RuDi: Explaining Behavior Sequence Models by Automatic Statistics Generation and Rule Distillation

Yao Zhang
Yun Xiong*
yaozhang@fudan.edu.cn
yunx@fudan.edu.cn
Shanghai Key Laboratory of Data
Science, School of Computer Science,
Fudan University
China

Yiheng Sun
elisun@tencent.com
Tencent Weixin Group
China

Caihua Shan
caihuashan@microsoft.com
Microsoft Research Asia
China

Tian Lu
lutian@asu.edu
Department of Information Systems,
W. P. Carey School of Business,
Arizona State University
United States

Hui Song
ivansong@tencent.com
Tencent Weixin Group
China

Yangyong Zhu
yyzhu@fudan.edu.cn
Shanghai Key Laboratory of Data
Science, School of Computer Science,
Fudan University
China

ABSTRACT

Risk scoring systems have been widely deployed in many applications, which assign risk scores to users according to their behavior sequences. Though many deep learning methods with sophisticated designs have achieved promising results, the black-box nature hinders their applications due to fairness, explainability, and compliance consideration. Rule-based systems are considered reliable in these sensitive scenarios. However, building a rule system is labor-intensive. Experts need to find informative statistics from user behavior sequences, design rules based on statistics and assign weights to each rule. In this paper, we bridge the gap between effective but black-box models and transparent rule models. We propose a two-stage method, RuDi, that distills the knowledge of black-box teacher models into rule-based student models. We design a Monte Carlo tree search-based statistics generation method that can provide a set of informative statistics in the first stage. Then statistics are composed into logical rules with our proposed neural logical networks by mimicking the outputs of teacher models. We evaluate RuDi on three real-world public datasets and an industrial dataset to demonstrate its effectiveness.

CCS CONCEPTS

• **Information systems** → *Association rules*; Expert systems; • **Computing methodologies** → Game tree search.

KEYWORDS

statistics generation, rule distillation, sequence model explanation

*Corresponding author.

1 INTRODUCTION

Exponential growth in online activities has resulted in urgent needs for anomalous accounts detection to ensure the security and reliability of cyberspace. To evaluate risk scores of users, researchers and data scientists have proposed several approaches to model user behavior sequences, such as recurrent neural networks [30], or transformers [31].

Though these advanced methods have achieved promising results, the nature of opacity hinders its broader application. For example, when users of an electronic payment platform complain when their accounts have been banned, customer services may need to dig into the raw transactions and figure out if they are real fraudsters. In conservative businesses, such as banks, the results of these black-box machine learning models are hardly used directly but needed to be reviewed by human experts first, as shown in Figure 1(a). Instead, rule-based systems are considered more reliable in these scenarios, as shown in Figure 1(b). A set of (weighted) rules are manually designed by human experts with their understanding and observations from historical data. Then if a user's behavior sequence meets the rules more than a pre-defined threshold, his/her account is banned without human intervention. Though these rules are transparent and interpretable, they are limited by experts' experiences since experts may miss some patterns. Such systems are also hard to detect new risky patterns as fraudsters evolve.

To address the limitations of handcrafted rules, the researchers have proposed learning-based rule models [2, 26–28]. For example, the state-of-the-art method RRL [26] utilizes neural logical networks to learn a set of CNFs (conjunctive normal forms) or DNFs
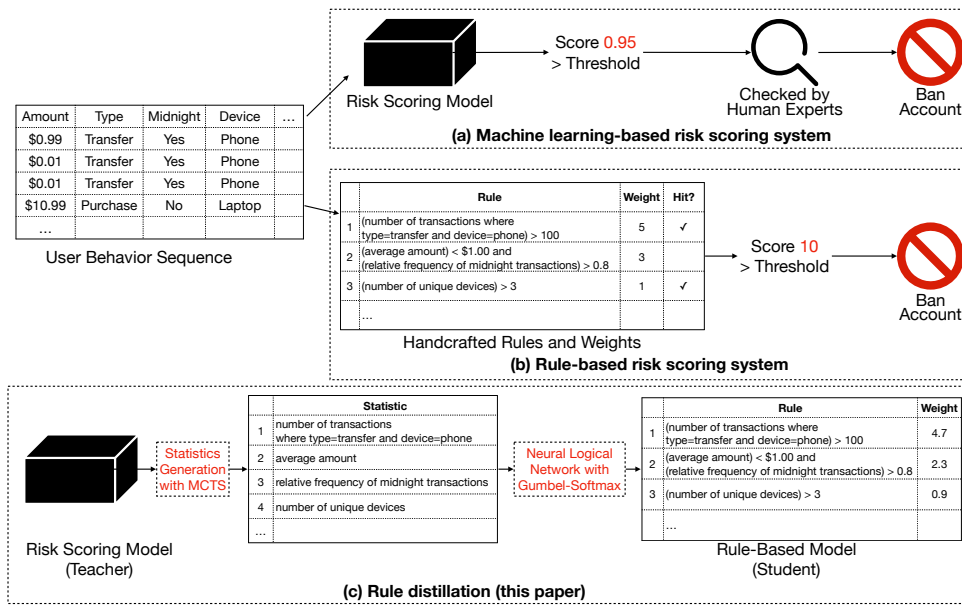
Figure 1: Illustration of two types of risk scoring systems and rule distillation. (a) A black-box risk scoring model, despite its high precision, is hardly used without human review due to fairness, explainability, and compliance requirements. (b) Rule-based systems are considered more reliable in conservative businesses. However, designing rules are labor-intensive. (c) In this paper, we bridge the gap by proposing a two-stage rule distillation method RuDi that can distill knowledge of black-box models into transparent rule-based systems.

(disjunctive normal forms) from data. But these models require inputs being fixed-length feature vectors and cannot handle sequences of user behaviors. Moreover, due to the discrete nature of these models, they require stronger supervision signals to train.

In this paper, we bridge the gap between effective but opaque machine learning models and transparent but simple rule systems when modeling user behavior sequences. Instead of learning a logical rule model from raw data directly, we distill a trained risk scoring model, *i.e.*, the teacher model, into a rule-based student model, as shown in Figure 1(c). In this case, the precise predictions can be made by the teacher model, while the experts can audit the model by examining the distilled rules. Further, the distilled rules can supplement the existing rule systems to capture new trends and provide new insights.

Despite its significance, distilling rules from a sequence model is challenging due to the following reasons:

Exponential number of statistics. As Figure 1(b) illustrates, rules are typically based on some statistics that human experts handcraft. A set of informative statistics is the cornerstone of effective rules. However, the number of possible statistics grows exponentially as the number of conditions grows, *e.g.*, the "where" clause in Figure 1(b). Such a large space prevents us from enumerating all statistics, and a systematic search strategy must be applied.

Tradeoff between effectiveness and interpretability of rule models. Unlike traditional knowledge distillation [11] where teachers and students share similar structures, rule distillation aims to use an interpretable rule model to mimic the outputs of a black-box teacher model. As pointed out by [26], traditional rule models

like decision trees and rule lists/sets are hard to get comparable performance with complex models. Tree ensembles [17] though effective, are commonly not viewed as interpretable models. On the other hand, recently proposed neural logical networks [26, 27] produce complicated rules where each clause may contain over dozens of literals. These models are also hard to train due to their discrete structures. So it is a challenge to propose an effective and interpretable rule model that can precisely mimic the outputs of teacher models.

To tackle the above challenges, we propose an effective two-stage method, named RuDi (**Ru**le **Di**stillation). In the first stage, we propose MCTS-SG, a novel **s**tatistics **g**eneration method based on **M**onte **C**arlo **T**ree **S**earch (MCTS) [4]. We pre-define a set of basic operators like Select, Mean, GroupBy, *etc.*. Then we model the statistics generation as a sequential decision process, where each statistic is composed of a series of operators. The objective is to maximize the (expected) correlation between the teacher model's outputs and the generated statistics. Each simulation of tree search only involves a small batch of data, and the search space of statistics is explored by MCTS systematically. In this way, we generate a set of statistics effectively and efficiently.

In the second stage, we learn a set of weighted rules with our proposed GS-NLN. GS-NLN is a novel and lightweight **n**eural **l**ogical **n**etwork based on **G**umbel-**s**oftmax [14]. Specifically, each neuron in the GS-NLN only selects two neurons from the previous layer and puts them into a conjunctive or disjunctive formula. Unlike previous methods using logical activation functions [23] as surrogates, our proposed model directly uses boolean operations with

Gumbel-softmax estimators and thus is easier to train. Thanks to the simplicity of structures, the learned rules are more interpretable and practicable than RRL. The experimental results prove that these simple rules can achieve satisfying performance.

The main contributions of this paper are as follows:

• We propose RuDi, a two-stage method that distills black-box behavior sequence models into rule models. The distilled rules are interpretable and transparent, and reveal the underlying mechanisms of teacher models. To the best of our knowledge, this is the first study on post hoc explanations of user behavior sequence models using rule distillation.

• We model the statistics generation as a sequential decision process of composing operators and design a Monte Carlo Tree Search-based search algorithm MCTS-SG. Our approach can generate informative statistics efficiently while only requiring small batches of data during tree searching.

• We propose a novel and lightweight neural logical network GS-NLN. Each neuron in the proposed model only connects to two neurons in the previous layer resulting in simpler forms of rules and better interpretability.

• Extensive experiments on three real-world datasets prove the effectiveness of our proposed RuDi. Case studies on an industrial dataset demonstrate the interpretability of learned rules.

2 RELATED WORK

In this section, we introduce three related topics: automatic feature generation, ruled-based models and knowledge distillation.

Automatic Feature Generation. Feature engineering is one of the most crucial and time-consuming steps in data mining tasks. For tabular data, many researchers [16, 18, 24, 32] have proposed automatic feature generation methods by doing feature combination and crossing. In [15] the authors proposed DSM, which enumerates features in relational databases. OneBM [21] extends DSM by introducing entity graph traversal methods and selecting features with Chi-squared tests. However, these methods cannot handle sequence data, not to mention extracting statistics from data. They also overlook the combination explosion problems and can only generate shallow features. There are some studies [5, 6] on feature engineering with Monte Carlo tree search (MCTS) [4], but they focus on feature selection for tabular data.

In this paper, we propose an MCTS-based statistics generation method MCTS-SG. We use the term "statistic" instead of "feature" because we treat it as a composition of basic operators. We systematically explore the search space of statistics with MCTS, and only small batches of data are used during searching. By defining basic operators, our proposed MCTS-SG can generate informative statistics effectively and efficiently.

Rule-based Models and Neural Logical Networks. It has been shown [26] that traditional rule models [2, 27, 28] with decision trees, rule lists, or rule sets are hard to get comparable performance. Moreover, most methods produce deep and hierarchical IF-THEN rules that are less practical in rule-based systems. In [26], the authors proposed the state-of-the-art rule-based model, RRL, which is a kind of neural network with logical activation functions [23]. RRL is hard to train due to its discrete structures. Besides, an extra continuous version of the network must be maintained

to provide gradients. Though RRL learns flat rules, the number of literals in a rule is uncontrollable even with shallow network structures, leading to perplexing results.

In this paper, we propose GS-NLN, a Gumbel-softmax-based neural logical network with the constraint that each neuron can only connect to two neurons in the previous layer. This constraint simplifies the network structure and makes the model easy to train with Gumbel-softmax estimators [14]. The learned rules are also neat and sound.

Knowledge Distillation and Post-hoc Explanation with Surrogate Models. In [12] the authors proposed knowledge distillation, where a small student model is trained to mimic the outputs of a large teacher model. Following the idea, some researchers distill the knowledge of black-box models into transparent models [7, 25], *i.e.*, surrogate models, as a way of post-hoc model explanation. For example, the model auditing method [25] detects what features were used in the teacher models. These approaches only produce importance of features. In [10, 22] the authors proposed methods to distill neural networks into decision trees. Another branch of research [23, 24] locally fits teacher models to generate instance-wise explanations. There is another work called rule knowledge distillation [13]. It distills rules into deep neural networks, which is opposite to our setting. Nonetheless, most existing methods cannot deal with sequence data.

In this paper, we distill a black-box sequence model into a set of rules with the help of the proposed MCTS-SG and GS-NLN. This paper is the first study on distilling sequence models into rule systems to the best of our knowledge.

3 METHODOLOGY

3.1 Overview

Let $\{X_1, \ldots, X_N\}$ denote the dataset of N users, where X_i is the sequence of user i's behaviors. All sequences contains m_1 categorical variables (columns) $\{F_1, \ldots, F_{m_1}\}$ and m_2 numerical variables (columns) $\{F_{m_1+1}, \ldots, F_{m_1+m_2}\}$. The sample spaces of categorical variables are denoted as $\mathbb{F}_j = \{f_{j1}, \ldots, f_{jp}\}$, while the sample spaces of the numerical ones are $\mathfrak{R}$. For notation simplicity, here we assume $|\mathbb{F}_1| = \cdots = |\mathbb{F}_{m_1}| = p$.

A trained risk scoring model, *e.g.*, a recurrent neural network, takes the sequence X_i as input and produces its continuous prediction $y_i \in \mathfrak{R}$. Note that we do not constrain the range of y_i, since we treat the risk scoring model as a black-box, and its outputs may correspond to probabilities, log-probabilities, or just real-valued risk scores. The task of rule distillation is to find a set of weighted rules that can mimic the teacher model's outputs $y \in \mathfrak{R}^N$.

The framework of our proposed RuDi is illustrated in Figure 2. RuDi is a two-stage method. In the first stage, we use an MCTS-based statistics generation method to generate a small set of statistics. Then in the second stage, we binarize the statistics into literals and train a neural logical network by minimizing the ranking loss with respect to y. After training, the weights and neurons in the last prediction layer are extracted as the distilled rules.
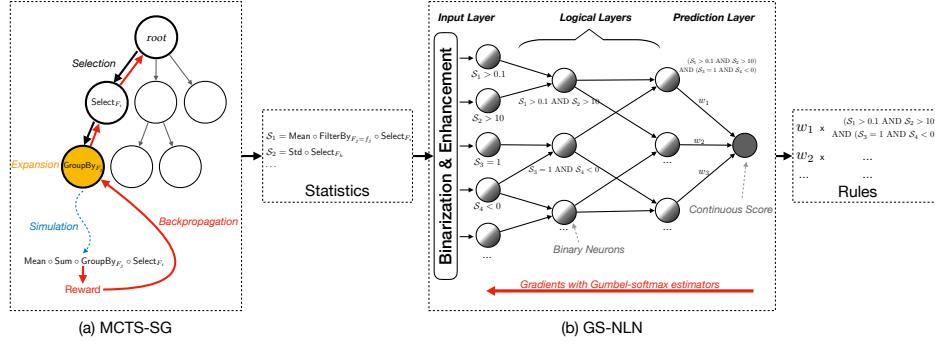
(a) MCTS-SG

(b) GS-NLN

Figure 2: The framework of our proposed RuDi. (a) Statistics generation with Monte Carlo tree search. (b) Neural logical network with Gumbel-softmax. Here we discard disjunction parts and skip connections for simplicity.

3.2 Stage 1: Statistics Generation with MCTS

In the first stage, we search for promising statistics that can be transformed into literals. The algorithm of MCTS-SG is summarized in Algorithm 1.

3.2.1 Statistics as compositions of operators. In this paper, we define a statistic[1] as a composition of basic operators:

$$\mathcal{S} = \mathcal{T}_d \circ \cdots \circ \mathcal{T}_1, \ 2 \leq d \leq D,$$

where D is the maximum depth of the statistics, and $\mathcal{T}_i \in \mathbb{T}$ is selected from the following pre-defined sets: $\mathbb{T} = \mathbb{T}_{select} \cup \mathbb{T}_{agg} \cup \mathbb{T}_{by} \cup \mathbb{T}_{tsfm}$.

Select operators:

$$\mathbb{T}_{select} = \{Select_{F_j} | j = 1, \ldots, m_1 + m_2\}.$$

The operator $Select_{F_j}$ selects the j-th column of the data. If F_j is categorical, then values in the columns are transformed into one-hot representations, and the following operators work on each dimension of data separately. The first operator must be from this set, *i.e.*, $\mathcal{T}_1 \in \mathbb{T}_{select}$.

Aggregation operators:

$$\begin{aligned} \mathbb{T}_{agg} = \ & \{Mean, Max, Min, Sum, Mean, Std, Ptp, Count, First\} \\ & \cup \{Percentile_k | k = 5, 10, 25, 50, 75, 90, 95\}. \end{aligned}$$

$$(1)$$

As the name implies, an aggregation operator returns the corresponding statistics (Mean, Max, Min, Sum, Mean, Std, Ptp[2]) of the input column. First returns the first element, and Count returns the length of the sequence. $Percentile_k$ computes the k-th percentile of the selected column. An aggregation operator and a select operator can compose the shortest statistics. Take $\mathcal{S}_2 = Std \circ Select_{F_k}$ in Figure 2(a) as an example. $\mathcal{S}_2(\mathcal{X}_i)$ returns the standard deviation of the k-th column in sequence $\mathcal{X}_i$.

By operators:

$$\begin{aligned} \mathbb{T}_{by} = \ & \{GroupBy_{F_j} & | j = 1, \ldots, m_1\} \\ & \cup \{FilterBy_{F_j = f_j} & | j = 1, \ldots, m_1, f_j \in \mathbb{F}_j\} \\ & \cup \{RetainBy_{F_j = f_j} & | j = 1, \ldots, m_1, f_j \in \mathbb{F}_j\} \\ & \cup \{SortBy_{F_j, k} & | j = m_1 + 1, \ldots, m_1 + m_2, k \in \{asc, desc\}\} \end{aligned}$$

---

[1] We use the term statistic to represent a statistical function or the value of the function interchangeably.
[2] Ptp: peak-to-peak

Algorithm 1 MCTS-SG

Require: Dataset $\{\mathcal{X}_i\}$, teacher model's outputs $y \in \mathfrak{R}^N$
1: $\hat{y} = y$
2: **for** $k = 1, 2, \ldots, K$ **do**
3:     Initialize $\mathcal{S} \leftarrow$ IdentityMapping
4:     **for** $d = 1, 2, \ldots, D$ **do**
5:         Initialize the root node v_0 with $S(v_0) = \mathcal{S}$
6:         **for** $it = 1, 2, \ldots, iterations$ **do**
7:             Grow the tree according to Algorithm 2
8:         **end for**
9:         Select the best child v of the root according to Eq. 6
10:         $\mathcal{T} \leftarrow A(v), \mathcal{S} \leftarrow \mathcal{T} \circ \mathcal{S}$
11:         **if** $\mathcal{S}$ is a valid statistic **then**
12:             break
13:         **end if**
14:     **end for**
15:     Mark $\mathcal{S}$ as invalid
16:     $X_k = (\mathcal{S}(\mathcal{X}_1), \cdots, \mathcal{S}(\mathcal{X}_N))^\top$
17:     Update $\hat{y}$ according to Eq. 7
18: **end for**
19: Return $X_1, \ldots, X_K$

The $GroupBy_{F_j}$ operator partitions an input sequence into several groups according to the categorical feature F_j. A GroupBy operator must be followed by an aggregation operator that computes statistics within each group. The $FilterBy_{F_j = f_j}$ ($RetainBy_{F_j = f_j}$) operator removes (keeps) rows where the categorical feature F_j is f_j. The $SortBy_{F_j, \cdot}$ operator sorts the input sequence in ascending/descending order.

Transformation operators:

$$\mathbb{T}_{tsfm} = \{Top_5, Abs\}$$

The Top_5 returns the first 5 rows of the input data. The Abs returns the absolute values of the input data.

Note that our proposed method is highly extensible as users can define their own operators. A valid statistic $\mathcal{S}(\mathcal{X}_i)$ is a scalar if a numerical column is selected, otherwise a p-dimensional vector. We use $\mathbb{V}$ to denote the set of all valid statistics within the maximum depth D. For instance, Mean $\circ$ Mean $\circ$ Select $\cdot$ is not a valid statistic.

Algorithm 2 Tree Growth

Require: root node v_0, dataset $\{X_i\}$, target $\hat{y}$
1: $v \leftarrow v_0$
2: **while** v is not terminal **do**   ▷ Selection
3:     **if** v is not fully expanded **then**   ▷ Expansion
4:         Choose an untried valid operator $\mathcal{T}$
5:         Create a child node v' of v with $Q(v') = C(v') = 0, A(v') = \mathcal{T}, S(v') = \mathcal{T} \circ S(v)$; $v \leftarrow v'$
6:         break
7:     **else**
8:         Select the child v' according to Eq. 2; $v \leftarrow v'$
9:     **end if**
10: **end while**
11: **if** v is terminal **then**   ▷ Simulation
12:     $S \leftarrow S(v)$
13: **else**
14:     Sample a valid statistic S according to Eq. 3
15: **end if**
16: Sample a batch index set $\mathbb{B}$
17: Compute reward r according to Eq. 8 or Eq. 9
18: **while** v is not root **do**   ▷ Backpropagation
19:     $C(v) \leftarrow C(v) + 1$
20:     $S(v) \leftarrow S(v) + r$
21:     $v \leftarrow parent(v)$
22: **end while**

3.2.2 MCTS Formulation. Now we focus on generating one statistic alone. We formulate the statistics generation as a sequential decision process. Assume that we are going to generate a statistic S, and we have generated a partial sequence of operators $\mathcal{T}_1, \cdots, \mathcal{T}_{i-1}$.

To determine the next optimal operator $\mathcal{T}_i$, we firstly initialize a search tree with a single root node v_0. Each node v (except the root) has four fields: the state $S(v)$, the action $A(v)$, the cumulative reward $Q(v)$ and the number of visits $C(v)$. The root node only has state $S(v_0) = \mathcal{T}_{i-1} \circ \cdots \circ \mathcal{T}_i$.

Then we repeat the following steps to grow the tree as summarized in Algorithm 2.

(Selection) From the root, we select the best child iteratively until we reach a not fully expanded node or a terminal node. A node v is not fully expanded if there exists valid operator $\mathcal{T} \in \{\mathcal{T} | \exists \mathcal{T}_1', \ldots, \mathcal{T}_d' \text{ s.t. } \mathcal{T}_d' \circ \cdots \circ \mathcal{T}_1' \circ \mathcal{T} \circ S(v) \in \mathbb{V}\}$ that is not the child of v. A node is terminal if its state $S(v)$ corresponds to a valid statistic. The best child is selected according to the UCT algorithm [19]:

$$\arg\max_{v' \in Children(v)} \frac{Q(v')}{C(v')} + c \cdot \sqrt{\frac{2 \ln C(v)}{C(v')}}, \tag{2}$$

where v is the current node (initialized as v_0) and $c = 1/\sqrt{2}$ is the exploration constant. The first term in Eq. 2 is the estimated state value, and the second term gives high values when a child is less frequently visited. In this case, we balance exploitation and exploration.

(Expansion) If the selected node v is not fully expanded, we pick an untried valid operator $\mathcal{T} \in \mathbb{T}$. We create a new child node v' of v with $Q(v') = C(v') = 0, A(v') = \mathcal{T}$, and $S(v') = \mathcal{T} \circ S(v)$. Then we return $v = v'$.

(Simulation) Now we estimate the reward of the selected node v. We denote the state of the node, *i.e.*, statistic, as $S = S(v)$. If the selected node v is not a terminal node, *i.e.*, $S(v) = \mathcal{T}_{i+j} \circ \cdots \circ \mathcal{T}_1$ is not a valid statistic, then we randomly sample operators to form a valid one:

$$S \leftarrow \mathcal{T}_D \circ \cdots \circ \mathcal{T}_{i+j+1} \circ \mathcal{T}_{i+j} \circ \cdots \circ \mathcal{T}_1. \tag{3}$$

Since we want to generate a statistic relevant to the teacher model's outputs, we design the reward function as the (absolute) correlation between them:

$$r = |\text{corr}(x_{\mathbb{B}}, y_{\mathbb{B}})|, \tag{4}$$

where $\mathbb{B}$ is a randomly sampled batch of indices with $|\mathbb{B}| = B$. The statistics and target values are $x_{\mathbb{B}} = \left(S(X_{b(1)}), \ldots, S(X_{b(B)})\right)^\top \in \mathfrak{R}^B$ and $y_{\mathbb{B}} = \left(y_{b(1)}, \ldots, y_{b(B)}\right)^\top \in \mathfrak{R}^B$, where $b(i)$ is the index of the i-th data in the batch. And $\text{corr}(x, y) = \frac{\bar{x}^\top \bar{y}}{\sqrt{\bar{x}^\top \bar{x} \cdot \bar{y}^\top \bar{y}}}$ is the Pearson correlation coefficient, where $\bar{x}, \bar{y}$ are centered x, y.

If the Select operator selects a categorical feature, the output is denoted as

$$X_{\mathbb{B}} = \left(S(X_{b(1)}), \ldots, S(X_{b(B)})\right)^\top \in \mathfrak{R}^{B \times p}.$$

Then we define the reward as the absolute value of the coefficient of multiple correlation [1]:

$$r = |\text{corr}(y', y_{\mathbb{B}})|, \tag{5}$$
$$\text{where} \quad \beta, \beta_0 = \arg\min_{\beta \in \mathfrak{R}^p, \beta_0 \in \mathfrak{R}} \|y_{\mathbb{B}} - X_{\mathbb{B}} \cdot \beta + \beta_0\|_2,$$
$$y' = X_{\mathbb{B}} \cdot \beta + \beta_0.$$

In this case we compute the Pearson correlation coefficient between the actual values and the fitted values of a linear regression model.

(Backpropagation) Once we obtain the reward of the node v, we can back propagate the reward to its ancestors. Specifically, do the following updates until we reach the root node:

$$C(v) \leftarrow C(v) + 1,$$
$$Q(v) \leftarrow Q(v) + r,$$
$$v \leftarrow parent(v).$$

After a certain number of iterations, we can terminate the tree growing algorithm and return the best child of the root:

$$v = \arg\max_{v' \in Children(v_0)} \frac{Q(v')}{C(v')}, \tag{6}$$

We update the partial solution with $\mathcal{T}_i = A(v)$. Then we create a new search tree with the root $\mathcal{T}_i \circ \cdots \circ \mathcal{T}_1$ and repeat the algorithm until we construct a valid statistic.

3.2.3 Generating top-K statistics. In the previous subsection, we describe the algorithm for generating one statistic. Can we invoke the algorithm K times to generate K statistics? Apparently no. The algorithm's objective is to maximize the absolute correlation between the statistic and the teacher model's outputs. If we naively run the algorithm multiple times, we are likely to obtain multiple highly correlated and redundant statistics, for example, the mean, the sum, and the 50th percentile of "amount" in Figure 1.

So in this subsection, we introduce two modifications to improve the diversity of the generated statistics. First, once we obtain a

statistic S_k, we mark it invalid by updating $\mathbb{V} \leftarrow \mathbb{V} \backslash \{S_k\}$. In this case, it would not be generated again even in simulation processes (Eq. 3).

Before generating the $(k + 1)$-th statistic, we change the target values with:

$$\hat{y} = y - \left(\sum_{i=1}^{k} X_i \cdot \beta_i + \beta_0 \right), \tag{7}$$

$$\text{where} \quad \beta_0, \beta_1, \ldots, \beta_k = \underset{\beta_0, \ldots \beta_k}{\arg \min} \left\| y - \left(\sum_{i=1}^{k} X_i \cdot \beta_i + \beta_0 \right) \right\|_2,$$

$$X_i = (S_i(\mathcal{X}_1), \cdots, S_i(\mathcal{X}_N))^\top \in \mathfrak{R}^N \text{ or } \mathfrak{R}^{N \times p},$$

$$y = (y_1, \ldots, y_N) .$$

Here $X_i \in \mathfrak{R}$ or $\mathfrak{R}^{N \times p}$ and $\beta_i \in \mathfrak{R}$ or $\mathfrak{R}^p$ depend on whether the statistic acts on a categorical column. Then we replace $y_{\mathbb{B}}$ in Eq. 4 and Eq. 5 by $\hat{y}_{\mathbb{B}}$:

$$r = |\text{corr}(x_{\mathbb{B}}, \hat{y}_{\mathbb{B}})|, \text{ (for numerical columns)} \tag{8}$$

$$r = |\text{corr}(\hat{y}', \hat{y}_{\mathbb{B}})|, \text{ (for categorical columns)} \tag{9}$$

Now the objective of the algorithm is to maximize the (absolute) partial correlation between the statistic and the outputs of the teacher, given existing generated statistics. In this case, we reduce the collinearity of statistics.

3.2.4 Discussion. In Eqs. 7, 8 and 9 we assume the linearity of data. We can extend these formula into more general cases by considering dependency rather than linear correlation. For example, we can replace the reward function in Eq. 8 with the following equation

$$r = \text{CondDep}(x_{\mathbb{B}}, y_{\mathbb{B}} | X_{1, \mathbb{B}}, \ldots, X_{k, \mathbb{B}}),$$

where CondDep corresponds to some conditional dependency tests [29], and $X_{i, \mathbb{B}}$ are values of statistics on the batch. Moreover, the general idea of Algorithm 1 and Eq. 7 is similar to the forward selection phase of forward-backward causal feature selection methods [29]. It is also possible to incorporate the backward elimination phase to remove unsatisfying statistics generated in early steps.

3.2.5 Complexity. During the tree search, there are $K \times D \times B \times$ (number_of_simulations) data points being evaluated, which is independent of the dataset size n or the space of statistics. Since to evaluate the statistics of all data points $\mathcal{X}_i$, another $K \times N$ evaluations are inevitable, Eq. 7 only introduces little computational overhead, *i.e.*, a regression problem with linear complexity w.r.t N. We use MCTS to explore the space of statistics systematically, and Algorithm 1 is highly scalable.

3.3 Stage 2: Rule Distillation with Neural Logical Network

In this section, we describe how to distill rules from the teacher model with the generated statistics. As demonstrated in Figure 2(b), our proposed GS-NLN contains an input layer, several logical layers, and a prediction layer.

3.3.1 Input Layer. Assume that we have obtained the values of top-K statistics:

$$X = (X_1, \ldots, X_K) \in \mathfrak{R}^{N \times P},$$

where P is the corresponding dimension after concatenation. Some columns of X have already been binary, *e.g.*, categorical features with Max operator, while the others have not. So we iterate over the columns of X. If a column corresponds to a continuous statistic, we binarize it according to its percentiles:

$$x'_j = \begin{cases} x_j & \text{if } x_j \in \{0, 1\}^N, \\ (\mathbb{I}_{x_j > x_j^0}, \ldots, \mathbb{I}_{x_j > x_j^{90}}, \mathbb{I}_{x_j > p_j^{100}}) \in \{0, 1\}^{N \times 11} & \text{otherwise.} \end{cases} \tag{10}$$

Here, $\{x_j^k | k = 0, \ldots, 90, 100\}$ are percentiles of x_j, $\mathbb{I}_{\text{cond}}$ is the element-wise indicator function that returns 1 if cond is true, otherwise 0. We denote the binarized features as $X' = (x'_1, \ldots, x'_P) \in \{0, 1\}^{N \times P'}$, and each element in X' corresponds to a boolean literal.

Now let us focus on a certain row of X', *i.e.*, the boolean representations of the i-th user, denoted as $z_i = \text{row}_i(X')^\top \in \{0, 1\}^{P'}$. We enhance the input by

$$z_i^0 = (z_i^\top, (1 - z_i)^\top, 1, 0)^\top \in \{0, 1\}^{h_0},$$

where $h_0 = 2 \cdot P' + 2$. Since the following logical layers are only cable of doing AND/OR operations, we enhance the input by appending its negations. Moreover, extra 1 and 0 are added to enable the following layers to learn shortcuts.

3.3.2 Logical Layers. Our proposed GS-NLN has L logical layers, and each is composed of a conjunction part and a disjunction part. Let $z_i^{l-l} \in \{0, 1\}^{h_{l-1}}$ denote the binary hidden representation after the $(l - 1)$-th layer. The j-th neuron in the conjunction part would select two elements in z_i^{l-l} by sampling from the parameterized categorical distributions:

$$q_j^{l,k} \sim \text{onehot}\left(\text{softmax}(w_j^{l,k}) \right), \; k = 1, 2 \tag{11}$$

where $w_j^{l,k} \in \mathfrak{R}^{h_{l-1}}$ are parameters. Then the value of the neuron is the boolean AND between the input values:

$$u_j^l = (q_j^{l,1})^\top z_i^{l-1} \cdot (q_j^{l,2})^\top z_i^{l-1} \in \{0, 1\},$$

Similarly, neurons in the disjunction part conduct the boolean OR operations:

$$v_j^l = 1 - (1 - (q_j^{l,3})^\top z_i^{l-1}) \cdot (1 - (q_j^{l,4})^\top z_i^{l-1}) \in \{0, 1\},$$
$$q_j^{l,k} \sim \text{onehot}\left(\text{softmax}(w_j^{l,k}) \right), \; k = 3, 4. \tag{12}$$

The sampling operations in Eq. 11 and Eq. 12 cause the network no longer differentiable. So we resort to the Gumbel-softmax estimator [14], and re-write them into

$$q_j^{l,k} = \text{softmax}((w_j^{l,k} + g_k) \cdot \tau^{-1}), \; k = 1, 2, 3, 4, \tag{13}$$

where $g_k \in \mathfrak{R}^{h_{l-1}}$ are h_{l-1} i.i.d. samples from the Gumbel distribution Gumbel$(0, 1)$, and τ is the temperature. As τ approaches 0, Eq. 13 becomes identical to Eq. 11 and Eq. 12. During training, we start at a high temperature $\tau = 1$ and anneal to a low temperature, *e.g.*, $\tau = 0.0001$.

The output of the layer is composed of

$$z^l = (u_1, \ldots, u_H, v_1, \ldots, v_H, (z^{l-1})^\top)^\top \in \{0, 1\}^{h_l},$$

where $h_l = 2 \cdot H + h_{l-1}$, and H is the hidden size. We also use skip connections so that the model can adjust the complexity of rules adaptively. But at the last layer, we do not use the skip connection

and use $\frac{R}{2}$ neurons in conjunction and disjunction parts respectively, where R is the number of desired rules.

3.3.3 Prediction Layer.
After L logical layers, we obtain $z_i^L \in \{0, 1\}^R$ where each element represents a logical rule. Now we get the score of the instance by a linear prediction layer $\tilde{y}_i = w^\top z_i^L$, where $w \in \mathfrak{R}^R$ is the model parameter representing the weights of rules. It is possible to pass w or $\tilde{y}_i$ through non-linear functions. For example, we could use the softplus function if we want the weights of rules nonnegative, or use sigmoid function to constrain scores to be with range (0, 1). We leave this for future work.

3.3.4 Model Training.
The training objective is to mimic the outputs of the teacher model. We use the following ranking-based loss function

$$\mathcal{L} = \frac{1}{N \times (N-1)} \sum_{i \neq j} \Big(\mathbb{I}_{y_i > y_j} \ln \sigma(\tilde{y}_i - \tilde{y}_j) + \mathbb{I}_{y_i \leq y_j} \ln \sigma(\tilde{y}_j - \tilde{y}_i) \Big),$$

where $\sigma(\cdot)$ is the sigmoid function. In practice, we use the stochastic gradient descent to optimize the loss function, and an instance i is only compared to the samples in the same mini-batch.

3.3.5 Differences between GS-NLN and RRL.
RRL [26] is the state-of-the-art algorithm that learns logical rules. Though our proposed GS-NLN and RRL have similar outputs, they are highly different. RRL aims to learn rule-based representations for interpretable classification, while GS-NLN is to distill the knowledge of a teacher into a rule system. So in GS-NLN, a neuron can only connect to two input units to maintain the low complexity of the rules. On the contrary, in RRL a neuron can connect to any number of units for better performance.

Another difference lies in the way discrete activations are handled in networks. In RRL, an extra continuous version of the network is maintained, where boolean operations are implemented with logical activation functions [23]. The authors proposed the gradient grafting technique, and they use the grafted gradients of the continuous network to guide the updates of the discrete one. In our work, since each neuron only connects to two units, we would hardly face notorious vanishing gradient problems of logical activation functions, and straightforward Gumbel-softmax estimators can learn parameters very well.

4 EXPERIMENTS

In this section, we prove the effectiveness of our proposed RuDi. The codes of RuDi and pre-processing scripts can be found at https://github.com/yzhang1918/cikm2022rudi. Processed datasets and trained teacher models with train/valid/test splits are also included.

4.1 Datasets and Teacher Models

Table 1: Statistics of datasets.

	VEWS	Elo	RedHat	Industrial
# Users	33,576	201,917	144,639	222,306
# Records	772,530	19,249,694	1,889,213	34,722,334
# Cat. Features	7	12	11	6
# Num. Features	0	1	0	1

Table 2: Default values of hyper-parameters in RuDi.

Hyper-parameter	Value
(MCTS-SG)	
Statistics Depth D	4
Number of statistics K	20
Batch size B	128
Number of simulations	500
(GS-NLN)	
Number of rules R	20
Hidden size H	20
Number of logical layers L	2
Optimizer	Adam
Batch size	128
Epochs	500
Learning rate	$0.1 \to 0.001$ (linear decay)
Temperature τ	$1.0 \to 0.0001$ (linear decay)

We use 3 public datasets: (1) VEWS [20] contains 33,576 users and their edit behaviors on Wikipedia, among which Wikipedia administrators blocked 17,027 users for vandalism. (2) Elo[3] contains transactions of 201,917 users, and 97,609 users are marked positive due to their customer loyalty. (3) RedHat[4] contains activities of 144,639 users, and 62,115 users are positive considering their potential business values. An extra industrial dataset is analyzed in Section 4.5. The statistics of the datasets are summarized in Table 1.

Two models are used as teachers: (1) GRU: We use a bi-directional 2-layer GRU with 128 hidden units and dropout ratio 0.2. (2) Light-GBM: As one of the most popular methods in data mining competitions, we use LightGBM [17] as the second teacher model. We use all aggregation operators introduced in Eq. 1 to transform raw sequences into tabular forms.

We use 80% users' sequences as the training data, among which 1,000 sequences are used for validation. The remaining are used for testing. We train GRU and LightGBM with early stopping.

Unless otherwise stated, we use the default hyper-parameters of RuDi summarized in Table 2. Experiments on public datasets were conducted on a single server with 72 cores, 128GB memory, and four Nvidia Tesla V100 GPUs.

4.2 Experiment 1: Statistics Generation

In the first group of experiments, we test if our proposed MCTS-SG can generate informative statistics.

As we reviewed in Section 2, most automatic feature engineering methods cannot be directly applied to behavioral sequences. We consider the following alternative approaches. (1) Random: We generate K statistics by randomly composing operators introduced in Section 3.2.1. (2) Lasso [9]: We apply all aggregation operators (Eq. 1) on the datasets to transform sequences into fixed-length feature vectors, then train Lasso models to perform feature selection. (3) LightGBM [17]: Similarly, we train LightGBM on the flattened feature vectors and then select features with the highest importance

---

[3] https://www.kaggle.com/c/elo-merchant-category-recommendation/
[4] https://www.kaggle.com/c/predicting-red-hat-business-value/

Table 3: Experimental results of statistics generation.

	VEWS	Elo	RedHat
Random	0.7048	0.4198	0.5906
Lasso	0.9302	0.6363	0.6229
LightGBM	0.9240	0.6769	0.6292
OneBM	0.9120	0.6868	0.6555
MCTS-SG (This paper)	**0.9429**	**0.7262**	**0.7872**

scores. (4) OneBM [21]: This method can generate features on relational databases and use Chi-squared testing to select features. We transform user sequences into two relational tables to meet its input requirements. Other feature generation methods on relational databases like DSM [15] have been shown inferior to OneBM and thus are omitted here.

Since we use LightGBM as a baseline here, we choose GRU as the teacher model. We measure the coefficients of multiple correlation as Eq. 5 between generated statistics/features and the teacher model's outputs on the training splits. The higher the value is, the more informative the generated statistics are.

The results are shown in Table 3. As we can see, the proposed MCTS-SG outperforms baselines significantly. Lasso, LightGBM and OneBM cannot generate deep statistics as MCTS-SG and thus only achieve suboptimal performance. Though Random can generate deep statistics, the exponentially large search space makes it impossible to generate useful statistics without systematic search strategies. These results prove the effectiveness of our MCTS-based statistics generation method.
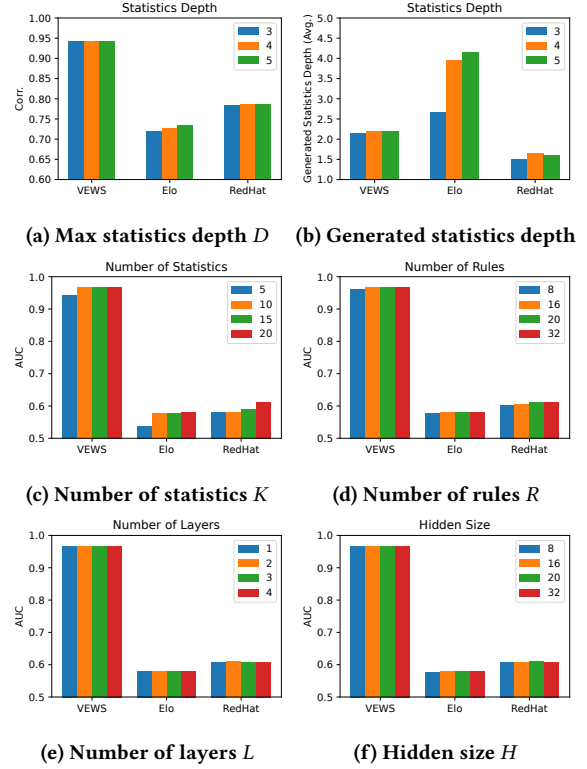
4.3 Experiment 2: Rule Distillation

Now we show the results of rule distillation tasks where the goal is to distill the knowledge of teacher models into rule-based systems.

To the best of our knowledge, this paper is the first study on sequence model explanation. There are no out-of-the-box baseline models, and most of them require fixed-length feature vectors as inputs. In the first group of experiments, we have shown that MCTS-SG can generate informative statistics. So we include the following alternatives with inputs being statistics generated by MCTS-SG: (1) Lasso: We train Lasso models to mimic the outputs of teacher models and tune penalty coefficients so that the number of nonzero weights is approximately R. (2) Lasso (Binary): Considering Lasso can only assign weights to the features and is not a real rule model, we use the binarized statistics (Eq. 10) as input features. (3) CART [3]: This is a classic decision tree model that can learn hierarchical rules. (4) RRL [26]: This is the state-of-the-art neural logical model we have introduced before (Section 3.3.5). Other logical models like SBRL [28], CORELS [2], CRS [27] have been shown inferior to RRL and thus are not included.

We show AUC scores to test the performance of distilled rule models. Also, we use the following score to test if the distilled models are with fidelity to their teachers.

$$\text{Fidelity} = \frac{1}{N \times (N-1)} \sum_{i \neq j} \mathbb{I}_{y_i > y_j} \odot \mathbb{I}_{\tilde{y}_i > \tilde{y}_j},$$

(a) Max statistics depth D   **(b) Generated statistics depth**

(c) Number of statistics K   **(d) Number of rules R**

(e) Number of layers L   **(f) Hidden size H**

Figure 3: Parameter study.

where $\odot$ is the logical XNOR operation. The higher the value, the more faithful the student.

We train comparing models on the training splits, and results on testing splits are summarized in Table 4. We can observe that our proposed two-stage method RuDi achieves the best performance constantly and significantly with $p<0.05$ [8]. RRL achieves comparable results, but the most complicated rule has 40 literals, which reduces its interpretability. On the contrary, our proposed RuDi with two logical layers produces neat rules with at most 4 literals.

4.4 Parameter Sensitivity

In this section, we study the importance of several model parameters with GRU being the teacher, and show results in Figure 3.

First we study how the max depth of statistics D affects the quality of generated statistics. We present the coefficients of multiple correlation in Figure 3(a), and the average depth of generated statistics in Figure 3(b). We can observe that on Elo dataset, as we increase the max depth, the generated statistics become deeper and result in better performance. On the contrary, on VEWS and RedHat, the generated statistics are relatively shallow even the max depth D is large. This means that our proposed statistics generation method can adaptively adjust the complexity of statistics according to datasets thanks to MCTS-based search strategy.

We also study the importance of model parameters by varying the number of statistics K, the number of rules R, the number of layers L, and the hidden size H. From Figure 3(c) we can observe that

Table 4: Experimental results of rule distillation.

	VEWS		Elo		RedHat	
	Fidelity	AUC	Fidelity	AUC	Fidelity	AUC
(Teacher: GRU)		(0.9764)		(0.6345)		(0.6381)
Lasso	0.8863±0.0001	0.9637±0.0001	0.6828±0.0003	0.5491±0.0001	0.7347±0.0011	0.5828±0.0003
Lasso (Binary)	0.8874±0.0001	0.9615±0.0001	0.7166±0.0001	0.5763±0.0001	0.7249±0.0023	0.5769±0.0008
CART	0.8203±0.0001	0.9457±0.0001	0.7363±0.0001	0.5784±0.0001	0.7510±0.0001	0.5920±0.0001
RRL	0.7901±0.0055	0.9149±0.0069	0.7268±0.0038	0.5619±0.0041	0.7621±0.0037	0.5911±0.0045
RuDi (This paper)	**0.9206**±0.0020	**0.9666**±0.0006	**0.7625**±0.0004	**0.5807**±0.0007	**0.8007**±0.0020	**0.6095**±0.0013
(Teacher: LightGBM)		(0.9726)		(0.6211)		(0.6895)
Lasso	0.8865±0.0007	0.9673±0.0001	0.8619±0.0001	0.6114±0.0001	0.6881±0.0008	0.5883±0.0001
Lasso (Binary)	0.8721±0.0005	0.9647±0.0001	0.8789±0.0004	0.6135±0.0001	0.6870±0.0022	0.5868±0.0011
CART	0.8409±0.0001	0.9477±0.0001	0.8858±0.0001	0.6137±0.0001	0.6977±0.0001	0.5879±0.0001
RRL	0.8215±0.0066	0.9234±0.0091	0.8755±0.0071	0.6131±0.0021	0.7002±0.0052	0.6030±0.0072
RuDi (This paper)	**0.9161**±0.0004	**0.9680**±0.0003	**0.9048**±0.0004	**0.6181**±0.0006	**0.7464**±0.0032	**0.6198**±0.0017

Table 5: The learned rules and their meanings on industrial dataset.

Weight	Rule	Meaning
-0.3722	(Sum ∘ Select$_{cardtype=debit}$ > 14) OR (Sum ∘ Select$_{money}$ > 44916)	Users with over 14 debit card transactions or 44916 total payment amount are less likely to be fraudulent.
-0.3042	Sum ∘ Select$_{cardtype=credit}$ > 9	Users with over 9 credit card transactions are less likely to be fraudulent.
+0.5174	(Sum ∘ Select$_{industry=virtual}$ > 9) AND (NOT Sum ∘ Select$_{industry=household}$ > 17)	Users with over 9 transactions in virtual products and no more than 17 transactions in household products are more likely to be fraudulent.
+0.4564	(NOT Sum ∘ Select$_{money}$ > 4457.3) AND (NOT Mean ∘ Select$_{scene=travel}$ > 0)	Users with no more than 4457.3 total payment amount and no travel-related transactions are more likely to be fraudulent.

more statistics would lead to better performance. On VEWS and Elo datasets, it seems 20 statistics are enough, while on RedHat a better result may be achieved by generating more statistics. The number of rules (Figure 3(d)) also positively impacts the performance. RuDi is insensitive to the number of layers and hidden size as shown in Figures 3(e) and 3(f). These results inspire us to achieve better results with more statistics and rules (larger K and R) and maintain the low complexity of each rule (small L and H).

4.5 Case Study

We do a case study on a sampled industrial dataset in this subsection. Tencent Mobile Payment Dataset is a private real-world financial fraudulent detection dataset from the Tencent Mobile Payment[5]. The dataset includes 34.7 million transactions from 222,306 users. Less than 4.56% of users are labeled as fraudsters. The teacher model is an ensemble model with manually-designed features by senior business analysts. A part of distilled rules is summarized in Table 5. Clearly, RuDi successfully explains the black-box scoring model

with a rule-based system, and rules are highly aligned with the industrial scenarios.

5 CONCLUSION

In this paper, we propose RuDi, which can distill the knowledge of a user behavior sequence model into a set of rules. An MCTS-based statistic generation method, MCTS-SG, is proposed to extract informative statistics from behavior sequences. Then a novel neural logical model, GS-NLN, is utilized to compose rules from the statistics. Experiments on three public datasets and one industrial dataset prove the effectiveness of our method.

As for future work, we will explore general dependency-based reward functions instead of linear correlation. Besides, the proposed MCTS-based search method shows potential for supporting relational databases, which is also worth exploring.

---

[5]The dataset is desensitized and sampled properly only for experiment purpose, and does not imply any commercial information. All personal identity information (PII) has been removed. Besides, the experiment was conducted locally on Tencent's server by formal employees.

REFERENCES

[1] Paul D Allison. 1999. *Multiple regression: A primer.* Pine Forge Press.
[2] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. 2017. Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research* (2017).
[3] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. 2017. *Classification and regression trees.* Routledge.
[4] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.
[5] Muhammad Umar Chaudhry and Jee-Hyong Lee. 2018. Feature selection for high dimensional data using monte carlo tree search. *IEEE Access* 6 (2018), 76036–76048.
[6] Muhammad Umar Chaudhry and Jee-Hyong Lee. 2018. MOTiFS: Monte carlo tree search based feature selection. *Entropy* 20, 5 (2018), 385.
[7] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. 2015. Distilling knowledge from deep networks with applications to healthcare domain. *arXiv preprint arXiv:1512.03542* (2015).
[8] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* 7 (2006), 1–30.
[9] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. 2004. Least angle regression. *The Annals of statistics* 32, 2 (2004), 407–499.
[10] Nicholas Frosst and Geoffrey Hinton. 2017. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784* (2017).
[11] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129, 6 (2021), 1789–1819.
[12] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2, 7 (2015).
[13] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. 2016. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318* (2016).
[14] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations.*
[15] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *IEEE International Conference on Data Science and Advanced Analytics.* 1–10.
[16] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. 2016. Explorekit: Automatic feature generation and selection. In *IEEE International Conference on Data Mining.* 979–984.
[17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting

[18] decision tree. In *Conference on Neural Information Processing Systems*, Vol. 30.
[18] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. 2016. Cognito: Automated feature engineering for supervised learning. In *IEEE International Conference on Data Mining.* 1304–1307.
[19] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning.* Springer, 282–293.
[20] Srijan Kumar, Francesca Spezzano, and VS Subrahmanian. 2015. Vews: A wikipedia vandal early warning system. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 607–616.
[21] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. 2017. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327* (2017).
[22] Xuan Liu, Xiaoguang Wang, and Stan Matwin. 2018. Improving the interpretability of deep neural networks with knowledge distillation. In *IEEE International Conference on Data Mining Workshops.* 905–912.
[23] Ali Payani and Faramarz Fekri. 2019. Learning algorithms via neural logic networks. *arXiv preprint arXiv:1904.01554* (2019).
[24] Mengnan Song, Jiasong Wang, Tongtong Zhang, Guoguang Zhang, Ruijun Zhang, and Suisui Su. 2020. Effective Automated Feature Derivation via Reinforcement Learning for Microcredit Default Prediction. In *International Joint Conference on Neural Networks.* 1–8.
[25] Sarah Tan, Rich Caruana, Giles Hooker, and Yin Lou. 2018. Distill-and-compare: Auditing black-box models using transparent model distillation. In *AAAI Conference on Artificial Intelligence.* 303–310.
[26] Zhuo Wang, Wei Zhang, Ning Liu, and Jianyong Wang. 2021. Scalable Rule-Based Representation Learning for Interpretable Classification. In *Conference on Neural Information Processing Systems*, Vol. 34.
[27] Zhuo Wang, Wei Zhang, LIU Ning, and Jianyong Wang. 2020. Transparent classification with multilayer logical perceptrons and random binarization. In *AAAI Conference on Artificial Intelligence*, Vol. 34. 6331–6339.
[28] Hongyu Yang, Cynthia Rudin, and Margo Seltzer. 2017. Scalable Bayesian rule lists. In *International Conference on Machine Learning.* 3921–3930.
[29] Kui Yu, Xianjie Guo, Lin Liu, Jiuyong Li, Hao Wang, Zhaolong Ling, and Xindong Wu. 2020. Causality-based feature selection: Methods and evaluations. *Comput. Surveys* 53, 5 (2020), 1–36.
[30] Shuhan Yuan, Panpan Zheng, Xintao Wu, and Qinghua Li. 2019. Insider threat detection via hierarchical neural temporal point processes. In *IEEE International Conference on Big Data.* 1343–1350.
[31] Shuhan Yuan, Panpan Zheng, Xintao Wu, and Hanghang Tong. 2020. Few-shot insider threat detection. In *Conference on Information and Knowledge Management.* 2289–2292.
[32] Guanghui Zhu, Zhuoer Xu, Xu Guo, Chunfeng Yuan, and Yihua Huang. 2020. DIFER: Differentiable Automated Feature Engineering. *arXiv preprint arXiv:2010.08784* (2020).