RRL: A SCALABLE CLASSIFIER FOR INTERPRETABLE RULE-BASED REPRESENTATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Rule-based models, e.g., decision trees, are widely used in scenarios demanding high model interpretability for their transparent inner structures and good model expressivity. However, rule-based models are hard to optimize, especially on large data sets, due to their discrete parameters and structures. Ensemble methods and fuzzy/soft rules are commonly used to tackle these issues, but they sacrifice the model interpretability. In this paper, we propose a new classifier, named Rule-based Representation Learner (RRL), that automatically learns interpretable non-fuzzy rules for data representation. To train the non-differentiable RRL effectively, we project it to a continuous space and propose a novel training method, called Gradient Grafting, that can directly optimize the discrete model using gradient descent. An improved design of logical activation functions is also devised to increase the scalability of RRL and enable it to discretize the continuous features end-to-end. Exhaustive experiments on 9 small and 4 large data sets show that RRL outperforms the competitive approaches, has low complexity close to the simple decision trees, and is rational for its main technical contributions.

1 INTRODUCTION

Although Deep Neural Networks (DNNs) have achieved impressive results in various machine learning tasks (Goodfellow et al., 2016), rule-based models, benefiting from their transparent inner structures and good model expressivity, still play an important role in domains demanding high model interpretability, such as medicine, finance, and politics (Doshi-Velez & Kim, 2017). In practice, rule-based models can easily provide explanations for users to earn their trust and help protect their rights (Molnar, 2019; Lipton, 2016). By analyzing the learned rules, practitioners can understand the decision mechanism of models, and use their knowledge to improve or debug the models (Chu et al., 2018). Moreover, even if post-hoc methods can provide interpretations for DNNs, the interpretations from rule-based models are more faithful and specific (Murdoch et al., 2019). However, conventional rule-based models are hard to optimize, especially on large data sets, due to their discrete parameters and structures, which limits their application scope. To take advantage of rule-based models in more scenarios, we urgently need to improve their scalability.

Studies in recent years provide some solutions to improve conventional rule-based models in different aspects. Ensemble methods and soft/fuzzy rules are proposed to improve the performance and scalability of rule-based models but at the cost of model interpretability (Ke et al., 2017; Breiman, 2001; Irsoy et al., 2012). Bayesian framework is also leveraged to more reasonably restrict and adjust the structures of rule-based models (Letham et al., 2015; Wang et al., 2017; Yang et al., 2017). However, due to the non-differentiable model structure, they have to use methods like MCMC or Simulated Annealing, which could be time-consuming for large models. Another way to improve rule-based models is to let a high-performance but complex model (e.g., DNN) teach a rule-based model (Frosst & Hinton (2017); Ribeiro et al. (2016)). However, to learn from the complex model, it requires soft rules, or the fidelity of the student model is not guaranteed. Wang et al. (2020) try to extract hierarchical rule sets from a tailored neural network. Although the extracted rules could behave differently from the neural network when the network is large, combined with binarized networks (Courbariaux et al., 2015), it inspires us that we can search for the discrete solution of rule-based models in a continuous space and leverage optimization methods like gradient descent.

In this paper, we propose a novel rule-based model, named **Rule-based Representation Learner (RRL)** (see Figure 1a), which owns three key technical contributions: (i) To achieve **model transparency**, RRL is formulated as a hierarchical model, with layers supporting both conjunction and disjunction operations. This paves the way for automatically learning interpretable non-fuzzy rules for data representation and classification. (ii) To facilitate **training effectiveness**, RRL exploits a novel gradient-based discrete model training method, Gradient Grafting, that directly optimizes the discrete model and uses the gradient information at both continuous and discrete points to suit more scenarios. (iii) To ensure **data scalability**, RRL utilizes improved logical activation functions to handle high-dimensional features. By further combining the improved logical activation functions with a tailored feature binarization layer, it realizes the continuous feature discretization in an end-to-end manner.

We conduct experiments on 9 small data sets and 4 large data sets to validate the advantages of our model over other representative classification models. The benefits of the model's key components are also verified by experiments.

2 Related Work

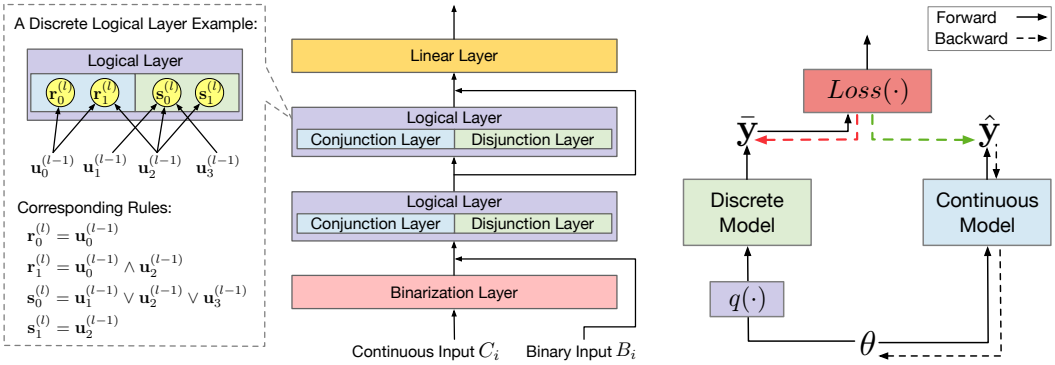
Rule-based Models. Decision tree, rule list, and rule set are the widely used structures in rule-based models. For their discrete parameters and non-differentiable structures, we have to train them by employing various heuristic methods (Quinlan, 1993; Breiman, 2017; Cohen, 1995) which may not find the globally best solution or a solution with close performance. Alternatively, train them with search algorithms (Wang et al., 2017; Angelino et al., 2017), which could take too much time on large data sets. In recent studies, Bayesian frameworks are leveraged to restrict and adjust model structure more reasonably (Letham et al., 2015; Wang et al., 2017; Yang et al., 2017). Lakkaraju et al. (2016) learn independent if-then rules with smooth local search. Using algorithmic bounds and efficient data structures, Angelino et al. (2017) try to accelerate the learning of certifiably optimal rule lists. However, except heuristic methods, most existing rule-based models need frequent itemsets mining and/or long-time searching, which limits their applications. Moreover, it is hard for these rule-based models to get comparable performance with complex models like Random Forest.

Ensemble models like Random Forest (Breiman, 2001) and Gradient Boosted Decision Trees (Ke et al., 2017), have better performance than the single rule-based model. However, for the decision is made by hundreds of models, ensemble models are commonly not considered as interpretable models (Hara & Hayashi, 2016). Soft or fuzzy rules are also used to improve model performance (Irsoy et al., 2012; Ishibuchi & Yamamoto, 2005), but non-discrete rules are much harder to understand than discrete ones. Other studies try to teach the rule-based model by a complex model, e.g., DNN, or extract rule-based models from complex models (Frosst & Hinton, 2017; Ribeiro et al., 2016; Wang et al., 2020). However, the fidelity of the student model or extracted model is not guaranteed.

Gradient-based Discrete Model Training. The gradient-based discrete model training methods are mainly proposed to train binary or quantized neural networks for network compression and acceleration. Courbariaux et al. (2015; 2016) propose to use Straight-Through Estimator (STE) for binary neural network training and achieve empirical success. However, STE requires gradient information at discrete points, which limits its applications. ProxQuant (Bai et al., 2018) formulates quantized network training as a regularized learning problem and optimizes it via the prox-gradient method. ProxQuant can use gradients at non-discrete points, but cannot directly optimize for the discrete model. The Random Binarization (RB) method (Wang et al., 2020) trains a neural network with random binarization for its weights to ensure the discrete and continuous model behave samely. However, when the model is large, the differences between the discrete and continuous model are inevitable. Our method, i.e., Gradient Grafting, is different from all the aforementioned works in using gradient information of both discrete and continuous models in each backpropagation.

3 Rule-based Representation Learner

Notation Description. Let $\mathbb{D} = \{(X_1, \mathbf{y}_1), \ldots, (X_N, \mathbf{y}_N)\}$ denote the training data set with N instances, where X_i is the observed feature vector of the i-th instance with the j-th entry as $X_{i,j}$, and $\mathbf{y}_i$ is the corresponding class label, $i \in \{1, \ldots, N\}$. Each feature value can be either discrete or continuous. All the classes take discrete values, and the number of class labels is denoted by



(a) (b)

Figure 1: (a) A Rule-based Representation Learner example. The dashed box shows an example of a discrete logical layer and its corresponding rules. (b) A simplified computation graph of Gradient Grafting. Arrows with solid lines represent forward pass while arrows with dashed lines represent backpropagation. The green arrow denotes the grafted gradient, a copy of the gradient represented by the red arrow. After grafting, there exists a backward path from loss function to the parameter θ.

M. We use one-hot encoding to represent all discrete features as binary features. Let $C_i \in \mathbb{R}^m$ and $B_i \in \{0,1\}^b$ denote the continuous feature vector and the binary feature vector of the i-th instance respectively. Therefore, $X_i = C_i \oplus B_i$, where $\oplus$ represents the operator that concatenates two vectors. Throughout this paper, we use 1 (True) and 0 (False) to represent the two states of a Boolean variable. Thus each dimension of a binary feature vector corresponds to a Boolean variable.

Overall Structure. A Rule-based Representation Learner (RRL), denoted by $\mathcal{F}$, is a hierarchical model consisting of three different types of layers. Each layer in RRL not only contains a specific number of nodes, but also has trainable edges connected with its previous layer. Let $\mathcal{U}^{(l)}$ denote the l-th layer of RRL, $\mathbf{u}_j^{(l)}$ indicate the j-th node in the layer, and $\boldsymbol{n}_l$ represent the corresponding number of nodes, $l \in \{0, \ldots, L\}$. The output of the l-th layer is a vector containing the values of all the nodes in the layer. For ease of expression, we denote this vector by $\mathbf{u}^{(l)}$. There are only one binarization layer, i.e., $\mathcal{U}^{(0)}$, and one linear layer, i.e., $\mathcal{U}^{(L)}$, in RRL, but the number of middle layers, i.e., logical layers, can be flexibly adjusted according to the specific situation. The logical layers mainly aim to learn the non-linear part of the data while the linear layer aims to learn the linear part. One example of RRL is shown in Figure 1a.

When we input the i-th instance to RRL, the binarization layer will first binarize the continuous feature vector C_i into a new binary vector $\bar{C}_i$. Then, $\bar{C}_i$ and B_i are concatenated together as $\mathbf{u}^{(0)}$ and inputted to the first logical layer. The logical layers are designed to automatically learn data representations using logical rules, and the stacked logical layers can learn rules in more complex forms. After going through all the logical layers, the output of the last logical layer can be considered as the new feature vector to represent the instance, wherein each feature corresponds to one rule formulated by the original features. As such, the whole RRL is composed of a feature learner and a linear classifier (linear layer). Moreover, the skip connections in RRL can skip unnecessary logical layers. In what follows, the details of these components will be elaborated.

3.1 Logical Layer

Considering the binarization layer needs the help of its following logical layer to binarize features in an end-to-end way, we introduce logical layers first. As mentioned above, logical layers can learn data representations using logical rules automatically. To achieve this, logical layers are designed to have a discrete version and a continuous version. The discrete version is used for training, testing and interpretation while the continuous version is only used for training. It is worth noting that the discrete RRL indicates the parameter weights of logical layers take discrete values (i.e., 0 or 1).

Discrete Version. One logical layer consists of one conjunction layer and one disjunction layer. In discrete version, let $\mathcal{R}^{(l)}$ and $\mathcal{S}^{(l)}$ denote the conjunction and disjunction layer of $\mathcal{U}^{(l)}$ ($l \in \{1, 2, \ldots, L-1\}$) respectively. We denote the i-th node in $\mathcal{R}^{(l)}$ by $\mathbf{r}_i^{(l)}$, and the i-th node in $\mathcal{S}^{(l)}$

by $\mathbf{s}_i^{(l)}$. Specifically speaking, node $\mathbf{r}_i^{(l)}$ corresponds to the conjunction of nodes in the previous layer connected with $\mathbf{r}_i^{(l)}$, while node $\mathbf{s}_i^{(l)}$ corresponds to the disjunction of nodes in previous layer connected with $\mathbf{s}_i^{(l)}$. Formally, the two types of nodes are defined as follows:

$$\mathbf{r}_i^{(l)} = \bigwedge_{W_{i,j}^{(l,0)}=1} \mathbf{u}_j^{(l-1)}, \qquad \mathbf{s}_i^{(l)} = \bigvee_{W_{i,j}^{(l,1)}=1} \mathbf{u}_j^{(l-1)}, \tag{1}$$

where $W^{(l,0)}$ denote the adjacency matrix of the conjunction layer $\mathcal{R}^{(l)}$ and the previous layer $\mathcal{U}^{(l-1)}$, and $W_{i,j}^{(l,0)} \in \{0,1\}$. $W_{i,j}^{(l,0)} = 1$ indicates there exists an edge connecting $\mathbf{r}_i^{(l)}$ to $\mathbf{u}_j^{(l-1)}$, otherwise $W_{i,j}^{(l,0)} = 0$. Similarly, $W^{(l,1)}$ is the adjacency matrix of the disjunction layer $\mathcal{S}^{(l)}$ and $\mathcal{U}^{(l-1)}$. Similar to neural networks, we regard these adjacency matrices as the weight matrices of logical layers. $\mathbf{u}^{(l)} = \mathbf{r}^{(l)} \oplus \mathbf{s}^{(l)}$, where $\mathbf{r}^{(l)}$ and $\mathbf{s}^{(l)}$ are the outputs of $\mathcal{R}^{(l)}$ and $\mathcal{S}^{(l)}$ respectively.

The function of the logical layer is similar to the notion "level" in Wang et al. (2020). However, one level in that work, which actually consists of two layers, can only represent rules in Disjunctive Normal Form (DNF), while two logical layers can represent rules in DNF and Conjunctive Normal Form (CNF) at the same time. Connecting nodes in $\mathcal{R}^{(l)}$ with nodes in $\mathcal{S}^{(l-1)}$, we get rules in CNF, while connecting nodes in $\mathcal{S}^{(l)}$ with nodes in $\mathcal{R}^{(l-1)}$, we get rules in DNF. The flexibility of logical layer is quite important. For instance, the length of CNF rule $(\boldsymbol{a}_1 \vee \boldsymbol{a}_2) \wedge \cdots \wedge (\boldsymbol{a}_{2n-1} \vee \boldsymbol{a}_{2n})$ is $2n$, but the length of its corresponding DNF rule $(\boldsymbol{a}_1 \wedge \boldsymbol{a}_3 \cdots \wedge \boldsymbol{a}_{2n-1}) \vee \cdots \vee (\boldsymbol{a}_2 \wedge \boldsymbol{a}_4 \cdots \wedge \boldsymbol{a}_{2n})$ is $n \cdot 2^n$, which means layers that only represent DNF can hardly learn this CNF rule.

Continuous Version. Although the discrete logical layers have good interpretability, they are hard to train for their discrete parameters and non-differentiable structures. Inspired by the training process of binary neural networks that searches the discrete solution in a continuous space, we extend the discrete logical layer to a continuous version.

Let $\hat{\mathcal{U}}^{(l)}$ denote the continuous logical layer, and $\hat{\mathcal{R}}^{(l)}$ and $\hat{\mathcal{S}}^{(l)}$ denote the continuous conjunction and disjunction layer respectively, $l \in \{1, 2, \ldots, L-1\}$. Let $\hat{W}^{(l,0)}$ and $\hat{W}^{(l,1)}$ denote the weight matrices of $\hat{\mathcal{R}}^{(l)}$ and $\hat{\mathcal{S}}^{(l)}$ respectively. $\hat{W}_{i,j}^{(l,0)}, \hat{W}_{i,j}^{(l,1)} \in [0,1]$. To make the whole Equation 1 differentiable, we leverage the logical activation functions proposed by Payani & Fekri (2019):

$$\textit{Conj}(\mathbf{h}, W_i) = \prod_{j=1}^{n} F_c(\mathbf{h}_j, W_{i,j}), \qquad \textit{Disj}(\mathbf{h}, W_i) = 1 - \prod_{j=1}^{n} (1 - F_d(\mathbf{h}_j, W_{i,j})), \tag{2}$$

where $F_c(h, w) = 1 - w(1-h)$ and $F_d(h, w) = h \cdot w$. In Equation 2, if $\mathbf{h}$ and W_i are both binary vectors, then $\textit{Conj}(\mathbf{h}, W_i) = \bigwedge_{W_{i,j}=1} \mathbf{h}_j$ and $\textit{Disj}(\mathbf{h}, W_i) = \bigvee_{W_{i,j}=1} \mathbf{h}_j$. $F_c(h, w)$ and $F_d(h, w)$ decide how much $\mathbf{h}_j$ would affect the operation according to $W_{i,j}$. If $W_{i,j} = 0$, $\mathbf{h}_j$ would have no effect on the operation.

After using continuous weights and logical activation functions, the nodes in $\hat{\mathcal{R}}^{(l)}$ and $\hat{\mathcal{S}}^{(l)}$ are defined as follows:

$$\hat{\mathbf{r}}_i^{(l)} = \textit{Conj}(\hat{\mathbf{u}}^{(l-1)}, \hat{W}_i^{(l,0)}), \qquad \hat{\mathbf{s}}_i^{(l)} = \textit{Disj}(\hat{\mathbf{u}}^{(l-1)}, \hat{W}_i^{(l,1)}) \tag{3}$$

Now the whole logical layer is differentiable and can be trained by gradient descent. However, above logical activation functions suffer from the serious vanishing gradient problem. The main reason can be found by analyzing the partial derivative of each node w.r.t. its directly connected weights and w.r.t. its directly connected nodes as follows:

$$\frac{\partial \hat{\mathbf{r}}_i^{(l)}}{\partial \hat{W}_{i,j}^{(l,0)}} = (\hat{\mathbf{u}}_j^{(l-1)} - 1) \cdot \prod_{k \neq j} F_c(\hat{\mathbf{u}}_k^{(l-1)}, \hat{W}_{i,k}^{(l,0)}), \qquad \frac{\partial \hat{\mathbf{r}}_i^{(l)}}{\partial \hat{\mathbf{u}}_j^{(l-1)}} = \hat{W}_{i,j}^{(l,0)} \cdot \prod_{k \neq j} F_c(\hat{\mathbf{u}}_k^{(l-1)}, \hat{W}_{i,k}^{(l,0)}) \tag{4}$$

Due to $\hat{\mathbf{u}}_k^{(l-1)}$ and $\hat{W}_{i,k}^{(l,0)}$ are in the range $[0,1]$, the values of $F_c(\cdot)$ in Equation 4 are in the range $[0,1]$ as well. If $\mathbf{n}_{l-1}$ is large and most of the values of $F_c(\cdot)$ are not 1, then the derivative is close to 0 because of the multiplications (See Appendix A for the analysis of $\hat{\mathbf{s}}_i^{(l)}$). Wang et al. (2020) try to use weight initialization to make $F_c(\cdot)$ close to 1 at the beginning. However, when dealing with hundreds of features, the vanishing gradient problem is still inevitable.

We found that using the multiplications to simulate the logical operations in Equation 2 is the main reason for vanishing gradients and propose improved logical activation functions. One straightforward thought is to convert multiplications into additions using logarithm, e.g., $\log(\prod_{j=1}^{n} F_c(\mathbf{h}_j, W_{i,j})) = \sum_{j=1}^{n} \log(F_c(\mathbf{h}_j, W_{i,j}))$. However, after taking the logarithm, the logical activation functions in Equation 2 cannot keep the characteristics of logical operations any more, and the ranges of $Conj(\cdot)$ and $Disj(\cdot)$ are not $[0, 1]$. To deal with this problem, we need a projection function to fix it. Apparently, the inverse function of $\log(x)$, i.e., e^x, is not suitable.

For the projection function g, three conditions must be satisfied: (i) $g(0) = e^0 = 1$. (ii) $\lim_{x\to-\infty} g(x) = \lim_{x\to-\infty} e^x = 0$. (iii) $\lim_{x\to-\infty} \frac{e^x}{g(x)} = 0$. Condition (i) and (ii) aim to keep the range and tendency of logical activation functions. Condition (iii) aims to lower the speed of approaching zero when $x \to -\infty$. In this work, we choose $g(x) = \frac{-1}{-1+x}$ as the projection function, and the improvement of logical activation functions can be summarized as the function $\mathbb{P}(v) = \frac{-1}{-1+\log(v)}$. The improved conjunction function $\textit{Conj}_+$ and disjunction function $\textit{Disj}_+$ are given by:

$$\textit{Conj}_+(\mathbf{h}, W_i) = \mathbb{P}(\prod_{j=1}^{n}(F_c(\mathbf{h}_j, W_{i,j}) + \epsilon)), \qquad \textit{Disj}_+(\mathbf{h}, W_i) = 1 - \mathbb{P}(\prod_{j=1}^{n}(1 - F_d(\mathbf{h}_j, W_{i,j}) + \epsilon)), \tag{5}$$

where ϵ is a small constant, e.g., 10^{-10}. The improved logical activation functions can avoid the vanishing gradient problem in most scenarios and are much more scalable than the originals. Moreover, considering that $\frac{d\mathbb{P}(v)}{dv} = \frac{\mathbb{P}^2(v)}{v}$, when n in Equation 5 is extremely large, $\frac{d\mathbb{P}(v)}{dv}$ may be very close to 0 due to $\mathbb{P}^2(v)$. One trick to deal with it is replacing $\frac{\mathbb{P}^2(v)}{v}$ with $\frac{\mathbb{P}(\mathbb{P}^2(v))}{v}$ for $\mathbb{P}$ can lower the speed of approaching 0 while keeping the value range and tendency.

3.2 Binarization Layer

Binarization layer is mainly used to divide the continuous feature values into several bins. By combining one binarization layer and one logical layer, we can automatically choose the appropriate bins for feature discretization (binarization), i.e., binarizing features in an end-to-end way.

For the j-th continuous feature, there are k lower bounds $(\mathcal{L}_{j,1}, \ldots, \mathcal{L}_{j,k})$ and k upper bounds $(\mathcal{H}_{j,1}, \ldots, \mathcal{H}_{j,k})$. All these bounds are randomly selected (e.g., from uniform distribution) in the value range of the j-th continuous feature, and these bounds are not trainable. When inputting one continuous feature vector $\boldsymbol{c}$, the binarization layer will check if $\boldsymbol{c}_j$ satisfies the bounds and get the following binary vector:

$$Q_j = [q(\boldsymbol{c}_j - \mathcal{L}_{j,1}), \ldots, q(\boldsymbol{c}_j - \mathcal{L}_{j,k}), q(\mathcal{H}_{j,1} - \boldsymbol{c}_j), \ldots, q(\mathcal{H}_{j,k} - \boldsymbol{c}_j)], \tag{6}$$

where $q(x) = \mathbf{1}_{x>0}$. If we input the i-th instance, i.e., $\boldsymbol{c} = C_i$, then $\bar{C}_i = Q_1 \oplus Q_2 \cdots \oplus Q_m$ and $\mathbf{u}^{(0)} = \bar{C}_i \oplus Bi$. After inputting $\mathbf{u}^{(0)}$ to the logical layer $\mathcal{U}^{(1)}$, the edge connections between $\mathcal{U}^{(1)}$ and $\mathcal{U}^{(0)}$ indicate the choice of bounds (bins). For example, if $\mathbf{r}_i^{(1)}$ is connected to the nodes corresponding to $q(\boldsymbol{c}_j - \mathcal{L}_{j,1})$ and $q(\mathcal{H}_{j,2} - \boldsymbol{c}_j)$, then $\mathbf{r}_i^{(1)}$ contains the bin $(\mathcal{L}_{j,1} < \boldsymbol{c}_j) \wedge (\boldsymbol{c}_j < \mathcal{H}_{j,2})$. If we replace $\mathbf{r}_i^{(1)}$ with $\mathbf{s}_i^{(1)}$ in the example, we can get $(\mathcal{L}_{j,1} < \boldsymbol{c}_j) \vee (\boldsymbol{c}_j < \mathcal{H}_{j,2})$. It should be noted that, in practice, if $\mathcal{L}_{j,1} \geq \mathcal{H}_{j,2}$, then $\mathbf{r}_i^{(1)} = 0$, and if $\mathcal{L}_{j,1} < \mathcal{H}_{j,2}$, then $\mathbf{s}_i^{(1)} = 1$. When using the continuous version, the weights of logical layers are trainable, which means we can choose bounds in an end-to-end way. For the number of bounds is $2k$ times of features, which could be large, only logical layers with improved logical activation functions are capable of choosing the bounds.

3.3 Gradient Grafting

Although RRL can be differentiable with the continuous logical layers, it is challenging to search for a discrete solution in a continuous space. To tackle this problem, one commonly used method is Straight-Through Estimator (STE) (Courbariaux et al. (2016)). The STE method needs gradients at discrete points to update the parameters. However, the gradients of RRL at discrete points have no useful information in most cases (See Appendix B). Therefore STE is not suitable for RRL. Other methods like ProxQuant (Bai et al. (2018)) and Random Binarization (Wang et al. (2020)) cannot directly optimize for the discrete model and be scalable at the same time.

Inspired by plant grafting, we propose a new training method, called Gradient Grafting, that can effectively train RRL. In stem grafting, one plant is selected for its roots, i.e., rootstock, and the other plant is selected for its stems, i.e., scion. By grafting, we obtain a new plant with the advantages of both two plants. In Gradient Grafting, the gradient of the loss function w.r.t. the output of discrete model is the scion, and the gradient of the output of continuous model w.r.t. the parameters of continuous model is the rootstock. Specifically, let θ denote the parameter vector and θ^t denote the parameter vector at step t. $q(\boldsymbol{x}) = \mathbf{1}_{\boldsymbol{x}>0.5}$ is the binarization function that binarizes each dimension of $\boldsymbol{x}$ with 0.5 as the threshold. Let $\hat{\mathbf{y}}$ and $\bar{\mathbf{y}}$ denote the output of the continuous model $\hat{\mathcal{F}}$ and discrete model $\mathcal{F}$ respectively, then $\hat{\mathbf{y}} = \hat{\mathcal{F}}(\theta^t, X)$, $\bar{\mathbf{y}} = \mathcal{F}(q(\theta^t), X)$. The parameters update with Gradient Grafting is formulated by:

$$\theta^{t+1} = \theta^t - \eta \frac{\partial L(\bar{\mathbf{y}})}{\partial \bar{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \theta^t}, \tag{7}$$

where η is the learning rate and $L(\cdot)$ is the loss function. One simplified computation graph of Gradient Grafting is shown in Figure 1b for intuitive understanding.

Gradient Grafting can directly optimize the loss of discrete models and use the gradient information at both continuous and discrete points, which overcomes the problems occurring in RRL training when using other gradient-based discrete model training methods.

3.4 Model Interpretation

After training with Gradient Grafting, the discrete RRL can be used for testing and interpretation. RRL is easy to interpret for we can simply consider it as a feature learner and a linear classifier. The binarization layer and logical layers are the feature learner, and they use logical rules to build and describe the new features. The linear classifier, i.e., the linear layer, makes decisions based on the new features. We can first find the important new features by the weights of the linear layer, then understand each new feature by analyzing its corresponding rules. The L1/L2 regularization can be used during training to search for an RRL with shorter rules. The dead nodes detection and redundant rules elimination proposed by Wang et al. (2020) can also be used for better interpretability.

4 Experiments

In this section, we conduct experiments to evaluate the proposed model and answer the following questions: (i) How is the classification performance of RRL compared to the competitive approaches? (ii) How is the model complexity of RRL? (iii) How is the convergence of Gradient Grafting compared to other gradient-based discrete model training methods?

4.1 Dataset Description and Experimental Settings

We took 9 small and 4 large public datasets to conduct our experiments, all of which are often used to test classification performance and model interpretability (Dua & Graff, 2017; Xiao et al., 2017; Anguita et al., 2013; Moro et al., 2016). Appendix C summarizes the statistics of these 13 datasets. Together they show the data diversity, ranging from 178 to 102944 instances, from 2 to 26 classes, and from 4 to 4714 original features.

We adopt the F1 score (Macro) as the classification performance metric since some of the data sets are imbalanced, i.e., the numbers of different classes are quite different. We adopt 5-fold cross-validation to evaluate the classification performance more fairly. When parameter tuning is required, 95% of the training set is used for training and 5% for validation. Considering that reused structures exist in rule-based models, e.g., one branch in Decision Tree can correspond to several rules, we use the total number of edges instead of the total length of all rules as the metric of model complexity for rule-based models.

4.2 Classification Performance

We compare the classification F1 score (Macro) of RRL, C4.5 (Quinlan, 1993), CART (Breiman, 2017), Scalable Bayesian Rule Lists (SBRL) (Yang et al., 2017), Concept Rule Sets (CRS) (Wang et al., 2020), Logistic Regression (LR) (Kleinbaum et al., 2002), Piecewise Linear Neural Network

Table 1: 5-fold cross validated F1 score of comparing models on 9 small and 4 large datasets.

Dataset	**RRL**	C4.5	CART	SBRL	CRS	LR	SVM	PLNN(MLP)	$\text{GBDT}_{e=100}$	$\text{RF}_{e=10}$	$\text{RF}_{e=100}$
adult	80.72	75.40	74.77	79.88	**80.95**	78.43	63.63	73.55	80.36	77.48	78.83
bank-marketing	**76.29**	71.24	70.21	72.67	73.34	69.81	66.78	72.40	75.28	69.89	72.01
banknote	**100.0**	98.45	97.85	94.44	94.93	98.82	**100.0**	**100.0**	99.48	99.11	99.19
chess	78.83	79.90	79.15	26.44	**80.21**	33.06	36.83	77.85	71.41	66.38	74.25
connect-4	**71.23**	61.66	61.24	48.54	65.88	49.87	50.17	64.55	64.45	61.95	62.72
letRecog	96.15	88.20	87.62	64.32	84.96	72.05	74.90	92.34	**96.51**	93.61	96.15
magic04	86.33	80.31	80.05	82.52	80.87	75.72	75.64	83.07	**86.67**	84.90	86.48
tic-tac-toe	**99.77**	91.70	94.21	98.39	**99.77**	98.12	98.07	98.26	99.19	94.85	98.37
wine	98.23	95.48	94.39	95.84	97.78	95.16	96.05	76.07	**98.44**	96.90	98.31
activity	98.17	94.24	93.35	11.34	5.05	98.47	98.67	98.27	**99.02**	96.93	97.80
dota	**60.12**	52.08	51.91	34.83	56.31	59.34	57.76	59.46	58.81	54.04	57.16
facebook	**90.27**	80.76	81.50	31.16	11.38	88.62	87.20	89.43	85.51	82.88	86.85
fashion	89.01	80.49	79.61	47.38	66.92	84.53	84.46	89.36	**89.91**	85.76	88.05
Average	**86.55**	80.76	80.45	60.60	69.10	77.08	76.17	82.66	85.00	81.90	84.32

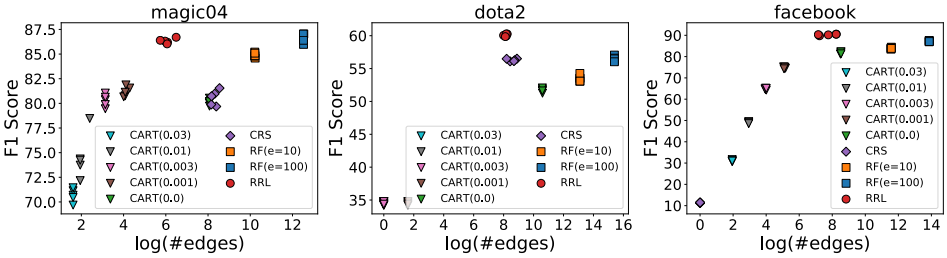


Figure 2: Scatter plot of F1 score against log(#edges) for 3 datasets for all 5 folds.

(PLNN) (Chu et al., 2018), Support Vector Machines (SVM) (Scholkopf & Smola, 2001) with linear, RBF or Ploy kernel, Gradient Boosted Decision Tree (Ke et al., 2017) with 100 estimators ($\text{GBDT}_{e=100}$), and Random Forest (Breiman, 2001) with 10 estimators ($\text{RF}_{e=10}$) and 100 estimators ($\text{RF}_{e=100}$). C4.5, CART, SBRL and CRS are all rule-based models, and LR is a linear model. These five models are considered as interpretable models. PLNN is a Multilayer Perceptron (MLP) that adopts piecewise linear activation functions, e.g., ReLU (Nair & Hinton, 2010). PLNN, SVM, GBDT and Random Forest are considered as complex models.

The results are shown in Table 1, and the first 9 data sets are small data sets while the last 4 are large data sets. We can observe that RRL performs well on almost all the data sets and gets the best results on 6 data sets. RRL outperforms other interpretable models and only two complex models, i.e., $\text{GBDT}_{e=100}$ and $\text{RF}_{e=100}$, have comparable results. Compared RRL with LR and other rule-based models, we can see RRL can fit both linear and non-linear data well. CRS performs well on small data sets but fails on large datasets due to the limitation of its logical activation functions and training method. Good results on both small and large data sets verify RRL has good scalability. Moreover, SBRL and CRS do not perform well on continuous feature data sets like *letRecog* and *magic04* for they need preprocessing to discretize continuous features, which may bring bias to the data sets. On the contrary, RRL overcomes this problem by discretizing features end-to-end.

4.3 Model Complexity

Interpretable models need to keep low model complexity while ensuring high accuracy. To show the relationships between accuracy and model complexity, we draw scatter plots of F1 score against log(#edges) for rule-based models in Figure 2 (see Appendix E for results on other data sets). The value in CART(0.03) denotes the complexity parameter used for Minimal Cost-Complexity Pruning (Breiman, 2017), and a higher value corresponds to a simpler tree. We can observe that RRL is much simpler than random forests, and its complexity is close to CRS and decision trees without pruning. Meanwhile, RRL has higher F1 scores than models with close complexity. It also indicates that RRL can make better use of rules than models using heuristic and ensemble methods.

4.4 Ablation Study

Training Method for Discrete Model. To show the effectiveness of Gradient Grafting for training RRL, we compare it with other representative gradient-based discrete model training methods, i.e.,

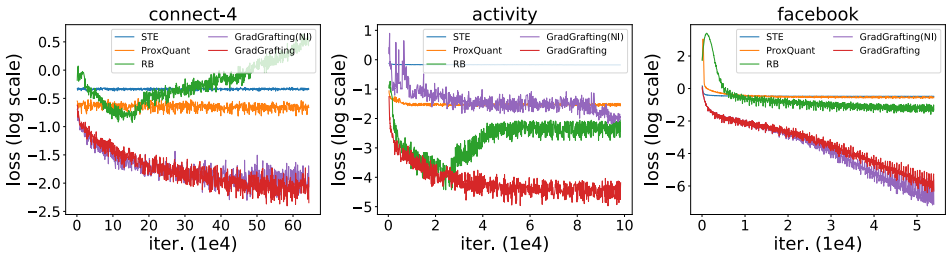


Figure 3: Training loss of 3 compared discrete model training methods and Gradient Grafting with or without improved logical activation functions on 3 data sets.

Weight	Rule
0.995	-122.5 < balance < 2606.1 ∧ marital = married ∧ campaign < 5 ∧ poutcome = success ∧ previous > 0
0.753	1757.2 < balance < 7016.7 ∧ marital = married ∧ contact = telephone ∧ 6 < day < 27 ∧ previous < 5
0.733	age > 36 ∧ balance < 7016.7 ∧ marital = married ∧ campaign < 5 ∧ pdays < 104 ∧ poutcome = success
0.731	36 < age < 60 ∧ balance > -122.5 ∧ campaign < 7 ∧ day > 22 ∧ pdays > 304 ∧ previous > 0
0.728	age > 28 ∧ -669.1 < balance < 5813.7 ∧ campaign < 6 ∧ pdays > 304 ∧ 0 < previous < 6

Figure 4: Logical rules obtained from RRL trained on the *bank-marketing* data set.

STE (Courbariaux et al., 2015; 2016), ProxQuant (Bai et al., 2018) and RB (Wang et al., 2020), by training RRL with the same structure. Hyperparameters are set to be the same for each method except exclusive hyperparameters, e.g., random binarization rate for RB, are fine-tuned. In Figure 3, we can see that the convergence of Gradient Grafting is faster and stabler than other methods on all data sets. As we mentioned in Section 3.3, RRL has little useful gradient information at discrete points, thus STE cannot converge. Due to the difference between discrete and continuous RRL, ProxQuant and RB cannot converge well as well.

Improved Logical Activation Functions. We also compare RRL trained by Gradient Grafting with or without improved logical activation functions. The results are also shown in Figure 3, and GradGrafting(NI) represents RRL using original logical activation functions instead of improved logical activation functions. We can observe that the original activation functions work well on small data sets but fail on the large data set *activity* while the improved activation functions work well on all data sets, which means the improved logical activation functions make RRL more scalable. It should be noted that GradGrafting(NI) works well on the large data set *facebook*, the reason is *facebook* is a very sparse data set, and the number of 1 in each binary feature vector is less than 30 (See Appendix A for detailed analyses).

4.5 Case Study

We show how the learned rules look like by case studies. In Figure 4, we show the learned rules, with high weights, of RRL trained on *bank-marketing* data set (See Appendix D for *fashion* data set). These rules are used to predict if the client will subscribe a term deposit by telesales. Different types of features are marked in different colors, e.g., purple for previous behaviours of the bank. We can clearly see that middle-aged married persons with low balance are more likely to subscribe a deposit, and the previous behaviour of the bank would also affect the client. Then the bank can change their strategies according to these rules.

5 Conclusion

We propose a new scalable classifier, named Rule-based Representation Learner (RRL), that can automatically learn interpretable rules for data representation and classification. For the particularity of RRL, we propose a new gradient-based discrete model training method, i.e., Gradient Grafting, that directly optimizes the discrete model. We also propose an improved design of logical activation functions to increase the scalability of RRL and make RRL capable of discretizing the continuous features end-to-end. Our experimental results show that RRL enjoys both high classification performance and low model complexity on data sets with different scales.

REFERENCES

Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *The Journal of Machine Learning Research*, 18 (1):8753–8830, 2017.

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Esann*, 2013.

Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. *arXiv preprint arXiv:1810.00861*, 2018.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Leo Breiman. *Classification and regression trees*. Routledge, 2017.

Lingyang Chu, Xia Hu, Juhua Hu, Lanjun Wang, and Jian Pei. Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. In *SIGKDD*, pp. 1244–1253. ACM, 2018.

William W Cohen. Fast effective rule induction. In *MLP*, pp. 115–123. Elsevier, 1995.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NeurIPS*, pp. 3123–3131, 2015.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Satoshi Hara and Kohei Hayashi. Making tree ensembles interpretable. *arXiv preprint arXiv:1606.05390*, 2016.

Ozan Irsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 1819–1822. IEEE, 2012.

Hisao Ishibuchi and Takashi Yamamoto. Rule weight specification in fuzzy rule-based classification systems. *IEEE transactions on fuzzy systems*, 13(4):428–435, 2005.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pp. 3146–3154, 2017.

David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. *Logistic regression*. Springer, 2002.

Himabindu Lakkaraju, Stephen H Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *SIGKDD*, pp. 1675–1684. ACM, 2016.

Benjamin Letham, Cynthia Rudin, Tyler H McCormick, David Madigan, et al. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3):1350–1371, 2015.

Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

Christoph Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/.

Sérgio Moro, Paulo Rita, and Bernardo Vala. Predicting social media performance metrics and evaluation of the impact on brand building: A data mining approach. *Journal of Business Research*, 69(9):3341–3351, 2016.

W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Interpretable machine learning: definitions, methods, and applications. *arXiv preprint arXiv:1901.04592*, 2019.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pp. 807–814, 2010.

Ali Payani and Faramarz Fekri. Learning algorithms via neural logic networks. *arXiv preprint arXiv:1904.01554*, 2019.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1558602402.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *SIGKDD*, pp. 1135–1144. ACM, 2016.

Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. A bayesian framework for learning rule sets for interpretable classification. *JMLR*, 18(1):2357–2393, 2017.

Zhuo Wang, Wei Zhang, Ning Liu, and Jianyong Wang. Transparent classification with multilayer logical perceptrons and random binarization. In *AAAI*, pp. 6331–6339, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Hongyu Yang, Cynthia Rudin, and Margo Seltzer. Scalable bayesian rule lists. In *ICML*, pp. 3921–3930. JMLR. org, 2017.

A VANISHING GRADIENT PROBLEM

The partial derivative of each node in $\hat{\mathcal{S}}^{(l)}$ w.r.t. its directly connected weights and w.r.t. its directly connected nodes are given by:

$$\frac{\partial \hat{\mathbf{s}}_i^{(l)}}{\partial \hat{W}_{i,j}^{(l,1)}} = \hat{\mathbf{u}}_j^{(l-1)} \cdot \prod_{k \neq j} (1 - F_d(\hat{\mathbf{u}}_k^{(l-1)}, \hat{W}_{i,k}^{(l,1)})) \tag{8}$$

$$\frac{\partial \hat{\mathbf{s}}_i^{(l)}}{\partial \hat{\mathbf{u}}_j^{(l-1)}} = \hat{W}_{i,j}^{(l,1)} \cdot \prod_{k \neq j} (1 - F_d(\hat{\mathbf{u}}_k^{(l-1)}, \hat{W}_{i,k}^{(l,1)})) \tag{9}$$

Similar to the analysis of Equation 4, due to $\hat{\mathbf{u}}_k^{(l-1)}$ and $\hat{W}_{i,k}^{(l,1)}$ are in the range $[0, 1]$, the values of $(1 - F_d(\cdot))$ in Equation 8 and 9 are in the range $[0, 1]$ as well. If $\mathbf{n}_{l-1}$ is large and most of the values of $(1 - F_d(\cdot))$ are not 1, then the derivative is close to 0 because of the multiplications. Therefore, both the conjunction function and the disjunction function suffer from vanishing gradient problem.

If the large data set is very sparse and the number of 1 in each binary feature vector (for RRL the binary feature vector is $\mathbf{u}^{(0)}$) is less than about one hundred, there will be no vanishing gradient problem for nodes in $\hat{\mathcal{S}}^{(1)}$. The reason is when the number of 1 in each feature vector is less than about one hundred, in Equation 8 and 9, most of the values of $(1 - F_d(\cdot))$ are 1, and only less than one hundred values of $(1 - F_d(\cdot))$ are not 1, then the result of the multiplication is not very close to 0. The *facebook* data set is an example of this case. However, if the number of 1 in each binary feature vector is more than about one hundred, the vanishing gradient problem comes again.

B GRADIENTS AT DISCRETE POINTS

The gradients of RRL with original logical activation functions at discrete points can be obtained by Equation 4, 8 and 9. Take Equation 8 as an example, discrete points mean all the weights of logical layers are 0 or 1, which also means the values of all the nodes in $\hat{\mathcal{U}}^{(l)}$ are 0 or 1, $l \in \{0, 1, \ldots, L-1\}$. Hence, in Equation 8, $\hat{\mathbf{u}}_j^{(l-1)}, (1-F_d(\cdot)) \in \{0, 1\}$, and the whole equation is actually multiplications of several 0 and several 1. Only when $\hat{\mathbf{u}}_j^{(l-1)}$ and $(1 - F_d(\cdot))$ are all 1, the derivative in Equation 8 is 1, otherwise, the derivative is 0. Therefore, the gradients at discrete points have no useful information in most cases. The analyses of Equation 4 and 9 are similar.

To analyze the gradients of RRL with improved logical activation functions at discrete points, we first calculate the partial derivative of each node w.r.t. its directly connected weights and w.r.t. its directly connected nodes:

$$\frac{\partial \hat{\mathbf{r}}_i^{(l)}}{\partial \hat{W}_{i,j}^{(l,0)}} = \frac{(\hat{\mathbf{r}}_i^{(l)})^2}{F_c(\hat{\mathbf{u}}_j^{(l-1)}, \hat{W}_{i,j}^{(l,0)}) + \epsilon} \cdot (\hat{\mathbf{u}}_j^{(l-1)} - 1) \tag{10}$$

$$\frac{\partial \hat{\mathbf{r}}_i^{(l)}}{\partial \hat{\mathbf{u}}_j^{(l-1)}} = \frac{(\hat{\mathbf{r}}_i^{(l)})^2}{F_c(\hat{\mathbf{u}}_j^{(l-1)}, \hat{W}_{i,j}^{(l,0)}) + \epsilon} \cdot \hat{W}_{i,j}^{(l,0)} \tag{11}$$

$$\frac{\partial \hat{\mathbf{s}}_i^{(l)}}{\partial \hat{W}_{i,j}^{(l,1)}} = \frac{(1 - \hat{\mathbf{s}}_i^{(l)})^2}{1 - F_d(\hat{\mathbf{u}}_j^{(l-1)}, \hat{W}_{i,j}^{(l,1)}) + \epsilon} \cdot \hat{\mathbf{u}}_j^{(l-1)} \tag{12}$$

$$\frac{\partial \hat{\mathbf{s}}_i^{(l)}}{\partial \hat{\mathbf{u}}_j^{(l-1)}} = \frac{(1 - \hat{\mathbf{s}}_i^{(l)})^2}{1 - F_d(\hat{\mathbf{u}}_j^{(l-1)}, \hat{W}_{i,j}^{(l,1)}) + \epsilon} \cdot \hat{W}_{i,j}^{(l,1)} \tag{13}$$

Take Equation 10 for example, when all the weights of logical layers are 0 or 1, the $\hat{\mathbf{r}}_i^{(l)}$, $F_c(\cdot) + \epsilon$ and $(\hat{\mathbf{u}}_j^{(l-1)} - 1)$ are all very close to 0 or 1 as well. For the initialized weights are randomly selected, $\hat{\mathbf{r}}_i^{(l)}$ is close to 0 in most cases. Hence, the derivative in Equation 10 is close to 0 in most cases, and the analyses of Equation 11, 12 and 13 are similar. Therefore, the gradients at discrete points have little useful information.

C DATA SETS PROPERTIES

Table 2 summarizes the statistics of 13 datasets that we used in the experiments. The first 9 data sets are small data sets while the last 4 are large data sets. Discrete or continuous feature type indicates features in that data set are all discrete or all continuous. The mixed feature type indicates the data set has both discrete and continuous features. The density is the averaged ratio of the number of 1 in each binary feature vector after one-hot encoding.

Table 2: Data sets properties.

Dataset	#instances	#classes	#features	feature type	density
adult	32561	2	14	mixed	-
bank-marketing	45211	2	16	mixed	-
banknote	1372	2	4	continuous	-
chess	28056	18	6	discrete	0.150
connect-4	67557	3	42	discrete	0.333
letRecog	20000	26	16	continuous	-
magic04	19020	2	10	continuous	-
tic-tac-toe	958	2	9	discrete	0.333
wine	178	3	13	continuous	-
activity	10299	6	561	continuous	-
dota2	102944	2	116	discrete	0.087
facebook	22470	4	4714	discrete	0.003
fashion	70000	4	784	continuous	-

D CASE STUDY

Although RRL is not designed for image classification tasks, due to its high scalability, it can still provide intuition by visualizations. Take *fashion* dataset for example, for each class, we combine the first ten rules, ordered by linear layer weights, for feature (pixel) visualization. In Figure 5, a black/white pixel indicates the combined rule asks for a color close to black/white here in the original input image, and the grey pixel means no requirement in the rule. According to these figures, we can see how RRL classifies the images, e.g., distinguish T-shirt from Pullover by sleeves.

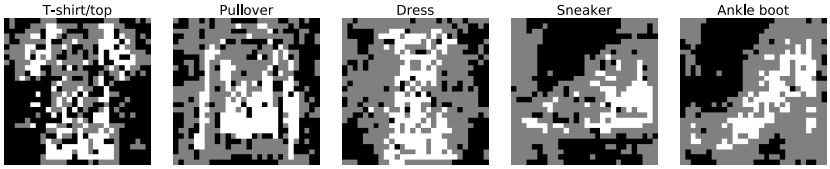


Figure 5: Decision mode for the *fashion* data set summarized from rules of RRL.

E MODEL COMPLEXITY

Figure 6 shows the scatter plots of F1 score against log(#edges) for rule-based models trained on the other 10 data sets. The value in CART(0.03) denotes the complexity parameter used for Minimal Cost-Complexity Pruning (Breiman, 2017), and a higher value corresponds to a simpler tree. On these 10 data sets, we can still observe that RRL is much simpler than random forests, and its complexity is close to CRS and decision trees without pruning. Meanwhile, RRL has higher F1 scores than models with close complexity in most cases.

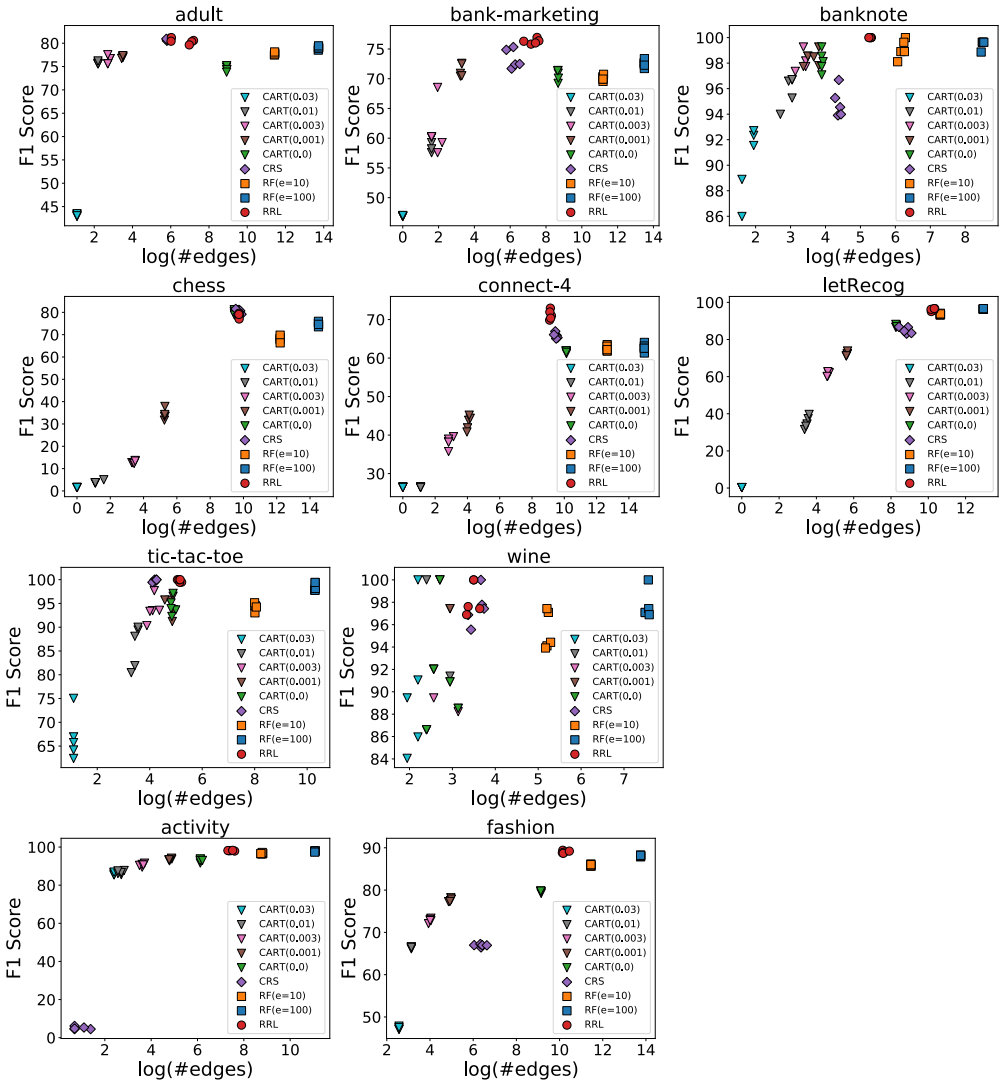


Figure 6: Scatter plot of F1 score against log(#edges) for 10 datasets for all 5 folds.