Received November 26, 2020, accepted December 18, 2020, date of publication January 6, 2021, date of current version January 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3049569

Extending the Tsetlin Machine With Integer-Weighted Clauses for Increased Interpretability

K. DARSHANA ABEYRATHNA, OLE-CHRISTOFFER GRANMO, AND MORTEN GOODWIN
Centre for Artificial Intelligence Research, University of Agder, 4879 Grimstad, Norway
Corresponding author: K. Darshana Abeyrathna (darshana.abeyrathna@uia.no)

ABSTRACT Building models that are both interpretable and accurate is an unresolved challenge for many pattern recognition problems. In general, rule-based and linear models lack accuracy, while deep learning interpretability is based on rough approximations of the underlying inference. However, recently, the rule-based *Tsetlin Machines* (TMs) have obtained competitive performance in terms of accuracy, memory footprint, and inference speed on diverse benchmarks (image classification, regression, natural language understanding, and game-playing). TMs construct rules using human-interpretable conjunctive clauses in propositional logic. These, in turn, are combined linearly to solve complex pattern recognition tasks. This paper addresses the accuracy-interpretability challenge in machine learning by introducing a TM with integer weighted clauses – the Integer Weighted TM (IWTM). The intent is to increase TM interpretability by reducing the number of clauses required for competitive performance. The IWTM achieves this by weighting the clauses so that a single clause can replace multiple duplicates. Since each TM clause is formed adaptively by a Tsetlin Automata (TA) team, identifying effective weights becomes a challenging online learning problem. We solve this problem by extending each team of TA with another kind of automaton: the *stochastic searching on the line* (SSL) automaton. We evaluate the performance of the new scheme empirically using five datasets, along with a study of interpretability. On average, IWTM uses 6.5 times fewer literals than the vanilla TM and 120 times fewer literals than a TM with real-valued weights. Furthermore, in terms of average memory usage and F1-Score, IWTM outperforms simple Multi-Layered Artificial Neural Networks, Decision Trees, Support Vector Machines, K-Nearest Neighbor, Random Forest, Gradient Boosted Trees (XGBoost), Explainable Boosting Machines (EBMs), as well as the standard and real-value weighted TMs. IWTM finally outperforms Neural Additive Models on Fraud Detection and StructureBoost on CA-58 in terms of Area Under Curve, while performing competitively on COMPAS.

INDEX TERMS Tsetlin machine, integer-weighted Tsetlin machine, interpretable AI, interpretable machine learning, XAI, rule-based learning, decision support system.

I. INTRODUCTION

Interpretable Machine Learning refers to machine learning models that obtain transparency by providing the reasons behind their output. Linear Regression, Logistic Regression, Decision Trees, and Decision Rules are traditional interpretable machine learning approaches. However, as discussed in [1], the degree of interpretability of these algorithms vary. More importantly, such methods struggle with obtaining high accuracy for complex problems, especially in comparison to deep learning. On the other hand, deep learning inference cannot easily be interpreted [2] and is thus less suitable for high-stakes domains such as credit-scoring [3], [4], medicine [5], [6], bioinformatics [7], [8], churn prediction [9], [10] healthcare, and criminal justice [11]. Therefore, developing machine learning algorithms capable of achieving a better trade-off between interpretability and accuracy is of significant importance and continues to be an active area of research.

One of the ways to tackle the above-stated research problem is explaining the deep learning inference. Different

The associate editor coordinating the review of this manuscript and approving it for publication was Pengcheng Liu.

approaches have been proposed for *local* interpretability, i.e., explaining individual predictions [12]. However, they fail to provide clear explanations of model behavior globally [13]. Recently, Agarwal *et al.* [11] proposed a novel deep learning approach that belongs to the family of Neural Additive Models (NAMs). Even though NAMs are inherently interpretable, they are still surpassed by regular deep learning algorithms when it comes to accuracy [11].

Interpretable linear and rule-based methods can sometimes offer a better trade-off between interpretability and accuracy. Learning propositional formulae to represent data patterns has a long history, with association rule learning [14] being one well-known approach, which has been used to predict sequential events [15]. Other examples include the work of Feldman on the hardness of learning formulae in Disjunctive Normal Form (DNF) [16] and Probably Approximately Correct (PAC) learning, which has provided fundamental insight into machine learning as well as a framework for learning formulae in DNF [17]. Approximate Bayesian approaches have recently been introduced to provide more robust learning of rules [18], [19]. Furthermore, hybrid Logistic Circuits have had success in image classification [20]. Logical operators in one layer of the logistic circuit are wired to logical operators in the next layer, and the whole system can be represented as a logistic regression function. This approach uses local search to build a Bayesian model that captures the logical expression, and learns to classify by employing stochastic gradient descent. Yet, in general, rule-based machine learning scales poorly and is prone to noise. Indeed, for data-rich problems, in particular those involving natural language and sensory inputs, rule-based machine learning is inferior to deep learning.

Tsetlin Machines (TMs) are entirely based on logical operators and summation, founded on TA-based bandit learning [21]–[27]. Despite being rule-based, TMs have obtained competitive performance in terms of accuracy, memory footprint, and inference speed on diverse benchmarks, including image classification, regression, natural language understanding, and game-playing. Employing a team of TA [28], a TM learns a linear combination of conjunctive clauses in propositional logic, producing decision rules similar to the branches in a decision tree (e.g., **if** X **satisfies** condition A **and not** condition B **then** Y = 1) [23].

A. RECENT PROGRESS ON TMS

Recent research on TMs reports several distinct TM properties. The TM performs competitively on several classic datasets, such as Iris, Digits, Noisy XOR, and MNIST, compared to Support Vector Machines (SVMs), Decision Trees (DTs), Random Forest (RF), Naive Bayes Classifier, Logistic Regression, and simple Artificial Neural Networks (ANNs) [21]. The TM can further be used in convolution, providing competitive performance on MNIST, Fashion-MNIST, and Kuzushiji-MNIST, in comparison with CNNs, K-Nearest Neighbour (KNN), SVMs, RF, Gradient Boosting, BinaryConnect, Logistic Circuits and ResNet [29]. The TM has also achieved promising results in text classification by using the conjunctive clauses to capture textual patterns [23]. Further, hyper-parameter search can be simplified with multi-granular clauses, eliminating the pattern specificity parameter [25]. By indexing the clauses on the features that falsify them, up to an order of magnitude faster inference and learning has been reported [26]. Furthermore, TM hardware has demonstrated up to three orders of magnitude reduced energy usage and faster learning, compared to neural networks alike [27]. While TMs are binary throughout, binarization schemes open up for continuous input [30]. Finally, the Regression Tsetlin Machine addresses continuous output problems, obtaining on par or better accuracy on predicting dengue incidences, stock price, real estate value and aerofoil noise, in comparison to Regression Trees, RF, and Support Vector Regression [24].

B. PAPER CONTRIBUTIONS

Although TMs are capable of achieving competitive performance levels, they often require a large number of clauses to do so, which impedes interpretability. To overcome this accuracy-interpretability challenge in TMs, we propose the IWTM, encompassing the following contributions.

- We extend each clause with the Stochastic Searching on the Line (SSL) automaton [31]. This automaton is to learn an effective weight for its clause by interacting with the corresponding TA team. As a result, the set of clauses can be rendered significantly more compact, without sacrificing accuracy.
- Through the above scheme, we allow the TM to identify which clauses are inaccurate. These clauses are given smaller weights so that they must team up to obtain high accuracy as a team. Furthermore, the clauses that are sufficiently accurate are assigned larger weights so that they can operate more independently.
- Empirically, we evaluate the IWTM using the eight data sets: Bankruptcy, Balance Scale, Breast Cancer, Liver Disorders, Heart Disease, Fraud Detection, COMPAS, and CA-58. The results show that IWTM on average uses 6.5 times fewer literals than the vanilla TM, and 120.0 times fewer literals than a TM with real-valued weights [22]. Furthermore, performance is competitive with recent state-of-the-art machine learning models.

C. PAPER ORGANIZATION

In Section II, we present the IWTM and describe how each team of TA, composing the clauses, is extended with the SSL. We further discuss the adaptive learning procedure which simultaneously update both clauses and weights. Then in Section III, we evaluate the classification accuracy of IWTM empirically using five datasets, including a study of rule extraction for Bankruptcy prediction in detail and compare against several ANNs, DTs, SVMs, KNN, RF, Gradient Boosted Trees (XGBoost), EBMs (the current state-of-the-art of Generalized Additive Models (GAMs) [32], [33])

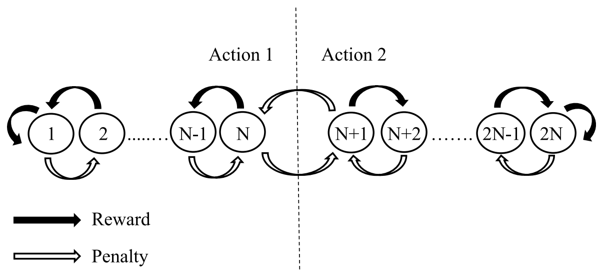

FIGURE 1. Transition graph of a two-action Tsetlin Automaton.

and competing TMs. Further, we contrast the performance of IWTM against reported results on recent state-of-the-art machine learning models, namely NAMs [11] and StructureBoost [34]. Finally, the paper is concluded in Section IV.

II. INTEGER-WEIGHTED TSETLIN MACHINE

In this section, we introduce the integer weighting scheme for TMs. First, we cover the basics of TA, which determine the composition of the TM clauses. Then we introduce the basic TM structure, before we present how integer weights are assigned to the TM clauses. We cover how the individual clauses are trained to learn sub-patterns and how the weight values are updated using SSL. We conclude the section by analysing the computational complexity of the IWTM.

A. TSETLIN AUTOMATA

In the TM, a collective of two-action TA [28] (reviewed in [35]) is used for bandit-based learning. FIGURE 1 shows a two-action TA with $2N$ states. As illustrated, a TA decides its next action from its present state. States from 1 to N trigger Action 1, while states from $N + 1$ to $2N$ trigger Action 2. The TA iteratively interacts with an environment. At each iteration, the environment produces a reward or a penalty in response to the action performed by the TA, according to an unknown probability distribution. Reward feedback reinforces the action performed and penalty feedback weakens it. In order to reinforce an action, the TA changes state towards one of the "deeper" states, direction depending on the current state. Conversely, an action is weakened by changing state towards the center states ($N/N+1$). Hence, penalty feedback eventually forces the TA to change action, shifting its state from N to $N+1$ or vice versa. In this manner, with a sufficient number of states, a TA converges to perform the action with the highest probability of receiving a reward – the optimal action – with probability arbitrarily close to unity, as long as the reward probability is greater than 0.5 [28].

B. TM STRUCTURE

The goal of a basic TM is to categorize input feature vectors $\mathbf{X}$ into one of two classes, $y \in \{0, 1\}$. As shown in FIGURE 2, $\mathbf{X}$ consists of o propositional variables, $x_k \in \{0, 1\}^o$. Further, a TM also incorporates the negation $\neg x_k$ of the variables to capture more sophisticated patterns. Together these are referred to as literals: $\mathbf{L} = [x_1, x_2, \ldots, x_o, \neg x_1, \neg x_2, \ldots, \neg x_o] = [l_1, l_2, \ldots, l_{2o}]$.

1) CLAUSE CONSTRUCTION

At the core of a TM one finds a set of m conjunctive clauses. The conjunctive clauses are to capture the sub-patterns associated with each output y. All of the clauses in the TM receive identical inputs, which is the vector of literals $\mathbf{L}$. We formulate a TM clause as follows:

$$c_j = \bigwedge_{k \in I_j} l_k. \tag{1}$$

Notice that each clause, indexed by j, includes distinct literals. The indexes of the included literals are contained in the set $I_j \subseteq \{1, \ldots, 2o\}$. For the special case of $I_j = \emptyset$, i.e., an empty clause, we have:

$$c_j = \begin{cases} 1 & \textbf{during} \text{ learning} \\ 0 & \textbf{otherwise}. \end{cases} \tag{2}$$

That is, during learning, empty clauses output 1 and during classification they output 0.

It is the two-action TAs that assign literals to clauses. Each clause is equipped with $2 \times o$ TAs, one per literal k, as shown in *Clause-1* of FIGURE 2. The TA states from 1 to N map to the *exclude* action, which means that the corresponding literal is excluded from the clause. For states from $N + 1$ to $2N$, the decision becomes *include*, i.e., the literal is included instead. The states of all the TAs in all of the clauses are jointly stored in the matrix $\mathbf{A}$: $\mathbf{A} = (a_{j,k}) \in \{1, \ldots, 2N\}^{m \times 2o}$, with j referring to the clause and k to the literal. Hence, the literal indexes contained in the set I_j can be expressed as $I_j = \{k | a_{j,k} > N, 1 \leq k \leq 2o\}$.

2) CLAUSE OUTPUT

The output of the clauses can be produced as soon as the decisions of the TAs are known. Since the clauses are conjunctive, they evaluate to 0 if any of the literals included are of value 0. For a given input X, let the set I_X^1 contain the indexes of the literals of value 1. Then the output of clause j can be expressed as:

$$c_j = \begin{cases} 1 & \text{if} \quad I_j \subseteq I_X^1, \\ 0 & \text{otherwise}. \end{cases} \tag{3}$$

In the following, we let the vector $\mathbf{C}$ denote the complete set of clause outputs $\mathbf{C} = (c_j) \in \{0, 1\}^m$, as defined above.

3) CLASSIFICATION IN TM

The TM classifies data into two classes, which means that sub-patterns associated with both classes must be identified. This is done by dividing clauses into two groups. Clauses with odd index are assigned positive polarity (c_j^+), and they are to capture sub-patterns of output $y = 1$. Clauses with even index, on the other hand, are assigned negative polarity (c_j^-) and they seek the sub-patterns of output $y = 0$.

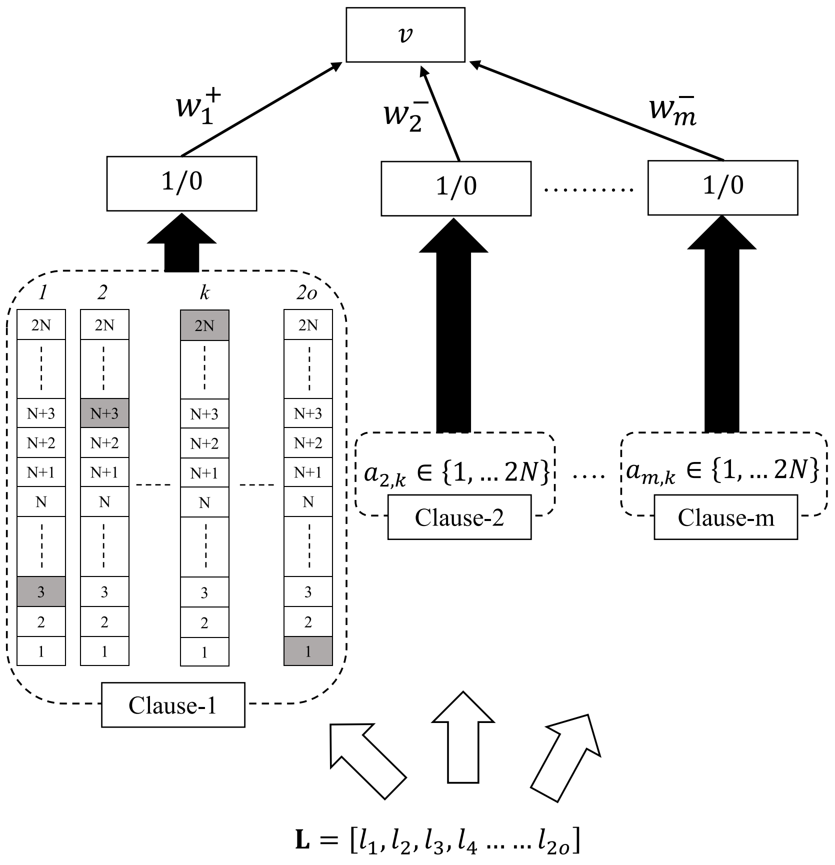
v
w_1^+
w_2^-
w_m^-
1/0
1/0
1/0
1 2 k 2o
2N
N+3
N+2
N+1
N
3
2
1
$a_{2,k} \in \{1, \ldots 2N\}$
$a_{m,k} \in \{1, \ldots 2N\}$
Clause-1
Clause-2
Clause-m
$\mathbf{L} = [l_1, l_2, l_3, l_4 \ldots \ldots l_{2o}]$

FIGURE 2. The Integer Weighted Tsetlin Machine structure.

Once a clause recognizes a sub-pattern, it outputs 1, casting a vote according to its polarity. The final output of the TM is found by summing up the clause outputs, subtracting the votes of the clauses with negative polarity from the votes of the clauses with positive polarity. With v being the difference in clause output, $v = \sum_j c_j^+ - \sum_j c_j^-$, the output of the TM is decided as follows:

$$\hat{y} = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0. \end{cases} \quad (4)$$

C. INCORPORATING INTEGER WEIGHTS INTO THE TM

In contrast to the weighting scheme proposed by Phoulady *et al.* [22], which employs real-valued weights that require multiplication and an additional hyperparameter, our scheme is parameter-free and uses increment and decrement operations to update the weights.

1) CLASSIFICATION IN IWTM

The weights decide the impact of each clause during classification, replacing Eq. 4 with:

$$\hat{y} = \begin{cases} 1 & \text{if } \sum_j w_j^+ c_j^+ - \sum_j w_j^- c_j^- \geq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Above, w_j^+ is the weight of the j^{th} clause with positive polarity, while w_j^- is the weight of the j^{th} clause with negative polarity.

D. LEARNING PROCEDURE

In this sub-section, we first discuss how individual clauses are trained to learn sub-patterns. Then the procedure of updating weights is explained in detail.

1) CLAUSE LEARNING

A TM learns online, processing one training example (X, y) at a time. Within each clause, a local team of TAs decide the clause output by selecting which literals are included in the clause. Jointly, the TA teams thus decide the overall output of the TM, mediated through the clauses. This hierarchical structure is used to update the state of each TA, with the purpose of maximizing output accuracy. We achieve this with two kinds of reinforcement: Type I and Type II feedback. Type I and Type II feedback control how the individual TAs either receive a reward, a penalty, or inaction feedback, depending on the context of their actions. In the following we focus on clauses with positive polarity. For clauses with negative polarity, Type I feedback replaces Type II, and vice versa.

Type I feedback: Type I feedback consists of two sub-feedback schemes: Type Ia and Type Ib. Type Ia feedback reinforces *include* actions of TAs whose corresponding literal value is 1, however, only when the clause output also is 1. Type Ib feedback combats over-fitting by reinforcing *exclude* actions of TAs when the corresponding literal is 0 or when the clause output is 0. Consequently, both Type Ia and Type Ib feedback gradually force clauses to output 1.

Type I feedback is given to clauses with positive polarity when $y = 1$. This stimulates suppression of *false negative* output. To diversify the clauses, they are targeted for Type I feedback stochastically as follows:

$$p_j^+ = \begin{cases} 1 & \text{with probability } \dfrac{T - \max(-T, \min(T, v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Here, p_j^+ is the decision whether to target clause j with positive polarity for feedback. The user set target T for the clause output sum v decides how many clauses should be involved in learning a particular sub-pattern. Higher T increases the robustness of learning by allocating more clauses to learn each sub-pattern. The decisions for the complete set of positive clauses are organized in the vector $\mathbf{P}^+ = (p_j^+) \in \{0, 1\}^{\frac{m}{2}}$. Similarly, decisions for the complete set of negative clauses can be found in $\mathbf{P}^- = (p_j^-) \in \{0, 1\}^{\frac{m}{2}}$.

If a clause is eligible to receive feedback per Eq. 6, the individual TAs of the clause are singled out stochastically using a user-set parameter s ($s \geq 1$). The decision whether the k^{th} TA of the j^{th} clause of positive polarity is to receive Type Ia feedback, $r_{j,k}^+$, and Type Ib feedback, $q_{j,k}^+$, are stochastically made as follows:

$$r_{j,k}^+ = \begin{cases} 1 & \text{with probability } \dfrac{s-1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

$$q_{j,k}^+ = \begin{cases} 1 & \text{with probability } \dfrac{1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

The above decisions are respectively stored in the two matrices $\mathbf{R}^+$ and $\mathbf{Q}^+$, i.e., $\mathbf{R}^+ = (r_{j,k}^+) \in \{0, 1\}^{m \times 2o}$ and $\mathbf{Q}^+ = (q_{j,k}^+) \in \{0, 1\}^{m \times 2o}$. Using the complete set of conditions, TA indexes selected for Type Ia are $I^{\text{Ia}} = \{(j, k) | l_k = 1 \land c_j^+ = 1 \land p_j^+ = 1 \land r_{j,k}^+ = 1\}$. Similarly TA indexes selected for Type Ib are $I^{\text{Ib}} = \left\{(j, k) | (l_k = 0 \lor c_j^+ = 0) \land p_{j,y}^+ = 1 \land q_{j,k}^+ = 1\right\}$.

Once the indexes of the TAs are identified, the states of those TAs are updated. Available updating options are $\oplus$ and $\ominus$, where $\oplus$ adds 1 and $\ominus$ subtracts 1 from the current state. The processing of the training example ends with the state matrix $\mathbf{A}^+$ being updated as follows: $\mathbf{A}^+ \leftarrow \left(\mathbf{A}^+ \oplus I^{\text{Ia}}\right) \ominus I^{\text{Ib}}$.

Type II feedback: Type II feedback is given to clauses with positive polarity for target output $y = 0$. Clauses to receive Type II feedback are again selected stochastically. The decision for the j^{th} clause of positive polarity is made as follows:

$$p_j^+ = \begin{cases} 1 & \text{with probability } \dfrac{T + \max(-T, \min(T, v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

The idea behind Type II feedback is to change the output of the affected clauses from 1 to 0. This is achieved simply by including a literal of value 0 in the clause. TAs selected for Type II can accordingly be found in the index set: $I^{\text{II}} = \{(j, k) | l_k = 0 \land c_j^+ = 1 \land p_j^+ = 1\}$. To obtain the intended effect, these TAs are reinforced to include their literals in the clause by increasing their corresponding states: $\mathbf{A}^+ \leftarrow \mathbf{A}^+ \oplus I^{\text{II}}$.

When training has been completed, the final decisions of the TAs are recorded, and the resulting clauses can be deployed for operation.

2) WEIGHT LEARNING

The learning of weights is based on increasing the weight of clauses that receive Type Ia feedback (due to true positive output) and decreasing the weight of clauses that receive Type II feedback (due to false positive output). The overall rationale is to determine which clauses are inaccurate and thus must team up to obtain high accuracy as a team (low weight clauses), and which clauses are sufficiently accurate to operate more independently (high weight clauses).

The weight updating procedure is summarized in Algorithm 1 and in the flowchart in FIGURE 3. Here, $w_j(n)$ is the weight of clause j at the n^{th} training round (ignoring polarity to simplify notation). The first step of a training round is to calculate the clause output as per Eq. 3. The weight of a clause is only updated if the clause output $c_j(n)$ is 1 and the clause has been selected for feedback ($p_j = 1$). Then the polarity of the clause and the class label y decide the type of feedback given. That is, like a regular TM, positive polarity clauses receive Type Ia feedback if the clause output is a true positive and Type II feedback if the clause output is a false positive. For clauses with negative polarity, the feedback types switch roles.

When clauses receive Type Ia or Type II feedback, their weights are updated accordingly. We use the stochastic searching on the line (SSL) automaton to learn appropriate weights. SSL is an optimization scheme for unknown stochastic environments pioneered by Oommen [31]. The goal is to find an unknown location λ^* within a search interval $[0, 1]$. In order to find λ^*, the only available information for the Learning Mechanism (LM) is the possibly faulty feedback from its attached environment (E).

In SSL, the search space λ is discretized into N points, $\{0, 1/N, 2/N, \ldots, (N-1)/N, 1\}$, with N being the discretization resolution. During the search, the LM has a location $\lambda \in \{0, 1/N, 2/N, \ldots, (N-1)/N, 1\}$, and can freely move to the left or to the right from its current location. The environment E provides two types of feedback: $E = 1$ is the environment suggestion to increase the value of λ by one

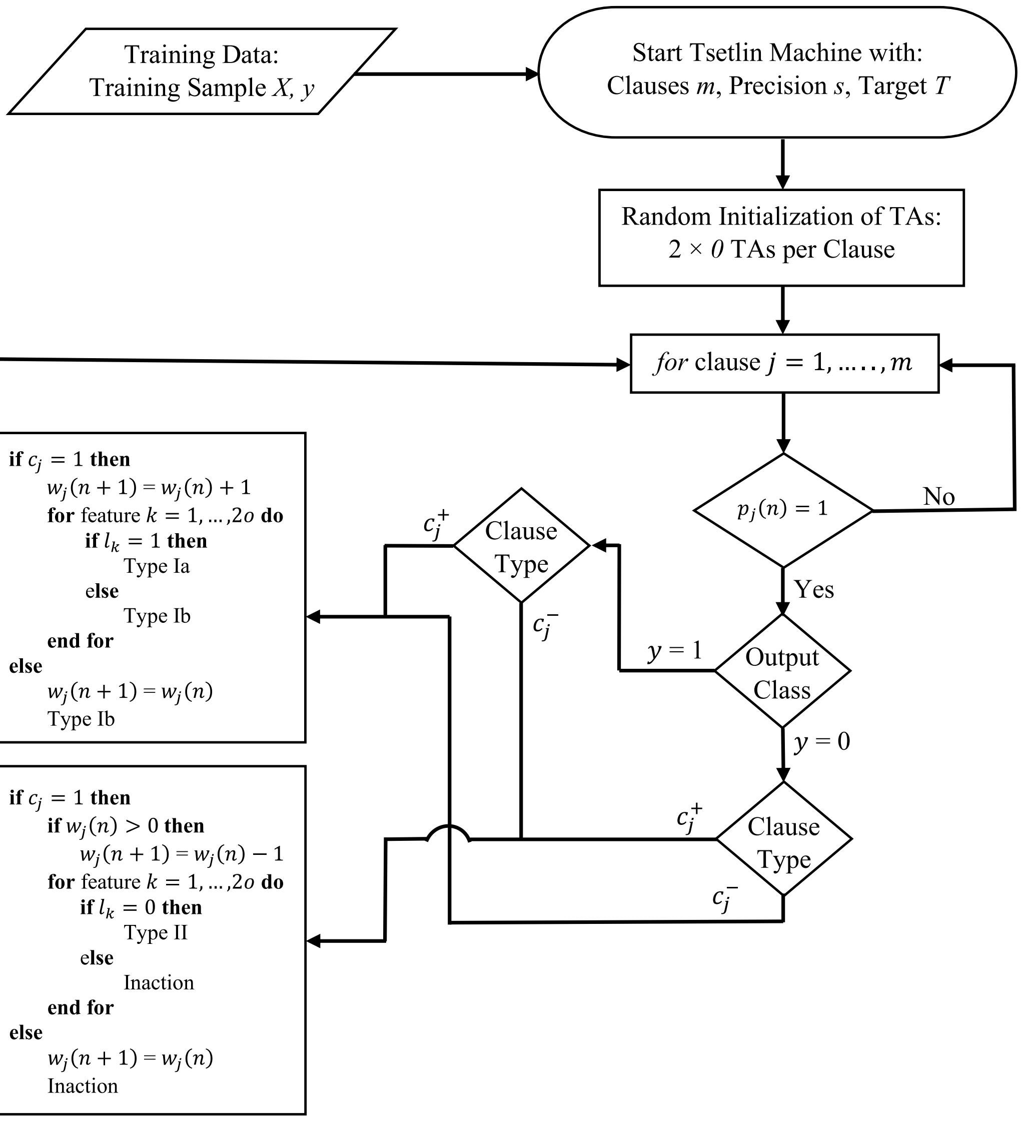

FIGURE 3. The complete learning process of the IWTM in a flowchart.

step, and $E = 0$ is the environment suggestion to decrease the value of λ by one step. The next location of λ, $\lambda(n+1)$ can thus be expressed as follows:

$$\lambda(n+1) = \begin{cases} \lambda(n) + 1/N, & \text{if } E(n) = 1, \\ \lambda(n) - 1/N, & \text{if } E(n) = 0. \end{cases} \tag{10}$$

$$\lambda(n+1) = \begin{cases} \lambda(n), & \text{if } \lambda(n) = 1 \text{ and } E(n) = 1, \\ \lambda(n), & \text{if } \lambda(n) = 0 \text{ and } E(n) = 0. \end{cases} \tag{11}$$

Asymptotically, the learning mechanics is able to find a value arbitrarily close to λ^* when $N \to \infty$ and $n \to \infty$.

In our case, the search space of clause weights is $[0, \infty]$, so we use resolution $N = 1$, with no upper bound for λ. Accordingly, we operate with integer weights. As seen in FIGURE 3, if the clause output is a true positive, we simply increase the weight by 1. Conversely, if the clause output is a false positive, we decrease the weight by 1.

By following the above procedure, the goal is to make low precision clauses team up by giving them low weights, so that they together can reach the summation target T. By teaming up, precision increases due to the resulting ensemble effect. Clauses with high precision, however, gets a higher weight, allowing them to operate more independently.

The above weighting scheme has several advantages. First of all, increment and decrement operations on integers are computationally less costly than multiplication based updates of real-valued weights. Additionally, a clause with an integer weight can be seen as multiple copies of the same clause, making it more interpretable than real-valued weighting, as studied in the next section. Additionally, clauses can be turned completely off by setting their weights to 0 if they do not contribute positively to the classification task.

Algorithm 1 The Complete IWTM Learning Process

```
1: Input: Training data $(\mathbf{X}, y)$, $m$, $T$, $s$
2: Initialize: Random initialization of TAs
3: Begin: $n^{th}$ training round
4: for $j = 1, \ldots, m$ do if $p_j = 1$                          ▷ Eq. (6) and (9)
5:     if ($y = 1$ and $j$ is odd) or ($y = 0$ and $j$ is even) then
6:         if $c_j = 1$ then                                      ▷ Eq. (3)
7:             $w_j(n+1) \leftarrow w_j(n) + 1$                   ▷ Eq. (10-11)
8:             for feature $k = 1, \ldots, 2o$ do
9:                 if $l_k = 1$ then
10:                    Type Ia Feedback
11:                else:
12:                    Type Ib Feedback
13:                end if
14:            end for
15:        else:
16:            $w_j(n+1) \leftarrow w_j(n)$                       ▷ Eq. (10-11)
17:            Type Ib Feedback
18:        end if
19:    else: ($y = 1$ and $j$ is even) or ($y = 0$ and $j$ is odd)
20:        if $c_j = 1$ then                                      ▷ Eq. (3)
21:            if $w_j(n) > 0$ then
22:                $w_j(n+1) \leftarrow w_j(n) - 1$               ▷ Eq. (10-11)
23:            end if
24:            for feature $k = 1, \ldots, 2o$ do
25:                if $l_k = 0$ then
26:                    Type II Feedback
27:                else:
28:                    Inaction
29:                end if
30:            end for
31:        else:
32:            $w_j(n+1) \leftarrow w_j(n)$                       ▷ Eq. (10-11)
33:            Inaction
34:        end if
35:    end if
36: end for
```

E. COMPUTATIONAL COMPLEXITY OF THE IWTM

To evaluate computational complexity, we introduce the three constants α, β, and γ, where α represents the computational cost to perform the conjunction of two bits, β is the computational cost of computing the summation of two integers, and γ is the computational cost to update the state of a single automaton (TA or SSL) in IWTM.

We here consider worst-case computational costs for training and testing, assuming all the TAs in all of the clauses are operative and updated. In a TM with m clauses and when the input vector consists of o features, the TM performs $2o \times m$ number of TA updates for a single training sample. In the IWTM, this becomes $(2o + 1) \times m$ updates due to the weights. Hence, we compute the computational cost of updating TA states during the IWTM training as $\gamma \times (2o + 1) \times m$. The cost is simply d times higher when there are d number of training samples in the dataset, i.e., $d \times \gamma \times (2o + 1) \times m$.

The other two TM operations are to compute the output of clauses and to sum up the outputs to get the vote difference. Assuming all the TAs in all of the clauses have decided to include their corresponding literals in the clause, the computational cost for obtaining the clause outputs becomes $\alpha \times 2o \times m$. Once the clause outputs are ready, the vote difference is calculated. This requires a computational cost of $\beta \times (m-1)$. We only encounter the above stated computational requirements during the testing phase. However, both these components have to be multiplied with the number of samples to obtain the training cost, i.e., $d[\alpha \times 2o \times m + \beta \times (m-1)]$.

Accordingly, we can formulate the computational complexity as a function of d which exhibits how the computational complexity of the IWTM varies with d during the IWTM training. Combining the costs of updating TAs, computing clause outputs, and calculating the vote difference, we get a linear function $f(d)$:

$$f(d) = d[\gamma \times (2o + 1) \times m + \alpha \times 2o \times m + \beta \times (m - 1)].$$

Then, using the Big O notation [36], the computational complexity of IWTM training becomes $\mathcal{O}(d)$, which means that the complexity of the IWTM increases linearly with the number of training samples d. Similarly, complexity grows linearly with the number of clauses m and with the number of inputs o.

III. EMPIRICAL EVALUATION

In this section, we empirically evaluate the impact of integer weighting on the TM using five real-world datasets. Three of these datasets are from the health sector: *Breast Cancer dataset*, *Liver Disorder dataset*, and *Heart Disease dataset*, while the two other ones are the *Balance Scale* and *Corporate Bankruptcy* datasets. We use the latter dataset to examine interpretability more closely.

In the comparison, the IWTM is compared with the vanilla TM as well as the TM with real-valued weights (RWTM). Additionally, we contrast performance against the standard machine learning techniques Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), Decision Trees (DTs), K-Nearest Neighbor (KNN), Random Forest (RF), Gradient Boosted Trees (XGBoost) [37], Explainable Boosting Machines (EBMs) [32] along with two recent state-of-the-art machine learning approaches: Neural Additive Models [11] and StructureBoost [34]. For comprehensiveness, three ANN architectures are used: ANN-1 – with one hidden layer of 5 neurons; ANN-2 – with two hidden layers of 20 and 50 neurons each, and ANN-3 – with three hidden layers and 20, 150, and 100 neurons.

In the experiments, we use the binarization scheme based on thresholding proposed in [30] for continuous and categorical features. The results are average measures over 50 independent experiment trials. We used 80% of the data for training and 20% for testing. Hyperparameters were set using manual binary search.

TABLE 1. Binarizing categorical features in the Bankruptcy dataset.

Category	Integer Code	Thresholds		
		≤ 0	≤ 1	≤ 2
A	0	1	1	1
N	1	0	1	1
P	2	0	0	1

TABLE 2. Clauses produced by TM, RWTM, and IWTM for *m* = 10.

Clause	Class	TM	RWTM		IWTM	
		Literals	Literals	w	Literals	w
1	1	¬11	¬14	0.0287	-	5
2	0	¬13, 14	¬13, 14	0.0001	¬13, 14	6
3	1	¬14	¬14	0.0064	-	5
4	0	¬13, 14	¬13, 14	0.0001	¬13, 14	2
5	1	¬14	¬11	0.7001	¬11	0
6	0	¬13, 14	¬13, 14	0.0001	¬13, 14	2
7	1	-	¬14	0.1605	-	7
8	0	¬13, 14	¬13, 14	0.0001	¬13, 14	5
9	1	¬14	¬14	0.1425	¬14	1
10	0	¬13, 14	¬13, 14	0.0001	¬13, 14	6
Accuracy (Training/Testing)		0.98/1.00	0.99/0.96		0.98/1.00	

A. BANKRUPTCY

In finance, accurate prediction of bankruptcy is important to mitigate economic loss [38]. However, since the decisions made related to bankruptcy can have critical consequences, interpretable machine learning algorithms are often preferred over black-box methods.

Consider the historical records of 250 companies in the Bankruptcy dataset.[1] Each record consists of six features pertinent to predicting bankruptcy: 1) Industrial Risk, 2) Management Risk, 3) Financial Flexibility, 4) Credibility, 5) Competitiveness, and 6) Operation Risk. These are categorical features where each feature can be in one of three states: Negative (N), Average (A), or Positive (P). The two target classes are Bankruptcy and Non-bankruptcy. While the class output is binary, the features are ternary. We thus binarize the features using thresholding [30], as shown in TABLE 1. Thus, the binarized dataset contains 18 binary features.

We first investigate the behavior of TM, RWTM, and IWTM with very few clauses (10 clauses). The clauses produced are summarized in TABLE 2.

Five out of the ten clauses (clauses with odd index) vote for class 1 and the remaining five (those with even index) vote for class 0. In the TM, the first clause contains just one literal, which is the negation of feature 11. From the binarized feature set, we recognize that the 11^{th} feature is *Negative Credibility*. Likewise, clauses 2, 4, 6, 8, 10 contain the same two literals – the negation of *Average Competitiveness* and *Negative Competitiveness* non-negated. The clauses 3, 5, and 9, on the other hand, include *Negative Competitiveness* negated. There is also a free vote for class 1 from the "empty" clause 7, which is ignored during classification.

[1] Available from https://archive.ics.uci.edu/ml/datasets/qualitative_bankruptcy.

TABLE 3. Clauses produced by TM, RWTM, and IWTM for *m* = 2.

Clause	Vote for class	TM	RWTM		IWTM	
		Literals	Literals	w	Literals	w
1	1	¬14	¬14	0.5297	¬14	0
2	0	¬13, 14	¬13, 14	7.0065	¬13, 14	3
Accuracy (Training/Testing)		0.99/0.96	0.99/0.96		0.96/0.98	

The clause outputs of the TM in TABLE 2 are visualized in FIGURE 4. From the figure, it is clear how the trained TM operates. It uses only two features, *Credibility* and *Competitiveness*, and their negations. Further, observe how the TM implicitly introduces weighting by duplicating the clauses.

TABLE 2 also contains the clauses learnt by RWTM and IWTM. The most notable difference is that RWTM puts little emphasis on the clauses for class 0, giving them weight 0.0001. Further, it puts most emphasis on the negation of *Negative Credibility* and *Negative Competitiveness*. The IWTM, on the other hand, like the TM, focuses on the negation of *Average Competitiveness* and non-negated *Negative Competitiveness*. Note also that, without loss of accuracy, IWTM simplifies the set of rules by turning off negated *Negative Credibility* by giving clause 5 weight zero. The three literals remaining are the negation of *Average Competitiveness* and *Negative Competitiveness*, negated and non-negated. Because *Negative Competitiveness* implies negated *Average Competitiveness*, IWTM ends up with the simple classification rule (ignoring the weights):

$$\text{Outcome} = \begin{cases} \text{Bankruptcy} & \textbf{if } \text{Negative Competitiveness} \\ \text{Non-bankruptcy} & \textit{otherwise}. \end{cases} \tag{12}$$

By asking the TMs to only produce two clauses, we can obtain the above rule more directly, as shown in TABLE 3. As seen, again, TM, RWTM, and IWTM achieve similar accuracy. Further, IWTM turns off *Negative Competitiveness* negated, producing the simplest rule set of the three approaches.

The previous accuracy results represent the majority of experiment trials. However, some of the trials fail to reach an optimal TM configuration. Instead of re-running learning a few times, one can increase the number of clauses for increased robustness in *every trial*. This comes at the cost of reduced interpretability, however. TABLE 4, TABLE 5, and TABLE 6 contain average performance (Precision, Recall, F1-Score, Accuracy, Specificity) over 50 experiment trials, showing how robustness increases with more clauses, up to a certain point.

TABLE 4 reports the results for a standard TM. Our goal is to maximize F1-Score, since accuracy can be misleading for imbalanced datasets. Notice how the F1-Score increases with the number of clauses, peaking when m equals 2000. At this point, the average number of literals (include actions) across the clauses is 3622 (rounded to nearest integer). The RWTM behaves similarly, as seen in TABLE 5. However, it peaks with an F1-Score of 0.999 at $m = 2000$. Then 3478 literals have been included on average.

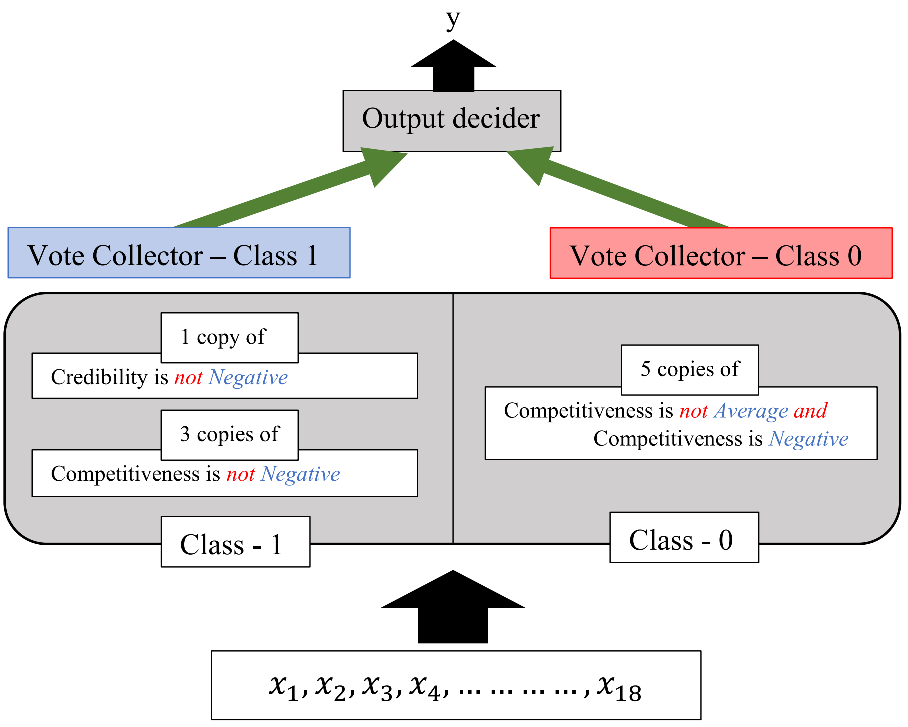
y
Output decider
Vote Collector – Class 1
Vote Collector – Class 0
1 copy of
Credibility is not Negative
3 copies of
Competitiveness is not Negative
5 copies of
Competitiveness is not Average and Competitiveness is Negative
Class - 1
Class - 0
$x_1, x_2, x_3, x_4, \ldots\ldots\ldots\ldots, x_{18}$

FIGURE 4. TM classification process for the Bankruptcy dataset.

TABLE 4. Performance of TM on Bankruptcy dataset.

m	2	10	100	500	2000	8000
Precision	0.754	0.903	0.997	0.994	0.996	0.994
Recall	1.000	1.000	1.000	0.998	1.000	1.000
F1-Score	0.859	0.948	0.984	0.996	0.998	0.997
Accuracy	0.807	0.939	0.998	0.996	0.998	0.996
Specificity	0.533	0.860	0.995	0.993	0.996	0.990
No. of Lit.	19	88	222	832	3622	15201

TABLE 5. Performance of RWTM on Bankruptcy dataset.

m	2	10	100	500	2000	8000
Precision	0.736	0.860	0.885	0.996	0.998	0.997
Recall	1.000	1.000	1.000	0.998	1.000	0.998
F1-Score	0.846	0.924	0.933	0.997	0.999	0.998
Accuracy	0.792	0.906	0.915	0.997	0.999	0.997
Specificity	0.511	0.785	0.824	0.996	0.998	0.995
No. of Lit.	20	97	893	825	3478	14285

TABLE 6. Performance of IWTM on Bankruptcy dataset.

m	2	10	100	500	2000	8000
Precision	0.636	0.765	0.993	0.998	0.998	0.991
Recall	1.000	1.000	1.000	1.000	1.000	1.000
F1-Score	0.774	0.862	0.996	0.999	0.999	0.995
Accuracy	0.654	0.814	0.996	0.999	0.999	0.995
Specificity	0.177	0.584	0.991	0.998	0.998	0.990
No. of Lit.	8	45	148	379	1969	8965

The IWTM, on the other hand, achieves its best F1-Score when *m* is 500. At that point, an average of 379 literals are included (only considering clauses with a weight larger than 0), which is significantly smaller than what was obtained with TM and RWTM.

How the number of literals increases with the number of clauses is shown in FIGURE 5. The IWTM consistently

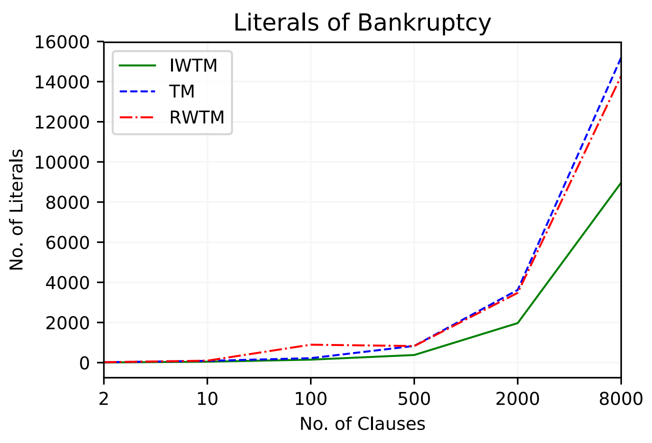

FIGURE 5. The number of literals included in different TM setups to work with Bankruptcy dataset.

produces fewer literals than the other two schemes, and the difference increases with the number of clauses.

We finally compare the performance of TM, RWTM, and IWTM against several standard machine learning algorithms, namely ANN, DT, SVM, KNN, RF, XGBoost, and EBM. The performance of all of the techniques is compiled in TABLE 7. The best F1-Score is obtained by RWTM and IWTM, which produce identical results expect that IWTM uses fewer literals, less memory during training, and less training time per epoch. Also, in terms of Accuracy, RWTM and IWTM obtains the best average results. Further notice that all of the TMs achieve a Recall of 1.0. Additionally,

TABLE 7. Performance comparison for Bankruptcy dataset.

	Precision	Recall	F1	Accuracy	Specificity	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.990	1.000	0.995	0.994	0.985	-	≈ 942.538KB / ≈ 26.64KB	0.227 sec.
ANN-2	0.995	0.997	0.996	0.995	0.993	-	≈ 3476.76KB / ≈ 590.76KB	0.226 sec.
ANN-3	0.997	0.998	0.997	0.997	0.995	-	≈ 28862.65KB / ≈ 1297.12KB	0.266 sec.
DT	0.988	1.000	0.993	0.993	0.985	-	≈ 0.00KB / ≈ 0.00KB	0.003 sec.
SVM	1.000	0.989	0.994	0.994	1.000	-	≈ 90.11KB / ≈ 0.00KB	0.001 sec.
KNN	0.998	0.991	0.995	0.994	0.998	-	≈ 0.00KB / ≈ 286.71KB	0.001 sec.
RF	0.979	0.923	0.949	0.942	0.970	-	≈ 180.22KB / ≈ 0.00KB	0.020 sec.
XGBoost	0.996	0.977	0.983	0.983	0.992	-	≈ 4964.35KB / ≈ 0.00KB	0.009 sec.
EBM	0.987	1.000	0.993	0.992	0.980	-	≈ 1425.40KB / ≈ 0.00KB	13.822 sec.
TM	0.997	1.000	0.998	0.998	0.995	3622	≈ 0.00KB / ≈ 0.00KB	0.148 sec.
RWTM	0.998	1.000	0.999	0.999	0.998	3478	≈ 94.20KB / ≈ 0.00KB	0.148 sec.
IWTM	0.998	1.000	0.999	0.999	0.998	379	≈ 0.00KB / ≈ 0.00KB	0.013 sec.

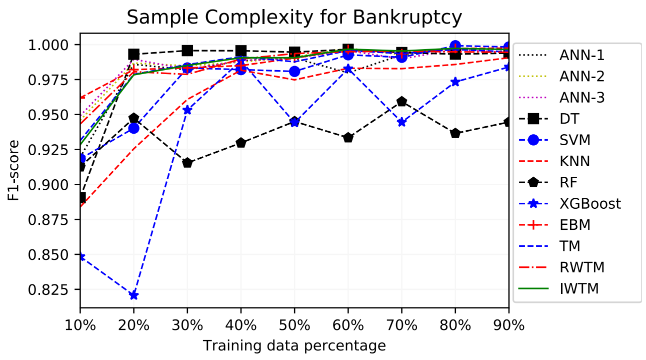

FIGURE 6. Sample complexity analysis for the Bankruptcy dataset.

only DT, TM, and IWTM require memory close to zero, both during training and testing. RWTM uses more memory in training than the other two TMs since it has to represent the weights of those 2000 clauses as floating point numbers.

We also perform a sample complexity analysis for all the techniques. As FIGURE 6 manifests, all the techniques except RF and XGBoost surpasses an F1-Score of 0.975 when training on 40 percent of the data. The F1-Scores of RF and XGBoost start relatively low and fluctuate around 0.950 after 40% of the training samples have been processed.

B. BALANCE SCALE

For the remaining datasets, we focus on TM, RWTM and IWTM configurations that provide robust performance over interpretability, comparing with selected machine learning techniques.

We first cover the Balance Scale dataset,[2] which contains three classes: balance scale tip to the right, tip to the left, or is in balance. The dataset also contains four features: 1) size of the weight on the left-hand side, 2) distance from the center to the weight on the left, 3) size of the weight on the right-hand side, and 4) distance from the center to the weight on the right. To make the output binary, we remove the "balanced" class ending up with 576 data samples.

TABLE 8, TABLE 9, and TABLE 10 contain the results of TM, RWTM, and IWTM, respectively, with varying m. For

[2]Available from http://archive.ics.uci.edu/ml/datasets/balance+scale.

TABLE 8. Performance of TM on Balance Scale dataset.

m	2	10	100	500	2000	8000
Precision	0.647	0.820	0.966	0.949	0.926	0.871
Recall	0.986	0.965	0.930	0.934	0.884	0.746
F1-Score	0.781	0.886	0.945	0.933	0.880	0.749
Accuracy	0.728	0.875	0.948	0.936	0.889	0.780
Specificity	0.476	0.782	0.966	0.935	0.905	0.819
No. of Lit.	17	77	790	3406	15454	60310

TABLE 9. Performance of RWTM on Balance Scale dataset.

m	2	10	100	500	2000	8000
Precision	0.631	0.779	0.885	0.914	0.919	0.916
Recall	0.973	0.965	0.970	0.942	0.911	0.949
F1-Score	0.765	0.860	0.925	0.926	0.914	0.931
Accuracy	0.709	0.842	0.921	0.927	0.917	0.931
Specificity	0.457	0.720	0.874	0.915	0.924	0.913
No. of Lit.	18	90	890	4406	17454	66310

TABLE 10. Performance of IWTM on Balance Scale dataset.

m	2	10	100	500	2000	8000
Precision	0.655	0.811	0.946	0.934	0.936	0.853
Recall	0.973	0.965	0.966	0.920	0.908	0.830
F1-Score	0.783	0.881	0.954	0.916	0.905	0.800
Accuracy	0.719	0.868	0.953	0.917	0.905	0.808
Specificity	0.444	0.767	0.941	0.912	0.896	0.794
No. of Lit.	9	39	120	710	2602	9607

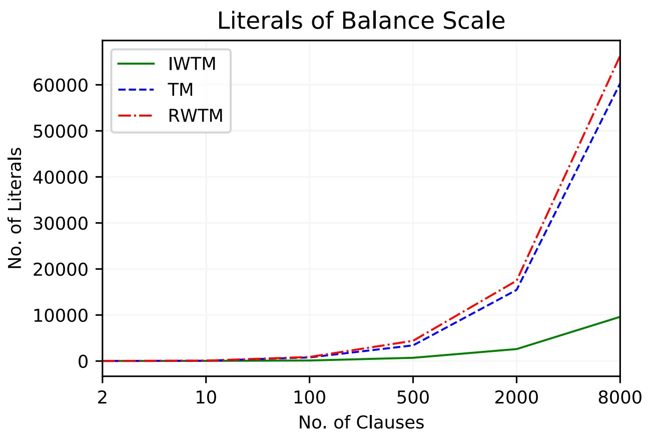

FIGURE 7. The number of literals included in different TM setups to work with Balance Scale dataset.

the TM, F1-Score peaks at 0.945 when $m = 200$. At the peak, 790 literals are used on average. RWTM obtains its

TABLE 11. Performance comparison for Balance Scale dataset.

	Precision	Recall	F1	Accuracy	Specificity	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.993	0.987	0.990	0.990	0.993	-	$\approx$ 966.57KB / $\approx$ 24.56KB	0.614 sec.
ANN-2	0.995	0.995	0.995	0.995	0.994	-	$\approx$ 3612.65KB / $\approx$ 589.82KB	0.588 sec.
ANN-3	0.995	0.995	0.995	0.995	0.995	-	$\approx$ 33712.82KB / $\approx$ 1478.64KB	0.678 sec.
DT	0.984	0.988	0.986	0.986	0.985	-	$\approx$ 131.07KB / $\approx$ 0.00KB	0.007 sec.
SVM	0.887	0.889	0.887	0.887	0.884	-	$\approx$ 65.53KB / $\approx$ 241.59KB	0.001 sec.
KNN	0.968	0.939	0.953	0.953	0.969	-	$\approx$ 249.77KB / $\approx$ 126.87KB	0.001 sec.
RF	0.872	0.851	0.859	0.860	0.871	-	$\approx$ 0.00KB / $\approx$ 0.00KB	0.021 sec.
XGBoost	0.942	0.921	0.931	0.931	0.942	-	$\approx$ 1126.39KB / $\approx$ 0.00KB	0.030 sec.
EBM	1.000	1.000	1.000	1.000	1.000	-	$\approx$ 1642.49KB / $\approx$ 0.00KB	15.658 sec.
TM	0.966	0.930	0.945	0.948	0.966	790	$\approx$ 16.37KB / $\approx$ 0.00KB	0.011 sec.
RWTM	0.916	0.949	0.931	0.931	0.913	66310	$\approx$ 65.53KB / $\approx$ 0.00KB	0.910 sec.
IWTM	0.946	0.966	0.954	0.953	0.941	120	$\approx$ 16.37KB / $\approx$ 0.00KB	0.015 sec.

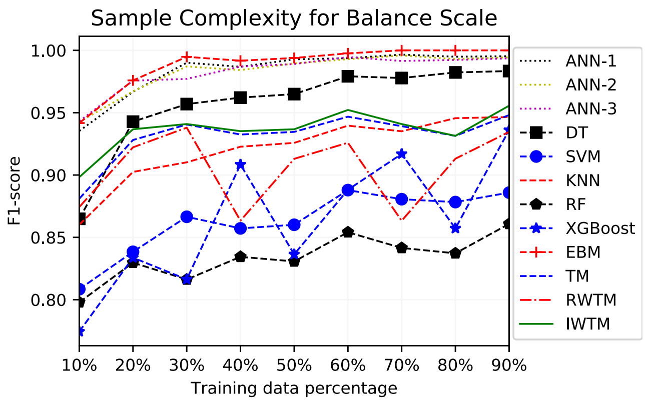

FIGURE 8. Sample complexity analysis for the Balance Scale dataset.

best F1-Score with 500 clauses, using an average of 4406 literals overall. In contrast, IWTM reaches its best F1-Score using only 120 literals, distributed among 100 clauses. Again, IWTM uses significantly fewer literals than TM and RWTM.

The average number of literals used for varying number of clauses is plotted in FIGURE 7. IWTM uses the least number of literals, with the difference increasing with number of clauses.

TABLE 11 summarises the performance also of the other machine learning techniques we contrast against. Here, EBM obtains the highest F1-Score and Accuracy. Out of the three TMs, IWTM achieves the highest F1-Score and Accuracy, using similar or less training memory. The training time required by IWTM is close to the training time of TM, and roughly 60 times less compared to RWTM. According to the sample complexity analysis in FIGURE 8, IWTM reaches an F1-Score of 0.90 with merely 10% of the training data, and approaches 0.95 from 20%.

C. BREAST CANCER

The Breast Cancer dataset[3] covers recurrence of breast cancer, and consists of nine features: Age, Menopause, Tumor Size, Inv Nodes, Node Caps, Deg Malig, Side (left or right), the Position of the Breast, and Irradiation Status. The dataset contains 286 patients (201 with non-recurrence and 85 with recurrence). However, some of the patient samples miss some of the feature values. These samples are removed from the dataset in the present experiment.

[3]Available from https://archive.ics.uci.edu/ml/datasets/Breast+Cancer

TABLE 12. Performance of TM on Breast Cancer dataset.

m	2	10	100	500	2000	8000
Precision	0.518	0.485	0.295	0.101	0.058	0.054
Recall	0.583	0.380	0.416	0.205	0.200	0.250
F1-Score	0.531	0.389	0.283	0.089	0.090	0.088
Accuracy	0.703	0.737	0.644	0.633	0.649	0.581
Specificity	0.742	0.864	0.731	0.800	0.800	0.750
No. of Lit.	21	73	70	407	1637	6674

TABLE 13. Performance of RWTM on Breast Cancer dataset.

m	2	10	100	500	2000	8000
Precision	0.461	0.645	0.781	0.768	0.530	0.250
Recall	0.576	0.389	0.306	0.220	0.099	0.027
F1-Score	0.493	0.472	0.423	0.334	0.162	0.047
Accuracy	0.706	0.767	0.778	0.770	0.740	0.722
Specificity	0.758	0.913	0.961	0.975	0.992	1.000
No. of Lit.	4	59	232	445	1532	6608

TABLE 14. Performance of IWTM on Breast Cancer dataset.

m	2	10	100	500	2000	8000
Precision	0.396	0.555	0.182	0.242	0.289	0.189
Recall	0.766	0.502	0.055	0.144	0.255	0.284
F1-Score	0.511	0.510	0.070	0.104	0.172	0.163
Accuracy	0.604	0.731	0.727	0.705	0.643	0.644
Specificity	0.545	0.824	0.979	0.895	0.815	0.783
No. of Lit.	2	9	25	84	355	1306

The accuracy and number of literals included for TM, RWTM, and IWTM are respectively summarized in TABLE 12, TABLE 13, and TABLE 14. In contrast to the previous two datasets, the F1-Score peaks at $m = 2$, and then drops with increasing m. For $m = 2$, the average number of literals used by TM, RWTM, and IWTM are 21, 4, and 2, respectively. As seen in FIGURE 9, IWTM requires the least amount of literals overall.

The performance of the other machine learning techniques is similar in terms of F1-Score, with DT, RF, SVM, XGBoost, and EBM providing the worst performance as summarized in TABLE 15. The best F1-Score is obtained by TM while IWTM provides the second-best. Yet, the moderate increase of F1-Score from 0.511 to 0.531 for TM comes at the cost of 19 extra literals. The three TMs also uses the least memory, requiring negligible memory both during training and

TABLE 15. Performance comparison for Breast Cancer dataset.

	Precision	Recall	F1	Accuracy	Specificity	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.489	0.455	0.458	0.719	0.822	-	≈ 1001.97KB / ≈ 35.74KB	0.249 sec.
ANN-2	0.430	0.398	0.403	0.683	0.792	-	≈ 3498.47KB / ≈ 608.71KB	0.248 sec.
ANN-3	0.469	0.406	0.422	0.685	0.808	-	≈ 38645.07KB / ≈ 1837.76KB	0.288 sec.
DT	0.415	0.222	0.276	0.706	0.915	-	≈ 102.39KB / ≈ 0.00KB	0.005 sec.
SVM	0.428	0.364	0.384	0.678	0.805	-	≈ 241.66KB / ≈ 299.00KB	0.001 sec.
KNN	0.535	0.423	0.458	0.755	0.871	-	≈ 249.85KB / ≈ 61.43KB	0.001 sec.
RF	0.718	0.267	0.370	0.747	0.947	-	≈ 139.26KB / ≈ 0.00KB	0.020 sec.
XGBoost	0.428	0.344	0.367	0.719	0.857	-	≈ 1327.10KB / ≈ 0.00KB	0.026 sec.
EBM	0.713	0.281	0.389	0.745	0.944	-	≈ 1724.41KB / ≈ 0.00KB	6. 007 sec.
TM	0.518	0.583	0.531	0.703	0.742	21	≈ 0.00KB / ≈ 0.00KB	0.001 sec.
RWTM	0.461	0.576	0.493	0.706	0.758	4	≈ 0.00KB / ≈ 0.00KB	0.001 sec.
IWTM	0.396	0.766	0.511	0.604	0.545	2	≈ 0.00KB / ≈ 0.00KB	0.001 sec.

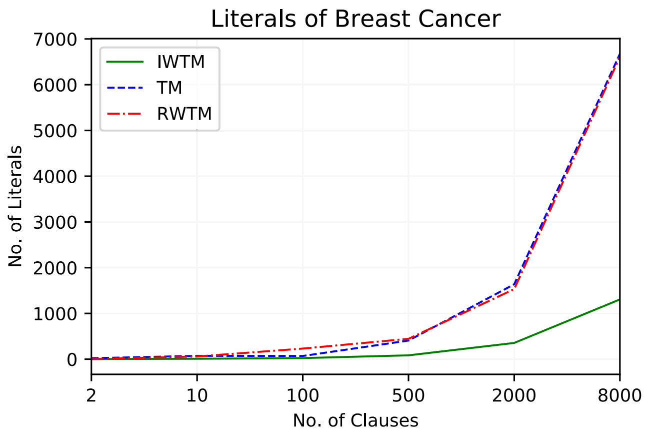

FIGURE 9. The number of literals included in different TM setups to work with Breast Cancer dataset.

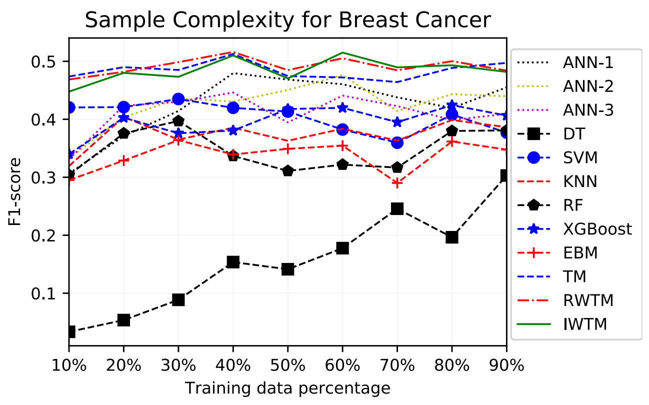

FIGURE 10. Sample complexity analysis for the Breast Cancer dataset.

testing. Training time per epoch for the TM approaches is also small, amounting to 0.001 seconds, which is the lowest of all the algorithms. The TMs also maintain better F1-Scores across all training data sizes in comparison with the other techniques, as seen from the sample complexity analysis in FIGURE 10.

D. LIVER DISORDERS

The Liver Disorders dataset[4] was created by BUPA Medical Research and Development Ltd. (hereafter "BMRDL") during the 1980s as part of a larger health-screening database. The dataset consists of 7 attributes, namely Mean Corpuscular Volume, Alkaline Phosphotase, Alamine Aminotransferase, Aspartate Aminotransferase, Gamma-Glutamyl Transpeptidase, Number of Half-Pint Equivalents of Alcoholic Beverages (drunk per day), and Selector (used to split data into training and testing sets). However, McDermott and Forsyth [39] claim that many researchers have used the dataset incorrectly, considering the Selector attribute as class label. Based on the recommendation of McDermott and Forsythof, we here instead use Number of Half-Pint Equivalents of Alcoholic Beverages as the dependent variable, binarized using the threshold $\geq$ 3. The Selector attribute is discarded. The remaining attributes represent the results of various blood tests, and we use them as features.

TABLE 17, TABLE 18, and TABLE 19 summarizes the performance of TM, RWTM, and IWTM, respectively. As seen, all of the TM F1-Scores peak at $m = 2$. TM uses an average of 27 literals, RWTM uses 29, while IWTM uses 9. FIGURE 11 plots how the number of literals increases with number of clauses, again confirming that IWTM uses fewer literals overall.

Considering the other machine learning techniques (TABLE 16), RF produces the highest F1-Score 0.729, obtained with the smallest memory usage for both training and testing. However, this performance is comparable to the DT F1-Score of 0.728, spending negligible testing memory. Of the three TM approaches, RWTM obtains the highest F1-Score - the fourth highest among all of the techniques. All three TMs use insignificant memory during both training and testing, requiring the same amount of training time per epoch.

[4]Available from https://archive.ics.uci.edu/ml/datasets/Liver+Disorders.

TABLE 16. Performance comparison for Liver Disorders dataset.

	Precision	Recall	F1	Accuracy	Specificity	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.651	0.702	0.671	0.612	0.490	-	≈ 985.13KB / ≈ 18.53KB	0.305 sec.
ANN-2	0.648	0.664	0.652	0.594	0.505	-	≈ 3689.39KB / ≈ 598.26KB	0.305 sec.
ANN-3	0.650	0.670	0.656	0.602	0.508	-	≈ 38365.46KB / ≈ 1758.23KB	0.356 sec.
DT	0.591	0.957	0.728	0.596	0.135	-	≈ 49.15KB / ≈ 0.00KB	0.025 sec.
SVM	0.630	0.624	0.622	0.571	0.500	-	≈ 1597.43KB / ≈ 0.00KB	0.005 sec.
KNN	0.629	0.651	0.638	0.566	0.440	-	≈ 0.00KB / ≈ 434.17KB	0.001 sec.
RF	0.618	0.901	0.729	0.607	0.192	-	≈ 0.00KB / ≈ 0.00KB	0.017 sec.
XGBoost	0.641	0.677	0.656	0.635	0.568	-	≈ 3219.45KB / ≈ 0.00KB	0.081 sec.
EBM	0.641	0.804	0.710	0.629	0.406	-	≈ 7790.59KB / ≈ 0.00KB	10.772 sec.
TM	0.566	0.799	0.648	0.533	0.204	27	≈ 0.00KB / ≈ 0.00KB	0.003 sec.
RWTM	0.607	0.811	0.688	0.581	0.226	29	≈ 0.00KB / ≈ 0.00KB	0.003 sec.
IWTM	0.570	0.869	0.680	0.576	0.140	9	≈ 0.00KB / ≈ 0.00KB	0.003 sec.

TABLE 17. Performance of TM on Liver Disorders dataset.

m	2	10	100	500	2000	8000
Precision	0.566	0.540	0.506	0.455	0.442	0.417
Recall	0.799	0.597	0.508	0.595	0.500	0.593
F1-Score	0.648	0.550	0.389	0.450	0.375	0.437
Accuracy	0.533	0.540	0.516	0.522	0.526	0.504
Specificity	0.204	0.436	0.497	0.395	0.500	0.396
No. of Lit.	27	51	117	509	2315	8771

TABLE 18. Performance of RWTM on Liver Disorders dataset.

m	2	10	100	500	2000	8000
Precision	0.607	0.625	0.645	0.632	0.613	0.608
Recall	0.811	0.691	0.616	0.621	0.596	0.546
F1-Score	0.688	0.653	0.627	0.620	0.599	0.571
Accuracy	0.581	0.580	0.566	0.573	0.556	0.532
Specificity	0.226	0.417	0.492	0.508	0.501	0.513
No. of Lit.	29	68	238	995	3877	10584

TABLE 19. Performance of IWTM on Liver Disorders dataset.

m	2	10	100	500	2000	8000
Precision	0.570	0.500	0.366	0.233	0.285	0.258
Recall	0.869	0.708	0.459	0.398	0.500	0.450
F1-Score	0.680	0.575	0.376	0.293	0.362	0.327
Accuracy	0.576	0.557	0.504	0.470	0.510	0.479
Specificity	0.140	0.339	0.553	0.602	0.500	0.550
No. of Lit.	9	13	26	116	353	1014

With 10% of the training data available, IWTM obtains the highest F1-Score among all the techniques – a score of 0.725. After that, the score fluctuates with increasing training data size, as depicted in FIGURE 12.

E. HEART DISEASE

The Heart Disease dataset[5] concerns prediction of heart disease. To this end, 13 features are available, selected among 75. Out of the 13 features, 6 are real-valued, 3 are binary, 3 are nominal, and one is ordered.

TABLE 21, TABLE 22, and TABLE 23 summarize the performance of TM, RWTM, and IWTM on the Heart Disease dataset. For the TM, the best F1-Score occurs with $m = 10$, achieved by using 346 literals on average. The RWTM F1-Score peaks at $m = 2000$ with 18 528 literals. IWTM peaks

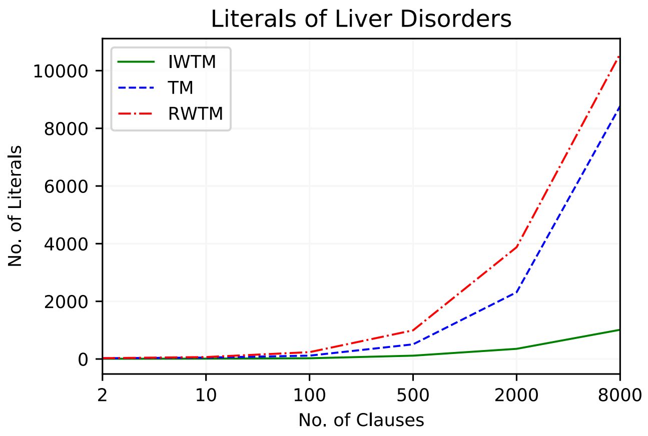

FIGURE 11. The number of literals included in different TM setups to work with the Liver Disorders dataset.

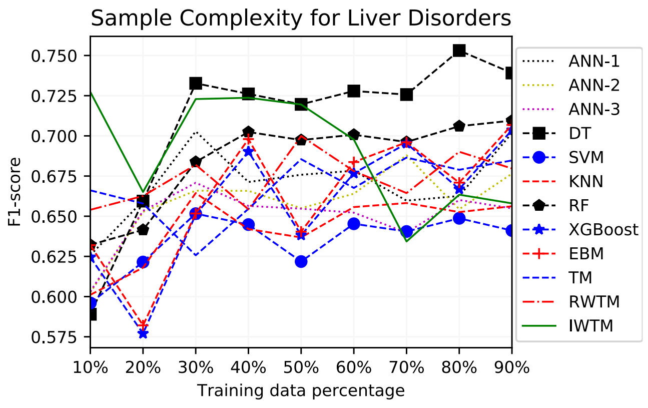

FIGURE 12. Sample complexity analysis for the Liver Disorders dataset.

at $m = 10$, with slightly lower F1-Score, however, employing only 226 literals on average.

Considering the number of literals used with increasing number of clauses (FIGURE 13), TM and IWTM behave similarly, while RWTM requires significantly more literals.

Out of the considered machine learning models, EBM obtains the best F1-Score, while RWTM, IWTM, and ANN-2 follow closely behind (TABLE 20). However, EBM needs the highest training time and uses the second largest training

[5]Available from https://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29.

TABLE 20. Performance comparison for Heart Disease dataset.

	Precision	Recall	F1	Accuracy	Specificity	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.764	0.724	0.738	0.772	0.811	-	≈ 973.64KB / ≈ 16.46KB	0.297 sec.
ANN-2	0.755	0.736	0.742	0.769	0.791	-	≈ 3659.59KB / ≈ 578.11KB	0.266 sec.
ANN-3	0.661	0.662	0.650	0.734	0.784	-	≈ 33952.49KB / ≈ 1513.41KB	0.308 sec.
DT	0.827	0.664	0.729	0.781	0.884	-	≈ 0.00KB / ≈ 266.23KB	0.016 sec.
SVM	0.693	0.674	0.679	0.710	0.740	-	≈ 1363.96KB / ≈ 262.14KB	0.004 sec.
KNN	0.682	0.615	0.641	0.714	0.791	-	≈ 0.00KB / ≈ 319.48KB	0.001 sec.
RF	0.810	0.648	0.713	0.774	0.879	-	≈ 413.69KB / ≈ 0.00KB	0.017 sec.
XGBoost	0.712	0.696	0.701	0.788	0.863	-	≈ 3694.58KB / ≈ 0.00KB	0.057 sec.
EBM	0.827	0.747	0.783	0.824	0.885	-	≈ 4763.64KB / ≈ 0.00KB	11.657 sec.
TM	0.607	0.815	0.687	0.672	0.566	346	≈ 0.00KB / ≈ 0.00KB	0.014 sec.
RWTM	0.735	0.788	0.757	0.790	0.784	18528	≈ 237.56KB / ≈ 0.00KB	1.559 sec.
IWTM	0.694	0.801	0.740	0.743	0.692	226	≈ 0.00KB / ≈ 0.00KB	0.010 sec.

TABLE 21. Performance of TM on Heart Disease dataset.

m	2	10	100	500	2000	8000
Precision	0.547	0.607	0.835	0.507	0.351	0.360
Recall	0.938	0.815	0.626	0.408	0.646	0.486
F1-Score	0.682	0.687	0.665	0.383	0.446	0.392
Accuracy	0.593	0.672	0.749	0.619	0.533	0.584
Specificity	0.306	0.566	0.848	0.803	0.460	0.665
No. of Lit.	118	346	810	1425	11399	52071

TABLE 22. Performance of RWTM on Heart Disease dataset.

m	2	10	100	500	2000	8000
Precision	0.567	0.610	0.672	0.697	0.735	0.752
Recall	0.908	0.855	0.806	0.820	0.788	0.769
F1-Score	0.695	0.707	0.725	0.748	0.757	0.754
Accuracy	0.640	0.693	0.740	0.779	0.790	0.781
Specificity	0.417	0.571	0.691	0.752	0.784	0.792
No. of Lit.	160	458	1031	16512	18528	58432

TABLE 23. Performance of IWTM on Heart Disease dataset.

m	2	10	100	500	2000	8000
Precision	0.550	0.694	0.797	0.701	0.725	0.848
Recall	0.929	0.801	0.723	0.696	0.661	0.523
F1-Score	0.687	0.740	0.716	0.619	0.609	0.580
Accuracy	0.625	0.743	0.773	0.678	0.669	0.696
Specificity	0.371	0.692	0.797	0.646	0.669	0.823
No. of Lit.	81	226	609	1171	7459	40894

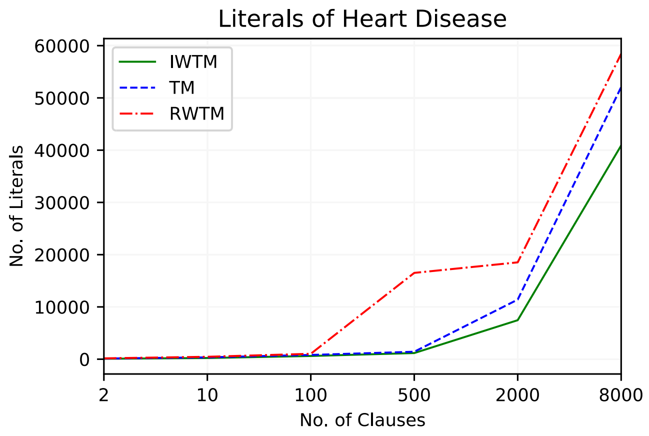

FIGURE 13. The number of literals included in different TM setups to work with Heart Disease dataset.

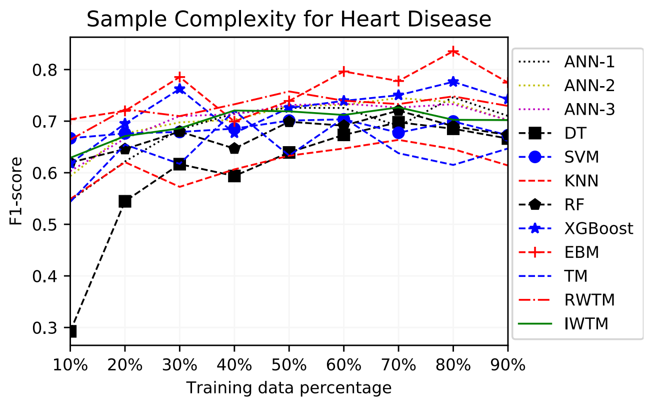

FIGURE 14. Sample complexity analysis for the Heart Disease dataset.

memory, while all three TMs use negligible memory during both training and testing. Apart from DT, all of the machine learning models surpass an F1-Score of 0.5 using only 10% of the training data, as shown in FIGURE 14. From 30% onward, the F1-Score of all the models behaves similarly, with EBM being superior after 60%.

F. SUMMARY OF EMPIRICAL EVALUATION

To compare overall performance of the various techniques, we calculate average F1-Score across the datasets. Further to evaluate overall interpretability of TM, RWTM and IWTM, we also report average number of literals used, overall.

In all brevity, the average F1-Score of ANN-1, ANN-2, ANN-3, DT, SVM, KNN, RF, TM, XGBoost, EBM, RWTM, and IWTM are 0.770, 0.757, 0.744, 0.742, 0.713, 0.737, 0.724 0.728, 0.775, 0.762, 0.774, and 0.777, respectively. Out of all the considered models, IWTM obtains the best average F1-Score, which is 0.777. Also notice that increasing ANN model complexity (from ANN-1 to ANN-3) reduces overall F1-Score, which can potentially be explained by the small size of the datasets.

The F1-Score of RWTM is also competitive, however, it requires much more literals than IWTM. Indeed, the average number of literals employed are 961 for TM, 17 670 for RWTM, and 147 for IWTM. That is, IWTM uses 6.5 times fewer literals than TM, and 120 times fewer literals than RWTM.

The average combined memory requirement (training + testing) by TM, RWTM, and IWTM are 3.27 KB, 79.46 KB, and 3.27, respectively. The combined memory usage of

TABLE 24. Performance (in AUC) comparison against recent state-of-the-art machine learning models.

Model	Fraud Detection	COMPAS	CA-58
Logistic Regression	0.975	0.730	-
DT	0.956	0.723	-
NAMs	0.980	0.741	-
EBM	0.976	0.740	-
XGBoost	0.981	0.742	-
DNNs	0.978	0.735	-
LightBoost	-	-	≈ 0.760†
CatBoost	-	-	≈ 0.760†
StructureBoost	-	-	≈ 0.764†
IWTM	0.990	0.732	0.772

†These results were extracted from graphs in [34]

IWTM is significantly less compared to the other models – ANN-1: ≈ 305 times, ANN-2: ≈ 1 278 times, ANN-3: ≈ 11 096 times, DT: ≈ 34 times, SVM: ≈ 255 times, KNN: ≈ 106 times, RF: ≈ 45 times, XGBoost: ≈ 877 times, EBM: ≈ 1226 times, and RWTM: ≈ 24 times.

G. COMPARISON AGAINST RECENT STATE-OF-THE-ART MACHINE LEARNING MODELS

In this section, we compare IWTM accuracy with reported results on recent state-of-the-art machine learning models. First, we perform experiments on Fraud Detection and COMPAS: Risk Prediction in Criminal Justice datasets to study the performance of IWTM in comparison with Neural Additive Models [11]. A Neural Additive Model is a novel member of so-called general adaptive models. In Neural Additive Models, the significance of each input feature towards the output is learned by a dedicated neural network. During the training phase, the complete set of neural networks are jointly trained to learn complex interactions between inputs and outputs.

To compare the performance against StructureBoost [34], we use the CA weather dataset [40]. For simplicity, we use only the CA-58 subset of the dataset in this study. StructureBoost is based on gradient boosting and is capable of exploiting the structure of categorical variables. StructureBoost outperforms established models such as CatBoost and LightBoost on multiple classification tasks [34].

Since the performance of both of the above techniques has been measured in terms of Area under the ROC Curve (AUC), we here use a soft TM output layer [41] to calculate AUC. The performance characteristics are summarized in TABLE 24.

TABLE 24 shows that on Fraud Detection, IWTM outperforms NAMs and all the other techniques mentioned in [11]. On the COMPAS dataset, IWTM exhibits competitive performance compared to NAMs, EBM, XGBoost, and DNNs. IWTM shows, however, superior performance compared to Logistic Regression and DT on COMPAS. The performance of IWTM on CA-20 is better in comparison to StructureBoost, LightBoost, and CatBoost models, reported in [34].

IV. CONCLUSION

In this paper, we proposed a novel Tsetlin Machine (TM) having integer weights attached to clauses, to address the accuracy-interpretability challenge in machine learning. In our proposed TM (denoted IWTM), the weights are learnt using the stochastic searching on the line (SSL) automaton. The weights attached to the clauses help the TM to the represent sub-patterns in a more compact way. Since integer weights can turn off unimportant clauses by setting their weight to 0, this allows the TM to create a classifier with fewer number of literals compared to the vanilla TM and the Real-Value Weighted TM (RWTM). We have provided empirical evidence by generating rules for several datasets. In conclusion, the IWTM obtains on par or better accuracy compared to the vanilla TM and the RWTM while using respectively 6.5 and 125 times fewer literals. Furthermore, in terms of average F1-Score, the proposed IWTM also outperforms several state-of-the-art machine learning algorithms.

In our future work, we intend to investigate more advanced SSL schemes, such as Continuous Point Location with Adaptive Tertiary Search (CPL-ATS) [42] and Random Walk-based Triple Level Learning Algorithm (RWTLA) [43], including their impact on interpretability.

REFERENCES

[1] C. Molnar, *Interpretable Machine Learning*. Morrisville, NC, USA: Lulu, 2019.

[2] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, “Deep learning for healthcare: Review, opportunities and challenges,” *Briefings Bioinf.*, vol. 19, no. 6, pp. 1236–1246, Nov. 2018.

[3] B. Baesens, C. Mues, M. De Backer, J. Vanthienen, and R. Setiono, “Building intelligent credit scoring systems using decision tables,” in *Enterprise Information Systems V*. Dordrecht, The Netherlands: Springer, 2004, pp. 131–137.

[4] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, “An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models,” *Decis. Support Syst.*, vol. 51, no. 1, pp. 141–154, Apr. 2011.

[5] R. Bellazzi and B. Zupan, “Predictive data mining in clinical medicine: Current issues and guidelines,” *Int. J. Med. Informat.*, vol. 77, no. 2, pp. 81–97, Feb. 2008.

[6] S. Mani, W. R. Shankle, and M. J. Pazzani, “Acceptance of rules generated by machine learning among medical experts,” *Methods Inf. Med.*, vol. 40, no. 5, pp. 380–385, 2001.

[7] A. A. Freitas, D. C. Wieser, and R. Apweiler, “On the importance of comprehensible classification models for protein function prediction,” *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 7, no. 1, pp. 172–182, Jan. 2010.

[8] D. Szafron, P. Lu, R. Greiner, D. S. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell, A. Fyshe, and D. Meeuwis, “Proteome analyst: Custom predictions with explanations in a Web-based tool for high-throughput proteome annotations,” *Nucleic Acids Res.*, vol. 32, no. 2, pp. W365–W371, Jul. 2004.

[9] E. Lima, C. Mues, and B. Baesens, “Domain knowledge integration in data mining using decision tables: Case studies in churn prediction,” *J. Oper. Res. Soc.*, vol. 60, no. 8, pp. 1096–1106, Aug. 2009.

[10] W. Verbeke, D. Martens, C. Mues, and B. Baesens, “Building comprehensible customer churn prediction models with advanced rule induction techniques,” *Expert Syst. Appl.*, vol. 38, no. 3, pp. 2354–2364, Mar. 2011.

[11] R. Agarwal, N. Frosst, X. Zhang, R. Caruana, and G. E. Hinton, “Neural additive models: Interpretable machine learning with neural nets,” 2020, *arXiv:2004.13912*. [Online]. Available: http://arxiv.org/abs/2004.13912

[12] M. T. Ribeiro, S. Singh, and C. Guestrin, “Why should I trust you?' Explaining the predictions of any classifier,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144.

[13] C. Rudin, “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,” *Nature Mach. Intell.*, vol. 1, no. 5, pp. 206–215, May 2019.

[14] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, Jun. 1993.

[15] T. McCormick, C. Rudin, and D. Madigan, "A hierarchical model for association rule mining of sequential events: An approach to automated medical symptom prediction," MIT Sloan Res. Paper, Jan. 2011. [Online]. Available: https://ssrn.com/abstract=1736062

[16] V. Feldman, "Hardness of approximate two-level logic minimization and PAC learning with membership queries," *J. Comput. Syst. Sci.*, vol. 75, no. 1, pp. 13–26, Jan. 2009.

[17] L. G. Valiant, "A theory of the learnable," *Commun. ACM*, vol. 27, no. 11, pp. 1134–1142, Nov. 1984.

[18] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl, and P. MacNeille, "A Bayesian framework for learning rule sets for interpretable classification," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 2357–2393, 2017.

[19] J. R. Hauser, O. Toubia, T. Evgeniou, R. Befurt, and D. Dzyabura, "Disjunctions of conjunctions, cognitive simplicity, and consideration sets," *J. Marketing Res.*, vol. 47, no. 3, pp. 485–496, Jun. 2010.

[20] Y. Liang and G. Van den Broeck, "Learning logistic circuits," in *Proc. 33rd AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4277–4286.

[21] O.-C. Granmo, "The tsetlin machine—A game theoretic bandit driven approach to optimal pattern recognition with propositional logic," 2018, *arXiv:1804.01508*. [Online]. Available: http://arxiv.org/abs/1804.01508

[22] A. Phoulady, O.-C. Granmo, S. R. Gorji, and H. A. Phoulady, "The weighted tsetlin machine: Compressed representations with weighted clauses," 2019, *arXiv:1911.12607*. [Online]. Available: https://arxiv.org/abs/1911.12607

[23] G. T. Berge, O.-C. Granmo, T. O. Tveit, M. Goodwin, L. Jiao, and B. V. Matheussen, "Using the tsetlin machine to learn human-interpretable rules for high-accuracy text categorization with medical applications," *IEEE Access*, vol. 7, pp. 115134–115146, 2019.

[24] K. D. Abeyrathna, O.-C. Granmo, X. Zhang, L. Jiao, and M. Goodwin, "The regression Tsetlin machine: A novel approach to interpretable nonlinear regression," *Phil. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 378, no. 2164, Feb. 2020, Art. no. 20190165.

[25] S. R. Gorji, O.-C. Granmo, A. Phoulady, and M. Goodwin, "A Tsetlin machine with multigranular clauses," in *Proc. 39th Int. Conf. Innov. Techn. Appl. Artif. Intell. (SGAI)*, in Lecture Notes in Computer Science, vol. 11927. Cham, Switzerland: Springer, 2019, pp. 146–151.

[26] S. Gorji, O. C. Granmo, S. Glimsdal, J. Edwards, and M. Goodwin, "Increasing the inference and learning speed of Tsetlin machines with clause indexing," in *Proc. Int. Conf. Ind., Eng. Other Appl. Appl. Intell. Syst.* Cham, Switzerland: Springer, 2020, pp. 695–708.

[27] A. Wheeldon, R. Shafik, A. Yakovlev, J. Edwards, I. Haddadi, and O.-C. Granmo, "Tsetlin machine: A new paradigm for pervasive AI," in *Proc. SCONA Workshop Design, Automat. Test Eur. (DATE)*, 2020. [Online]. Available: https://www.date-conference.com/node/468

[28] M. L. Tsetlin, "On behaviour of finite automata in random medium," *Avtomatika I Telemekhanika*, vol. 22, no. 10, pp. 1345–1354, 1961.

[29] O.-C. Granmo, S. Glimsdal, L. Jiao, M. Goodwin, C. W. Omlin, and G. T. Berge, "The convolutional Tsetlin machine," 2019, *arXiv:1905.09688*. [Online]. Available: http://arxiv.org/abs/1905.09688

[30] K. D. Abeyrathna, O.-C. Granmo, X. Zhang, and M. Goodwin, "A scheme for continuous input to the Tsetlin machine with applications to forecasting disease outbreaks," in *Proc. Int. Conf. Ind., Eng. Other Appl. Appl. Intell. Syst.* Cham, Switzerland: Springer, 2019, pp. 564–578.

[31] B. J. Oommen, "Stochastic searching on the line and its applications to parameter learning in nonlinear optimization," *IEEE Trans. Syst. Man, Cybern. B, Cybern.*, vol. 27, no. 4, pp. 733–739, Aug. 1997.

[32] H. Nori, S. Jenkins, P. Koch, and R. Caruana, "InterpretML: A unified framework for machine learning interpretability," 2019, *arXiv:1909.09223*. [Online]. Available: http://arxiv.org/abs/1909.09223

[33] Y. Lou, R. Caruana, and J. Gehrke, "Intelligible models for classification and regression," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2012, pp. 150–158.

[34] B. Lucena, "StructureBoost: Efficient gradient boosting for structured categorical variables," 2020, *arXiv:2007.04446*. [Online]. Available: http://arxiv.org/abs/2007.04446

[35] K. S. Narendra and M. A. Thathachar, *Learning Automata: An Introduction*. Chelmsford, MA, USA: Courier Corporation, 2012.

[36] I. Chivers and J. Sleightholme, "An introduction to algorithms and the big O notation," in *Introduction to Programming With Fortran*. Cham, Switzerland: Springer, 2015, pp. 359–364.

[37] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794.

[38] M.-J. Kim and I. Han, "The discovery of experts' decision rules from qualitative bankruptcy data using genetic algorithms," *Expert Syst. Appl.*, vol. 25, no. 4, pp. 637–646, Nov. 2003.

[39] J. McDermott and R. S. Forsyth, "Diagnosing a disorder in a classification benchmark," *Pattern Recognit. Lett.*, vol. 73, pp. 41–43, Apr. 2016.

[40] B. Lucena, "Exploiting categorical structure using tree-based methods," 2020, *arXiv:2004.07383*. [Online]. Available: http://arxiv.org/abs/2004.07383

[41] K. D. Abeyrathna, O.-C. Granmo, and M. Goodwin, "On obtaining classification confidence, ranked predictions and AUC with Tsetlin machines," in *Proc. IEEE Symp. Series Comput. Intell. (ISSC)*, 2020.

[42] B. J. Oommen and G. Raghunath, "Automata learning and intelligent tertiary searching for stochastic point location," *IEEE Trans. Syst. Man, Cybern. B, Cybern.*, vol. 28, no. 6, pp. 947–954, Dec. 1998.

[43] W. Jiang, D.-S. Huang, and S. Li, "Random walk-based solution to triple level stochastic point location problem," *IEEE Trans. Cybern.*, vol. 46, no. 6, pp. 1438–1451, Jun. 2016.

K. DARSHANA ABEYRATHNA received the B.Sc. degree in mechatronics engineering from AIT University, Thailand, in 2015, and the M.Sc. degree from the Big Data Research Group, Thammasat University, Thailand, in 2017. He is currently a Ph.D. Research Fellow with the University of Agder. He is also with CAIR, working on a project which develops a global grid that facilitates real-time compilation, and management and analysis of spatio-temporal data. His research interests include artificial neural networks, data mining, optimization, and operations research.

OLE-CHRISTOFFER GRANMO received the master's and Ph.D. degrees from the University of Oslo, Norway, in 1999 and 2004, respectively. He is currently a Professor with the University of Agder and the Founding Director of the Centre for Artificial Intelligence Research (CAIR). He has authored in excess of 140 refereed articles within machine learning, encompassing learning automata, bandit algorithms, Tsetlin machines, Bayesian reasoning, reinforcement learning, and computational linguistics. Apart from his academic endeavors, he co-founded the company Anzyz Technologies AS. He is also a Co-Founder of the Norwegian Artificial Intelligence Consortium (NORA).

MORTEN GOODWIN received the master's degree from the University of Agder, Norway, in 2005, and the Ph.D. degree from the Department of Computer Science, Aalborg University, Denmark, in 2011, where he applied machine learning algorithms on eGovernment indicators which are difficult to measure automatically. He is currently an Associate Professor with the University of Agder. His research interests include data-mining, optimization, machine learning, swarm intelligence, deep learning, and adaptive learning. He has written more than 40 peer-reviewed scientific publications in this area and supervised more than 60 student projects at the master's and Ph.D. level.

• • •