	54	44	36	278	**9**
BANKNOTE	0.96	0.99	0.99	0.99	0.98	**1.00**	0.98	29	8	27	23	**4**	11	**4**	71	21	52	44	12	37	**8**
TICTACTOE	0.97	**1.00**	0.92	0.92	0.94	0.72	0.95	33	9	49	57	10	73	**8**	117	**28**	97	112	**28**	399	29

Table 12: AUC scores and rule set complexities expressed in the total number of rules and conditions, for different rule learning methods across datasets. TURS excels on most datasets in the obtained AUC scores. However, it constructs rule sets containing lots of rules and conditions, which makes it hard to interpret. In contrast, CG can construct very sparse rule sets that are still accurate, outperforming the classic rule learning methods and CLASSY. Our method R2N-Tab obtains good results on the larger datasets heloc, house, adult and magic, constructing sparse rule sets that have still accurate classification performance. On the smaller datasets, our method struggles to find the best trade-off between predictive performance and rule set complexity, due to the limited number of examples.

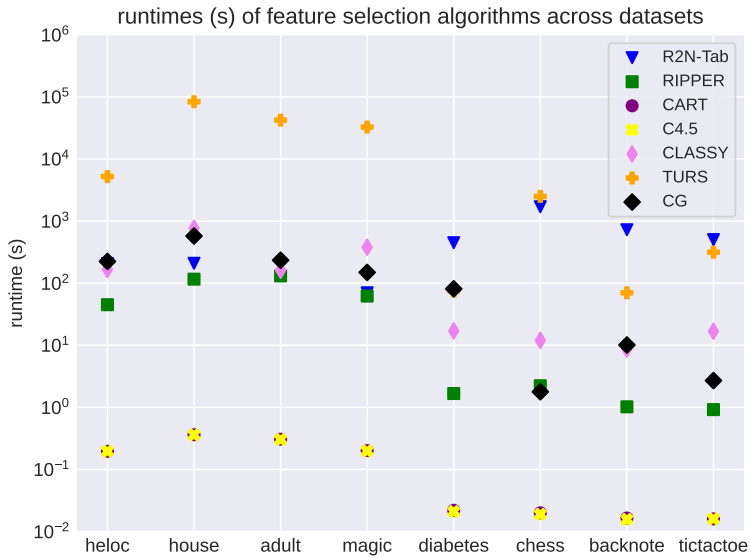


Figure 10: Running times per fold in seconds per dataset, comparing different rule learning algorithms. R2N-Tab takes longer on the smaller datasets to construct rule sets compared to the other rule learners. On the bigger datasets, R2N-Tab takes a comparable execution time compared to the other rule learners.

Chapter 6

Conclusion

6.1 Summary

In this thesis project, we proposed an extension to DR-Net, an existing approach where a two-layer neural network architecture can be mapped to a set of rules. We reused the two-layer architecture and added a deep learning layer for feature selection in front. The layer indicates features that do not affect the predictive performance of the classifier by labeling them with a negative weight, which results in the features being "canceled out" such that they are not included in the rule sets. The main objective of the thesis project was to learn a model that best fits a trade-off between predictive performance and model complexity, with as few epochs as possible. We developed a three-layer neural network architecture consisting of the CancelOut Layer explicitly for feature selection, and DR-Net afterward for constructing rule sets. We build a global optimization heuristic (17) that can find the Pareto optimal solution, maximizing the predictive performance while minimizing the model complexity. We built an experiment (5.1) comparing the original DR-Net to our new proposed method R2N-Tab. We empirically showed that the additional layer explicitly for feature selection resulted in reduced model complexity, while the predictive performance was harmed minimally.

To evaluate our feature selection layer, we compared our method R2N-Tab to well-known embedded feature selection approaches, that can be used for feature set reduction and classification at the same time. We developed an experiment (5.2) comparing the feature set reduction scores with the obtained classification performance. Compared to, e.g., Gradient Boosting, R2N-Tab further reduces the model complexity, while maintaining good predictive performance. Overall, we showed that feature selection layer finds the best fit between removing features and maintaining predictive performance on most datasets, removing only irrelevant features.

To position our method R2N-Tab as a rule learner, we compared it to related approaches in this domain in the final experiment (5.3). We showed that our method outperforms most rule learning approaches on the large datasets heloc, house, adult, and magic. We obtained similar results compared to the Column Generation method, which finds, just like our method, very sparse rule sets with still good predictive performance. On the smaller datasets, we observed that our method struggled to find a good fit between predictive performance and model complexity. We argued that this is due to the limited number of examples, complicating training a robust deep learning model.

6.2 Future work

As the original two-layer neural network architecture focuses on binary classification, our research was also limited to this domain. Consequently, in future work, the extension to multiclass classification could be made, allowing for a broader use of our proposed method. This could be established by creating output nodes for each class in the corresponding dataset and applying a softmax function afterward, that outputs the predicted class. Due to time resources, we kept the original two-layer architecture.

As can be observed from the experiment performed in 5.3, our method struggles to find the best fit between predictive performance and model complexity on smaller datasets, that have a limited number of examples. For improving the performance of our deep learning model on smaller datasets as future work, several techniques could be used. For example, one of these techniques includes data augmentation, which generates new data based on the existing data. As a result, datasets increase in the number of examples, which could be beneficial for the performance of our method.

Chapter 7

Appendix A

In this chapter, an overview of configurations used in all experiments is provided for reproducibility. For the optimization of neural net weights the Adam optimizer was used. For computing the loss, the BCELoss was used. All results were averaged over 5 runs. In Table 12, the configurations for hyper parameteres that were constant in all experiments are shown. The batch sizes used for each dataset are also constant in all experiments. For the larger datasets, heloc, house, adult, and magic, we use a batch size of 400. For the smaller datasets, diabetes, chess, banknote, and tictactoe, we use a batch size of 40.

hyper parameter	**configuration**	**description**
out_features	1	number of output nodes
device	'cpu'	hardware utilization
lr	1e-2	DR-Net learning rate
lr_cancel	5e-3	CancelOut learning rate
and_lam	1e-2	Rules Layer sparsity strength
or_lam	1e-5	OR Layer sparsity strength
num_alter	500	alternating Rules Layer/OR Layer

Figure 11: Global configurations used in all experiments.

7.1 Experiment 1

In the first experiment, we compared R2N-Tab to the original method DR-Net. We started with num_rules set to 50, and used λ_3 = 1e-2, λ_3 = 1e-4, and λ_3 = 1e-6. For the remaining hyper parameters, we used the configurations as described in Table 12.

7.2 Experiment 2

In the second experiment, we compared R2N-Tab to different embedded feature selection methods. For SVM, we used a linear kernel, which allows us to perform a classification task. For the other feature selection methods, we used their default parameters. For our method R2N-Tab, we used 1000 epochs on most datasets. The chess dataset however required 6000 epochs. For the λ_3 coefficients, we used the best values per dataset as found in experiment 1.

7.3 Experiment 3

In the final experiment, we design the Pareto front for the different rule learning approaches. A couple of rule learners include a max_depth constraint, that corresponds with the model complexity. We used the configurations as shown in Table 12.

rule learner	**depth**
RIPPER	[50, 75, 100, 200, 300]
CART	[2, 3, 5, 7, 8]
C4.5	[2, 3, 5, 7, 8]
CLASSY	[1, 2, 5, 7, 9]

Figure 12: Global configurations used in all experiments.

For the TURS and CG approaches, no max_depth constraint is included. For our method R2N-Tab, it depends on the dataset that is used. The configurations we used for our method are shown in Table 13.

dataset	**configuration**
heloc	[10, 25, 40, 55, 70]
house	[20, 35, 50, 65, 80]
adult	[25, 40, 55, 70, 95]
magic	[60, 75, 90, 105, 120]
diabetes	[110, 120, 140, 160, 180]
chess	[120, 130, 140, 160, 180]
banknote	[110, 120, 140, 160, 180]
tictactoe	[110, 120, 140, 160, 180]

Figure 13: Global configurations used in all experiments.

Bibliography

[1] H. Abdi and L. J. Williams. Principal component analysis. *WIREs Computational Statistics* **2**, pages 433–459, 2010.

[2] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer* **49**, pages 54–63, 2016.

[3] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation **abs/1308.3432**. *CoRR*, 2013.

[4] V. Borisov, J. Haug, and G. Kasneci. Cancelout: A layer for feature selection in deep neural networks. In *Artificial Neural Networks and Machine Learning* **11728**, pages 72–83. ICANN, 2019.

[5] L. Breiman, J. H. Freidman, R. A. Olshen, and C. J. Stone. Cart: Classification and regression trees. CRC press, 1984.

[6] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers and Electrical Engineering* **40**, pages 16–28, 2014.

[7] P. Clark and R. Boswell. Rule induction with CN2: some recent improvements. In *Machine Learning* **482**, pages 151–163. Springer, 1991.

[8] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning* **3**, pages 261–283, 1989.

[9] W. W. Cohen. Fast effective rule induction. In *Machine Learning Proceedings*, pages 115–123. Elsevier, 1995.

[10] S. Dash, O. Günlük, and W. Wei. Boolean decision rules via column generation. In *NeurIPS* **31**, pages 4660–4670, 2018.

[11] J. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics* **29**, pages 1189–1232, 2001.

[12] J. Fürnkranz and T. Kliegr. A brief overview of rule learning. In *Rule Technologies: Foundations, Tools, and Applications* **9202**, pages 54–69. Springer, 2015.

[13] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics* **37**, pages 362–386, 2020.

[14] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *NeurIPS*, 2022.

[15] J. Hühn and E. Hüllermeier. FURIA: an algorithm for unordered fuzzy rule induction. *Data Mining and Knowledge Discovery* **19**, pages 293–319, 2009.

[16] H. Lakkaraju, S. Bach, and J. Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1675–1684. ACM, 2016.

[17] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature* **521**, pages 436–44, 2015.

[18] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu. Feature selection: A data perspective. *ACM Computing Surveys* **50**, pages 94:1–94:45, 2018.

[19] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 1998.

[20] C. Nguyen, Y. Wang, and H. Nguyen. Random forest classifier combined with feature selection for breast cancer diagnosis and prognostic. *Journal of Biomedical Science and Engineering* **6**, pages 551–560, 2013.

[21] M. Nguyen and F. de la Torre. Optimal feature selection for support vector machines. *Pattern Recognition* **43**, pages 584–591, 2010.

[22] D. W. Otter, J. R. Medina, and J. K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems* **32**, pages 604–624, 2021.

[23] H. M. Proença and M. van Leeuwen. Interpretable multiclass classification by MDL-based rule lists. *Information Sciences* **512**, pages 1372–1393, 2020.

[24] L. Qiao, W. Wang, and B. Lin. Learning accurate and interpretable decision rule sets from neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence*, pages 4303–4311. AAAI Press, 2021.

[25] J. Quinlan. C4.5: programs for machine learning. Elsevier, 2014.

[26] M. Sahakyan, Z. Aung, and T. Rahwan. Explainable artificial intelligence for tabular data: A survey. *IEEE Access*, 9:135392–135422, 2021.

[27] I. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science* **2**, page 160, 2021.

[28] P. Shinde and S. Shah. A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, 2018.

[29] H. X. Tan, C. H. D. Teo, P. S. Ang, W. P. C. Loke, M. Y. Tham, S. H. Tan, B. L. S. Soh, P. Q. B. Foo, Z. J. Ling, W. L. J. Yip, Y. Tang, J. Yang, K. H. A. Tung, and S. R. Dorajoo. Combining machine learning with a rule-based algorithm to detect and identify related entities of documented adverse drug reactions on hospital discharge summaries. *Drug Safety* **45**, pages 853–862, 2022.

[30] A. Tharwat, T. Gaber, A. Ibrahim, and A. Hassanien. Linear discriminant analysis: A detailed tutorial. *AI Communications* **30**, pages 169–190, 2017.

[31] A. Voulodimos, N. Doulamis, A. D. Doulamis, and E. Protopapadakis. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience* **2018**, pages 7068349:1–7068349:13, 2018.

[32] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl, and P. MacNeille. A bayesian framework for learning rule sets for interpretable classification. *Journal of Machine Learning Research* **18**, pages 70:1–70:37, 2017.

[33] H. Yang, C. Rudin, and I. Margo. Scalable bayesian rule lists. In *Proceedings of the 34th International Conference on Machine Learning (ICML)* **70**, pages 3921–3930. PMLR, 2017.

[34] L. Yang and M. van Leeuwen. Truly unordered probabilistic rule sets for multi-class classification. In *Machine Learning and Knowledge Discovery in Databases*, pages 87–103. Springer Nature Switzerland, 2023.

[35] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017.

[36] G. Zhang and A. Gionis. Diverse rule sets. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1532–1541. ACM, 2020.