Novel Tsetlin Machine Mechanisms for Logic-based Regression and Classification with Support for Continuous Input, Clause Weighting, Confidence Assessment, Deterministic Learning, and Convolution

K. Darshana Abeyrathna

Novel Tsetlin Machine Mechanisms for Logic-based Regression and Classification with Support for Continuous Input, Clause Weighting, Confidence Assessment, Deterministic Learning, and Convolution

Doctoral Dissertation for the Degree *Philosophiae Doctor (PhD)* at the Faculty of Engineering and Science, Specialisation in Information and Communication Technology

University of Agder
Faculty of Engineering and Science
2022

Doctoral Dissertations at the University of Agder 359
ISSN: 1504-9272
ISBN: 978-82-8427-068-5

Printed by 07 Media
Oslo

Preface

To explain my philosophy of *productivity in life*, imagine a plot of a person's achievements in the life with his/her average effort vs the time. Another two series can be drawn which are the imaginary achievements of the same person when the effort is worst (below the line of the average effort) and the best (above the line of the average effort). Once a person dies, if we plot the true achievements of his/her life and get the area under the curve, we measure something related to productivity of that person in life. But it's unfair to compare it with someone else as we all born with different abilities and raised in different family setups (e.g., poor and rich if we just discretize it). However, someone who born with worse abilities can still be more productive in life than another person who born with better abilities if the first person gives his/her best effort while the second person gives his/her worst. Further, it's fair to compare your true productivity with your imaginary best and the worst. So you are the one who decide where you stand in these scales.

During the years between 1997 and roughly 2002, the school books were washed away for the every single rain, no matter how mild or strong it was. The school teachers started not believing it when I came that as the reason to not to complete the homework. From those days to today, all the obstacles were overcome and now I can't imagine I am going to start writing my Ph.D. dissertation. According to my measure of productivity in life above, I proudly believe that I have given my best effort to get the maximum productivity in my life so far.

I believe that the life experience have helped me to battle against the Ph.D. stress. However, it is nothing without the help, guidance, and motivations of my Ph.D. supervisor Professor Ole-Christoffer Granmo and the co-supervisor Professor Morten Goodwin.

Together with them, I carried out my Ph.D. research at the Department of Information Communication Technology, Faculty of Engineering and Science, University of Agder, Norway. The research goal was to develop and expand the Tsetlin Machine so that it is applicable to solve problems from a wide range of application domains.

I believe that the outcome of my Ph.D research would serve the Artificial Intelligence and Machine Learning communities to conduct further, enhanced research in this area of research in the future.

Acknowledgments

This dissertation is based on research activities carried out at the University of Agder during the period January 2018 to September 2021. Here, I want to express my gratitude to all who supported me during my PhD studies.

Foremost, I would like to express my sincere gratitude to my supervisors Professor Ole-Christoffer Granmo and Professor Morten Goodwin for their invaluable guidance and support throughout my PhD research activities. Special thank to my main supervisor Professor Ole-Christoffer Granmo for providing research directions, facilities, motivation and guidance to complete the research. I'm extremely grateful to co-supervisor Professor Morten Goodwin for his great support, motivation and guidance for improving technical content and publications of the research by allocating plentiful time from his schedule.

I would also like to mention the co-authors - Associate Professor Lei Jiao, researcher Xuan Zhang, Associate Professor Harsha Sandaruwan, Professor Vladimir A. Oleshchuk, researcher Sasanka N. Ranasinghe, researcher Sinziana Rasca, researhcer Karin Markvica, Professor Rishad Shafik, Professor Alex Yakovlev, researcher Adrian Wheeldon, and researcher Jie Lei in my publications. Your support, contribution, and guidance helped me to publish a number of research articles during my PhD studies.

I am thankful to the Head of the Department, Folke Haugland and PhD coordinator, Emma E. Horneman for providing me the necessary facilities and pleasant working environment for research activities. I should also thank the rest of the administration staff at the ICT department of the university and the fellow researchers at the Center for Artificial Intelligence Research (CAIR) for all the help I received from you.

The fellow Srilankans helped me, supported me, and encouraged me from the first day in Norway until I finish this adventures journey. Thank you Dr. Indika Anuradha Mendis, Associate Professor Harsha Sandaruwa, Sasanka Ranasinghe, Madhawa Jayathilaka, Kalpani Mendis, Dr. Jagath Sri Lal, Dr. Thilina Weerasinghe, Sandun Konara, Wajira Senanayaka, and Pabasara.

Last but not least, I would like to thank my family in Sri Lanka - parents Kuruge. Abeyrathna and Jayanthi Edirisinghe and sister Nirmani Abeyrathna for their continued love, encouragement, and understanding.

K. Darshana Abeyrathna
Grimstad, Norway
December 2021

Abstract

The recently introduced Tsetlin Machine (TM) is built on Tsetlin Automata (TAs), developed by M. L. Tsetlin in the early 1960s. Relying only on a single integer as memory, a TA learns the optimal action on-line, embedded in a random environment. Used in teams, TAs have previously solved a number of machine learning and stochastic optimization problems. The TM uses TAs to learn patterns for pattern recognition. The TAs then represent literals – input features and their negations. The literals, in turn, form conjunctive clauses in propositional logic, as decided by the TAs. The final TM output can be seen as a disjunction of all the specified clauses. In this manner, the pattern composition and learning procedure of the TM is transparent, facilitating human interpretation. In addition, the TM has an inherent computational advantage. That is, the inputs and outputs of the TM can naturally be represented as bits, and recognition and learning is performed by manipulating those bits. The operation of the TM thus demands relatively small computational resources and supports hardware-near and parallel computation e.g. on GPUs.

The TM has provided competitive results in comparison with traditional machine learning techniques on various classification tasks. However, the original TM architecture is limited by its propositional foundation in several ways. Firstly, while many applications require continuous input and output, TMs process propositional input with propositional operators, producing propositional output. As such, TMs are not natively suited for dealing with continuous values, for instance in regression problems. Further, it is unclear how to go from propositional output to measuring output confidence, which can be essential in decision-making. Finally, learning requires extensive random number generation to compensate for lack of continuous representations. Random number generation increases computation time and makes hardware implementation more costly energy-wise.

In this thesis, we address the above challenges by designing, analysing and evaluating several new mechanisms for TM learning. The purpose of the mechanisms is to deal with continuous input and output. To this end, we propose several strategies for dealing with continuous features, including adaptive thresholding with Stochastic Searching on the Line automata. We further propose an architecture for mapping clauses to a real-valued output for regression problems, also supporting regression with convolution on 2D input. We also introduce pattern weighing by attaching integer weights to the TM clauses, facilitating more compact and interpretable representation of models by reducing pattern duplication. Further, we assess classification confidence instead of merely classifying a sample into a class. Finally, we introduce deterministic learning automata that replaces random number generation with multi-step state-changes, providing a foundation for more energy efficient TM micro-controller implementations.

Overall, the above mechanisms have improved the performance of the TM in terms of accuracy, interpretability, inference speed, energy usage, and memory consumption. Apart from introducing the mechanisms, the thesis provides comparisons with selected state-of-the-art machine learning models, reporting comparable or better performance.

Summary

Menneskehjernen (og dyrehjernen) lærer blant annet fra syns-, smaks-, og hørselsinntrykk, gjennom erfaring. Maskinlæring (ML) er denne læringens kunstige tvilling. ML er en gren innenfor kunstig intelligens som fokuserer på læring fra samhandling med et miljø, hvor erfaringene oversettes til numeriske verdier.

Det finnes mange ML-algoritmer. Ulike algoritmer gir ulike fordeler og ulemper. Det er miljøtypen og brukerkravene som bestemmer hvilken ML-algoritme som er best egnet.

Tsetlin-maskinen (TM) er en slik ML-algoritme, som er relativt ny innenfor ML-forskningen. TMen har unike egenskaper og fordeler. Den kan imidlertid ikke brukes når læringsmiljøet representerer erfaring som kontinuerlige verdier. TMen er binær og kan derfor ikke enkelt håndtere kontinuerlig innverdier eller produsere kontinuerlige utverdier. Videre blir den binære mønsterrepresentasjonen og læringen mindre økonomisk. Dermed reduseres tolkbarheten og beslutningshastighet, mens energiforbruk og minneavtrykket øker. En binær utverdi forteller heller ikke noe om hvor sikker TMen er når brukeren ønsker å estimere treffsikkerheten bak en utverdi.

I dette Ph.D.-arbeidet adresserer vi TM-begrensningene beskrevet ovenfor ved å endre TM-arkitekturen og læringsprosedyren på fire måter:

- Vi foreslår hvordan kontinuerlige verdier kan håndteres ved å (1) representere kontinuerlige innverdier statisk ved hjelp av binære terskler, (2) innføre adaptiv læring av kontinuerlige verdier direkte i TM-arkitekturen, og (3) omforme TM-arkitekturen slik at den støtter regresjon: en Regresjons-TM (RTM) som produserer kontinuerlige utverdier.

- Vi har laget en ny TM-arkitektur som knytter vekter til de logiske mønstrene TMen lærer og viktigheten av disse i dataene. Vektene gjør den innlærte modellen mer kompakt, øker tolkbarheten og resonneringshastigheten, samtidig som minneforbruket reduseres.

- Inkludering og ekskludering av inn-verdier i de logiske mønstrene bestemmes av såkalte Tsetlin-automater (TAer). Vi reduserer energiforbruket i læringsfasen ved å introdusere en ny type læringsautomat som vi kaller en flertrinns variabel-struktur endelig tilstandsautomat. Denne automaten gjør deterministiske tilstandshopp mens TAene i den originale TM er avhengig av energikostbar generering av tilfeldige tall.

- Til slutt foreslår vi en metode for å erstatte den harde beslutningsfunksjonen som brukes av TMen for å produsere utverdier. Beslutningsfunksjonen erstattes av en

logistisk funksjon. Denne tilnærmingen gjør det mulig å måle treffsikkerheten til TMen når den predikerer uverdier.

Ved å overvinne begrensningene over ser vi at TMen yter sammenlignbart eller bedre enn en rekke populære maskinlæringsalgoritmer.

Publications

The following list of publications consists of fourteen published papers and two submitted papers. All the papers in the list contribute to the goals of the thesis, with the research carried out during the Ph.D. study. Only the first eleven papers, the most central ones, are included in the dissertation.

Papers Contributing to the Dissertation.

Paper 1: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. "A Scheme for Continuous Input to the Tsetlin Machine with Applications to Forecasting Disease Outbreaks." In International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pp. 564-578. Springer, Cham, 2019.

Paper 2: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. "The Regression Tsetlin Machine: A Novel Approach to Interpretable Nonlinear Regression." Philosophical Transactions of the Royal Society A 378, no. 2164, 2020.

Paper 3: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. "Adaptive Continuous Feature Binarization for Tsetlin Machines Applied to Forecasting Dengue Incidences in the Philippines." In 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2084-2092. IEEE, 2020.

Paper 4: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, and Morten Goodwin. "Integer Weighted Regression Tsetlin Machines." In International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pp. 686-694. Springer, Cham, 2020.

Paper 5: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, and Morten Goodwin. "On Obtaining Classification Confidence, Ranked Predictions and AUC with Tsetlin Machines." In 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 662-669. IEEE, 2020.

Paper 6: Abeyrathna, K. Darshana, Harsha S. Gardiyawasam Pussewalage, Sasanka N. Ranasinghe, Vladimir A. Oleshchuk, and Ole-Christoffer Granmo. "Intrusion Detection with Interpretable Rules Generated Using the Tsetlin Machine." In 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1121-1130. IEEE, 2020.

Paper 7: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, and Morten Goodwin. "Extending the Tsetlin Machine with Integer-Weighted Clauses for Increased Interpretability." IEEE Access 9: 8233-8248, 2021.

Paper 8: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, and Morten Goodwin. "A multi-step finite-state automaton for arbitrarily deterministic Tsetlin Machine learning." Accepted at Expert Systems, September, 2021.

Paper 9: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, and Morten Goodwin. "Adaptive Sparse Representation of Continuous Input for Tsetlin Machines Based on Stochastic Searching on the Line." Electronics 10, no. 17: 2107, 2021.

Paper 10: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, and Morten Goodwin. "Convolutional Regression Tsetlin Machine: An Interpretable Approach to Convolutional Regression." Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, September, 2021.

Paper 11: Abeyrathna, K. Darshana, Sinziana Rasca, Karin Markvica, and Ole-Christoffer Granmo "Public Transport Passenger Count Forecasting in Pandemic Scenarios Using Regression Tsetlin Machine. Case Study of Agder, Norway." In Smart Transportation Systems (pp. 27-37). Springer, Singapore, 2021.

Other Related Publications Not Included in the Dissertation.

Paper 12: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, Lei Jiao, and Morten Goodwin. "The regression Tsetlin machine: a Tsetlin machine for continuous output problems." In EPIA Conference on Artificial Intelligence, pp. 268-280. Springer, Cham, 2019.

Paper 13: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, Rishad Shafik, Alex Yakovlev, Adrian Wheeldon, Jie Lei, and Morten Goodwin. "A Novel Multi-step Finite-State Automaton for Arbitrarily Deterministic Tsetlin Machine Learning." In International Conference on Innovative Techniques and Applications of Artificial Intelligence, pp. 108-122. Springer, Cham, 2020.

Paper 14: Jiao, Lei, Xuan Zhang, Ole-Christoffer Granmo, and Abeyrathna, K. Darshana."On the Convergence of Tsetlin Machines for the XOR Operator." Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, Jan, 2021.

Paper 15: Abeyrathna, K. Darshana, Ole-Christoffer Granmo, and Morten Goodwin. "Convolutional Regression Tsetlin Machine: An Interpretable Approach to Convolutional Regression." In 2021 6th International Conference on Machine Learning Technologies, pp. 65-73. 2021.

Paper 16: Abeyrathna, K. Darshana, Bimal Bhattarai, Morten Goodwin, Saeed Rahimi Gorji, Ole-Christoffer Granmo, Lei Jiao, Rupsa Saha, and Rohan K. Yadav. "Massively parallel and asynchronous Tsetlin Machine architecture supporting almost constant-time scaling." In International Conference on Machine Learning, pp. 10-20. ICML, 2021.

Contents

List of Figures

List of Tables

List of Abbreviations

ANNs multi-layered Artificial Neural Networks.

ASIC Application-Specific Integrated Circuit.

AUC area under the characteristic curve.

C-RTM Convolutional Regression Tsetlin Machine.

CNF Conjunctive Normal Form.

CNNs Convolutional Neural Networks.

CTM Convolutional Tsetlin Machine.

DNF Disjunctive Normal Form.

DTs Decision Trees.

EBMs Explainable Boosting Machines.

IWTM Integer weighted Tsetlin Machine.

LAs Learning Automata.

LR Logistic Regression.

MAE mean absolute error.

MPs Multilayer Perceptrons.

NB Naıve Bayes.

NE Nash equilibria.

PAC Probably Approximately Correct.

RF Random Forest.

RTM Regression Tsetlin Machine.

RTs Regression Trees.

SSL stochastic searching on the line automata.

SVMs Support Vector Machines.

SVR Support Vector Regression.

TAs Tsetlin Automata.

TM Tsetlin Machine.

WTM Weighted Tsetlin Machine.

XGBoost Gradient Boosted Trees.

Part I

Chapters

Chapter 1

Introduction

This provides necessary background knowledge required to conduct the entire research. The chapter starts by giving an introduction to Learning Automata (LAs), the foundation of the Tsetlin Machine (TM), in Section 1.1. The principles of so-called Tsetlin Automata (TAs) and other selected LAs are further explained in detail. We then summarize their contributions to the pattern recognition field. There after selected related propositional logic based pattern recognition algorithms are discussed. On this foundation, we establish the Research Questions, Motivations and Main Objectives of the research. Research approach and the dissertation outline is presented towards the end of the chapter.

1.1 Learning Automata

The origins of LAs can be traced back to the work of M. L. Tsetlin in the early 1960s [1]. The objective of an LA is to learn the optimal action through trial and error in a stochastic environment. Various types of LAs are available depending on the nature of the application [2]. The family of *two-action finite-state* LAs caught significant attention and effort in the early stages of LA research due to their simplicity, reliability, flexibility, and applicability [3]. A two-action finite-state LA interacts with its environment iteratively. In each iteration, the action that it performs next is decided by its present state (the memory). The environment, in turn, randomly produces a reward or a penalty according to an unknown probability distribution, responding to the action selected by the LA. If the LA receives a reward, it reinforces the selected action. A penalty, on the other hand, weakens it. Across as few iterations as possible, the goal is to identify the action with the highest reward probability [3].

The transitions between states can be deterministic or stochastic. Deterministic transitions occur with probability 1.0, while stochastic transitions are randomly performed based on a preset probability. If the transition probabilities are changing, we have a *variable structure* automaton, otherwise, we end up with a *fixed structure.*

1.1.1 The Tsetlin Automaton (TA)

TA is one particular kind of finite-state, fixed-structure and deterministic LA. Figure 1.1 shows a two-action TA with $2N$ states. As illustrated in the figure, the next action

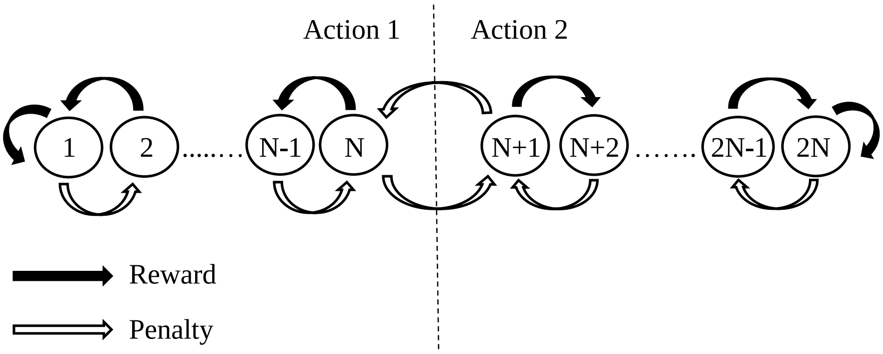


Figure 1.1: Transition graph of a two-action Tsetlin Automaton.

that the TA performs is decided by its present state. States numbered from 1 to N map to Action 1, while states from $N+1$ to 2N map to Action 2. The TA interacts with its environment iteratively. In each iteration, the TA performs the action associated with its current state. This, in turn, randomly triggers a reward or a penalty from the environment, according to an unknown probability distribution. If the TA receives a reward, it reinforces the action performed by moving to a "deeper" state, meaning that the TA state is one step closer to one of the ends (left or right side). If the action results in a penalty, the TA moves one step towards the middle state to weaken the performed action, ultimately jumping to the middle state of the other action. In this manner, with a sufficient number of states, a TA converges to performing the action with the highest probability of producing rewards – the optimal action – with probability arbitrarily close to unity, merely by interacting with the environment (as long as the reward probability of the optimal action is larger than 0.5) [3].

1.1.2 Other Variants of Learning Automata

While the TA changes its state in single steps, the deterministic Krinsky Automaton introduces multi-step state transitions when it is rewarded [3]. The purpose of the multi-step state transitions is to reinforce an action more strongly when it is rewarded, and more weakly when penalized. The Krinsky Automaton behaves as a TA when the response from the environment is a penalty. However, when it is a reward, any state from 2 to N transitions to state 1, and any state from $N+1$ to $2N-1$ transitions to state $2N$. In effect, N consecutive penalties are needed to offset a single reward.

Another variant of LA is the Krylov Automaton. A Krylov Automaton makes both deterministic and stochastic single-step transitions [3]. The state transitions of the Krylov Automaton is identical to those of a TA for rewards. However, when it receives a penalty, it performs the corresponding TA state change randomly, with probability 0.5.

The $G_{2N,2}$ automata scheme [1] is somewhat similar to TA (TA is typically expressed as $L_{2N,2}$ where the subscripts $2N$ and 2 in both $G_{2N,2}$ and $L_{2N,2}$ indicate the number of memory states and the number of actions, respectively). However, in the $G_{2N,2}$ scheme, when the current state is N and the environment feedback is a penalty, the state moves from N to $2N$ (instead of to $N+1$ as in TA). Similarly, if the current state is $N+1$ and the environment feedback is a penalty, the state moves from $N+1$ to 1 (instead of to N

as in TA). Hence, in the $G_{2N,2}$ scheme, at least N penalties are needed to change action again after a switch.

All the automata discussed so far, and also the Ponomarev Automaton [4] and the Cover-Hellman Automaton [5], are fixed-structure automata whose state transition probabilities are fixed and do not vary with the training iterations. On the contrary, there exist another kind of automata, called variable-structure automata, whose transition probabilities are updated over the training iterations. The linear reward-penalty scheme [6], referred to as the L_{R-P} scheme, is a popular variable-structure automaton. The reward parameter λ_R ($0 \leq \lambda_R < 1$) and the penalty parameter λ_P ($0 \leq \lambda_P < 1$) update the probability of selecting each action after every training iteration. In other words, if the feedback is a reward, the probability of selecting the other action is multiplied by λ_R. Hence, the update reduces the probability of selecting the other action. Similarly, if the feedback is a penalty, the probability of selecting the same action is multiplied by λ_P. Accordingly, the update reduces the probability of selecting the same action. The linear reward-inaction (L_{R-I}) scheme [7] behaves the same as L_{R-P} for rewards, but its penalty parameter λ_P is 0.

1.2 Learning Automata in Pattern Recognition

Learning Automata have also been used for pattern recognition over more than four decades due to their ability of learning the optimal action when operating in unknown stochastic environments [8]. One example of such approaches is the automata-based pattern classifier introduced by Zahiri [9]. This classifier is able to approximate the decision hyper-planes in the feature space without knowing the priori probabilities and class distributions. The performance of the proposed classifier tested on datasets containing nonlinear and overlapping class boundaries is on par or better than the selected set of typical machine learning classifiers. Further, Sastry et al., have also proposed a team of continuous-action-set learning automata based algorithm for noise-tolerant learning of linear hyper-plane classifiers [10]. Sastry and Thathachar [11] also conducted a study on learning automata-based algorithms which are capable of learning optimal discriminant functions for pattern classification under both pattern noise and classification noise. They explore how different algorithms have different advantages and disadvantages when they are utilized. Further, when it comes to optimization, they also find similarities between the automata approaches they study and so-called genetic algorithms. The robust discriminant function built based on LA in [12] and the LA-based pattern classification algorithm in [13] for multi-class classification problem are other relatively recent approaches from the LA field.

In addition to classification tasks, LA have also been applied in image segmentation [14, 15], motion estimation [16], person identification (by eye gaze detection) [17], object detection [18], optimization [19], classification as deterministic optimization [20], online tracking of event patterns [21], as well as solving the string taxonomy problem [22].

The performance of LA might be further enhanced by hybridizing them with other algorithms. Motieghader et al., propose such an algorithm for gene selection in cancer classification [23]. The proposed algorithm is an integration of Genetic Algorithm and Learning Automata (GALA) and is able to perform comparably against other cancer classification algorithms. In [20], LAs are used to mix constituent prediction models. The LAs then assign and update a suitable weight per model. The whole system as one unit is used to predict resource usage in cloud computing environments. A similar approach is presented in [24]. In the latter work, the influence of each individual method on the final ensemble output is controlled by the LA. A coefficient attached to each single model is updated online during the training phase using LA-based learning. Finally, the LA-based classifier by Barto and Anandan [25] combines stochastic LA with a pattern-classification algorithm based on stochastic approximation. They claim that the capabilities of LAs allow the classifier to learn from less informative training data. However, a closer examination of the above LA-based approaches shows that they address relatively simple pattern recognition tasks with some exception.

Although the original TA shown in Figure 1.1 is rather simple, it forms the basis for more advanced algorithm designs.These extensions include the Hierarchy of Twofold Resource Allocation Automata (H-TRAA) for resource allocation [26] and the stochastic searching on the line automata (SSL) algorithm by Oommen et al. [27]. Furthermore, teams of TAs have been used to create a distributed coordination system [28], to solve the graph coloring problem [29], and to forecast dengue outbreaks in the Philippines [30].

The TM is a recent addition to the field of TA, addressing complex pattern recognition. The TM uses the TA as a building block to solve complex pattern recognition tasks. The TM operates as follows. Firstly, propositional formulas in disjunctive normal form are used to represent patterns. The TM is thus a general function approximator. The propositional formulas are learned through training on labelled data by employing a collective of TAs, organized in a game. The payoff matrix of the game has been designed so that the Nash equilibria (NE) correspond to the optimal configurations of the TM. As a result, the architecture of the TM is relatively simple, facilitating transparency and interpretation of both learning and classification. Additionally, the TM is designed for bit-wise operation. That is, it takes bits as input and uses fast bit-wise operators for both learning and classification. This gives the TM an inherent computational advantage.

1.3 Selected Related Approaches not Based on Learning Automata

While this thesis seeks to advance the field of LA in rule-based pattern recognition, other areas of research also address rule-based pattern recognition. We here give an overview of selected work for the sake of completeness.

Propositional logic is a well-explored framework for knowledge-based pattern classification. Here, data is represented in propositional form based on experimentation or on expert knowledge. There also exists approaches where classifiers are built directly from a truth table, for instance expressed in Disjunctive Normal Form (DNF) or Conjunctive Normal Form (CNF). One such example can be found in [31]. Here, continuous features in clinical and genomic data are converted to truth values by thresholding. Experts' knowledge is used to find the effective thresholds for each continuous feature. Then, DNF is used to represent the patterns among them. In this manner, logical functions are created for predicting recurrence and non-recurrence of liver cancer, without relying on training data.

However, in machine learning applications, when large number of input features and data samples are available, building classifiers through a truth table is not necessarily feasible. In such cases, data-driven techniques to learn the propositional formulas from data are more commonly used. This family of techniques is typically called rule induction techniques. Learning propositional formulas to represent patterns in data has a long history [32]. Feldman investigated the hardness of learning DNF [33], Klivans used Polynomial Threshold Functions to build logical expressions [34], while Feldman leveraged Fourier analysis [35]. Furthermore, so-called Probably Approximately Correct (PAC) learning has provided fundamental insight into machine learning, as well as providing a framework for learning formulas in DNF [36]. An integer programming approach is proposed in [37] to learn CNF formulas, providing promising results based on a Bayesian method. In addition to the above techniques, association rule mining models have also been extensively applied in [38, 39] to predict sequential events using a set of rules. Furthermore, recent approaches combine Bayesian reasoning with propositional formulas in DNF for robust learning of formulas from data [32]. Overviews and comparisons of other rule induction algorithms can be found in [40, 41, 42, 43], while example applications are provided in [44, 45, 46, 47].

One special case of rule induction is building the classifiers when features are unknown and must be learnt from data. As an example, Santa Cruz et al. use Boolean expressions to capture visual primitives for visual recognition (e.g., **IF** the image contains "(**NOT** large wingspan) **AND** hooked beak" **THEN** a gull bird is predicted). The important visual concepts (e.g., wingspan, beak) were not explicitly specified [48]. Instead, they used neural network models both to extract features and to learn the classifier.

The TM can be seen as a rule induction method. The class patterns are recognized by clauses, with a clause being a conjunction of selected relevant input features. As mentioned earlier, the team of TA of each clause decides the composition of the clause — which features should be included and which features should be excluded. At the end of training, the reasoning behind the classification of future samples can be expressed as human-interpretable IF-THEN rules (e.g., **IF** white **AND** wheels **AND** (**NOT** wings) **THEN** a car).

1.4 Research Questions, Motivations and Main Objectives

Despite its simplicity, the TM has provided competitive results in comparison with traditional machine learning techniques, including Multilayer Perceptrons (MPs), Support Vector Machines (SVMs), Logistic Regression (LR), and Naıve Bayes (NB), in well-known benchmarks such as hand written digits classification (MNIST), Iris data classification, and classification of Noisy XOR data with non-informative features.

However, the original TM architecture is limited by its propositional foundation in several ways. Firstly, while many applications require continuous input and output, TMs process propositional input with propositional operators, producing propositional output. As such, TMs are not inherently suited for dealing with continuous values, for instance in regression problems. Further, it is unclear how to go from propositional output to measuring output confidence, which can be essential in decision-making. Finally, learning requires extensive random number generation to compensate for lack of continuous representations. Random number generation increases computation time and makes hardware implementation more costly energy-wise. The above challenges motivate the following research questions and objectives:

Research Question 1: How can the TM process continuous input features?

Motivation: The original TM operates inherently only on propositional features. These features and their negations are directly fed into the clauses without any further modification. However, continuous features are arguably more common than propositional ones in real-world machine learning applications. Hence, to broaden the application areas of TMs, it is important to find methods that allow the TM to operate on continuous features.

Objective: Our first objective is thus to find effective ways of turning continuous features into propositional ones by means of pre-processing the data. Then we intend to modify the learning mechanisms of the TM to allow it to operate directly on continuous features, without any pre-processing.

Research Question 2: How can the TM predict continuous output?

Motivation: The TM has been designed for classification, not for producing continuous output. However, many machine learning problems require regression.

Objective: Hence, our objective is to recast the TM architecture for regression rather than classification. This requires also discovering an appropriate learning procedure that minimizes the regression error instead of the classification error.

Research Question 3: How can the TM pattern duplication be reduced by means of clause weighing, to improve memory usage and interpretability?

Motivation: Although TMs are capable of achieving competitive performance, they often require a large number of clauses to do so, which impedes interpretability and increases memory consumption. To overcome this issue, there is a need to represent patterns more compactly. While Phoulady et al. introduced continuous clause weights [49], these require multiplication for updating and an additional hyper-parameter.

Objective: Building upon the Weighted Tsetlin Machine (WTM) by Phoulady et al. [49], we intend to introduce weights to TM regression. While the WTM uses real-valued weights for classification, we instead intend to use *integer* weights. In addition to the computational benefits, we also argue that the integer weighted clauses are more interpretable than the real-valued ones because they can be seen as multiple copies of the same clause. Finally, our aim is to avoid additional hyper-parameters, which WTM relies on. This objective also entails developing an accompanying weight learning mechanism using increment/decrement operations rather than multiplication.

Research Question 4: How can one measure TM classification confidence, rank its predictions, and thus compute area under the characteristic curve (AUC)?

Motivation: The TM decisions are binary (hard), making them less suitable for applications that require trading off precision against recall. Indeed, Huang et al. [50] and Bradley [51] stress the importance of receiver operating characteristic (ROC) curves and area under ROC (AUC) for assessing machine learning techniques, as they capture the relationship between sensitivity and specificity. Further, getting access to the confidence of classifications can be important in many applications. One example is direct marketing, where it may be advantageous to approach the most likely buyers first, given limited time and resources. Accordingly, the ranking of predictions based on confidence, opens up for additional applications than mere classification [52].

Objective: Our next objective is thus to modify the output layer of the TM so that it is able to measure prediction confidence. We further intend to investigate how our proposed confidence measure can be used to calculate AUC and rank predictions.

Research Question 5: How can the Convolutional Tsetlin Machine (CTM) be modified to predict continuous image properties rather than classifying images into distinct classes?

Motivation: Convolutional regression applications are ample in the machine learning field. A few of them are near infrared (NIR) calibration [53], Spectrum analysis [54], depth prediction in digital holography [55], and vehicle detection and counting in aerial images [56]. To natively handle such applications, the CTM architecture for classification needs to be replaced by an architecture that supports regression.

Objective: We aim to expand upon the second research question, so that it is also possible to support convolutional regression. This will entail discovering how we can combine sub-patterns in images to produce regression output.

Research Question 6: How can the TM learning be improved by reducing the energy-costly random number generation that stochastically guide the TA clause composition?

Motivation: TMs rely on energy-costly random number generation to stochastically guide a team of TAs to a NE of the TM game. This is a serious issue for TM Application-Specific Integrated Circuit (ASIC) implementations aimed at low energy on-chip learning. Hence, there is a need to find ways to reduce the need for random number generation during TM learning.

Objective: Our objective is thus to reduce the need for random number generation by designing a new kind of LA that replaces the stochastic reward probability-guided learning with deterministic multi-step state transitions.

Research Question 7: To what degree can the TM solve real-life problems with competitive accuracy, while maintaining interpretability?

Motivation: Being a new machine learning technique, it is important to investigate how well the TM can solve real-life problems.

Objective: We intend to explore two applications to this end, network intrusion detection and public transport passenger count forecasting.

1.5 Overall Research Approach

To achieve our research objectives, this thesis blends the three main paradigms of computing research: theory, abstraction/modelling, and design [57]. Theory concerns proving properties and relationships among the objects of study. Abstraction investigates such relationships and properties by modeling them. If the predictions of the models match empirical observations from experiments, the belief in the validity of the models is strengthened. Theory and abstraction inform design of systems, which are tested against system requirements in the engineering paradigm.

Our overall research approach is visualized in Figure 1.2. As seen in the figure, we base our modelling of TM learning mechanisms on probability theory, LAs, Markov chains, and game theory. The resulting models then provide the basis for design and implementation of prototypes that are explored empirically. This exploration, in turn, leads to new insight that informs further theoretical analysis and modelling.

We have evaluated the effectiveness of our new designs not only on artificial data produced through simulation, but also on well-established public benchmark datasets. Further, performance has been compared against state-of-the-art methods.

We have pursued the above iterative research approach throughout the thesis work, with an emphasis on design and empirical exploration. This process has allowed us to

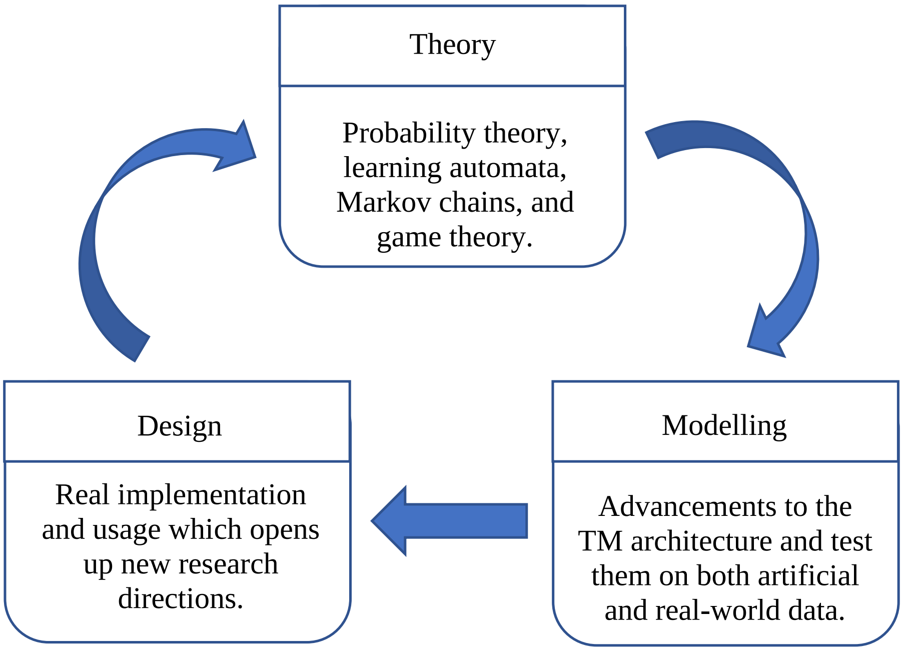

Theory
Probability theory, learning automata, Markov chains, and game theory.
Modelling
Advancements to the TM architecture and test them on both artificial and real-world data.
Design
Real implementation and usage which opens up new research directions.

Figure 1.2: The overall research approach, blending theory, modelling, and design.

discover new effective TM learning mechanisms, which has advanced state-of-the-art in TMs along several directions.

1.6 Dissertation Outline

The rest of the dissertation is organized as follows. In Chapter 2, we introduce the TM. We organize the introduction according to five conceptual TM layers, explaining the task of each layer. The chapter concludes with demonstrating the entire learning procedure on a toy dataset. We then present the contribution of this research in Chapter 3, organized according to the corresponding research questions. Chapter 4 provides a summary of the thesis publications, relating them to the thesis contributions presented in Chapter 3. Finally, we conclude and summarize the thesis findings in Chapter 5.

Chapter 2

Background on Tsetlin Machines

The TM uses the TAs as building blocks to solve complex pattern recognition tasks. The TM operates as follows. Firstly, propositional formulas in disjunctive normal form are used to represent patterns. The TM is thus a general function approximator. The propositional formulas are learned through training on labelled data by employing a collective of TAs organized in a game. The payoff matrix of the game has been designed so that the NE correspond to the optimal configurations of the TM. As a result, the architecture of the TM is relatively simple, facilitating transparency and interpretation of both learning and classification. Additionally, the TM is designed for bit-wise operation. That is, it takes bits as input and uses fast bit manipulation operators for both learning and classification. This gives the TM an inherent computational advantage.

This chapter introduces the architecture of the TM, which contains (conceptually) five layers. The function of each of these layers are discussed in more detail. Thereafter, we continue with presenting the learning procedure of the TM. At end of the chapter, we provide a walk-through of the learning steps by using a simple dataset, namely, XOR-data.

2.1 The Tsetlin Machine Architecture

As shown in Figure 2.1, TM can be decomposed conceptually into five layers for recognizing sub-patterns in the data and categorizing them into classes. In this section, we explain the function of each of these layers in the pattern recognition and learning phases of the TM.

2.1.1 Layer 1: The Input Layer

In the input layer, the TM receives a vector of o propositional variables: $\mathbf{X} = [x_1, x_2, \ldots, x_o]$, $x_k \in \{0, 1\}$. The objective of the TM is to classify this feature vector into one of the two classes, $y \in \{0, 1\}$. However, as shown in Figure 2.1, the input layer also includes the negations of the original features, $\neg x_k$, in the feature vector to capture more sophisticated patterns. Collectively, the elements in the augmented feature vector are called as literals: $\mathbf{L} = [x_1, x_2, \ldots, x_o, \neg x_1, \neg x_2, \ldots, \neg x_o] = [l_1, l_2, \ldots, l_{2o}]$.

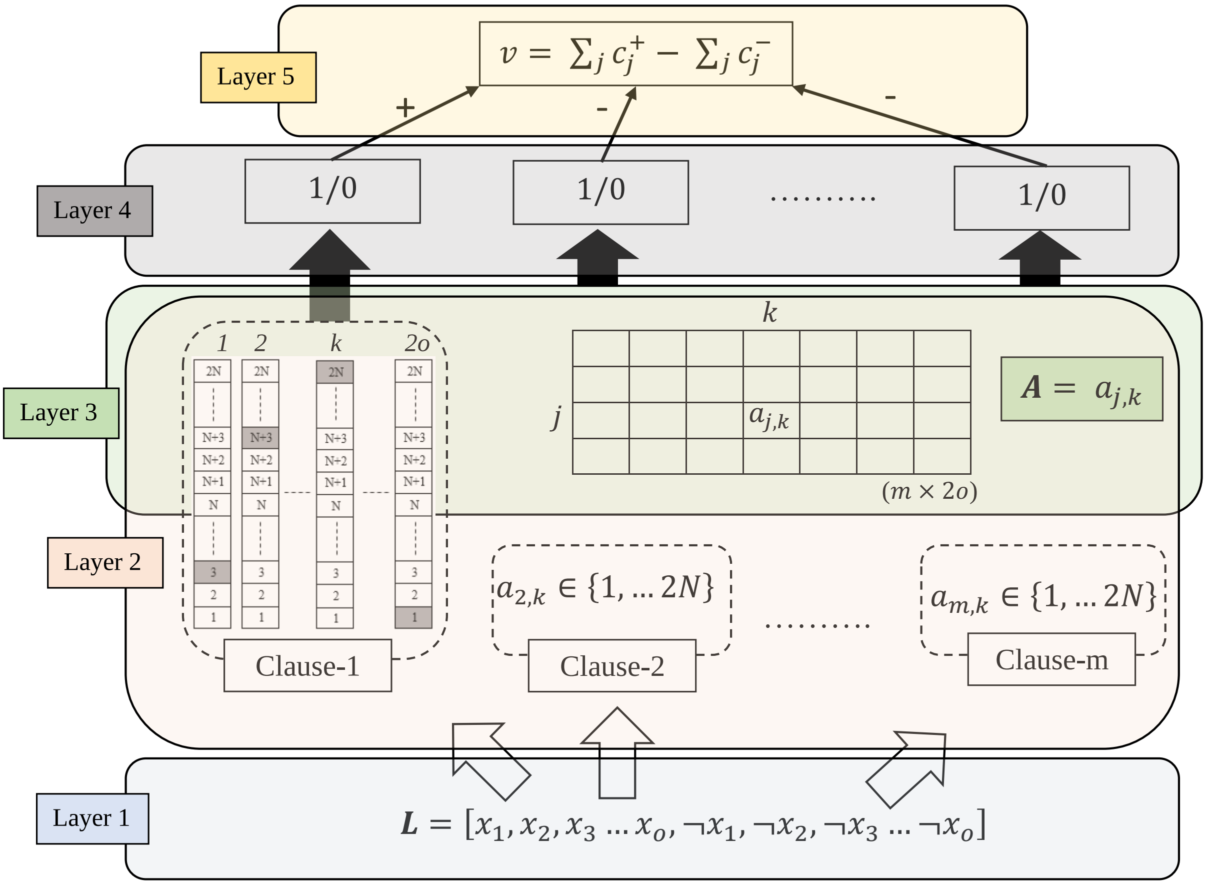

Layer 5
$v = \sum_j c_j^+ - \sum_j c_j^-$
+
-
-
Layer 4
1/0
1/0
....
1/0
Layer 3
1
2
k
2o
k
j
$a_{j,k}$
$(m \times 2o)$
$\boldsymbol{A} = a_{j,k}$
Layer 2
$a_{2,k} \in \{1, \dots 2N\}$
$a_{m,k} \in \{1, \dots 2N\}$
....
Clause-1
Clause-2
Clause-m
Layer 1
$\boldsymbol{L} = [x_1, x_2, x_3 \dots x_o, \neg x_1, \neg x_2, \neg x_3 \dots \neg x_o]$

Figure 2.1: The TM structure.

2.1.2 Layer 2: The Clause Construction

The sub-patterns associated with class 1 and class 0 are captured by m conjunctive clauses. The value m is set by the user. Accordingly, given that more complex problems might require larger m. All of the clauses receive the same augmented feature vector $\mathbf{L}$, assembled at the input layer. However, to perform the conjunction, only a fraction of the literals are selected, with the conjunction performed as follows:

$$c_j = \bigwedge_{k \in I_j} l_k. \tag{2.1}$$

Notice how the composition of a clause varies from another clause depending on the indexes of the included literals in the set $I_j \subseteq \{1, \dots, 2o\}$. For the special case of $I_j = \emptyset$, i.e., an empty clause, we have:

$$c_j = \begin{cases} 1 & \textbf{during } \text{learning} \\ 0 & \textbf{otherwise}. \end{cases} \tag{2.2}$$

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

2.1.3 Layer 3: Storing States of TAs of Clauses in the Memory

The TM employs two-action TAs as illustrated in Figure 1.1, to decide which literals are included in which clauses. Since we have $2 \times o$ number of literals in $\mathbf{L}$, the same number of TAs, one per literal k, is needed by a clause to decide the included literals in the clause. This is visualized in *Clause-1* of Figure 2.1. The states on the left hand side of the automaton (states from 1 to N) ask to *Exclude* the corresponding literal from the clause while the states on the right hand side of the automation (states from $N+1$ to $2N$) ask to *Include* the literal in the clause. The systematic storage of states of TAs 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, allows us to find the index set of the included literals in clause j, I_j as $I_j = \{k | a_{j,k} > N, \, 1 \leq k \leq 2o\}$.

2.1.4 Layer 4: Clause Output

Once the TA decisions are available, the clause output can be easily computed. Since the clauses are conjunctive, a single literal of value 0 is enough to turn the clause output to 0, if its corresponding TA has decided to *Include* it in the clause. For the ease of understanding, we introduce the set I_X^1, which contains the indexes of the literals of value 1. Then, the output of clause j can be expressed as:

$$c_j = \begin{cases} 1 & \text{if} \quad I_j \subseteq I_X^1, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

The clause outputs, computed as above, are now stored in the vector $\mathbf{C}$, i.e., $\mathbf{C} = (c_j) \in \{0, 1\}^m$.

2.1.5 Layer 5: Classification

The TM structure shown in Figure 2.1 classifies data into two classes. Hence, sub-patterns associated with each class have to be separately learned. For this purpose, the clauses are divided into two groups, where one group learns the sub-pattern of class 1 while the other learns the sub-patterns of class 0. For simplicity, clauses with odd index are assigned with positive polarity (c_j^+), and they are supposed to capture sub-patterns of output $y = 1$. Clauses with even index, on the other hand, are assigned with negative polarity (c_j^-) and they supposed to capture the sub-patterns of output $y = 0$.

The clauses which recognize sub-patterns output 1. This makes the classification process easier as we just need to sum the clause outputs of each class and assign the sample into the class which has the highest sum. A higher sum simply means that more sub-patterns have been identified from the designated class, and the input sample has a higher chance of being of that class. Hence, 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:

$$\hat{y} = \begin{cases} 1 & \text{if} \ \ v \ \geq \ 0 \\ 0 & \text{if} \ \ v \ < \ 0\,. \end{cases} \tag{2.4}$$

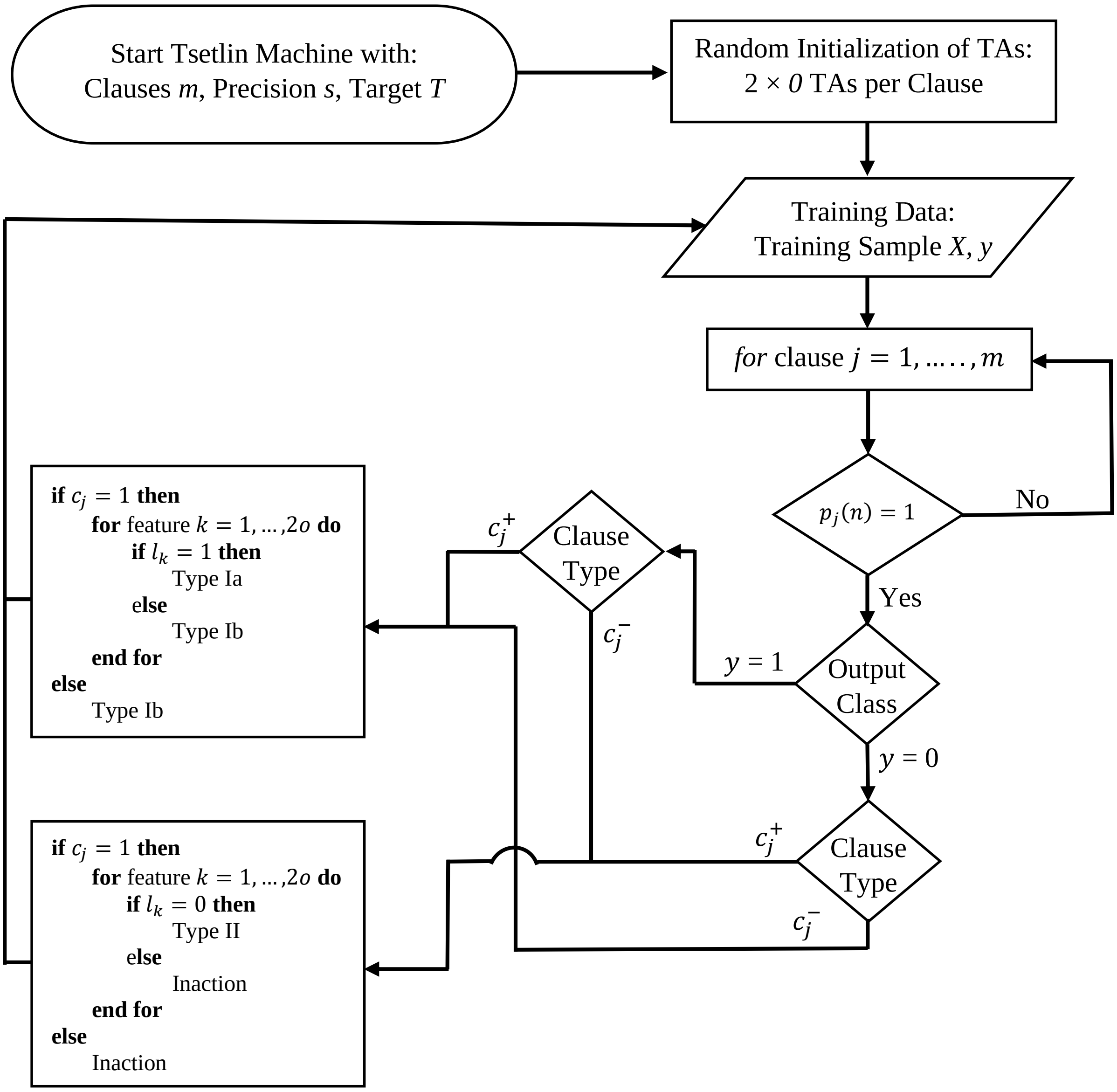

Start Tsetlin Machine with:
Clauses m, Precision s, Target T
Random Initialization of TAs:
2 × o TAs per Clause
Training Data:
Training Sample X, y
for clause $j = 1, \ldots\ldots, m$
$p_j(n) = 1$
No
Yes
Output Class
$y = 1$
$y = 0$
Clause Type
c_j^+
c_j^-
Clause Type
c_j^+
c_j^-
if $c_j = 1$ **then**
for feature $k = 1, \ldots, 2o$ **do**
if $l_k = 1$ **then**
Type Ia
else
Type Ib
end for
else
Type Ib
if $c_j = 1$ **then**
for feature $k = 1, \ldots, 2o$ **do**
if $l_k = 0$ **then**
Type II
else
Inaction
end for
else
Inaction

Figure 2.2: The complete learning process of the TM in a flowchart.

2.2 The Learning Procedure

As shown in Figure 2.2, a TM learns online, updating its internal parameters according to one training sample (X, y) at a time. As we discussed in Layer 4, a TA team decides the clause output, and, collectively, the output of all the clauses decide the TM output. Hence, to maximize the accuracy of the TM output, it is important to sensibly guide the individual TAs in the clauses. We achieve this with two kinds of reinforcement: Type I and Type II feedback. Type I and Type II feedback decide if the TAs in clauses receive a reward, a penalty, or inaction feedback, depending on the context of their actions. How the type of feedback is decided and how the TAs are updated according to the selected feedback type are discussed below in more detail.

2.2.1 Type I Feedback

Type I feedback has been designed to reinforce the true positive outputs of the clauses and to combat the false negative outputs. To reinforce the true positive output of a clause (clause output is 1 when it has to be 1), *Include* actions of TAs whose corresponding literal value is 1 are strengthened. Simultaneously, Type I Feedback strengthens the *Exclude* actions of TAs in the same clause whose corresponding literal value is 0. To combat false negative clause output (clause output is 0 when it has to be 1), we gradually erase the currently identified pattern. To do so, the *Exclude* actions of TAs, regardless of their corresponding literal values, are strengthened. We now sub-divide the Type I feedback into Type Ia and Ib, where Type Ia handles reinforcement of the *Include* action while Type Ib reinforces the *Exclude* action. Together, Type Ia and Type Ib feedback force clauses to output 1. Hence, clauses with positive polarity need Type I feedback when $y = 1$ and clauses with negative polarity need Type I feedback when $y = 0$. To diversify the clauses, they are targeted for Type I feedback stochastically as follows:

$$p_j = \begin{cases} 1 & \text{with probability } \frac{T-\max(-T,\min(T,v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

All clauses in each class should not learn the same sub-pattern, nor only a few. Hence, clauses should be smartly allocated among the sub-patterns. The user set target T in (2.5) does this while deciding the probability of receiving a Type I feedback. In effect, T number of clauses are available to learn each sub-pattern in each class. Higher T increases the robustness of learning by allocating more clauses to learn each sub-pattern. Now, T together with v decide the probability of clause j receiving Type I feedback and accordingly the decision p_j is made. The decisions for the complete set of clauses to receive Type I feedback are organized in the vector, $\mathbf{P} = (p_j) \in \{0,1\}^m$.

Once the clauses to receive Type I feedback are singled out as per (2.5), the probability of updating individual TAs in selected clauses is calculated using the user-set parameter s (s ≥ 1), separately for Type Ia and Type Ib. According to the above probabilities, the decision whether the k^{th} TA of the j^{th} clause is to receive Type Ia feedback, $r_{j,k}$, and Type Ib feedback, $q_{j,k}$, are stochastically computed as follows:

$$r_{j,k} = \begin{cases} 1 & \text{with probability } \frac{s-1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

$$q_{j,k} = \begin{cases} 1 & \text{with probability } \frac{1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

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 $I^{\text{Ib}} = \{(j,k)|(l_k = 0 \lor c_j = 0) \land p_{j,y} = 1 \land q_{j,k} = 1\}$.

The states of the identified TAs are now ready to be updated. Since Type Ia strengthens the *Include* action of TAs, the current state should move more towards the *Include* action direction. We denote this as $\oplus$ and here $\oplus$ adds 1 to the current state value of

Algorithm 1 The complete TM learning process

```
Input: Training data ($\mathbf{X}$, $y$), $m$, $T$, $s$
Initialize: Random initialization of TAs
Begin: $n^{th}$ training round
for $j = 1, ..., m$ do
    if $p_j = 1$ then                                              ▷ Eq. (2.5) and (2.8)
        if ($y = 1$ and $j$ is odd) or ($y = 0$ and $j$ is even) then
            if $c_j = 1$ then                                                ▷ Eq. (2.3)
                for feature $k = 1, ..., 2o$ do
                    if $l_k = 1$ then
                        Type Ia Feedback
                    else:
                        Type Ib Feedback
                    end if
                end for
            else:
                Type Ib Feedback
            end if
        else: ($y = 1$ and $j$ is even) or ($y = 0$ and $j$ is odd)
            if $c_j = 1$ then                                                ▷ Eq. (2.3)
                for feature $k = 1, ..., 2o$ do
                    if $l_k = 0$ then
                        Type II Feedback
                    else:
                        Inaction
                    end if
                end for
            else:
                Inaction
            end if
        end if
    end if
end for
```

the TA. The Type Ib feedback, on the other hand, moves the state of the selected TA towards *Exclude* action direction to strengthen the *Exclude* action of TAs. We denote this by $\ominus$ and here $\oplus$ subtracts 1 from the current state value of the TA. Accordingly, the states of TAs in $\mathbf{A}$ are updated as: $\mathbf{A} \leftarrow \left(\mathbf{A} \oplus I^{\text{Ia}}\right) \ominus I^{\text{Ib}}$.

2.2.2 Type II Feedback

Type II feedback has been designed to combat false positive clause output (clause output is 1 when it has to be 0). To turn this clause output from 1 to 0, a literal of value of 0 can simply be included in the clause. Clauses with positive polarity need Type II feedback when $y = 0$ and clauses with negative polarity need this when $y = 1$ since they do not want to vote for the opposite class. Again using the user-set target T, the decision for the j^{th} clause is made as follows:

$$p_j = \begin{cases} 1 & \text{with probability } \frac{T+\max(-T,\min(T,v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

The states of the TAs whose corresponding literal is of value 0 for the clauses select according to (2.8) are now moved in the *Include* action direction with probability one. Hence, the index set of this kind can be identified as: $I^{\text{II}} = \{(j,k)|l_k = 0 \land c_j = 1 \land p_j = 1\}$. Accordingly, the states of TAs in $\mathbf{A}$ are updated as: $\mathbf{A} \leftarrow \mathbf{A} \oplus I^{\text{II}}$.

The complete training procedure of the TM has also been summarized in Algorithm 1. When training has been completed, the final decisions of the TAs are recorded, and the resulting clauses can be deployed for operation.

2.3 Walk-through of Learning Using Type I and Type II Feedback to Learn Sub-Patterns

Dataset: In this section, we demonstrate how Type I and Type II feedback in the TM guide TA teams in clauses to learn sub-patterns in classes for classification. The practical example we consider here is training a traffic control system for a single lane road. We make it more specific by considering the case of guiding a car and a bus driving in opposite directions, as illustrated in Figure 2.3.

Figure 2.3: Illustration of the single lane crossing issue of a car and a bus.

Setting up the TM: Here, the bus and the car can drive past each other only if one of them wait on the side and let the other one drive. If both wait or both try to drive, they

Table 2.1: Driving control scenarios: Car and Bus columns say if they drive (*Yes*) or not (*No*) while Crossing column says if it's possible to pass each other (*Yes*) or not (*No*).

Scenario Index	Car	Bus	Crossing
1	Yes	Yes	No
2	Yes	No	Yes
3	No	Yes	Yes
4	No	No	No

do not succeed in passing each other. The possible scenarios of this driving control are summarized in Table 2.1[1].

Now we are going to see how a TM can be trained to learn this small practical example with the help of Type I and Type II feedback. Then, the learned rules can be used to build the traffic control system of such situations. As we can see in the above table, each class (crossing: Yes or No) has only two sub-patterns (e.g., one sub-pattern for class 'Yes' is (Car **AND NOT** Bus)). In this tutorial, we show how patterns of only one class can be learned. Learning of the patterns in the other class follows the same procedure.

Since each class has only two sub-patterns, they can be learned merely with two clauses. We select the case of learning sub-patterns of the 'Yes' class using two positive clauses: c_1^+ and c_2^+. They are both open to learn any sub-pattern in the 'Yes' class, but a

[1] 'Yes' and 'No' can be coded as 1 and 0 for the TM.

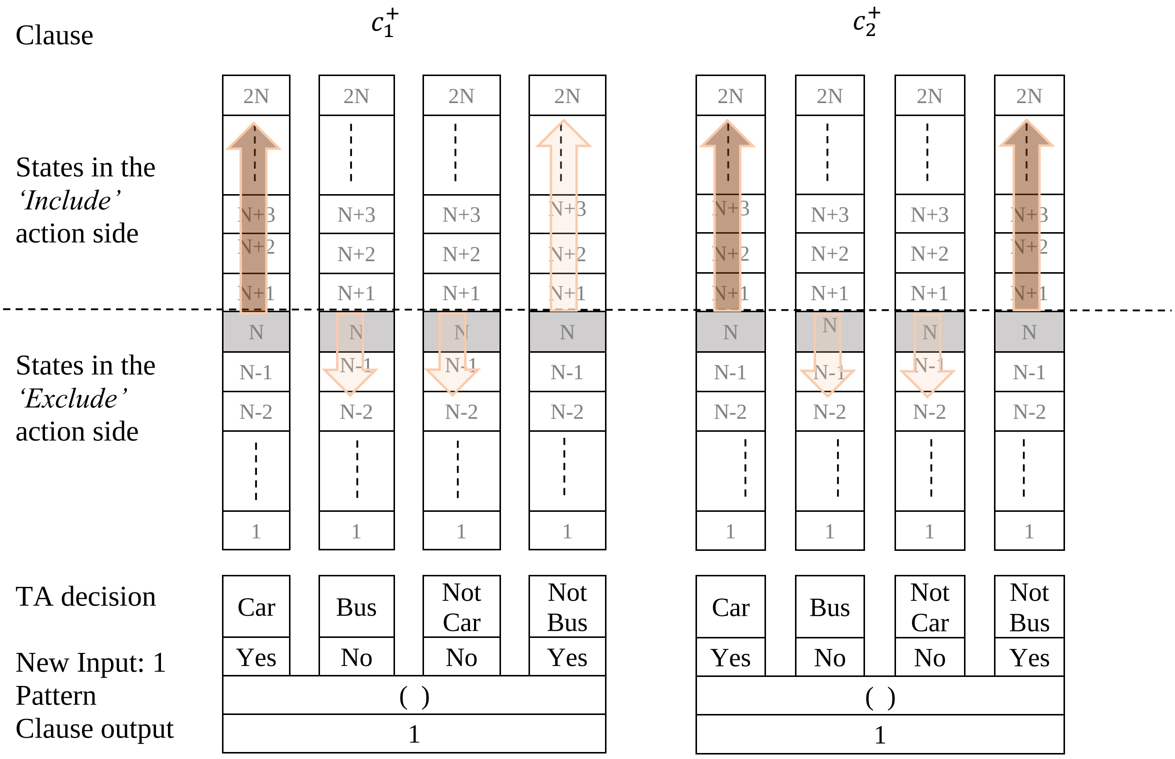


Figure 2.4: TA states of two positive clauses in the TM. They here receive the first training sample (input) and are ready to update the states for the next round.

threshold $T = 1$ makes sure that the two clauses are optimally utilized. Each clause needs four TAs to represent the input literals (Car, Bus, Not Car, and Not Bus). As we can see in Figure 2.4, the TA states of all the TAs in both clauses initiate from the weakest state of the *Exclude* action. Since all literals are excluded from both clauses, they still have not learned any pattern. According to (2.2), each clause outputs 1. Now, we feed different scenarios in Table 2.1 randomly at each training iteration and see how the states of TAs move.

Processing of first training sample with updating of the TA states: We first feed the second scenario in Table 2.1 (**IF** Car **AND NOT** Bus **THEN** Crossing: Yes) to the TM clauses. Since the training sample is from the same class as the class which the clauses represent, they are eligible to receive Type I feedback (Flowchart in Figure 2.2). We assume that both the clauses satisfy the activation condition in (2.5). Since the clauses output 1, the literals which do not change the clause output (from 1 to 0) when they are included in the clause, have a higher probability ($\frac{s-1}{s}$) to be included in the clause (Type Ia feedback: (2.6). In the TM, since **AND** of 1s is 1, every TA which represent literals whose literal value is 1 receives Type Ia feedback. In this example, 1 is 'Yes'). Hence, the states of the TAs which appear for literal 'Car' and 'Not Bus' have a higher probability to change their states from N to $N+1$. This has been marked by long arrows in Figure 2.4. However, due to the stochastic nature of state transitions, TA which represents the literal 'Car' in the first clause and TAs which appear for the literal 'Car' and 'Not Bus' in the second clause change their states. These TAs are marked with dark long arrows.

The Type Ib feedback, on the other hand, is applied on the TAs which represent the literal of value 0. Including a 0-valued literal in the clause makes the claue evaluate to 0, so every TA which represents literals whose literal value is 0 receives Type Ib feedback. In this example, 0 is 'No'). These are Bus and Not Car. Since the state transition probability of Type Ib feedback is relatively small, the chance of the above TAs changing state, strengthening the exclude action, is relatively small ($\frac{1}{s}$). This has been indicated by small arrows in Figure 2.4.

Studying the learned patterns in the updated clauses after first round of training: The new states of the TAs in the clauses are shown in Figure 2.5. After the above changes to the states of the TAs in first clause, the recognized sub-pattern by the clause is (Car) (evaluation of clause output: (2.1)). This means, regardless of the action of the Bus, whenever the car drives, the clause outputs 1. Since the clause outputs 1 when the Car drives, we can say that the clause has recognized a sub-pattern, however, obviously an incomplete one. The TM expect clauses to identify more fine-tuned sub-patterns. How? We are going to see it soon. The second clause after the first training round has recognized the pattern (Car **AND NOT** Bus). This means, the clause activates only when the second scenario appears in the training: (Car **AND NOT** Bus).

Processing the second training sample with corresponding TA state updates: We assume that the randomly selected next training sample is again the same. Accordingly, we are ready to update the states of the TAs of each clause in a second round of training. In the second round of training, we suppose only c_1^+ is eligible to receive feedback

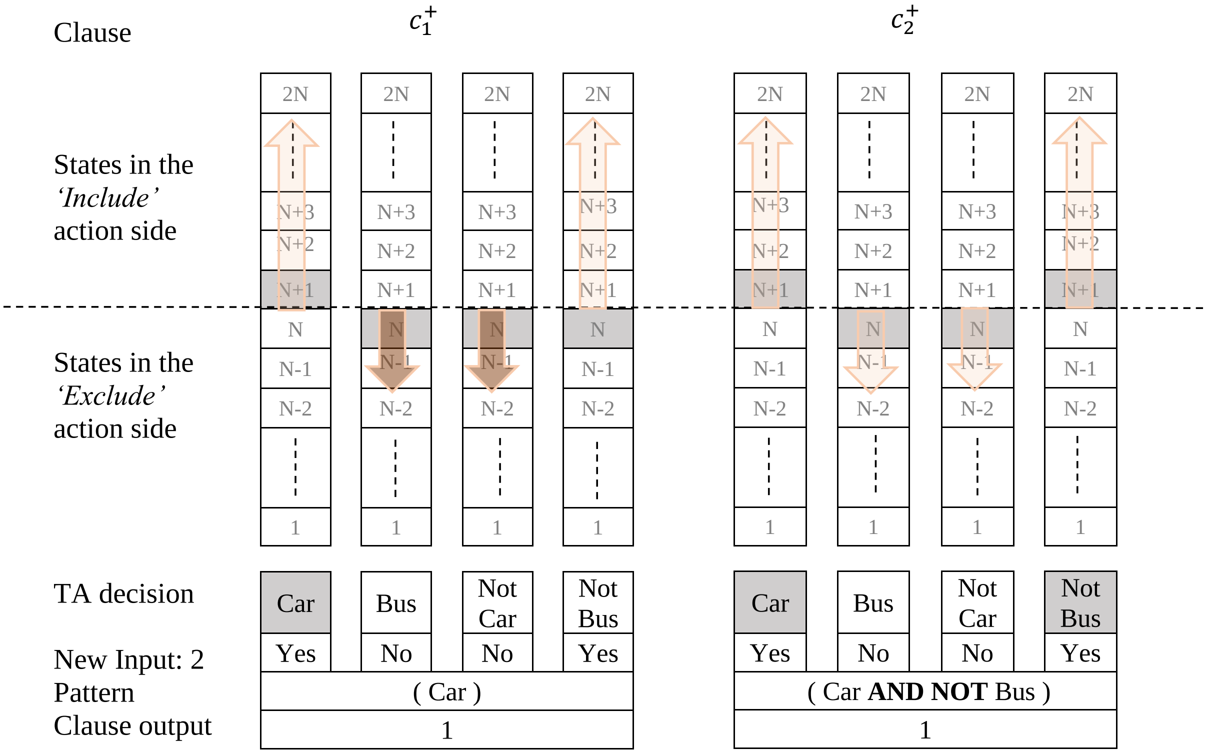


Figure 2.5: TA states of two positive clauses in the TM. They here receive a second sample and are ready to update the states for the next round.

(due to the stochasticity in (2.5)). Since the recognized pattern by the first clause is (Car) and that is also part of the training sample scenario, for the new training round, c_1^+ again outputs 1. Since this clause receives Type I feedback, TA states of literal Car and Not Bus have a higher chance to move further towards *Include* action while the TA states of literal Bus and Not Car have a smaller chance of moving further towards *Exclude* action. Due to the stochasticity, TA states of only literal Bus and Not Car are updated (which has been indicated by dark small arrow in Figure 2.5).

Studying the learned patterns in the updated clauses after second round of training: As one can see (Figure 2.6), the changes made to the TA states at the second training round do not make any difference to the pattern recognized by any of the clauses: c_1^+ still recognizes (Car) while c_2^+ recognizes (Car **AND NOT** Bus).

Processing a third training sample with TA state updating: Now, the training sample is the third scenario in Table 2.1. If we again assume that both the clauses are eligible to receive feedback, clause evaluation should be done on both of them (clause output calculation: (2.1)). Literals of value 0 makes clauses that include them evaluate to 0. For instance, c_1^+ wants the car to move (includes Car). However, when the pattern in the training sample says that the car does not move (Not Car), the clause identifies that it is not the pattern it learned and outputs 0. If we take the binary version of the considered example to calculate the clause output using (2.1), inclusion of 0 (the literal value of car is 'No' and 0 appears for 'No' in the binary version of the example) in the clause makes the clause outputs 0. Similarly, clause c_1^+ also outputs 0.

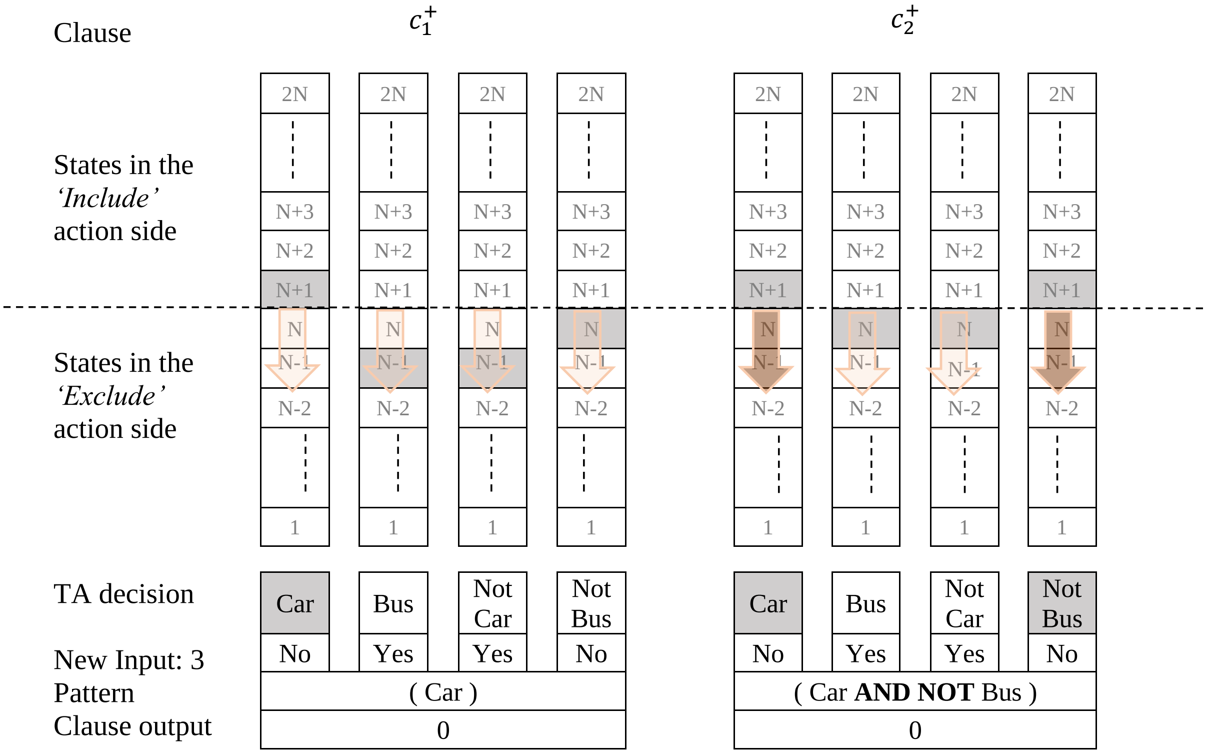


Figure 2.6: TA states of two positive clauses in the TM, taking third training sample and ready to update the states for the next round.

Even though the training sample is from class 'Yes' and the clauses were assigned to identify the sub-patterns of the same class, both clauses output 0. This is the situation of false negative outputs of clauses. Hence, the Type Ib feedback is applied on the TAs to erase what the clauses learned and make them ready for learning new sub-patterns. As we can observer from Figure 2.6, the TA states of all literals in both clauses have a chance to move towards the *Exclude* action of the TA, however, with a small probability. Due to the randomization in learning, only the TAs which represent literal Car and Not Bus in the second clause succeed.

Studying the learned patterns in the updated clauses after third round of training: Since there were no changes to the states of the TAs in the first clause, the pattern it recognized stays the same: (Car). Yet, the changes made on the states of the TAs in the second clause have erased the pattern it learned completely. The states of all the TAs in the second clause are now at the starting states (weakest *Exclude*) and ready to learn a new pattern. These new updated clauses after the third training round can be found in Figure 2.7.

Processing of the fourth training sample and updating TA states accordingly: The randomly selected next training sample is again the third scenario in Table 2.1. This makes the first clause output 0: the pattern identified by c_1^+ is (Car), but the input scenario says that the car is not moving. The c_2^+ has been reset to its initial state. Hence, according to (2.1), it automatically outputs 1.

Now, if we assume that both clauses receive the feedback after (2.5), both the clauses

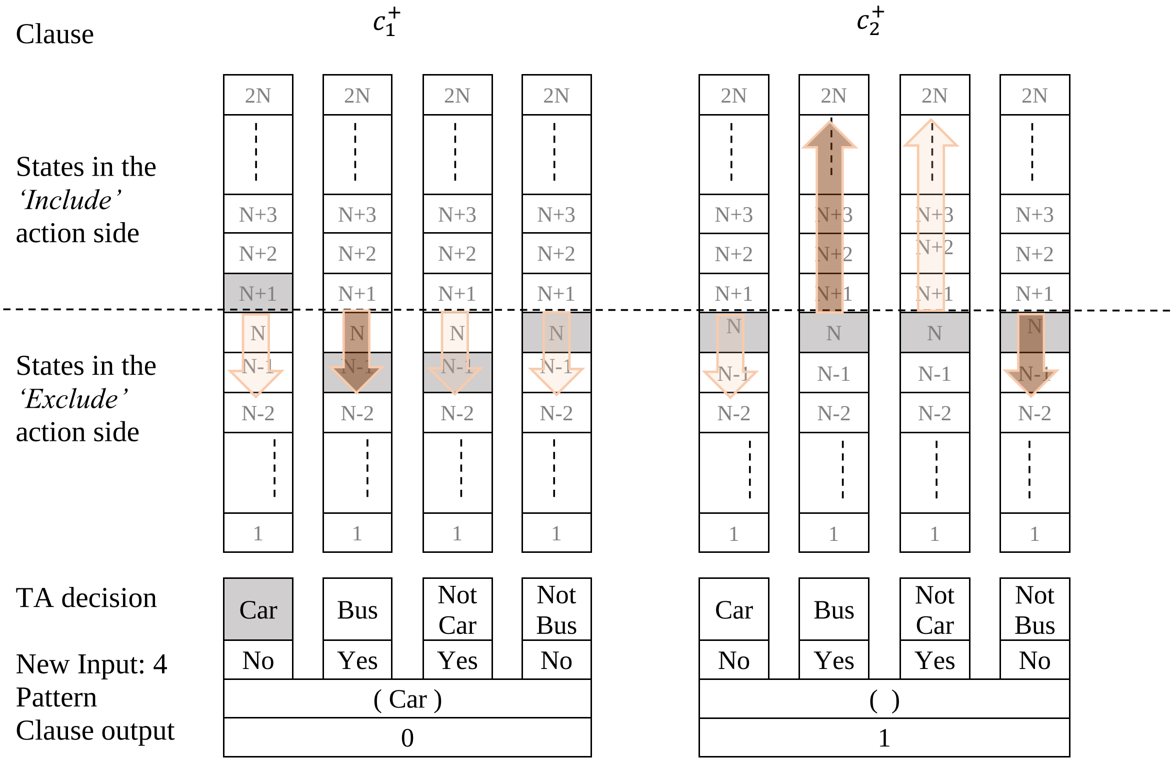


Figure 2.7: TA states of two positive clauses in the TM, taking forth training sample and ready to update the states for the next round.

still receive Type I feedback, since they are positive clauses and they are there to recognize pattern of the class 'Yes'. Next, the TAs which receive Type Ia and Type Ib in them should be decided. Once we have Type I feedback and the clause output is 0, all TAs in that clause receive Type Ib feedback. Thus, in this training round, TAs in clause c_1^+ receive Type Ia feedback. This has been marked with arrows in the first clause in Figure 2.7. The TA with the dark arrow, however, is the only TA makes actual changes to its states due to randomness. The TAs which appears for literal Car and Not Bus in the second clauses also receive Type Ib feedback (since inclusion of them in the clause makes the clause outputs 0). Nevertheless, the TA which represents only Not Bus literal is able to make changes to its states. Now, the clause c_2^+ has the opportunity to learn the pattern (Bus **AND NOT** Car). To do so, those literals should be included in the clause. In TM, this is done by giving the TAs which represent those literals Type Ia feedback. Even though the chance of moving towards *Include* action in those TAs is high, TA which appears for Bus succeed. This has been marked with long arrows in Figure 2.7.

Studying the learned patterns in the updated clauses after fourth round of training: As we can see from Figure 2.8, the changes made on states of TAs in the first clause at the fourth training round have not changed the pattern it recognized. Hence, it still activates only for training samples containing 'Car driving' scenario, regardless of the action of the bus. Contrarily, the second clause recognizes a new pattern after the updates of its TA states during the fourth training round. This new pattern recognized by c_2^+ is (Bus).

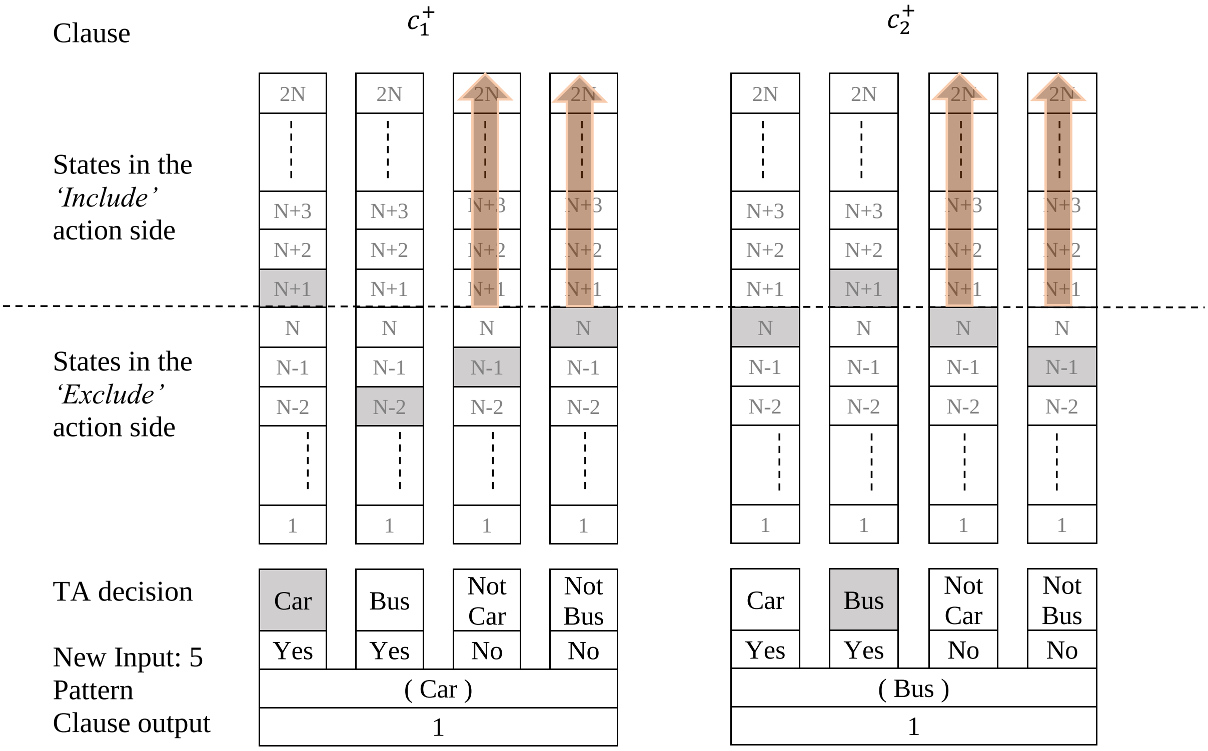


Figure 2.8: TA states of two positive clauses in the TM, taking fifth training sample and ready to update the states for the next round.

Processing the fifth training sample and updating TA states accordingly: The last training sample we consider in this demonstration is the first scenario in Table 2.1: (**IF** Car **AND** Bus **THEN** Crossing: No). Even though, the training sample is not from the class which the clause represent, the clauses output 1. We thus have clauses that produce false positive output (they recognize the wrong class). To combat this, Type II is applied. As we already know, the job of Type II feedback is to change the clause output of the selected clauses from 1 to 0. Type II feedback does this by simply including literals which evaluate the clause output to 0 in the clause. In a TM with binary literals, these are the literals with value 0 ('No' in this example). The equivalent literals in this training sample are the Not Car and Not Bus. Hence, the states of the TAs which represent these literals are pushed towards *Include* action with probability equal to unity, assuming that both clauses are eligible to receive feedback after the assessment in (2.8). This is displayed by long dark arrows in Figure 2.8.

Studying the learned patterns in the updated clauses after fifth round of training: The state transitions made during the fifth training round has now included literal Not Bus in clause c_1^+ in addition to the previous literal Car it had in the previous training round. Similarly, these updates have also included Not Car literal in the second clause in addition to the literal Bus it had previously. Together with these newly included literals, c_1^+ now recognizes the pattern (Car **AND NOT** Bus) while c_2^+ recognizes the pattern (**NOT** Car **AND** Bus). The new states of TAs and the patterns recognized by the clauses are shown in Figure 2.9.

Now the two clauses studied have recognized both the sub-patterns of the class 'Yes'.

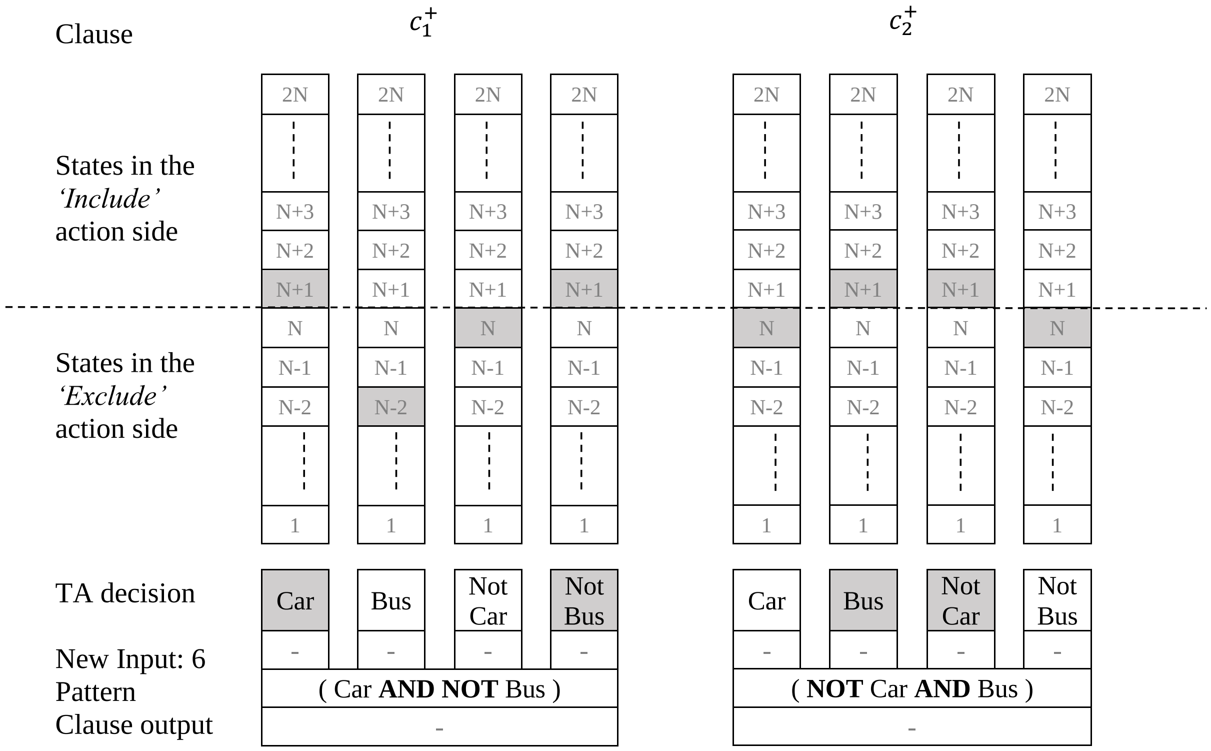


Figure 2.9: TA states of two positive clauses in the TM, taking sixth training sample and ready to update the states for the next round.

However, the clauses are still unstable as the states of the TAs in these clauses still fluctuate around the center. As soon as additional training samples are processed from each sub-pattern, the states gradually change towards the end-states, increasing the confidence in the actions learnt.

We have now investigated the procedure for updating the states of the TAs of the clauses for class 'Yes'. The updating procedure for the clauses which represent the class 'No' is the same. Additionally, updating of states of TAs in clauses in a complex problem still follows the same procedure, usually with more clauses to represent more sub-patterns in each class.

Chapter 3

Thesis Contributions

The TM, being based on propositional logic to form the classifier and straightforward bitwise operations for recognition and learning, is computationally simple. This structure makes the TM interpretable, yet it achieves competitive accuracy for many pattern recognition problems.

However, the original TM architecture is limited by its propositional foundation in several ways. Firstly, while many applications require continuous inputs and outputs, TMs process propositional inputs with propositional operators, producing propositional outputs. As such, TMs are not natively suited for dealing with continuous values, for instance in regression problems. This thesis addresses this limitation by introducing (1) effective pre-processing and learning techniques for continuous features, (2) mechanisms for producing continuous output, (3) integer clause weighing for more compact pattern representation, (4) classification confidence assessment, (5) regression-based analysis of visual imagery, and (6) deterministic multi-step learning that replaces random number generation for guiding the TA, to increase the efficiency of micro-controller TM implementations.

In this chapter, we present the detailed overview of the research approach and contributions of the thesis.

3.1 Continuous Input to the Tsetlin Machine

Approach: To address Research Question 1, we propose a data preprocessing procedure that transforms the input losslessly into a binary representation that maintains the semantic relationship between numbers. In brief, the preprocessing procedure considers each unique data sample in each continuous feature as a potential threshold. The original data samples are then compared with the complete set of thresholds to create a new feature matrix only containing bits. This method is presented in Paper A.

We further enhance the above method, first by standardizing features to support scale shifts in the transition from training data to real-world operation and then by sampling to reduce the number of binarization thresholds, relying on stratification to minimize loss of accuracy. The method is presented in Paper B.

In Paper C, we propose a novel approach to represent continuous TM inputs. Instead of using one TA for every unique threshold found when Booleanizing continuous input, we employ two SSL automata to learn discriminative lower and upper bounds.

Evaluation: The thresholding approach is evaluated and analyzed using an artificial dataset. The resulting TM is further applied to forecast dengue outbreaks of all the seventeen regions in Philippines using the spatio-temporal properties of the data. Experimental results show that the dengue outbreak forecasts made by the TM are more accurate than those obtained by a SVM, Decision Trees (DTs), and several multi-layered Artificial Neural Networks (ANNs), both in terms of forecasting precision and F1-score.

Results after improving the thresholding approach with standardizing and sampling show that the loss of accuracy due to threshold sampling is insignificant.

We evaluate the performance of the third scheme empirically using five datasets, along with a study of interpretability. On average, TMs with SSL feature representation use 4.3 times fewer literals than the TM with static threshold-based features. Furthermore, in terms of average memory usage and F1-Score, our approach outperforms simple Multi-Layered Artificial Neural Networks (ANNs), DTs, SVMs, K-Nearest Neighbor, Random Forest (RF), Gradient Boosted Trees (XGBoost), Explainable Boosting Machines (EBMs), as well as the standard and real-value weighted TMs. We further outperform Neural Additive Models on Fraud Detection and StructureBoost on CA-58 in terms of AUC while performing competitively on COMPAS.

3.2 Continuous Output from the Tsetlin Machine

Approach: We address Research Question 2 as follows. To produce continuous output that leverages the natural ordering of numbers, we modify the inner learning and inference mechanism of the TM. That is, the inputs are transformed into a single continuous output, rather than to distinct categories. We achieve this by eliminating the polarities of the clauses, and by summing all the non-polarized votes, mapping the sum into a continuous output. The new variant of the TM is called Regression Tsetlin Machine (RTM). This approach is presented in Paper D.

Evaluation: The behavior of the new algorithm was studied by applying it to both artificial and real-world datasets. In brief, RTM demonstrated superior performance in comparison with Regression Trees (RTs), RF, and Support Vector Regression (SVR), when predicting Dengue incidences in the Philippines, heating load in the Energy Performance dataset, Real Win Rate in the Stock Selection dataset, and house price per unit area in the Real Estate Valuation dataset. The RTM also extrapolates reasonably well outside the minimum and maximum output values found in the training data.

3.3 Integer-Weighted Clauses for Compact Pattern Representation in the Tsetlin Machine

Approach: The challenges posed by Research Question 3 is overcome by extending each clause with the SSL automaton [58]. We adapt this automaton so that it can learn an effective clause weight by interacting with the corresponding TA team. As a result, the set

of clauses can be rendered significantly more compact, without sacrificing the 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 higher accuracy as a team. Furthermore, the clauses that are sufficiently accurate are assigned larger weights so that they can operate more independently. This approach is applied on RTM in Paper E and on the basic TM in Paper F.
Evaluation: We evaluate the potential of the integer weighted RTM empirically using two artificial datasets. The results show that the integer weighted RTM is able to acquire on par or better accuracy using significantly less computational resources compared to regular RTM and an RTM with real-valued weights.

For the basic TM, we evaluate the performance of the new scheme empirically using five datasets, along with a study of interpretability. On average, our Integer weighted Tsetlin Machine (IWTM) uses 6.5 times fewer literals than the original 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 ANNs, DTs, SVMs, K-Nearest Neighbor, RF, XGBoost, EBMs, as well as the standard and real-value weighted TMs. IWTM finally outperforms Neural Additive Models on Fraud Detection and Structure Boost on CA-58 in terms of AUC, while performing competitively on COMPAS.

3.4 Classification Confidence, Ranked Predictions and AUC with Tsetlin Machines

Approach: To address Research Question 4, instead of using a hard decision function in the TM output layer for binary classification, we propose a novel approach to measure the classification confidence score using a logistic function. The resulting TM output layer is able to measure classification confidence from the propositional logic expressions that match the input. Based on the confidence score, we demonstrate how to rank predictions, producing ROC curves and calculating AUC. The complete details of the approach can be found in Paper G.
Evaluation: Empirically, using four real-world datasets, we show that the AUC is a more sensitive measure of the TM performance when compared to the Accuracy. Further, the AUC-based evaluations show that the TM performs on par or better than widely used machine learning algorithms. We thus believe our scheme will make the TM more suitable for use in decision support, where the user needs to inspect and validate predictions, in particular, those being uncertain.

3.5 Identifying Patterns in Images Using the Convolutional Tsetlin Machine and Use Them to Produce Continuous output

Approach: We introduce the Convolutional Regression Tsetlin Machine (C-RTM) to address Research Question 5. C-RTM is a novel approach that combines the properties of

both CTM and RTM. In brief, the patterns recognized in an image by the Convolutional TM clauses are piled together and mapped to a continuous output as in the RTM. Paper H presents the approach in detail.
Evaluation: We evaluate the performance of C-RTM using 72 different artificial datasets, with and without noise in the training data. Our empirical results show the competitive performance of the C-RTM in comparision to two standard CNNs. The interpretability of the identified sub-patterns by C-RTM clauses is also analyzed and discussed. Additionally, the performance of the C-RTM on two real-world datasets is compared against CNN. In addition to the competitive performance in terms of mean absolute error (MAE), C-RTM filters perform significantly fewer calculations (only **AND** operations) during the convolution. Further, the filters learn which image locations are important, and then move directly to these lo-cations during inference. This approach helps the C-RTM to consume significantly less memory both during training as well as testing

3.6 A Multi-Step Finite-State Automaton for Arbitrarily Deterministic Tsetlin Machine Learning

Approach: To deal with Research Question 6, we propose a novel finite state LA that can replace the TAs of the TM, for increased determinism. The new automaton uses multi-step deterministic state jumps to reinforce sub-patterns. Simultaneously, flipping a coin to skip every d'th state update ensures diversification by randomization. The d-parameter thus allows the degree of randomization to be finely controlled. A detailed description of the approach can be found in Paper I.
Evaluation: Both theoretically and empirically, we establish that the proposed automaton converges to the optimal action almost surely. Further, used together with the TM, only substantial degrees of determinism reduces accuracy. Energy-wise, random number generation constitutes switching energy consumption of the TM, saving up to 11 mW power for larger datasets with high d values[1]. Our new learning automaton approach thus facilitate low-energy machine learning.

3.7 Applications

Approach: In regards to Research Question 7, we select two applications from two domains where both accuracy and interpretability are of significant importance - network intrusion detection and public transport passenger count forecasting in pandemic scenarios[2]. For the first application, to separate intrusions from the normal behaviors, we use

[1] The real implementation of the proposed algorithm on a micro-controller and energy calculations were done by Rishad Shafik, Alex Yakovlev, and Adrian Wheeldon, co-authors of the corresponding paper. The same authors wrote the relevant section in the paper. The rest of the co-authors contributed to the paper mainly in the writing phase.

[2] Data gathering (public transport passenger count), analyzing, and preparing to apply them on TM were done by Sinziana Rasca, one of the other authors of the paper. The same author wrote those relevant sections in the paper. The rest of the co-authors contributed to the paper mainly in the writing phase.

a basic TM which classifies data into relevant classes (Paper J). For the application of forecasting passenger counts, RTM is utilized (Paper K).

Evaluation: We evaluate the TM over the Knowledge Discovery and Data Mining 1999 (KDD'99) dataset and the experimental results demonstrate that the proposed TM based approach is capable of achieving superior classification performance in comparison to several ANNs, SVMs, DTs, RF, and K-Nearest Neighbor machine learning algorithms while preserving the interpretability.

Results of the second application show that RTM obtains the lowest mean absolute error for forecasting the variation in public transport ridership in comparison to all other machine learning models tested (RF, RTs, SVR, Moving Average). Our evaluation also demonstrates interpretability through the formulation of forecasting rules.

Chapter 4

Publications

Eleven papers from the total of 16 published papers have been incorporated in this dissertation. The contributions of these papers address the research questions presented in Section 3 as follows:

- Research question 1 is addressed in Paper A, Paper B, and Paper C.
- Research question 2 is addressed in Paper D.
- Research question 3 is addressed in Paper E and Paper F.
- Research question 4 is addressed in Paper G.
- Research question 5 is addressed in Paper H.
- Research question 6 is addressed in Paper I.
- Additionally, Paper J and Paper K present potential two applications of TMs in more details.

The papers and their abstracts have been listed below:

Paper A: A Scheme for Continuous Input to the Tsetlin Machine with Applications to Forecasting Disease Outbreaks.

In this paper, we apply a new promising tool for pattern classification, namely, the *TM*, to the field of disease forecasting. The TM is interpretable because it is based on manipulating expressions in propositional logic, leveraging a large team of TAs. Apart from being interpretable, this approach is attractive due to its low computational cost and its capacity to handle noise. To attack the problem of forecasting, we introduce a preprocessing method that extends the TM so that it can handle continuous input. Briefly stated, we convert continuous input into a binary representation based on thresholding. The resulting extended TM is evaluated and analyzed using an artificial dataset. The TM is further applied to forecast dengue outbreaks of all the seventeen regions in Philippines using the spatio-temporal properties of the data. Experimental results show that dengue outbreak forecasts made by the TM are more accurate than those obtained by SVM, DTs, and ANNs, both in terms of forecasting precision and F1-score.

Paper B: Adaptive Continuous Feature Binarization for Tsetlin Machines Applied to Forecasting Dengue Incidences in the Philippines

The *TM* is a recent interpretable machine learning algorithm that requires relatively modest computational power, yet attains competitive accuracy in several benchmarks. TMs are inherently binary; however, many machine learning problems are continuous. While binarization of continuous data through brute-force thresholding has yielded promising accuracy, such an approach is computationally expensive and hinders extrapolation. In this paper, we address these limitations by standardizing features to support scale shifts in the transition from training data to real-world operation, typical for e.g. forecasting. For scalability, we employ sampling to reduce the number of binarization thresholds, relying on stratification to minimize loss of accuracy. We evaluate the approach empirically using two artificial datasets before we apply the resulting TM to forecast dengue outbreaks in the Philippines using the spatio-temporal properties of the data. Our results show that the loss of accuracy due to threshold sampling is insignificant. Furthermore, the dengue outbreak forecasts made by the TM are more accurate than those obtained by SVMs, DTs, and several ANNs, both in terms of forecasting precision and F1-score.

Paper C: Adaptive Sparse Representation of Continuous Input for Tsetlin Machines Based on Stochastic Searching on the Line

This paper introduces a novel approach to representing continuous inputs in TMs. Instead of using one TA for every unique threshold found when Booleanizing continuous input, we employ two SSL to learn discriminative lower and upper bounds. The two resulting Boolean features are adapted to the rest of the clause by equipping each clause with its own team of SSLs, which update the bounds during the learning process. Two standard TAs finally decide whether to include the resulting features as part of the clause. In this way, only four automata altogether represent one continuous feature (instead of potentially hundreds of them). We evaluate the performance of the new scheme empirically using five datasets, along with a study of interpretability. On average, TMs with SSL feature representation use 4.3 times fewer literals than the TM with static threshold-based features. Furthermore, in terms of average memory usage and F1-Score, our approach outperforms simple Multi-Layered Artificial Neural Networks, Decision Trees, SVMs, K-Nearest Neighbor, RF, XGBoost, and EBMs, as well as the standard and real-value weighted TMs. Our approach further outperforms Neural Additive Models on Fraud Detection and StructureBoost on CA-58 in terms of the Area Under Curve while performing competitively on COMPAS.

Paper D: The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression

Relying simply on bitwise operators, the recently introduced *TM* has provided competitive pattern classification accuracy in several benchmarks, including text understanding. In this paper, we introduce the *RTM*, a new class of TMs designed for continuous input and output, targeting non-linear regression problems. In all brevity, we convert continuous input into a binary representation based on thresholding, and transform the propositional formula formed by the TM into an aggregated continuous output. Our empirical comparison of the RTM with state-of-the-art regression techniques reveals either superior or on par performance on five datasets.

Paper E: Integer Weighted Regression Tsetlin Machines

The RTM addresses the lack of interpretability impeding state-of-the-art nonlinear regression models. It does this by using conjunctive clauses in propositional logic to capture the underlying non-linear frequent patterns in the data. These, in turn, are combined into a continuous output through summation, akin to a linear regression function, however, with non-linear components and binary weights. However, the resolution of the RTM output is proportional to the number of clauses employed. This means that computation cost increases with resolution. To address this problem, we here introduce integer weighted RTM clauses. Our integer weighted clause is a compact representation of multiple clauses that capture the same sub-pattern — w repeating clauses are turned into one, with an integer weight w. This reduces computation cost w times, and increases interpretability through a sparser representation. We introduce a novel learning scheme, based on so-called stochastic searching on the line. We evaluate the potential of the integer weighted RTM empirically using two artificial datasets. The results show that the integer weighted RTM is able to acquire on par or better accuracy using significantly less computational resources compared to regular RTM and an RTM with real-valued weights.

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

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 *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 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 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 *SSL*. 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 ANNs, DTs, SVMs, K-Nearest Neighbor, RF, XGBoost, 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.

Paper G: On Obtaining Classification Confidence, Ranked Predictions and AUC with Tsetlin Machines

TMs are a promising approach to machine learning that uses Tsetlin Automata to produce patterns in propositional logic, leading to binary (hard) classifications. In many applications, however, one needs to know the confidence of classifications, e.g. to facilitate risk management. In this paper, we propose a novel scheme for measuring TM confidence based on the logistic function, calculated from the propositional logic patterns that match the input. We then use this scheme to trade off precision against recall, producing AUC for TMs. Empirically, using four real-world datasets, we show that AUC is a more sensitive measure of TM performance compared to Accuracy. Further, the AUC-based evaluations show that the TM performs on par or better than widely used machine learning algorithms. We thus believe our scheme will make the TM more suitable for use in decision support, where the user needs to inspect and validate predictions, in particular, those being uncertain.

Paper H: Convolutional Regression Tsetlin Machine: An Interpretable Approach to Convolutional Regression

The CTM, a variant of the TM, represents patterns as straightforward AND-rules, to address the high computational complexity and the lack of interpretability of Convolutional Neural Networks (CNNs). The CTM has shown competitive performance on MNIST, Fashion-MNIST, and Kuzushiji-MNIST pattern classification benchmarks, both in terms of accuracy and memory footprint. However, the CTM has so far only been applied to binary output. This paper proposes the C-RTM that extends the CTM to support continuous output problems in image analysis. C-RTM identifies patterns in images using the convolution operation as in the CTM and then maps the identified patterns into a real-valued output in the RTM. The C-RTM thus unifies the two approaches.

We evaluate the performance of C-RTM using 72 different artificial datasets, with and without noise in the training data. Our empirical results show the competitive performance of C-RTM compared to two standard CNNs. The interpretability of the identified sub-patterns by C-RTM clauses is also analyzed and discussed. Additionally, the performance of C-RTM on two real-world datasets is compared against CNN. In addition to the competitive performance in terms of mean absolute error, C-RTM filters perform significantly fewer calculations (only **AND** operations) during the convolution. Further, the filters learn which image locations are important, and then move directly to these locations during inference. This approach helps C-RTM consume significantly less memory both during training and testing.

Paper I: A Multi-Step Finite-State Automaton for Arbitrarily Deterministic Tsetlin Machine Learning

Due to the high arithmetic complexity and scalability challenges of deep learning, there is a critical need to shift research focus towards energy efficiency. TMs are a recent approach to machine learning that has demonstrated significantly reduced energy compared to neural networks alike, while providing comparable accuracy on several benchmarks. However, TMs rely heavily on energy-costly random number generation to stochastically guide a team of TAs in TM learning. In this paper, we propose a novel finite-state learning automaton that can replace the TA in the TM, for increased determinism. The new automaton uses multi-step deterministic state jumps to reinforce sub-patterns, without resorting to randomization. A determinism parameter d finely controls trading off the energy consumption of random number generation, against randomization for increased accuracy. Randomization is controlled by flipping a coin before every d'th state jump, ignoring the state jump on tails. E.g., $d = 1$ makes every update random and $d = \infty$ makes the automaton completely deterministic. Both theoretically and empirically, we establish that the proposed automaton converges to the optimal action almost surely. Further, used together with the TM, only substantial degrees of determinism reduces accuracy. Energy-wise, random number generation constitutes switching energy consumption of the TM, saving up to 11 mW power for larger datasets with high d values. Our new learning automaton approach thus facilitate low-energy machine learning.

Paper J: Intrusion Detection with Interpretable Rules Generated Using the Tsetlin Machine

The rapid deployment in information and communication technologies and internet-based services have made anomaly based network intrusion detection ever so important for safeguarding systems from novel attack vectors. To this date, various machine learning mechanisms have been considered to build intrusion detection systems. However, achieving an acceptable level of classification accuracy while preserving the interpretability of the classification has always been a challenge. In this paper, we propose an efficient anomaly based intrusion detection mechanism based on the TM. We have evaluated the proposed mechanism over the Knowledge Discovery and Data Mining 1999 (KDD'99) dataset and the experimental results demonstrate that the proposed TM based approach is capable of achieving superior classification performance in comparison to several simple ANNs, SVMs, DTs, RF, and K-Nearest Neighbor machine learning algorithms while preserving the interpretability.

Paper K: Public Transport Passenger Count Forecasting in Pandemic Scenarios Using Regression Tsetlin Machine. Case Study of Agder, Norway

Challenged by the effects of the *COVID-19 pandemic*, public transport is suffering from low ridership and staggering economic losses. One of the factors which triggered such losses was the lack of preparedness among governments and public transport providers. The present paper explores the use of a novel machine learning algorithm, namely *Regression Tsetlin Machine*, in using historical passenger transport data from the current COVID-19 pandemic and pre-pandemic period to forecast pandemic-scenarios for public transport patronage variations. Results show that the *Regression Tsetlin Machine* has the best accuracy of forecasts compared to four other models usually employed in the field.

Chapter 5

Conclusion

In this thesis, we addressed several fundamental TM limitations caused by the propositional nature of the TMs. As such, the TMs lack support for continuous inputs, cannot produce numerically ordered regression outputs, lacks pattern weighing, cannot measure classification confidence, and requires extensive random number generation in learning. We have resolved these limitations by designing, analysing and evaluating several new TM mechanisms.

To ensure that the TM can deal with continuous features, we proposed a data preprocessing procedure that transforms the inputs losslessly into a binary representation that maintains semantic relationship between numbers (thresholding approach). We further enhanced the above method, first by standardizing features to support scale shifts in the transition from training data to real-world operation. We then significantly reduced the required number of binarization thresholds by means of sampling, relying on stratification to minimize loss of accuracy. To avoid pre-processing altogether, we next employed two SSL automata to learn discriminative lower and upper bounds, instead of using one TA for every unique threshold found when Booleanizing continuous input. Our conclusion is that the thresholding approach enables the TM to work with continuous features. Further, our experiments with input standardizing and sampling show that the loss of accuracy due to threshold sampling is insignificant. Furthermore, on average, the TMs with SSL feature representation use 4.3 times fewer literals than the TM with static threshold-based features representation. As discuss further in the relevant papers, the results on real and artificial data and the performance comparisons against the state-of-the-art approaches support the above conclusions.

To maintain the natural order of numbers, we next proposed an approach that transforms the inputs into a single continuous output, rather than to distinct categories, which we coined as the RTM. We achieved this by eliminating the polarities of the clauses, and by summing all the non-polarized votes, mapping the sum into a continuous output. The RTM demonstrated superior performance in comparison with RTs, RFs, and SVR on four real-world benchmark datasets. We also modified the CTM in a similar way to also deal with regression for image input, giving rise to the C-RTM. The C-RTM analyzes visual imagery and predicts additive clause output as a continuous value instead of a class. The interpretability of the identified sub-patterns by the C-RTM clauses was also analyzed and discussed using artificial data. Additionally, the performance of the C-RTM on two real-world datasets was compared against CNN. In addition to the competitive performance in terms of MAE, the C-RTM filters performed significantly fewer calculations (only AND operations) during the convolution. Further, the filters learned which image locations are important, and then moved directly to these locations during inference. This approach helped C-RTM consume significantly less memory both during training and testing.

The proposed pattern weighing scheme, by extending each clause with an SSL automaton, was able to represent class patterns in a more compact way. In the learning phase of these weights, the clauses that are sufficiently accurate learned larger weight values so that they could operate more independently. The inaccurate clauses, on the other hand, obtained smaller weight values so that they had to team up to obtain high accuracy as a team. The weighing scheme was evaluated both on the TM and the RTM using artificial and real datasets. The results showed that the proposed integer weighted RTM was able to acquire on par or better accuracy using significantly less computational resources compared to the regular TM and the TM with real-valued weights. Additionally, in terms of F-1 score, the TM with the proposed weighing scheme (IWTM) outperformed simple ANNs, DTs, SVMs, K-Nearest Neighbor, RF, XG-Boost, and EBMs.

We modified the output layer of the TM by replacing the hard decision function, used for binary classification, with a logistic function to measuring the classification confidence score. Based on the confidence score, we demonstrated how to rank predictions, producing ROC curves and calculating the AUC. Using four real-world datasets, we showed that the AUC was a more sensitive measure of the TM performance when compared to Accuracy. Further, the AUC-based evaluations showed that the TM performed on par or better than widely used machine learning algorithms on above real-world datasets.

Finally, to minimize the extensive random number generation during the TM learning, we proposed a novel finite state learning automaton that can replace the TAs of the TM. The new automaton uses multi-step deterministic state jumps to reinforce sub-patterns. Simultaneously, flipping a coin to skip every d'th state update ensures diversification by randomization. The d-parameter thus allows the degree of randomization to be finely controlled. Both theoretically and empirically, we established that the proposed automaton converges to the optimal action almost surely. Further, used together with the TM, only substantial degrees of determinism reduced accuracy. Energy-wise, random number generation constitutes switching energy consumption of the TM, saving up to 11mW power for larger datasets with high d values. Our new learning automaton approach thus facilitate low-energy machine learning.

Overall, the above modifications have improved the performance of the TM in terms of accuracy, interpretability, inference speed, and memory consumption. In conclusion, the performance on the majority of the applications addressed in our papers are on par or better in comparison to the state-of-the-art machine learning models.

Bibliography

[1] Michael Lvovitch Tsetlin. “On Behaviour of Finite Automata in Random Medium”. In: *Avtomat. i Telemekh* 22.10 (1961), pp. 1345–1354.

[2] M A L Thathachar and P S Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004.

[3] Kumpati S Narendra and Mandayam AL Thathachar. *Learning Automata: An Introduction*. Courier corporation, 2012.

[4] V. A. Ponomarev. “A Construction of an Automaton Which Is Asymptotically Optimal in a Stationary Random Medium”. In: *Biofizika* 9 (1964), pp. 104–110.

[5] T Cover and M Hellman. “The Two-Armed-Bandit Problem with Time-Invariant Finite Memory”. In: *IEEE Transactions on Information Theory* 16.2 (1970), pp. 185–195.

[6] Robert R Bush and Frederick Mosteller. “Stochastic Models for Learning”. In: (1955).

[7] M Frank Norman. “On the Linear Model With two Absorbing Barriers”. In: *Journal of Mathematical Psychology* 5.2 (1968), pp. 225–241.

[8] Ole-Christoffer Granmo. “The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic”. In: *arXiv preprint arXiv:1804.01508* (2018).

[9] Seyed-Hamid Zahiri. “Learning Automata Based Classifier”. In: *Pattern Recognition Letters* 29.1 (2008), pp. 40–48.

[10] Pidaparthy S Sastry, G Dwarakanath Nagendra, and Naresh Manwani. “A Team of Continuous-Action Learning Automata for Noise-Tolerant Learning of Half-Spaces”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 40.1 (2009), pp. 19–28.

[11] PS Sastry and MAL Thathachar. “Learning Automata Algorithms for Pattern Classification”. In: *Sadhana* 24.4 (1999), pp. 261–292.

[12] Morten Goodwin, Anis Yazidi, and Tore Møller Jonassen. “Distributed Learning Automata for Solving a Classification Task”. In: *2016 IEEE congress on evolutionary computation (CEC)*. IEEE. 2016, pp. 3999–4006.

[13] Sorour Afshar, Mohammad Mosleh, and Mohammad Kheyrandish. “Presenting a New Multiclass Classifier Based on Learning Automata”. In: *Neurocomputing* 104 (2013), pp. 97–104.

[14] Qian Sang, Zongli Lin, and Scott T Acton. "Learning Automata for Image Segmentation". In: *Pattern Recognition Letters* 74 (2016), pp. 46–52.

[15] Erik Cuevas, Daniel Zaldivar, and Marco Pérez-Cisneros. "Seeking Multi-Thresholds for Image Segmentation with Learning Automata". In: *Machine Vision and Applications* 22.5 (2011), pp. 805–818.

[16] Bahman Damerchilu, Mohammad Sadegh Norouzzadeh, and Mohammad Reza Meybodi. "Motion Estimation Using Learning Automata". In: *Machine Vision and Applications* 27.7 (2016), pp. 1047–1061.

[17] Hasan Farsi, Reza Nasiripour, and Sajjad Mohammadzadeh. "Eye Gaze Detection Based on Learning Automata by Using SURF Descriptor". In: *Information Systems & Telecommunication* 6.1 (2018), pp. 41–49.

[18] Erik Cuevas, Fernando Wario, Daniel Zaldivar, and Marco Pérez-Cisneros. "Circle Detection on Images Using Learning Automata". In: *Artificial Intelligence, Evolutionary Computing and Metaheuristics.* Springer, 2013, pp. 545–570.

[19] Morten Goodwin and Anis Yazidi. "Distributed Learning Automata-Based Scheme for Classification Using Novel Pursuit Scheme". In: *Applied Intelligence* 50.7 (2020), pp. 2222–2238.

[20] Ali Asghar Rahmanian, Mostafa Ghobaei-Arani, and Sajjad Tofighy. "A Learning Automata-Based Ensemble Resource Usage Prediction Algorithm for Cloud Computing Environment". In: *Future Generation Computer Systems* 79 (2018), pp. 54–71.

[21] Wen Jiang, Cheng-Lin Zhao, Sheng-Hong Li, and Lawson Chen. "A New Learning Automata Based Approach for Online Tracking of Event Patterns". In: *Neurocomputing* 137 (2014), pp. 205–211.

[22] B John Oommen and EV de St Croix. "String Taxonomy Using Learning Automata". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27.2 (1997), pp. 354–365.

[23] Habib Motieghader, Ali Najafi, Balal Sadeghi, and Ali Masoudi-Nejad. "A Hybrid Gene Selection Algorithm for Microarray Cancer Classification Using Genetic Algorithm and Learning Automata". In: *Informatics in Medicine Unlocked* 9 (2017), pp. 246–254.

[24] Mohammad Savargiv, Behrooz Masoumi, and Mohammad Reza Keyvanpour. "A New Ensemble Learning Method Based on Learning Automata". In: *Journal of Ambient Intelligence and Humanized Computing* (2020), pp. 1–16.

[25] Andrew G Barto and P Anandan. "Pattern-Recognizing Stochastic Learning Automata". In: *IEEE Transactions on Systems, Man, and Cybernetics* 3 (1985), pp. 360–375.

[26] Ole-Christoffer Granmo and B John Oommen. "Solving Stochastic Nonlinear Resource Allocation Problems Using a Hierarchy of Twofold Resource Allocation Automata." In: *IEEE Transaction on Computers* (2010).

[27] B John Oommen, Sang-Woon Kim, Mathew T Samuel, and Ole-Christoffer Granmo. “A Solution to the Stochastic Point Location Problem in Metalevel Nonstationary Environments”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.2 (2008), pp. 466–476.

[28] Brian Tung and Leonard Kleinrock. “Using Finite State Automata to Produce Self-Optimization and Self-Control”. In: *IEEE transactions on parallel and distributed systems* 7.4 (1996), pp. 439–448.

[29] Noureddine Bouhmala and Ole-Christoffer Granmo. “Stochastic Learning for SAT-Encoded Graph Coloring Problems”. In: *International Journal of Applied Metaheuristic Computing (IJAMC)* 1.3 (2010), pp. 1–19.

[30] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. “A Novel Tsetlin Automata Scheme to Forecast Dengue Outbreaks in the Philippines”. In: *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2018, pp. 680–685.

[31] Hiroyuki Ogihara, Yusuke Fujita, Yoshihiko Hamamoto, Norio Iizuka, and Masaaki Oka. “Classification Based on Boolean Algebra and Its Application to the Prediction of Recurrence of Liver Cancer”. In: *Pattern Recognition (ACPR), 2013 2nd IAPR Asian Conference on*. IEEE. 2013, pp. 838–841.

[32] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. “A Bayesian Framework for Learning Rule Sets for Interpretable Classification”. In: *The Journal of Machine Learning Research (JMLR)* 18.1 (2017), pp. 2357–2393.

[33] Vitaly Feldman. “Hardness of Approximate Two-Level Logic Minimization and PAC Learning with Membership Queries”. In: *Jrnl. of Computer and System Sciences* 75.1 (2009), pp. 13–26.

[34] Adam R Klivans and Rocco A Servedio. “Learning DNF in Time 2O (n1/3)”. In: *Journal of Computer and System Sciences* 68.2 (2004), pp. 303–318.

[35] Vitaly Feldman. “Learning DNF Expressions From Fourier Spectrum”. In: *Conference on Learning Theory*. 2012, pp. 17–1.

[36] Leslie G Valiant. “A Theory of the Learnable”. In: *Communications of the ACM* 27.11 (1984), pp. 1134–1142.

[37] John R Hauser, Olivier Toubia, Theodoros Evgeniou, Rene Befurt, and Daria Dzyabura. “Disjunctions of Conjunctions, Cognitive Simplicity, and Consideration Sets”. In: *Jrnl. of Marketing Research* 47.3 (2010), pp. 485–496.

[38] Cynthia Rudin, Benjamin Letham, and David Madigan. “Learning Theory Analysis for Association Rules and Sequential Event Prediction”. In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 3441–3492.

[39] Tyler McCormick, Cynthia Rudin, and David Madigan. “A Hierarchical Model for Association Rule Mining of Sequential Events: An Approach to Automated Medical Symptom Prediction”. In: *Annals of Applied Statistics* (2011).

[40] William W Cohen. “Fast Effective Rule Induction”. In: *Machine learning proceedings 1995*. Elsevier, 1995, pp. 115–123.

[41] JH Donald. “Rule Induction-Machine Learning Techniques”. In: *Computing & Control Engineering Journal* 5.5 (1994), pp. 249–255.

[42] Alberto Fernandez, Salvador Garca, Julian Luengo, Ester Bernado-Mansilla, and Francisco Herrera. “Genetics-Based machine Learning for Rule Induction: State of the art, Taxonomy, and Comparative Study”. In: *IEEE Transactions on Evolutionary Computation* 14.6 (2010), pp. 913–941.

[43] J Juan Liu and J Tin-Yau Kwok. “An Extended Genetic Rule Induction Algorithm”. In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*. Vol. 1. IEEE. 2000, pp. 458–463.

[44] Pat Langley and Herbert A Simon. “Applications of Machine Learning and Rule Induction”. In: *Communications of the ACM* 38.11 (1995), pp. 54–64.

[45] Wouter Verbeke, David Martens, Christophe Mues, and Bart Baesens. “Building Comprehensible Customer Churn Prediction Models with Advanced Rule Induction Techniques”. In: *Expert systems with applications* 38.3 (2011), pp. 2354–2364.

[46] Helmut Braun and John S Chandler. “Predicting Stock Market Behavior Through Rule Induction: An Application of the Learning-From-Example Approach”. In: *Decision Sciences* 18.3 (1987), pp. 415–429.

[47] Sašo Džeroski, Jasna Grbović, William J Walley, and Boris Kompare. “Using Machine Learning Techniques in the Construction of Models. II. Data Analysis with Rule Induction”. In: *Ecological Modelling* 95.1 (1997), pp. 95–111.

[48] Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. “Neural Algebra of Classifiers”. In: *arXiv preprint arXiv:1801.08676* (2018).

[49] Adrian Phoulady, Ole-Christoffer Granmo, Saeed Rahimi Gorji, and Hady Ahmady Phoulady. “The Weighted Tsetlin Machine: Compressed Representations with Clause Weighting”. In: *Ninth International Workshop on Statistical Relational AI (StarAI 2020)*. 2020.

[50] Jin Huang and Charles X Ling. “Using AUC and Accuracy in Evaluating Learning Algorithms”. In: *IEEE Transactions on knowledge and Data Engineering* 17.3 (2005), pp. 299–310.

[51] Andrew P Bradley. “The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms”. In: *Pattern recognition* 30.7 (1997), pp. 1145–1159.

[52] Charles X Ling and Chenghui Li. “Data Mining for Direct Marketing: Problems and Solutions”. In: *Kdd*. Vol. 98. 1998, pp. 73–79.

[53] Chenhao Cui and Tom Fearn. “Modern Practical Convolutional Neural Networks for Multivariate Regression: Applications to NIR Calibration”. In: *Chemometrics and Intelligent Laboratory Systems* 182 (2018), pp. 9–20.

[54] Ine L Jernelv, Dag Roar Hjelme, Yuji Matsuura, and Astrid Aksnes. “Convolutional Neural Networks for Classification and Regression Analysis of One-Dimensional Spectral Data”. In: *arXiv preprint arXiv:2005.07530* (2020).

[55] Tomoyoshi Shimobaba, Takashi Kakue, and Tomoyoshi Ito. “Convolutional Neural Network-Based Regression for Depth Prediction in Digital Holography”. In: *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*. IEEE. 2018, pp. 1323–1326.

[56] Hilal Tayara, Kim Gil Soo, and Kil To Chong. “Vehicle Detection and Counting in High-Resolution Aerial Images Using Convolutional Regression Neural Network”. In: *IEEE Access* 6 (2017), pp. 2220–2230.

[57] Peter J. Denning, Douglas E Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R Young. “Computing as a Discipline”. In: *Computer* 22.2 (1989), pp. 63–70.

[58] B John Oommen. “Stochastic Searching On the Line and Its Applications to Parameter Learning in Nonlinear Optimization”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27.4 (1997), pp. 733–739.

Part II

Papers Contributing to the Dissertation

Paper A

A Scheme for Continuous Input to the Tsetlin Machine with Applications to Forecasting Disease Outbreaks

In this paper, we apply a new promising tool for pattern classification, namely, the Tsetlin Machine *(TM), to the field of disease forecasting. The TM is interpretable because it is based on manipulating expressions in propositional logic, leveraging a large team of Tsetlin Automata (TA). Apart from being interpretable, this approach is attractive due to its low computational cost and its capacity to handle noise. To attack the problem of forecasting, we introduce a preprocessing method that extends the TM so that it can handle continuous input. Briefly stated, we convert continuous input into a binary representation based on thresholding. The resulting extended TM is evaluated and analyzed using an artificial dataset. The TM is further applied to forecast dengue outbreaks of all the seventeen regions in Philippines using the spatio-temporal properties of the data. Experimental results show that dengue outbreak forecasts made by the TM are more accurate than those obtained by a Support Vector Machine (SVM), Decision Trees (DTs), and several multi-layered Artificial Neural Networks (ANNs), both in terms of forecasting precision and F1-score.*

A.1 Introduction

The Tsetlin Machine (TM) is a recent pattern classification method that manipulates expressions in propositional logic based on a team of Tsetlin Automata (TAs) [1]. A Tsetlin Automaton (TA) is a fixed structure deterministic automaton that learns the optimal action among the set of actions offered by an environment. Figure A.1 shows a two-action TA with 2N states. The action that the TA performs next is decided by the present state of the TA. States from 1 to N maps to Action 1, while states from N+1 to 2N maps to Action 2. The TA interacts with its environment in an iterative way. In each iteration, the TA performs the action associated with its current state. This, in turn,

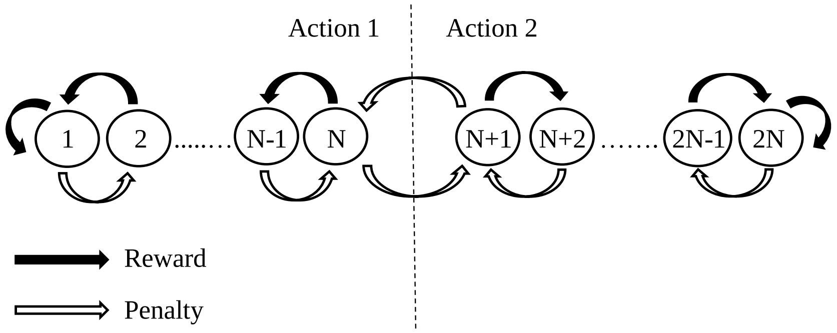


Figure A.1: Transition graph of a two-action Tsetlin Automaton.

randomly triggers a reward or a penalty from the environment, according to an unknown probability distribution. If the TA receives a reward, it reinforces the action performed by moving to a “deeper” state, one step closer to one of the ends (left or right side). If the action results in a penalty, the TA moves one step towards the middle state, to weaken the performed action, ultimately jumping to the middle state of the other action. In this manner, with a sufficient number of states, a TA converges to performing the action with the highest probability of producing rewards – the optimal action – with probability arbitrarily close to unity, merely by interacting with the environment[2].

The TM, introduced in 2018 by Granmo [1], uses the TA as a building block to solve complex pattern recognition tasks. The TM operates as follows. Firstly, propositional formulas in disjunctive normal form are used to represent patterns. The TM is thus a general function approximator. The propositional formulas are learned through training on labelled data by employing a collective of TAs organized in a game. The payoff matrix of the game has been designed so that the Nash equilibria (NE) correspond to the optimal configurations of the TM. As a result, the architecture of the TM is relatively simple, facilitating transparency and interpretation of both learning and classification. Additionally, the TM is designed for bit-wise operation. That is, it takes bits as input and uses fast bit manipulation operators for both learning and classification. This gives the TM an inherent computational advantage. Experimental results show that TM outperforms ANNs, Support Vector Machines (SVMs), the Naïve Bayes Classifier (NBC), Random Forests (RF), and Logistic Regression (LR) in diverse benchmarks [1, 3]. These promising properties and results make the TM an interesting target for further research.

In this paper, we introduce a novel scheme that improves the accuracy of the TM when features are continuous. In Section 2, we provide an overview of related work. Then, in Section 3, we present our scheme for handling continuous features. In all brevity, we encode continuous features in binary form based on thresholding. The behavior of the resulting TM is studied in Section 4 based on both an artificial dataset and real-life data, focusing on dengue fever forecasting. Section 5 summarizes our research and provides pointers for further work.

A.2 Related Work

Propositional logic is a well-explored framework for knowledge based pattern classification. In [4], Disjunctive Normal Form (DNF) is used to represent the patterns in clinical and genomic data to find the recurrence of liver cancer. Data is converted to bits by setting thresholds for continuous features. Based on the input features, logical functions for recurrence and non-recurrence are created. Another example is the use of Boolean expressions to capture visual primitives for visual recognition, rather than relying on a data driven approach [5]. In all brevity, the advantage of propositional logic for pattern classification, and knowledge based approaches in general, as opposed to data driven statistical models, is that patterns can be identified even without a single training sample.

Learning propositional formulas to represent patterns in data has a long history [6]. Feldman investigates the hardness of learning DNF [7], Klivans use Polynomial Threshold Functions to build logical expressions [8], while Feldman leverages Fourier analysis[9]. Furthermore, so-called Probably Approximately Correct (PAC) learning has provided fundamental insight into machine learning, as well as providing a framework for learning formulas in DNF [10]. An integer programming approach is applied in [11] to learn disjunctions of conjunctions, providing promising results based on a Bayesian method. In addition to the above techniques, association rule mining models have been extensively applied in [12, 13] to predict sequential events using set of rules. Recent approaches combine Bayesian reasoning with propositional formulas in DNF for robust learning of formulas from data [6]. However, these techniques still suffer when facing noisy non-linear data, which may trap the learning mechanisms in local optima.

An attractive property of TA is that they support online learning in particularly noisy environment. Over several decades the basic TA, shown in Figure A.1, has been extended in several directions. These extensions include the Hierarchy of Twofold Resource Allocation Automata (H-TRAA) for resource allocation [14] and the stochastic searching on the line algorithm by Oommen et al. [15]. Furthermore, teams of Tsetlin Automata have been used to create a distributed coordination system [16], to solve the graph coloring problem [17], and to forecast dengue outbreaks in the Philippines [18]. The TM is a recent addition to the field of TA, addressing complex pattern recognition.

In order to attack the problem of forecasting, this paper introduces a preprocessing method that extends the TM so that it can handle continuous input. To achieve this, we convert continuous input into a binary representation based on thresholding. We use an artificial dataset as well as a real-life dataset to evaluate this approach, namely, forecasting of dengue fever outbreaks in the Philippines.

Different techniques have already been applied to forecast dengue outbreaks in different regions of the world. For instance, Seasonal Autoregressive Integrated Moving Average model is applied to forecast future dengue incidences in Guadeloupe [19] and Bangladesh [20]. In their research, temperature is identified as the best weather parameter to improve the forecasting performances. Here, 1-month ahead forecasting produces the highest accuracy, compared to 3-months and 1-year ahead forecasting. Similarly, dengue incidences in Rio de Janeiro, Brazil [21], Northeastern Thailand [22], and Southern Thailand [23] are forecasted using Auto-regressive Integrated Moving Average models. Phung et al. investigate the forecasting ability of three regression models on dengue fever incidences in Can Tho city in Vietnam [24]. They find that a Standard Multiple Regression model provides poor forecasting capability. However, the Poisson Distributed Lag model performs well in 12-months ahead forecasting and Seasonal Autoregressive Integrated Moving Average model performs well in 3-months ahead forecasting. The importance of utilizing data from neighboring regions to forecast dengue incidences is identified in [25], using an Artificial Neural Network as the forecasting model with data from the Philippines.

In contrast to the above approaches, we will here investigate whether a rule based approach, based on the Tsetlin Machine, can forecast outbreaks surpassing a decision threshold, across the regions of the Philippines.

A.3 Methodology

A.3.1 The Tsetlin Machine Architecture

The TM addresses pattern classification problems where a class can be represented by a collection of sub-patterns, each fixing certain features to distinct values. The TM is designed to uncover these sub-patterns in an effective, yet relatively simple manner. In all brevity, the TM represents a class using a series of clauses. Each clause, in turn, captures a sub-pattern by means of a conjunction of literals, where a literal is a propositional variable or its negation. Each propositional variable takes the value *False* or *True* (in bit form, 0 or 1 respectively).

Let $\boldsymbol{X} = [x_1, x_2, x_3, \ldots, x_n]$ be a feature vector consisting of n propositional variables x_k with domain $\{0, 1\}$. Now suppose the pattern classification problem involves q outputs, and m sub-patterns per output that we need to recognize. Then the resulting pattern classification problem can be captured using $q \times m$ conjunctive clauses C_i^j, $1 \leq j \leq q$, $1 \leq i \leq m$. The output y^j, $1 \leq j \leq q$, of the classifier is given as:

$$C_i^j = 1 \wedge \left(\bigwedge_{k \in I_i^j} x_k \right) \wedge \left(\bigwedge_{k \in \bar{I}_i^j} \neg x_k \right). \tag{A.1}$$

$$y^j = \bigvee\nolimits_{i=1}^{m} C_i^j. \tag{A.2}$$

Above, I_i^j and $\bar{I}_i^j$ are non-overlapping subsets of the input variable indexes, $I_i^j, \bar{I}_i^j \subseteq \{1, ...n\}$, $I_i^j \cap \bar{I}_i^j = \emptyset$. The subsets decide which of the propositional variables take part in the clause, and whether they are negated or not.

In the TM, the disjunction operator is replaced by a summation operator to increase classification robustness [1]. The structure of the multiclass TM is depicted in Figure A.2b. The sub-figures in Figure A.2 illustrate the three phases of the classification process, i.e., (a) how a team of TAs forms a clause that processes the input features; (b) how a TM is composed by multiple TA teams; and (c) how a group of TMs are connected to handle multiclass classification problems. We will now detail these phases one by one.

The TA team:

Inputs and literals. The TM takes n propositional variables $x_1, x_2, x_3, \ldots, x_n$ as input. For each variable x_k, there are two literals, the variable itself and its negation $\neg x_k$.

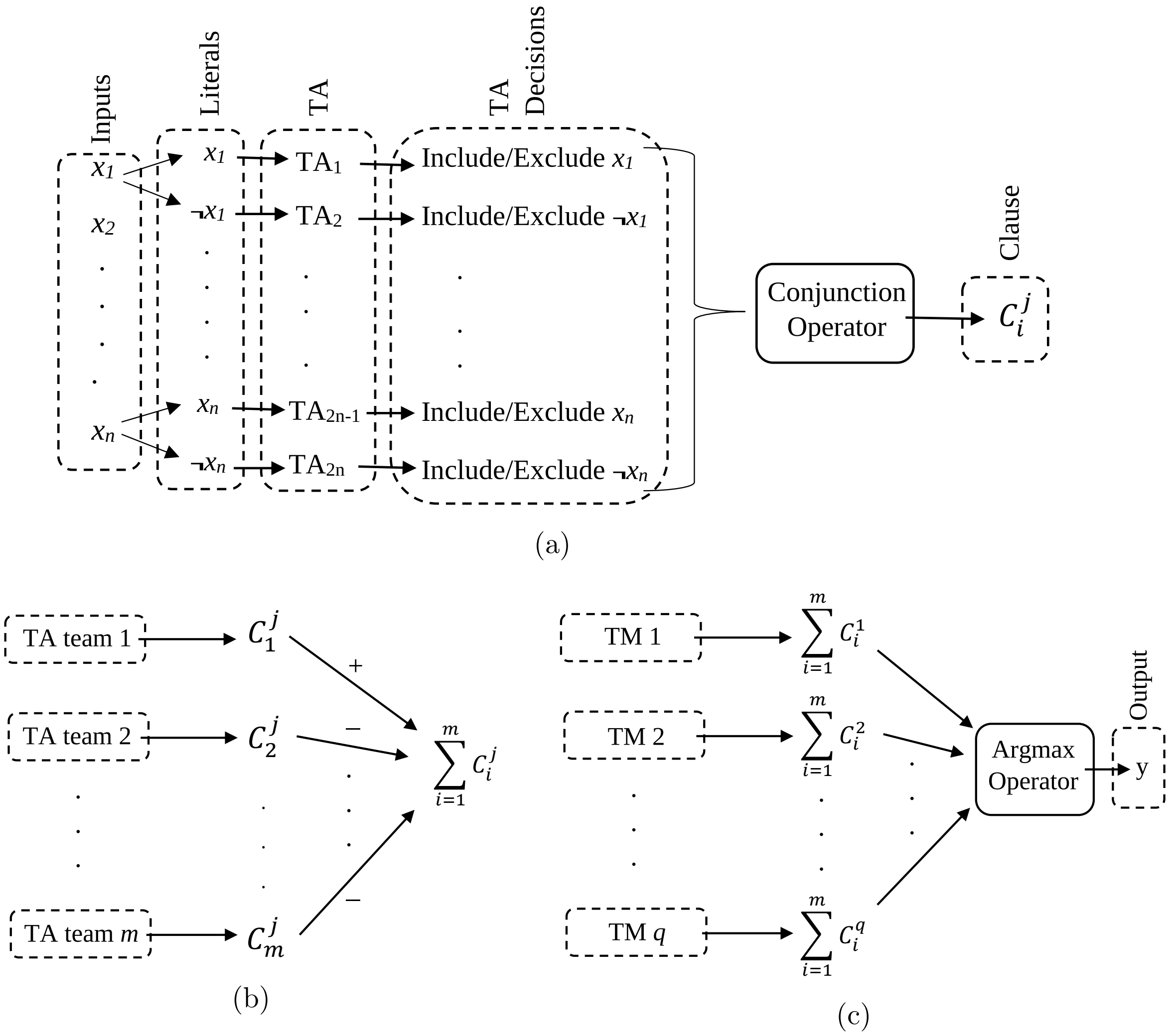


Figure A.2: (a) A TA team forms the clause C_i^j, $1 \leq j \leq q$, $1 \leq i \leq m$. (b) A TM. (c) A multiclass TM.

Tsetlin Automata and their decisions. For each clause C_i^j, each literal is assigned a unique TA. This TA decides whether to *include* or *exclude* its assigned literal in the given clause. Thus, for n input variables, we need $2n$ TA. This collective of TA is called a team. The TA team composes a conjunction of the literals that the team has chosen to be included. The conjunction outputs 1 if all of the included literals evaluate to 1, otherwise, the clause outputs 0.

The TM:

Clauses and their role in a TM. A TM consists of m clauses, each associated with a TA team. The number of clauses needed for a particular class depends on the number of subpatterns associated with the class. Each clause casts a vote, so that the m clauses jointly decide the output of the TM. Clauses with odd indexes are assigned positive polarity (+) and clauses with even indexes are assigned negative polarity (−). The summation operator aggregates the votes by subtracting the number of negative votes from the number of positive votes.

Note that clauses with positive polarity cast their votes to favor the decision that the input belongs to the class represented by the TM, whereas clauses with negative polarity vote for the input belonging to one of the other classes.

The multiclass TM:

Obtaining the final output. With multiple TMs we get a multiclass TM. As shown in Figure A.2c, the final decision is made by the argmax operator to classify the input data to the class that obtained the highest vote sum.

A.3.2 The TA Game and Orchestration Scheme

We organize learning in the TM as a game being played among the TAs. The Nash Equilibria of the game corresponds to the goal state of the TA, providing the final classifier. In the worst case, the single action of any TA has the power to disrupt the whole game. Therefore, the TAs must be guided carefully towards optimal pattern recognition.

To achieve this, the Tsetlin Machine is built around two kinds of feedback: Type I and Type II feedback. The Reward, Inaction, and Penalty probabilities under these two feedback types are summarized in Table A.1, and they are determined based on the clause output (1 or 0), the literal value (1 or 0), and the current action of the TA (include or exclude). Rewards and Penalties are fed to the TA as normal. Inaction means that the state of the TA remains unchanged.

The training process of the TM thus contains several interacting mechanisms. To clarify their roles, we provide a flow chart for the complete procedure, shown in Figure 3, and explored in the following.

Type I Feedback. As seen in the flowchart, briefly stated, Type I feedback is only activated when the actual output $\hat{y}$ is 1. When the output of the target clause also is 1, Type I Feedback has three roles:

- It reinforces true positive output by assigning a large reward probability $\frac{s-1}{s}$ to the action of including literals that evaluate to 1, and thus contributing to the result of the clause output being 1.

- Conversely, exclude actions are penalized with the same magnitude under these conditions. This is to "tighten" the clause, because it would still output 1 also when the literal considered is included instead.

- Furthermore, if the value of the literal is 0, excluding the literal is the way to go, and exclude actions are thus rewarded with probability $\frac{1}{s}$.

When the output of the clause is 0 (false negative output), Type I feedback has the following effect:

- Type I feedback systematically penalizes include actions with probability $\frac{1}{s}$. Indeed, excluding literals is the only way to invert the output of a clause that outputs 0.

- When the action is exclude, this is rewarded with probability $\frac{1}{s}$, because reinforcing exclude actions will sooner or later invert the output of the clause to 1.

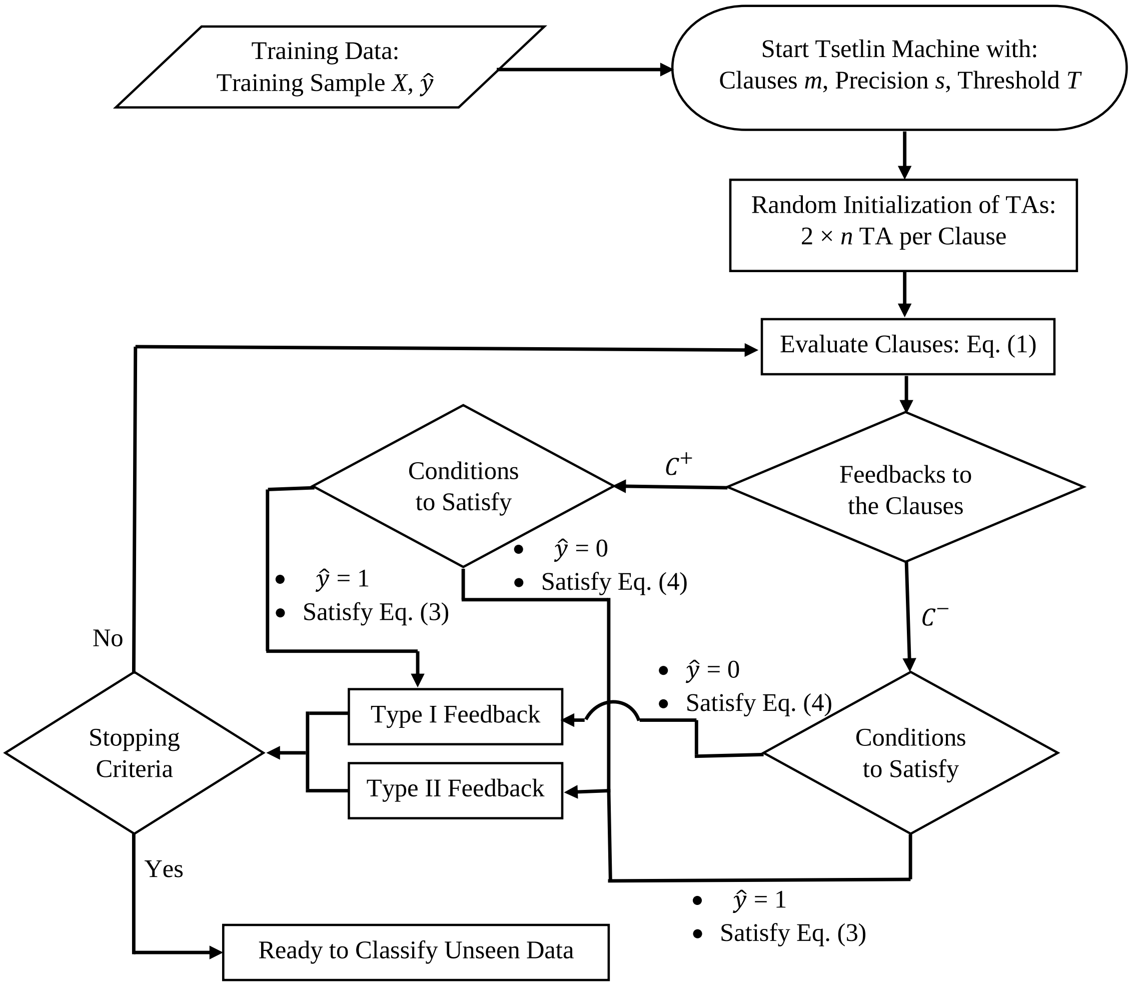


Figure A.3: The training work-flow.

Thus, eventually, Type I feedback combats false negative output, and encourages true positive output.

Type II Feedback. Type II feedback is activated when the actual output $\hat{y}$ is 0, as shown in the flowchart. This type of feedback is designed to eliminate false positive output. That is, when the clause output should be 0, but the clause erroneously evaluates to 1, Type II feedback is triggered. In brief, repeated Type II feedback forces in the end the offending clause to evaluate to 0, simply by including a literal that has the value 0 into the clause (which makes the conjunction of literals evaluate to 0 as well). This is achieved by penalizing, with probability 1, exclude actions for literals that evaluate to 0.

To summarize, Type I Feedback reinforces true positive output, while simultaneously reducing false negative output. These dynamics are countered by Type II Feedback, which systematically reduces false positive output.

The Clause Feedback Activation Function. In [1], an additional feedback mechanism is introduced, aiming at allocating the sparse pattern representation resources provided by the clauses as effectively as possible. This is achieved by introducing a target value T for the number of clauses voting from a specific pattern. The idea is to gradually reduce the frequency of feedback for a specific pattern, as the number of votes approaches T. In all brevity, the feedback activation function is basically an activation probability

Table A.1: Type I and Type II feedback to battle against false negatives and false positives.

Feedback Type			I				II			
Clause Output			1		0		1		0	
Literal Value			1	0	1	0	1	0	1	0
Current State	Include (Probability)	Reward	(s-1)/s	NA	0	0	0	NA	0	0
		Inaction	1/s	NA	(s-1)/s	(s-1)/s	1	NA	1	1
		Penalty	0	NA	1/s	1/s	0	NA	0	0
	Exclude (Probability)	Reward	0	1/s	1/s	1/s	0	0	0	0
		Inaction	1/s	(s-1)/s	(s-1)/s	(s-1)/s	1	0	1	1
		Penalty	(s-1)/s	0	0	0	0	1	0	0

*s is the precision and controls the granularity of the sub-patterns in [1]

controlled by a Threshold T. The probability of activating Type I Feedback for a specific clause is:

$$\frac{T - max(-T, min(T, \sum_{i=1}^{m} C_i^j))}{2T} \tag{A.3}$$

For Type II Feedback, the probability is:

$$\frac{T + max(-T, min(T, \sum_{i=1}^{m} C_i^j))}{2T} \tag{A.4}$$

As seen, for Eq. (A.3), the activation probability decreases as the number of votes approaches T, and finally when T is reached, the probability becomes 0. Thus ultimately, Type I feedback will not be activated when enough clauses are producing the correct number of votes. This in turn "freezes" the affected clauses since TAs will no longer change state. The crucial point here is that this frees other clauses to seek other sub-patterns, because the "frozen" pattern is no longer attractive for the TA. The same rationale holds for Eq. (A.4) for Type II feedback. In this way, the pattern representation resources can be allocated more effectively.

A.3.3 Data Pre-Processing

We now come to one of the main contributions of this paper, namely a scheme that allows the Tsetlin Machine to successfully recognize patterns consisting of continuous features, despite being constrained to an internal binary representation. As the TM only takes binary variables as input, we transform continuous features into binary form in a preprocessing step, detailed in the following.

Table A.2 illustrates the transformation procedure, using one continuous feature as an example. The same procedure is repeated for each continuous feature in turn. First of all, all the unique values $\{v_1, v_2, \ldots, v_u\}$ of the continuous feature found in the dataset are identified. We consider each unique value v_w to be a potential threshold "$\leq v_w$". Thus each unique value provides a new derived binary feature: is the threshold condition fulfilled or not fulfilled for a particular continuous value v.

Table A.2: Conversion of original input features into bits.

Raw Data	Thresholds		
	≤ 3.834	≤ 5.779	≤ 10.008
5.779	0	1	1
10.008	0	0	1
5.779	0	1	1
3.834	1	1	1

As an example, column 1 in Table A.2 contains the values of the continues features. As seen, there are three unique values, and these provides three thresholds $\leq$ 3.834, $\leq$ 5.779, and $\leq$ 10.008. Accordingly, three new binary features are introduced, encoding the original raw continuous values, also shown in the table. If the raw continuous value is greater than the threshold, the corresponding bit in the binary form is assigned the value 0; and if the raw continuous value is less than or equal to the threshold, it is given the value 1. For example, in Table A.2, for the first value 5.779, it is greater than the first threshold 3.834, so the corresponding binary feature is assigned the value 0 (in column 2). However, being equal to the threshold value of the second threshold 5.779, and less than the third threshold value 10.008, both column 3 and column 4 are assigned the value 1. Therefore, the final binary bits that represent 5.779 becomes 011. Similarly, 10.008 and 3.834 are represented by 001 and 111, respectively.

This new representation becomes particularly powerful due to the capability of the TM to negate features, allowing a clause to specify intervals for continuous features. In the following section, we evaluate this procedure both on an artificial dataset, as well as for the real-life application of forecasting dengue fever outbreaks in the Philippines.

A.4 Experiments

First, the behavior of the TM is studied using an artificial dataset. Actions chosen by TAs in clauses, clause outputs, and TM outputs, are extensively studied with this dataset. Then, the TM is applied to forecast the dengue outbreaks in the Philippines. Data and the TM preparation for both tasks are discussed in the following subsections.

A.4.1 Behavior in Dealing with Artificial Data

A.4.1.1 Experimental Setup

The dataset consists of two inputs (integers, $0 \leq x_1 \leq 4$ and $0 \leq x_2 \leq 5$). If the sum of the inputs is equal to 9, they are assigned class 1 and the rest is assigned class 0. Since the features in this process are categorical, we use one-hot-encoding to convert them into bits instead of the procedure proposed in the previous section. Input x_1 takes one of five values (0, 1, 2, 3, 4) and input x_2 takes one of six values (0, 1, 2, 3, 4, 5). Therefore, these two features can be expressed using 5 and 6 bits, respectively. An example data sample converted to bits can be found in Table A.3.

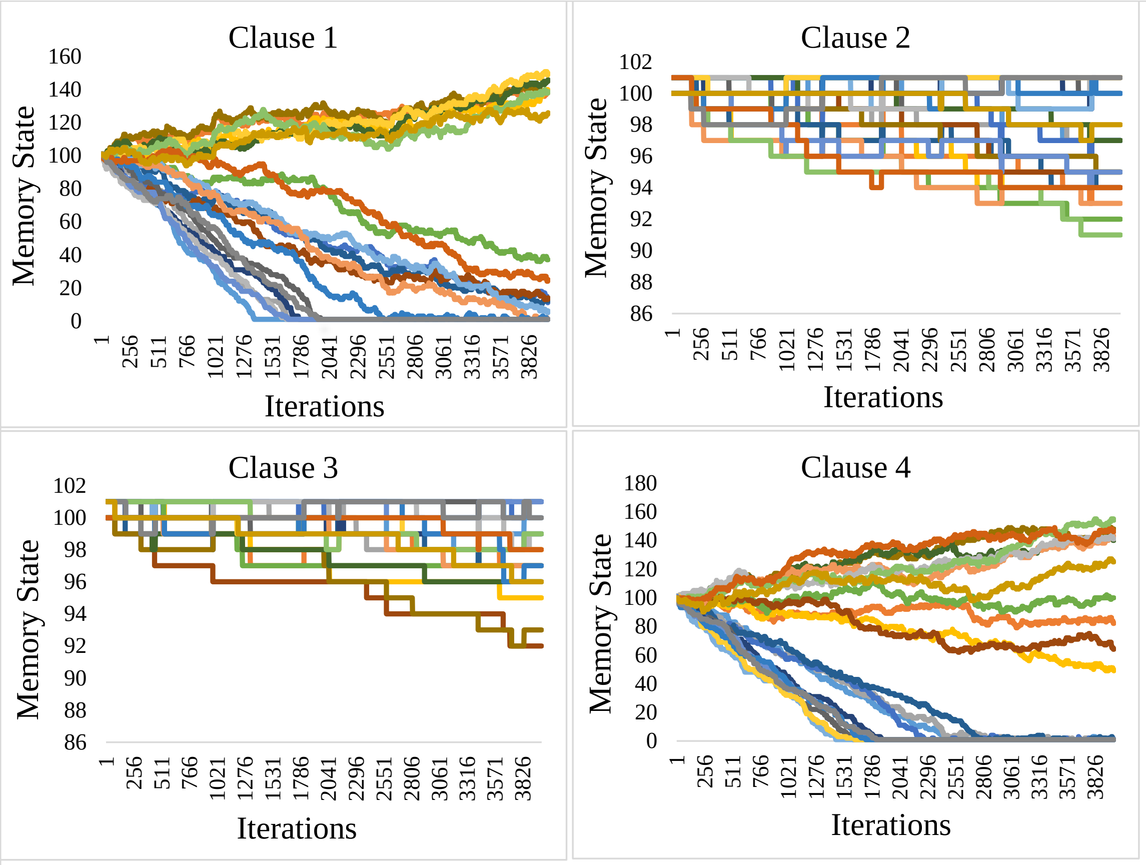


Figure A.4: Variation of actions of TA to classify artificial data.

A Tsetlin Machine with 4 clauses is used to classify the artificial data. Since there are 11 input bits, 22 TAs are needed to form a clause. Each TA is given 100 states per action. Two of those four clauses will vote in favour of class 0. The other two clauses will vote in favour of class 1. The two remaining hyper parameters: Threshold and Precision are set to 1 and 8, respectively.

A.4.1.2 Behavior Analysis

During the training process, the states of the TAs in each clause are recorded and plotted in Figure A.4. Clause 1 and 3 have positive polarities and clause 2 and 4 have negative polarities. Clause 1 and 4 vote in favour of class 0 while clause 2 and 3 vote in favour of class 1.

The change in states of the TAs in clause 1 and 4 are more dynamic compared to the TAs in clauses 2 and 3. This is due to the much larger number of training samples that

Table A.3: Converting integer training samples to bits.

Original Sample	x_1					x_2						Out
	3					5						8
Bit Positions	0	1	2	3	4	0	1	2	3	4	5	
Sample in Bits	0	0	0	1	0	0	0	0	0	0	1	0

Table A.4: Regions that provide their data to forecast dengue incidences of their neighbors

Target	**Selected Regions**	**Target**	**Selected Regions**
I	II,III, IVA, XIV, Total	**IX**	X, XI, XII
II	I, III, IVA, XIV, Total	**X**	IX, XI, XII, XIV, XV
III	I, II, IVA, XVI	**XI**	IX, X, XII, XV
IVA	III, IVB, V, XVI, Total	**XII**	IX, X, XI, XV, Total
IVB	II, IVA, VI, Total	**XIII**	IX, X, XII, XV, Total
V	IVA,VI, Total	**XIV**	I, II, III, IVA, IVB, Total
VI	IVB, V, VII, XII,Total	**XV**	IX, X, XI, XII
XII	IVA, V, VI, XII,Total	**XVI**	I, III, IVA, V
VIII	V, VI		

belong to class 0. Since more training samples belong to class 0 ($\sim$ 8/9 of the data) than class 1 ($\sim$ 1/9 of the data), clauses 1 and 4 receive feedback more frequently. As seen, the TAs in clauses 2 and 3 move slowly towards the *exclude* action (memory states from 0 to 100) since they receive Type II feedback more often (from class 0 data). However, once the TM is trained, it can classify all the 200 testing samples with 100% accuracy.

A.4.2 Predicting Disease Outbreaks

A.4.2.1 Experimental Setup

The number of patients who suffer from dengue haemorrhagic fever or dengue shock syndrome has been increasing over the years. Therefore, dengue haemorrhagic fever is considered as an important public health issue, especially in tropical and subtropical countries. To control the mortality rate due to dengue fever, an early warning system, which helps directly on emergency preparedness and resource planning, is called for [24].

The Philippines has 17 administrative regions (from I to XVI with two IV regions; IVA and IVB). Department of Health in the Philippines has collected the number of monthly dengue incidences separately for all these regions from 2008 to 2016.

The number of monthly dengue incidences in most of the regions has been growing from 2008 and peaked in 2013. However, from 2013, the number of incidences has dropped to reach an average value of 9.22 patients per 100,000 population. Considering these trends, we decided to use more than 20 monthly dengue incidences per 100,000 population as an indication of outbreak. Using the data from 2008 to 2015, dengue outbreaks in the months of 2016 are to be predicted for all the regions.

In addition to the dengue incidences in the previous-month and previous-year-same-month of the same region, historical dengue incidences from the neighboring regions and total dengue incidences are considered as input features to forecast the dengue outbreaks. These regions and total dengue incidences are selected based on their correlation to the target series. Dengue incidences in the previous-month of the selected regions and total dengue incidences are used as input features. The selected regions to forecast each region are summarized in Table A.4.

Table A.5: Summary of the forecasting outcomes by different models.

	TM	ANN-1	ANN-2	ANN-3	ANN-4	SVM	DT
Precision	0.44±0.02	0.35±0.02	0.39±0.01	0.37±0.02	0.36±0.02	0.43±0.01	0.29±0.02
Recall	0.37±0.02	0.23±0.02	0.31±0.02	0.36±0.02	0.33±0.03	0.14±0.01	0.41±0.02
F1	0.40±0.01	0.28±0.02	0.34±0.02	0.36±0.02	0.34±0.03	0.21±0.01	0.34±0.01
Accuracy	0.88±0.01	0.87±0.01	0.87±0.01	0.87±0.02	0.87±0.01	0.89±0.01	0.83±0.01

Once the input features are determined, they are converted to bits using the procedure proposed in Section 3. Then they are fed into the TM to predicts dengue outbreaks in each region separately. Each TM has 2000 clauses and the associated TAs are given 100 states per action. The other two hyper parameters, Threshold and Precision, are set to 15 and 8, respectively.

A.4.2.2 Results

Possible dengue outbreaks in the Philippines for the year 2016 are forecasted by the TM. Results from the TM are compared with results from three other machine learning techniques: ANNs, a SVM, and a Decision Tree (DT). For comprehensiveness, four ANN architectures are used to forecast the dengue outbreaks: ANN-1 – one hidden layer with 5 neurons, ANN-2 – one hidden layer with 20 neurons, ANN-3 – three hidden layers with 20, 150, and 100 neurons, respectively, and ANN-4 – five hidden layers with 20, 200, 150, 100, and 50 neurons. The SVM uses a Radial Basis Function kernel to capture the non-linear patterns in the data. The regularization parameter (C) in this case is fixed at 1.0 with $gamma$ = 1/(the number of input features) to maximize prediction accuracy. The parameters which decide the quality of the DT output, such as maximum tree depth (=max), minimum number of samples required for split (=2), and the minimum number of samples required for a leaf node (=1) are again all adjusted to optimize prediction accuracy. Since there are 17 regions, all of the 204 testing samples are utilized to test the accuracy of each technique. These samples encompass 22 outbreaks to be identified. Each model is executed using 30-fold cross-validation to calculate precision, recall, F1-score, and accuracy. The means and the 95% confidence intervals of these scores can be found in Table A.5.

The TM obtains the highest mean values for precision and F1-score. The second highest precision (0.43) is obtained by the SVM, at the sacrifice of a much lower recall. Conversely, DT produces the highest recall, however, precision suffers. Considering overall performance, captured by the F1 score, the TM obtains the highest mean F1-score (0.40) while ANN-3 obtains the second highest mean F1-score (0.36). Even though mean F1 score peaks at 0.36, as a result of increasing the structural complexity of the ANNs, the score drops again when complexity is increased further. Due to the imbalance of the dataset (182 non-outbreaks and 22 outbreaks), the SVM produces a particularly high accuracy (0.89) by mostly classifying instances as non-outbreaks. Finally, note that both the mean values of precision, recall, F1-score, and accuracy of the TM are higher than what we were able to achieve with the ANN models.

A.5 Conclusion

In this paper, we proposed a feature pre-processing procedure for the TM so that it can effectively handle continuous input features. This opens up for promising applications in e.g. forecasting, where continuous features are typical. We applied the resulting TM approach to forecast dengue outbreaks in the Philippines, after performing an empirical study on an artificial dataset. While the experiments with the artificial dataset confirmed the desired properties of the new scheme, the results on the real-life dataset further demonstrated competitive performance also with respect to other machine learning approaches. Indeed, it turned out that the TM is more accurate than the evaluated SVMs, Decision Trees, and several multi-layered ANNs, both in terms of forecasting precision and F1-score.

In our further work, we intend to exploit this approach also in other pattern recognition domains where continuous features are dominant. We further intend to investigate how also the output of the TM can be rendered continuous.

Bibliography

[1] Ole-Christoffer Granmo. “The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic”. In: *arXiv preprint arXiv:1804.01508* (2018).

[2] Kumpati S Narendra and Mandayam AL Thathachar. *Learning Automata: An Introduction*. Courier corporation, 2012.

[3] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. “Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications”. In: *IEEE Access* 7 (2019), pp. 115134–115146.

[4] Hiroyuki Ogihara, Yusuke Fujita, Yoshihiko Hamamoto, Norio Iizuka, and Masaaki Oka. “Classification Based on Boolean Algebra and Its Application to the Prediction of Recurrence of Liver Cancer”. In: *Pattern Recognition (ACPR), 2013 2nd IAPR Asian Conference on*. IEEE. 2013, pp. 838–841.

[5] Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. “Neural Algebra of Classifiers”. In: *arXiv preprint arXiv:1801.08676* (2018).

[6] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. “A Bayesian Framework for Learning Rule Sets for Interpretable Classification”. In: *The Journal of Machine Learning Research (JMLR)* 18.1 (2017), pp. 2357–2393.

[7] Vitaly Feldman. “Hardness of Approximate Two-Level Logic Minimization and PAC Learning with Membership Queries”. In: *Jrnl. of Computer and System Sciences* 75.1 (2009), pp. 13–26.

[8] Adam R Klivans and Rocco A Servedio. “Learning DNF in Time 2O (n1/3)”. In: *Journal of Computer and System Sciences* 68.2 (2004), pp. 303–318.

[9] Vitaly Feldman. “Learning DNF Expressions From Fourier Spectrum”. In: *Conference on Learning Theory*. 2012, pp. 17–1.

[10] Leslie G Valiant. “A Theory of the Learnable”. In: *Communications of the ACM* 27.11 (1984), pp. 1134–1142.

[11] John R Hauser, Olivier Toubia, Theodoros Evgeniou, Rene Befurt, and Daria Dzyabura. “Disjunctions of Conjunctions, Cognitive Simplicity, and Consideration Sets”. In: *Jrnl. of Marketing Research* 47.3 (2010), pp. 485–496.

[12] Cynthia Rudin, Benjamin Letham, and David Madigan. "Learning Theory Analysis for Association Rules and Sequential Event Prediction". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 3441–3492.

[13] Tyler McCormick, Cynthia Rudin, and David Madigan. "A Hierarchical Model for Association Rule Mining of Sequential Events: An Approach to Automated Medical Symptom Prediction". In: *Annals of Applied Statistics* (2011).

[14] Ole-Christoffer Granmo and B John Oommen. "Solving Stochastic Nonlinear Resource Allocation Problems Using a Hierarchy of Twofold Resource Allocation Automata." In: *IEEE Transaction on Computers* (2010).

[15] B John Oommen, Sang-Woon Kim, Mathew T Samuel, and Ole-Christoffer Granmo. "A Solution to the Stochastic Point Location Problem in Metalevel Nonstationary Environments". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.2 (2008), pp. 466–476.

[16] Brian Tung and Leonard Kleinrock. "Using Finite State Automata to Produce Self-Optimization and Self-Control". In: *IEEE transactions on parallel and distributed systems* 7.4 (1996), pp. 439–448.

[17] Noureddine Bouhmala and Ole-Christoffer Granmo. "Stochastic Learning for SAT-Encoded Graph Coloring Problems". In: *International Journal of Applied Metaheuristic Computing (IJAMC)* 1.3 (2010), pp. 1–19.

[18] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "A Novel Tsetlin Automata Scheme to Forecast Dengue Outbreaks in the Philippines". In: *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2018, pp. 680–685.

[19] Myriam Gharbi, Philippe Quenel, Joël Gustave, Sylvie Cassadou, Guy La Ruche, Laurent Girdary, and Laurence Marrama. "Time Series Analysis of Dengue Incidence in Guadeloupe, French West Indies: Forecasting Models Using Climate Variables as Predictors". In: *BMC infectious diseases* 11.1 (2011), p. 166.

[20] Zamil MAH Choudhury, Shahera Banu, and Amirul M Islam. "Forecasting Dengue Incidence in Dhaka, Bangladesh: A Time Series Analysis." In: *Dengue Bulletin* 32 (2008).

[21] Paula M Luz, Beatriz VM Mendes, Claudia T Codeço, Claudio J Struchiner, and Alison P Galvani. "Time Series Analysis of Dengue Incidence in Rio de Janeiro, Brazil". In: *The American journal of tropical medicine and hygiene* 79.6 (2008), pp. 933–939.

[22] Tassanee Silawan, Pratap Singhasivanon, Jaranit Kaewkungwal, Suchitra Nimmanitya, and Wanapa Suwonkerd. "Temporal Patterns and Forecast of Dengue Infection in Northeastern Thailand". In: *Southeast Asian Journal of Tropical Medicine and Public Health* 39.1 (2008), p. 90.

[23] S Promprou, M Jaroensutasinee, and K Jaroensutasinee. "Forecasting Dengue Haemorrhagic Fever Cases in Southern Thailand Using ARIMA Models." In: *Dengue Bulletin* 30 (2006).

[24] Dung Phung, Cunrui Huang, Shannon Rutherford, Cordia Chu, Xiaoming Wang, Minh Nguyen, Nga Huy Nguyen, and Cuong Do Manh. "Identification of the Prediction Model for Dengue Incidence in Can Tho City, a Mekong Delta Area in Vietnam". In: *Acta tropica* 141 (2015), pp. 88–96.

[25] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "Effect of Data From Neighbouring Regions to Forecast Dengue Incidences in Different Regions of Philippines Using Artificial Neural Networks". In: *2018: Norsk Informatikkonferanse* (2018).

Paper B

Adaptive Continuous Feature Binarization for Tsetlin Machines Applied to Forecasting Dengue Incidences in the Philippines

The Tsetlin Machine *(TM) is a recent interpretable machine learning algorithm that requires relatively modest computational power, yet attains competitive accuracy in several benchmarks. TMs are inherently binary; however, many machine learning problems are continuous. While binarization of continuous data through brute-force thresholding has yielded promising accuracy, such an approach is computationally expensive and hinders extrapolation. In this paper, we address these limitations by standardizing features to support scale shifts in the transition from training data to real-world operation, typical for e.g. forecasting. For scalability, we employ sampling to reduce the number of binarization thresholds, relying on stratification to minimize loss of accuracy. We evaluate the approach empirically using two artificial datasets before we apply the resulting TM to forecast dengue outbreaks in the Philippines using the spatio-temporal properties of the data. Our results show that the loss of accuracy due to threshold sampling is insignificant. Furthermore, the dengue outbreak forecasts made by the TM are more accurate than those obtained by Support Vector Machines (SVMs), Decision Trees (DTs), and several multi-layered Artificial Neural Networks (ANNs), both in terms of forecasting precision and F1-score.*

B.1 Introduction

In many machine learning applications, one is mainly concerned with minimizing prediction error, and tasks can be safely automated when a satisfactory accuracy can be obtained. However, when the consequences of prediction errors are sufficiently severe, human quality assurance of the predictions can be critical. Examples of such scenarios are credit scoring [1, 2], medical treatment [3, 4], bioinformatics [5, 6], and churn prediction [7, 8].

Human quality assurance requires that the models employed are interpretable [9]. One needs to know which factors were involved in making the prediction as well as the role of each factor. While deep learning approaches have shown great potential in tackling difficult non-linear pattern recognition tasks, those schemes are known to be hard to interpret. There is thus increasing interest in developing techniques that are interpretable, yet capable of dealing with complex non-linear problems.

There exist several well-established interpretable machine learning techniques, such as Linear Regression, Logistic Regression, Decision Trees, Random Forest, and Decision Rules. However, in general, these models are relatively simplistic in comparison with deep learning approaches, leading to lower prediction accuracy.

One exception is arguably the recently introduced Tsetlin Machine (TM) [10], which has provided competitive accuracy in several challenging pattern recognition benchmarks. It maintains interpretability by building patterns as conjunctive clauses in propositional logic, in a manner that produces frequent patterns with high discrimination power. By controlling the number of conjunctive clauses employed, the TM yields a trade-off between interpretability and accuracy.

TMs have, for instance, outperformed Multilayer Perceptron architectures, Support Vector Machines, Logistic Regression, and Naïve Bayes on handwritten digits classification (MNIST), Iris data classification, and classification of Noisy XOR data with non-informative features [10]. Additionally, in natural language text classification, such as classification of electronic health records and IMDb movie reviews, TMs are competitive with vanilla deep learning techniques, including Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) Neural Networks [11].

Brute-force thresholding: A TM requires binary input, however, by employing thresholding one can convert continuous features into a binary feature matrix [12]. First, the unique feature values are identified, feature by feature. These values are used as thresholds for the respective features. Secondly, each continuous feature value, one input at a time, is compared against each threshold, forming a binary representation. If the continuous value is smaller than or equal to the threshold, the associated binary feature becomes “1”. Conversely, if the continuous value is larger than the threshold, the corresponding binary feature becomes “0”. In this way, each continuous feature value is converted into several binary feature values, one per threshold.

Research problem: Although the above proposed scheme accurately maps a continuous domain into a set of binary values, it has two main weaknesses:

- **Limited ability to extrapolate.** Accuracy drops when there is a feature scale shift in the transition from training data to testing data (real-world operation). That is, the static thresholds impede feature extrapolation.
- **High computational cost.** Some applications produce huge amounts of data, which can lead to a large number of unique values per feature. This, in turn, translates to increased memory usage and computation time to convert the continuous features into binary form.

Solution: To overcome the above two weaknesses, in this paper, we enrich the brute-force binarization scheme by adding two mechanisms. In order to address the first issue, we standardize the continuous training features before converting them into a binary representation. The same procedure is repeated on operational data. The second issue is addressed by reducing the number of thresholds in the binarization process. Instead of considering all the unique values as thresholds, we select some of them. In all brevity, we use a modified version of *stratified sampling* to sample a representative selection of threshold values. We also investigate uniform sampling strategies for comparison purposes.

Evaluation: The above mechanisms are evaluated on two artificial datasets as well as on prediction of dengue incidences in the Philippines. The latter task consists of classifying whether the monthly dengue incidences in each administrative region is higher or lower than a predefined threshold, based on the historical patterns in the considered region and its neighbours. As baseline, the resulting TM performance is compared with widely used machine learning algorithms. Despite only using a sample of the unique feature values as thresholds, it turns out that the TM can attain competitive accuracy.

Paper organization: The rest of the paper is organized as follows. In Section 2, we detail the basics of TMs. Then, in Section 3, we present our scheme for handling continuous features. The performance of the scheme is studied in Section 4 based on both artificial and real-life data. We summarise our findings in Section 5.

B.2 Tsetlin Machines

Conceptually, a TM consists of five layers, as illustrated by Figure B.1. In this section, we first explain the role of each of these layers when performing classification. We then go into the details of TM learning.

B.2.1 Layer 1 – Input Layer

A TM takes a feature vector $\mathbf{X} \in \{0,1\}^o$ of o propositional variables $x_k \in \{0,1\}$ as input. To increase expression power, this feature vector is extended with the negation of the original features: $\mathbf{X}' = [x_1, x_2, x_3, \ldots, x_o, \neg x_1, \neg x_2, \neg x_3, \ldots, \neg x_o]$. Jointly, the elements of the extended feature vector $\mathbf{X}'$ are referred to as literals.

B.2.2 Layer 2 – Clause Layer

The clause layer processes literals from the input layer. To this end, the clause layer comprises m conjunctive clauses, which are to capture sub-patterns in the data. Each conjunctive clause j is defined by the literals it includes:

$$c_j = 1 \land \left(\bigwedge_{k \in I_j^I} x_k \right) \land \left(\bigwedge_{k \in \bar{I}_j^I} \neg x_k \right). \tag{B.1}$$

Above, the set I_j^I contains the indexes of the *original* variables that are included in clause j. Similarly, the set $\bar{I}_j^I$ consists of the indexes of the included *negated* variables. These sets are thus subsets of the complete set of indexes $I_j^I, \bar{I}_j^I \subseteq \{1, \ldots, o\}$.

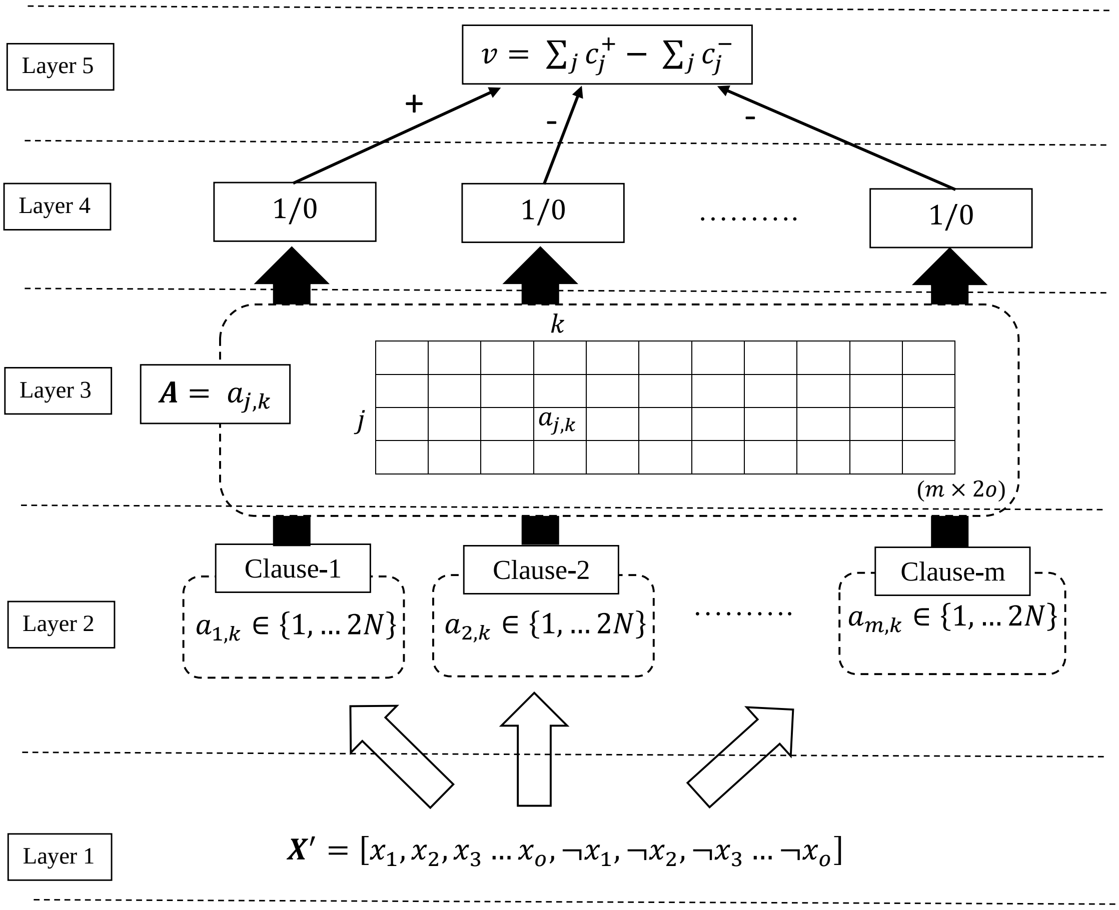

Layer 5
$v = \sum_j c_j^+ - \sum_j c_j^-$
+
-
-
Layer 4
1/0
1/0
....
1/0
Layer 3
$A = a_{j,k}$
k
j
$a_{j,k}$
$(m \times 2o)$
Layer 2
Clause-1
Clause-2
Clause-m
$a_{1,k} \in \{1, \ldots 2N\}$
$a_{2,k} \in \{1, \ldots 2N\}$
....
$a_{m,k} \in \{1, \ldots 2N\}$
Layer 1
$X' = [x_1, x_2, x_3 \ldots x_o, \neg x_1, \neg x_2, \neg x_3 \ldots \neg x_o]$

Figure B.1: The TM structure.

B.2.3 Layer 3 – Memory Layer

The decisions of including or excluding literals in clauses are made by two-action Tsetlin Automata (TAs) [13]. That is, $2 \times o$ TAs are attached to each clause, one per literal $x'_k, k \in \{1, \ldots, 2o\}$. Each TA maintains a memory state $a_{j,k} \in \{1, \ldots, 2N\}$, with j referring to the clause and k to the literal. States 1 to N map to the *exclude* action: exclude the k^{th} literal from the j^{th} clause. Conversely, states $N+1$ to $2N$ map to the *include* action: include the k^{th} literal in the j^{th} clause.

The memory layer is responsible for keeping track of all the TA states, which can be organized as a matrix $\mathbf{A}$: $\mathbf{A} = (a_{j,k}) \in \{1, \ldots, 2N\}^{m \times 2o}$. Once the states of the TAs are given, the elements in the index set I_j^I can be written as: $I_j^I = \{k | a_{j,k} > N, 1 \leq k \leq 2o\}$.

B.2.4 Layer 4 – Voting Layer

The input feature vector provides the literal values and the TA decisions compose the clauses, which can then be evaluated. Since the clauses are conjunctive, a clause evaluates to 1 if and only if all of the included literals are of value 1. Let $I_{X'}^1$ contain the indexes of the 1-valued literals from $\mathbf{X}'$. The value c_j of clause j can then be succinctly defined as:

$$c_j = \begin{cases} 1 & \text{if} \quad I_j^I \subseteq I_{X'}^1, \\ 0 & \text{otherwise.} \end{cases} \tag{B.2}$$

Algorithm 2 Clause Learning

```
    input Training example $(\mathbf{X}', y)$, voting sum $v$, clause output $c_j$, positive polarity indicator $p_j \in \{0, 1\}$, voting target $T \in [1, \infty)$, pattern specificity $s \in [1.0, \infty)$
 1: procedure UpdateClause($\mathbf{X}', v, c_j, p_j, T, s$)
 2:     $v^c \leftarrow \mathbf{clip}\,(v, -T, T)$
 3:     $e = T - v^c$ if $y_i = 1$ else $T + v_i^c$
 4:     if rand() $\leq \frac{e}{2T}$ then
 5:         if $y_i$ xor $p_j$ then
 6:             TypeIIFeedback($\mathbf{X}', c_j$)
 7:         else
 8:             TypeIFeedback($\mathbf{X}', c_j, s$)
 9:         end if
10:     end if
11: end procedure
```

We finally represent the complete collection of clause outputs in vector form: $\mathbf{C} = (c_j) \in \{0, 1\}^m$.

A two-class TM organizes the m clauses in two groups of equal size. Clauses with odd indexes are to capture the sub-patterns of class $y = 1$ and they are given positive polarity, denoted c_j^+. The clauses with even indexes, on the other hand, are to capture sub-patterns of class $y = 0$. These are given negative polarity, denoted c_j^-.

B.2.5 Layer 5 – Output Layer

The output layer receives the polarized clause outputs from the voting layer, which are aggregated into a majority vote: $v = \sum_j c_j^+ - \sum_j c_j^-$. The output of the TM is finally decided as follows:

$$y = \begin{cases} 1 & \text{if} \quad v \quad \geq \quad 0 \\ 0 & \text{if} \quad v \quad < \quad 0\,. \end{cases} \tag{B.3}$$

That is, if the negative clauses of value 1 (voting for $y = 0$) outnumber the positive clauses of value 1 (voting for $y = 1$), the final output becomes $y = 0$. Otherwise, it becomes $y = 1$.

B.2.6 Learning Procedure

Recall that a clause j is composed by its attached team of TAs and that it is TA state $a_{j,k}$ that decides whether literal x'_k is included in clause j. Learning which literals to include is based on two types of reinforcement: Type I and Type II. As described in the following, Type I feedback produces frequent patterns, while Type II feedback increases the discrimination power of the patterns.

TMs learn on-line, processing one training example $(\mathbf{X}, y)$ at a time. In all brevity, after a forward pass through the layers described above, each clause is updated according to Algorithm 2.

The first step is to decide whether the clause is to be updated (Lines 2-4). Here, resource allocation dynamics ensure that clauses distribute themselves across the frequent patterns, rather than missing some and over-concentrating on others. That is, for any input $\mathbf{X}'$, the probability of reinforcing a clause gradually drops to zero as the voting sum v approaches a user-set target T for $y = 1$ (and $-T$ for $y = 0$).

As seen, if a clause is not reinforced, it does not give feedback to its TAs, and these are thus left unchanged. In the extreme, when the voting sum v equals or exceeds the target T (the TM has successfully recognized the input $\mathbf{X}'$), no clauses are reinforced. They are then free to learn new patterns, naturally balancing the pattern representation resources [10].

If a clause is going to be updated, the updating is either of Type I or Type II (Lines 5-9):

Type I feedback is given to clauses with positive polarity when $y = 1$ and to clauses with negative polarity when $y = 0$ (Line 8). Each TA of the clause is then reinforced based on: (1) the clause output c_j; (2) the action of the TA – *include* or *exclude*; and (3) the value of the literal x'_k assigned to the TA. Two rules govern Type I feedback:

- *Include* is rewarded and *exclude* is penalized with probability $\frac{s-1}{s}$ **if** $c_j = 1$ **and** $x'_k = 1$. If this happens, the corresponding TA state $a_{j,k}$ is increased by 1, up to $2N$. This reinforcement is strong (triggered with high probability) and makes the clause remember and refine the pattern it recognizes in $\mathbf{X}'$.[1]
- *Include* is penalized and *exclude* is rewarded with probability $\frac{1}{s}$ **if** $c_j = 0$ **or** $x'_k = 0$. If this happens, the corresponding TA state $a_{j,k}$ is decreased by 1, down to 1. This reinforcement is weak (triggered with low probability) and coarsens infrequent patterns, making them frequent.

Above, hyper-parameter s controls pattern frequency.

Type II feedback is given to clauses with positive polarity when $y = 0$ and to clauses with negative polarity when $y=1$ (Line 6). It penalizes *exclude* with probability 1 **if** $c_j = 1$ **and** $x'_k = 0$. If this happens, the corresponding TA state $a_{j,k}$ is increased by 1. Thus, this feedback introduces literals for discriminating between $y = 0$ and $y = 1$.

B.3 Adaptive Binarization of Continuous Features

In this section, we present the preprocessing scheme that allows TMs to effectively handle continuous features. The scheme is based on a brute-force binarization approach that employs all unique continuous values as binarization thresholds [12, 14]. The latter scheme binarizes continuous input through five steps, examplified in Table B.1. Features are converted into binary form, one feature at a time, as follows:

1. Identify the unique values $\{n_1, n_2, \ldots, n_u\}$ of the selected feature.
2. Sort the identified unique values from smallest to largest.
3. Consider each unique value $n_i, i = 1, 2, ..., u$, as a threshold "$\leq n_i$", as shown in sorted order in the "Thresholds" row in the table.

[1] Note that the probability $\frac{s-1}{s}$ is replaced by 1 when boosting true positives.

Table B.1: Binarization of two continuous features.

Raw Feature		Thresholds						
1	2	≤ 3.834	≤ 5.779	≤ 10.008	≤ 11.6	≤ 25.7	≤ 32.4	≤ 56.1
5.779	25.7	0	1	1	0	1	1	1
10.008	56.1	0	0	1	0	0	0	1
5.779	11.6	0	1	1	1	1	1	1
3.834	32.4	1	1	1	0	0	1	1

4. Compare each original continuous value in the feature with each of the sorted thresholds. If the feature value is greater than the threshold, set the corresponding binary variable to 0, otherwise, set it to 1.

5. Repeat steps (i) to (iv) until all the continuous values have been converted into binary form.

As an example, consider the first feature in Table B.1, which manifests the three unique values 5.779, 10.008, and 3.834. These values are sorted, forming the thresholds ≤ 3.834, ≤ 5.779, and ≤ 10.008. Now, each original feature value in the first column is binarized by comparing the the value with the thresholds identified for that feature. For instance, the feature value 5.779 is greater than 3.834, equal to 5.779, and less than 10.008. Hence, the corresponding bit representation becomes 011. Similarly, feature values 10.008 and 3.834 are presented as 001 and 111, respectively. Notice that the number of bits required in binary form equals the number of thresholds, which leads to scalability issues when the number of unique feature values grows.

Once all of the continuous feature values in the first feature column have been binarized, conversion of the second feature column starts. As one can see in the table, the final binarized feature matrix is obtained by concatenating the binary representation of each original continuous feature.

As discussed in Section B.1, the above preprocessing scheme faces extrapolation challenges with scale shifts in the transition from training data to real-world operation. Hence, we here first investigate how standardizing the training and testing data separately using Eq. (B.4) can help accommodate such shifts:

$$\bar{x}_{k,i} = \frac{x_{k,i} - \mu_{Xk}}{\sigma_{Xk}}. \tag{B.4}$$

Above, $x_{k,i}$ refers to the value of feature k in input vector i, while $\bar{x}_{k,i}$ refers to the feature value after it has been standardized using the mean μ_{Xk} and standard deviation σ_{Xk} of the complete set of feature values.

Once the dataset is standardized, it can be binarized using the brute-force approach. However, for better scalability in terms of memory usage and processing time, we employ *stratified sampling* to select representative thresholds. For comparison, we also evaluate uniform sampling.

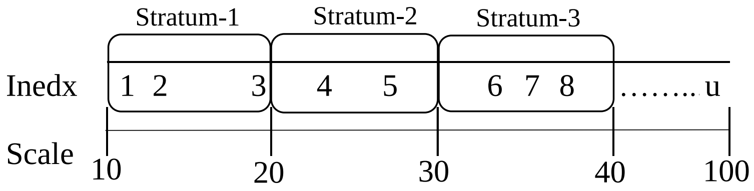


Figure B.2: Selecting thresholds from the set of unique values using the “stratified” method.

B.3.1 Stratified Sampling of Thresholds

Stratified random sampling is a popular sampling technique in statistics. The technique involves division of a population into sub-groups and samples are selected equally and randomly from those sub-groups. These sub-groups are called “strata”. The strata are formed based on the characteristics of the individual members of the population.

Our stratified threshold sampling procedure can be summarized as follows. First, we sort the unique values from smallest to largest. The strata are formed by simply dividing the range between the smallest and largest unique values into a preset number of strata. For instance, if the smallest unique value is 10, the largest unique values is 100, and the required number of strata is 9, then the size of a stratum is 10 [$(100 - 10)/9 = 10$]. As shown in Figure B.2, unique values are partitioned so that unique values between 10 and 20 are in the first stratum, unique values between 20 and 30 are in the second stratum, and so on. Thus, the strata may not contain the same number of values.

Once the strata are formed, we generate one threshold value from each stratum. Instead of selecting a random unique value, however, we compute the mean of the unique values in each stratum separately and consider the computed mean as the threshold. If there are empty strata, we simply discard them.

The total number of binary features in the resulting feature matrix can then be controlled by varying the total number of strata. With a larger number of strata, more thresholds are produced, potentially leading to higher accuracy at the expense of more binary features. Conversely, fewer strata provides a sparser representation, at the potential cost of reduced accuracy.

B.3.2 Gap-based Thresholding

Once the unique values of a feature have been identified, we can select thresholds by skipping a fixed number of unique values as an alternative to stratified sampling. Again, the unique values are sorted from smallest to large, $n_i, i = 1, 2, ..., u$. The indexes of the sorted unique values are shown in the x-axis of the Figure B.3.

Here, the user sets a fixed “gap” to sample the thresholds. For instance, a gap size of four leads to the 1^{st}, 6^{th}, 11^{th} values being used as thresholds, as illustrated in Figure B.3.

With this sampling technique, it is the gap size that decides the sparseness of the binarization. Smaller gaps result in a large number of thresholds, providing a more accurate representation. Larger gap sizes, on the other hand, produces fewer thresholds and a more granular, but sparser representation.

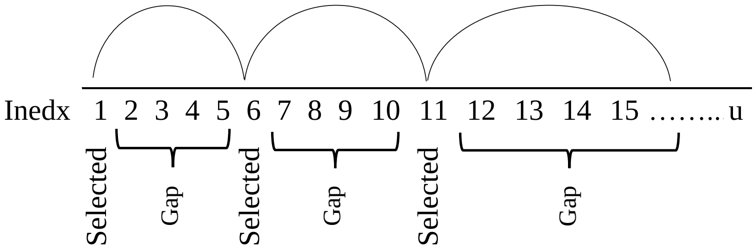


Figure B.3: Selecting thresholds from the set of unique values using the "gap" method.

Once the thresholds are sampled using any of the above techniques, the conversion process continues from Step 4 in the previously described five-step conversion process.

B.4 Empirical Results

We first study the impact of the different preprocessing schemes using two artificial datasets, before we investigate performance on forecasting dengue outbreaks in the Philippines.

B.4.1 Artificial Data

B.4.1.1 Experimental Setup

Both artificial datasets consist of a single continuous feature. The feature values of the first artificial dataset are drawn from a Normal distribution with mean 50 and standard deviation 5. The feature values of the second artificial dataset are drawn from a Gamma distribution with mean 50 and standard deviation of 15. We generate 8000 training samples and 2000 testing samples. However, in order to evaluate the effect of standardizing input features, we scale up the testing feature values by multiplying the original feature values by 3.

The continuous output target y_i^c is calculated as $y_i^c = 3 \times x_i^c + 5$, with x_i^c being the input feature value of sample i. The superscript c indicates that the values are continuous and have not been binarized yet. The resulting datasets are plotted in Figure B.4. In each subplot, sample 1 to $8,000$ are used for training, while sample 8001 to $10,000$ are used for testing. Figure B.4(a) depicts the original training and testing input features drawn from a Normal distribution and the corresponding continuous outputs. Figure B.4(b) illustrates the standardized training and testing input features, drawn from the Normal distribution and corresponding continuous outputs. Figure B.4(c) outlines the original training and testing input features drawn from a Gamma distribution and the corresponding continuous outputs. Figure B.4(d) renders the standardized training and testing input features drawn from the Gamma distribution and the corresponding continuous outputs.

In order to binarize the continuous output y_i^c, we employ Eq. (B.5):

$$\hat{y}_i = \begin{cases} 1 \quad \text{if} \quad & y_i^c \ \geq \ P_{80} \ \text{ of } \ y^c, \\ 0 \quad \text{if} \quad & y_i^c \ < \ P_{80} \ \text{ of } \ y^c. \end{cases} \tag{B.5}$$

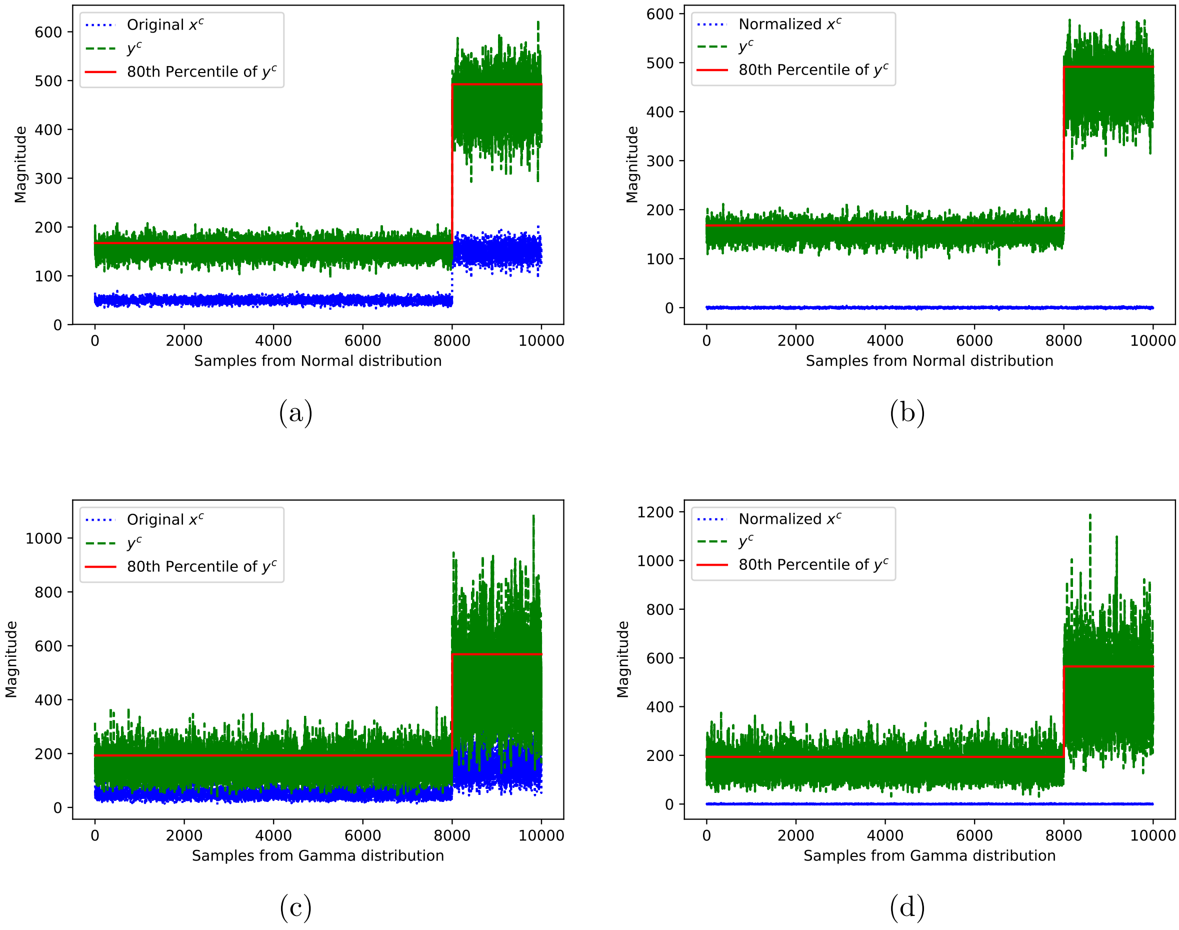


Figure B.4: (a) Original input features drawn from a Normal distribution and corresponding continuous outputs. (b) Standardized input features drawn from a Normal distribution and corresponding continuous outputs. (c) Original input features drawn from a Gamma distribution and corresponding continuous outputs. (d) Standardized input features drawn from a Gamma distribution and corresponding continuous outputs.

P_{80} in Eq. (B.5) is the 80^{th} percentile of the output series. Note that the training and testing outputs are binarized separately with different percentile values since their scales are different. The percentile-based threshold for binarizing the output value is marked in Figure B.4 by a solid red line. If the considered continuous output is higher than or equal to P_{80}, the categorical output $\hat{y}_i$ becomes 1, otherwise, it becomes 0.

The training and testing input feature values of both artificial datasets are binarized using the proposed preprocessing schemes. In order to properly examine the extrapolation ability, we binarize the features of both datasets before and after standardizing them. We are also interested in seeing the impact of producing thresholds from training data alone, and from training and test data combined. We finally investigate the behaviour of the different threshold sampling methods.

A TM with 200 clauses is used to classify the artificial data. The target T and specificity s are set to 20 and 2, respectively. The TAs in all the clauses maintain 100 states per action.

B.4.1.2 Analysis of Results

Table B.2 and Table B.3 contain the training and testing accuracies produced from the above experiment configurations when input features are from the Normal and Gamma distributions, respectively. As seen, the difference between testing and training accuracy caused by standardizing the feature values is the most notable result in these tables.

Table B.2: Training and testing accuracies when original and standardized features are from the Normal distribution

Normal Distribution				
Preprocessing	Binarizing	Sampling	Accuracy	
			Training	Testing
Original features	Thresholds from both training and testing features	All unique values	1.0000	0.2000
		Stratified	0.9981	0.2000
		Thresholds with gap	0.9981	0.2000
	Thresholds from only training features	All unique values	1.0000	0.2000
		Stratified	0.9986	0.2000
		Thresholds with gap	1.0000	0.2000
Standardized features	Thresholds from both training and testing	All unique values	1.0000	0.9927
		Stratified	0.9986	0.9920
		Thresholds with gap	0.9987	0.9915
	Thresholds from only training features features	All unique values	1.0000	0.9985
		Stratified	0.9986	0.9983
		Thresholds with gap	0.9984	0.9984

Table B.3: Training and testing accuracies when original and standardized features are from the Gamma distribution

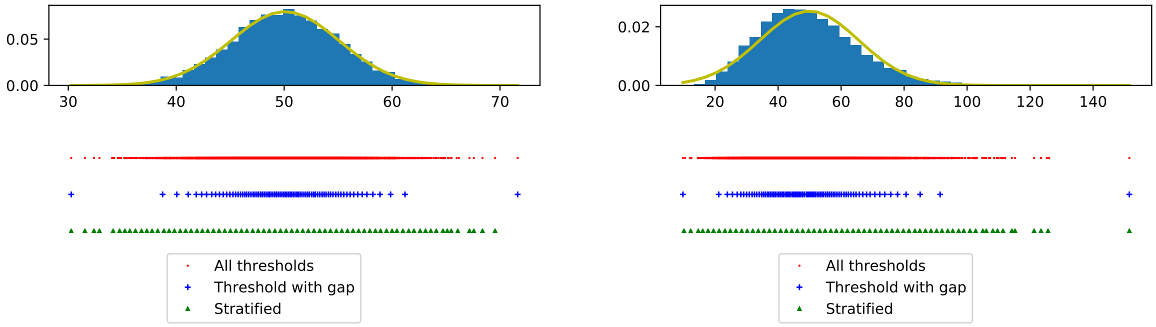
Gamma Distribution				
Preprocessing	Binarizing	Sampling	Accuracy	
			Training	Testing
Original features	Thresholds from both training and testing features	All unique values	1.0000	0.2000
		Stratified	0.9977	0.2000
		Thresholds with gap	0.9981	0.2000
	Thresholds from only training features	All unique values	1.0000	0.2000
		Stratified	0.9992	0.2000
		Thresholds with gap	0.9990	0.2000
Standardized features	Thresholds from both training and testing	All unique values	1.0000	0.9999
		Stratified	0.9986	0.9910
		Thresholds with gap	0.9979	0.9895
	Thresholds from only training features features	All unique values	1.0000	0.9905
		Stratified	0.9994	0.9960
		Thresholds with gap	0.9993	0.9890

When the data is not standardized, testing accuracy is 0.2 regardless of remaining preprocessing. This can be explained by the relatively large shift of feature values occurring from the training to the test data (mean shifting from 50 to 155), highlighting the limited extrapolation capacity of static unnormalized input feature thresholds. Indeed, the shift makes the TM always predict $y = 1$, consequently, providing an accuracy of 0.2. However, standardizing the input features, as shown in Figure B.4(b) and Figure B.4(d), makes the following TM training more robust towards feature value shifts and scaling.

We next investigate whether it is necessary to include the testing data when producing the binarization thresholds. As shown, it turns out that including the testing data has limited effect, which is advantageous for real-world operation on new data.

When all unique values are considered as thresholds in the binarization process, the TM can achieve 1.0 training accuracy for both Normal and Gamma distributed feature values. Regardless of the sampling technique, the TM further obtains testing accuracy surpassing 0.99 when the input features are Normally distributed and standardized. However, when the input features are Gamma-distributed and standardized, testing accuracies greater than 0.99 are only achieved when all unique values are considered as thresholds, or when the stratified sampling is applied.

The reason can be explained using Figure B.5, which illustrates the input feature distribution and selected thresholds when features are Normally distributed (Figure B.5(a)) and when feature are Gamma-distributed (Figure B.5(b)). Below each histogram, the thresholds selected by each binarization scheme are plotted. Blue crosses represent the "gap"-based thresholds, while the green triangles represent the stratification-based thresholds. By comparing with the complete range of unique data points (red dots) and the corresponding histograms, the advantage of stratified sampling is apparent. Gap-based sampling is adversely affected by the long right tail in Figure B.5(b), concentrating significantly more binarization thresholds around the mean. This leads to poorer representation of the tails, causing a potential loss in testing accuracy.



(a)

(b)

Figure B.5: (a) Sampling unique values as thresholds when features are from the Normal distribution. (b) Sampling unique values as thresholds when features are from the Gamma distribution.

B.4.2 Real-World Data

We now study the different data preprocessing schemes using a real-world dataset, i.e., Dengue Incidences.

B.4.2.1 Experimental Setup

The dataset contains monthly dengue incidences of all 17 administrative regions in the Philippines from 2008 to 2016, per 100,000 population. More than 20 monthly dengue incidences per 100,000 population is used as an outbreak indicator. Using the data from 2008 to 2015 for training, dengue outbreaks in the months of 2016 are to be predicted for all the regions. Dengue incidences of previous months of the same and neighboring regions are used as input features. More details about the selection of features can be found in [12].

Dengue incidences are predicted with a TM employing 2000 clauses. The target T and specificity s are set to 15 and 8, respectively. Each experiment is performed 20 times, and we report mean performance across these trials.

Results from the TM are compared with results from three other machine learning techniques: ANNs, a SVM, and a Decision Tree (DT). For comprehensiveness, four ANN architectures are used to forecast the dengue outbreaks: ANN-1 – one hidden layer with 5 neurons, ANN-2 – one hidden layer with 20 neurons, ANN-3 – three hidden layers with 20, 150, and 100 neurons, respectively, and ANN-4 – five hidden layers with 20, 200, 150, 100, and 50 neurons. The SVM uses a Radial Basis Function kernel to capture the non-linear patterns in the data. The regularization parameter (C) in this case is fixed at 1.0 with $gamma = 1/(\text{the number of input features})$ to maximize prediction accuracy. The parameters which decide the quality of the DT output, such as maximum tree depth (=max), minimum number of samples required for split (=2), and the minimum number of samples required for a leaf node (=1) are again all adjusted to optimize prediction accuracy.

B.4.2.2 Results

Table B.4 contains a summary of forecasting accuracies on the Dengue Incidences dataset attained by the various forecasting models with different data preprocessing techniques.

Since there are 17 regions in Philippines, all of the 204 testing samples are utilized to measure the accuracy of each technique. These samples encompass 22 outbreaks. For each model, precision, recall, F1-score, and accuracy are calculated.

When the complete set of unique values are used as thresholds, the TM obtains the highest mean values for precision and F1-score. The second highest precision (0.429) is obtained by the SVM, with the sacrifice of a much lower recall. Conversely, DT produces the highest recall, however, precision suffers. Considering overall performance, captured by the F1 score, the TM obtains the highest mean score (0.400) while ANN-3 obtains the second highest (0.364). Even though the mean F1-score peaks at 0.364, as a result of increasing the structural complexity of the ANNs, the score drops again when complexity is increased further. Due to the imbalance of the dataset (182 non-outbreaks and 22

Table B.4: Summary of forecasting accuracies on Dengue data by different forecasting models with different data preprocessing techniques.

		TM	ANN-1	ANN-2	ANN-3	ANN-4	SVM	DT
All threshold	Recall	0.364	0.227	0.318	0.364	0.318	0.136	0.409
	Precision	0.444	0.357	0.389	0.364	0.368	0.429	0.290
	F1-score	0.400	0.278	0.350	0.364	0.341	0.207	0.340
	Accuracy	0.882	0.873	0.873	0.863	0.868	0.887	0.828
Stratified	Recall	0.318	0.227	0.318	0.318	0.318	0.136	0.364
	Precision	0.368	0.238	0.333	0.350	0.333	0.375	0.242
	F1-score	0.341	0.233	0.326	0.333	0.326	0.200	0.291
	Accuracy	0.868	0.838	0.858	0.863	0.858	0.882	0.809
Threshold with gap	Recall	0.318	0.227	0.318	0.318	0.318	0.136	0.364
	Precision	0.333	0.238	0.292	0.304	0.304	0.333	0.235
	F1-score	0.326	0.233	0.304	0.311	0.311	0.194	0.286
	Accuracy	0.858	0.838	0.843	0.848	0.848	0.877	0.804

outbreaks), the SVM produces a particularly high accuracy (0.887) by mostly classifying instances as non-outbreaks. Finally, note that the precision, recall, F1-score, and accuracy of the TM are higher than what we were able to achieve with the ANN models.

When the thresholds are selected with the two sampling techniques, recall, precision, F1-score, and accuracy drop. Both sampling approaches attain the same recall. However, precision, F1-score, and accuracy are better with stratified sampling. Note that despite the drop in F1-score, both sampling techniques attain better F1-score than the other machine learning techniques. That is, the drop in mean F1-score from using all the unique feature values as thresholds to stratified sampling is of little significance.

B.5 Conclusion

In this paper, we extended a brute-force threshold-based binarization approach that allows TMs to accurately process continuous features. Our goal was to handle feature scale shifts and to improve scalability without loosing accuracy. We applied the resulting binarization approach to forecast dengue outbreaks in the Philippines after an empirical empirical investigation on two artificial datasets. While the experiments with the artificial datasets confirmed the advantages of data standardization, the results on the real-life dataset further demonstrated how sampling can reduce the number of thresholds needed for binarizing continuous features, without significant accuracy loss. Even with a sparser representation of continuous features, the TM is competitive with other widely used machine learning approaches. Indeed, it turned out that the TM is more accurate than the evaluated SVMs, Decision Trees, and several multi-layered ANNs.

Bibliography

[1] Bart Baesens, Christophe Mues, Manu De Backer, Jan Vanthienen, and Rudy Setiono. “Building Intelligent Credit Scoring Systems Using Decision Tables”. In: *Enterprise Information Systems V*. Springer, 2004, pp. 131–137.

[2] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. “An Empirical Evaluation of the Comprehensibility of Decision Table, Tree and Rule Based Predictive Models”. In: *Decision Support Systems* 51.1 (2011), pp. 141–154.

[3] Riccardo Bellazzi and Blaz Zupan. “Predictive Data Mining in Clinical Medicine: Current Issues and Guidelines”. In: *International journal of medical informatics* 77.2 (2008), pp. 81–97.

[4] Michael J Pazzani, S Mani, and William R Shankle. “Acceptance of Rules Generated by Machine Learning Among Medical Experts”. In: *Methods of information in medicine* 40.05 (2001), pp. 380–385.

[5] Alex A Freitas, Daniela C Wieser, and Rolf Apweiler. “On the Importance of Comprehensible Classification Models for Protein Function Prediction”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7.1 (2008), pp. 172–182.

[6] Duane Szafron, Paul Lu, Russell Greiner, David S Wishart, Brett Poulin, Roman Eisner, Zhiyong Lu, John Anvik, Cam Macdonell, Alona Fyshe, et al. “Proteome Analyst: Custom Predictions With Explanations in a Web-Based Tool for High-Throughput proteome Annotations”. In: *Nucleic acids research* 32.suppl_2 (2004), W365–W371.

[7] Elen Lima, Christophe Mues, and Bart Baesens. “Domain Knowledge Integration in Data Mining Using Decision Tables: Case Studies in Churn Prediction”. In: *Journal of the Operational Research Society* 60.8 (2009), pp. 1096–1106.

[8] Wouter Verbeke, David Martens, Christophe Mues, and Bart Baesens. “Building Comprehensible Customer Churn Prediction Models with Advanced Rule Induction Techniques”. In: *Expert systems with applications* 38.3 (2011), pp. 2354–2364.

[9] Christoph Molnar. *Interpretable Machine Learning*. Lulu. com, 2019.

[10] Ole-Christoffer Granmo. “The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic”. In: *arXiv preprint arXiv:1804.01508* (2018).

[11] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. “Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications”. In: *IEEE Access* 7 (2019), pp. 115134–115146.

[12] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. “A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2019, pp. 564–578.

[13] Michael Lvovitch Tsetlin. “On Behaviour of Finite Automata in Random Medium”. In: *Avtomat. i Telemekh* 22.10 (1961), pp. 1345–1354.

[14] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. “The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression”. In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

Paper C

Adaptive Sparse Representation of Continuous Input for Tsetlin Machines Based on Stochastic Searching on the Line

This paper introduces a novel approach to representing continuous inputs in Tsetlin Machines (TMs). Instead of using one Tsetlin Automaton (TA) for every unique threshold found when Booleanizing continuous input, we employ two Stochastic Searching on the Line (SSL) automata to learn discriminative lower and upper bounds. The two resulting Boolean features are adapted to the rest of the clause by equipping each clause with its own team of SSLs, which update the bounds during the learning process. Two standard TAs finally decide whether to include the resulting features as part of the clause. In this way, just four automata altogether represent one continuous feature (instead of potentially hundreds of them). We evaluate the performance of the new scheme empirically using five datasets, along with a study of interpretability. On average, TMs with SSL feature representation use 4.3 *times fewer literals than the TM with static threshold-based features. Furthermore, in terms of average memory usage and F1-Score, our approach 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. Our approach further outperform Neural Additive Models on Fraud Detection and StructureBoost on CA-58 in terms of Area Under Curve while performing competitively on COMPAS.*

C.1 Introduction

Deep learning (DL) has significantly advanced state-of-the-art models in Machine Learning (ML) over the last decade, attaining remarkable accuracy in many ML application domains. One of the issues with DL, however, is that DL inference cannot easily be interpreted [1]. This limits the applicability of DL in high-stakes domains such as medicine [2, 3], credit-scoring [4, 5], churn prediction [6, 7], bioinformatics [8, 9], crises analysis [10], and criminal justice [11]. In this regard, the simpler and more interpretable ML algorithms like Decision Trees, Logistic Regression, Linear Regression, and Decision Rules, can be more suitable. Yet, they are all hampered by low-accuracy when facing complex problems [12]. This limitation has urged researchers to develop machine learning algorithms that are capable of achieving a better trade-off between interpretability and accuracy.

While some researchers focus on developing entirely new machine learning algorithms as scoped above, other researchers try to render DL interpretable. A recent attempt to make DL interpretable is the work of Agarwal et al. [11]. They introduce a Neural Additive Models (NAMs), which treats each feature independently. The assumption of independence makes NAMs interpretable but impedes accuracy compared with regular DL [11]. Another approach is to try to explain DL inference with surrogate models. Here, one strives to attain *local* interpretability, i.e., explaining individual predictions [13]. Nevertheless, these explanations are only approximate and cannot explain the complete DL model (globally interpretability) [14].

Many prominent interpretable ML approaches are based on natively interpretable rules, tracing back to some of the well-known learning models such as association rule learning [15]. These have for instance been used to predict sequential events [16]. Other examples include the work of Feldman on the hardness of learning formulae in Disjunctive Normal Form (DNF) [17] and Probably Approximately Correct (PAC) learning, which has provided fundamental insight into machine learning as well as a framework for learning formulae in DNF [18]. Approximate Bayesian techniques are another set of approaches for robust learning of rules [19, 20]. Hybrid Logistic Circuits (HLC), introduced in [21], is yet another novel approach to interpretable machine learning. Here, layered logical operators tranlates into a logistic regression function. HLC has demonstrated promising accuracy in image classification. 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 DL.

Another recent interpretable approach to machine learning is Explainable Boosting Machines (EBMs) [22]. EBMs are highly intelligible and explainable while accuracy is comparable to state-of-the-art machine learning methods like Random Forest and BoostedTrees [23]. Indeed, EBMs is recognized as state-of-the-art within Generalized Additive Models (GAMs) [22, 23]. The EBMs learn feature functions independently, using methods such as gradient boosting or bagging. This allows the user to see how much each feature contributes to the model's prediction and is hence directly interpretable.

The recently introduced **Tsetlin Machines (TMs)** [24], however, have obtained competitive accuracy while producing human-interpretable outputs in a wide range of domains[25, 26, 27, 28, 29]. At the same time, TMs utilize comparably low computational resources [30]. Employing a team of TA [31], 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) [26]. In other words, the TM can be said to unify logistic regression and rule-based learning in a way that boosts accuracy, while maintaining interpretability.

Recent progress on TMs. The TM has recently been adopted for various application domains such as natural language understanding [32, 33], image classification [34] and speech processing [35]. Simultaneously, the TM architecture and learning mechanism has been improved in terms of accuracy, computation speed, and energy usage. The convolutional TM provide competitive performance on MNIST, Fashion-MNIST, and Kuzushiji-MNIST, in comparison with CNNs, K-Nearest Neighbor, Support Vector Machines, Random Forests, Gradient Boosting, Binary Connect, Logistic Circuits and ResNet [36]. The regression TM [27] opens up for continuous output, achieving on par or better performance compared to Random Forest, Regression Trees, and Support Vector Regression. With the proposed Booleanization scheme in [37], the TM is also able to operate with continuous features.

Furthermore, clauses are enhanced with weights in [25]. The weights reduce the number of clauses required without any loss of accuracy. Later, integer weights replaced real-valued weights to both reduce the number of clauses and to increase the interpretability of the TM [38] . On several benchmarks, the integer-weighted TM version outperformed simple Multi-Layered Artificial Neural Networks, Support Vector Machines, Decision Trees, Random Forest, K-Nearest Neighbor, Random Forest, Explainable Boosting Machines (EBMs), Gradient Boosted Trees (XGBoost), as well as the standard TM. Further, the introduced multi-granular clauses to TM in [28] eliminating the pattern specificity parameter from the TM and which consequently simplify the hyper-parameter search. By indexing the clauses on the features that falsify them, up to an order of magnitude faster inference and learning has been reported [29]. Several researchers have further introduced techniques and architectures that reduce memory footprint and energy usage [39], while other techniques improve learning speed [40, 39] and support explainability [41, 42].

Recent theoretical work proves convergence to the correct operator for “identity” and “not”. It is further shown that arbitrarily rare patterns can be recognized, using a quasi-stationary Markov chain-based analysis. The work finally proves that when two patterns are incompatible, the most accurate pattern is selected [43]. Convergence for the “XOR” operator has also recently been proven by [44].

Paper Contributions: The approach proposed in [37] is so far the most effective way of representing continuous features through Booleanization. However, the approach requires a large number of TAs to represent the Booleanized continuous features. Indeed, one needs one TA per unique continuous value. Consequently, this increases the training time of the TM as it needs to update all the TAs in all of clauses, per training iteration. Further, this adds more post-processing work for generating interpretable rules out of TM outputs. To overcome this challenge in TMs, we propose a novel approach to represent continuous features in the TM, encompassing the following contributions.

- Instead of representing each unique threshold, found in the Booleanization process, by a TA, we use Stochastic Searching on the Line (SSL) automaton [45] to learn the lower and upper limits of the continuous feature values. These limits decide the Boolean representation of the continuous value inside the clause. Merely two TAs then decide whether to include these bounds in the clause or to exclude them from the clause. In this way, one continuous feature can be represented by just four automata, instead of representing it by hundreds of TAs (decided by the number of unique feature values within the feature).

- A new approach to calculating the clause output is introduced to match with the above Booleanization scheme.
- We update the learning procedure of the TM accordingly, however, building upon Type I and Type II feedback to learn the lower and upper bounds of the continuous input.
- Empirically, we evaluate our new scheme using eight data sets: Bankruptcy, Balance Scale, Breast Cancer, Liver Disorders, Heart Disease, Fraud Detection, COMPAS, and CA-58. With the first five datasets, we show how our novel approach affects memory consumption, training time, and the number of literals included in clauses, in comparison with the threshold-based scheme [46]. Furthermore, performance on all these datasets are compared against recent state-of-the-art machine learning models.

Paper Organization: In Section C.2, we present the learning automata foundation we build upon and discuss the SSL automaton in more detail. Then, in Section C.3, we introduce the TM and how it tradtionally has dealt with continuous features. We then propose our new SSL-based scheme. We evaluate the performance of our new scheme empirically using five datasets in Section C.5. In this section, we use Bankruptcy dataset to demonstrate how rules are extracted from TM clause outputs. The prediction accuracy of the TM SSL-based continous feature representation are then compared against several competing techniques, including ANNs, SVMs, DTs, RF, KNN, EBMs (the current state-of-the-art of Generalized Additive Models (GAMs) [22, 23]), Gradient Boosted Trees (XGBoost), and TM with regular continuous feature representation. Further, we contrast the performance of the TM against reported results on recent state-of-the-art machine learning models, namely NAMs [11] and StructureBoost [47]. Finally, we conclude our paper in Section C.6.

C.2 Learning Automata and the Stochastic Searching on the Line Automaton

The origins of Learning Automata (LA) [48] can be traced back to the work of M. L. Tsetlin in the early 1960s [31]. In a stochastic environment, an automaton is capable of learning the optimum action which has the lowest penalty probability through trial and error. There exists different types of automata. Which one to use is decided by the nature of the application [49].

Initially, the LA randomly perform an action from its available set of actions. This action is then evaluated by its attached environment. The environment randomly produces feedback, i.e., a reward or a penalty as a response to the action selected by the LA. Depending on the feedback, the state of the LA is adjusted. If the feedback is a reward, the state changes towards the end state of the selected action, reinforcing the

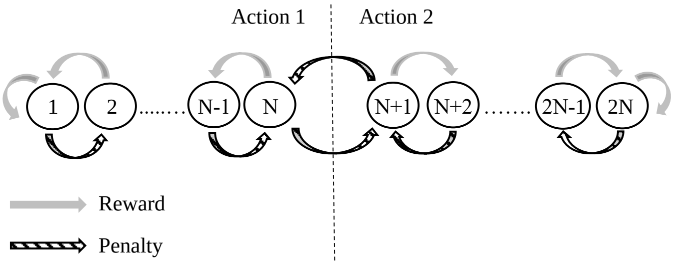


Figure C.1: Transition graph of a two-action Tsetlin Automaton with 2N memory states.

action. When the feedback is a penalty, the state changes towards the center state of the selected action, weakening the action and eventually switching action. The next action of the automaton is then decided by the new state. In this manner, an LA interacts with its environment iteratively. With a sufficiently large number of states and a reasonably large number of interactions with the environment, an LA learn to chose the optimum action with probability arbitrarily close to 1.0 [48].

During LA learning, the automaton can make deterministic or stochastic jumps as a response to the environment feedback. LA make stochastic jumps by randomly changing states according to a given probability. If this probability is 1.0, the state jumps are deterministic. Automaton of this kind are called deterministic automata. If the transition graph of the automaton is kept static, we refer to it as a fixed-structure automaton. The TM employs TAs to decide which literals to include in the clauses in clauses. A TA is deterministic and has a fixed structure, formulated as a finite-state automaton [31]. A TA with $2N$ states is depicted in Figure C.1. States 1 to N map to Action 1 and states $N+1$ to $2N$ map to Action 2.

The stochastic searching on the line (SSL) automaton pioneered by Oommen [45] is somewhat different from the regular automata. The SSL automaton is an optimization scheme designed to find an unknown optimum location λ^* on a line, seeking a value between 0 to 1, $[0,1]$.

In SSL learning, λ^* can be one of the N points. In other words, the search space is divided into N points, $\{0, 1/N, 2/N, \ldots, (N-1)/N, 1\}$, with N being the discretization resolution. Depending on the possibly faulty feedback from the attached environment (E), λ moves towards the left or right from its current state on the created discrete search space. We consider the environment feedback 1, $E=1$ as an indication to move towards right (or to increase the value of λ) by one step. The environment feedback 0, $E=0$, on the other hand, is considered as an indication to move towards the left (or to decrease the value of λ) by one step. The next location of λ, $\lambda(n+1)$ can thus be expressed as follows:

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

$$\lambda(n+1) = \begin{cases} \lambda(n), & \text{if} \quad \lambda(n) = 1 \text{ and } E(n) = 1 \ , \\ \lambda(n), & \text{if} \quad \lambda(n) = 0 \text{ and } E(n) = 0 \ . \end{cases} \tag{C.2}$$

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

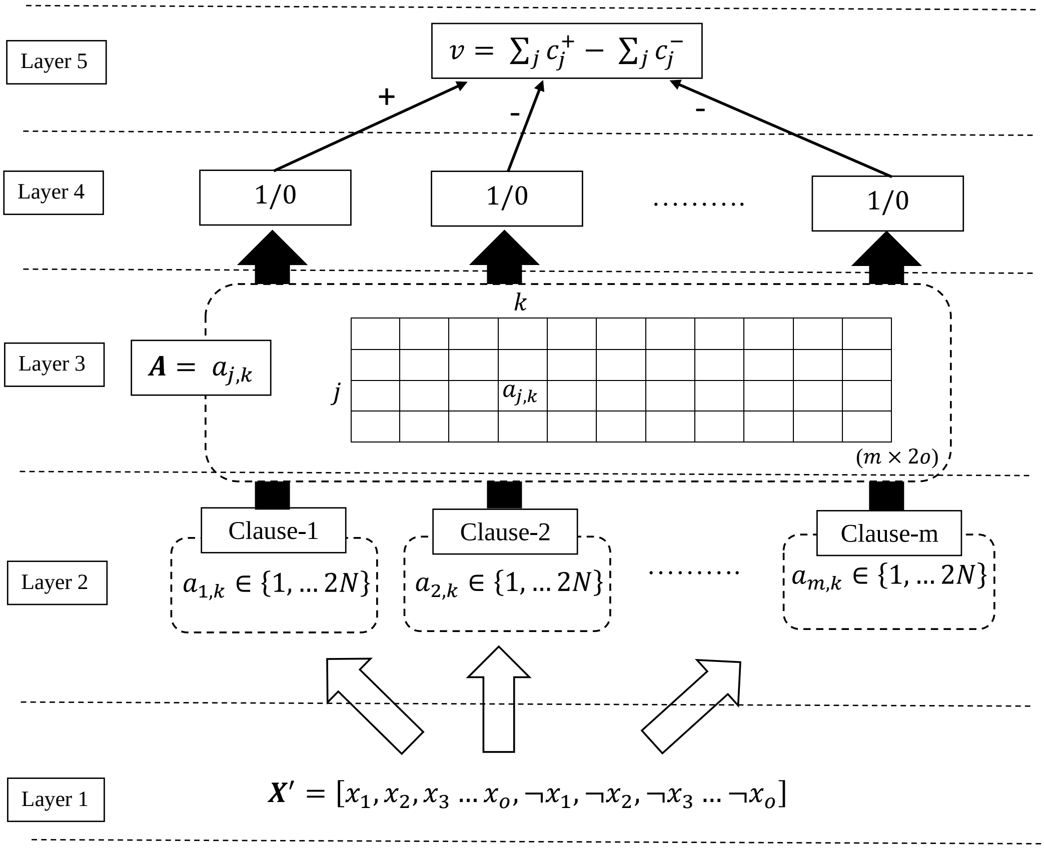

Layer 5
$v = \sum_j c_j^+ - \sum_j c_j^-$
+
-
-
Layer 4
1/0
1/0
....
1/0
Layer 3
$A = a_{j,k}$
k
j
$a_{j,k}$
$(m \times 2o)$
Layer 2
Clause-1
Clause-2
Clause-m
$a_{1,k} \in \{1, \ldots 2N\}$
$a_{2,k} \in \{1, \ldots 2N\}$
$a_{m,k} \in \{1, \ldots 2N\}$
Layer 1
$X' = [x_1, x_2, x_3 \ldots x_o, \neg x_1, \neg x_2, \neg x_3 \ldots \neg x_o]$

Figure C.2: The TM structure.

C.3 Tsetlin Machine (TM) for Continuous Features

As seen in Figure C.2, conceptually, TM decomposes into five layers for recognizing sub-patterns in the data and categorizing them into classes. In this section, we explain the job of each of these layers in the pattern recognition and learning phases of the TM. The parameters and symbols used in this section are explained and summarized in Table C.1.

Layer 1: the input. In the input layer, the TM receives a vector of o propositional variables: $\mathbf{X}$, $x_k \in \{0,1\}^o$. The objective here of the TM is to classify this feature vector into one of the two classes, $y \in \{0,1\}$. However, as shown in Figure C.2, the input layer also includes negations of the original features, $\neg x_k$ in the feature set to capture more sophisticated patterns. Collectively, the elements in the augmented feature set are called literals: $\mathbf{L} = [x_1, x_2, \ldots, x_o, \neg x_1, \neg x_2, \ldots, \neg x_o] = [l_1, l_2, \ldots, l_{2o}]$.

Layer 2: clause construction. The sub-patterns associated with class 1 and class 0 are captured by m conjunctive clauses. The value m is set by the user where more complex problems might demand large m. All clauses receive the same augmented features set formulated at the input layer, $\mathbf{L}$. However, to perform the conjunction, only a fraction of literals is utilized. The TM employs two-action TAs in Figure C.1 to decide which literals are included in which clauses. Since we found $2 \times o$ number of literals in $\mathbf{L}$, the same number of TAs, one per literal k, is needed by a clause to decide the included literals in the clause. When the index set of the included literals in clause j is given in I_j the conjunction of the clause can be performed as follows:

Table C.1: Parameters and symbols used in Section C.3

Parameter/ Symbol	Description	Parameter/ Symbol	Description
$\boldsymbol{X}$	Input vector containing o propositional variables	$\boldsymbol{L}$	The augmented feature set containing both original and negated features
x_k	k^{th} propositional variable	l_k	k^{th} literal
m	The number of clauses	c_j	j^{th} clause and stored in vector $\mathbf{C}$
I_j	The index set of the included literals in clause j	N	The number of states per action in the TA
$a_{j,k}$	State of the k^{th} literal in the j^{th} clause and stored in matrix $\mathbf{A}$	I_X^1	The indexes of the literals of value 1
v	The difference between positive and negative clause outputs	p_j	Decision on receiving Type I or Type II feedback and stored in vector $\mathbf{P}$
T	Feedback threshold	s	Learning sensitivity
$r_{j,k}$	The decision whether the k^{th} TA of the j^{th} clause is to receive Type Ia feedback and stored in matrix $\mathbf{R}$	$q_{j,k}$	The decision whether the k^{th} TA of the j^{th} clause is to receive Type Ib feedback and stored in matrix $\mathbf{Q}$
I^{Ia}	Stores TA indexes selected for Type Ia feedback	I^{Ib}	Stores TA indexes selected for Type Ib feedback
$\oplus$	Denotes adding 1 to the current state value of the TA	$\ominus$	Denotes substracting 1 from the current state value of the TA
I^{II}	Stores TA indexes selected for Type II feedback		

$$c_j = \bigwedge_{k \in I_j} l_k. \tag{C.3}$$

Notice how the composition of a clause varies from another clause depending on the indexes of the included literals in the set $I_j \subseteq \{1, \ldots, 2o\}$. For the special case of $I_j = \emptyset$, i.e., an empty clause, we have:

$$c_j = \begin{cases} 1 & \textbf{during } \text{learning} \\ 0 & \textbf{otherwise}. \end{cases} \tag{C.4}$$

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

Layer 3: storing states of TAs of clauses in the memory. The TA states on the left hand side of the automaton (states from 1 to N) ask to *exclude* the corresponding literal from the clause while the states on the right hand side of the automaton (states from $N + 1$ to $2N$) ask to *include* the literal in the clause. The systematic storage of states of TAs 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, allows us to find the index set of the included literals in clause j, I_j as $I_j = \{k | a_{j,k} > N, \, 1 \leq k \leq 2o\}$.

Layer 4: clause output. Once the TA decisions are available, the clause output can be easily computed. Since the clauses are conjunctive, a single literal of value 0 is enough to turn the clause output to 0 if its corresponding TA has decided to *include* it in the clause. To make the understanding easier, we introduce set I_X^1, which contains the indexes of the literals of value 1. Then the output of clause j can be expressed as:

$$c_j = \begin{cases} 1 & \text{if} \quad I_j \subseteq I_X^1, \\ 0 & \text{otherwise}. \end{cases} \tag{C.5}$$

The clause outputs, computed as above, are now stored in vector $\mathbf{C}$, i.e., $\mathbf{C} = (c_j) \in \{0, 1\}^m$.

Layer 5: classification: The TM structure given in Figure C.2 is to classify data into two classes. Hence, sub-patterns associated with each class have to be separately learned. For this purpose, the clauses are divided into two groups, where one group learns the sub-pattern of class 1 while the other learn the sub-patterns of class 0. For simplicity, 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 $y = 0$.

The clauses which recognize sub-patterns output 1. This makes the classification process easier as we just need to sum the clause outputs of each class and assign the sample into the class which has the highest sum. Higher sum means more sub-patters identified from the designated class and has a higher chance of being the sample in that class. Hence, 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:

$$\hat{y} = \begin{cases} 1 & \text{if} \;\; v \;\geq\; 0 \\ 0 & \text{if} \;\; v \;<\; 0 \, . \end{cases} \tag{C.6}$$

A TM learns online, updating its internal parameters according to one training sample (X, y) at a time. As we discussed, a TA team decide the clause output and collectively, output of all the clauses decide the TM output. Hence, to maximize the accuracy of the TM output, it is important to sensibly guide individual TAs in clauses. We achieve this with two kinds of reinforcement: Type I and Type II feedback. Type I and Type II feedback decide if the TAs in clauses receive a reward, a penalty, or inaction feedback, depending on the context of their actions. How the type of feedback is decided and how the TAs are updated according to the selected feedback type are discussed below in more details.

Type I feedback: Type I feedback has been designed to reinforce the true positive outputs of the clauses and to combat against the false negative outputs of the clauses. To reinforce the true positive output of a clause (clause output is one when it has to be one), *include* actions of TAs whose corresponding literal value is 1 are strengthened. However, more fine-tuned patterns can be identified by strengthening the *exclude* actions of TAs in the same clause whose corresponding literal value is 0. To combat the false negative outputs of the clauses (clause output is zero when it has to be one), we erase the identified patter by the clause and make it available for a new pattern. To do so, the *exclude* actions of TAs, regardless of their corresponding literal values, are strengthened. We now sub-divide the Type I feedback into Type Ia and Ib where Type Ia handle the reinforcing of *exclude* action while Type Ib work on reinforcing *exclude* of TAs. Together, Type Ia and Type Ib feedback force clauses to output 1. Hence, clauses with positive polarity need Type I feedback when $y = 1$ and clauses with negative polarity need Type I feedback when $y = 0$. To diversify the clauses, they are targeted for Type I feedback stochastically as follows:

$$p_j = \begin{cases} 1 & \text{with probability } \frac{T-\max(-T,\min(T,v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.7}$$

All clauses in each class should not learn the same sub-pattern nor only a few. Hence, clauses should be smartly allocated among the sub-patterns. The user set target T in (C.7) does this while deciding the probability of receiving Type I feedback, i.e., T number of clauses are available to learn each sub-pattern in each class. Higher T increases the robustness of learning by allocating more clauses to learn each sub-pattern. Now, T together with v decides the probability of clause j receiving Type I feedback and accordingly the decision p_j is made. The decisions for the complete set of clauses to receive Type I feedback are organized in the vector $\mathbf{P} = (p_j) \in \{0, 1\}^m$.

Once the clauses to receive Type I feedback are singled out as per (C.7), the probability of updating individual TAs in selected clauses is calculated using the user-set parameter s (s ≥ 1), separately for Type Ia and Type Ib. According to the above probabilities, the decision whether the k^{th} TA of the j^{th} clause is to receive Type Ia feedback, $r_{j,k}$, and Type Ib feedback, $q_{j,k}$, are stochastically made as follows:

$$r_{j,k} = \begin{cases} 1 & \text{with probability } \frac{s-1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.8}$$

$$q_{j,k} = \begin{cases} 1 & \text{with probability } \frac{1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.9}$$

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 $I^{\text{Ib}} = \{(j, k)|(l_k = 0 \lor c_j = 0) \land p_{j,y} = 1 \land q_{j,k} = 1\}$.

The states of the identified TAs are now ready to be updated. Since Type Ia strengthens the *include* action of TAs, the current state should move more towards the *include* action direction. We denote this as $\oplus$ and here $\oplus$ adds 1 to the current state value of the TA. The Type Ib feedback, on the other hand, moves the state of the selected TA towards *exclude* action direction to strengthen the *exclude* action of TAs. We denote this by $\ominus$ and here $\ominus$ subtracts 1 from the current state value of the TA. Accordingly, the states of TAs in $\mathbf{A}$ are updated as: $\mathbf{A} \leftarrow \left(\mathbf{A} \oplus I^{\text{Ia}}\right) \ominus I^{\text{Ib}}$.

Type II feedback: Type II feedback has been designed to combat the false positive output of clauses (clause output is one when it has to be zero). To turn this clause output from 1 to 0, a literal value of 0 can be simply included in the clause. Clauses with positive polarity need Type II feedback when $y = 0$ and clauses with negative polarity need this when $y = 1$ since they do not want to vote for the opposite class. Again using the user-set target T, the decision for the j^{th} clause is made as follows:

$$p_j = \begin{cases} 1 & \text{with probability } \frac{T+\max(-T,\min(T,v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.10}$$

The states of the TAs whose corresponding literal of value 0 in selected clauses according to (C.10) are now moved towards the *include* action direction with probability one. Hence, the index set of this kind can be identified as: $I^{\text{II}} = \{(j,k)|l_k = 0 \land c_j = 1 \land p_j = 1\}$. Accordingly, the states of TAs in $\mathbf{A}$ are updated as: $\mathbf{A} \leftarrow \mathbf{A} \oplus I^{\text{II}}$.

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

Booleanization of Continuous Features: In the TM we discussed so far, the input layer accepted only Boolean features, i.e., $\boldsymbol{X} = [x_1, x_2, x_3, \ldots, x_o]$ with $x_k, k = 1, 2, ..., o$, being 0 or 1. These features and their negations were directly fed into the clauses without any further modifications. However, continuous features in machine learning applications are more common to have than they being just 1 or 0. We, in one of our previous papers [37], presented a systematic procedure of transforming continuous features into Boolean ones while maintaining ranking relationships among the continuous feature values.

We here summarize the previous Booleanization scheme using the example presented in [27]. As seen in Table C.2, we are going to Booleanize the two continuous features listed in table column 1 and column 2.

1. First, in each feature, the unique values are identified.
2. The unique values are then sorted from smallest to largest.
3. The sorted unique values are considered as thresholds. In the table, these values can be seen in the "Thresholds" row.
4. The original feature values are then compared with identified thresholds, only from its own feature value set. If the feature value is greater than the threshold, set the corresponding Boolean variable to 0, otherwise, set the bit to 1.
5. the above steps are repeated until all the features are converted into Boolean form.

The first feature in the first column of the table contains three unique values, i.e., 5.779, 10.008, and 3.834 (step (i)). Once they are sorted as asked in step ii, we get 3.834,

Table C.2: Preprocessing of two continuous features.

Raw Feature		Thresholds						
1	2	≤ 3.834	≤ 5.779	≤ 10.008	≤ 11.6	≤ 25.7	≤ 32.4	≤ 56.1
5.779	25.7	0	1	1	0	1	1	1
10.008	56.1	0	0	1	0	0	0	1
5.779	11.6	0	1	1	1	1	1	1
3.834	32.4	1	1	1	0	0	1	1

5.779, and 10.008. Now we consider them as thresholds, ≤ 3.834, ≤ 5.779, and ≤ 10.008 (step (iii)). Here we get the idea that each original feature in the column 1 is going to represent in 3 bit values. According to step iv, we now compare the original values in the first feature against its thresholds. The first feature value 5.779 is greater than 3.834 (resulting in 0), equal to 5.779 (resulting in 1), and less than 10.008 (resulting in 1). Hence we replace 5.779 with 011. Similarly, 10.008 and 3.834 can be replaced with 001 and 111, respectively.

The conversion of the feature values in the second feature is stared once all the feature values in the first feature is completed. This procedure is iterated until all the continuous values in all the continuous features have been converted to Boolean form (step (v)).

This Boolean representation of continuous features is particularly powerful as it allows the TM to reason about the ordering of the values, forming conjunctive clauses that specify rules based on thresholds, and with negated features, also rules based on intervals. This can be explained again with the following example.

The threshold ≤ 3.834 in the "Threshold" row stands for the continuous value 3.834 of the first feature. Similarly, threshold ≤ 5.779 and ≤ 10.008 represent the continuous value 5.779 and 10.008, respectively. Consider a clause having threshold ≤ 5.779 included in the clause and that is the only threshold included in the clause. Then for any input value *less than or equal to* ≤ 5.779 from that feature, the clause outputs 1. Now consider the case of having two thresholds, ≤ 5.779 and ≤ 10.008 included in the clause. Still the threshold ≤ 5.779 decides the clause output due to the fact that **AND** of ≤ 5.779 and ≤ 10.008 threshold columns in Table C.2 yields the threshold column ≤ 5.779. The inclusion of negated thresholds effects the clause output oppose to the original thresholds. Consider a clause having only the negation of the threshold ≤ 3.834 included in the clause. Now the clause outputs 1 for all the values *higher than* ≤ 3.834 from that feature as **NOT** of ≤ 3.834 is equivalent to $3.834 <$.

The above explanation of the threshold selection reveals that the lowest original threshold included in the clause and the highest negated threshold included in the clause decide upper and lower boundary of the feature values and these thresholds are the only important thresholds to calculate the clause output. Hence, this motivates us to represent the continuous features in clauses in a new way and train the clauses accordingly as follows.

Table C.3: Parameters and symbols used in Section C.4

Parameter/ Symbol	Description	Parameter/ Symbol	Description
$\boldsymbol{X}^c$	Input vector containing o continuous features	x_k^c	k^{th} continuous feature
$SSL_{j,k}^l$	Lower limit of the continuous feature k in clause j	$SSL_{j,k}^u$	Upper limit of the continuous feature k in clause j
$E_{j,k}^l$	Feedback to the SSL automata which represent the lower limit of the feature k in clause j	$E_{j,k}^u$	Feedback to the SSL automata which represent the upper limit of the feature k in clause j
$TA_{j,k}^l$	TA which decides whether to include or exclude lower limit of the k^{th} feature in clause j	$TA_{j,k}^u$	TA which decides whether to include or exclude upper limit of the k^{th} feature in clause j
$r_{j,k}^l$	The decision whether the TA of the lower limit of k^{th} feature in the j^{th} clause is to receive Type Ia feedback	$r_{j,k}^u$	The decision whether the TA of the upper limit of k^{th} feature in the j^{th} clause is to receive Type Ia feedback
$q_{j,k}^l$	The decision whether the TA of the lower limit of k^{th} feature in the j^{th} clause is to receive Type Ib feedback	$q_{j,k}^u$	The decision whether the TA of the upper limit of k^{th} feature in the j^{th} clause is to receive Type Ib feedback
$l_{j,k}$	Computed literal value for the k^{th} feature in clause j		

C.4 Sparse Representation of Continuous Features

However, the above representation of continuous values in clauses is costly as it needs two times the total number of unique values of TAs per clause. This is more severe when the dataset is large and when there are a large number of input features to be considered. Hence, having the same properties of the previous Booleanization scheme, we introduce SLL automata to represent the upper and lower limits of the continuous features. With the new representation, a continuous feature can be then represented by merely two automata instead of having two times the number of unique values in the considered continuous feature. The new parameters and symbols used in this section are explained and summarized in Table C.3.

Input Features. As we discussed earlier, the TM takes o propositional variables as input, $\boldsymbol{X} = [x_1, x_2, x_3, \ldots, x_o]$. In this section, we discuss how TM maps $\boldsymbol{X}^c$ which contains o continuous features, $\boldsymbol{X}^c = [x_1^c, x_2^c, x_3^c, \dots, x_o^c]$ into one of two classes, $y = 1$ or $y = 0$.

Feature Representation. Each continuous feature is assigned two SSLs to represent the upper and lower limits of the continuous value in a clause, i.e., $SSL_1^l, SSL_1^u, \dots, SSL_k^l$, $SSL_k^u, \dots, SSL_o^l, SSL_o^u$. Here, SSL_k^l and SSL_k^u are lower and upper limit values of the k^{th} continuous feature, respectively. However, step size N within an SSL in this case is not constant. When E in (C.1) and (C.2) is 1, the SSL state moves to the higher neighboring unique value of the attached continuous feature of the SSL. Similarly, when E is 0, SSL state moves to the lower neighboring unique value of the considered continuous feature.

Clauses. Each conjunctive clause in the TM receives $\boldsymbol{X}^c$ as an input. The inclusion and exclusion decisions of the corresponding upper and lower bounds of x_k^c in the clause are made by TAs. Hence, each clause now needs $2o$ TAs, where half of them make the decision related to the lower bound of the continuous features while the other half make the decision related to the upper bound of the continuous features. The the matrix $\mathbf{A}$ hence still contains $m \times 2o$ elements: $\mathbf{A} = (a_{j,k}) \in \{1, \dots, 2N\}^{m \times 2o}$.

In the phase of calculating clause outputs, both limit values given by SSLs and TA decisions on their corresponding SSLs are considered. The value of k^{th} literal, $l_{j,k}$ which represents the k^{th} continuous feature inside the clause, j to perform the conjunction is evaluated as follows:

- **Condition 1:** Both $TA_{j,k}^l$ and $TA_{j,k}^u$ which respectively make the decision on $SSL_{j,k}^l$ and $SSL_{j,k}^u$ decide to include them in the clause. Then,

$$l_{j,k} = \begin{cases} 1, & \text{if} \quad SSL_{j,k}^l < x_k^c \leq SSL_{j,k}^u \ , \\ 0, & \text{if} \quad x_k^c \leq SSL_{j,k}^l \vee SSL_{j,k}^u < x_k^c \ . \end{cases} \tag{C.11}$$

- **Condition 2:** The $TA_{j,k}^l$ decides to include $SSL_{j,k}^l$ in the clause and $TA_{j,k}^u$ decides to exclude $SSL_{j,k}^u$ from the clause. Then,

$$l_{j,k} = \begin{cases} 1, & \text{if} \quad SSL_{j,k}^l < x_k^c \ , \\ 0, & \text{if} \quad x_k^c \leq SSL_{j,k}^l \ . \end{cases} \tag{C.12}$$

- **Condition 3:** The $TA_{j,k}^u$ decides to include $SSL_{j,k}^u$ in the clause and $TA_{j,k}^l$ decides to exclude $SSL_{j,k}^l$ from the clause. Then,

$$l_{j,k} = \begin{cases} 1, & \text{if} \quad x_k^c \leq SSL_{j,k}^u \ , \\ 0, & \text{if} \quad SSL_{j,k}^u < x_k^c \ . \end{cases} \tag{C.13}$$

- **Condition 4:** Both $TA_{j,k}^l$ and $TA_{j,k}^u$ decide to exclude their corresponding SSLs from the clause. Which consequently takes the lower limit to the lowest possible and upper limit to the highest possible values. Hence, $l_{j,k}$ becomes always 1, or can be excluded when conjunction is performed.

Hence, when at least one of the TAs which represent lower and upper limits decides to include its corresponding limit in the j^{th} clause, the index of the feature is included in I_j, $I_j \subseteq \{1, \ldots, o\}$. Then, depending on the literal value according to the above conditions, clause output in computed, $c_j = \bigwedge_{k \in I_j} l_k, j = 1, \ldots, m$.

Classification. Similar to the standard TM, the vote difference v is computed as $v = \sum_j c_j^+(X^c) - \sum_j c_j^-(X^c)$. Once the vote difference is known, the output class is decided using (C.6).

Learning. In the next setup, the clauses still receive Type I and Type II feedback. However, both TAs and SSLs have to be updated as feedback is received. In other words, Type I and Type II feedback should be able to guide SSLs to learn the optimal lower and upper limits of the continuous features in each clause and lead TAs to correctly decide which limits should be included or excluded in individual clauses.

As discussed earlier, **Type Ia feedback** reinforces true positive outputs of clauses by rewarding the include action of TAs when the literal value is 1. In the new setting, Type Ia feedback updates both SSLs and TAs when x_k^c is within the lower and upper boundaries, $SSL_{j,k}^l < x_k^c \leq SSL_{j,k}^u$ and when the clause output is 1 when it has to be 1 (positive clauses when $y = 1$ and negative clauses when $y = 0$). Under these conditions, the decision whether both the TAs of upper and lower bound of k^{th} feature in the j^{th} clause are to receive Type Ia feedback, $r_{j,k}^l$ and $r_{j,k}^u$, is stochastically made as follows:

$$r_{j,k}^l = \begin{cases} 1 & \text{with probability } \frac{s-1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.14}$$

$$r_{j,k}^u = \begin{cases} 1 & \text{with probability } \frac{s-1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.15}$$

The environment feedback $E_{j,k}^l$ to update $SSL_{j,k}^l$ and $E_{j,k}^u$ to update $SSL_{j,k}^u$ are 0 and 1, respectively. By doing so, we force SSLs to tighten up the boundary of the continuous feature k and include them in the clause j by reinforcing the include action of TAs. Notice that above updates are made only if the condition in (C.7) is satisfied.

Type Ib feedback activates if x_k^c is outside any of the upper or lower boundary, $x_k^c \leq SSL_{j,k}^l \vee SSL_{j,k}^u < x_k^c$ or if the clause output is 0. For the case where $x_k^c \leq SSL_{j,k}^l$ or clause output is 0 when it has to be 1, the decision on the TA of the lower bound of the k^{th} feature in the j^{th} clause to receive Type Ib feedback, $q_{j,k}^l$ is stochastically made as follows:

$$q_{j,k}^l = \begin{cases} 1 & \text{with probability } \frac{1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.16}$$

Similarly, when it violates the upper bound requirement, i.e., $SSL_{j,k}^u < x_k^c$ or if the clause output is 0 when it has to be 1, the decision to receive Type Ib feedback on the TA which represents upper bound is made as,

$$q_{j,k}^u = \begin{cases} 1 & \text{with probability } \frac{1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{C.17}$$

The environment feedback for the SSLs when the Type Ib feedback is applied are 0 and 1 for $E^l_{j,k}$ and $E^u_{j,k}$, respectively. In this way, SSLs are forced to expand the boundary and TAs are discouraged the inclusion of their respective SSLs in the clause.

Once the eligible clauses (positive clauses when $y = 0$ and negative clauses when $y = 1$) to receive **Type II feedback** are stochastically selected using (C.10), states of the individual SSLs and TAs in them are updated. The original idea of the Type II feedback is to combat false positive output of the clause. In the new updating scheme, this is achieved by expanding the boundaries of the k^{th} feature if x^c_k is outside of the boundary and including them in the clause, which then turns the clause output to 0 eventually. Hence, if $x^c_k \leq SSL^l_{j,k}$, the environment feedback on $SSL^l_{j,k}$, $E^l_{j,k}$ becomes 0 and the state of the TA which appears for $SSL^l_{j,k}$ increases one step with probability 1. Likewise, if $SSL^u_{j,k} < x^c_k$, the environment feedback on $SSL^u_{j,k}$, $E^u_{j,k}$ becomes 1 and the state of the TA which appears for $SSL^u_{j,k}$ increases one step with probability 1.

The above decisions on receiving Type Ia, Type Ib, and Type II are stored respectively in I^{Ia}, I^{Ib}, and I^{II}. The processing of the training example in the new scheme ends with the state matrix $\mathbf{A}$ of TAs being updated as, $\mathbf{A} \leftarrow \left(\left(\mathbf{A} \oplus I^{\text{Ia}}\right) \ominus I^{\text{Ib}}\right) \oplus I^{\text{II}}$ and states of SSLs being updated according to (C.1) and (C.2) with the identified environment feedback of individual SSLs, E.

C.5 Empirical Evaluation

In this section, the impact of the new continuous input feature representation to the TM is empirically evaluated using five real-world datasets[1] . The dataset *Liver Disorder dataset*, *Breast Cancer dataset*, and *Heart Disease dataset* are from the health sector. The *Balance Scale* and *Corporate Bankruptcy* datasets are the two other remaining datasets. The *Liver Disorder dataset*, *Breast Cancer dataset*, *Heart Disease dataset*, and *Corporate Bankruptcy* datasets have been selected since these applications from health sector and finance demand both interpretability and the accuracy in predictions. The *Balance Scale* dataset has been added to diversify the selected applications. Taking as an example, we use the *Corporate Bankruptcy* dataset to examine the interpretability of the TM using both the previous continuous feature representation and the proposed one. A summary of these datasets have been tabulated in Table C.4.

The performance of the TM is also contrasted against several other standard machine learning algorithms, namely, Artificial Neural Networks (ANNs), Decision Trees (DTs), Support Vector Machines (SVMs), Random Forest (RF), K-Nearest Neighbor (KNN), Explainable Boosting Machines (EBMs), [22] Gradient Boosted Trees (XGBoost) [50] along with two recent state-of-the-art machine learning approaches: StructureBoost [47] and Neural Additive Models [11]. 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. The other hyperparameters of each of these models are decided using trial and

[1] The implementation of Tsetlin Machine with versions of relevant software libraries and frameworks can be found at https://github.com/cair/TsetlinMachine.

Table C.4: Binarizing categorical features in the Bankruptcy dataset.

Dataset	Number of Instance	Number of Attributes	Interpretability needed
Corporate Bankruptcy	250	7	Yes
Balance Scale	625	4	Not necessarily
Breast Cancer	286	9	Yes
Liver Disorders	345	7	Yes
Heart Disease	270	13	Yes

Table C.5: 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

error. The reported results in this sections are the average measure over 50 independent experiment trials. The data are randomly divided into training (80%) and testing (20%) at each experiment.

C.5.1 Bankruptcy

In finance, interpretable machine learning algorithms are preferred over black-box methods to predict bankruptcy since the bankruptcy related decisions are sensitive. However, at the same time, accuracy of the predictions are also important to mitigate financial losses [51].

The Bankruptcy dataset [2] we consider in this experiment contains historical records of 250 companies. The output, i.e., Bankruptcy or Non-bankruptcy is determined by pertinent six features: 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 output "Bankruptcy" is considered as class 0 while the "Non-bankruptcy" is class 1. The features are, however, ternary. Thus, the TM has to be used with the proposed SSLs scheme to represent categorical features directly in clauses or features should be Booleanized using the Booleanization scheme before feeding them into the TM. If the features are Booleanized beforehand, each feature value can be represented in three Boolean features as shown in Table C.5. Thus, the complete Booleanized dataset contains 18 Boolean features.

First, the behavior of the TM with 10 clauses is studied. The included literals in all these 10 clauses at end training are summarized in Table C.6. In the TM with Booleanized

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

Table C.6: Clauses produced by TM with Booleanization and SSLs schemes for $m = 10$.

Clause	Class	TM with	
		Booleanized	SSLs
1	1	$\neg\, x_{11}$	-
2	0	$\neg\, x_{13} \land x_{14}$	$1 < x_5^c \leq 2$
3	1	$\neg\, x_{14}$	$2 < x_5^c$
4	0	$\neg\, x_{13} \land x_{14}$	$1 < x_5^c \leq 2$
5	1	$\neg\, x_{14}$	-
6	0	$\neg\, x_{13} \land x_{14}$	$1 < x_5^c \leq 2$
7	1	-	-
8	0	$\neg\, x_{13} \land x_{14}$	$1 < x_5^c \leq 2$
9	1	$\neg\, x_{14}$	-
10	0	$\neg\, x_{13} \land x_{14}$	$1 < x_5^c \leq 2$
Accuracy (Training/Testing)		0.98/1.00	0.99/0.96

Table C.7: Clauses produced by TM with Booleanization and SSLs schemes for $m = 2$.

Clause	Class	TM with	
		Booleanized	SSLs
1	1	$\neg\, x_{14}$	$2 < x_5^c$
2	0	$\neg\, x_{13} \land x_{14}$	$1 < x_5^c \leq 2$
Accuracy (Training/Testing)		0.99/0.96	0.96/0.98

features, the TAs in clause 1 has decided to include only the negation of feature 11, $\neg x_{11}$. Feature 11 is the *Negative Credibility* which we can find after binarizing all features. The TAs in clauses 2, 4, 6, 8, 10 have decided to include the negation of *Average Competitiveness* and *Negative Competitiveness* non-negated in clauses. The TAs in clauses 3, 5, and 9, on the other hand, have decided to include *Negative Competitiveness* negated. The clause 7 is "empty" where TAs in this clause have decided not to include any literal in the clause.

Table C.6 also contains the clauses learnt by TM when SSLs continuous feature approach is used. The clauses 2, 4, 6, 8, and 10 which vote for Bankruptcy activate for *Negative Competitiveness.* On the other hand, the clause 3, which recognizes the subpatterns of class 1 outputs 1 for *Positive Competitiveness.* There are four free votes for class 1 from the "empty" clauses 1, 5,7, and 9 which are again ignored during classification. Note also that, without loss of accuracy, TM with SSLs approach simplifies the set of rules by not including *Negative Credibility* in any of the clauses. With identified thresholds for the continuous values (categorical in this application), TM with SSLs approach ends up with the simple classification rule:

$$\text{Outcome} = \begin{cases} \text{Bankruptcy} & \textbf{if } \text{Negative Competitiveness} \\ \text{Non-bankruptcy} & \textbf{otherwise}. \end{cases} \tag{C.18}$$

By asking the TMs to utilize only two clauses, we can obtain the above rule more

Table C.8: Performance of TM with Booleanized continuous features 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 C.9: Performance of TM with SSLs continuous feature scheme on Bankruptcy dataset.

m	2	10	100	500	2000	8000
Precision	0.622	0.777	0.975	0.995	0.994	0.996
Recall	0.978	0.944	0.994	1.000	0.997	0.995
F1-Score	0.756	0.843	0.984	0.997	0.995	0.995
Accuracy	0.640	0.787	0.982	0.997	0.995	0.994
Specificity	0.191	0.568	0.967	0.994	0.993	0.993
No. of Lit.	8	40	94	398	1534	7286

directly, as shown in Table C.7. As seen, again, TM with both feature representations achieves similar accuracy.

The previous accuracy results represent the majority of experiment trials. Some experiments fail to get this optimum accuracy. Instead of conducting the experiments multiple time to find the optimum clause configuration in the TM, the number of clauses can be increased to find more robust configurations of clauses. Even though this provides stable higher accuracy for almost all the trials, large number of clauses affects the interpretability. This is where we have to think of achieving a balance between accuracy and interpretability. For the Bankruptcy dataset, how robustness increases with clauses can be seen in Table C.8 and Table C.9. Average performance (Precision, Recall, F1-Score, Accuracy, Specificity) has been summarized in tables for the TM with both feature arrangements, respectively.

Table C.8 reports the results of the TM with regular Booleanization scheme. The goal here is to maximize F1-Score, since accuracy can be misleading for imbalanced datasets. As one can notice, the F1-Score increases with clauses and peaks at $m = 2000$. To obtain this performance with the Booleanized features, the TM classifier uses 3622 (rounded to nearest integer) number of literals (include actions).

Despite the slight reduction from F1-Score, the TM with the proposed continuous feature representation reaches its best F-Score having merely 398 literals in the TM classifier. This reduction of literals is higher than 9 times and which is more significant compared to the reduction of F1-Score (0.001). At this point, the number of clauses, m equals to 500.

Table C.10: Performance comparison for Bankruptcy dataset.

	Prec.	Reca.	F1	Acc.	Spec.	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.990	1.000	0.995	0.994	0.985	-	$\approx$ 942.538KB / $\approx$ 26.64KB	0.227 sec.
ANN-2	0.995	0.997	0.996	0.995	0.993	-	$\approx$ 3476.76KB / $\approx$ 590.76KB	0.226 sec.
ANN-3	0.997	0.998	0.997	0.997	0.995	-	$\approx$ 28862.65KB / $\approx$ 1297.12KB	0.266 sec.
DT	0.988	1.000	0.993	0.993	0.985	-	$\approx$ 0.00KB / $\approx$ 0.00KB	0.003 sec.
SVM	1.000	0.989	0.994	0.994	1.000	-	$\approx$ 90.11KB / $\approx$ 0.00KB	0.001 sec.
KNN	0.998	0.991	0.995	0.994	0.998	-	$\approx$ 0.00KB / $\approx$ 286.71KB	0.001 sec.
RF	0.979	0.923	0.949	0.942	0.970	-	$\approx$ 180.22KB / $\approx$ 0.00KB	0.020 sec.
XGBoost	0.996	0.977	0.983	0.983	0.992	-	$\approx$ 4964.35KB / $\approx$ 0.00KB	0.009 sec.
EBM	0.987	1.000	0.993	0.992	0.980	-	$\approx$ 1425.40KB / $\approx$ 0.00KB	13.822 sec.
TM (Booleanized)	0.996	1.000	0.998	0.998	0.996	3622	$\approx$ 0.00KB / $\approx$ 0.00KB	0.148 sec.
TM (SSLs)	0.995	1.000	0.997	0.997	0.994	398	$\approx$ 0.00KB / $\approx$ 0.00KB	0.119 sec.

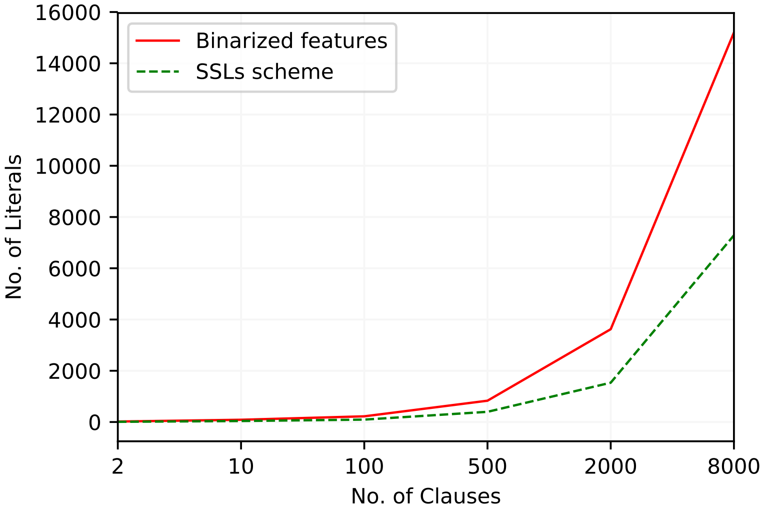


Figure C.3: The number of literals included in TM clauses to work with Bankruptcy dataset.

Figure C.3 outlines how the number of literals vary with the increase of the number of clauses. The TM with the new continuous feature representation scheme consistently uses fewer literals in its classifier than TM with regular feature representation. The difference between the number of literals by both approach increases with the number of clauses.

The performance of the TM with both continuous feature arrangements is compared against multiple standard machine learning models: namely ANN, KNN, XGBoost, DT, RF, SVM, and EBM. The performance of these techniques along with the best performance of the TM setups are summarized in Table C.10. The best F1-Score is obtained by TM with regular Booleanized features. The second best F1-Score belongs to ANN-3 and TM with SSLs scheme. Memory wise, the TM with both input feature representations together with DT need close to zero memory at both training and testing while ANN-3 requires training memory of 28862.65KB and testing memory of 1297.12KB. More importantly, the training time per epoch and the number of literals in closes are lower with the SSLs scheme for the TM than with the Booleanization approach.

C.5.2 Balance Scale

We then move to the Balance Scale dataset[3]. The Balance Scale dataset consists of three classes: balance scale tip to the left, tip to the right, or is in balance. The above class is decided collectively by 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. However, we remove the third class, i.e., "balanced", and contract the output to Boolean. The resulting dataset ends up with 576 data samples.

Table C.11 and Table C.12 contain the results of the TM with two continuous feature representations, with varying m. The F1-Score reaches its maximum of 0.945 at $m = 100$ for the TM with the Boolean feature arrangement. The average number of literals used

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

to achieve the above performance is 790. TM with SSLs scheme reaches its maximum performance when $m = 500$. The number of literals used in the classifier to achieve this permanence is 668.

The variation of the number of literals over different number of clauses in the TM with these two continuous feature arrangement is graphed in Figure C.4. The TM with SSLs scheme uses lower number of literals for all the considered number of clauses, with the difference increasing with number of clauses.

For the Balance Scale dataset, the performance of the other machine learning algorithms are also obtained. Along with the TM performance, the prediction accuracies of other models are abridged in Table C.13. The highest F1-Score from all the considered models is procured by EBM. Out of the two TM approaches, TM with SSLs scheme shows the best performance in terms of F1-Score, however, using less training time and training memory.

C.5.3 Breast Cancer

The nine features in the reast Cancer dataset[4] predicts the recurrence of breast cancer. The nine features in the dataset are: Age, Menopause, Tumor Size, Inv Nodes, Node Caps, Deg Malig, Side (left or right), the Position of the Breast, and Irradiation Status. The number of samples in the 'non-recurrence' and 'recurrence' classes are 201 and 85, respectively. However, some of these samples are removed as they content missing values

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

Table C.11: Performance of TM with Booleanized continuous features 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 C.12: Performance of TM with SSLs continuous feature scheme on Balance Scale dataset.

m	2	10	100	500	2000	8000
Precision	0.579	0.717	0.919	0.961	0.877	0.851
Recall	0.957	0.947	0.972	0.938	0.867	0.794
F1-Score	0.717	0.812	0.944	0.946	0.854	0.781
Accuracy	0.612	0.777	0.943	0.948	0.854	0.795
Specificity	0.254	0.598	0.916	0.959	0.840	0.795
No. of Lit.	4	17	140	668	2469	9563

Table C.13: Performance comparison for Balance Scale dataset.

	Prec.	Reca.	F1	Acc.	Spec.	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 (Booleanized)	0.966	0.930	0.945	0.948	0.966	790	$\approx$ 16.37KB / $\approx$ 0.00KB	0.011 sec.
TM (SSLs)	0.961	0.938	0.946	0.948	0.959	668	$\approx$ 9.43KB / $\approx$ 0.00KB	0.004 sec.

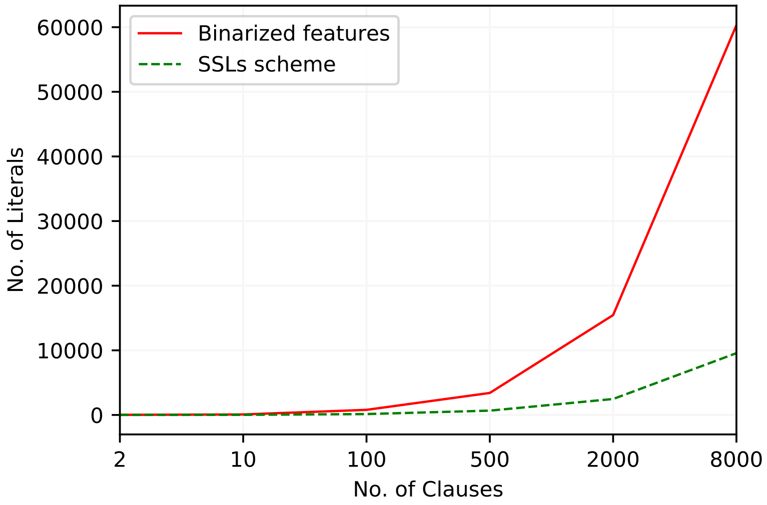


Figure C.4: The number of literals included in TM clauses to work with Balance Scale dataset.

Table C.14: Performance of TM with Booleanized continuous features 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 C.15: Performance of TM with SSLs continuous feature scheme on Breast Cancer dataset.

m	2	10	100	500	2000	8000
Precision	0.465	0.468	0.071	0.126	0.090	0.070
Recall	0.759	0.575	0.233	0.467	0.333	0.233
F1-Score	0.555	0.494	0.109	0.195	0.141	0.107
Accuracy	0.645	0.701	0.630	0.525	0.589	0.628
Specificity	0.599	0.753	0.778	0.551	0.682	0.775
No. of Lit.	4	16	101	321	997	4276

in their features.

The performance of the TMs with two feature arrangements and the number of literals they included in their clauses to achieve this performance are summarize in Table C.14 and Table C.15, respectively for the Booleanized scheme and the SSLs scheme. In contrast to the previous two datasets, the F1-Score for the TMs with both feature arrangements peaks at $m = 2$. The performance then decreases with the increase of m. The number of literals at this phase in TMs with Booleanized and SSLs feature arrangements are 21

Table C.16: Performance comparison for Breast Cancer dataset.

	Prec.	Reca.	F1	Acc.	Spec.	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.489	0.455	0.458	0.719	0.822	-	$\approx$ 1001.97KB / $\approx$ 35.74KB	0.249 sec.
ANN-2	0.430	0.398	0.403	0.683	0.792	-	$\approx$ 3498.47KB / $\approx$ 608.71KB	0.248 sec.
ANN-3	0.469	0.406	0.422	0.685	0.808	-	$\approx$ 38645.07KB / $\approx$ 1837.76KB	0.288 sec.
DT	0.415	0.222	0.276	0.706	0.915	-	$\approx$ 102.39KB / $\approx$ 0.00KB	0.005 sec.
SVM	0.428	0.364	0.384	0.678	0.805	-	$\approx$ 241.66KB / $\approx$ 299.00KB	0.001 sec.
KNN	0.535	0.423	0.458	0.755	0.871	-	$\approx$ 249.85KB / $\approx$ 61.43KB	0.001 sec.
RF	0.718	0.267	0.370	0.747	0.947	-	$\approx$ 139.26KB / $\approx$ 0.00KB	0.020 sec.
XGBoost	0.428	0.344	0.367	0.719	0.857	-	$\approx$ 1327.10KB / $\approx$ 0.00KB	0.026 sec.
EBM	0.713	0.281	0.389	0.745	0.944	-	$\approx$ 1724.41KB / $\approx$ 0.00KB	6. 007 sec.
TM (Booleanized)	0.518	0.583	0.531	0.703	0.742	21	$\approx$ 0.00KB / $\approx$ 0.00KB	0.001 sec.
TM (SSLs)	0.465	0.759	0.555	0.645	0.599	4	$\approx$ 0.00KB / $\approx$ 0.00KB	0.001 sec.

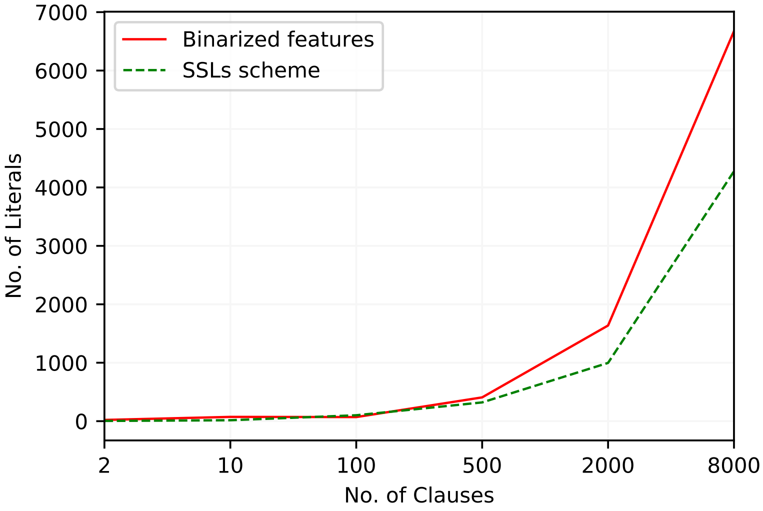


Figure C.5: The number of literals included in TM clauses to work with Breast Cancer dataset.

and 4, respectively. Overall, TM with SSLs scheme requires the least amount of literals, which we can seen in Figure C.5.

All the other algorithms get F1-Score of less than 0.5. Performance of DT, RF, SVM, XGBoost, and EBM can be identified as the worst from all models as summarized in Table C.16. The best F1-Score is obtained by TM with SSLs feature representation procedure while TM with Booleanized features obtain the second best F1-Score. The increase of F1-Score from 0.531 to 0.555 comes also with the advantage of having 19 less literals in clauses for the SSLs approach. Both training and testing memory usage of TM with these two feature arrangements are negligible. The TM also takes the lowest training time from all algorithms, amounting to 0.001 seconds.

C.5.4 Liver Disorders

The Liver Disorders dataset[5] was created in 1980s by BUPA Medical Research and Development Ltd. (hereafter "BMRDL") as a part of the larger health-screening database. The dataset contains data in seven columns: Mean Corpuscular Volume, Alkaline Phosphotase,

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

Table C.17: Performance of TM with Booleanized continuous features 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

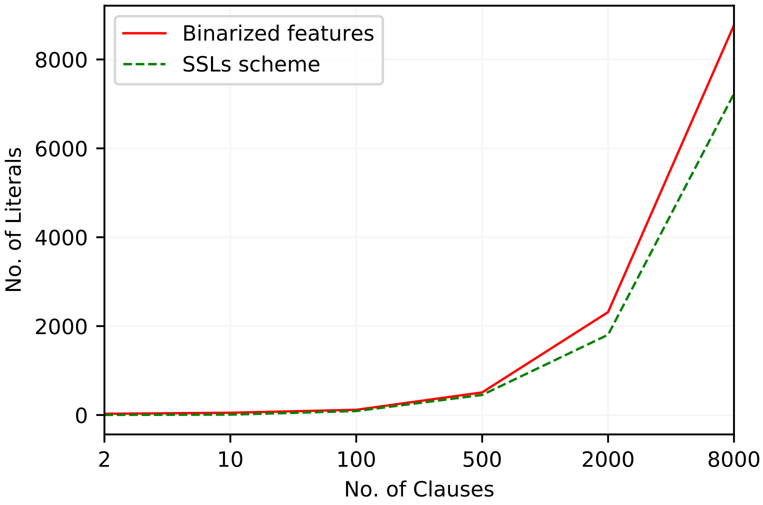


Figure C.6: The number of literals included in TM clauses to work with the Liver Disorders dataset.

Alamine Aminotransferase, Aspartate Aminotransferase, Gamma-Glutamyl Transpeptidase, Number of Half-Pint Equivalents of Alcoholic Beverages (drunk per day), and Selector. By taking the Selector attribute as class labels, some researchers have used this dataset incorrectly [52]. However in our experiments, the ”Number of Half-Pint Equivalents of Alcoholic Beverages” is used as the dependent variable, Booleanized using the threshold ≥ 3. Further, only results of various blood tests are used as feature attributes, i.e., Selector attribute is discarded.

Table C.17 and Table C.18 summarize the performance of TM with two feature arrangements. As seen, the F1-Scores of the TM with Booleanized continuous features peak at $m = 2$ while this value of TM with SSLs scheme is at $m = 10$. With 10 clauses, TM with SLLs way of representing continuous features consider merely 9 literals in clauses to acquire a better F1-Score. The increase of the number of literals included in TM clauses with the increase of the number of clauses can be seen in Figure C.6. Again, it confirms that TM with SSLs scheme uses considerably less number of literals overall.

From the performance of the other machine learning models, summarized in Table C.19, we can observe that the highest F1-Score (0.729) has been produced by RF. The performance of DT in terms of F1-Score is comparable to the performance of RF.

Table C.18: Performance of TM with SSLs continuous feature scheme on Liver Disorders dataset.

m	2	10	100	500	2000	8000
Precision	0.619	0.591	0.546	0.420	0.414	0.522
Recall	0.905	0.924	0.605	0.700	0.700	0.407
F1-Score	0.705	0.709	0.447	0.525	0.520	0.298
Accuracy	0.587	0.574	0.526	0.546	0.543	0.461
Specificity	0.101	0.098	0.400	0.300	0.300	0.600
No. of Lit.	2	9	89	452	1806	7229

Table C.19: Performance comparison for Liver Disorders dataset.

	Prec.	Reca.	F1	Acc.	Spec.	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.651	0.702	0.671	0.612	0.490	-	$\approx$ 985.13KB / $\approx$ 18.53KB	0.305 sec.
ANN-2	0.648	0.664	0.652	0.594	0.505	-	$\approx$ 3689.39KB / $\approx$ 598.26KB	0.305 sec.
ANN-3	0.650	0.670	0.656	0.602	0.508	-	$\approx$ 38365.46KB / $\approx$ 1758.23KB	0.356 sec.
DT	0.591	0.957	0.728	0.596	0.135	-	$\approx$ 49.15KB / $\approx$ 0.00KB	0.025 sec.
SVM	0.630	0.624	0.622	0.571	0.500	-	$\approx$ 1597.43KB / $\approx$ 0.00KB	0.005 sec.
KNN	0.629	0.651	0.638	0.566	0.440	-	$\approx$ 0.00KB / $\approx$ 434.17KB	0.001 sec.
RF	0.618	0.901	0.729	0.607	0.192	-	$\approx$ 0.00KB / $\approx$ 0.00KB	0.017 sec.
XGBoost	0.641	0.677	0.656	0.635	0.568	-	$\approx$ 3219.45KB / $\approx$ 0.00KB	0.081 sec.
EBM	0.641	0.804	0.710	0.629	0.406	-	$\approx$ 7790.59KB / $\approx$ 0.00KB	10.772 sec.
TM (Booleanized)	0.566	0.799	0.648	0.533	0.204	27	$\approx$ 0.00KB / $\approx$ 0.00KB	0.003 sec.
TM (SSLs)	0.591	0.924	0.709	0.574	0.098	9	$\approx$ 0.00KB / $\approx$ 0.00KB	0.001 sec.

However, DT requires training memory of 49.15 KB while RF uses negligibly small memory both on training and testing to work with the Liver Disorders dataset. The TM, on the other hand, performs better with SSLs continuous features representation than with the Booleanized continuous features. This performance is the fourth best among all the other models. For training and testing, TM with both feature representation approaches require insignificantly small amount of memory. However, TM with SSLs feature representation takes lesser time on training.

C.5.5 Heart Disease

The last dataset we use is the Heart Disease dataset[6]. The goal of this dataset is to predict the future heart disease risk based on historical data. The complete dataset consists of 75 features. However in this experiment, the updated version of the dataset, containing 13 features, is used: one ordered, six real-valued, three nominal, and three Boolean.

Table C.20 and Table C.21 summarize the performance of TM with two feature arrangement schemes. For the TM with Boolean features, the best F1-Score occurs with $m = 10$, achieved by using 346 literals on average. The F1-Score of TM with SSLs continuous features peaks again at $m = 10$ with just 42 literals. Even though, accuracy wise TM with Boolean features performs better, TM with SSLs feature representation outperforms the Boolean representation of continuous features by obtaining a higher F1-Score.

Considering the number of literals used with increasing number of clauses (Figure C.7),

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

Table C.20: Performance of TM with Booleanized continuous features 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 C.21: Performance of TM with SSLs continuous feature scheme on Heart Disease dataset.

m	2	10	100	500	2000	8000
Precision	0.529	0.588	0.562	0.305	0.674	0.687
Recall	0.971	0.915	0.504	0.431	0.660	0.667
F1-Score	0.680	0.714	0.510	0.343	0.571	0.555
Accuracy	0.591	0.674	0.709	0.630	0.633	0.581
Specificity	0.272	0.471	0.853	0.701	0.582	0.512
No. of Lit.	10	42	151	783	3152	12365

Table C.22: Performance comparison for Heart Disease dataset.

	Prec.	Reca.	F1	Acc.	Spec.	No. of Lit.	Memory Required (Training/Testing)	Training Time
ANN-1	0.764	0.724	0.738	0.772	0.811	-	$\approx$ 973.64KB / $\approx$ 16.46KB	0.297 sec.
ANN-2	0.755	0.736	0.742	0.769	0.791	-	$\approx$ 3659.59KB / $\approx$ 578.11KB	0.266 sec.
ANN-3	0.661	0.662	0.650	0.734	0.784	-	$\approx$ 33952.49KB / $\approx$ 1513.41KB	0.308 sec.
DT	0.827	0.664	0.729	0.781	0.884	-	$\approx$ 0.00KB / $\approx$ 266.23KB	0.016 sec.
SVM	0.693	0.674	0.679	0.710	0.740	-	$\approx$ 1363.96KB / $\approx$ 262.14KB	0.004 sec.
KNN	0.682	0.615	0.641	0.714	0.791	-	$\approx$ 0.00KB / $\approx$ 319.48KB	0.001 sec.
RF	0.810	0.648	0.713	0.774	0.879	-	$\approx$ 413.69KB / $\approx$ 0.00KB	0.017 sec.
XGBoost	0.712	0.696	0.701	0.788	0.863	-	$\approx$ 3694.58KB / $\approx$ 0.00KB	0.057 sec.
EBM	0.827	0.747	0.783	0.824	0.885	-	$\approx$ 4763.64KB / $\approx$ 0.00KB	11.657 sec.
TM (Booleanized)	0.607	0.815	0.687	0.672	0.566	346	$\approx$ 0.00KB / $\approx$ 0.00KB	0.014 sec.
TM (SSLs)	0.588	0.915	0.714	0.674	0.471	42	$\approx$ 0.00KB / $\approx$ 0.00KB	0.001 sec.

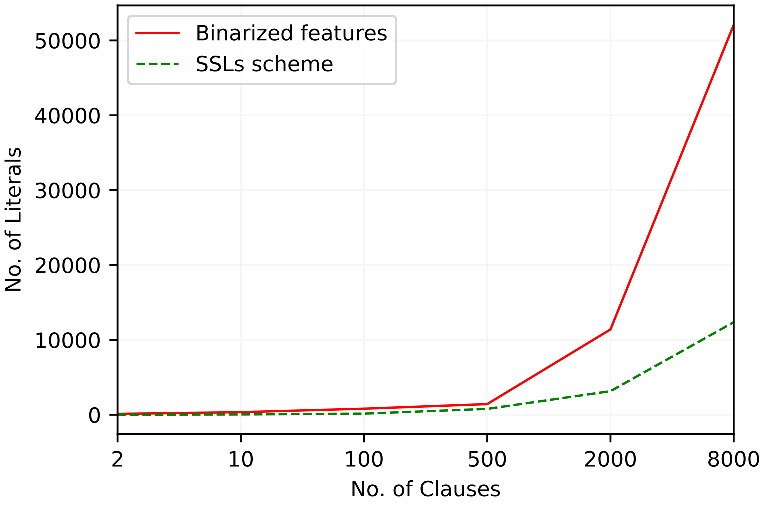


Figure C.7: The number of literals included in TM clauses to work with Heart Disease dataset.

both approaches behave almost similarly until $m = 500$, and then the TM with Booleanized features include more literals in clauses than the proposed approach.

Out of the considered machine learning models, as summarized in Table C.22 EBM obtains the best F1-Score. However, EBM needs the highest training time and uses the second largest training memory, while both TMs use negligible memory during both training and testing and consume much less training time than EBM.

C.5.6 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 TMs, 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, XGBoost, EBM, TM (Booleanized), and TM (SSLs) are 0.770, 0.757, 0.744, 0.742, 0.713, 0.737, 0.724, 0.728, 0.775, 0.762, and 0.782, respectively. Out of all the considered models, TM with SSLs continuous feature representation obtains the best average F1-Score, which is 0.782. 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.

Indeed, the average number of literals employed are 961 for TM with Booleanized continuous features and 224 for TM with SSLs feature scheme. That is, TM with SSLs feature representation uses 4.3 times fewer literals than TM with Booleanized continuous features.

The average combined memory requirement (training + testing) by TM approaches are 3.27 KB and 1.89 KB for Booleanized features and SSLs features, respectively. The combined memory usage of TM with SSLs feature representation is significantly less compared to the other models – ANN-1: $\approx$ 528 times, ANN-2: $\approx$ 2 211 times, ANN-3: $\approx$ 19 197 times, DT: $\approx$ 59 times, SVM: $\approx$ 441 times, KNN: $\approx$ 1836 times, RF: $\approx$ 78 times,

Table C.23: 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†
TM (SSLs)	0.981	0.732	0.770

†These results were extracted from graphs in [47]

XGBoost: ≈ 1 517 times, and EBM: ≈ 2 121 times.

Also note that increasing the number of clauses stabilises the Precision, Recall, FI-score, Accuracy, and Specificity measures, rendering variance insignificant. That is, variance becomes negligible for all the datasets and feature representations.

C.5.7 Comparison against recent state-of-the-art machine learning models

In this section, we compare TM 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 TM 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 [47], we use the CA weather dataset [53]. 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 [47].

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 [54] to calculate AUC. The performance characteristics are summarized in Table C.23.

Table C.23 shows that on Fraud Detection, TM with SSLs continuous feature representation approach performs on par with XGBoost and outperforms NAMs and all the other techniques mentioned in [11]. On the COMPAS dataset, TM with SSLs feature arrangement exhibits competitive performance compared to NAMs, EBM, XGBoost, and

DNNs. TM with SSLs feature representation shows, however, superior performance compared to Logistic Regression and DT on COMPAS. The performance of TM on CA-20 is better in comparison to StructureBoost, LightBoost, and CatBoost models, reported in [47].

C.6 Conclusion

In this paper, we proposed a novel continuous feature representation to the Tsetlin Machines (TMs) using the stochastic searching on the line (SSL) automata. The SSLs learn the lower and upper limits of the continuous feature values inside clauses. These limits decide the Boolean representation of the continuous value inside the clauses. We have provided empirical evidence to show that the novel way of representing continuous features in the TMs can reduce the number of literals included in the learned TM clauses 4.3 times compared to the Booleanization scheme without loss of performance. Further, the new continuous feature representation is able to decrease the total training time from 0.177 sec. to 0.126 sec. per epoch and the combined total memory usage from 16.35 KB to 9.45 KB while having on par or better performance. In terms of average F1-Score, the TM with the proposed feature representation also outperforms several state-of-the-art machine learning algorithms.

In our future work, we intend to investigate possibility of applying a similar feature representation on multi-class and regression versions of the TM.

Bibliography

[1] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. "Deep Learning for Healthcare: Review, Opportunities and Challenges". In: *Briefings in bioinformatics* 19.6 (2018), pp. 1236–1246.

[2] Riccardo Bellazzi and Blaz Zupan. "Predictive Data Mining in Clinical Medicine: Current Issues and Guidelines". In: *International journal of medical informatics* 77.2 (2008), pp. 81–97.

[3] Michael J Pazzani, S Mani, and William R Shankle. "Acceptance of Rules Generated by Machine Learning Among Medical Experts". In: *Methods of information in medicine* 40.05 (2001), pp. 380–385.

[4] Bart Baesens, Christophe Mues, Manu De Backer, Jan Vanthienen, and Rudy Setiono. "Building Intelligent Credit Scoring Systems Using Decision Tables". In: *Enterprise Information Systems V*. Springer, 2004, pp. 131–137.

[5] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. "An Empirical Evaluation of the Comprehensibility of Decision Table, Tree and Rule Based Predictive Models". In: *Decision Support Systems* 51.1 (2011), pp. 141–154.

[6] Elen Lima, Christophe Mues, and Bart Baesens. "Domain Knowledge Integration in Data Mining Using Decision Tables: Case Studies in Churn Prediction". In: *Journal of the Operational Research Society* 60.8 (2009), pp. 1096–1106.

[7] Wouter Verbeke, David Martens, Christophe Mues, and Bart Baesens. "Building Comprehensible Customer Churn Prediction Models with Advanced Rule Induction Techniques". In: *Expert systems with applications* 38.3 (2011), pp. 2354–2364.

[8] Alex A Freitas, Daniela C Wieser, and Rolf Apweiler. "On the Importance of Comprehensible Classification Models for Protein Function Prediction". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7.1 (2008), pp. 172–182.

[9] Duane Szafron, Paul Lu, Russell Greiner, David S Wishart, Brett Poulin, Roman Eisner, Zhiyong Lu, John Anvik, Cam Macdonell, Alona Fyshe, et al. "Proteome Analyst: Custom Predictions With Explanations in a Web-Based Tool for High-Throughput proteome Annotations". In: *Nucleic acids research* 32.suppl_2 (2004), W365–W371.

[10] Mehdi Ben Lazreg, Morten Goodwin, and Ole-Christoffer Granmo. “Deep Learning for Social Media Analysis in Crises Situations”. In: *The 29th Annual Workshop of the Swedish Artificial Intelligence Society (SAIS) 2–3 June 2016, Malmö, Sweden.* 2016, p. 31.

[11] Rishabh Agarwal, Nicholas Frosst, Xuezhou Zhang, Rich Caruana, and Geoffrey E Hinton. “Neural Additive Models: Interpretable Machine Learning with Neural Nets”. In: *arXiv preprint arXiv:2004.13912* (2020).

[12] Christoph Molnar. *Interpretable Machine Learning.* Lulu. com, 2019.

[13] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?” Explaining the Predictions of any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining.* 2016, pp. 1135–1144.

[14] Cynthia Rudin. “Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead”. In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215.

[15] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. “Mining Association Rules Between Sets of Items in Large Databases”. In: *SIGMOD Rec.* 22.2 (1993), pp. 207–216. ISSN: 0163-5808.

[16] Tyler McCormick, Cynthia Rudin, and David Madigan. “A Hierarchical Model for Association Rule Mining of Sequential Events: An Approach to Automated Medical Symptom Prediction”. In: *Annals of Applied Statistics* (2011).

[17] Vitaly Feldman. “Hardness of Approximate Two-Level Logic Minimization and PAC Learning with Membership Queries”. In: *Jrnl. of Computer and System Sciences* 75.1 (2009), pp. 13–26.

[18] Leslie G Valiant. “A Theory of the Learnable”. In: *Communications of the ACM* 27.11 (1984), pp. 1134–1142.

[19] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. “A Bayesian Framework for Learning Rule Sets for Interpretable Classification”. In: *The Journal of Machine Learning Research (JMLR)* 18.1 (2017), pp. 2357–2393.

[20] John R Hauser, Olivier Toubia, Theodoros Evgeniou, Rene Befurt, and Daria Dzyabura. “Disjunctions of Conjunctions, Cognitive Simplicity, and Consideration Sets”. In: *Jrnl. of Marketing Research* 47.3 (2010), pp. 485–496.

[21] Yitao Liang and Guy Van den Broeck. “Learning Logistic Circuits”. In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence.* Vol. 33. 2019, pp. 4277–4286.

[22] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. “InterpretML: A Unified Framework for Machine Learning Interpretability”. In: *arXiv preprint arXiv:1909.09223* (2019).

[23] Yin Lou, Rich Caruana, and Johannes Gehrke. "Intelligible Models for Classification and Regression". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, pp. 150–158.

[24] Ole-Christoffer Granmo. "The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic". In: *arXiv preprint arXiv:1804.01508* (2018).

[25] Adrian Phoulady, Ole-Christoffer Granmo, Saeed Rahimi Gorji, and Hady Ahmady Phoulady. "The Weighted Tsetlin Machine: Compressed Representations with Clause Weighting". In: *Ninth International Workshop on Statistical Relational AI (StarAI 2020)*. 2020.

[26] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications". In: *IEEE Access* 7 (2019), pp. 115134–115146.

[27] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. "The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression". In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

[28] Saeed Rahimi Gorji, Ole-Christoffer Granmo, Adrian Phoulady, and Morten Goodwin. "A Tsetlin Machine with Multigranular Clauses". In: *Lecture Notes in Computer Science: Proceedings of the Thirty-ninth International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI-2019)*. Vol. 11927. Springer International Publishing, 2019.

[29] Saeed Gorji, Ole Christoffer Granmo, Sondre Glimsdal, Jonathan Edwards, and Morten Goodwin. "Increasing the Inference and Learning Speed of Tsetlin Machines with Clause Indexing". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2020.

[30] Adrian Wheeldon, Rishad Shafik, Alex Yakovlev, Jonathan Edwards, Ibrahim Haddadi, and Ole-Christoffer Granmo. "Tsetlin Machine: A New Paradigm for Pervasive AI". In: *Proceedings of the SCONA Workshop at Design, Automation and Test in Europe (DATE)*. 2020.

[31] Michael Lvovitch Tsetlin. "On Behaviour of Finite Automata in Random Medium". In: *Avtomat. i Telemekh* 22.10 (1961), pp. 1345–1354.

[32] Rohan Kumar Yadav, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. "Interpretability in Word Sense Disambiguation using Tsetlin Machine". In: *13th International Conference on Agents and Artificial Intelligence (ICAART), Vienna, Austria*. INSTICC. 2021.

[33] Bimal Bhattarai, Lei Jiao, and Ole-Christoffer Granmo. "Measuring the Novelty of Natural Language Text Using the Conjunctive Clauses of a Tsetlin Machine Text Classifier". In: *13th International Conference on Agents and Artificial Intelligence (ICAART), Vienna , Austria*. INSTICC. 2021.

[34] Jivitesh Sharma, Rohan Yadav, Ole-Christoffer Granmo, and Lei Jiao. "Human Interpretable AI: Enhancing Tsetlin Machine Stochasticity with Drop Clause". In: *arXiv preprint arXiv:2105.14506* (2021). URL: https://arxiv.org/abs/2105.14506.

[35] Jie Lei, Tousif Rahman, Rishad Shafik, Adrian Wheeldon, Alex Yakovlev, Ole-Christoffer Granmo, Fahim Kawsar, and Akhil Mathur. "Low-Power Audio Keyword Spotting Using Tsetlin Machines". In: *Journal of Low Power Electronics and Applications* 11 (18 2021). URL: https://www.mdpi.com/2079-9268/11/2/18.

[36] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. "The Convolutional Tsetlin Machine". In: *arXiv preprint:1905.09688* (2019).

[37] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. "A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2019, pp. 564–578.

[38] K. Darshana Abeyrathna, O. -C. Granmo, and M. Goodwin. "Extending the Tsetlin Machine With Integer-Weighted Clauses for Increased Interpretability". In: *IEEE Access* 9 (2021), pp. 8233–8248. DOI: 10.1109/ACCESS.2021.3049569.

[39] Adrian Wheeldon, Rishad Shafik, Tousif Rahman, Jie Lei, Alex Yakovlev, and Ole-Christoffer Granmo. "Learning Automata Based Energy-efficient AI Hardware Design for IoT". In: *Philosophical Transactions of the Royal Society A* (2020). URL: https://eprints.ncl.ac.uk/268038.

[40] K. Darshana Abeyrathna, Bimal Bhattarai, Morten Goodwin, Saeed Gorji, Ole-Christoffer Granmo, Lei Jiao, Rupsa Saha, and Rohan K. Yadav. "Massively Parallel and Asynchronous Tsetlin Machine Architecture Supporting Almost Constant-Time Scaling". In: *The Thirty-eighth International Conference on Machine Learning (ICML 2021)*. ICML. 2021.

[41] Rishad Shafik, Adrian Wheeldon, and Alex Yakovlev. "Explainability and Dependability Analysis of Learning Automata based AI Hardware". In: *IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE. 2020.

[42] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "A Regression Tsetlin Machine with Integer Weighted Clauses for Compact Pattern Representation". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2020.

[43] Xuan Zhang, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. "On the Convergence of Tsetlin Machines for the IDENTITY- and NOT Operators". In: *arXiv preprint arXiv:2007.14268* (2020).

[44] Lei Jiao, Xuan Zhang, Ole-Christoffer Granmo, and K. Darshana Abeyrathna. “On the Convergence of Tsetlin Machines for the XOR Operator”. In: *arXiv preprint arXiv:2101.02547* (2021).

[45] B John Oommen. “Stochastic Searching On the Line and Its Applications to Parameter Learning in Nonlinear Optimization”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27.4 (1997), pp. 733–739.

[46] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. “Adaptive Sparse Representation of Continuous Input for Tsetlin Machines Based on Stochastic Searching on the Line”. In: *In Preparation* (2020).

[47] Brian Lucena. “StructureBoost: Efficient Gradient Boosting for Structured Categorical Variables”. In: *arXiv preprint arXiv:2007.04446* (2020).

[48] Kumpati S Narendra and Mandayam AL Thathachar. *Learning Automata: An Introduction.* Courier corporation, 2012.

[49] M A L Thathachar and P S Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization.* Kluwer Academic Publishers, 2004.

[50] Tianqi Chen and Carlos Guestrin. “Xgboost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining.* 2016, pp. 785–794.

[51] Myoung-Jong Kim and Ingoo Han. “The Discovery of Experts’ Decision Rules From Qualitative Bankruptcy Data Using Genetic Algorithms”. In: *Expert Systems with Applications* 25.4 (2003), pp. 637–646.

[52] James McDermott and Richard S Forsyth. “Diagnosing a Disorder in a Classification Benchmark”. In: *Pattern Recognition Letters* 73 (2016), pp. 41–43.

[53] Brian Lucena. “Exploiting Categorical Structure Using Tree-Based Methods”. In: *arXiv preprint arXiv:2004.07383* (2020).

[54] K. Darshana Abeyrathna, O. -C. Granmo, and M. Goodwin. “On Obtaining Classification Confidence, Ranked Predictions and AUC with Tsetlin Machines”. In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI).* 2020, pp. 662–669. DOI: 10.1109/SSCI47803.2020.9308460.

Paper D

The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression

Relying simply on bitwise operators, the recently introduced Tsetlin Machine *(TM) has provided competitive pattern classification accuracy in several benchmarks, including text understanding. In this paper, we introduce the* Regression Tsetlin Machine *(RTM), a new class of TMs designed for continuous input and output, targeting non-linear regression problems. In all brevity, we convert continuous input into a binary representation based on thresholding, and transform the propositional formula formed by the TM into an aggregated continuous output. Our empirical comparison of the RTM with state-of-the-art regression techniques reveals either superior or on par performance on five datasets.*

D.1 Introduction

Over the last few years, there has been tremendous progress in research on predictive models, resulting in increasingly higher predictive accuracy. However, high predictive accuracy is not always sufficient to trust a predictive model. For applications where humans are making the final decision, the rationale *behind* a prediction can sometimes be essential to the decision making process. Therefore, human interpretability of results is of great importance [1]. This importance has been emphasized in many application domains such as credit scoring [2, 3], medicine [4, 5], bioinformatics [6, 7], and churn prediction [8, 9].

The Tsetlin Machine (TM) is a recent pattern recognition approach that attempts to bridge the gap between the interpretability of rule-based techniques and the high predictive accuracy of deep learning [10].

The TM is relatively simple computationally, being based on propositional logic to form the classifier, and straightforward bitwise operations for recognition and learning. This structure makes the TM interpretable, yet it achieves competitive accuracy for many pattern recognition problems.

The inputs and output of the TM are propositional variables, represented as bits. In the first TM layer, the inputs and their negations, referred to as literals, are connected by conjunctions to form so-called *conjunctive clauses*. In the second layer, the output of the TM is decided by a majority vote among the clauses that have been formed in layer one. For learning patterns, each clause is assigned one Tsetlin Automaton (TA) [11] per literal. A literal takes part in a clause if the designated TA decides to include it. To make this decision, the TA interacts with its environment (the rest of the Tsetlin Machine) in an iterative manner. It decides the next action based on its current state, which, in turn, is influenced by the feedback it receives. Feedback in the TM is governed by a novel game, with each TA trying to learn the optimal action after a series of interactions with the environment [12].

The use of TAs to solve complex problems has a long history. Illustrative work includes forecasting disease outbreaks [13], graph coloring [14], distributed coordination [15], stochastic searching on the line [16], and resource allocation [17]. Despite its simplicity, the TM has provided competitive results in comparison with traditional machine learning techniques, including Multilayer Perceptrons, Support Vector Machines, Logistic Regression, and Naïve Bayes, in well-known benchmarks such as handwritten digits classification (MNIST), Iris data classification, and classification of Noisy XOR data with non-informative features [10]. Recently, Berge et al. also studied the ability of the TM to categorize natural language text simply based on a document word presence vector [18]. They further investigated the interpretability of the produced clauses, for analyzing electronic health records. This work demonstrated that the TM in some cases can outperform vanilla deep learning techniques, including Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) Neural Networks, while keeping the important property of interpretability.

Paper Contributions: The classic TM has been designed for binary inputs and output. Although continuous input and output can be encoded in bit form, the natural ordering of numbers is lost. We address this limitation in the present paper by introducing the Regression Tsetlin Machine (RTM), which consists of three parts:

1. For continuous input, we propose a data preprocessing procedure that transforms the input losslessly into a binary representation that maintains semantic relationship between numbers. In brief, the preprocessing procedure considers each unique data sample in each continuous feature as a potential threshold. The original data samples are then compared with the complete set of thresholds to create a new feature matrix that only contains bits.

2. To produce continuous output that leverages the natural ordering of numbers, we modify the inner inference mechanism of the TM. That is, the inputs are transformed into a single continuous output, rather than to distinct categories. We achieve this by eliminating the polarities of the clauses, and by summing all the non-polarized votes, mapping the sum into a continuous output.

3. Finally, we propose a new feedback scheme for guiding the TA of the RTM, to support regression. This scheme minimizes the discrepancy between the predicted and the target outputs, with the aid of a modified stochastic clause activation function.

Paper Organization: The remainder of the paper is organized as follows. In Section 2, we discuss the structure and inference mechanisms of the TM, with a particular focus on the parts of the architecture that we build upon. Then, in Section 3 and 4, we present the main contribution of this paper, which is the data preprocessing and regression procedures, leading to the RTM. The RTM is analysed empirically in Section 5 by the help of both artificial and real-world datasets, in comparison with selected state-of-the-art regression techniques. We conclude our work in Section 6.

D.2 The Tsetlin Machine (TM)

In classification problems, a class can be represented by its constituting sub-patterns. The TM has been designed to capture these sub-patterns explicitly by operating upon a series of conjunctive clauses, how many decided by the user. The structure of the TM as well as the procedures for recognition and learning are described in the following.

TM structure: Consider an input feature vector $\mathbf{X} = (x_k) \in \{0,1\}^o$ consisting of o propositional variables x_k with domain $\{0,1\}$. The TM considers both the features x_k themselves as well as their negations $\neg x_k$ (the literals) when forming the clauses. In all brevity, each clause j takes the following form:

$$c_j = 1 \land \left(\bigwedge_{k \in I_j^I} x_k\right) \land \left(\bigwedge_{k \in \bar{I}_j^I} \neg x_k\right). \tag{D.1}$$

Above, I_j^I and $\bar{I}_j^I$ are non-overlapping subsets of the input variable indexes, $I_j^I, \bar{I}_j^I \subseteq \{1,\ldots,o\}$, $I_j^I \cap \bar{I}_j^I = \emptyset$, that specify which of the literals take part in the clause j. The indexes of the included original features are found in I_j^I while the indexes of the included negated features are found in $\bar{I}_j^I$. The upper index I signals that the literals in the subsets are *included.* Conversely, I_j^E and $\bar{I}_j^E$ specify the literals that are *excluded.*

To simplify notation, we now introduce an augmented feature vector $\mathbf{X}'$, which can be written as $\mathbf{X}' = [x_1, x_2, x_3, \ldots, x_o, \neg x_1, \neg x_2, \neg x_3, \ldots, \neg x_o]$, after concatenating the original features with the negated features. The elements of the augmented feature vector x'_k are now the literals, and we then only need to consider I_j^I and I_j^E, however, for the expanded feature index set $\{1,\ldots,2o\}$.

The TM employs two-action TAs to decide which feature indexes $1,\ldots,2o$ go into I_j^I, with one team of TAs per clause j. That is, with o features, we need $2 \times o$ TAs per clause. Half of them represent the original features and the remaining represent the negated features.

The two actions available to each TA are *include* and *exclude.* Here, *include* refers to including the literal assigned to the TA and *exclude* means excluding it. The action that the TA performs is decided by its current state, $a_{j,k} \in \{1,\ldots,2N\}$. If the state is less than or equal to N, the literal is excluded from the clause. Therefore, the subset of indexes of the excluded literals can be stated as, $I_j^E = \{k | a_{j,k} \leq N, 1 \leq k \leq 2o\}$. Oppositely, if the state is higher than N, the corresponding literal is inserted in the clause. The subset of the indexes of the included literals can be then written as $I_j^I = \{k | a_{j,k} > N, 1 \leq k \leq 2o\}$.

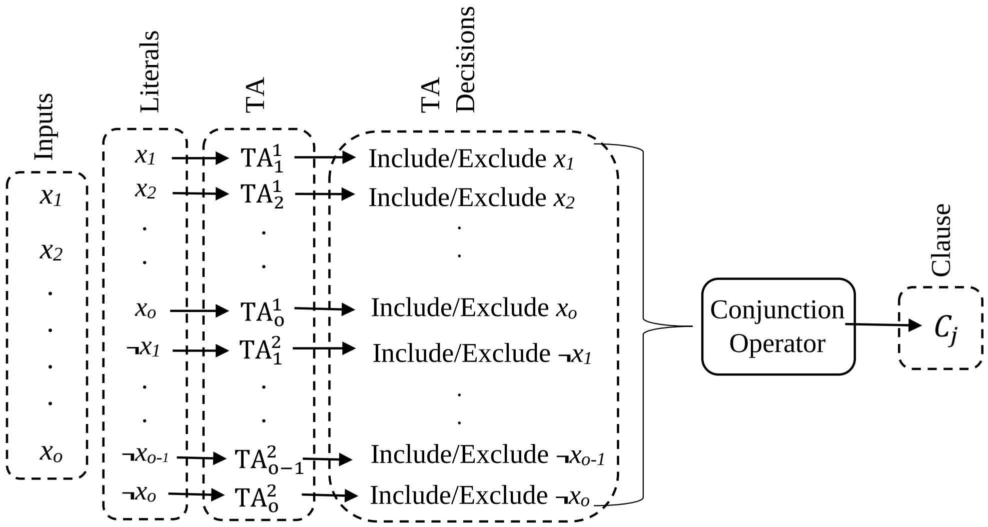


Figure D.1: Forming a clause using input features and the actions of the TAs.

All these states $a_{j,k}$ of all the clauses are organized in an $m \times 2o$ matrix $\mathbf{A}$: $\mathbf{A} = (a_{j,k}) \in \{1, \ldots, 2N\}^{m \times 2o}$.

Figure D.1 illustrates the above described TM structure. In the figure, the upper index t of TA_k^t denotes the type of the feature (1 for the original and 2 for the negated) while the lower index k denotes the original feature index.

Clause output: Since the TM clause is a conjunction of literals, including any literal of value 0, even a single one, will make the clause output 0. Let the index set $I_{X'}^1$ be the indexes k of all of the literals of value $x'_k = 1$ in the input $\mathbf{X}'$. Accordingly, a clause evaluates to 1 if and only if the indexes of the included literals I_j^I is a subset of the indexes $I_{X'}^1$ of literals of value 1:

$$c_j = \begin{cases} 1 & \text{if} \quad I_j^I \subseteq I_{X'}^1, \\ 0 & \text{otherwise.} \end{cases} \tag{D.2}$$

Clause voting: The above clause outputs are then organized in a vector $\mathbf{C} = (c_j) \in \{0, 1\}^m$. To increase the expressiveness of the TM, clauses are assigned positive or negative polarity. Clauses with positive polarity (C^+) vote for the class $y = 1$ and clauses with negative polarity (C^-) vote for the class $y = 0$. To avoid bias, the clauses are equally divided among class 0 and class 1. E.g., clauses with odd indexes can be assigned positive polarity and clauses with even indexes can be assigned negative polarity. The TA state matrix $\mathbf{A}$ then has two separate sections: $\mathbf{A}^+$ and $\mathbf{A}^-$. The section $\mathbf{A}^+$ maintains the states of TAs of positive clauses: $\mathbf{A}^+ = (a_{j,k}^+) \in \{1, \ldots, 2N\}^{\frac{m}{2} \times 2o}$, representing the odd rows of matrix $\mathbf{A}$. The section $\mathbf{A}^-$ maintains the states of TAs of negative clauses: $\mathbf{A}^- = (a_{j,k}^-) \in \{1, \ldots, 2N\}^{\frac{m}{2} \times 2o}$, representing the even rows of matrix $\mathbf{A}$.

The resulting two-class architecture is illustrated in Figure D.2a. The summation operator at the end sums the votes for each class separately and considers the difference: $v = \sum_j c_j^+ - \sum_j c_j^-$. At last, the final output is decided as in (D.3).

$$y = \begin{cases} 1 & \text{if} \quad v \geq 0 \\ 0 & \text{if} \quad v < 0\,. \end{cases} \tag{D.3}$$

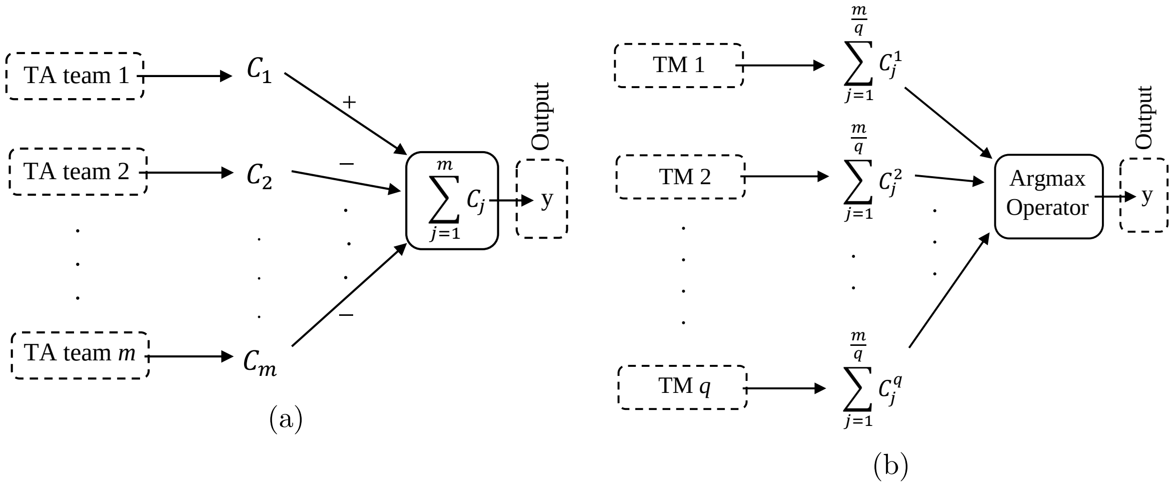


Figure D.2: (a) The basic TM for two class problems. (b) Multi-class version of the TM.

Note that for categorization tasks with more classes than two, separate TMs are needed for each class as shown in Figure D.2b. In such situations, clauses are partitioned equally among the classes. Clauses with positive polarity vote in favor of the considered class, and clauses with negative polarity vote against the considered class. In the end, an *argmax* operator arbitrates the class predicted for the input, based on the votes collected for each class. When there are q classes and v^i is the difference between positive and negative votes for class i, the output, y, can be expressed as:

$$y = \text{argmax}_{i=1,\ldots,q}\{v^i\}. \tag{D.4}$$

Learning procedure: The conjunctive clauses of the TM are formed by the decisions made by the collective of TAs associated with all of the clauses. Hence, training in the TM entails careful guiding of the TAs so that they as a collective make the optimal *include* and *exclude* decisions. We achieve this by reinforcement learning, organized as a game between the TAs. In all brevity, the TAs adjust their states based on feedback from the game (environment), encompassing rewards, penalties, and inaction feedback. To achieve the learning goal, the probabilities of receiving these different types of feedback have been designed to account for critical factors, namely, the actual output, the clause outputs, the literal values, and current state of the TAs.

In contrast to gradient-based learning, learning in TMs naturally combats false positives and false negatives. As a consequence, it eludes the issues attached to gradient-based algorithms such as vanishing/exploding gradients. In the case of categorization problems with two classes, the basic idea is to penalize voters when they vote to procure a false positive or false negative and to reward voters when they vote to procure a true positive or true negative. In the TM, this is done by two types of feedback – Type I and Type II.

Type I feedback: The clauses that receive Type I feedback is decided stochastically using the following conditions:

$$p_{j,y} = \begin{cases} 1 & \text{with probability } \frac{T-\max(-T,\min(T,v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{D.5}$$

Here, the lower indexes j and y of p represent the clause and target class, respectively. A user defined target T decides the robustness of the learning. Higher T makes more clauses get involved in learning each sub-pattern, diversifying the representation. However, increasing T also demands more clauses, increasing computational complexity.

Type I feedback is given to clauses with positive polarity when the actual output $\hat{y}$ is 1 and to clauses with negative polarity when the actual output $\hat{y}$ is 0. Therefore, the complete matrix, $\mathbf{P}_y$, that picks the clauses for feedback Type I, is a combination of $\mathbf{P}_1^+$ and $\mathbf{P}_0^-$ where $\mathbf{P}_1^+ = (p_{j,1}^+) \in \{0,1\}^{\frac{m}{2}}$ and $\mathbf{P}_0^- = (p_{j,0}^-) \in \{0,1\}^{\frac{m}{2}}$. Type I feedback does two jobs. Type Ia reinforces *include* actions of TAs whose corresponding literal value is 1 when the clause output is 1. Type Ib combats over-fitting by reinforcing *exclude* actions of TAs when the corresponding literal is 0 or when the clause output is 0. Both Type Ia and Type Ib feedback are provided to TAs stochastically using a user set parameter s (s ≥ 1).

Type Ia feedback: The matrix $\mathbf{R}$, i.e., $\mathbf{R} = (r_{j,k}) \in \{0,1\}^{m\times 2o}$ stores the decisions whether the k^{th} TA of the j^{th} clause has been selected as candidate for feedback Type Ia. The decisions are made stochastically as follows:

$$r_{j,k} = \begin{cases} 1 & \text{with probability } \frac{s-1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{D.6}$$

Now, using the complete set of conditions, the indexes I_C^{Ia} of the TAs which receive Type Ia feedback can be identified as $I_C^{\text{Ia}} = \{(j,k)|x'_k = 1 \land c_j = 1 \land p_{j,y} = 1 \land r_{j,k} = 1\}$.

Type Ib feedback: Similarly, candidates for receiving Type Ib feedback are decided stochastically and decisions are stored in matrix $\mathbf{Q}$, i.e., $\mathbf{Q} = (q_{j,k}) \in \{0,1\}^{m\times 2o}$. In this situation, the decision for the k^{th} TA of the j^{th} clause is

$$q_{j,k} = \begin{cases} 1 & \text{with probability } \frac{1}{s}, \\ 0 & \text{otherwise.} \end{cases} \tag{D.7}$$

Again, considering all other conditions, the indexes I_C^{Ib} of the TAs that receive feedback Type Ib are expressed as $I_C^{\text{Ib}} = \{(j,k)|(x'_k = 0 \lor c_j = 0) \land p_{j,y} = 1 \land q_{j,k} = 1\}$.

The identified TAs are updated by changing their internal state. The available update operations are $\oplus$ and $\ominus$. The operator $\oplus$ adds 1 to the current state while operator $\ominus$ subtracts 1 from the current state within the given state space: $\mathbf{A} \leftarrow \left(\mathbf{A} \oplus I_C^{\text{Ia}}\right) \ominus I_C^{\text{Ib}}$.

In conclusion, Type I feedback stimulates positive clauses to output 1 when the actual output $\hat{y}$ is 1, while it stimulates negative clauses to output 1 when the actual output $\hat{y}$ is 0. Hence, Type I feedback reinforces true positive output.

Type II feedback: Type II feedback is given to clauses with positive polarity when the actual output $\hat{y}$ is 0 and clauses with negative polarity when the actual output $\hat{y}$ is 1. Whether the j^{th} clause for target output $\hat{y}$ receives Type II feedback is decided as follows:

$$p_{j,y} = \begin{cases} 1 & \text{with probability } \frac{T+\max(-T,\min(T,v))}{2T}, \\ 0 & \text{otherwise.} \end{cases} \tag{D.8}$$

Similar to Type I feedback, T is a user decided voting target, and decisions are stored in the matrix $\mathbf{P}_y$. The idea is to alter the clause output of the clauses which output 1. This can be done by simply including literals of value 0. The TAs that match the above conditions can be identified as $I_C^{\text{II}} = \{(j,k)|x'_k = 0 \land c_j = 1 \land p_{j,y} = 1\}$. The identified TAs are updated by changing their internal state: $\mathbf{A} \leftarrow \mathbf{A} \oplus I_C^{\text{II}}$.

In this way, Type II feedback stimulates clauses to output 0. Type II feedback is applied to clauses of positive polarity when the actual output $\hat{y}$ is 0 and to clauses of negative polarity when the actual output $\hat{y}$ is 1. In this manner, it combats false positives.

The training procedure of the multi-class version of the TM is similar to the above training procedure. Clauses of the class being the target of the current training sample are treated as if $\hat{y} = 1$, while the clauses of a randomly selected class from the remaining classes are treated as if $\hat{y} = 0$. In each class, clauses with positive polarity vote to say that the output belongs to the considered class. Similarly, the clauses with negative polarity vote to indicate that the output does not belong to the considered class.

D.3 An Encoding Scheme for Continuous Input to the Tsetlin Machine

The input feature vector of the TM that we presented in the previous section only had binary features, i.e., $\boldsymbol{X} = [x_1, x_2, x_3, \ldots, x_o]$ with $x_k, k = 1, 2, ..., o$, being 0 or 1. Clauses are directly composed by these features or their negations, as decided by the TAs. However, in many real-world applications, one cannot necessarily expect binary features. Instead, the input features are often continuous valued. In order to cope with such applications, we introduce a preprocessing procedure that transforms continuous features into binary ones, while maintaining ranking relationships among the continuous feature values.

By way of example, Table D.1 shows how two continuous features are binarized. As seen, the two features are respectively listed in table column 1 and column 2. The preprocessing procedure follows the following steps to convert them into binary form, one feature at a time.

1. Identify the unique values $\{v_1, v_2, \ldots, v_u\}$ of the feature.

2. Sort the identified unique values from smallest to largest.

3. Consider each unique value $v_i, i = 1, 2, ..., u$, as a threshold "$\leq v_i$", as shown in sorted order in the "Thresholds" row in the table.

4. Compare each original continuous value in the feature with each of the sorted thresholds. If the feature value is greater than the threshold, set the corresponding binary variable to 0, otherwise, set the bit to 1.

5. Repeat steps (i) to (iv) until all the continuous values are converted into binary form.

According to above the procedure, the first feature in the given example has three unique values, i.e., 5.779, 10.008, and 3.834 (step (i)). These unique values are sorted and considered as thresholds, $\leq$ 3.834, $\leq$ 5.779, and $\leq$ 10.008 (step (ii) and (iii)). Notice that the number of binary feature bits we are going to have after converting the continuous

Table D.1: Preprocessing of two continuous features.

Raw Feature		Thresholds						
1	2	≤ 3.834	≤ 5.779	≤ 10.008	≤ 11.6	≤ 25.7	≤ 32.4	≤ 56.1
5.779	25.7	0	1	1	0	1	1	1
10.008	56.1	0	0	1	0	0	0	1
5.779	11.6	0	1	1	1	1	1	1
3.834	32.4	1	1	1	0	0	1	1

inputs equals the number of thresholds. In this case, for Feature 1 in the table (first column), three new binary feature bits are introduced, each of which corresponds to its respective threshold. If we compare the first raw continuous value of Feature 1, namely, 5.779, with the identified three thresholds, we get a 011 as the raw value is greater than 3.834 (resulting in 0), equal to 5.779 (resulting in 1), and less than 10.008 (resulting in 1) (Step iv). Similarly, the remaining raw continuous values, 10.008 and 3.834, are converted into 001 and 111, respectively.

Once all the raw continuous values of the first feature are converted, conversion of the second feature can start. This procedure is iterated until all the continuous values in all the continuous features have been converted to binary form (step (v)).

Note that the thresholds found in the above procedure are open-ended at the extreme ends. Thus, arbitrarily low or high feature values can be processed, independent of the actual values in the training data.

The new binary representation of continuous features becomes particularly powerful because it allows the TM to reason about the ordering of the values, forming conjunctive clauses that specify rules based on thresholds, and with negated features, also rules based on intervals.

D.4 The Regression Tsetlin Machine (RTM)

For many real-world prediction problems, the target y is continuous valued, referred to as regression problems. Although continuous output can be encoded in bit form, the inherent properties of continuous values are lost, such as the ordering of values. Thus, the binary output of the classic TM shown in Figure D.2a and Figure D.2b is not ideal for representing continuous output. In order to address this problem, we here propose a new type of TM — the *Regression Tsetlin Machine (RTM).*

The structure of the RTM is illustrated in Figure D.3. As the figure shows, we remove the polarity of the clauses in the vanilla TM structure. Instead, we use clauses as additive building blocks to calculate the continuous outputs. That is, the summation operator in Figure D.3 counts the total number of clauses that evaluate to 1, and the resulting sum is further mapped into a continuous value according to Eq. (D.9):

$$y = \frac{\sum_{j=1}^{m} C_j(\hat{X})}{T} \times \hat{y}_{\max}. \tag{D.9}$$

Above, T is a user specified output resolution and $\hat{y}_{\max}$ is the maximum value of the target

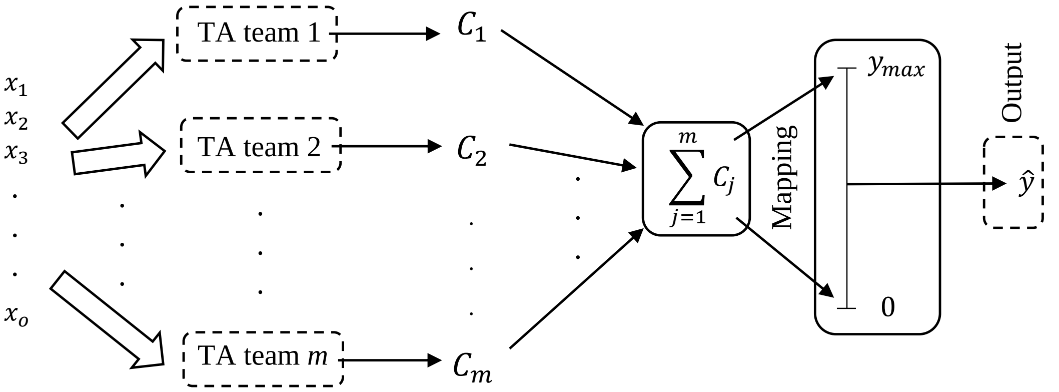


Figure D.3: The RTM structure.

output found in the training data, assuming positive output[1]. *This maximum value is merely a reference point, to establish a scale, and should not be considered as an upper bound for the output.*

Once a continuous output value y has been calculated, it is compared with the target output $\hat{y}$. Following the convention of regression analysis, we refer to the difference between the output y and $\hat{y}$ as the error of the prediction. The output itself and the error are, in turn, fed back to the RTM, and influence the decision made by the TAs so that the error can be minimized. The goal is for the RTM to learn patterns appropriate for regression from the training dataset.

Feedback for the RTM is derived slightly differently compared to the vanilla TM. That is, whether it is given Type I or Type II feedback is based on the difference between y and $\hat{y}$, as determined by Eq. (D.10):

$$Feedback = \begin{cases} \text{Type I,} & \text{if} \quad y < \hat{y}\ , \\ \\ \text{Type II,} & \text{if} \quad y > \hat{y}\ . \end{cases} \tag{D.10}$$

The rationale behind Eq. (D.10) is as follows. We want to increase the number of clauses that output 1 when the predicted output is less than the target output ($y < \hat{y}$). To achieve this, Type I feedback is introduced. Conversely, Type II feedback is applied to decrease the number of clauses that evaluate to 1 when the predicted output is higher than the target output ($y > \hat{y}$).

If the feedback determined by (D.10) is offered to all the clauses, the predicted value will keep oscillating around the target output throughout the training process. Hence, we randomly select a subset of the clauses to receive feedback, with the prediction error controlling the cardinality of the subset. We achieve this by introducing the following feedback activation probability function, P_{act}, by which the RTM decides stochastically which clauses to be updated:

$$P_{act} = \frac{K\times \mid y - \hat{y} \mid}{\hat{y}_{\max}}\ . \tag{D.11}$$

[1] Our scheme also supports negative output values by employing clauses of negative polarity.

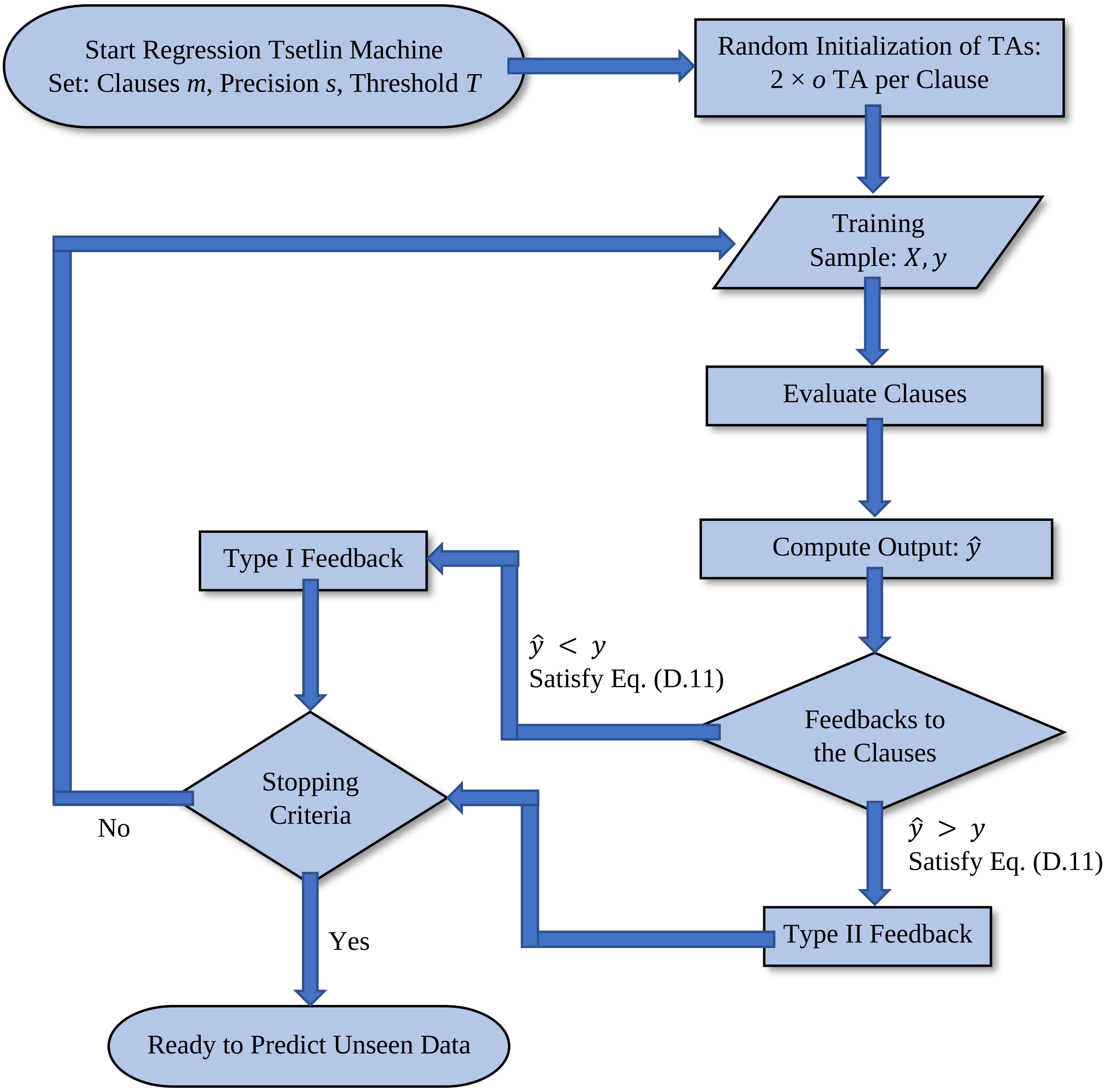


Figure D.4: The training work-flow of the RTM.

Note that in Eq. (D.11), the parameter K is added to adjust the activation probability according to the total number of clauses. In other words, the probability varies proportionally to the error, and is reasonably scaled to fit the total number of clauses.

The flowchart given in Figure D.4 summarizes the complete training procedure of the RTM, including the data preprocessing step introduced in Section D.3.

D.5 Empirical Results and Analysis

We now study the properties of the RTM empirically, by means of two artificial and five real-world datasets.[2]

[2] The artificial datasets and RTM code are available to download at https://github.com/cair/regression-tsetlin-machine

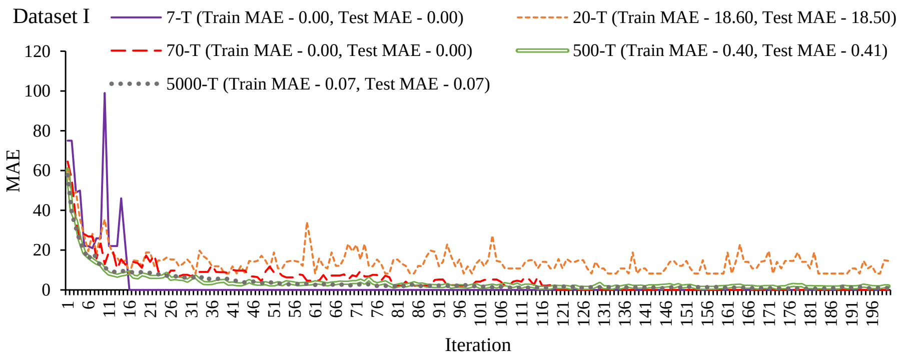


Figure D.5: Training error per epoch for Dataset I. The dataset is processed with different T.

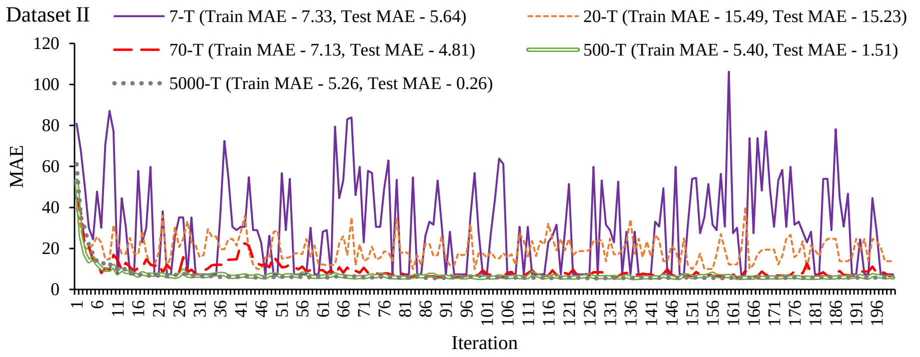


Figure D.6: Training error per epoch for Dataset II. The dataset is processed with different T.

D.5.1 Artificial Datasets

The two artificial datasets have been designed to demonstrate how the RTM form non-linear clauses, which in turn are used as additive components in the regression. The first dataset, Dataset I, has 8 distinct outputs, each of which is triggered by a specific 3-bit input. Each input bit has an equal probability of being set to either 0 or 1, leading to an uniform distribution of bit values. The continuous output is quite simply calculated by multiplying 100 with the decimal representation of the input. For example, the continuous output for input (011) becomes $100 \times 3 = 300$. The complete dataset consists of $10,000$ samples, split into 80% training data and 20% test data. The second dataset, Dataset II, is built in the same way, except that the output of the training data is perturbed by adding a random value in the range $[-10, 10]$ to introduce noise.

We train and test the RTM separately on the above two datasets, with Mean Absolute Error (MAE) utilized as evaluation criterion. For all of our experiments on artificial data we use a specificity value s of 2. Figure D.5 and Figure D.6 depict training MAE across 1000 epochs, examining the effect of resolution T. The final training and test MAEs, i.e., the ones obtained after 1000 epochs, are provided in the legends. As seen in Figure D.5, noise-free data can be perfectly learnt by an RTM with merely resolution 7, utilizing 7 clauses. However, in general, as seen in Figure D.6, higher resolution progressively reduces MAE under noise.

Let us now study the case with 7 clauses and noise-free data more carefully to cast further light on how the RTM learns. With 3 uniformly distributed bits, there are 8 unique sub-patterns, illustrated in Figure D.7. Each of these occurs with probability

Table D.2: Computing outputs for different 3 bit inputs.

Input	Represented Patterns	Output Computation	Output
000	None	$4 \times (1 \ast \ast) \times 0 = 0$ $2 \times (\ast\ 1 \ast) \times 0 = 0$ $1 \times (\ast \ast\ 1) \times 0 = 0$	$0 \times 100 = 0$
001	$(\ast \ast\ 1)$	$4 \times (1 \ast \ast) \times 0 = 0$ $2 \times (\ast\ 1 \ast) \times 0 = 0$ $1 \times (\ast \ast\ 1) \times 1 = 1$	$1 \times 100 = 100$
010	$(\ast\ 1 \ast)$	$4 \times (1 \ast \ast) \times 0 = 0$ $2 \times (\ast\ 1 \ast) \times 1 = 2$ $1 \times (\ast \ast\ 1) \times 0 = 0$	$2 \times 100 = 200$
011	$(\ast\ 1 \ast)$ $(\ast \ast\ 1)$	$4 \times (1 \ast \ast) \times 0 = 0$ $2 \times (\ast\ 1 \ast) \times 1 = 2$ $1 \times (\ast \ast\ 1) \times 1 = 1$	$3 \times 100 = 300$
100	$(1 \ast \ast)$	$4 \times (1 \ast \ast) \times 1 = 4$ $2 \times (\ast\ 1 \ast) \times 0 = 0$ $1 \times (\ast \ast\ 1) \times 0 = 0$	$4 \times 100 = 400$
101	$(1 \ast \ast)$ $(\ast \ast\ 1)$	$4 \times (1 \ast \ast) \times 1 = 4$ $2 \times (\ast\ 1 \ast) \times 0 = 0$ $1 \times (\ast \ast\ 1) \times 1 = 1$	$5 \times 100 = 500$
110	$(1 \ast \ast)$ $(\ast\ 1 \ast)$	$4 \times (1 \ast \ast) \times 1 = 4$ $2 \times (\ast\ 1 \ast) \times 1 = 2$ $1 \times (\ast \ast\ 1) \times 0 = 0$	$6 \times 100 = 600$
111	$(1 \ast \ast)$ $(\ast\ 1 \ast)$ $(\ast \ast\ 1)$	$4 \times (1 \ast \ast) \times 1 = 4$ $2 \times (\ast\ 1 \ast) \times 1 = 2$ $1 \times (\ast \ast\ 1) \times 1 = 1$	$7 \times 100 = 700$

$\frac{1}{8}$. The shaded area in the figure contains the pattern $(1 \ast \ast)$, whose probability of occurrence is $\frac{4}{8}$. Here, $\ast$ means either 0 or 1. Similarly, the probabilities of occurrence for both the pattern $(\ast\ 1 \ast)$ and the pattern $(\ast \ast\ 1)$ is $\frac{4}{8}$, too. According to the Tsetlin Machine dynamics explained in [10], in order to capture these patterns, specificity should be set to $s = 2$.

With only 7 clauses available, the clauses need to encode the input in a very compact way, shown in Table D.2, to produce the correct output. As seen, we need four clauses for the pattern $(1 \ast \ast)$, two clauses for the pattern $(\ast\ 1 \ast)$, and one clause for the pattern $(\ast \ast\ 1)$. Now, consider input (0 1 1) as an example. This input matches both pattern $(\ast\ 1 \ast)$ and pattern $(\ast \ast\ 1)$. Therefore, it activates three clauses from the table. Consequently, after scaling, output 300 is correctly computed from (0 1 1). In this manner, the correct output can be calculated for all 3-bit inputs from Table D.2. Returning to Figure D.5, we notice that the MAE of the RTM is exactly 0 for 7 clauses/resolution 7, meaning that the RTM has found the structure of Table D.2 by itself. Hence, the power of the scheme!

For real-world cases, however, the exact number of clauses required is generally un-

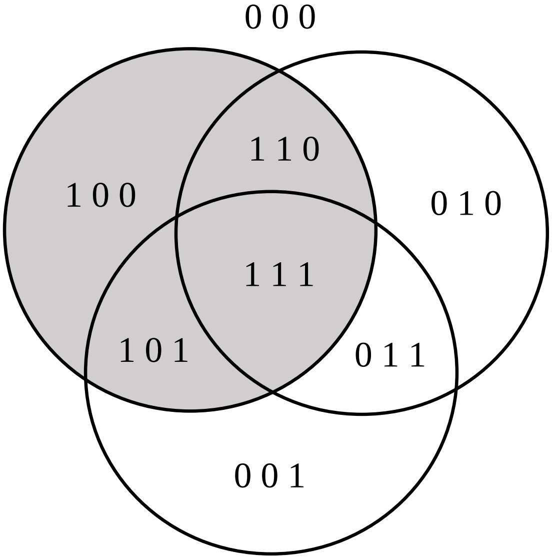


Figure D.7: Pattern distribution for the 3-bits input datasets.

known. Then, selecting an inappropriate low resolution may lead to a high MAE. This is seen in Figure D.5 for $T = 20$. Then one must instead resort to increasing the resolution to a sufficient level, which also helps with dealing with noisy data. That is, as seen, both training and testing MAEs drops with $T = 500$ and $T = 5000$ for Dataset I. The latter reasoning also applies to the results for Dataset II (see Figure D.6). I.e., the training and testing MAEs decrease with increasing T. Indeed, it turns out that the testing MAE can be made arbitrarily close to 0 for the present task simply by increasing the resolution, the number of training examples, and the number of training epochs.

D.5.2 Real-World Datasets

We now study the behavior of the RTM on five different real-world datasets:

Dengue Incidences:[3] This dataset consists of monthly dengue incidences in the Philippines per 100,000 population. The Philippines has 17 administrative regions. The department of health in the Philippines collected the data from all of these regions separately from 2008 to 2016. In this experiment, the RTM is trained separately for each region using the data from 2008 to 2015. Dengue incidences from the neighboring regions are used as input features. More details about data preprocessing and feature selection can be found in [19]. Dengue incidences in 2016 are used as testing data.

Energy Performance:[4] In this application, heating load of residential buildings has to be estimated using eight input features: glazing area distribution, glazing area, orientation, relative compactness, wall area, surface area, overall height, roof area, and orientation [20]. The complete dataset comprises 768 samples. We utilize 80% of the data samples to train the model and the rest to evaluate it.

Stock Selection:[5] The US historical stock market data was used to simulate the weights of the stock-picking concepts in [21]. These weights are employed to build a stock

[3] Available from https://www.kaggle.com/grosvenpaul/dengue-cases-in-the-philippines

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

[5] Available from https://archive.ics.uci.edu/ml/datasets/Stock+portfolio+performance

selection decision support system. The dataset consists of six output variables. Only two of them, i.e., Annual Return and Real Win Rate is used here. Again, 80% of the data samples are utilized to train the model and the rest is used for testing.

Real Estate Valuation:[6] This is a time series dataset that has been used to estimate the house price per unit area in Taiwan [22]. The house price per unit area (Dollars per Ping, 1 Ping = 3.3 meter squared) is estimated using six features, namely the transaction date, the house age, the distance to the nearest MRT station, the number of convenience stores in the living circle on foot, and the geographical coordinates (latitude and longitude). Again we use (80%) of the dataset for training and (20%) for testing.

Aerofoil Noise:[7] The data set comprises different sizes of NACA 0012 airfoils (an airfoil shape for aircraft wings) at various wind tunnel speeds and angles of attack. The goal is to predict the self-generated noise of an airfoil blade in decibels using five attributes, namely, Frequency (in Hertzs), Angle of attack (in degrees), Chord length (in meters), Free-stream velocity (in meters per second), and Suction side displacement thickness (in meters). Out of the total 1503 data samples, 625 are randomly selected for doing the experiment. The RTM is trained on 80% of the selected data and tested on the rest of them.

Apart from Dengue Incidences, we used the same number of clauses and the same resolution for all of the datasets, respectively, 2 000 000 and 1 000 000. This setting was found by increasing the number of clauses and resolution until the accuracy gain became insignificant. The specificity s for each dataset was found by a manual binary search (4 for Energy Performance, 3 for Annual Return, 2.5 for Real Win Rate, 3.8 for Real Estate Valuation, and 2.7 for Aerofoil Noise). For forecasting dengue incidences across all the regions in the Philippines we used 200 000 clauses, a resolution of 100 000, and an s-value of 6.

Table D.3 reports average MAE obtained by replicating each experiment 20 times, together with 95% confidence intervals. To evaluate the performance of the RTM, we used three other state-of-the-art machine learning models as a baseline. These models are Regression Tree (RT), Random Forest (RF), and Support Vector Regression (SVR). Each of these were configured based on a thorough parameter search in order to facilitate a fair comparison. Each model predicts six different data series using their relevant input features. In conclusion, RTM obtained the best MAE for four of the prediction tasks (Dengue Incidences, Energy Performance, Real Win Rate, and Real Estate Valuation). For Annual Return, SVR produced the lowest MAE, while RTM achieved the second best. Finally, RF provided the best MAE for Aerofoil Noise. Here, RTM obtained the third best result.

In order to investigate the ability of the RTM to extrapolate, we cross-validated the model using the Energy Performance dataset. To produce test data with extreme values not found in the training data, ten different datasets were created by randomly shuffling the original dataset. Collectively, this procedure created 22 test outcomes outside the minimum and maximum output values in the training data. The MAE of those 22 testing outputs was 0.516, which is only slightly higher than the overall MAE in Table D.3, mean-

[6] Available from https://archive.ics.uci.edu/ml/datasets/Real+estate+valuation+data+set

[7] Available from https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise#

ing that the RTM extrapolates reasonably well. This can be explained by the additive nature of RTM regression. By summing the binary output of non-linear clauses, the RTM avoids extreme non-linear distortions when operating outside the value ranges found in the training data.

Table D.3: Mean MAE with 95% confidence intervals for selected models on five datasets.

Model	Dengue Incidences	Energy Performance	Stock Selection		Real Estate Valuation	Aerofoil Noise
			Annual Return	Real Win Rate		
RTM	5.184	0.503±0.00	0.077±0.00	0.089±0.00	5.218±0.00	2.244±0.00
RT	7.769	0.598±0.00	0.091±0.00	0.096±0.00	6.033±0.11	2.280±0.07
RF	9.122	0.563±0.00	0.088±0.00	0.097±0.00	5.360±0.00	1.921±0.00
SVR	5.305	1.155±0.00	0.073±0.00	0.092±0.00	5.697±0.00	2.156±0.00

D.6 Conclusion

In this paper, we proposed the Regression Tsetlin Machine (RTM), a novel variant of the Tsetlin Machine that supports continuous output in non-linear regression problems. We also introduced a data preprocessing procedure that converts continuous inputs into a lossless binary feature matrix. In RTM, the polarities in clauses are removed and the inner inference mechanism is modified. That is, the input patterns are transformed into individual continuous outputs, rather than to distinct categories. The activation of clauses during RTM learning is based on a linear activation probability function that is governed by the magnitude of the regression error. The behavior of the new algorithm was studied by applying it to both artificial and real-world datasets. In brief, RTM demonstrated superior performance in comparison with Regression Trees, Random Forests, and Support Vector Regression, when predicting Dengue incidences in the Philippines, heating load in the Energy Performance dataset, Real Win Rate in the Stock Selection dataset, and house price per unit area in the Real Estate Valuation dataset. The RTM also extrapolates reasonably well outside the minimum and maximum output values found in the training data.

Bibliography

[1] Alex A Freitas. "Comprehensible Classification Models: A Position Paper". In: *ACM SIGKDD explorations newsletter* 15.1 (2014), pp. 1–10.

[2] Bart Baesens, Christophe Mues, Manu De Backer, Jan Vanthienen, and Rudy Setiono. "Building Intelligent Credit Scoring Systems Using Decision Tables". In: *Enterprise Information Systems V*. Springer, 2004, pp. 131–137.

[3] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. "An Empirical Evaluation of the Comprehensibility of Decision Table, Tree and Rule Based Predictive Models". In: *Decision Support Systems* 51.1 (2011), pp. 141–154.

[4] Riccardo Bellazzi and Blaz Zupan. "Predictive Data Mining in Clinical Medicine: Current Issues and Guidelines". In: *International journal of medical informatics* 77.2 (2008), pp. 81–97.

[5] Michael J Pazzani, S Mani, and William R Shankle. "Acceptance of Rules Generated by Machine Learning Among Medical Experts". In: *Methods of information in medicine* 40.05 (2001), pp. 380–385.

[6] Alex A Freitas, Daniela C Wieser, and Rolf Apweiler. "On the Importance of Comprehensible Classification Models for Protein Function Prediction". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7.1 (2008), pp. 172–182.

[7] Duane Szafron, Paul Lu, Russell Greiner, David S Wishart, Brett Poulin, Roman Eisner, Zhiyong Lu, John Anvik, Cam Macdonell, Alona Fyshe, et al. "Proteome Analyst: Custom Predictions With Explanations in a Web-Based Tool for High-Throughput proteome Annotations". In: *Nucleic acids research* 32.suppl_2 (2004), W365–W371.

[8] Elen Lima, Christophe Mues, and Bart Baesens. "Domain Knowledge Integration in Data Mining Using Decision Tables: Case Studies in Churn Prediction". In: *Journal of the Operational Research Society* 60.8 (2009), pp. 1096–1106.

[9] Wouter Verbeke, David Martens, Christophe Mues, and Bart Baesens. "Building Comprehensible Customer Churn Prediction Models with Advanced Rule Induction Techniques". In: *Expert systems with applications* 38.3 (2011), pp. 2354–2364.

[10] Ole-Christoffer Granmo. "The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic". In: *arXiv preprint arXiv:1804.01508* (2018).

[11] Michael Lvovitch Tsetlin. “On Behaviour of Finite Automata in Random Medium”. In: *Avtomat. i Telemekh* 22.10 (1961), pp. 1345–1354.

[12] Kumpati S Narendra and Mandayam AL Thathachar. *Learning Automata: An Introduction*. Courier corporation, 2012.

[13] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. “A Novel Tsetlin Automata Scheme to Forecast Dengue Outbreaks in the Philippines”. In: *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2018, pp. 680–685.

[14] Noureddine Bouhmala and Ole-Christoffer Granmo. “Stochastic Learning for SAT-Encoded Graph Coloring Problems”. In: *International Journal of Applied Metaheuristic Computing (IJAMC)* 1.3 (2010), pp. 1–19.

[15] Brian Tung and Leonard Kleinrock. “Using Finite State Automata to Produce Self-Optimization and Self-Control”. In: *IEEE transactions on parallel and distributed systems* 7.4 (1996), pp. 439–448.

[16] B John Oommen, Sang-Woon Kim, Mathew T Samuel, and Ole-Christoffer Granmo. “A Solution to the Stochastic Point Location Problem in Metalevel Nonstationary Environments”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 38.2 (2008), pp. 466–476.

[17] Ole-Christoffer Granmo and B John Oommen. “Solving Stochastic Nonlinear Resource Allocation Problems Using a Hierarchy of Twofold Resource Allocation Automata”. In: *IEEE Transactions on Computers* 59.4 (2010), pp. 545–560.

[18] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. “Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications”. In: *IEEE Access* 7 (2019), pp. 115134–115146.

[19] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. “A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2019, pp. 564–578.

[20] Athanasios Tsanas and Angeliki Xifara. “Accurate Quantitative Estimation of Energy Performance of Residential Buildings Using Statistical Machine Learning Tools”. In: *Energy and Buildings* 49 (2012), pp. 560–567.

[21] Yi-Cheng Liu and I-Cheng Yeh. “Using Mixture Design and Neural Networks to Build Stock Selection Decision Support Systems”. In: *Neural Computing and Applications* 28.3 (2017), pp. 521–535.

[22] I-Cheng Yeh and Tzu-Kuang Hsu. “Building Real Estate Valuation Models with Comparative Approach Through Case-Based Reasoning”. In: *Applied Soft Computing* 65 (2018), pp. 260–271.

Paper E

Integer Weighted Regression Tsetlin Machines

The Regression Tsetlin Machine (RTM) addresses the lack of interpretability impeding state-of-the-art nonlinear regression models. It does this by using conjunctive clauses in propositional logic to capture the underlying non-linear frequent patterns in the data. These, in turn, are combined into a continuous output through summation, akin to a linear regression function, however, with non-linear components and binary weights. However, the resolution of the RTM output is proportional to the number of clauses employed. This means that computation cost increases with resolution. To address this problem, we here introduce integer weighted RTM clauses. Our integer weighted clause is a compact representation of multiple clauses that capture the same sub-pattern — w repeating clauses are turned into one, with an integer weight w. This reduces computation cost w times, and increases interpretability through a sparser representation. We introduce a novel learning scheme, based on so-called stochastic searching on the line. We evaluate the potential of the integer weighted RTM empirically using two artificial datasets. The results show that the integer weighted RTM is able to acquire on par or better accuracy using significantly less computational resources compared to regular RTM and an RTM with real-valued weights.

E.1 Introduction

The recently introduced Regression Tsetlin Machine (RTM) [1, 2] is a propositional logic based approach to interpretable non-linear regression, founded on the Tsetlin Machine (TM) [3]. Being based on disjunctive normal form, like Karnaugh maps, the TM can map an exponential number of input feature value combinations to an appropriate output [4]. Recent research reports several distinct TM properties. The clauses that a TM produces have an interpretable form (e.g., **if** X **satisfies** condition A **and not** condition B **then** Y = 1), similar to the branches in a decision tree [5]. For small-scale pattern recognition problems, up to three orders of magnitude lower energy consumption and inference time has been reported, compared to neural networks alike [6]. Like neural networks, the TM can be used in convolution, providing competitive memory usage, computation speed, and accuracy on MNIST, F-MNIST and K-MNIST, in comparison with simple 4-layer CNNs, K-Nereast Neighbors, SVMs, Random Forests, Gradient Boosting, BinaryConnect, Logistic Circuits, and ResNet [7]. By introducing clause weights that allow one clause to represent multiple, it has been demonstrated that the number of clauses can be reduced up to 50×, without loss of accuracy, leading to more compact clause sets [4]. Finally, hyper-parameter search can be simplified with multi-granular clauses, eliminating the pattern specificity parameter [8].

Paper contributions: In the RTM, regression resolution is proportional to the number of conjunctive clauses employed. In other words, computation cost and memory usage grows proportionally with resolution. Building upon the Weighted TM (WTM) by Phoulady et al. [4], this paper introduces weights to the RTM scheme. However, while the WTM uses real-valued weights for classification, we instead propose a novel scheme based on *integer* weights, targeting *regression*. In brief, we use the stochastic searching on the line approach pioneered by Oommen in 1997 [9] to eliminate multiplication from the weight updating. In addition to the computational benefits this entails, we also argue that integer weighted clauses are more interpretable than real-valued ones because they can be seen as multiple copies of the same clause. Finally, our scheme does not introduce additional hyper-parameters, whereas the WTM relies on weight learning speed.

Paper Organization: The remainder of the paper is organized as follows. In Section 2, the basics of RTMs are provided. Then, in Section 3, the SPL problem and its solution are explained. The main contribution of this paper, the integer weighting scheme for the RTM, is presented in detail in Section 4 and evaluated empirically using two artificial datasets in Section 5. We conclude our work in Section 6.

E.2 The Regression Tsetlin Machine (RTM)

The RTM performs regression based on formulas in propositional logic. In all brevity, the input to an RTM is a vector $\mathbf{X}$ of o propositional variables x_k, $\mathbf{X} \in \{0,1\}^o$. These are further augmented with their negated counterparts $\bar{x}_k = 1 - x_k$ to form a vector of literals: $\mathbf{L} = [x_1, \ldots, x_o, \bar{x}_1, \ldots, \bar{x}_o] = [l_1, \ldots, l_{2o}]$. In contrast to a regular TM, the output of an RTM is real-valued, normalized to the domain $y \in [0,1]$.

Regression Function: The regression function of an RTM is simply a linear summation of products, where the products are built from the literals:

$$y = \frac{1}{T} \sum_{j=1}^{m} \prod_{k \in I_j} l_k. \tag{E.1}$$

Above, the index j refers to one particular product of literals, defined by the subset I_j of literal indexes. If we e.g. have two propositional variables x_1 and x_2, the literal index sets $I_1 = \{1,4\}$ and $I_2 = \{2,3\}$ define the function: $y = \frac{1}{T}(x_1\bar{x}_2 + \bar{x}_1 x_2)$. The user set parameter T decides the resolution of the regression function. Notice that each product in the summation either evaluates to 0 or 1. This means that a larger T requires more literal products to reach a particular value y. Thus, increasing T makes the regression function increasingly fine-grained. In the following, we will formulate and refer to the products as *conjunctive clauses*, as is typical for the regular TM. The value c_j of each product is then a conjunction of literals:

$$c_j = \prod_{k \in I_j} l_k = \bigwedge_{k \in I_j} l_k. \tag{E.2}$$

Finally, note that the number of conjunctive clauses m in the regression function also is a user set parameter, which decides the expression power of the RTM.

Tsetlin Automata Teams: The composition of each clause is decided by a team of Tsetlin Automata (TAs) [10]. There are $2 \times o$ number of TAs per clause j. Each represents a particular literal k and decides whether to *include* or *exclude* that literal in the clause. The decision depends on the current state of the TA, denoted $a_{j,k} \in \{1, \ldots, 2N\}$. States from 1 to N produce an *exclude* action and states from $N+1$ to $2N$ produce an *include* action. Accordingly, the set of indexes I_j can be defined as $I_j = \{k | a_{j,k} > N, 1 \leq k \leq 2o\}$. The states of all of the TAs are organized as an $m \times 2o$ matrix $\mathbf{A}$: $\mathbf{A} = (a_{j,k}) \in \{1, \ldots, 2N\}^{m \times 2o}$ where m is the number of clauses.

Learning Procedure: Learning in RTM is done through an online reinforcement scheme that updates the state matrix $\mathbf{A}$ by processing one training example $(\hat{X}_i, \hat{y}_i)$ at a time, as detailed below.

The RTM employs two kinds of feedback, Type I and Type II, further defined below. Type I feedback triggers TA state changes that eventually make a clause output 1 for the given training example $\hat{X}_i$. Conversely, Type II feedback triggers state changes that eventually make the clause output 0. Thus, overall, regression error can be systematically reduced by carefully distributing Type I and Type II feedback:

$$Feedback = \begin{cases} \text{Type I,} & \text{if} \quad y < \hat{y}_i, \\ \text{Type II,} & \text{if} \quad y > \hat{y}_i. \end{cases} \tag{E.3}$$

In effect, the number of clauses that evaluates to 1 is increased when the predicted output is less than the target output $(y < \hat{y}_i)$ by providing Type I feedback. Type II feedback, on the other hand, is applied to decrease the number of clauses that evaluates to 1 when the predicted output is higher than the target output $(y > \hat{y}_i)$.

Activation Probability: Feedback is handed out stochastically to regulate learning. The feedback probability p_j is proportional to the absolute error of the prediction, $\mid y - \hat{y}_i \mid$. Clauses activated for feedback are the stored in the matrix $\mathbf{P} = (p_j) \in \{0, 1\}^m$.

Type I feedback: Type I feedback subdivides into Type Ia and Type Ib. Type Ia reinforces *include* actions of TAs whose corresponding literal value is 1, however, only when the clause output is 1. The probability of k^{th} TA of the j^{th} clause receives Type Ia feedback $r_{j,k}$ is $\frac{s-1}{s}$, where s (s ≥ 1) is a user set parameter. Type Ib combats over-fitting by reinforcing *exclude* actions of TAs when the corresponding literal is 0 or when the clause output is 0. The probability of k^{th} TA of the j^{th} clause receives Type Ib feedback $q_{j,k}$ is $\frac{1}{s}$.

Using the complete set of conditions, the TAs selected for Type Ia feedback are singled out by the indexes $I^{\text{Ia}} = \{(j,k) | l_k = 1 \land c_j = 1 \land p_j = 1 \land r_{j,k} = 1\}$. Similarly, TAs selected for Type Ib are $I^{\text{Ib}} = \{(j,k) | (l_k = 0 \lor c_j = 0) \land p_j = 1 \land q_{j,k} = 1\}$.

Once the TAs have been targeted for Type Ia and Type Ib feedback, their states are updated. Available updating operators are $\oplus$ and $\ominus$, where $\oplus$ adds 1 to the current state while $\ominus$ subtracts 1. Thus, before a new learning iterations starts, the states in the matrix $\mathbf{A}$ are updated as follows: $\mathbf{A} \leftarrow \left(\mathbf{A} \oplus I^{\text{Ia}}\right) \ominus I^{\text{Ib}}$.

Type II feedback: Type II feedback eventually changes the output of a clause from 1 to 0, for a specific input $\hat{X}_i$. This is achieved simply by including one or more of the literals that take the value 0 for $\hat{X}_i$. The indexes of TAs selected for Type II can thus be singled out as $I^{\text{II}} = \{(j,k)|l_k = 0 \land c_j = 1 \land p_j = 1\}$. Accordingly, the states of the TAs are updated as follows: $\mathbf{A} \leftarrow \mathbf{A} \oplus I^{\text{II}}$.

E.3 Stochastic Searching on the Line

Stochastic searching on the line, also referred to as stochastic point location (SPL) was pioneered by Oommen in 1997 [9]. SPL is a fundamental optimization problem where one tries to locate an unknown unique point within a given interval. The only available information for the Learning Mechanism (LM) is the possibly faulty feedback provided by the attached environment (E). According to the feedback, LM moves right or left from its current location in a discretized solution space.

The task at hand is to determine the optimal value λ^* of a variable λ, assuming that the environment is informative. That is, that it provides the correct direction of λ^* with probability $p > 0.5$. In SPL, λ is assume to be any number in the interval $[0,1]$. The SPL scheme of Oommen discretizes the solution space by subdividing the unit interval into N steps, $\{0, 1/N, 2/N, ..., (N-1)/N, 1\}$. Hence, N defines the resolution of the learning scheme.

The current guess, $\lambda(n)$, is updated according to the feedback from the environment as follows:

$$\lambda(n+1) = \begin{cases} \lambda(n) + 1/N, & \text{if} \quad E(n) = 1 \text{ and } 0 \leqslant \lambda(n) < 1 \ , \\ \lambda(n) - 1/N, & \text{if} \quad E(n) = 0 \text{ and } 0 < \lambda(n) \leqslant 1 \ , \\ \lambda(n), & \text{Otherwise} \quad . \end{cases} \tag{E.4}$$

The feedback $E(n) = 1$ is the environment suggestion to increase the value of λ and $E(n) = 0$ is the environment suggestion to decrease the value of λ. Asymptotically, the learning mechanics is able to find a value arbitrarily close to λ^* when $N \rightarrow \infty$ and $n \rightarrow \infty$.

E.4 Regression Tsetlin Machine with Weighted Clauses

We now introduce clauses with integer weights to provide a more compact representation of the regression function. The regression function for the integer weighted RTM attaches a weight w_j to each clause output c_j, $j = 1, ..., m$. Consequently, the regression output can be computed according to Eq. E.5:

$$y = \frac{1}{T} \sum_{j=1}^{m} w_j \prod_{k \in I_j} l_k. \tag{E.5}$$

Weight learning: Our approach to learning the weight of each clause is similar to SPL. However, the solution space of each weight is $[0, \infty]$, while the resolution of the learning scheme is $N = 1$. The weight attached to a clause is updated when the clause receives Type Ia feedback or Type II feedback. The weight updating procedure is summarized in Algorithm 3. Here, $w_j(n)$ is the weight of clause j at the n^{th} training round.

Algorithm 3 Weight Learning

```
input Round $n$ updating of clause weights
procedure (Initialization (round 0):) $w_j(0) \leftarrow 0, j = 1, \ldots, m$
    Initialization (round $n$): $y$ is calculated according to Eq. E.5.
    for $j = 1, ..., m$ do
        if $y(n) < \hat{y}_i(n) \wedge c_j(n) = 1 \wedge p_j(n) = 1$ then
            $w_j(n+1) \leftarrow w_j(n) + N$
        else if $y(n) > \hat{y}_i(n) \wedge c_j(n) = 1 \wedge p_j(n) = 1 \wedge w_j(n) > 0$ then
            $w_j(n+1) \leftarrow w_j(n) - N$
        else
            $w_j(n+1) \leftarrow w_j(n)$
        end if
    end for

    Return $w_j(n+1), j = 1, \ldots, m$
end procedure
```

Note that since weights in this study can take any value higher than or equal to 0, an unwanted clause can be turned off by setting its weight to 0. Further, sub-patterns that have a large impact on the calculation of y can be represented with a correspondingly larger weight.

E.5 Empirical Evaluation

In this section, we study the behavior of the RTM with integer weighting (RTM-IW) using two artificial datasets similar to the datasets presented in [1], in comparison with a standard RTM and a real-value weighted RTM (RTM-RW). We use Mean Absolute Error (MAE) to measure performance.

Artificial Datasets: Dataset I contains 3-bit feature input. The output, in turn, is 100 times larger than the decimal value of the binary input (e.g., the input [0, 1, 0] produces the output 200). The training set consists of 8000 samples while the testing set consists of 2000 samples, both without noise. Dataset II contains the same data as Dataset I, except that the output of the training data is perturbed to introduce noise. Each input feature has been generated independently with equal probability of taking either the value 0 or 1, producing a uniform distribution of bit values.

Results and Discussion: The pattern distribution of the artificial data was analyzed in the original RTM study. As discussed, there are eight unique sub-patterns. The RTM is able to capture the complete set of sub-patterns utilizing no more than three types of

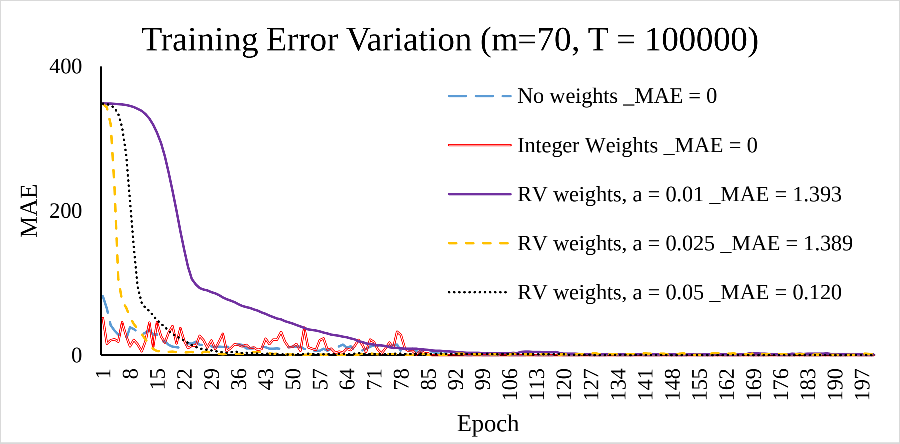


Figure E.1: The training error variation per training epoch for different RTM schemes.

clauses, i.e., $(1 \ast \ast)$, $(\ast\ 1 \ast)$, $(\ast \ast\ 1)$[1]. However, to produce the correct output, some clauses must be duplicated multiple times, depending on the input pattern. For instance, each dataset requires *seven* clauses to represent the three different patterns it contains, namely, $(4 \times (1 \ast \ast), 2 \times (\ast\ 1 \ast), 1 \times (\ast \ast\ 1))$[2]. So, with e.g. the input [1, 0, 1], four clauses which represent the pattern $(1 \ast \ast)$ and one clause which represents the pattern $(\ast \ast\ 1)$ activate to correctly output 500 (after normalization).

Notably, it turns out that the RTM-IW requires even fewer clauses to capture the sub-patterns in the above data, as outlined in Table E.1. Instead of having multiple clauses to represent one sub-pattern, RTM-IW utilizes merely one clause with the correct weight to do the same job. The advantage of the proposed integer weighting scheme is thus apparent. It learns the correct weight of each clause, so that it achieves an MAE of zero. Further, it is possible to ignore redundant clauses simply by giving them the weight zero. For the present dataset, for instance, decreasing m while keeping the same resolution, $T = 7$, does not impede accuracy. The RTM-RW on the other hand struggles to find the correct weights, and fails to minimize MAE. Here, the real valued weights were updated with a learning rate of $\alpha = 0.01$, determined using a binary hyper-parameter search.

[1] Here, $\ast$ means an input feature that can take an arbitrary value, either 0 or 1.

[2] In this expression, "*four* clauses to represent the pattern $(1 \ast \ast)$" is written as "$4 \times (1 \ast \ast)$"

Table E.1: Behavior comparison of different RTM schemes on Dataset III.

	m	T	Pattern	I_j	$\bar{I}_j$	No. of Clauses Required	w_j	Training MAE	Testing MAE
RTM	7	7	$(1 \ast \ast)$	{1}	{ }	4	-	0	0
			$(\ast\ 1 \ast)$	{2}	{ }	2	-		
			$(\ast \ast\ 1)$	{3}	{ }	1	-		
RTM-IW	3	7	$(1 \ast \ast)$	{1}	{ }	1	4	0	0
			$(\ast\ 1 \ast)$	{2}	{ }	1	2		
			$(\ast \ast\ 1)$	{3}	{ }	1	1		
RTM-RW	3	7	$(1 \ast \ast)$	{1}	{ }	1	3.987	1.857	1.799
			$(\ast\ 1 \ast)$	{2}	{ }	1	2.027		
			$(\ast \ast\ 1)$	{3}	{ }	1	0.971		

Figure E.1 casts further light on learning behaviour by reporting training error per epoch for the three different RTM schemes with $m = 70$ and $T = 100000$. As seen, both RTM and RTM-IW obtain relatively low MAE after just one training epoch, eventually reaching MAE zero (training MAE at end of training are given in the legend of each graph). RTM-RW, on the other hand, starts off with a much higher MAE, which is drastically decreasing over a few epochs, however, fails to reach MAE 0 after becoming asymptotically stable.

We also studied the effect of T on performance with noise free data by varying T, while fixing the number of clauses m. For instance, RTM was able to reach a training MAE of 1.9 and a testing error of 2.1 with $m = T = 300$ [1]. For the same dataset, RTM-IW can reach a training error of 0.19 and a testing error of 1.87 with $m = 200$ and $T = 2000$. Further, for $m = 200$ and $T = 20\ 000$, training error drops to 0.027 and testing error drops to 0.027. Finally, by increasing T to 200 000 training error falls to 0.0003 while testing error stabilises at 0.0002.

To further compare the performance of RTM-IW with RTM and RTM-RW, each approach was evaluated using a wide rage of m and T settings. Representative training and testing MAE for both datasets are summarized in Table E.2. Here, the number of clauses used with each dataset is also given. The T for the original RTM is equal to the number of clauses, while for the RTM with weights T is simply 100 times that number.

As seen, the training and testing MAE reach zero when the RTM operates with noise free data when $m = 7$. However, MAE approaches zero with RTM-IW and RTM-RW when increasing number of clauses m.

For noisy data, the minimum training MAE achieved by RTM is 5.500, obtained with $m = 5000$ clauses. The RTM-IW, on the other hand, obtains a lower MAE of 5.373 with less than half of the clauses ($m = 2000$). The accuracy of RTM-IW in comparison with RTM-RW is less clear, with quite similar MAE for noisy data. The average testing MAE across both the datasets, however, reveals that the average MAE of RTM-IW is lower than that of the RTM-RW (2.101 vs 2.168).

E.6 Conclusion

In this paper, we presented a new weighting scheme for the Regression Tsetlin Machine (RTM), RTM with Integer Weights (RTM-IW). The weights attached to the clauses helps the RTM represent sub-patterns in a more compact way. Since the weights are integer, interpretability is improved through a more compact representation of the clause set. We also presented a new weight learning scheme based on stochastic searching on the line, integrated with the Type I and Type II feedback of the RTM. The RTM-IW obtains on par or better accuracy with fewer number of clauses compared to RTM without weights. It also performs competitively in comparison with an alternative RTM with real-valued weights.

Table E.2: Training and testing MAE after 200 training epochs by various methods with different m and T.

Model		RTM				RTM-RW				RTM-IW			
MAE		Training		Testing		Training		Testing		Training		Testing	
Dataset		I	II	I	II	I	II	I	II	I	II	I	II
m	7	0.000	7.400	0.000	5.000	2.230	7.702	2.217	5.955	1.172	8.019	1.171	6.236
	20	14.600	13.800	14.200	14.500	1.023	7.863	1.036	6.007	0.487	9.844	0.493	8.499
	70	0.000	6.600	0.000	4.200	0.292	7.365	0.295	5.735	0.189	7.602	0.189	5.532
	300	1.900	5.800	2.100	3.300	0.104	5.800	0.106	2.226	0.078	5.685	0.078	2.234
	700	1.000	5.900	1.000	3.400	0.013	5.551	0.013	1.968	0.044	5.532	0.044	2.149
	2000	1.000	5.600	1.200	1.900	0.012	5.731	0.012	2.520	0.003	5.373	0.003	1.280
	5000	0.900	5.500	1.000	2.700	0.010	5.635	0.010	2.252	0.001	5.412	0.001	1.501

Bibliography

[1] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Lei Jiao, and Morten Goodwin. "The regression Tsetlin Machine: A Tsetlin Machine for Continuous Output Problems". In: *EPIA Conference on Artificial Intelligence.* Springer. 2019, pp. 268–280.

[2] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. "The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression". In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

[3] Ole-Christoffer Granmo. "The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic". In: *arXiv preprint arXiv:1804.01508* (2018).

[4] Adrian Phoulady, Ole-Christoffer Granmo, Saeed Rahimi Gorji, and Hady Ahmady Phoulady. "The Weighted Tsetlin Machine: Compressed Representations with Clause Weighting". In: *Ninth International Workshop on Statistical Relational AI (StarAI 2020).* 2020.

[5] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications". In: *IEEE Access* 7 (2019), pp. 115134–115146.

[6] Adrian Wheeldon, Rishad Shafik, Alex Yakovlev, Jonathan Edwards, Ibrahim Haddadi, and Ole-Christoffer Granmo. "Tsetlin Machine: A New Paradigm for Pervasive AI". In: *Proceedings of the SCONA Workshop at Design, Automation and Test in Europe (DATE).* 2020.

[7] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. "The Convolutional Tsetlin Machine". In: *arXiv preprint:1905.09688* (2019).

[8] Saeed Rahimi Gorji, Ole-Christoffer Granmo, Adrian Phoulady, and Morten Goodwin. "A Tsetlin Machine with Multigranular Clauses". In: *Lecture Notes in Computer Science: Proceedings of the Thirty-ninth International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI-2019).* Vol. 11927. Springer International Publishing, 2019.

[9] B John Oommen. "Stochastic Searching On the Line and Its Applications to Parameter Learning in Nonlinear Optimization". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27.4 (1997), pp. 733–739.

[10] Michael Lvovitch Tsetlin. “On Behaviour of Finite Automata in Random Medium”. In: *Avtomat. i Telemekh* 22.10 (1961), pp. 1345–1354.

Paper F

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

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.*

F.1 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 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, 22, 23, 24, 25, 26, 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].

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].

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.

Paper Organization: In Section F.2, 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 F.3, we evaluate the classification accuracy of IWTM empirically using five datasets, including a study of rule extraction for Bankruptcy prediction in detail and

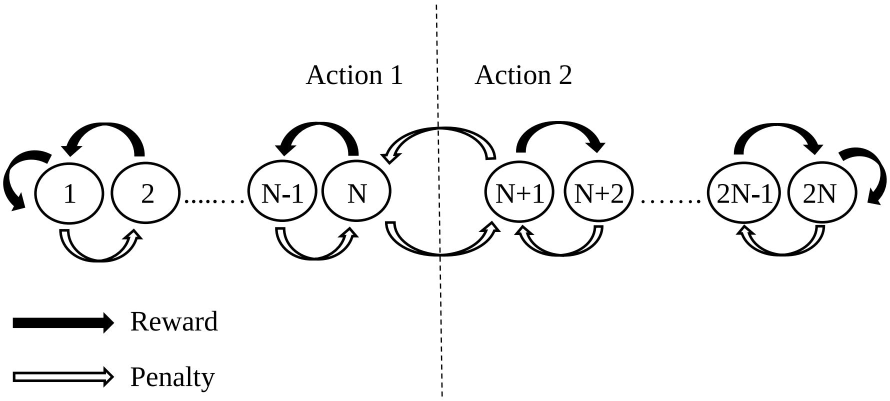


Figure F.1: Transition graph of a two-action Tsetlin Automaton.

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]) 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 F.4.

F.2 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.

F.2.1 Tsetlin Automata

In the TM, a collective of two-action TA [28] (reviewed in [35]) is used for bandit-based learning. Figure F.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].

F.2.2 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 F.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: $\mathbf{L} = [x_1, x_2, \ldots, x_o, \neg x_1, \neg x_2, \ldots, \neg x_o] = [l_1, l_2, \ldots, l_{2o}]$.

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:

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

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:

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

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

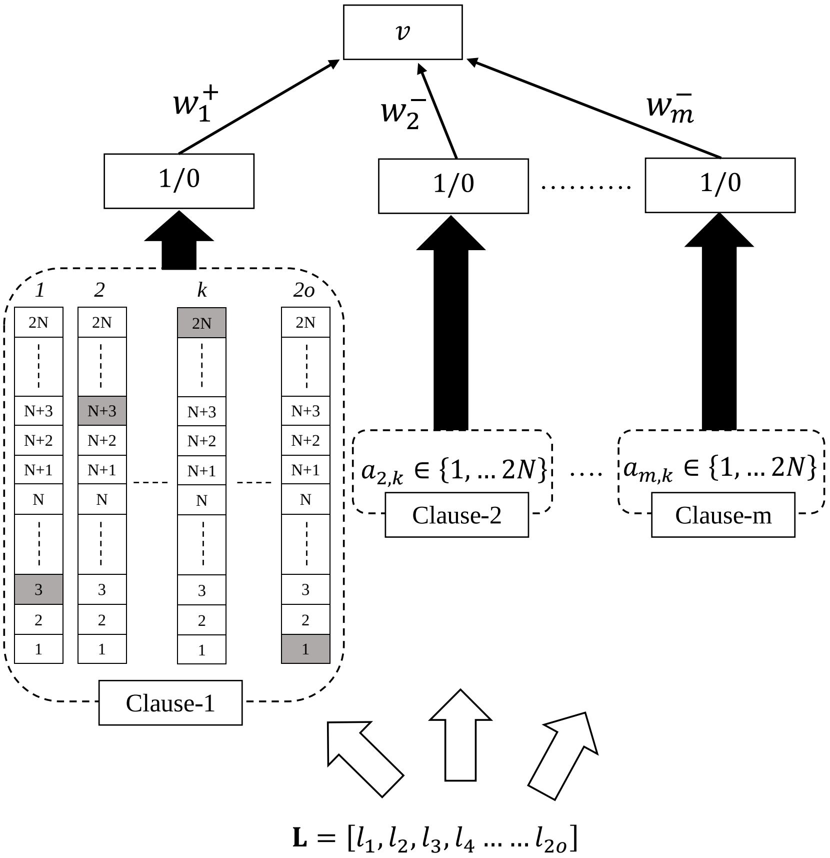
It is the two-action TAs that assign literals to clauses. Each clause is equipped with $2 \times o$ TAs, one per literal k, as shown in *Clause-1* of Figure F.2. The TA states from 1 to N map to the *exclude* action, which means that the corresponding literal is excluded from the clause. For states from $N+1$ 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 $I_j = \{k | a_{j,k} > N, 1 \leq k \leq 2o\}$.

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:

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

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.

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 F.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:

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

F.2.3 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.

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

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

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.

F.2.4 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.

F.2.4.1 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:

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

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. F.6, the individual TAs of the clause are singled out stochastically using a user-set parameter s (s ≥ 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:

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

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

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:

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

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: $\mathbf{A}^+ \leftarrow \mathbf{A}^+ \oplus I^{\text{II}}$.

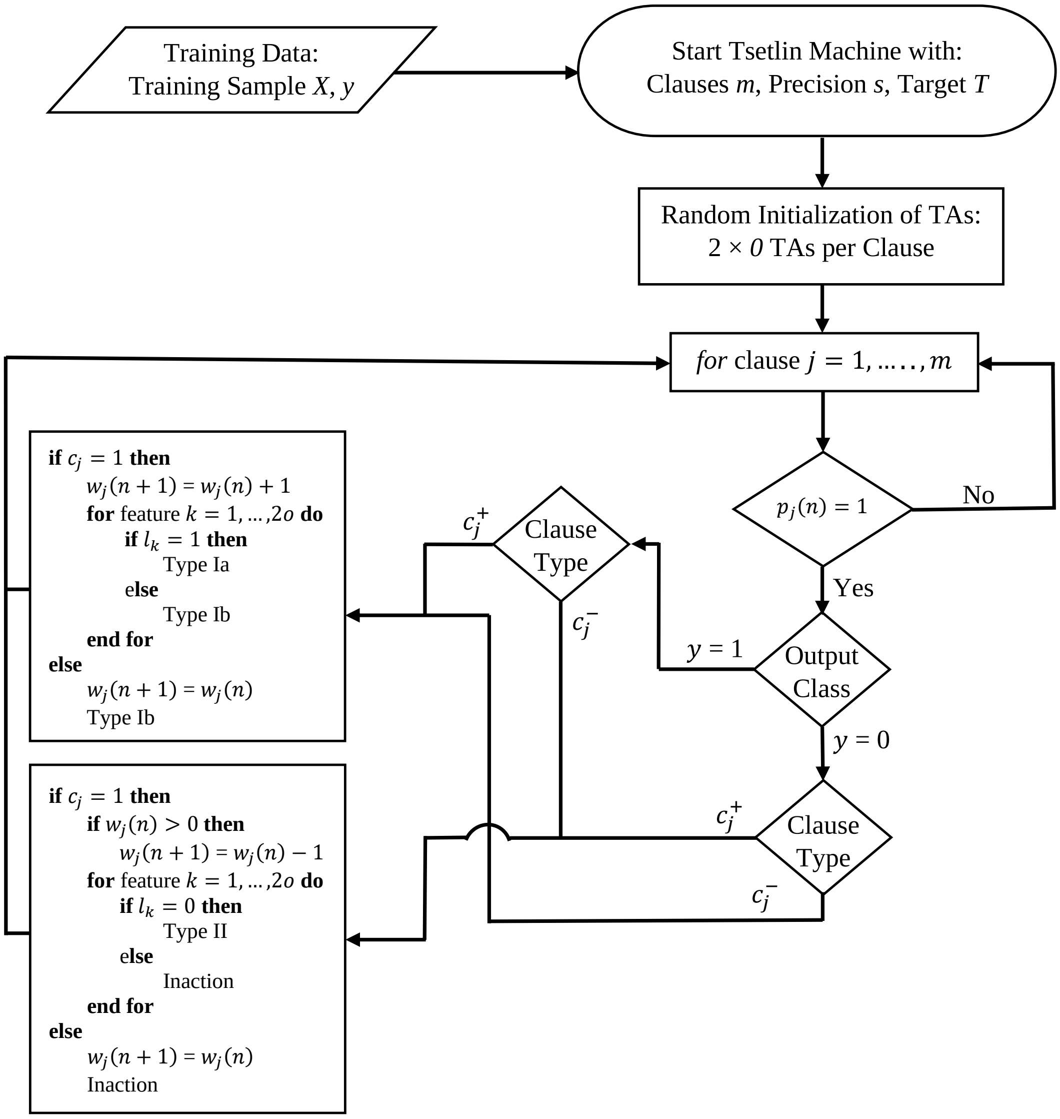
When training has been completed, the final decisions of the TAs are recorded, and the resulting clauses can be deployed for operation.

F.2.4.2 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 4 and in the flowchart in Figure F.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. F.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



```
if $c_j = 1$ then
    $w_j(n+1) = w_j(n) + 1$
    for feature $k = 1, \ldots, 2o$ do
        if $l_k = 1$ then
            Type Ia
        else
            Type Ib
    end for
else
    $w_j(n+1) = w_j(n)$
    Type Ib
```

```
if $c_j = 1$ then
    if $w_j(n) > 0$ then
        $w_j(n+1) = w_j(n) - 1$
    for feature $k = 1, \ldots, 2o$ do
        if $l_k = 0$ then
            Type II
        else
            Inaction
    end for
else
    $w_j(n+1) = w_j(n)$
    Inaction
```

Figure F.3: The complete learning process of the IWTM in a flowchart.

Algorithm 4 The complete IWTM learning process

```
Input: Training data ($\mathbf{X}$, $y$), $m$, $T$, $s$
Initialize: Random initialization of TAs
Begin: $n^{th}$ training round
for $j = 1, ..., m$ do if $p_j = 1$                          ▷ Eq. (F.6) and (F.9)
    if ($y = 1$ and $j$ is odd) or ($y = 0$ and $j$ is even) then
        if $c_j = 1$ then                                      ▷ Eq. (F.3)
            $w_j(n+1) \leftarrow w_j(n) + 1$                   ▷ Eq. (F.10-F.11)
            for feature $k = 1, ..., 2o$ do
                if $l_k = 1$ then
                    Type Ia Feedback
                else:
                    Type Ib Feedback
                end if
            end for
        else:
            $w_j(n+1) \leftarrow w_j(n)$                       ▷ Eq. (F.10-F.11)
            Type Ib Feedback
        end if
    else: ($y = 1$ and $j$ is even) or ($y = 0$ and $j$ is odd)
        if $c_j = 1$ then                                      ▷ Eq. (F.3)
            if $w_j(n) > 0$ then
                $w_j(n+1) \leftarrow w_j(n) - 1$               ▷ Eq. (F.10-F.11)
            end if
            for feature $k = 1, ..., 2o$ do
                if $l_k = 0$ then
                    Type II Feedback
                else:
                    Inaction
                end if
            end for
        else:
            $w_j(n+1) \leftarrow w_j(n)$                       ▷ Eq. (F.10-F.11)
            Inaction
        end if
    end if
end for
```

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, . . . , (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 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:

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

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

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 F.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.

F.2.5 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 $f(d)$:

$$f(d) = d[\gamma \ \times (2o+1) \times m + \alpha \times 2o \times m + \beta \times (m-1)]. \tag{F.12}$$

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.

F.3 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

Table F.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 F.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	

Table F.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	

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.

F.3.1 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 F.1. Thus, the binarized dataset contains 18 binary features.

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

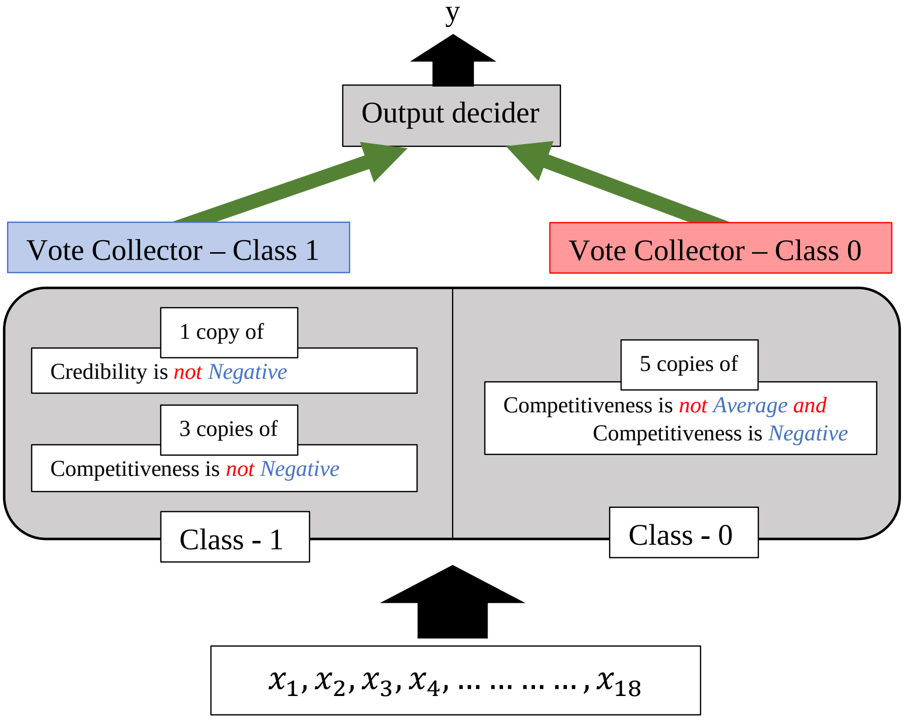

y
Output decider
Vote Collector – Class 1
Vote Collector – Class 0
1 copy of
Credibility is not Negative
3 copies of
Competitiveness is not Negative
5 copies of
Competitiveness is not Average and
Competitiveness is Negative
Class - 1
Class - 0
$x_1, x_2, x_3, x_4, \dots\dots\dots\dots, x_{18}$

Figure F.4: TM classification process for the Bankruptcy dataset.

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

The clause outputs of the TM in Table F.2 are visualized in Figure F.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 F.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):

$$\text{Outcome} = \begin{cases} \text{Bankruptcy} & \textbf{if} \text{ Negative Competitiveness} \\ \text{Non-bankruptcy} & \textbf{otherwise}. \end{cases} \tag{F.13}$$

By asking the TMs to only produce two clauses, we can obtain the above rule more directly, as shown in Table F.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 F.4, Table F.5, and Table F.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 F.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 F.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 F.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 F.5. However, it peaks with an F1-Score of 0.999 at $m = 2000$. Then 3478 literals have been included on average.

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.

Table F.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

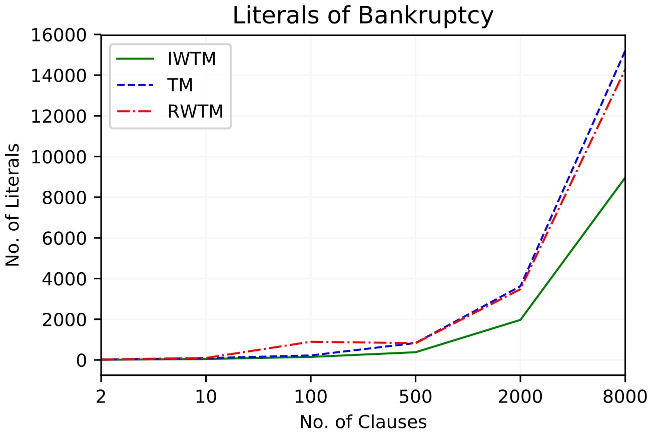


Figure F.5: The number of literals included in different TM setups to work with Bankruptcy dataset.

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

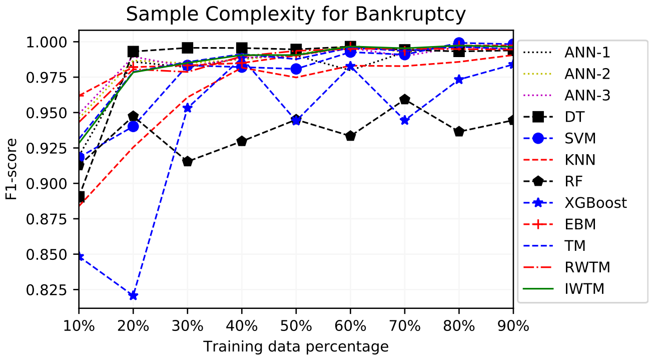


Figure F.6: Sample complexity analysis for the Bankruptcy dataset.

Table F.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.

Table F.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 F.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 F.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

notice that all of the TMs achieve a Recall of 1.0. Additionally, 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 F.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.

F.3.2 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.

Table F.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	-	≈ 966.57KB / ≈ 24.56KB	0.614 sec.
ANN-2	0.995	0.995	0.995	0.995	0.994	-	≈ 3612.65KB / ≈ 589.82KB	0.588 sec.
ANN-3	0.995	0.995	0.995	0.995	0.995	-	≈ 33712.82KB / ≈ 1478.64KB	0.678 sec.
DT	0.984	0.988	0.986	0.986	0.985	-	≈ 131.07KB / ≈ 0.00KB	0.007 sec.
SVM	0.887	0.889	0.887	0.887	0.884	-	≈ 65.53KB / ≈ 241.59KB	0.001 sec.
KNN	0.968	0.939	0.953	0.953	0.969	-	≈ 249.77KB / ≈ 126.87KB	0.001 sec.
RF	0.872	0.851	0.859	0.860	0.871	-	≈ 0.00KB / ≈ 0.00KB	0.021 sec.
XGBoost	0.942	0.921	0.931	0.931	0.942	-	≈ 1126.39KB / ≈ 0.00KB	0.030 sec.
EBM	1.000	1.000	1.000	1.000	1.000	-	≈ 1642.49KB / ≈ 0.00KB	15.658 sec.
TM	0.966	0.930	0.945	0.948	0.966	790	≈ 16.37KB / ≈ 0.00KB	0.011 sec.
RWTM	0.916	0.949	0.931	0.931	0.913	66310	≈ 65.53KB / ≈ 0.00KB	0.910 sec.
IWTM	0.946	0.966	0.954	0.953	0.941	120	≈ 16.37KB / ≈ 0.00KB	0.015 sec.

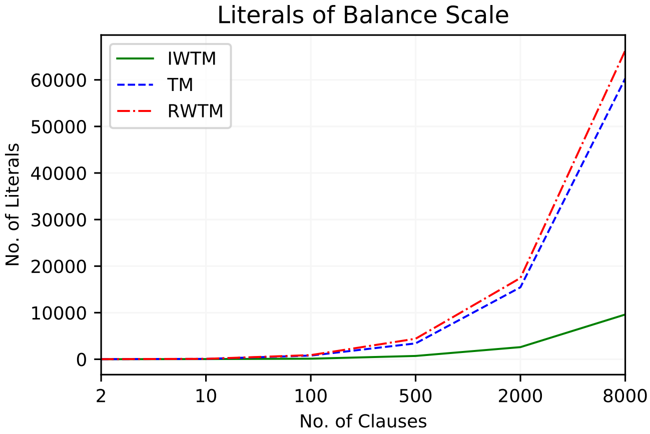


Figure F.7: The number of literals included in different TM setups to work with Balance Scale dataset.

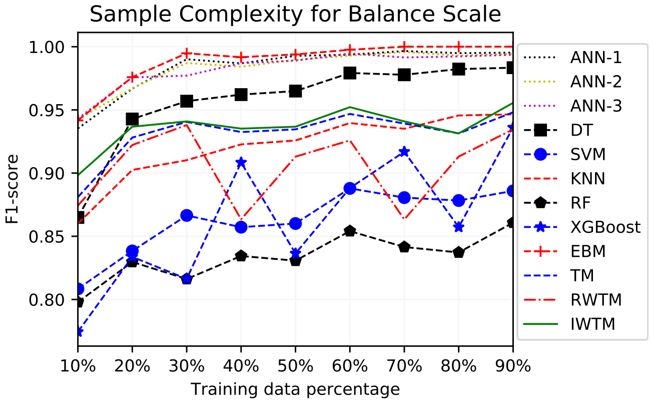


Figure F.8: Sample complexity analysis for the Balance Scale dataset.

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 F.8, Table F.9, and Table F.10 contain the results of TM, RWTM, and IWTM, respectively, with varying m. For the TM, F1-Score peaks at 0.945 when $m = 100$. At the peak, 790 literals are used on average. RWTM obtains its 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 F.7. IWTM uses the least number of literals, with the difference increasing with number of clauses.

Table F.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

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

Table F.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 F.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 F.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 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 F.8, IWTM reaches an F1-Score of 0.90 with merely 10% of the training data, and approaches 0.95 from 20%.

F.3.3 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 F.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	-	$\approx$ 1001.97KB / $\approx$ 35.74KB	0.249 sec.
ANN-2	0.430	0.398	0.403	0.683	0.792	-	$\approx$ 3498.47KB / $\approx$ 608.71KB	0.248 sec.
ANN-3	0.469	0.406	0.422	0.685	0.808	-	$\approx$ 38645.07KB / $\approx$ 1837.76KB	0.288 sec.
DT	0.415	0.222	0.276	0.706	0.915	-	$\approx$ 102.39KB / $\approx$ 0.00KB	0.005 sec.
SVM	0.428	0.364	0.384	0.678	0.805	-	$\approx$ 241.66KB / $\approx$ 299.00KB	0.001 sec.
KNN	0.535	0.423	0.458	0.755	0.871	-	$\approx$ 249.85KB / $\approx$ 61.43KB	0.001 sec.
RF	0.718	0.267	0.370	0.747	0.947	-	$\approx$ 139.26KB / $\approx$ 0.00KB	0.020 sec.
XGBoost	0.428	0.344	0.367	0.719	0.857	-	$\approx$ 1327.10KB / $\approx$ 0.00KB	0.026 sec.
EBM	0.713	0.281	0.389	0.745	0.944	-	$\approx$ 1724.41KB / $\approx$ 0.00KB	6. 007 sec.
TM	0.518	0.583	0.531	0.703	0.742	21	$\approx$ 0.00KB / $\approx$ 0.00KB	0.001 sec.
RWTM	0.461	0.576	0.493	0.706	0.758	4	$\approx$ 0.00KB / $\approx$ 0.00KB	0.001 sec.
IWTM	0.396	0.766	0.511	0.604	0.545	2	$\approx$ 0.00KB / $\approx$ 0.00KB	0.001 sec.

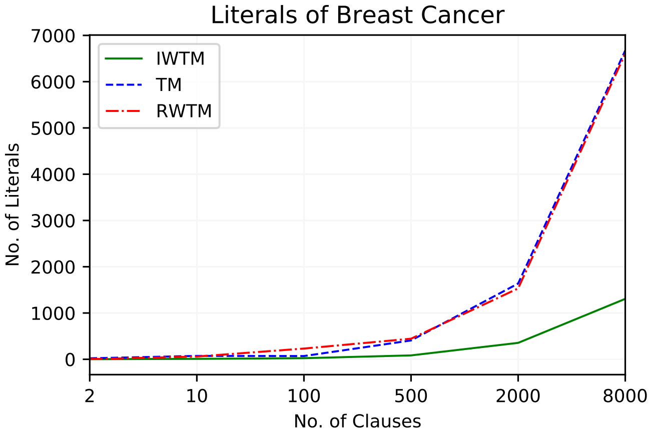


Figure F.9: The number of literals included in different TM setups to work with Breast Cancer dataset.

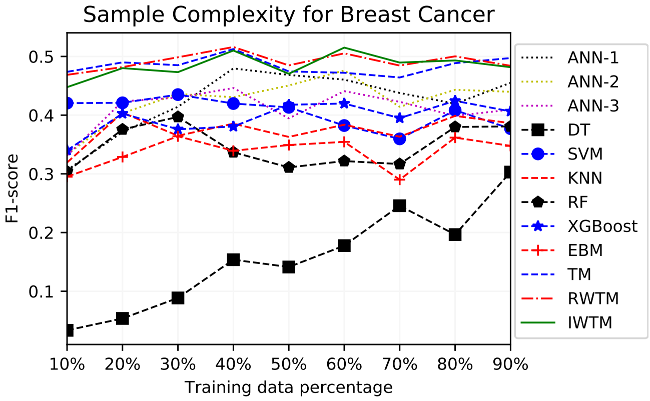


Figure F.10: Sample complexity analysis for the Breast Cancer dataset.

The accuracy and number of literals included for TM, RWTM, and IWTM are respectively summarized in Table F.12, Table F.13, and Table F.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 F.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 F.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 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 F.10.

F.3.4 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 ≥ 3. The Selector attribute is discarded. The remaining attributes represent the results of various blood tests, and we use them as features.

Table F.17, Table F.18, and Table F.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 F.11 plots how the number of literals increases with number of clauses, again confirming that IWTM uses fewer literals overall.

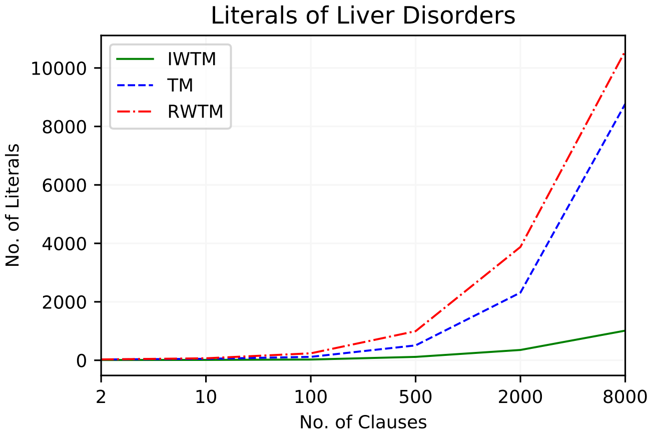


Figure F.11: The number of literals included in different TM setups to work with the Liver Disorders dataset.

Considering the other machine learning techniques (Table F.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.

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 F.12.

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

Table F.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 F.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 F.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

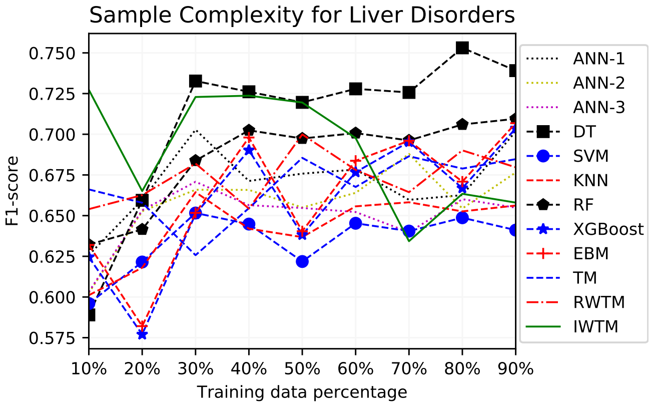


Figure F.12: Sample complexity analysis for the Liver Disorders dataset.

Table F.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

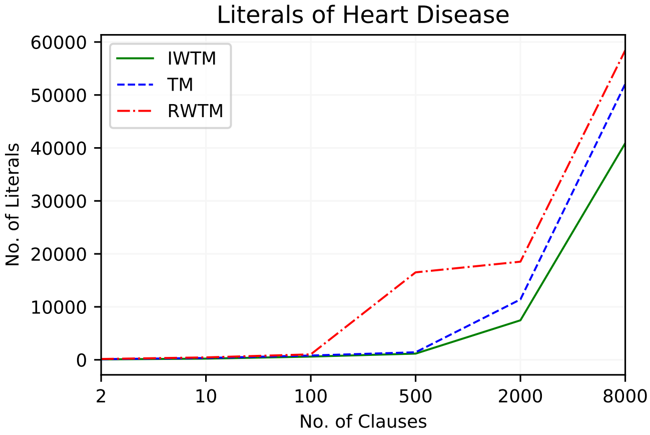


Figure F.13: The number of literals included in different TM setups to work with Heart Disease dataset.

F.3.5 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 F.21, Table F.22, and Table F.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 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 F.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 F.20). However, EBM needs the highest training time and uses the second largest training 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 F.14. From 30% onward, the F1-Score of all the models behaves similarly, with EBM being superior after 60%.

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

Table F.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	-	$\approx$ 973.64KB / $\approx$ 16.46KB	0.297 sec.
ANN-2	0.755	0.736	0.742	0.769	0.791	-	$\approx$ 3659.59KB / $\approx$ 578.11KB	0.266 sec.
ANN-3	0.661	0.662	0.650	0.734	0.784	-	$\approx$ 33952.49KB / $\approx$ 1513.41KB	0.308 sec.
DT	0.827	0.664	0.729	0.781	0.884	-	$\approx$ 0.00KB / $\approx$ 266.23KB	0.016 sec.
SVM	0.693	0.674	0.679	0.710	0.740	-	$\approx$ 1363.96KB / $\approx$ 262.14KB	0.004 sec.
KNN	0.682	0.615	0.641	0.714	0.791	-	$\approx$ 0.00KB / $\approx$ 319.48KB	0.001 sec.
RF	0.810	0.648	0.713	0.774	0.879	-	$\approx$ 413.69KB / $\approx$ 0.00KB	0.017 sec.
XGBoost	0.712	0.696	0.701	0.788	0.863	-	$\approx$ 3694.58KB / $\approx$ 0.00KB	0.057 sec.
EBM	0.827	0.747	0.783	0.824	0.885	-	$\approx$ 4763.64KB / $\approx$ 0.00KB	11.657 sec.
TM	0.607	0.815	0.687	0.672	0.566	346	$\approx$ 0.00KB / $\approx$ 0.00KB	0.014 sec.
RWTM	0.735	0.788	0.757	0.790	0.784	18528	$\approx$ 237.56KB / $\approx$ 0.00KB	1.559 sec.
IWTM	0.694	0.801	0.740	0.743	0.692	226	$\approx$ 0.00KB / $\approx$ 0.00KB	0.010 sec.

Table F.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 F.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 F.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

F.3.6 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.

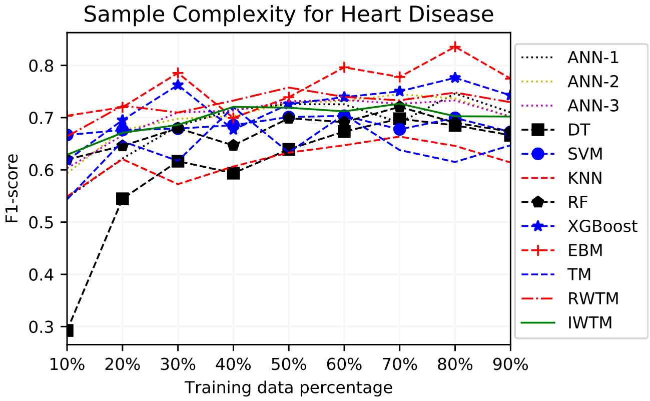


Figure F.14: Sample complexity analysis for the Heart Disease dataset.

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 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.

F.3.7 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 F.24.

Table F.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].

Table F.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]

F.4 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 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.

Bibliography

[1] Christoph Molnar. *Interpretable Machine Learning.* Lulu. com, 2019.

[2] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. “Deep Learning for Healthcare: Review, Opportunities and Challenges”. In: *Briefings in bioinformatics* 19.6 (2018), pp. 1236–1246.

[3] Bart Baesens, Christophe Mues, Manu De Backer, Jan Vanthienen, and Rudy Setiono. “Building Intelligent Credit Scoring Systems Using Decision Tables”. In: *Enterprise Information Systems V.* Springer, 2004, pp. 131–137.

[4] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. “An Empirical Evaluation of the Comprehensibility of Decision Table, Tree and Rule Based Predictive Models”. In: *Decision Support Systems* 51.1 (2011), pp. 141–154.

[5] Riccardo Bellazzi and Blaz Zupan. “Predictive Data Mining in Clinical Medicine: Current Issues and Guidelines”. In: *International journal of medical informatics* 77.2 (2008), pp. 81–97.

[6] Michael J Pazzani, S Mani, and William R Shankle. “Acceptance of Rules Generated by Machine Learning Among Medical Experts”. In: *Methods of information in medicine* 40.05 (2001), pp. 380–385.

[7] Alex A Freitas, Daniela C Wieser, and Rolf Apweiler. “On the Importance of Comprehensible Classification Models for Protein Function Prediction”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7.1 (2008), pp. 172–182.

[8] Duane Szafron, Paul Lu, Russell Greiner, David S Wishart, Brett Poulin, Roman Eisner, Zhiyong Lu, John Anvik, Cam Macdonell, Alona Fyshe, et al. “Proteome Analyst: Custom Predictions With Explanations in a Web-Based Tool for High-Throughput proteome Annotations”. In: *Nucleic acids research* 32.suppl_2 (2004), W365–W371.

[9] Elen Lima, Christophe Mues, and Bart Baesens. “Domain Knowledge Integration in Data Mining Using Decision Tables: Case Studies in Churn Prediction”. In: *Journal of the Operational Research Society* 60.8 (2009), pp. 1096–1106.

[10] Wouter Verbeke, David Martens, Christophe Mues, and Bart Baesens. “Building Comprehensible Customer Churn Prediction Models with Advanced Rule Induction Techniques”. In: *Expert systems with applications* 38.3 (2011), pp. 2354–2364.

[11] Rishabh Agarwal, Nicholas Frosst, Xuezhou Zhang, Rich Caruana, and Geoffrey E Hinton. “Neural Additive Models: Interpretable Machine Learning with Neural Nets”. In: *arXiv preprint arXiv:2004.13912* (2020).

[12] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why Should I Trust You?” Explaining the Predictions of any Classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining.* 2016, pp. 1135–1144.

[13] Cynthia Rudin. “Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead”. In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215.

[14] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. “Mining Association Rules Between Sets of Items in Large Databases”. In: *SIGMOD Rec.* 22.2 (1993), pp. 207–216. ISSN: 0163-5808.

[15] Tyler McCormick, Cynthia Rudin, and David Madigan. “A Hierarchical Model for Association Rule Mining of Sequential Events: An Approach to Automated Medical Symptom Prediction”. In: *Annals of Applied Statistics* (2011).

[16] Vitaly Feldman. “Hardness of Approximate Two-Level Logic Minimization and PAC Learning with Membership Queries”. In: *Jrnl. of Computer and System Sciences* 75.1 (2009), pp. 13–26.

[17] Leslie G Valiant. “A Theory of the Learnable”. In: *Communications of the ACM* 27.11 (1984), pp. 1134–1142.

[18] Tong Wang, Cynthia Rudin, Finale Doshi-Velez, Yimin Liu, Erica Klampfl, and Perry MacNeille. “A Bayesian Framework for Learning Rule Sets for Interpretable Classification”. In: *The Journal of Machine Learning Research (JMLR)* 18.1 (2017), pp. 2357–2393.

[19] John R Hauser, Olivier Toubia, Theodoros Evgeniou, Rene Befurt, and Daria Dzyabura. “Disjunctions of Conjunctions, Cognitive Simplicity, and Consideration Sets”. In: *Jrnl. of Marketing Research* 47.3 (2010), pp. 485–496.

[20] Yitao Liang and Guy Van den Broeck. “Learning Logistic Circuits”. In: *Proceedings of the 33rd AAAI Conference on Artificial Intelligence.* Vol. 33. 2019, pp. 4277–4286.

[21] Ole-Christoffer Granmo. “The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic”. In: *arXiv preprint arXiv:1804.01508* (2018).

[22] Adrian Phoulady, Ole-Christoffer Granmo, Saeed Rahimi Gorji, and Hady Ahmady Phoulady. “The Weighted Tsetlin Machine: Compressed Representations with Clause Weighting”. In: *Ninth International Workshop on Statistical Relational AI (StarAI 2020).* 2020.

[23] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications". In: *IEEE Access* 7 (2019), pp. 115134–115146.

[24] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. "The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression". In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

[25] Saeed Rahimi Gorji, Ole-Christoffer Granmo, Adrian Phoulady, and Morten Goodwin. "A Tsetlin Machine with Multigranular Clauses". In: *Lecture Notes in Computer Science: Proceedings of the Thirty-ninth International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI-2019).* Vol. 11927. Springer International Publishing, 2019.

[26] Saeed Gorji, Ole Christoffer Granmo, Sondre Glimsdal, Jonathan Edwards, and Morten Goodwin. "Increasing the Inference and Learning Speed of Tsetlin Machines with Clause Indexing". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer. 2020.

[27] Adrian Wheeldon, Rishad Shafik, Alex Yakovlev, Jonathan Edwards, Ibrahim Haddadi, and Ole-Christoffer Granmo. "Tsetlin Machine: A New Paradigm for Pervasive AI". In: *Proceedings of the SCONA Workshop at Design, Automation and Test in Europe (DATE).* 2020.

[28] Michael Lvovitch Tsetlin. "On Behaviour of Finite Automata in Random Medium". In: *Avtomat. i Telemekh* 22.10 (1961), pp. 1345–1354.

[29] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. "The Convolutional Tsetlin Machine". In: *arXiv preprint:1905.09688* (2019).

[30] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. "A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer. 2019, pp. 564–578.

[31] B John Oommen. "Stochastic Searching On the Line and Its Applications to Parameter Learning in Nonlinear Optimization". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27.4 (1997), pp. 733–739.

[32] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. "InterpretML: A Unified Framework for Machine Learning Interpretability". In: *arXiv preprint arXiv:1909.09223* (2019).

[33] Yin Lou, Rich Caruana, and Johannes Gehrke. "Intelligible Models for Classification and Regression". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining.* 2012, pp. 150–158.

[34] Brian Lucena. "StructureBoost: Efficient Gradient Boosting for Structured Categorical Variables". In: *arXiv preprint arXiv:2007.04446* (2020).

[35] Kumpati S Narendra and Mandayam AL Thathachar. *Learning Automata: An Introduction*. Courier corporation, 2012.

[36] Ian Chivers and Jane Sleightholme. "An Introduction to Algorithms and the Big O Notation". In: *Introduction to Programming with Fortran*. Springer, 2015, pp. 359–364.

[37] Tianqi Chen and Carlos Guestrin. "Xgboost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[38] Myoung-Jong Kim and Ingoo Han. "The Discovery of Experts' Decision Rules From Qualitative Bankruptcy Data Using Genetic Algorithms". In: *Expert Systems with Applications* 25.4 (2003), pp. 637–646.

[39] James McDermott and Richard S Forsyth. "Diagnosing a Disorder in a Classification Benchmark". In: *Pattern Recognition Letters* 73 (2016), pp. 41–43.

[40] Brian Lucena. "Exploiting Categorical Structure Using Tree-Based Methods". In: *arXiv preprint arXiv:2004.07383* (2020).

[41] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "On Obtaining Classification Confidence, Ranked Predictions and AUC with Tsetlin Machines". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2020.

[42] B John Oommen and Govindachari Raghunath. "Automata Learning and Intelligent Tertiary Searching for Stochastic Point Location". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 28.6 (1998), pp. 947–954.

[43] Wen Jiang, De-Shuang Huang, and Shenghong Li. "Random Walk-Based Solution to Triple Level Stochastic Point Location Problem". In: *IEEE transactions on cybernetics* 46.6 (2015), pp. 1438–1451.

Paper G

On Obtaining Classification Confidence, Ranked Predictions and AUC with Tsetlin Machines

Tsetlin machines (TMs) are a promising approach to machine learning that uses Tsetlin Automata to produce patterns in propositional logic, leading to binary (hard) classifications. In many applications, however, one needs to know the confidence of classifications, e.g. to facilitate risk management. In this paper, we propose a novel scheme for measuring TM confidence based on the logistic function, calculated from the propositional logic patterns that match the input. We then use this scheme to trade off precision against recall, producing area under receiver operating characteristic curves (AUC) for TMs. Empirically, using four real-world datasets, we show that AUC is a more sensitive measure of TM performance compared to Accuracy. Further, the AUC-based evaluations show that the TM performs on par or better than widely used machine learning algorithms. We thus believe our scheme will make the TM more suitable for use in decision support, where the user needs to inspect and validate predictions, in particular, those being uncertain.

G.1 Introduction

The demand for *interpretable* machine learning methods is rapidly increasing due to widespread deployment in high-risk domains where the consequences of errors can be severe. Examples of such domains are credit scoring [1, 2], medical treatment [3, 4], bioinformatics [5, 6], and churn prediction [7, 8]. These domains demand transparency of predictions, in addition to reliable metrics for performance assessment, including Precision, Recall, F1-Score, and Accuracy [9, 10, 11].

Tsetlin machines (TMs) [12, 13, 14] are a recent rule-based machine learning approach that forms a high-accuracy alternative to widely used interpretable techniques, such as Linear Regression, Logistic Regression and Decision Trees, which suffers from poor accuracy [15]. When compared to state-of-the-art machine learning techniques like Convolutional Neural Networks, LSTM, Random Forest, and XGBoost, TMs exhibit competitive performance with regard to accuracy [14, 13, 16], memory footprint [16, 17], energy [17], and learning speed [16, 17]. However, while TMs support both global and local interpretability [18], decisions are binary (hard), making them less suitable for applications that require trading off precision against recall.

Indeed, Huang et al. [19] and Bradley [20] stress the importance of receiver operating characteristic (ROC) curves and area under ROC (AUC) for assessing machine learning techniques, as they capture the relationship between sensitivity and specificity. Further, getting access to the confidence of classifications can be important in many applications. One example is direct marketing, where it may be advantageous to approach the most likely buyers first, given limited time and resources. Accordingly, the ranking of predictions, based on confidence, opens up for additional applications than mere classification [21].

Paper Contributions and Organization: In this paper, we propose a novel approach to measuring the classification confidence of TMs. In Section G.2, we introduce the basics of TMs. Then, in Section G.3, we propose an approach for calculating a classification confidence score from the propositional logic expressions that match the input. Based on the confidence score, we demonstrate how to rank predictions, producing ROC curves and calculating AUC in Section G.4. Then, in Section G.5, we show that AUC is a more sensitive measure of TM performance compared to Accuracy. We further provide AUC-based evaluations that show that the TM performs on par or better than widely used machine learning algorithms: Decision Tree (DT), Support Vector Machine (SVM), Logistic Regression (LR), K-Nearest Neighbor (KNN), Random Forest (RF), and Artificial Neural Networks (ANNs). Finally, we conclude in Section G.6, summarizing our key findings.

G.2 The Tsetlin Machine

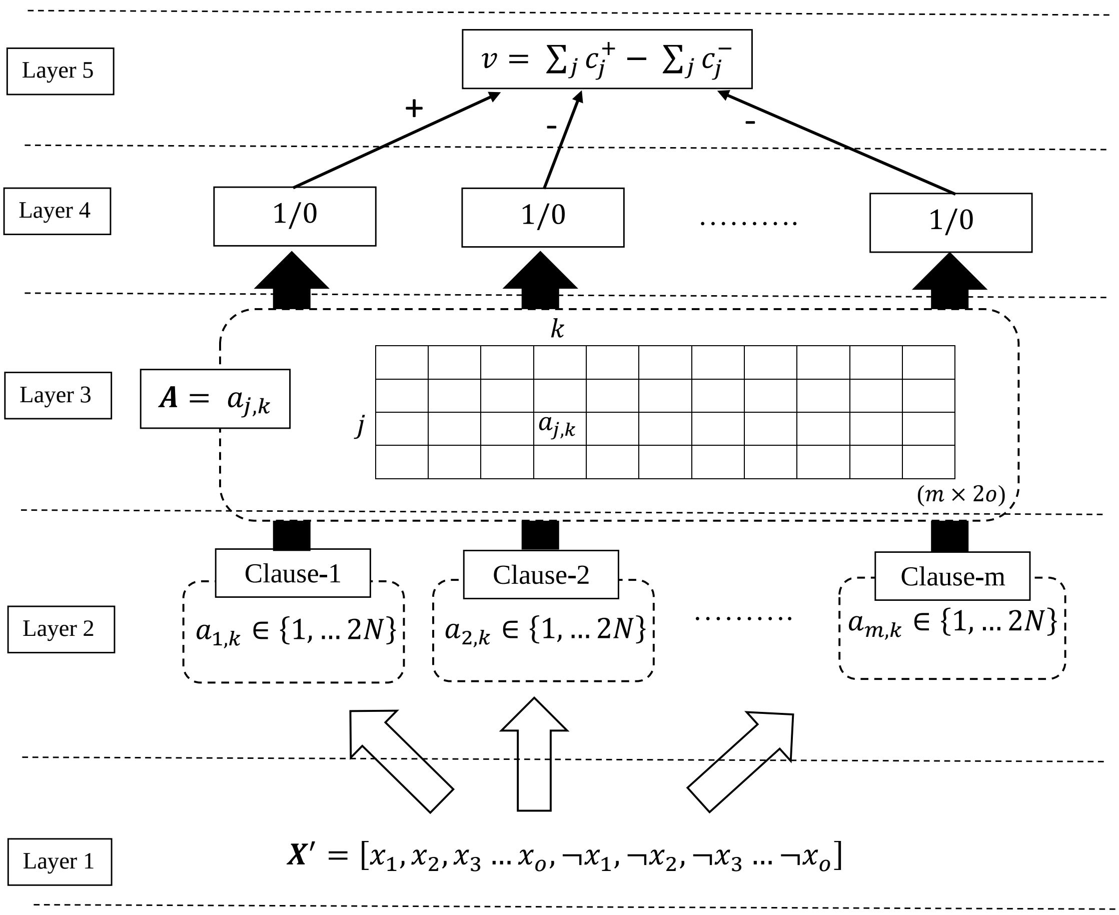
Conceptually, a TM consists of five layers, as illustrated by Figure G.1. In this section, we first explain the role of each of these layers when performing classification. We then go into the details of TM learning.

G.2.1 Layer 1 – Input Layer

A TM takes a feature vector $\mathbf{X} \in \{0,1\}^o$ of o propositional variables $x_k \in \{0,1\}$ as input. To increase expression power, this feature vector is extended with the negation of the original features: $\mathbf{X}' = [x_1, x_2, x_3, \ldots, x_o, \neg x_1, \neg x_2, \neg x_3, \ldots, \neg x_o]$. Jointly, the elements of the extended feature vector $\mathbf{X}'$ are referred to as literals.

G.2.2 Layer 2 – Clause Layer

The clause layer processes literals from the input layer. To this end, the clause layer comprises m conjunctive clauses, which are to capture sub-patterns in the data. Each



Layer 5
$v = \sum_j c_j^+ - \sum_j c_j^-$
+
-
-
Layer 4
1/0
1/0
....
1/0
Layer 3
$A = a_{j,k}$
k
j
$a_{j,k}$
$(m \times 2o)$
Layer 2
Clause-1
$a_{1,k} \in \{1, \ldots 2N\}$
Clause-2
$a_{2,k} \in \{1, \ldots 2N\}$
....
Clause-m
$a_{m,k} \in \{1, \ldots 2N\}$
Layer 1
$X' = [x_1, x_2, x_3 \ldots x_o, \neg x_1, \neg x_2, \neg x_3 \ldots \neg x_o]$

Figure G.1: The TM structure.

conjunctive clause j is defined by the literals it includes:

$$c_j = 1 \land \left(\bigwedge_{k \in I_j^I} x_k \right) \land \left(\bigwedge_{k \in \bar{I}_j^I} \neg x_k \right). \tag{G.1}$$

Above, the set I_j^I contains the indexes of the *original* variables that are included in clause j. Similarly, the set $\bar{I}_j^I$ consists of the indexes of the included *negated* variables. These sets are thus subsets of the complete set of indexes $I_j^I, \bar{I}_j^I \subseteq \{1, \ldots, o\}$.

G.2.3 Layer 3 – Memory Layer

The decisions of including or excluding literals in clauses are made by two-action Tsetlin Automata (TAs) [22]. That is, $2 \times o$ TAs are attached to each clause, one per literal $x'_k, k \in \{1, \ldots, 2o\}$. Each TA maintains a memory state $a_{j,k} \in \{1, \ldots, 2N\}$, with j referring to the clause and k to the literal. States 1 to N map to the *exclude* action: exclude the k^{th} literal from the j^{th} clause. Conversely, states $N + 1$ to $2N$ map to the *include* action: include the k^{th} literal in the j^{th} clause.

The memory layer is responsible for keeping track of all the TA states, which can be organized as a matrix $\mathbf{A}$: $\mathbf{A} = (a_{j,k}) \in \{1, \ldots, 2N\}^{m \times 2o}$. Once the states of the TAs are given, the elements in the index set I_j^I can be written as: $I_j^I = \{k | a_{j,k} > N, 1 \leq k \leq 2o\}$.

Algorithm 5 Clause Learning

```
**input** Training example $(\mathbf{X}', y)$, voting sum $v$, clause output $c_j$, positive polarity indicator $p_j \in \{0, 1\}$, voting target $T \in [1, \infty)$, pattern specificity $s \in [1.0, \infty)$
procedure UpdateClause($\mathbf{X}', v, c_j, p_j, T, s$)
    $v^c \leftarrow \mathbf{clip}\,(v, -T, T)$
    $e = T - v^c$ if $y_i = 1$ else $T + v_i^c$
    if rand() $\leq \frac{e}{2T}$ then
        if $y_i$ xor $p_j$ then
            TypeIIFeedback($\mathbf{X}', c_j$)
        else
            TypeIFeedback($\mathbf{X}', c_j, s$)
        end if
    end if
end procedure
```

G.2.4 Layer 4 – Voting Layer

The input feature vector provides the literal values and the TA decisions compose the clauses, which can then be evaluated. Since the clauses are conjunctive, a clause evaluates to 1 if and only if all of the included literals are of value 1. Let $I_{X'}^1$ contain the indexes of the 1-valued literals from $\mathbf{X}'$. The value c_j of clause j can then be succinctly defined as:

$$c_j = \begin{cases} 1 & \text{if} \quad I_j^I \subseteq I_{X'}^1, \\ 0 & \text{otherwise.} \end{cases} \tag{G.2}$$

We finally represent the complete collection of clause outputs in vector form: $\mathbf{C} = (c_j) \in \{0, 1\}^m$.

A two-class TM organizes the m clauses in two groups of equal size. Clauses with odd indexes are to capture the sub-patterns of class $y = 1$ and they are given positive polarity, denoted c_j^+. The clauses with even indexes, on the other hand, are to capture sub-patterns of class $y = 0$. These are given negative polarity, denoted c_j^-.

G.2.5 Layer 5 – Output Layer

The output layer receives the polarized clause outputs from the voting layer, which are aggregated into a majority vote: $v = \sum_j c_j^+ - \sum_j c_j^-$. The output of the TM is finally decided as follows:

$$y = \begin{cases} 1 & \text{if} \quad v \geq 0 \\ 0 & \text{if} \quad v < 0 \,. \end{cases} \tag{G.3}$$

That is, if the negative clauses of value 1 (voting for $y = 0$) outnumber the positive clauses of value 1 (voting for $y = 1$), the final output becomes $y = 0$. Otherwise, it becomes $y = 1$.

G.2.6 Learning Procedure

Recall that a clause j is composed by its attached team of TAs and that it is TA state $a_{j,k}$ that decides whether literal x'_k is included in clause j. Learning which literals to include is based on two types of reinforcement: Type I and Type II. As described in the following, Type I feedback produces frequent patterns, while Type II feedback increases the discrimination power of the patterns.

TMs learn on-line, processing one training example $(\mathbf{X}, y)$ at a time. In all brevity, after a forward pass through the layers described above, each clause is updated according to Algorithm 5.

The first step is to decide whether the clause is to be updated (Lines 2-4). Here, resource allocation dynamics ensure that clauses distribute themselves across the frequent patterns, rather than missing some and over-concentrating on others. That is, for any input $\mathbf{X}'$, the probability of reinforcing a clause gradually drops to zero as the voting sum v approaches a user-set target T for $y = 1$ (and $-T$ for $y = 0$).

As seen, if a clause is not reinforced, it does not give feedback to its TAs, and these are thus left unchanged. In the extreme, when the voting sum v equals or exceeds the target T (the TM has successfully recognized the input $\mathbf{X}'$), no clauses are reinforced. They are then free to learn new patterns, naturally balancing the pattern representation resources [12].

If a clause is going to be updated, the updating is either of Type I or Type II (Lines 5-9):

Type I feedback is given to clauses with positive polarity when $y = 1$ and to clauses with negative polarity when $y = 0$ (Line 8). Each TA of the clause is then reinforced based on: (1) the clause output c_j; (2) the action of the TA – *include* or *exclude*; and (3) the value of the literal x'_k assigned to the TA. Two rules govern Type I feedback:

- *Include* is rewarded and *exclude* is penalized with probability $\frac{s-1}{s}$ **if** $c_j = 1$ **and** $x'_k = 1$. If this happens, the corresponding TA state $a_{j,k}$ is increased by 1, up to $2N$. This reinforcement is strong (triggered with high probability) and makes the clause remember and refine the pattern it recognizes in $\mathbf{X}'$.[1]

- *Include* is penalized and *exclude* is rewarded with probability $\frac{1}{s}$ **if** $c_j = 0$ **or** $x'_k = 0$. If this happens, the corresponding TA state $a_{j,k}$ is decreased by 1, down to 1. This reinforcement is weak (triggered with low probability) and coarsens infrequent patterns, making them frequent.

Above, hyper-parameter s controls pattern frequency.

Type II feedback is given to clauses with positive polarity when $y = 0$ and to clauses with negative polarity when $y = 1$ (Line 6). It penalizes *exclude* with probability 1 **if** $c_j = 1$ **and** $x'_k = 0$. If this happens, the corresponding TA state $a_{j,k}$ is increased by 1. Thus, this feedback introduces literals for discriminating between $y = 0$ and $y = 1$.

In addition to the vanilla TM discussed in this section, there also exists several other architectures: Multiclass TM [12], Regression TM [13, 23], Convolutional TM, [16], Multi-Layered TM [24], Weighted Tsetlin Machine [25], Integer Weighted Regression TM [26],

[1] Note that the probability $\frac{s-1}{s}$ is replaced by 1 when boosting true positives.

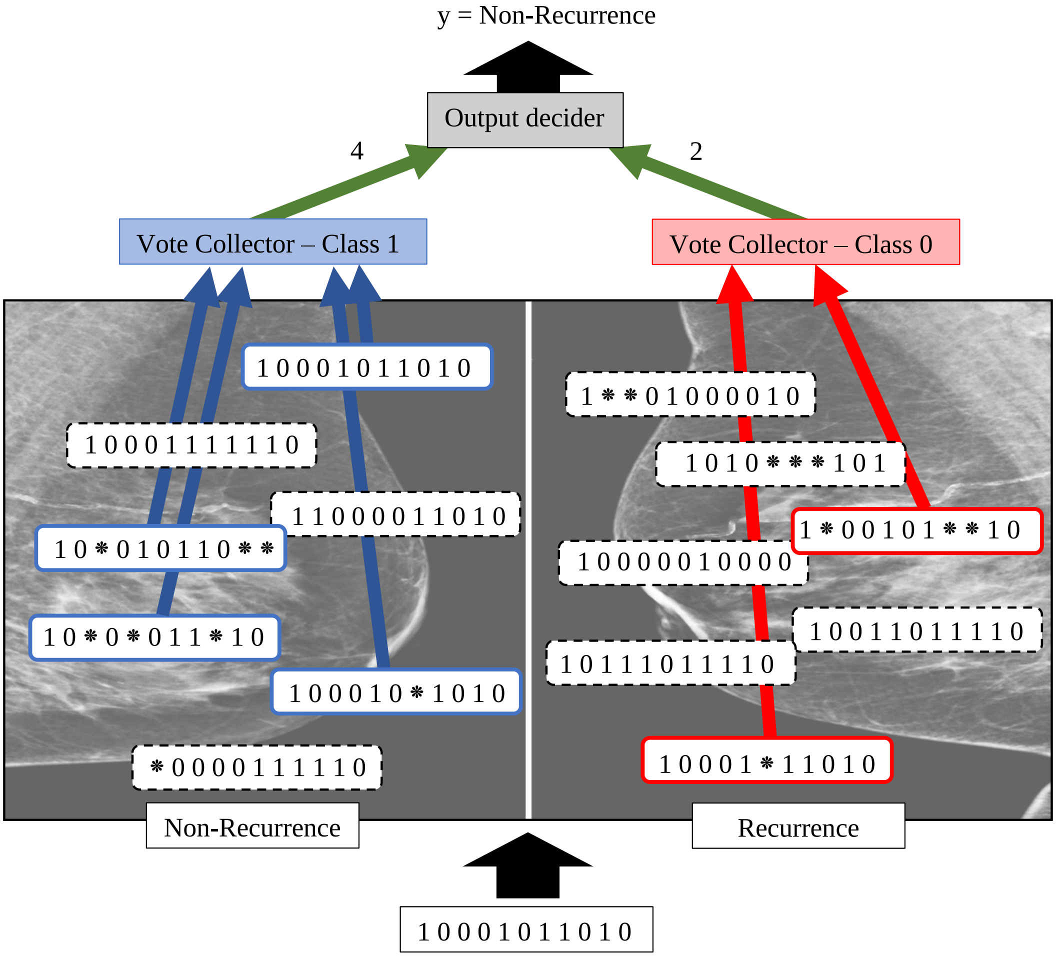


Figure G.2: The process of making predictions from the trained Tsetlin Machine.

and Integer Weighted TM [27]. Even though we in the following discuss the confidence of classifications and AUC calculations for the vanilla TM, the calculations can also be extended to the other TM architectures, except for the Regression TM.

G.3 TM Classification Confidence

G.3.1 TM Predictions

Consider the machine learning example illustrated in Figure G.2. The figure depicts a TM trained to recognize the recurrence of breast cancer based on a dataset of eleven binary features. The TAs of each clause have already made their decisions regarding which literals to include. As seen, we have allocated seven clauses to each class (recurrence or non-recurrence), each recognizing a specific pattern.

Consider the sub-pattern (1 0 0 0 1 0 $\ast$ 1 0 1 0) captured by one of the clauses supporting class $y = 1$ (Non-Recurrence) in the figure. Here, the pattern value 0 means that the TA of the corresponding negated feature has decided to include the negated feature in the clause. Conversely, the pattern value 1 means the TA of the original feature has decided to include the feature unnegated. The pattern value $\ast$, however, means that both the corresponding unnegated and the negated feature have been excluded. Representing the pattern as a clause, we get: $(x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge x_5 \neg x_6 \wedge x_8 \wedge \neg x_9 \wedge x_{10} \wedge \neg x_{11})$. Notice that neither the unnegated or negated versions of feature 7 are included.

The above clause outputs 1 for the example input [1 0 0 0 1 0 1 1 0 1 0]. Similarly, three of the other clauses associated with class $y = 1$ in the figure output 1, as well. In total, four clauses thus vote for class $y = 1$, collected by the 'Vote Collector' of class 1. Note further that two of the clauses supporting class $y = 0$ (Recurrence) also recognizes this input. These vote for class $y = 0$ and therefore against class $y = 1$, tallied by the 'Vote Collector' for class 0. In this case, class $y = 0$ is outvoted by class $y = 1$, hence the output layer of the TM (Eq. (G.3)) decides to output $y = 1$ (Non-Recurrence).

G.3.2 Calculating Confidence

Taking the above vote ratio of 4:2 (class 1 vs class 0) as a reference ratio, consider the vote ratios 7:0 and 1:1. According to Eq. (G.3), the output of the TM still remains the same. However, considering the vote difference, we can argue that a 7:0 ratio represents stronger confidence compared to e.g. the ratio 4:2 or 1:1. We therefore modify the prediction mechanism in the output layer to manage confidence, leveraging the vote totals of the individual vote collectors (or TMs for multi-class problems). To this end, we propose to replace Eq. (G.3) with the *logistic function* in Eq. (G.4). Depending on the vote difference, the logistic function outputs the confidence of the classification, measured as a value between 0 and 1.

$$Q_{1,i} = \frac{1}{1 + e^{-(v_i^p - v_i^n)}}. \tag{G.4}$$

In Eq. (G.4), $Q_{1,i}$ is the confidence of classifying the input i as class $y = 1$. The values v_i^p and v_i^n are the number of positive and negative votes collected by the "Vote Collectors" of class 0 and class 1 for the input i, respectively. For instance, using Eq. (G.4), we can calculate classification confidence for the vote ratios 4:2, 7:0, and 1:1, which are 0.88, 0.99, and 0.5, respectively. Hence, the confidence of the classifications can be ranked as 1:1 < 4:2 < 7:0 since their respective confidence scores are 0.5 < 0.88 < 0.99.

G.3.3 Prediction Ranking and Classification Thresholds

The confidence score in Eq. (G.4) has several uses. When the task is to classify input into two classes, the confidence in the target class decreases from score 1.0 to 0.5 and then increases again from 0.5 to 0.0, towards the other class. In this way, predictions can be ranked according to confidence.

Confidence scoring can further be used to set decision thresholds to achieve desired specificity-sensitivity ratios, giving priority to one of the classes. Such prioritization can be beneficial in applications where one knows the cost of misclassification. Confidence score thresholds can then be set to minimize expected misclassification cost. Note that when misclassification cost is unavailable, it is common to set the decision threshold to 0.5. In that case, classification follows Eq. (G.3).

Finally, ranked predictions can be used to produce ROC-curves and compute AUC scores, as explained in the following.

G.4 Calculating AUC for TMs

AUC is an important evaluation metric for binary classifiers because it considers how accurately a classifier can rank the confidence of its classifications [19]. AUC is based on ROC curves, which captures how the true positive rate ($=\frac{\text{true positives}}{\text{total positives}}$) varies with the false positive rate ($=\frac{\text{false positives}}{\text{total negatives}}$), at all possible decision thresholds. In all brevity, AUC measures the entire two-dimensional area under the ROC curve.

Based on the ranked predictions introduced in the previous section, we are now equipped to calculate AUC for TMs. One method for calculating AUC is the trapezoidal integration approach. However, as Hanley et al. explains [28], the trapezoidal method systematically underestimates AUC. Hence, in this study, we calculate AUC using Eq. (G.5), introduced by [29]:

$$\text{AUC} = \frac{S_0 - n_0(n_0+1)/2}{n_0 n_1}. \tag{G.5}$$

Above, the parameters n_o and n_1 are the number of positive (1s) and negative (0s) samples, respectively. The ranking of the inputs are taken into account by the parameter S_0, where S_0 is the sum of the rankings of all positive inputs, r_i, i.e, $S_0 = \sum r_i$.

G.5 Evaluation

In this section, we contrast AUC against Accuracy as evaluation metrics for TMs. We do this by computing *Degree of Consistency* and *Degree of Discriminancy*, as proposed in [19]. Further, we explore how our TM confidence score can be used to facilitate decision making. Finally, we provide AUC-based evaluations that show that the TM performs on par or better than several widely used machine learning algorithms.

For the experiments in this section, we use four datasets from real-world applications where expert support is needed to quality assure decisions:

Dataset I - Bankruptcy[2] considers historical cases of 250 companies, described by the features: Industrial Risk, Management Risk, Financial Flexibility, Credibility, Competitiveness, and Operation Risk. These are categorical features where each feature can take one of three states: Positive, Average, or Negative. The categorical outputs are Bankruptcy and Non-bankruptcy.

Dataset II - Breast cancer[3] covers recurrence of breast cancer, characterized by nine features, i.e., age, menopause, tumor size, inv nodes, node caps, deg malig, side (left of right), position of the breast, and irradiation status. The dataset contains 286 patients (201 of non-recurrence and 85 of recurrence), however, some of these lacks features and are thus not considered here.

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

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

Dataset III - Liver disorders[4] covers 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). In our study, the correct dependent variable [30] – number of alcoholic drinks taken per day by the subject – is binarized with the threshold ≥ 3. The selector attribute is discarded. The remaining attributes represent the results of various blood tests, and are used as input features.

Dataset IV - Heart disease[5] contains 270 observations, labeling whether a person is having heart disease or not. The presence or absence of heart disease is characterized by 13 attributes, selected from a total of 75. Out of the 13 attributes, 6 are real-valued, 3 are binary, 3 are nominal, and the remaining one is ordered.

G.5.1 AUC vs Accuracy

The degree of consistency and degree of discriminancy of AUC and Accuracy are measured separately, per Definition 1 and Definition 2, respectively (generalized versions of the ones given by [19]):

Definition 1 Degree of Consistency *is a probabilistic measure which defines the consistency of two measures. When two measures f and g are to evaluate algorithms a and b, the consistency can be calculated as*

$$\mathbf{C} = \frac{R}{R+S}. \tag{G.6}$$

Here, R is the number of similar classifications, e.g., $f(a) > f(b)$ and $g(a) > g(b)$, and S is the number of dissimilar classifications, e.g., $f(a) > f(b)$ and $g(a) < g(b)$.

Definition 2 Degree of Discriminancy *is a probabilistic measure which defines the discriminancy of two measures. When two measures f and g are to evaluate algorithms a and b, the discriminancy of f over g can be calculated as*

$$\mathbf{D} = \frac{V}{U}. \tag{G.7}$$

Above, V is the number of classifications in which f discriminates between a and b, but g cannot, e.g., $f(a) > f(b)$ and $g(a) = g(b)$, while U is the number of classifications in which g discriminates between a and b, but f cannot, e.g., $g(a) > g(b)$ and $f(a) = f(b)$.

In order to measure consistency, we train two TMs with greatly different hyper-parameter settings on all four datasets. The purpose is to measure how often AUC and Accuracy agrees on the ranking of the two TMs. For measuring disciminancy, on the other hand, two TMs with nearly identical hyper-parameter settings are used. The intent here is to see how often AUC is able to discriminate between the two TMs, relative to how many times Accuracy is able to do so. Each experiment is conducted 100 times to

[4] Available from https://archive.ics.uci.edu/ml/datasets/Liver+Disorders1

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

Table G.1: Consistency of AUC and Accuracy.

Dataset	Similar classifications	Dissimilar classifications	**C**
I	100	1	1.00
II	72	28	0.72
III	64	36	0.64
IV	91	9	0.91

Table G.2: Discriminance power of AUC vs. Accuracy.

Dataset	AUC discriminates but Accuracy cannot	Accuracy discriminates but AUC cannot	**D**
I	21	10	2.1
II	20	1	20.0
III	55	1	55.0
IV	16	1	16.0

calculate the R, S, V, and U values according to Definition 1 and Definition 2. In each experiment, a randomly selection of 60 samples are used for testing.

Table G.1 and Table G.2 show the experiment results for the consistency test and discriminancy tests, respectively. As one can see in Table G.1, Accuracy and AUC is agreeing more often than not, for all considered datasets. The highest level of agreement is 1.00, which is obtained on the Bankruptcy dataset. The lowest level of agreement is observed for the Liver disorders dataset where there is a 0.64 chance that Accuracy agrees with AUC.

Even though Accuracy to a large degree agrees with AUC, AUC has a higher discrimination power, as seen in Table G.2. The highest degree of discriminancy occurs with the Liver disorders dataset where AUC finds the difference between the two TMs 55 times more often than Accuracy does. The lowest degree of discriminancy happens on the Bankruptcy dataset where AUC discriminates between the two TMs 2.1 times more often than Accuracy does.

Considering the degree of consistency and the degree of discriminancy for the different datasets, AUC appears to be better suited for evaluating performance compared to Accuracy.

G.5.2 Explainable Rules and Decision Support

In this section, we first show how interpretable rules are formed from a trained TM, using the Bankruptcy dataset as an example. Then we discuss how the confidence of classifications can help experts in their decision-making process.

Consider the Bankruptcy dataset. As the first step, the categorical features are integer-encoded and binarized. For binarization, we employ the thresholding approach proposed in [31]. Since each feature has three categorical values, we can represent each original feature with three binary features, using Average (0), Negative (1), and Positive (2) as thresholds. The categorical value Average is represented by 111, the categorical value Negative is represented by 011, and the categorical value Positive is represented by 001. Therefore, the six original features are now represented by 18 binary features.

With merely 10 clauses, the TM achieves 98% training accuracy and 100% testing accuracy. The composition of the clauses after training is summarized in Table G.3. Out of the ten clauses, clauses with odd index vote for class 1 (*Non-Bankruptcy*) and clauses with even index vote for class 0 (*Bankruptcy*). As seen, the clauses voting for class 0 (2, 4, 6, 8, 10) represent the same sub-pattern, which is *Competitiveness is not Average and Competitiveness is Negative.* This rule can be further simplified as *Competitiveness is Negative.* For class 1, the identified sub-patterns are *Credibility is not Negative* (clause 1) and *Competitiveness is not Negative* (clauses 3, 5, and 9). There is also a free vote (all literals are excluded) for class 1 from clause 7.

Therefore, when Competitiveness is Negative, class 0 gets 5 votes. Yet, due to clause 3, 5, and 9, which recognizes *Competitiveness is not Negative*, the maximum number of votes class 1 can get is 2 (including the free vote from clause 7). Hence, if Competitiveness is Negative, regardless of the Credibility value, the output is class 0. On the other hand, if Competitiveness is not Negative, class 0 gets 0 votes and class 1 gets a minimum of 4 votes, without the vote from clause 1. Therefore, the output is class 1. Hence, the pattern recognized with clause 1 (*Credibility is not Negative*) can simply be ignored and the resulting TM classifier can be written as:

$$\mathrm{y} = \begin{cases} \text{Bankruptcy} & \text{if Competitiveness is Negative} \\ \text{Non-bankruptcy} & \text{Otherwise.} \end{cases} \tag{G.8}$$

With maximum possible testing accuracy, combined with easily understandable classification rules, we can argue that the model is reliable. However, trust in the model can be increased further by introducing *local* reliability, taking advantage of the classification confidence score. Local reliability refers to the reliability of a specific classification.

As we discussed in Section G.3.3, using two confidence-level-thresholds, we can characterize classifications as reliable or unreliable. With sufficiently conservative thresholds, only unreliable classifications needs to be investigated by experts, while the reliable classifications can simply be accepted as is. We here investigate the effect of different confidence-level thresholds. Threshold-1 characterize a classification as reliable if the classification confidence is higher than 0.90 or lower than 0.10 (< 0.10 or > 0.90), otherwise, it is characterized as unreliable. Likewise, three other thresholds are created: Threshold-2 - (<0.20 or >0.80), Threshold-3 - (<0.30 or >0.70), and Threshold-4 - (<0.40 or >0.60). Accordingly, classifications decided to be reliable with Threshold-1 are more reliable than classifications decided to be reliable with Threshold-4.

We now investigate the effect of the different thresholds for each of the datasets. The results are summarized in Table G.4. For each dataset, 60 random samples are tested 100 times. Hence, the ”Reliable samples” column contains the number of samples, out of 6000 testing samples, characterized as reliable according to each threshold.

For the Bankruptcy dataset, Threshold-1 characterises 99.68% of the classifications as reliable. These have a precision of 0.998. Likewise, Threshold-1 characterises 50% of recurrence of breast cancer classifications as reliable, however, these only have a precision

Table G.3: Clause outputs at end of TM training on Bankruptcy dataset when $m = 10$.

Original feature	1			2			3			4			5			6		
Binarized feature	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}
Clause 1	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	∗	∗	∗	∗	∗	∗	∗
Clause 2	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	1	∗	∗	∗	∗
Clause 3	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	∗	∗	∗	∗
Clause 4	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	1	∗	∗	∗	∗
Clause 5	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	∗	∗	∗	∗
Clause 6	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	1	∗	∗	∗	∗
Clause 7	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗
Clause 8	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	1	∗	∗	∗	∗
Clause 9	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	∗	∗	∗	∗
Clause 10	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	∗	0	1	∗	∗	∗	∗

Table G.4: The number of reliable samples (out of 6000) according to different confidence thresholds.

Dataset	Threshold	Reliable samples	Percentage	Precision
I	1	5981	99.68	0.998
	2	5991	99.85	0.997
	3	5997	99.96	0.996
	4	5997	99.96	0.996
II	1	2997	49.95	0.643
	2	3926	65.43	0.574
	3	5243	87.38	0.474
	4	5243	87.38	0.474
III	1	3450	57.50	0.824
	2	4428	73.80	0.699
	3	5491	91.52	0.688
	4	5491	91.52	0.688
IV	1	5218	86.97	0.772
	2	5538	92.30	0.753
	3	5837	97.28	0.739
	4	5837	97.28	0.739

of 0.643. The Liver disorder dataset provides 57.50% reliable classifications, providing a precision of 0.824. In general, depending on the costs of miss-classification, an appropriate threshold can be selected.

G.5.3 TM Compared to Baseline Models

We here compare the performance of the TM in terms of AUC against six other machine learning algorithms: Decision Tree (DT), Support Vector Machine (SVM), Logistic Regression (LR), K Nearest Neighbor (KNN), Random Forecasts (RF), and Artificial Neural Network (ANNs). For comprehensiveness, three ANN architectures are used: ANN-1 – with one hidden layer and 5 neurons in it, ANN-2 – with two hidden layers with 20 and 50 neurons in them, respectively, and ANN-3 – with three hidden layers and 20, 150, and 100 neurons in them, respectively.

A summary of the results is given in Table G.5. The results are the mean of AUC measures after conducting each experiment 100 times. In each trial, 60 randomly selected data samples are used for testing and the rest is used for training. The hyperparameters of all these machine learning algorithms are found by using binary search.

The TM used on the Bankruptcy dataset employs 1000 clauses. The s value and target T are set to 2.0 and 12, respectively. In each experiment, the TM is run for 400 epochs. For this dataset, TM achieves 1.00 AUC. However, the 1.00 AUC is achieved by all other machine learning algorithms as well, except DT.

For the Breast cancer dataset, a TM with 500 clauses is utilized. The s value and target T for this TM are set to 1.5 and 10, respectively. The number of epochs per

Table G.5: Performance of various machine learning algorithms in terms of AUC on four datasets.

Algorithm / Dataset	I	II	III	IV
DT	0.99	0.57	0.58	0.72
SVM	1.00	0.69	0.67	0.84
LR	1.00	0.71	0.71	0.87
KNN	1.00	0.65	0.58	0.74
RF	1.00	0.69	0.56	0.87
ANN-1	1.00	0.71	0.66	0.83
ANN-2	1.00	0.67	0.65	0.85
ANN-3	1.00	0.66	0.62	0.80
TM	1.00	0.71	0.57	0.89

experiment is 400. The TM is able to reach 0.71 AUC which is a tied best among LR, ANN-1, and TM. The lowest AUC is obtained by KNN.

The hyper-parameters of the TM for the Liver disorders dataset are: 500 clauses, an s value of 1.8, and a target T of 10. Mean AUC after 100 experiment rounds ends at 0.57, which is only higher than what is obtained with RF. However, the other schemes are close by, with AUCs varying around 60. The exception is LR, which attains an AUC score of 0.71.

The TM achieves the best AUC score for the Heart disease dataset – an AUC of 0.89. The TM employed here uses 1000 clauses. The s value is set to 2.0 and the target T to 12. Of the other algorithms, LR and RF share the second-best AUC. The lowest AUC for this dataset is observed for DT.

To summarize, out of the four tasks, the TM obtains the highest mean AUC on three out of five occasions, including two ties. At the same time, for some of the data sets, the competitive AUC is obtained with a relatively small number of rules, as demonstrated in Section G.5.2.

G.6 Conclusion

This paper introduced a novel scheme for measuring the confidence of Tsetlin Machine classification. The confidence is calculated directly from the clauses that recognize the input, using the logistic function. This allowed us to rank predictions, opening up for calculating the area under receiver operating characteristic curve (AUC) for TMs. Empirically, using four real-world datasets, our results show that AUC is a better evaluation measure for TMs than Accuracy. We further explored how ranking classifications by confidence can be used to single out reliable and unreliable classifications, to aid focusing the attention of human experts. Finally, we investigated performance of TMs in terms of AUC, reporting on par or better AUC compared to widely used machine learning approaches.

Bibliography

[1] Bart Baesens, Christophe Mues, Manu De Backer, Jan Vanthienen, and Rudy Setiono. "Building Intelligent Credit Scoring Systems Using Decision Tables". In: *Enterprise Information Systems V.* Springer, 2004, pp. 131–137.

[2] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. "An Empirical Evaluation of the Comprehensibility of Decision Table, Tree and Rule Based Predictive Models". In: *Decision Support Systems* 51.1 (2011), pp. 141–154.

[3] Riccardo Bellazzi and Blaz Zupan. "Predictive Data Mining in Clinical Medicine: Current Issues and Guidelines". In: *International journal of medical informatics* 77.2 (2008), pp. 81–97.

[4] Michael J Pazzani, S Mani, and William R Shankle. "Acceptance of Rules Generated by Machine Learning Among Medical Experts". In: *Methods of information in medicine* 40.05 (2001), pp. 380–385.

[5] Alex A Freitas, Daniela C Wieser, and Rolf Apweiler. "On the Importance of Comprehensible Classification Models for Protein Function Prediction". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 7.1 (2008), pp. 172–182.

[6] Duane Szafron, Paul Lu, Russell Greiner, David S Wishart, Brett Poulin, Roman Eisner, Zhiyong Lu, John Anvik, Cam Macdonell, Alona Fyshe, et al. "Proteome Analyst: Custom Predictions With Explanations in a Web-Based Tool for High-Throughput proteome Annotations". In: *Nucleic acids research* 32.suppl_2 (2004), W365–W371.

[7] Elen Lima, Christophe Mues, and Bart Baesens. "Domain Knowledge Integration in Data Mining Using Decision Tables: Case Studies in Churn Prediction". In: *Journal of the Operational Research Society* 60.8 (2009), pp. 1096–1106.

[8] Wouter Verbeke, David Martens, Christophe Mues, and Bart Baesens. "Building Comprehensible Customer Churn Prediction Models with Advanced Rule Induction Techniques". In: *Expert systems with applications* 38.3 (2011), pp. 2354–2364.

[9] Muhammad Aurangzeb Ahmad, Carly Eckert, and Ankur Teredesai. "Interpretable Machine Learning in Healthcare". In: *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics.* 2018, pp. 559–560.

[10] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. "Explainable Artificial Intelligence: A Survey". In: *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*. IEEE. 2018, pp. 0210–0215.

[11] Finale Doshi-Velez and Been Kim. "Towards a Rigorous Science of Interpretable Machine Learning". In: *arXiv preprint arXiv:1702.08608* (2017).

[12] Ole-Christoffer Granmo. "The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic". In: *arXiv preprint arXiv:1804.01508* (2018).

[13] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. "The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression". In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

[14] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications". In: *IEEE Access* 7 (2019), pp. 115134–115146.

[15] Christoph Molnar. *Interpretable Machine Learning*. Lulu. com, 2019.

[16] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. "The Convolutional Tsetlin Machine". In: *arXiv preprint:1905.09688* (2019).

[17] Adrian Wheeldon, Rishad Shafik, Tousif Rahman, Jie Lei, Alex Yakovlev, and Ole-Christoffer Granmo. "Learning Automata Based Energy-efficient AI Hardware Design for IoT". In: *Philosophical Transactions of the Royal Society A* (2020). URL: `https://eprints.ncl.ac.uk/268038`.

[18] Christian D. Blakely and Ole-Christoffer Granmo. "Closed-Form Expressions for Global and Local Interpretation of Tsetlin Machines with Applications to Explaining High-Dimensional Data". In: *arXiv preprint arXiv:2007.13885* (2020). URL: `https://arxiv.org/abs/2007.13885`.

[19] Jin Huang and Charles X Ling. "Using AUC and Accuracy in Evaluating Learning Algorithms". In: *IEEE Transactions on knowledge and Data Engineering* 17.3 (2005), pp. 299–310.

[20] Andrew P Bradley. "The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms". In: *Pattern recognition* 30.7 (1997), pp. 1145–1159.

[21] Charles X Ling and Chenghui Li. "Data Mining for Direct Marketing: Problems and Solutions". In: *Kdd*. Vol. 98. 1998, pp. 73–79.

[22] Michael Lvovitch Tsetlin. "On Behaviour of Finite Automata in Random Medium". In: *Avtomat. i Telemekh* 22.10 (1961), pp. 1345–1354.

[23] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Lei Jiao, and Morten Goodwin. "The regression Tsetlin Machine: A Tsetlin Machine for Continuous Output Problems". In: *EPIA Conference on Artificial Intelligence.* Springer. 2019, pp. 268–280.

[24] Ole-Christoffer Granmo. "The Multi-Layered Tsetlin Machine". In: *Preparation* (2020).

[25] Adrian Phoulady, Ole-Christoffer Granmo, Saeed Rahimi Gorji, and Hady Ahmady Phoulady. "The Weighted Tsetlin Machine: Compressed Representations with Clause Weighting". In: *Ninth International Workshop on Statistical Relational AI (StarAI 2020).* 2020.

[26] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "A Regression Tsetlin Machine with Integer Weighted Clauses for Compact Pattern Representation". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer. 2020.

[27] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "Extending the Tsetlin Machine with Integer-Weighted Clauses for Increased Interpretability". In: *IEEE Access* 9 (2021), pp. 8233–8248.

[28] James A Hanley and Barbara J McNeil. "The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve". In: *Radiology* 143.1 (1982), pp. 29–36.

[29] David J Hand and Robert J Till. "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems". In: *Machine learning* 45.2 (2001), pp. 171–186.

[30] James McDermott and Richard S Forsyth. "Diagnosing a Disorder in a Classification Benchmark". In: *Pattern Recognition Letters* 73 (2016), pp. 41–43.

[31] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. "A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer. 2019, pp. 564–578.

Paper H

Convolutional Regression Tsetlin Machine: An Interpretable Approach to Convolutional Regression

Chapter abstract:

The Convolutional Tsetlin Machine (CTM), a variant of the Tsetlin Machine (TM), represents patterns as straightforward AND-rules, to address the high computational complexity and the lack of interpretability of Convolutional Neural Networks (CNNs). The CTM has shown competitive performance on MNIST, Fashion-MNIST, and Kuzushiji-MNIST pattern classification benchmarks, both in terms of accuracy and memory footprint. However, the CTM has so far only been applied to binary output. This chapter proposes the Convolutional Regression Tsetlin Machine (C-RTM) that extends the CTM to support continuous output problems in image analysis. C-RTM identifies patterns in images using the convolution operation as in the CTM and then maps the identified patterns into a real-valued output in the Regression Tsetlin Machine (RTM). The C-RTM thus unifies the two approaches.

We evaluate the performance of C-RTM using 72 *different artificial datasets, with and without noise in the training data. Our empirical results show the competitive performance of C-RTM compared to two standard CNNs. The interpretability of the identified sub-patterns by C-RTM clauses is also analyzed and discussed. Additionally, the performance of C-RTM on two real-world datasets is compared against CNN. In addition to the competitive performance in terms of mean absolute error, C-RTM filters perform significantly fewer calculations (only* ***AND*** *operations) during the convolution. Further, the filters learn which image locations are important, and then move directly to these locations during inference. This approach helps C-RTM consume significantly less memory both during training and testing.*

H.1 Introduction

Recent progress on TMs: Tsetlin Machines (TMs) are a recent approach to pattern recognition that use simple AND-rules to represent patterns. Several teams of so-called Tsetlin Automata (TA) compose these AND-rules to maximize pattern recognition accuracy, learning from penalties and rewards obtained by playing a novel game. On several machine learning benchmarks including Iris Data Classification, Handwritten Digits Classification (MNIST), Predicting Optimum Moves in the Axis and Allies Board Game, and Classification of Noisy XOR Data with Non-Informative Features performance is competitive. The TM has further been expanded in other directions and evaluated on applications from several application domains. For instance, the TM has been used for text classification by using the conjunctive clauses to capture textual patterns [1]. The TM has also been extended by introducing real weights [2] and integer weights [3] for both reduction of the number of clauses without loss of accuracy and increased interpretability. The process of hyper-parameter search of the TM has been simplified by introducing multi-granular clauses in [4]. By indexing the clauses on the features that falsify them, up to an order of magnitude faster inference and learning has been reported [5]. Additionally, Regression Tsetlin Machine (RTM) compare favorably with Regression Trees, Random Forest Regression, and Support Vector Regression [6]. For applications with real-valud features, a binarization scheme based on thresholding was proposed in [7] and a scheme which learns the above thresholds using merely two Stochastic Searching on the Line Automata (SSLs) per feature has been proposed in [8].

TM Inference: In the TM, Tsetlin Automata (TAs) are used to represent literals – propositional input variables and their negations. The literals, in turn, form conjunctive clauses in propositional logic, as decided by the TAs. The final TM output is a disjunction of all the specified clauses. In this manner, the pattern composition and learning procedure of the TM is fully transparent and understandable, facilitating human interpretation. In addition, the TM has an inherent computational advantage. That is, the inputs and outputs of the TM can naturally be represented as bits, and recognition and learning is performed by manipulating those bits. The operation of the TM thus demands relatively small computational resources, and supports hardware-near and parallel computation, e.g. on GPUs.

Convolutional Neural Networks: Deep learning or deep neural networks refers to artificial neural networks with multiple layers. One of the standard deep learning algorithms to work with image data, computer vision, and natural language processing is the Convolutional Neural Networks (CNNs). A CNN is a combination of several layers, namely, convolutional layer, non-linearity layer, pooling layer and fully-connected layer [9]. These interacting layers lead to high pattern recognition accuracy; however, CNNs suffer from high computational complexity and the lack of interpretability. High computational cost and difficulties with explaining why CNNs perform well hinder further improvement [10].

Convolutional Tsetlin Machine: To address the above issues in CNNs, Granmo et al. [11] introduced the Convolutional Tsetlin Machine (CTM); a new variant of TM. If the TM is given the task of classifying an image, a clause in the TM considers the entire image at once. However, in the CTM, clauses act as convolution filters. Since a clause in the TM is a conjunction of selected features, the reasons for classification can be easily interpreted. The CTM provides competitive performance on MNIST, Fashion-MNIST, and Kuzushiji-MNIST, in comparison with CNNs, K-Nearest Neighbor, Support Vector Machines, Random Forests, Gradient Boosting, BinaryConnect, Logistic Circuits and ResNet.

Paper Contributions. Convolutional regression applications are ample in the machine learning field. A few of them are near infrared (NIR) calibration [12], Spectrum analysis [13], depth prediction in digital holography [14], and vehicle detection and counting in aerial images [15]. To work on these applications, the CTM needs to be modified as the CTM has been designed for classification, not for producing continuous output. In this paper, we introduce the Convolutional Regression Tsetlin Machine (C-RTM) to overcome the above limitation of the CTM. Hence, C-RTM is a novel ensemble approach that unifies the properties of both CTM and RTM. In brief, the image patterns recognized by CTM clauses are piled together to map continuous outputs as in RTM. The performance of the C-RTM is evaluated on 72 artificial datasets. The performance is also compared against two CNN setups, with additional results on real-life data.

Paper Organization. The remainder of the paper is organized as follows. In Section H.2, we detail the TM theory. In Section H.3 we present the main contribution of this paper, which is the C-RTM, and how we build it upon the CTM and RTM. We then investigate the behavior of the C-RTM using 72 different artificial datasets in Section H.4. We demonstrate empirically that the C-RTM is capable of learning patterns of a known input-output environment in a interpretable manner and then correctly produce the required regression output. Performance wise, C-RTM shows competitive accuracy compared to the predictions of CNNs. We conclude our work in Section H.5.

H.2 The Tsetlin Machine (TM)

The TM is the base for both CTM and RTM. The C-RTM combines unique properties from both CTM and RTM. Hence, before we introduce C-RTM in more details, we first discuss the basics of TM in this section.

H.2.1 The Tsetlin Machine Architecture

As illustrated in Figure H.1, conceptually, a TM consists of five layers. Here in the following sub-sections, the role of each of those layers is explained in more details.

H.2.1.1 Layer 1 – Input Layer

The TM receives features in binary form. When the feature vector $\mathbf{X}$ consists of o propositional variables $\mathbf{X} \in \{0,1\}^o$, the augmented feature vector, $\mathbf{X}'$ can be written as $\mathbf{X}' = [x_1, x_2, x_3, \ldots, x_o, \neg x_1, \neg x_2, \neg x_3, \ldots, \neg x_o]$, where $\mathbf{X}'$ contains both original features and their negations. Allowing a TM to take both original and negated features

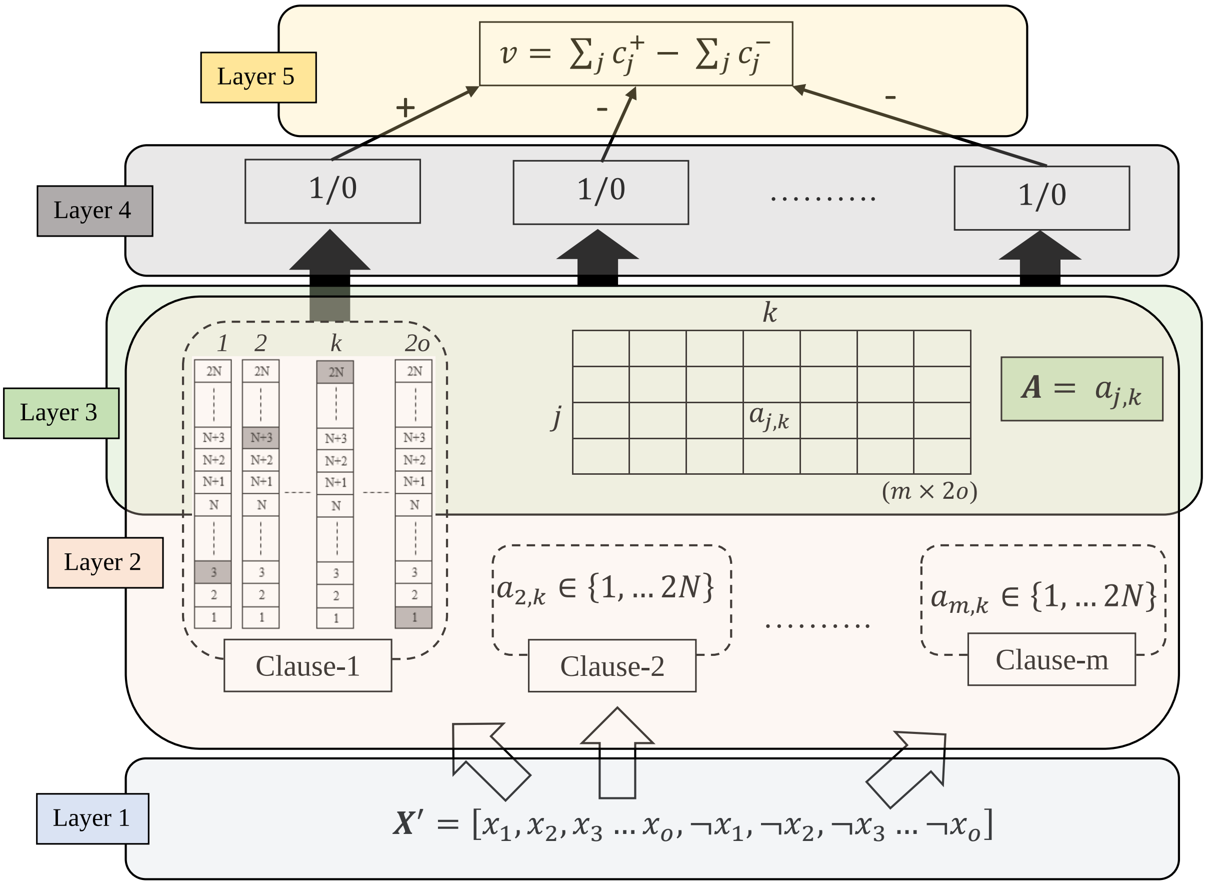


Figure H.1: The TM structure.

increases the expression power of patterns in data. The collective feature variables in $\mathbf{X}'$ are called literals.

H.2.1.2 Layer 2 – Clause Layer

The clauses in the TM recognizes the patterns in data. A TM comprises m conjunctive clauses. Each clause processes literals from the input layer. Different patterns of different classes make diverse clauses, by varying the literals included in a clause to represent a pattern. The set I_j^I contains the indexes of the *original* variables that are included in clause j. Similarly, the set $\bar{I}_j^I$ consists of the indexes of the included *negated* variables. These sets are thus subsets of the complete set of indexes $I_j^I, \bar{I}_j^I \subseteq \{1, \ldots, o\}$. Once the included literals in clauses are know, the clause j can be mathematically expressed as,

$$c_j = 1 \wedge \left(\bigwedge_{k \in I_j^I} x_k \right) \wedge \left(\bigwedge_{k \in \bar{I}_j^I} \neg x_k \right). \tag{H.1}$$

H.2.1.3 Layer 3 – Memory Layer

The indexes in set I_j^I and $\bar{I}_j^I$ are updated depending on the decision of Tsetlin Automata (TAs) attached to literals in clause j. Since there are $2 \times o$ literals in a clause, the same number of TAs is needed in a clause to decide the composition of the clause. Each

TA maintains a memory state $a_{j,k} \in \{1, \ldots, 2N\}$, with j referring to the clause, $c_j, j \in \{1, \ldots, m\}$ and k to the literal, $x'_k, k \in \{1, \ldots, 2o\}$. States 1 to N map to the *exclude* action: exclude the k^{th} literal from the j^{th} clause. Conversely, states $N+1$ to $2N$ map to the *include* action: include the k^{th} literal in the j^{th} clause.

Similar to storing weight values of a Neural Network in Deep Learning theories, a TM stores all the TA states in the memory layer, which can be organized as a matrix $\mathbf{A}$: $\mathbf{A} = (a_{j,k}) \in \{1, \ldots, 2N\}^{m \times 2o}$. Accordingly, $I_j^{I_{X'}}$, which now consists the indexes of the included literals from the augmented feature vector, $\mathbf{X}'$ can be written as: $I_j^{I_{X'}} = \{k | N < a_{j,k} < 2N, 1 \leq k \leq 2o\}$.

H.2.1.4 Layer 4 – Voting Layer

In the voting layer, firstly, the clause output is evaluated. When $I^1_{X'}$ contain the indexes of the 1-valued literals from $\mathbf{X}'$, since the clauses are conjunctive clauses, the clause output evaluates to 1 if $I_j^{I_{X'}}$ is a subset of $I^1_{X'}$. The value c_j of clause j can then be succinctly defined as:

$$c_j = \begin{cases} 1 & \text{if} \quad I_j^{I_{X'}} \subseteq I^1_{X'}, \\ 0 & \text{otherwise.} \end{cases} \tag{H.2}$$

H.2.1.5 Layer 5 – Output Layer

The output layer counts the sub-patterns identified by clauses of different classes. Polarized clause outputs from the voting layer are used to compute the vote difference between the two classes: $v = \sum_j c_j^+ - \sum_j c_j^-$. Then using a step function, the output of the TM is finally decided as:

$$y = \begin{cases} 1 & \text{if} \quad v \; \geq \; 0 \\ 0 & \text{if} \quad v \; < \; 0\,. \end{cases} \tag{H.3}$$

H.2.2 Learning Procedure

Learning in TM is carefully guiding the TAs in clauses to make the right decision: *include* or *exclude* its corresponding literal in the clause. The decision on literal x'_k in the clause j depends on the TA state $a_{j,k}$. Hence, during the training process, states of the TAs in clauses are updated so that eventually the TA teams in clauses make the most appropriate TM output. During the learning phase, the TM processes one training sample $(\mathbf{X}, y)$ at a time, leveraging the on-line learning properties. The states of TAs in clauses are updated using two types of reinforcement: Type I and Type II. As described in the following, Type I feedback produces frequent patterns, while Type II feedback increases the discrimination power of the patterns.

However, to diversify the clauses over different sub-patterns, for any input $\mathbf{X}$, the probability of reinforcing a clause gradually drops to zero as the voting sum v approaches a user-set target T. That is clauses which receives Type I feedback is sorted stochastically with probability $\frac{T - \max(-T, \min(T, v))}{2T}$. Clauses which receives Type II feedback is selected stochastically with probability $\frac{T+\max(-T,\min(T,v))}{2T}$. Here, higher T increases the robustness of learning by allocating more clauses to learn each sub-pattern.

Type I feedback: Type I feedback is given to clauses with positive polarity when $y = 1$. Clauses with negative polarity receive Type I feedback when $y = 0$. Furthermore, 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.

If a clause is eligible to receive feedback, the individual TAs of the clause are singled out stochastically using a user-set parameter s (s $\geq$ 1). The k^{th} TA of the j^{th} clause receives Type Ia feedback with probability $\frac{s-1}{s}$. If this happens, the corresponding TA state $a_{j,k}$ is increased by 1. This TA receives Type Ib feedback with probability $\frac{1}{s}$ to reduce the TA state $a_{j,k}$ by 1. As noticed, Type Ia feedback is stronger compared to Type Ib feedback where it makes the clause remember and refine the pattern it recognizes in $\mathbf{X}'$. Type Ib, on the other hand, fine-tunes the clause output if it is 1 by further pushing states of TAs of literal value 0 towards *exclude* action or makes the clause forget the pattern if the clause output is 0 by reinforcing *exclude* action, regardless of the literal value.

Type II feedback is given to clauses with positive polarity when $y{=}0$ and to clauses with negative polarity when $y{=}1$. Type II feedback does not update the states of TAs if the clause output is 0. However, when the clause output is 1, Type II feedback attempts to make it output 0 by including 0 literals in the clause. Hence, if x'_k is 0 in a such a clause, $a_{j,k}$ is increased by 1. Thus, this feedback introduces literals for discriminating between $y = 0$ and $y = 1$.

H.3 The Convolutional-Regression Tsetlin Machine (C-RTM)

The C-RTM is a novel ensemble approach based on the TM that unifies the unique properties of both CTM and RTM. This section introduces the C-RTM and explains how the patterns recognized during the convolutional operations of the CTM are merged into regression outputs as in the RTM.

H.3.1 The Convolutional Operation

Consider an image of size $X \times Y \times Z$ as in Figure H.2, in which, pixels are represented in binary form[1]. If the complete image is to be sent into the classical TM, the augmented feature vector, $\mathbf{X}'$ will contain $X \times Y \times Z \times 2$ propositional variables, $\mathbf{X}' = (x'_k) \in \{0,1\}^{X \times Y \times Z \times 2}$. Hence, clauses will consist of $X \times Y \times Z \times 2$ number of TAs each. These TAs will then decide which of the above literals of the image are important for a specific class and should be included in the clauses. However, instead of sending the entire image

[1] pixel values can be binarized using the thresholding[7], one-hot encoding, or any other binary encoding approach.

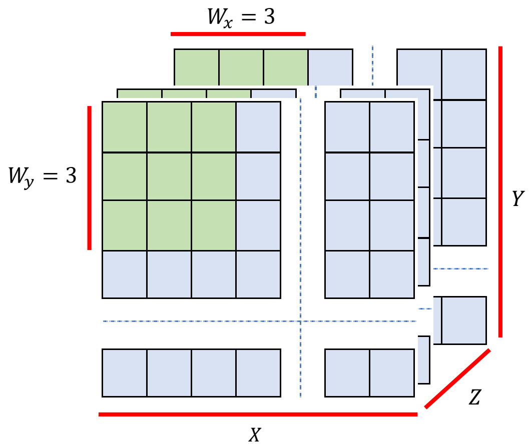


Figure H.2: A filter of size $W_x \times W_y \times Z$ on an image of size $X \times Y \times Z$.

into the TM, the CTM [11], which was inspired by the convolution in deep learning, considers a patch of the image at a time as further explained below.

In the CTM, filters of size $W_x \times W_y \times Z$ are used to extract the patterns of distinct locations in the image. In Figure H.2, a filter of size 3 has been located at the top-left corner of the image. A clause in the TM then receives the features of the image which are covered by the filter. The resulting feature vector for a clause then consists of $W_x \times W_y \times Z \times 2$ literals, $\mathbf{X}' = (x'_k) \in \{0,1\}^{W_x \times W_y \times Z \times 2}$.

This filter moves around the image. It moves d steps systematically towards the right from its current location and d steps to the bottom from its current location. As the filter moves around, the same clause receives multiple feature vectors from different locations of the image. Hence, to make the clause location-aware, the binary encoded location of the filter is also augmented to the feature vector. The number of feature vectors (equal to the number of image patches) received by a clause from an image is equal to B; $B = B_X \times B_Y$. Here, $B_X = [\frac{X - W_x}{d}] + 1$ and $B_Y = [\frac{Y - W_y}{d}] + 1$. Now with the binary encoded location (one propositional variable per position along each dimension), the feature vector can be rewritten as $\mathbf{X}' = (x'_k) \in \{0,1\}^{(W_x \times W_y \times Z + B_X + B_y)2}$.

The CTM uses m number of such filters to cover the sub-patterns in different locations of the images from different classes. As a result, the CTM needs m number of clauses to represent all these filters. However, each clause outputs B values per image, as the filter moves to B unique locations in the image. As opposed to the outputs of clauses in the classic TM, where it is the direct conjugation of the included literals, here in the CTM, clause output is 1 if the clause recognizes a pattern at least at any of the B locations in an image. Hence, the clause output of clause j, c_j can be written as:

$$c_j = \vee_{b=1}^{B} c_j^b. \tag{H.4}$$

where b is the index of the location, $b \in \{1, \ldots, B\}$ and j is clause index, $j \in \{1, \ldots, m\}$.

Similar to the TM, at the *Voting Layer*, the clauses in CTM are divided equally into the classes so that they recognize the patterns of their corresponding classes. In the *Output Layer*, still using the majority vote concept, the output is decided using a step function as in (2.4).

Yet, learning in the CTM utilizes the same Type I and Type II feedbacks. However, since the clauses in the CTM receive multiple inputs per image (B input patches), each clause should be trained to learn only one sub-pattern from the list of sub-patterns it recognizes at different locations of the image. The CTM randomly selects one of these patches among the patches that made the clause evaluate to 1. The randomness of the uniform distribution statistically spread the clauses for different sub-patterns in the target image.

H.3.2 Mapping Clause Outputs into a Regression Value

When the output is continuous, the output layer of the CTM in [11] is unable to map the input features into a correct output value. Hence, we now modify the structure and the learning procedure of the CTM, which then can map image inputs into a continuous output. The new ensemble approach is called as Convolutional-Regression Tsetlin Machine (C-RTM).

Now, since the goal of the C-RTM is not to map the output into a class, clause polarities are unnecessary, where they were initially used to recognize the classes. Hence, we now remove the polarity of clauses, since we intend to use the clauses as additive building blocks that can be used to calculate continuous output. As a result, the value v in the output layer takes a value between 0 and T instead of a value between $-T$ and T.

The goal of the modified output layer is to normalize the resulting v into a regression output. This is simply done by using the maximum output value $\hat{y}_{\max}$ among the n training samples, $\boldsymbol{Y} = [\hat{y}_1, \hat{y}_2, \hat{y}_3, \ldots, \hat{y}_n]$, and the user-set target value T, which now control the precision of the regression output. For instance, for the o^{th} training image, $(\hat{X}_o)$, the C-RTM output, y_o, is calculated as:

$$y_o = \frac{\sum_{j=1}^{m} c_j(\hat{X}_o) \quad \times \hat{y}_{\max}}{T} \ . \tag{H.5}$$

However, the above computed C-RTM output, y_o can be higher or lower than the target output, $\hat{y}_o$. Similarly to other machine learning methods, we need a mechanism for minimizing the error between the predicted output, y_o, and target output, $\hat{y}_o$. In the C-RTM, this is quite simply achieved by providing Type I and Type II feedbacks according to the following criteria:

$$Feedback = \begin{cases} \text{Type I,} & \text{if} \quad y_o < \hat{y}_o \ , \\ \\ \text{Type II,} & \text{if} \quad y_o > \hat{y}_o \ . \end{cases} \tag{H.6}$$

The idea is to apply Type I feedback, which forces clauses to output 1, when the predicted output is less than the target output ($y_o < \hat{y}_o$) and Type II feedback, which forces clauses to output 0, when the predicted output is higher than the target output ($y_o > \hat{y}_o$).

However, to stabilize the output value on to a target output, the number of clauses which receive the above feedback should be proportional to the magnitude of the error between the predicted and target output. That is, in the C-RTM, feedback to clauses is determined stochastically with a probability of $\frac{K\times|y_o-\hat{y}_o|}{\hat{y}_{\max}}$. As noticed, the magnitude of the function is adjusted with the constant K. The resulting activation function reduces the oscillation of the predicted value during the training process, stabilizing it around the target value.

H.4 Empirical Results

In this section, the performance of the proposed ensemble approach is evaluated. First, we utilize thirty six artificial datasets with known input-output mapping. With these artificial datasets, we explain how the C-RTM learns distinct patterns in correct numbers to produce the right regression output. Later we make the task difficult for the C-RTM by adding noise to the above datasets. The performance of C-RTM on noisy data is compared against CNN. Towards the end of the section, the performance of the C-RTM is evaluated and compared against the CNNs on two real-world applications: y-coordinate prediction and MNIST-regression.

H.4.1 Artificial Data

We study the behaviour of the C-RTM using three different sizes of artificially generated images. These images have been constructed to facilitate empirical analysis of the optimality of C-RTM learning, with the underlying input-output mapping being known. The first set of images are $3 \times 3 \times 1$ in size. The second and third set of images are $4 \times 4 \times 1$ and $5 \times 5 \times 1$ in size, respectively. The pixels in each image are random binary values with equal probability of being 0 or 1, leading to a more or less uniform distribution of bit values.

Each image in a selected dataset is then covered with a *weighted-mask*. The mask decide which pixels in the image are important and how important are they. The mask together with the original image decides the output label $\hat{y}$. For instance, Figure H.3 depicts a case of a $4 \times 4 \times 1$ image. Consider that (x, y) coordinates of the top-left corner is $(0, 0)$ and they increase towards right and bottom, respectively. We can see then the mask has decided that $(0, 0)$, $(2, 0)$, $(1, 2)$, and $(3, 3)$ are the important pixels in the image. The weights of these pixels are 2^3, 2^2, 2^1, and 2^0, respectively. By taking the multiplication of corresponding pixels of the image and the mask, a resulting image is

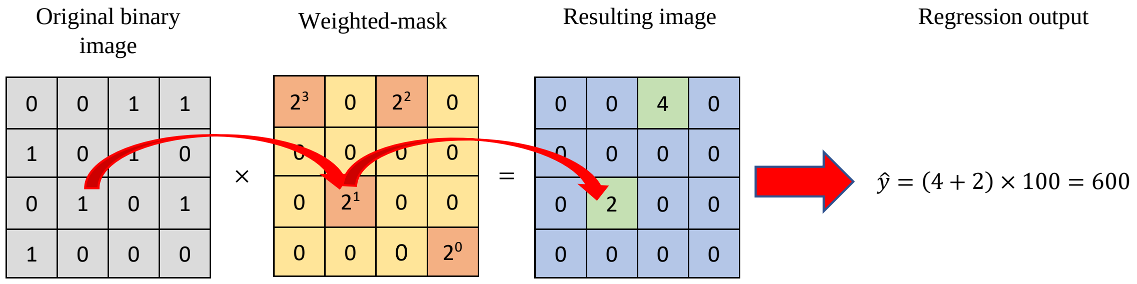


Figure H.3: Generating artificial data for C-RTM.

Table H.1: Summary of the artificial images generated to measure the performance of RTM

	Noise-free			Noisy		
Image Size	$3 \times 3 \times 1$	$4 \times 4 \times 1$	$5 \times 5 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$	$5 \times 5 \times 1$
No. of Important Pixels	2 3 4	2 3 4	2 3 4	2 3 4	2 3 4	2 3 4
No. of Different Masks	4 4 4	4 4 4	4 4 4	4 4 4	4 4 4	4 4 4
Total datasets	12	12	12	12	12	12
	36			36		

generated. The sum of the pixel values of the resulting image is then multiply by 100 to get the regression output of the image.

The number of datasets can be increased by changing the important pixel locations of the weighted-mask and by varying the number of important pixels in the weighted mask. The number of datasets can be further increased by introducing noise into training data. In our study, for each set of images, 12 different masks are used. In them, 4 masks highlight 2 important pixels at 4 different random locations, 4 masks highlight 3 important pixels at 4 different random locations, and 4 masks highlight 4 important pixels at 4 different random locations. Collectively this creates 36 different datasets. The number of datasets are then doubled by adding noise into the training data in another similar 36 datasets.

For the purpose of demonstrating the performance of TM in the next section, we summarize the above datasets in Table H.1. The performance of the TM is evaluated on these datasets and compared against two CNN setups as discussed in the next section.

H.4.2 Results and Discussion on Artificial Data

We use Mean Absolute Error (MAE) to measure the performance. The C-RTM in these experiments uses the optimum number of clauses discussed in [16] to work with noise-free data, i.e., *three* clauses when the number of important pixels is 2, *seven* clauses when the number of important pixels is 3, and *fifteen* clauses when the number of important pixels is 4. However, to handle the noise in data, C-RTM uses large number of clauses as further explained. The values for parameters T and s were found using a grid search in [16] for the RTM. These values are also employed here by the C-RTM.

The C-RTM obtains zero MAE for both training and testing on noise-free datasets as discussed in Table H.1. This can be explained using the case given in Figure H.3 as an example. The weighted-mask can create 16 different resulting images according to the combination of different binary values in the important pixels. Accordingly, 16 different regression outputs are generated. For instance, if all $(0, 0)$, $(2, 0)$, $(1, 2)$, and $(3, 3)$ pixels in the original image are 0s, the resulting $\hat{y}$ will be zero. If only $(3, 3)$ from the above important pixels is 1, the resulting $\hat{y}$ will be 100. In the C-RTM, these different output values are computed by taking the sum of the clause outputs as in Eqn. H.5. Hence, to correctly generate the output by making the correct number of clauses output 1, the

Table H.2: Four patterns recognized by filters (clauses) in the C-RTM to correctly predict the regression outputs associated with weighted-mask in Figure H.3.

Pattern 1	Pattern 2	Pattern 3	Pattern 4
1 ❋ ❋ ❋	❋ 1 ❋ ❋	❋ ❋ 1 ❋	❋ ❋ ❋ 1

C-RTM should recognize 4 different patterns in data. Assuming the filter size is 2-by-2, these patterns recognized by the clauses are summarized in Table H.2.

Here in patterns, '1' means the original literal related to the considered pixel has been included in the clause. The $*$ on the other hand means the pixel value can take an arbitrary value, either 0 or 1 (both original and negated literals related to the considered pixel have been excluded from the clause).

Now consider the new input image in Table H.3. The clause which recognizes the pattern 1 outputs 1 since image pixel at (0, 0) is 1 (the values of other pixels are not important since TAs representing their both original and negated pixels have decided to exclude them from the clause: denoted in $*$). Similarly, clauses which recognize the pattern 2 and 4 also output 1. The clause that represents the pattern 3 outputs 0 since the image pixel at (1, 2) is 0.

According to the calculations in Figure H.3, the corresponding regression output of the above image is 1300. As discussed early, the C-RTM needs 15 clauses to perform the convolutional-regression task in hand. During the C-RTM learning phase, these clauses are allocated to different pattern in right numbers. If *eight* clauses recognize pattern 1, *four* clauses recognize pattern 2, *two* clauses recognize pattern 3, and *one* clause recognizes pattern 4, the correct output 1300 for the above image can be calculated. Similarly, according to the above allocation of clauses to different patterns, any regression output for the considered convolutional-regression problem can be generated.

One might wonder why the clause which recognizes the pattern 1 does not output 1 at picture location $(2, 0)$, or why the clause which recognizes the pattern 2 does not output 1 at picture location $(3, 3)$. This is due to that these clauses also learn the correct location of the pattern.

In our case, we attach the location using the thresholding approach proposed in [7].

Table H.3: Placing filters on different image locations where they recognized those patterns.

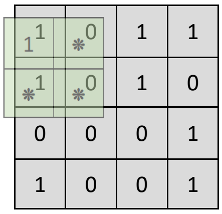
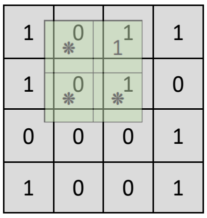
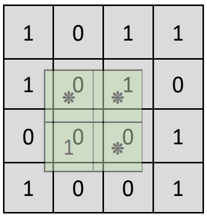
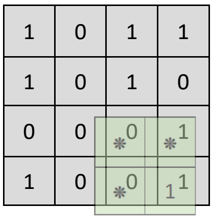
Pattern 1 at (0,0)	Pattern 2 at (1,0)	Pattern 3 at (1,1)	Pattern 4 at (2,2)
			

Table H.4: Binary representation of filter locations in the image

Location	Binary representation					
	Thresholds for B_X			Thresholds for B_Y		
	≤ 0	≤ 1	≤ 2	≤ 0	≤ 1	≤ 2
$(0,0)$	1	1	1	1	1	1
$(1,0)$	0	1	1	1	1	1
$(2,0)$	0	0	1	1	1	1
$(0,1)$	1	1	1	0	1	1
$(1,1)$	0	1	1	0	1	1
$(2,1)$	0	0	1	0	1	1
$(0,2)$	1	1	1	0	0	1
$(1,2)$	0	1	1	0	0	1
$(2,2)$	0	0	1	0	0	1

Since the size of the filter is 2-by-2 and the step size of the convolution, d is 1, B_X and B_Y can be calculated as, $B_X = [\frac{X-W}{d}] + 1 = 3$ and $B_Y = [\frac{Y-W}{d}] + 1 = 3$. Therefore, each filter moves to 9 (B) unique locations in the image. Table H.4 summarizes the binary feature representation of all these 9 locations of filters in the image. Now the clause which learns the pattern 1 also learns the location of the pattern, where it says x should be ≤ 0 (by including original ≤ 0 threshold of B_X in the clause) and y should be ≤ 0 (by including original ≤ 0 threshold of B_Y in the clause). Hence, for instance, the clause which recognized the pattern 1 does not output 1 at location $(2,0)$ where ≤ 0 of $(2,0)$ is 0.

So far in this section, we considered the case of having 4 important pixels in a 4-by-4 image as an example. However, regardless of the size of the image, 2-by-2 filters find similar patterns as in Table H.2 when there are 4 important pixels. Table H.5 outlines the complete set of patterns that clauses recognize depending on the number of important pixels in the image and how many of them needed to form the required output. The pattern 1 in Table H.2 has been written as $(1 * * *)$ in Table H.5. In this case, we use a 1-by-2 filter when the number of important pixels is 2, a 1-by-3 filter when the number of important pixels is 3, and a 2-by-2 filter when the number of important pixels is 4.

We observe that the C-RTM behaves similar to RTM [16] when T (= to the number of clauses) is not a multiplier of the optimum required clauses. In this situation, the C-RTM cannot align its output y_o to the target output $\hat{y}_o$ during the training phase. For instance, by assigning *sixteen* clauses when the number of important pixels is 4, the training will end up with e.g. allocating *five* clauses to represent the pattern $(* \ 1 * *)$ or *two* clauses to represent the pattern $(* * * \ 1)$. As a result, one or more output values cannot be computed correctly. For example, if there are *five* clauses for the pattern $(* \ 1 * *)$ after training, input image in Table H.3 activates 14 clauses, producing an incorrect output that is 1400.

As a strategy for problems where the number of important pixels is unknown, and for real-world applications where noise plays a significant role, the C-RTM can be initialized with a large number of clauses (= T). Then, since the output, y_o, is a fraction of the threshold, T, the error decreases. With that in mind, C-RTM is operated with 1000 clauses on all 36 noisy datasets. To compare the performance of the C-RTM, two CNNs are used: CNN-50 - containing 50 filters and CNN-1000 - containing 1000 filters. The training and testing MAEs for all the cases are summarized in Table H.6 and H.7, respectively.

All the considered models show competitive performance on noisy data as outlined in Table H.6 and H.7. In general, the average training MAEs are higher than the average testing MAEs as noise was added on training data. However, this shows that all models are able to generalize the datasets despite the noise in training data. Figure H.4 illustrates

Table H.5: Computing output for different images when having different number of important pixels by activating different clauses.

No. of important pixels	Output	Required number of clauses to represent different patterns††
2	0	None
	100	$1\times(*\ 1)$
	200	$2\times(1\ *)$
	300	$2\times(1\ *) + 1\times(*\ 1)$
3	0	None
	100	$1\times(*\ *\ 1)$
	200	$2\times(*\ 1\ *)$
	300	$2\times(*\ 1\ *) + 1\times(*\ *\ 1)$
	400	$4\times(1\ *\ *)$
	500	$4\times(1\ *\ *) + 1\times(*\ *\ 1)$
	600	$4\times(1\ *\ *) + 2\times(*\ 1\ *)$
	700	$4\times(1\ *\ *) + 2\times(*\ 1\ *) + 1\times(*\ *\ 1)$
4	0	None
	100	$1\times(*\ *\ *\ 1)$
	200	$2\times(*\ *\ 1\ *)$
	300	$2\times(*\ *\ 1\ *) + 1\times(*\ *\ *\ 1)$
	400	$4\times(*\ 1\ *\ *)$
	500	$4\times(*\ 1\ *\ *) + 1\times(*\ *\ *\ 1)$
	600	$4\times(*\ 1\ *\ *) + 2\times(*\ *\ 1\ *)$
	700	$4\times(*\ 1\ *\ *) + 2\times(*\ *\ 1\ *) + 1\times(*\ *\ *\ 1)$
	800	$8\times(1\ *\ *\ *)$
	900	$8\times(1\ *\ *\ *) + 1\times(*\ *\ *\ 1)$
	1000	$8\times(1\ *\ *\ *) + 2\times(*\ *\ 1\ *)$
	1100	$8\times(1\ *\ *\ *) + 2\times(*\ *\ 1\ *) + 1\times(*\ *\ *\ 1)$
	1200	$8\times(1\ *\ *\ *) + 4\times(*\ 1\ *\ *)$
	1300	$8\times(1\ *\ *\ *) + 4\times(*\ 1\ *\ *) + 1\times(*\ *\ *\ 1)$
	1400	$8\times(1\ *\ *\ *) + 4\times(*\ 1\ *\ *) + 2\times(*\ *\ 1\ *)$
	1500	$8\times(1\ *\ *\ *) + 4\times(*\ 1\ *\ *) + 2\times(*\ *\ 1\ *) + 1\times(*\ *\ *\ 1)$

†† for example, "*two* clauses to represent the pattern $(1\ *)$" is written as "$2\times(1\ *)$"

Table H.6: Average Training MAEs on noisy data by CNNs and C-RTM.

Image size		3 × 3 × 1			4 × 4 × 1			5 × 5 × 1		
No. of important pixels		2	3	4	2	3	4	2	3	4
No. of different masks		4	4	4	4	4	4	4	4	4
Average MAE	CNN-50	4.94	4.87	5.03	5.01	4.78	5.16	5.31	4.91	5.08
	CNN-1000	4.81	4.90	4.93	4.79	4.86	5.03	4.76	4.96	4.81
	C-RTM	5.34	5.01	4.83	4.97	4.96	5.18	5.15	5.11	4.91

Table H.7: Average Testing MAEs on noisy data by CNNs and C-RTM.

Image size		3 × 3 × 1			4 × 4 × 1			5 × 5 × 1		
No. of important pixels		2	3	4	2	3	4	2	3	4
No. of different masks		4	4	4	4	4	4	4	4	4
Average MAE	CNN-50	0.97	1.34	1.28	1.04	1.16	1.20	0.98	0.98	1.17
	CNN-1000	1.63	1.71	2.21	1.94	2.07	1.85	2.21	1.76	2.16
	C-RTM	1.17	1.14	1.21	1.09	1.32	1.17	0.98	1.22	1.03

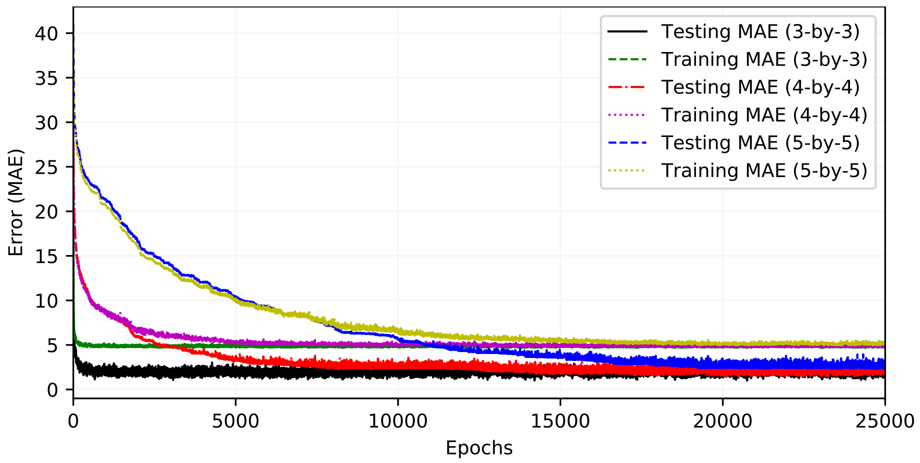


Figure H.4: C-RTM training and testing accuracy variation over epochs when different sizes of images are used.

C-RTM training and testing MAE variation over epochs for a case of having 3 important pixels in three different sizes of images. As shown, training MAEs converge to a value close to 5 while testing MAEs converge to a value above 0. The only difference is, when the size of the image is bigger, it requires more training epochs to find the correct clause configuration.

Average training MAEs of CNN-1000 on all considered 9 cases are lower than those of CNN-50. This might be due to the allocation of more filters in CNN-1000 than in CNN-50 to capture the patterns in data. Training MAEs of C-RTM do not show any clear difference to the MAEs of CNN-50 and CNN-1000. In some cases, MAE of C-RTM is higher than that of both CNNs. However, in some other cases, MAE of C-RTM is lower than that of both CNNs.

Despite better training MAEs of CNN-1000 compared to CNN-50, the testing MAEs of CNN-50 are lower than the MAEs of CNN-1000. A possible reason could be the overfitting of data by CNN-1000 during the training. ALL testing MAEs of C-RTM are better than those of CNN-1000. In four out of nine cases, CNN-50 obtains better MAEs. At same number of occasions, C-RTM beats CNN-50. In the case of 2 important pixels in a $5 \times 5 \times 1$ image, both CNN-50 and C-RTM acquire the MAE of 0.98.

In spite of more or less similar error, the C-RTM has an immense advantage of interpretability of identified patterns in data. These patterns for example can be seen in Table H.2. Interpretability is somehow not possible or extremely difficult to achieve with CNN, specially when large number of filters are used. Additionally, the C-RTM can also detail which of the learned patterns are important and add a higher value to the regression output by counting the number of clauses which learned the similar pattern. For comprehensiveness, we further compare C-RTM and CNN in more detail in Table H.8.

Since both CNN-1000 and C-RTM use the same number of filters, CNN-1000 is selected for the comparison against C-RTM. As we already discussed, C-RTM shows better performance in terms of the testing error on all different sizes of images compared to CNN-1000. The learned CNN filters to achieve the above performance contain float numbers while clauses in C-RTM contain only binary values. This can be seen in the example filters outlined in Table H.8. Furthermore, the filters in CNN move to all the possible locations in the image while clauses in C-RTM move only to some specific locations as these clauses also learn the general locations of the learned patterns. More specifically, 2-by-2 filters of CNN in a $5 \times 5 \times 1$ image move to 16 different locations. However, shown example C-RTM clause in the table only activates at top-left corner of the image, (0, 0). At each location, the CNN filter performs 4 multiplications and 3 summations. In total at 16 different locations, the CNN filter performs 64 multiplications and 48 summations. The C-RTM clause, only other hand, performs only one **AND** operation.

With the CNN, further weight analysis are needed before making the regression output. However with the C-RTM, the clause output is just sent to the voting layer and proceed with the output calculation. For all these internal operations in total, the CNN takes 4.34 seconds per training epoch. Foreseeably, the C-RTM takes only 0.53 seconds per epoch for its internal operations during training.

In addition to all the above advantages with C-RTM, the C-RTM consumes significantly lesser memory both during training and testing. The CNN requires 1009.6 more memory to work with the artificial data during training. The CNN also requires 2.2 MiB of memory during testing while C-RTM uses almost zero memory during testing.

H.4.3 Real-World Data: Experiments and Performance Comparison

We analyze the C-RTM with two real-world datasets: the y-coordinate prediction dataset and the MNIST-regression dataset. The y-coordinate prediction dataset contains the images of concrete beams. The goal here is to predict the y-coordinate of a specific feature on the beam surface[2]. The MNIST-regression dataset basically contains the MNIST image

[2] Dataset and more details about the dataset can be found at: https://forums.fast.ai/t/dataset-for-regression-cnn/30188

Table H.8: Detailed comparison of performance between CNN-1000 and C-RTM on artificial data.

<table>
<tr><th></th><th colspan="3">CNN-1000</th><th colspan="3">C-RTM</th></tr>
<tr><td>Image Size</td><td>$3 \times 3 \times 1$</td><td>$4 \times 4 \times 1$</td><td>$5 \times 5 \times 1$</td><td>$3 \times 3 \times 1$</td><td>$4 \times 4 \times 1$</td><td>$5 \times 5 \times 1$</td></tr>
<tr><td>Average MAE</td><td>1.85</td><td>1.95</td><td>2.04</td><td>1.17</td><td>1.19</td><td>1.08</td></tr>
<tr><td>An Example Filter
(For $5 \times 5 \times 1$ image)</td><td colspan="3"><table><tr><td>5.89</td><td>0.01</td></tr><tr><td>0.87</td><td>-0.59</td></tr></table></td><td colspan="3"><table><tr><td>1</td><td>*</td></tr><tr><td>*</td><td>*</td></tr></table></td></tr>
<tr><td>Filter Values</td><td colspan="3">Floats (Positive/Negative)</td><td colspan="3">Binary</td></tr>
<tr><td>Filter Location</td><td colspan="3">No location</td><td colspan="3">x $\leq$ 0, y $\leq$ 0</td></tr>
<tr><td>Moves</td><td colspan="3">Moves to 16 locations</td><td colspan="3">Just stays at (0,0) location</td></tr>
<tr><td>No. of Operations Performed
(per filter)</td><td colspan="3">64 Multiplications
48 Summations</td><td colspan="3">1 <b>AND</b> operation
(Maximum 3 <b>AND</b> operations)</td></tr>
<tr><td>After Convolutional Calculations</td><td colspan="3">If no other layers, the resulting image is sent to the fully connected layer and further weight analysis are performed</td><td colspan="3">Send the result (1 or 0) to the voting layer and proceed with output calculation</td></tr>
<tr><td>Training Time Per Epoch</td><td colspan="3">4.34 sec.</td><td colspan="3">0.53 sec.</td></tr>
<tr><td>Training Memory</td><td colspan="3">908.6 MiB</td><td colspan="3">0.9 MiB</td></tr>
<tr><td>Testing Memory</td><td colspan="3">2.2 MiB</td><td colspan="3">0.00 MiB</td></tr>
</table>

Table H.9: Detailed comparison of performance between CNN and C-RTM on y-coordinate dataset.

<table>
<tr><th></th><th>CNN-1000</th><th>C-RTM</th></tr>
<tr><td>Image Size</td><td>$200 \times 199 \times 1$</td><td>$200 \times 199 \times 1$</td></tr>
<tr><td>Average MAE</td><td>16.51</td><td>14.74</td></tr>
<tr><td>An Example Filter
(10×10)</td><td>
<table>
<tr><td>0.006</td><td>-0.011</td><td>..</td><td>-0.007</td><td>-0.003</td></tr>
<tr><td>-0.001</td><td>-0.014</td><td>..</td><td>0.010</td><td>-0.014</td></tr>
<tr><td>-0.009</td><td>-0.010</td><td>..</td><td>-0.006</td><td>-0.014</td></tr>
<tr><td>-0.002</td><td>-0.007</td><td>..</td><td>-0.008</td><td>-0.008</td></tr>
<tr><td>-0.010</td><td>-0.002</td><td>..</td><td>-0.014</td><td>-0.002</td></tr>
<tr><td>0.003</td><td>-0.012</td><td>..</td><td>0.000</td><td>-0.002</td></tr>
<tr><td>-0.012</td><td>-0.005</td><td>..</td><td>0.007</td><td>0.002</td></tr>
<tr><td>0.002</td><td>-0.004</td><td>..</td><td>-0.001</td><td>0.010</td></tr>
<tr><td>0.002</td><td>-0.004</td><td>..</td><td>-0.001</td><td>0.010</td></tr>
<tr><td>0.002</td><td>-0.004</td><td>..</td><td>-0.001</td><td>0.010</td></tr>
</table>
</td><td>
<table>
<tr><td>*</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>0</td><td>*</td><td>0</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>1</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>1</td><td>*</td><td>*</td><td>*</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>*</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>1</td><td>*</td><td>1</td><td>1</td><td>*</td><td>*</td><td>*</td><td>1</td></tr>
<tr><td>*</td><td>*</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>1</td><td>*</td></tr>
<tr><td>1</td><td>*</td><td>*</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>Filter Values</td><td>Floats (Positive/Negative)</td><td>Binary</td></tr>
<tr><td>Filter Location</td><td>No location</td><td>$x \leq 8$, $11 < y \leq 14$</td></tr>
<tr><td>Moves</td><td>Moves to 381 locations</td><td>Moves to 36 location</td></tr>
<tr><td>No. of Operations Performed
(per filter)</td><td>38,100 Multiplications
37,719 Summations</td><td>684 <b>AND</b> operations
(Maximum 3,600 <b>AND</b> operations)</td></tr>
<tr><td>After Convolutional Calculations</td><td>If no other layers, the resulting image is sent to the fully connected layer and further weight analysis are performed</td><td>Send the result (1 or 0) to the voting layer and proceed with output calculation</td></tr>
<tr><td>Training Time Per Epoch</td><td>5.79 sec.</td><td>33.18 sec.</td></tr>
<tr><td>Training Memory</td><td>1,934.1 MiB</td><td>578.8 MiB</td></tr>
<tr><td>Testing Memory</td><td>55.5 MiB</td><td>0.00 MiB</td></tr>
</table>

Table H.10: Detailed comparison of performance between CNN and C-RTM on MNIST-regression dataset.

<table>
<tr><th></th><th>CNN-1000</th><th>C-RTM</th></tr>
<tr><td>Image Size</td><td>$28 \times 28 \times 1$</td><td>$28 \times 28 \times 1$</td></tr>
<tr><td>Average MAE</td><td>0.56</td><td>0.61</td></tr>
<tr><td>An Example Filter
(10×10)</td><td>
<table>
<tr><td>-0.004</td><td>-0.007</td><td>..</td><td>-0.005</td><td>-0.004</td></tr>
<tr><td>-0.006</td><td>-0.003</td><td>..</td><td>-0.004</td><td>-0.006</td></tr>
<tr><td>-0.008</td><td>-0.008</td><td>..</td><td>-0.008</td><td>-0.008</td></tr>
<tr><td>-0.007</td><td>-0.003</td><td>..</td><td>-0.005</td><td>-0.003</td></tr>
<tr><td>-0.005</td><td>-0.004</td><td>..</td><td>-0.006</td><td>-0.005</td></tr>
<tr><td>-0.005</td><td>-0.006</td><td>..</td><td>-0.004</td><td>-0.008</td></tr>
<tr><td>-0.008</td><td>-0.007</td><td>..</td><td>-0.005</td><td>-0.004</td></tr>
<tr><td>-0.034</td><td>-0.006</td><td>..</td><td>-0.005</td><td>-0.007</td></tr>
<tr><td>-0.004</td><td>-0.008</td><td>..</td><td>-0.006</td><td>-0.006</td></tr>
<tr><td>-0.006</td><td>-0.004</td><td>..</td><td>-0.003</td><td>-0.025</td></tr>
</table>
</td><td>
<table>
<tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>1</td><td>1</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>1</td><td>1</td><td>1</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>1</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>1</td><td>*</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td><td>1</td><td>*</td><td>*</td><td>*</td><td>1</td></tr>
<tr><td>*</td><td>*</td><td>*</td><td>1</td><td>1</td><td>*</td><td>*</td><td>*</td><td>*</td><td>*</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>*</td><td>*</td><td>*</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>Filter Values</td><td>Floats (Positive/Negative)</td><td>Binary</td></tr>
<tr><td>Filter Location</td><td>No location</td><td>x $\leq$ 1, y $\leq$ 0</td></tr>
<tr><td>Moves</td><td>Moves to 38 locations</td><td>Moves to 2 location</td></tr>
<tr><td>No. of Operations Performed
(per filter)</td><td>3,800 Multiplications
3,762 Summations</td><td>40 <b>AND</b> operations
(Maximum 200 <b>AND</b> operations)</td></tr>
<tr><td>After Convolutional Calculations</td><td>If no other layers, the resulting image is sent to the fully connected layer and further weight analysis are performed</td><td>Send the result (1 or 0) to the voting layer and proceed with output calculation</td></tr>
<tr><td>Training Time Per Epoch</td><td>104.83 sec.</td><td>436.59 sec.</td></tr>
<tr><td>Training Memory</td><td>1,324.3 MiB</td><td>749.9 MiB</td></tr>
<tr><td>Testing Memory</td><td>77.4 MiB</td><td>0.00 MiB</td></tr>
</table>

data [17]. However the class output of each image is converted to a float value by randomly pulling a value from a normal distribution when class value is the mean and taking 0.02 as the standard deviation.

The images in the y-coordinate prediction dataset are cropped to be in the size of $200 \times 199 \times 1$. Both CNN and C-RTM contains 500 filters in them (clauses in the C-RTM case). When the number of filters in both CNN and C-RTM are similar, the performance gap, in terms of MAE, does not significantly vary with the number of filters. Hence, the number of filters we use here is a random number and keep it similar for both CNN and C-RTM.

Table H.9 outlines the performance of both CNN and C-RTM on y-coordinate prediction dataset. As one can notice, the C-RTM outperforms CNN by obtaining a better MAE. The 10×10 CNN filters containing float numbers moves to 381 locations of every image to perform 38,100 number of multiplications and 37,719 number of summations. The selected C-RTM filter with the same size moves only to 36 locations and performs only 648 number of **AND** operations. Even though the CNN takes lesser training time on y-coordinate prediction dataset, C-RTM consumes significantly less training and testing memory. These are 1,934.1 MiB compared to 578.8 MiB and 55.5 MiB compared to 0.00 MiB in numbers, respectively on training and testing.

The analysis in [17] tries to see the performance of CNN on MNIST-regression dataset. They convert the class labels into regression outputs by taking values from a normal distribution when class value is the mean of the distribution. They analyse the variation of the performance when different standard deviation values are used. As was foreseeable, the MAE decreases when the standard deviation increases. However in this study, we are just interested in comparing the performance of CNN and C-RTM. Hence, we just select one of their standard deviation values (0.02) to create the dataset. Then, we here build our own CNN to make a fair comparison between CNN and C-RTM by assigning both of them the same number of filters (20,000).

The CNN obtains slightly better performance on MNIST-regression dataset as indicated in Table H.10. Similar to the earlier cases with other datasets, filter values in CNN filters are floats while filter values in C-RTM filters are binary. The 10 filters in CNN moves to 38 image locations and performs 3,800 number of multiplications and 3,762 summations while the selected 10 filter in C-RTM moves only to 2 locations and performs merely 40 number of **AND** operations. The behavior of CNN and C-RTM on MNIST-regression dataset in terms of the consumption of training time and memory is similar to behavior on y-coordinate prediction dataset. In other words, even though the CNN takes roughly 4.16 times less training time, it consumes 1.77 time more training memory and 77.4 MiB testing memory while C-RTM uses close to zero testing memory.

H.5 Conclusion

This paper proposed the Convolutional-Regression Tsetlin Machine (C-RTM), a novel ensemble approach based on the classic Tsetlin Machine that supports continuous output problems in image analysis. The C-RTM combines properties of both Convolutional

Tsetlin Machine (CTM) and Regression Tsetlin Machine (CTM). The patterns in images are identified using the convolution operations as in the CTM and the identified patterns are then mapped into a continuous output as in the RTM. The learning in the C-RTM is a combination of learning in both CTM and RTM. The prediction power of this novel approach was studied using 72 different datasets, with noise-free and noisy training data. Our empirical results showed the competitive performance of C-RTM compared to two Convolutional Neural Networks (CNNs). However, the C-RTM has the additional advantage of interpretability. Additionally, two real world-datasets were also used to analyze the performance of C-RTM compared to CNN. On one of these two datasets, CNN outperforms C-RTM, and on the other, C-RTM outperforms CNN. However, on both datasets, the number of convolutional calculations of C-RTM filters is significantly lower. As they learn the location of the patterns, convolutional calculations can be performed only on some of the image locations. This helps C-RTM to consume significantly low memory both at training and testing.

Bibliography

[1] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. “Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications”. In: *IEEE Access* 7 (2019), pp. 115134–115146.

[2] Adrian Phoulady, Ole-Christoffer Granmo, Saeed Rahimi Gorji, and Hady Ahmady Phoulady. “The Weighted Tsetlin Machine: Compressed Representations with Clause Weighting”. In: *Ninth International Workshop on Statistical Relational AI (StarAI 2020)*. 2020.

[3] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. “Extending the Tsetlin Machine with Integer-Weighted Clauses for Increased Interpretability”. In: *IEEE Access* 9 (2021), pp. 8233–8248.

[4] Saeed Rahimi Gorji, Ole-Christoffer Granmo, Adrian Phoulady, and Morten Goodwin. “A Tsetlin Machine with Multigranular Clauses”. In: *Lecture Notes in Computer Science: Proceedings of the Thirty-ninth International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI-2019)*. Vol. 11927. Springer International Publishing, 2019.

[5] Saeed Gorji, Ole Christoffer Granmo, Sondre Glimsdal, Jonathan Edwards, and Morten Goodwin. “Increasing the Inference and Learning Speed of Tsetlin Machines with Clause Indexing”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2020.

[6] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. “The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression”. In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

[7] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. “A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer. 2019, pp. 564–578.

[8] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. “Adaptive Sparse Representation of Continuous Input for Tsetlin Machines Based on Stochastic Searching on the Line”. In: *In Preparation* (2020).

[9] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. “Understanding of a Convolutional Neural Network”. In: *2017 International Conference on Engineering and Technology (ICET)*. IEEE. 2017, pp. 1–6.

[10] Matthew D Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[11] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. “The Convolutional Tsetlin Machine”. In: *arXiv preprint:1905.09688* (2019).

[12] Chenhao Cui and Tom Fearn. “Modern Practical Convolutional Neural Networks for Multivariate Regression: Applications to NIR Calibration”. In: *Chemometrics and Intelligent Laboratory Systems* 182 (2018), pp. 9–20.

[13] Ine L Jernelv, Dag Roar Hjelme, Yuji Matsuura, and Astrid Aksnes. “Convolutional Neural Networks for Classification and Regression Analysis of One-Dimensional Spectral Data”. In: *arXiv preprint arXiv:2005.07530* (2020).

[14] Tomoyoshi Shimobaba, Takashi Kakue, and Tomoyoshi Ito. “Convolutional Neural Network-Based Regression for Depth Prediction in Digital Holography”. In: *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*. IEEE. 2018, pp. 1323–1326.

[15] Hilal Tayara, Kim Gil Soo, and Kil To Chong. “Vehicle Detection and Counting in High-Resolution Aerial Images Using Convolutional Regression Neural Network”. In: *IEEE Access* 6 (2017), pp. 2220–2230.

[16] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Lei Jiao, and Morten Goodwin. “The regression Tsetlin Machine: A Tsetlin Machine for Continuous Output Problems”. In: *EPIA Conference on Artificial Intelligence*. Springer. 2019, pp. 268–280.

[17] Ziheng Wang, Su Wu, Chang Liu, Shaozhi Wu, and Kai Xiao. “The Regression of MNIST Dataset Based on Convolutional Neural Network”. In: *International Conference on Advanced Machine Learning Technologies and Applications*. Springer. 2019, pp. 59–68.

Paper I

A Multi-Step Finite-State Automaton for Arbitrarily Deterministic Tsetlin Machine Learning

Due to the high arithmetic complexity and scalability challenges of deep learning, there is a critical need to shift research focus towards energy efficiency. Tsetlin Machines (TMs) are a recent approach to machine learning that has demonstrated significantly reduced energy compared to neural networks alike, while providing comparable accuracy on several benchmarks. However, TMs rely heavily on energy-costly random number generation to stochastically guide a team of Tsetlin Automata (TA) in TM learning. In this paper, we propose a novel finite-state learning automaton that can replace the TA in the TM, for increased determinism. The new automaton uses multi-step deterministic state jumps to reinforce sub-patterns, without resorting to randomization. A determinism parameter d finely controls trading off the energy consumption of random number generation, against randomization for increased accuracy. Randomization is controlled by flipping a coin before every d'th state jump, ignoring the state jump on tails. E.g., $d = 1$ makes every update random and $d = \infty$ makes the automaton completely deterministic. Both theoretically and empirically, we establish that the proposed automaton converges to the optimal action almost surely. Further, used together with the TM, only substantial degrees of determinism reduces accuracy. Energy-wise, random number generation constitutes switching energy consumption of the TM, saving up to 11 mW power for larger datasets with high d values. Our new learning automaton approach thus facilitate low-energy machine learning.

I.1 Introduction

State-of-the-art deep learning (DL) requires massive computational resources, resulting in high energy consumption [1] and scalability challenges [2]. Thus, there is a critical need to shift research focus towards dealing with energy efficiency [3, 4]. Tsetlin Machines [5] (TMs) are a recent approach to machine learning (ML) that has demonstrated significantly reduced energy usage compared to neural networks alike [6, 7]. Using a linear combination of conjunctive clauses in propositional logic, the TM has obtained competitive performance in terms of accuracy [8, 9, 10], memory footprint [10], energy [6], and learning speed [10, 6] on diverse benchmarks (image classification, regression and natural language understanding). Furthermore, the rules that TMs build seem to be interpretable, similar to the branches in a decision tree (e.g., in the form **if** X **satisfies** condition A **and not** condition B **then** Y = 1) [8]. The reported small memory footprint and low energy consumption make the TM particularly attractive for addressing the scalability and energy challenge in ML.

Recent progress on TMs. Recent research reports several distinct TM properties. The TM can be used in convolution, providing competitive performance on MNIST, Fashion-MNIST, and Kuzushiji-MNIST, in comparison with CNNs, K-Nearest Neighbor, SVMs, Random Forest, Gradient Boosting, BinaryConnect, Logistic Circuits and ResNet [10]. The TM has also achieved promising results in natural language processing, such as text classification [8], word sense disambiguation [11] and sentiment analysis [12]. By introducing clause weights, it has been demonstrated that the number of clauses can be reduced by up to 50×, without loss of accuracy [13]. Further, hyper-parameter search can be simplified with multi-granular clauses, eliminating the pattern specificity parameter [14]. By indexing the clauses on the features that falsify them, up to an order of magnitude faster inference and learning has been reported [15]. Additionally, regression TMs compare favorably with Regression Trees, Random Forest Regression, and Support Vector Regression [9]. In [16], stochastic searching on the line automata [17] learn integer clause weights, performing on-par or better than Random Forest, Gradient Boosting and Explainable Boosting Machines. While TMs are binary throughout, thresholding schemes open up for continuous input [18]. Finally, TMs have recently been shown to be fault-tolerant, completely masking stuck-at faults [19]. The convergence property of TM has recently been studied in [20, 21].

Paper Contributions. TMs rely heavily on energy-costly random number generation to stochastically guide a team of TAs to a Nash Equilibrium of the TM game. In this paper, we propose a novel finite state learning automaton that can replace the TAs of the TM, for increased determinism. The new automaton uses multi-step deterministic state jumps to reinforce sub-patterns. Simultaneously, flipping a coin to skip every d'th state update ensures diversification by randomization. The d-parameter thus allows the degree of randomization to be finely controlled. Both theoretically and empirically, we establish that the proposed new automaton converges to the optimal action almost surely, when it is trained over an infinite time horizon while having infinite number of memory states. We further evaluate the performance of TM with this new automaton empirically on five datasets, demonstrating that the d-parameter can be used to trade off accuracy against energy consumption.

Paper Organization. In Section 2, we introduce our new type of Learning Automaton (LA) – the multi-step variable-structure finite-state LA (MVF-LA). The convergence of the MVF-LA is studied both theoretically and empirically in Section 3. Replacing the

TA with MVF-LA, we describe the Arbitrarily Deterministic TM (ADTM) in Section 4. Then, in Section 5, we evaluate ADTM empirically using five datasets. The performance of ADTM is investigated by varying the d-parameter, contrasting against the regular TM and five other state-of-the-art machine learning algorithms. Effect of determinism on energy consumption is discussed in Section 6. We conclude our work in Section 7.

I.2 A Multi-Step Finite-State Learning Automaton

The origins of LA [22] can be traced back to the work of M. L. Tsetlin in the early 1960s [23]. The objective of an LA is to learn the optimal action through trial and error in a stochastic environment. Various types of LAs are available depending on the nature of the application [24]. Due to their computational simplicity, we here focus on two-action finite-state LA, which we extend by introducing a novel periodically changing structure (variable structure).

An LA interacts with its environment iteratively. In each iteration, the action that a finite-state LA performs next is decided by its present state (the memory). The environment, in turn, randomly produces a reward or a penalty according to an unknown probability distribution, responding to the action selected by the LA. If the finite-state LA receives a reward, it reinforces the action performed by moving to a "deeper" state. If the action results in a penalty, it instead changes state towards the middle state, to weaken the performed action, ultimately switching to the other action. In this manner, with a sufficient number of states, a finite-state LA converges to selecting the action with the highest probability of producing rewards – the optimal action – with probability arbitrarily close to 1.0 [22].

The transitions between states can be deterministic or stochastic. Deterministic transitions occur with probability 1.0, while stochastic transitions are randomly performed based on a preset probability. If the transition probabilities are changing, we have a variable structure automaton, otherwise, we have one with fixed structure. The pioneering TA, depicted in Figure I.1, is a deterministic fixed-structure finite-state automaton [23]. The state transition graph in the figure depicts a TA with $2N$ states. States 1 to N maps to Action 1 and states $N+1$ to $2N$ maps to Action 2.

While the TA changes state in single steps, the deterministic Krinsky Automaton introduces multi-step state transitions [22]. The purpose is to reinforce an action more strongly when it is rewarded, and more weakly when penalized. The Krinsky Automaton behaves as a TA when the response from the environment is a penalty. However, when it is a reward, any state from 2 to N transitions to state 1, and any state from $N+1$ to $2N-1$ transitions to state $2N$. In effect, N consecutive penalties are needed to offset a single reward.

Another variant of LA is the Krylov Automaton. A Krylov Automaton makes both deterministic and stochastic single-step transitions [22]. The state transitions of the Krylov Automaton is identical to those of a TA for rewards. However, when it receives a penalty, it performs the corresponding TA state change randomly, with probability 0.5.

We now introduce our new type of LA, the multi-step variable-structure finite-state

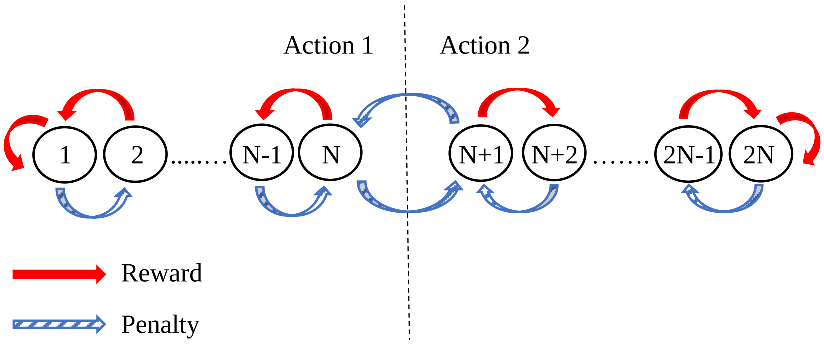


Figure I.1: Transition graph of a two-action Tsetlin Automaton with 2N memory states.

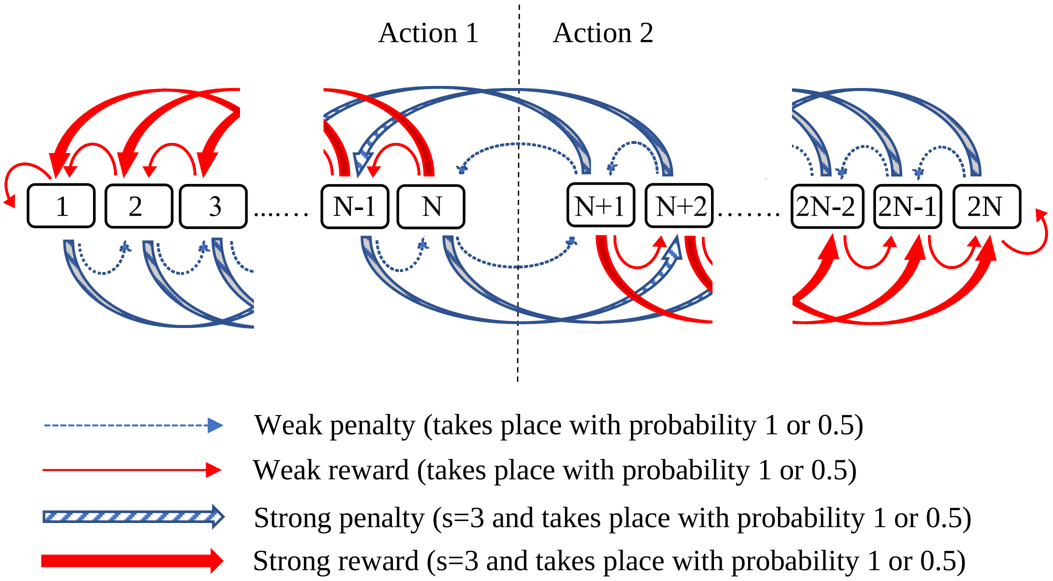

Action 1
Action 2
1
2
3
N-1
N
N+1
N+2
2N-2
2N-1
2N
Weak penalty (takes place with probability 1 or 0.5)
Weak reward (takes place with probability 1 or 0.5)
Strong penalty (s=3 and takes place with probability 1 or 0.5)
Strong reward (s=3 and takes place with probability 1 or 0.5)

Figure I.2: Transition graph of the Multi-Step Variable Structure Finite-State Learning Automaton.

LA (MVF-LA), shown in Figure I.2. The MVF-LA has two kinds of feedback, strong and weak. As covered in the next section, strong feedback is required by the TM to strongly reinforce frequent sub-patterns, while weak feedback is required to make the TM forget infrequent ones. To achieve this, weak feedback only triggers one-step transitions. Strong feedback, on the other hand, triggers s-step transitions. Thus, a single strong feedback is offset by s instances of weak feedback. Further, MVF-LA has a variable structure that changes *periodically*. That is, the MVF-LA switches between two different transition graph structures, one deterministic and one stochastic. The deterministic structure is as shown in the figure, while the stochastic structure introduces a transition probability 0.5, for every transition. The switch between structure is performed so that every d'th transition is stochastic, while the remaining transitions are deterministic.

I.3 Proof of the convergence of MVF-LA

In this section, we discuss the convergence of the proposed Multi-Step Variable Structure Finite-State Learning Automaton (MVF-LA). In Sec. I.3.1 we use a Markov chain model to analyze the convergence property of the MVF-LA. Thereafter, we simulate the MVF-LA and illustrate its convergence in different conditions in Sec. I.3.2.

I.3.1 Proof of the convergence of MVF-LA using Markov chain

To build the Markov chain, we utilize the memory states of the MVF-LA, i.e., 1 to $2N$, to represent the state space of Markov chain. The transition probability matrix, $\mathbf{P}$, for the Markov chain of MVF-LA is then to be established. The transition from any state i to another state j in MVF-LA can happen due to one of four types of feedback: strong reward, strong penalty, weak reward, and weak penalty. Apart from boundary conditions, the state transition from i to j may also not happen since the every d^{th} update is made with probability of 0.5. Considering these conditions, the probability of making the transition from i to j, $p_{i,j}$ can be calculated as follows.

Transition probability due to a strong reward, P_{sr} can be calculated as:

$$P_{\text{sr}} = P_{\text{Trans}} \times (1 - c) \times P_s \tag{I.1}$$

Here, P_{Trans} is the probability that any transition to other states happens. It includes two possibilities. (1) Transitions happen $d-1$ times for every d iteration. (2) Transitions happen with probability 0.5 at the remaining 1 of d iterations. Therefore, the overall probability of any transition, P_{Trans}, can be calculated as,

$$P_{\text{Trans}} = \frac{d-1}{d} + 0.5 \times \frac{1}{d} \tag{I.2}$$

The variable c in (I.1) is the penalty probability. The penalty probability c is the penalty probability of action 1 (c_1) if the starting state i in a transition from i to j is located in the state space of the action 1, i.e., $0 < i \leq N$. The penalty probability c on the other hand is the penalty probability of action 2 (c_2) if state i is in the state space of action 2, i.e., $N < i \leq 2N$. The probability P_s in the same equation is the probability of getting a strong feedback.

Similarly, transition probabilities due to strong penalty: P_{sp}, weak reward: P_{wr}, and weak penalty: P_{wp} are calculated as in (I.3), (I.4), and (I.5), respectively.

$$P_{sp} = P_{\text{Trans}} \times c \times P_s. \tag{I.3}$$

$$P_{wr} = P_{\text{Trans}} \times (1 - c) \times (1 - P_s). \tag{I.4}$$

$$P_{wp} = P_{\text{Trans}} \times c \times (1 - P_s). \tag{I.5}$$

Using the above transition probabilities, we form the transition probability matrix, $\mathbf{P}$ for the MVF-LA in Figure I.2. Matrix $\mathbf{P}$ exhibits the Markov chain property that the sum of the probabilities of each raw equals to one: $\sum_j p_{i,j} = 1$. For instance, consider the MVF-LA in Figure I.2. Here, when the starting state of a transition is N, a strong reward moves the state from N to $N-3$ ($s = 3$). Similarly, a weak reward moves the state from N to $N-1$. Weak and strong penalties, on the other hand, move the state N

in the state space of action 1 to the state space of action 2. While a weak penalty moves the state from N to $N+1$, a strong penalty moves it to $N+3$. The state N stays on the same state with probability P_{non}, where P_{non} is equal to $(1 - P_{\text{Trans}})$. At boundaries, if any of the above transitions can't be made, that transition probability is accommodated in P_{non}.

$$\mathbf{P} = \begin{array}{c||cccccc|cccccc||}
 & 1 & 2 & 3 & .. & N-1 & N & N+1 & N+2 & .. & 2N-2 & 2N-1 & 2N \\
1 & P_{\text{non}} & P_{wp} & - & .. & - & - & - & - & .. & - & - & - \\
2 & P_{wr} & P_{\text{non}} & P_{wp} & .. & - & - & - & - & .. & - & - & - \\
3 & - & P_{wr} & P_{\text{non}} & .. & - & - & - & - & .. & - & - & - \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
N-1 & - & - & - & .. & P_{\text{non}} & P_{wp} & - & P_{sp} & .. & - & - & - \\
N & - & - & - & .. & P_{wr} & P_{\text{non}} & P_{wp} & - & .. & - & - & - \\
\hline
N+1 & - & - & - & .. & - & P_{wp} & P_{\text{non}} & P_{wr} & .. & - & - & - \\
N+2 & - & - & - & .. & P_{sp} & - & P_{wp} & P_{\text{non}} & .. & - & - & - \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
2N-2 & - & - & - & .. & - & - & - & - & .. & P_{\text{non}} & P_{wr} & - \\
2N-1 & - & - & - & .. & - & - & - & - & .. & P_{wp} & P_{\text{non}} & P_{wr} \\
2N & - & - & - & .. & - & - & - & - & .. & - & P_{wp} & P_{\text{non}}
\end{array}$$

The Algorithm 6 shows the step-by-step procedure of building the transition matrix of MVF-LA.

Clearly, this Markov chain is indeed recurrent and non-periodical, and thus it is a *ergodic Markov chain.* As in [25], the probability of staying at a particular state at time n, $\pi(n)$, in the Markov chain can be computed as,

$$\pi(n) = \pi(n-1)\mathbf{P} = \pi(n-2)\mathbf{P}^2 = \cdots = \pi(0)\mathbf{P}^n \tag{I.6}$$

Here, $\pi(n)$ represents all the states in MVF-LA, $\pi(n) = [\pi_1(n), \pi_2(n), \ldots, \pi_{2N}(n)]$ and $\sum_i \pi_i(n) = 1$. When n goes to infinity, we obtain the steady state probabilities, π^*. The steady state probabilities π^* are independent of the initial state. Hence, we can also calculate π^* by,

$$\pi^* = \pi^* \mathbf{P} \tag{I.7}$$

Theoretically, from (I.6) and (I.7), π^* can be obtained by multiplying $\mathbf{P}$ itself for infinite number of times, $\pi^* = (\mathbf{P})^\infty$. In practice, we multiply $\mathbf{P}$ itself for a sufficiently large number of times until its entries converge and then the steady state probabilities are obtained. Once we know the steady state probabilities, we sum up the steady state probabilities that correspond to the action that has the lowest probability of penalty. If the sum of the probabilities converges to 1 when N approaches to infinity, we can conclude that, with sufficiently large number of memory states per actions, the LA converges to the correct action. For MVF-LA, based on the calculation of Algorithm 6, we conclude that as long as the best action's penalty probability is less than 0.5, the action selection probability converges to 1 when N goes to infinity.

Algorithm 6 Calculating the stationary distribution of the Makov chain for MVF-LA

```
Input: Number of states per action, $N$; Number of strong jumps, $s$; Deterministic parameter, $d$; Probability of getting a strong feedback, $P_s$; Penalty probability for action 1, $c_1$; Penalty probability for action 2, $c_2$.
Output: Limiting Matrix
Initialize: Transition Probability Matrix $\mathbf{P}$                ▷ $\mathbf{P}$ requires $2N$ by $2N$ space
Function:
for $i = 1, ..., 2N$ do
    if $i \leq N$ then
        $c = c_1$                                  ▷ $c =$ Penalty probability for action 1
    else
        $c = c_2$                                  ▷ $c =$ Penalty probability for action 2
    end if
    for $j = 1, ..., 2N$ do
        Compute $P_{Trans}$                                                    ▷ (I.2)
        if $(i \leq N$ and $i - s = j)$ or $(i > N$ and $i + s = j)$ then        ▷ strong reward
            $P_{i,j} = P_{\text{Trans}} \times P_s \times (1 - c)$
        else if $(i \leq N$ and $i + s = j)$ or $(i > N$ and $i - s = j)$ then ▷ strong penalty
            $P_{i,j} = P_{\text{Trans}} \times P_s \times c$
        else if $(i \leq N$ and $i - 1 = j)$ or $(i > N$ and $i + 1 = j)$ then  ▷ weak reward
            $P_{i,j} = P_{\text{Trans}} \times (1 - P_s) \times (1 - c)$
        else if $(i \leq N$ and $i + 1 = j)$ or $(i > N$ and $i - 1 = j)$ then ▷ weak penalty
            $P_{i,j} = P_{\text{Trans}} \times (1 - P_s) \times c$
        else (staying on the same state)
            if $i = j = 1$ or $i = j = 2N$ then ▷ weak and strong updates can't be made
                $P_{i,j} = (1 - P_{\text{Trans}}) + P_{\text{Trans}} \times (1 - c) \times (1 - P_s) + P_{\text{Trans}} \times (1 - c) \times P_s$
            else if $i = j$ then
                if $(i \leq N$ and $i - s < 0)$ or $(i > N$ and $i + s > 2N)$ then ▷ only weak updates can be made
                    $P_{i,j} = (1 - P_{\text{Trans}}) + P_{\text{Trans}} \times (1 - c) \times P_s$
                else                          ▷ both weak and strong updates can be made
                    $P_{i,j} = 1 - P_{\text{Trans}}$
                end if
            else
                $P_{i,j} = 0$
            end if
        end if
    end for
    $\mathbf{P}[i, j] \leftarrow$ Update                                  ▷ $\mathbf{P}[i, j] = p_{i,j}$
end for
End Function
Return: $(\mathbf{P})^{\infty}$                        ▷ Return the stationary distribution
```

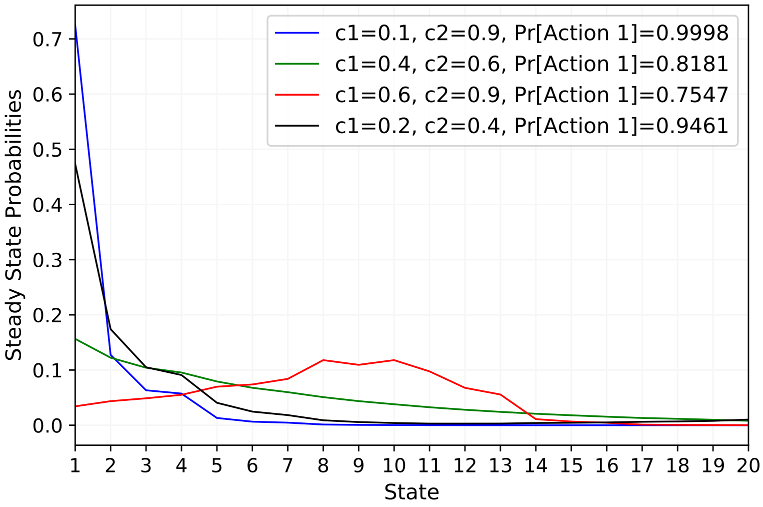


Figure I.3: The steady state probabilities of an MVF-LA with different penalty probabilities when $N = 10$.

To illustrate the convergence property of MVF-LA, we form different transition matrices with distinct parameter configurations using Algorithm 6. We keep s, P_s, and d as constant at 3, 0.67, and 10, respectively for the analysis. Without loss of generality, we always set action 1 as the best action. The steady state probability distribution over the states of MVF-LA can be seen in Figure I.3 when $N = 10$. The sum of the steady state probabilities of action 1, Pr[Action 1], for different penalty probability configurations are also illustrated in the figure. Although with only $N = 10$ memories, the action selection probability for action 1 is convincingly higher (> 0.94) when $c_1 = 0.1$, $c_2 = 0.9$ and $c_1 = 0.2$, $c_2 = 0.4$.

The probability of selecting action 1 for the remaining penalty probability setups are higher than 0.75. However, the probability distribution of these two setups show that the MVF-LA has not made the decision of selecting action 1 confidently as the steady state probabilities of the end states of action 1 are relatively lower than those of the previous two setups.

Nevertheless, according to the theory, the probability of selecting action 1 should increase with N and it will reach 1 as N goes to infinity, given that the best action has a penalty probability less than 0.5. This is verified by the plots in Figure I.4 where probability of selecting action 1 reaches 1 when N increases for all the cases except when $c_1 = 0.6$ and $c_2 = 0.9$. This is because the lowest penalty probability (c_1 in this case) is not less than 0.5. Therefore, even though the difference between penalty probabilities of the case $c_1 = 0.4$ and $c_2 = 0.6$ (0.2) lower than the case $c_1 = 0.6$ and $c_2 = 0.9$ (0.3), the case of $c_1 = 0.4$ and $c_2 = 0.6$ can be perfectly learned when N increases while the other case struggles.

I.3.2 Simulation analysis of MVF-LA

In this section, we simulate the MVF-LA and see if it behaves similar to the above stated convergence properties. First, we build the MVF-LA and iteratively update its

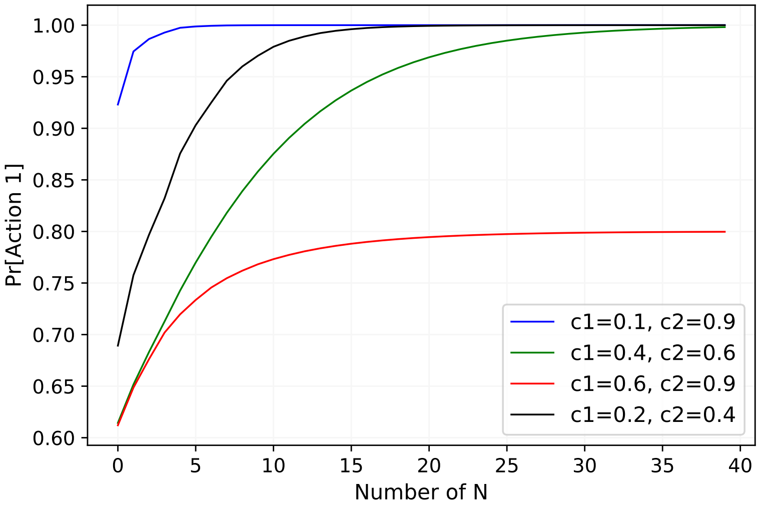


Figure I.4: The increase of the probability of selecting the correct action with N.

states by stochastically generating feedbacks for MVF-LA's actions according to known penalty probabilities. Then we analyse the behavior of the MVF-LA and compare with its theoretical outputs.

Here we introduce the new quantity $M(n)$, which is the average penalty after n training iterations. The $M(n)$ for a two-action automaton is computed as $M(n) = c_1 Pr[Action1] + c_2 Pr[Action2]$ [22]. According to the theory stated in Sec. I.3.1, when n and N go to infinity, the probability of selecting the action which has the lowest penalty probability should reaches 1 (consequently, the probability of selecting the other action goes to 0). Therefore, when n and N go to infinity, the average penalty, $M(n)$ should approximate to the lowest penalty probability.

In our simulation, to make the analysis easier, we always set the lowest penalty probability to the action 1. Then, we first analyse the variation of Pr[Action 1] against the number of training iterations, n. Figure I.5 depicts the 20-iterations moving average of

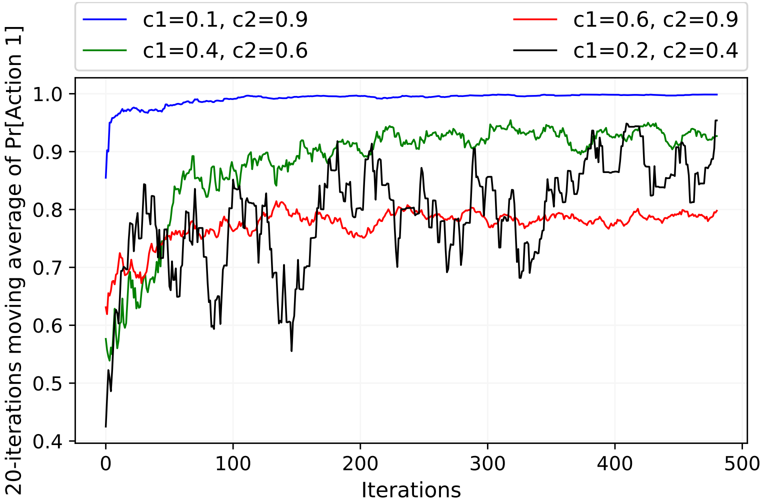


Figure I.5: The variation of the Pr[Action 1] against the number of training iterations, n for different penalty probabilities.

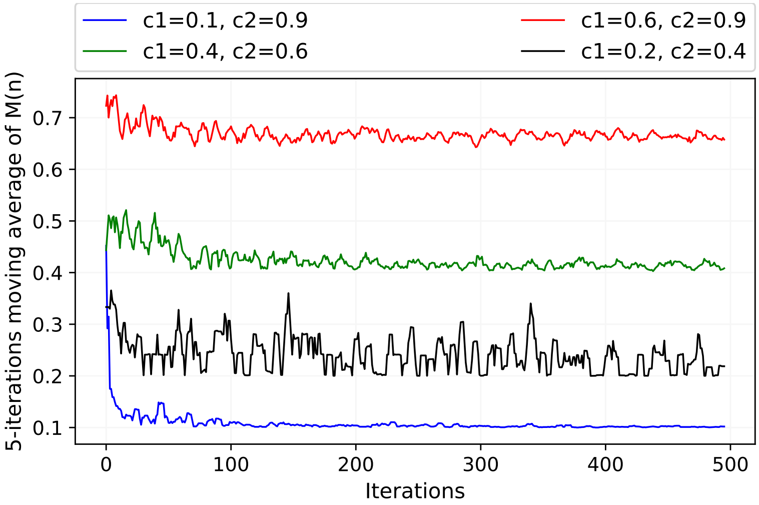


Figure I.6: The variation of average penalty $M(n)$ against the number of training iterations, n for different penalty probabilities.

Pr[Action 1] against the number of iterations. At each experiment round, the number of training iteration, n is increased and the final Pr[Action 1] is recorded. The N, s, d, and P_s in this simulation are fixed at 20, 3, 10, and 0.67. As expected, the Pr[Action 1] increases with n. The case with $c_1 = 0.1$ and $c_2 = 0.9$ has already approaches probability 1. The probabilities of selecting action 1 in experiments with $c_1 = 0.4$, $c_2 = 0.6$ and $c_1 = 0.2$, $c_2 = 0.4$ are slowly approaching 1. From these two, Pr[Action 1] variation of the case $c_1 = 0.4$ and $c_2 = 0.6$ is more stable than the other. The Pr[Action 1] variation of the experiment with $c_1 = 0.6$ and $c_2 = 0.9$ has stabilized around 0.8.

The change of the average penalty, $M(n)$ over n for the same experiment is illustrated in Figure I.6. Except for the experiment with $c_1 = 0.6$ and $c_2 = 0.9$, $M(n)$ has approximated to the lowest penalty probability with n. The 5-iterations moving average for the experiment with $c_1 = 0.2$ and $c_2 = 0.4$ is again unsteady. The reason here is both c_1 and c_2 are lower then 0.5 and therefore, there is a higher chance to get a reward for both the actions.

In the next arrangement, the change of Pr[Action 1] against n is studied for distinct N values. For this experiment, the c_1 and c_2 are fixed at 0.4 and 0.6, respectively. As expected, Figure I.7 displays that Pr[Action 1] of MVF-LA with higher N reaches highest possible probability faster.

I.4 The Arbitrarily Deterministic TM (ADTM)

In this section, we introduce the details of the ADTM, shown in Figure I.8, where the TA is replaced by the MVF-LA. The purpose of the ADTM is to control the amount of stochasticity generated, thus allowing management of energy consumption during learning.

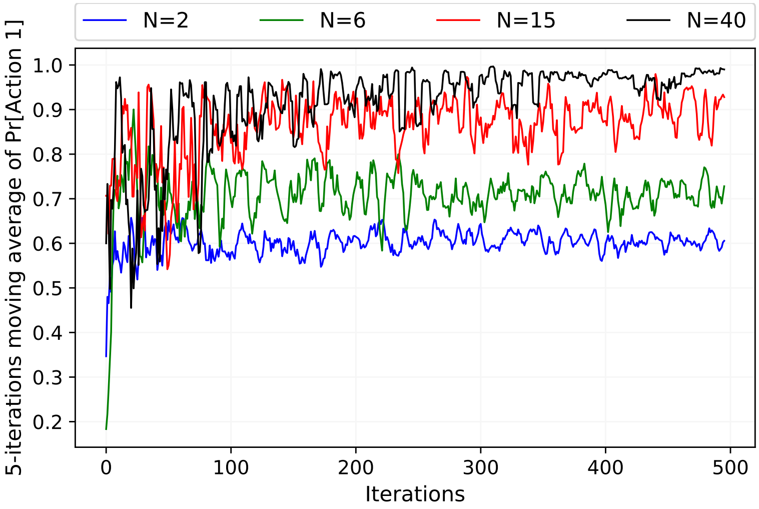


Figure I.7: The variation of the Pr[Action 1] against the number of training iterations, n for different number of states per action, N.

I.4.1 ADTM Inference

Input Features. Like the TM, an ADTM takes a feature vector of o propositional variables as input, $\boldsymbol{X} = [x_1, x_2, x_3, \ldots, x_o]$, to be classified into one of two classes, $y = 0$ or $y = 1$. These features are extended with their negation, to produce a set of literals: $\mathbf{L} = [x_1, x_2, \ldots, x_o, \neg x_1, \neg x_2, \ldots, \neg x_o] = [l_1, l_2, \ldots, l_{2o}]$.

Clauses. Patterns are represented by m conjunctive clauses. As shown for Clause-1 in the figure, a clause in the TM comprises $2o$ MVF-LAs, each controlling the inclusion of a specific literal. Let the set I_j, $I_j \subseteq \{1, \ldots, 2o\}$ denote the indexes of the literals that are included in clause j. When evaluating clause j on input literals $\boldsymbol{L}$, the literals included in the clause are ANDed: $c_j = \bigwedge_{k \in I_j} l_k, j = 1, \ldots, m$. Note that the output of an empty clause, $I_j = \emptyset$, is 1 during learning and 0 during inference.

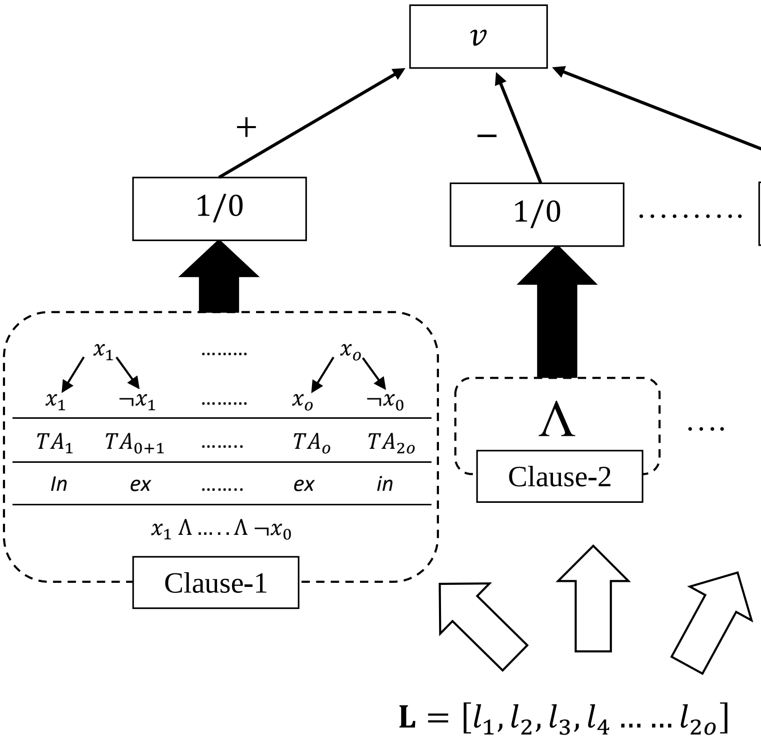
Classification. In order to identify the sub-patterns associated with both of the classes of a two-class ADTM, the clauses are grouped in two. The number of clauses employed is a user set parameter m. Half of the clauses are assigned positive polarity (c_j^+). The other half is assigned negative polarity (c_j^-). The clause outputs, in turn, are combined into a classification decision through summation and thresholding using the unit step function $u(v) = 1$ **if** $v \geq 0$ **else** 0:

$$\hat{y} = u\left(\sum_{j=1}^{m/2} c_j^+(X) - \sum_{j=1}^{m/2} c_j^-(X)\right). \tag{I.8}$$

That is, classification is based on a majority vote, with the positive clauses voting for $y = 0$ and the negative for $y = 1$.

I.4.2 The MVF-LA Game and Orchestration Scheme

The MVF-LAs in ADTM are updated by so-called Type I and Type II feedback. Depending on the class of the current training sample (X, y) and the polarity of the clause



v
+
−
−
1/0
1/0
1/0
x_1
x_o
x_1 $\neg x_1$ x_o $\neg x_0$
TA_1 TA_{0+1} TA_o TA_{2o}
In ex ex in
$x_1 \wedge \ldots \wedge \neg x_0$
Clause-1
$\wedge$
Clause-2
$\wedge$
Clause-m
$\mathbf{L} = [l_1, l_2, l_3, l_4 \ldots \ldots l_{2o}]$

Figure I.8: The ADTM structure.

(positive or negative), the type of feedback is decided. Clauses with positive polarity receive Type I feedback when the target output is $y = 1$, and Type II feedback when the target output is $y = 0$. For clauses with negative polarity, Type I feedback replaces Type II, and vice versa. In the following, we focus only on clauses with positive polarity.

Type I feedback: The number of clauses which receive Type I feedback is controlled by selecting them stochastically according to (I.9):

$$\frac{T - \max(-T, \min(T, v))}{2T}. \tag{I.9}$$

Above, $v = \sum_{j=1}^{m/2} c_j^+(X) - \sum_{j=1}^{m/2} c_j^-(X)$ is the aggregated clause output and T is a user set parameter that decides how many clauses should be involved in learning a particular sub-pattern. Increasing T proportionally with the number of clauses introduces an ensemble effect, for increased learning accuracy. Type I feedback consists of two kinds of sub-feedback: Type Ia and Type Ib. Type Ia feedback stimulates recognition of patterns by reinforcing the include action of MVF-LAs whose corresponding literal value is 1, however, only when the clause output also is 1. Note that an action is reinforced either by rewarding the action itself, or by penalizing the other action. Type Ia feedback is *strong*, with step size s (Figure I.2). Type Ib feedback, on the other hand, combats over-fitting by reinforcing the *exclude* actions of MVF-LAs when the corresponding literal is 0 or when the clause output is 0. Type Ib feedback is *weak* (Figure. I.2) to facilitate learning of frequent patterns.

Type II feedback: Clauses are also selected stochastically for receiving Type II feedback:

$$\frac{T + \max(-T, \min(T, v))}{2T}. \tag{I.10}$$

Type II feedback combats false positive clause output by seeking to alter clauses that output 1 so that they instead output 0. This is achieved simply by penalizing exclusion of literals of value 0. Thus, when the clause output is 1 and the corresponding literal value of an MVF-LA is 0, the exclude action of the MVF-LA is penalized. Type II feedback is *strong*, with step size s. Recall that in all of the above MVF-LA update steps, the parameter d decides the determinism of the updates.

I.5 Empirical Evaluation

We now study the performance of ADTM empirically using five real-world datasets.[1] The ADTM is compared against regular TMs to assess to what degree learning accuracy suffers from increased determinism. The ADTM is also compared against seven other state-of-the-are machine learning approaches: Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), Decision Trees (DTs), K-Nearest Neighbor (KNN), Random Forest (RF), Gradient Boosted Trees (XGBoost) [26], and Explainable Boosting Machines (EBMs) [27]. 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. Performance of these predictive models are summarized in Table I.6. We compute both F1-score (F1) and accuracy (Acc.) as performance measures. However, due to class imbalance, we emphasize F1-score when comparing the performance of the different predictive models.

I.5.1 Bankruptcy

The Bankruptcy dataset contains historical records of 250 companies[2]. The outcome, Bankruptcy or Non-bankruptcy, is characterized by six categorical features. We thus binarize the features using thresholding [18] before we feed them into the ADTM. We first tune the hyper-parameters of the TM and the best performance is reported in Table I.1, for $m = 100$ (number of clauses), $s = 3$ (step size for MVF-LA), and $T = 10$ (summation target). Each MVF-LA contains 100 states per action. The impact of determinism is reported in Table I.1, for varying levels of determinism. As seen, performance is indistinguishable for d-values 1, 10, and 100, and the ADTM achieves its highest classification accuracy. However, notice the slight decrease of F1-score and accuracy when determinism is further increased to 500, 1000, and 5000.

Figure I.9 shows how training and testing accuracy evolve over the training epochs. Only high determinism seems to influence learning speed and accuracy significantly. The performance of the other considered machine learning models is compiled in Table I.6. The best performance in terms of F1-score for the other models is obtained by ANN-3. However, ANN-3 is outperformed by the ADTM for all d-values except when $d = 5000$.

[1] An implementation of ADTM can be found at https://github.com/cair/Deterministic-Tsetlin-Machine.

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

Table I.1: Performance of TM and ADTM with different d on Bankruptcy

	TM	ADTM					
		d=1	d=10	d=100	d=500	d=1000	d=5000
F1	0.998	1.000	1.000	1.000	0.999	0.999	0.988
Acc.	0.998	1.000	1.000	1.000	0.999	0.999	0.987

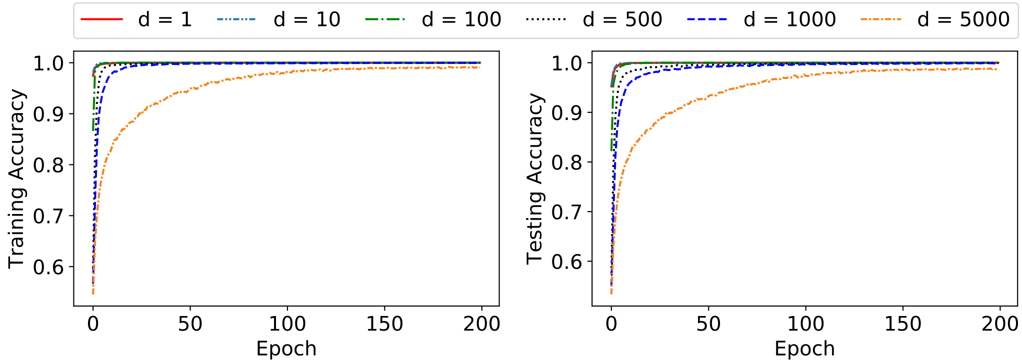


Figure I.9: Training and testing accuracy per epoch on Bankruptcy

I.5.2 Balance Scale

The Balance Scale dataset[3] contains three classes: balance scale tip to the right, tip to the left, or in balance. The class is decided by the size of the weight on both sides of the scale and the distance to each weight from the center. Hence the classes are characterized by four features. However, to make the output binary, we remove the “balanced” class ending up with 576 data samples. The ADTM is equipped with 100 clauses. Each MVF-LA is given 100 states per action. The remaining two parameters, i.e., s value and T are fixed at 3 and 10, respectively. Table I.2 contains the results obtained with TM and ADTM. Even though ADTM uses the same number of clauses as the TM, the performance with regards to F1-score and accuracy is better with ADTM when all updates on MVF-LAs are stochastic. The performance of the ADTM remains the same until the determinism-parameter surpasses 100. After that, performance degrades gradually.

Progress of training and testing accuracy per epoch can be found in Figure I.10. Each ADTM setup reaches its peak training and testing accuracy and becomes stable within a fewer number of training epochs. As can be seen, accuracy is maintained up to $d = 100$, thus reducing random number generation to 1% without accuracy loss. From the results listed in Table I.6 for the other machine learning approaches, EBM achieves the highest F1-score and accuracy.

I.5.3 Breast Cancer

The Breast Cancer dataset[4] contains 286 patients records related to the recurrence of breast cancer (201 with non-recurrence and 85 with recurrence). The recurrence of breast

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

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

Table I.2: Performance of TM and ADTM with different d on Balance Scale

	TM	ADTM					
		d=1	d=10	d=100	d=500	d=1000	d=5000
F1	0.945	0.982	0.983	0.982	0.968	0.951	0.911
Acc.	0.948	0.980	0.981	0.980	0.935	0.894	0.793

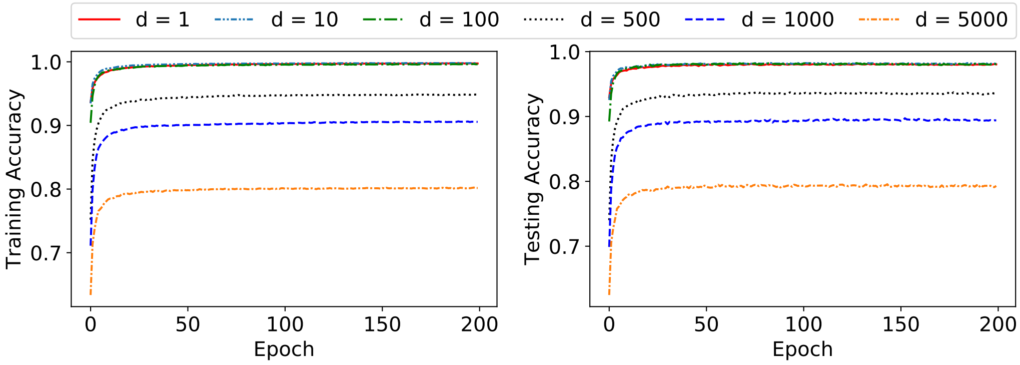


Figure I.10: Training and testing accuracy per epoch on the Balance Scale

cancer is to be estimated using nine features: Age, Menopause, Tumor Size, Inv Nodes, Node Caps, Deg Malig, Side (left or right), the Position of the Breast, and Irradiation. However, some of the patient samples miss some of the feature values. These samples are removed from the dataset in the present experiment. The ADTM is arranged with the following parameter setup: $m = 100$, $s = 5$, $T = 10$, and the number of states in MVF-LA per action is 100. The classification accuracy of the TM and ADTM are summarized in Table I.3. The performance of both TM and ADTM is here considerably lower than for the previous two datasets, and further decreases with increasing determinism. However, the F1 measures obtained by all the other considered machine learning models are also low, i.e., less than 0.500. The highest F1-score is obtained by ANN-1 and KNN.

Table I.3: Performance of TM and ADTM with different d on Breast Cancer

	TM	ADTM					
		d=1	d=10	d=100	d=500	d=1000	d=5000
F1	0.531	0.568	0.531	0.501	0.490	0.501	0.488
Acc.	0.703	0.702	0.698	0.691	0.690	0.690	0.693

The training and testing accuracy progress per epoch is reported in Figure I.11, showing a clear degradation of performance with increasing determinism.

I.5.4 Liver Disorders

The Liver Disorders dataset[5] was created by BUPA Medical Research and Development Ltd. (hereafter "BMRDL") during the 1980s as part of a larger health-screening database.

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

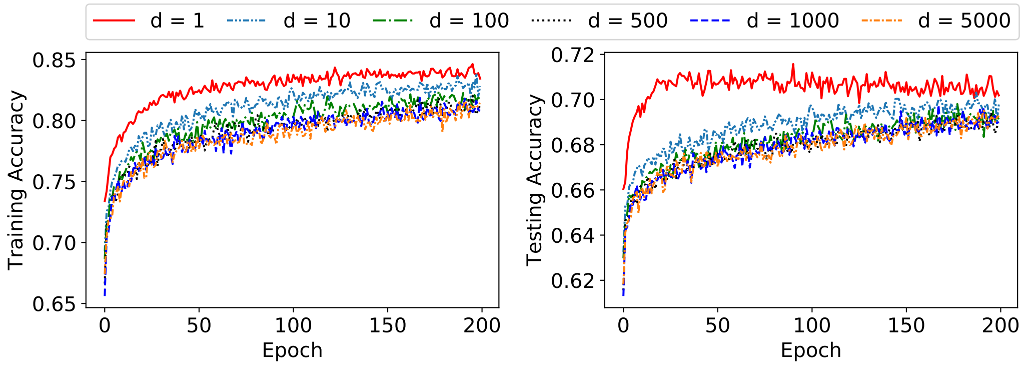


Figure I.11: Training and testing accuracy per epoch on Breast Cancer

Table I.4: Performance of TM and ADTM with different d on Liver Disorders

	TM	ADTM					
		d=1	d=10	d=100	d=500	d=1000	d=5000
F1	0.648	0.705	0.694	0.692	0.692	0.689	0.692
Acc.	0.533	0.610	0.610	0.612	0.612	0.610	0.611

The dataset consists of 7 attributes. However, [28] claim that many researchers have used the dataset incorrectly, considering the Selector attribute as the class label. Based on the recommendation of McDermott and Forsythof, we here instead use the Number of Half-Pint Equivalents of Alcoholic Beverages as the dependent variable, binarized using the threshold ≥ 3. The Selector attribute is discarded. The remaining attributes represent the results of various blood tests, and we use them as features.

Here, ADTM is given 10 clauses per class, with $s = 3$ and $T = 10$. Each MVF-LA action possesses 100 states. The performance of ADTM for different levels of determinism is summarized in Table I.4. For $d = 1$, the F1-score of ADTM is better than what is achieved with the standard TM. In contrast to the performance on previous datasets, the performance of ADTM on Liver Disorders dataset with respect to F1-score does not decrease significantly with d. Instead, it fluctuates around 0.690.

As shown in Figure I.12, unlike the other datasets, the ADTM with $d = 1$ requires more training rounds than with larger d-values, before it learns the final MVF-LA actions. It is also unable to reach the training accuracy obtained with higher d-values. Despite the diverse learning speed, testing accuracy becomes similar after roughly 50 training rounds. The other considered machine learning models obtain somewhat similar F1-scores, however,only DT, RF, and EBM surpass an F1-score of 0.700.

I.5.5 Heart Disease

The Heart Disease dataset[6] 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.

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

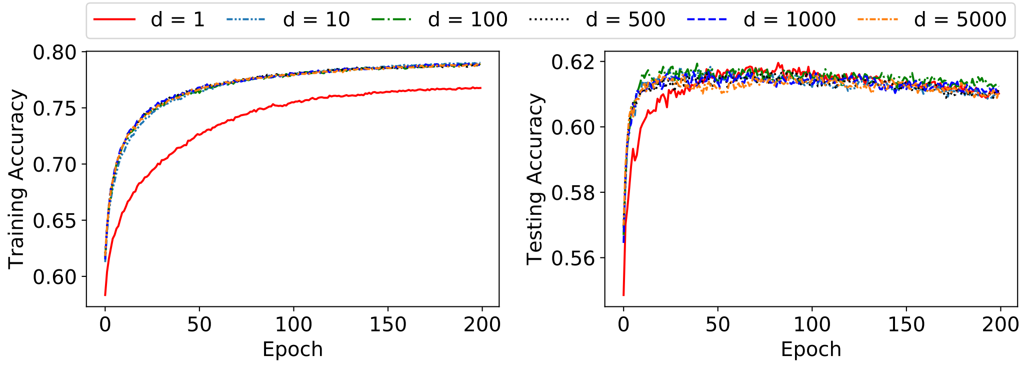


Figure I.12: Training and testing accuracy per epoch on Liver Disorders

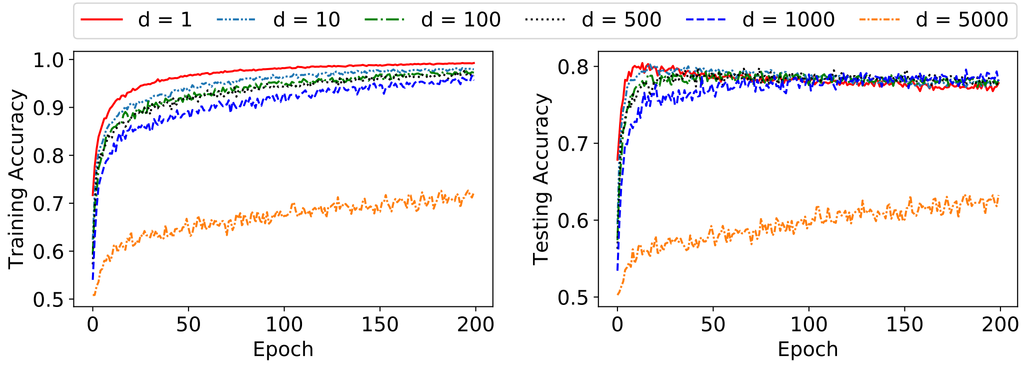


Figure I.13: Training and testing accuracy per epoch on Heart Disease

In this case, the ADTM is built on 100 clauses. The number of state transitions when the feedback is strong, s is equal to 3 while the target, T is equal to 10. The number of states per MVF-LA action in the ADTM is 100.

As one can see in Table I.5, the ADTM provides better performance than TM in terms of F1-score and accuracy when $d = 1$. F1-score then increases with d and peaks at $d = 100$. After some fluctuation, it drops to a value of 0.605 when $d = 5000$.

Figure I.13 shows similar training and testing accuracy for all d-values, apart from the significantly lower accuracy of $d = 5000$.

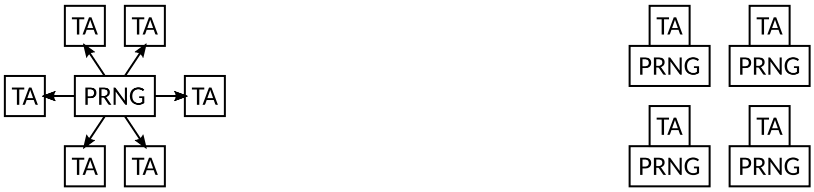
Out of other machine learning algorithms, EBM provides the best F1-score, as summarized in Table I.6. Even though ANN-1, ANN-2, DT, RF, and XGBoost obtain better F1-scores than TM, the F1 scores of ADTM when d equals to 1, 10, 100, 500, and 1000 are higher.

Table I.5: Performance of TM and ADTM with different d on Heart Disease

	TM	ADTM					
		d=1	d=10	d=100	d=500	d=1000	d=5000
F1	0.687	0.759	0.766	0.767	0.760	0.762	0.605
Acc.	0.672	0.778	0.780	0.783	0.773	0.781	0.633

Table I.6: Classification accuracy of selected machine learning models

	Bankruptcy		Balance Sca.		Breast Can.		Liver Dis.		Heart Dise.	
	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.
ANN-1	0.995	0.994	0.990	0.990	**0.458**	0.719	0.671	0.612	0.738	0.772
ANN-2	0.996	0.995	0.995	0.995	0.403	0.683	0.652	0.594	0.742	0.769
ANN-3	**0.997**	**0.997**	0.995	0.995	0.422	0.685	0.656	0.602	0.650	0.734
DT	0.993	0.993	0.986	0.986	0.276	0.706	0.728	0.596	0.729	0.781
SVM	0.994	0.994	0.887	0.887	0.384	0.678	0.622	0.571	0.679	0.710
KNN	0.995	0.994	0.953	0.953	**0.458**	**0.755**	0.638	0.566	0.641	0.714
RF	0.949	0.942	0.859	0.860	0.370	0.747	**0.729**	0.607	0.713	0.774
XGBoost	0.983	0.983	0.931	0.931	0.367	0.719	0.656	**0.635**	0.701	0.788
EBM	0.993	0.992	**1.000**	**1.000**	0.389	0.745	0.710	0.629	**0.783**	**0.824**



(a) Centralized PRNG

(b) Decentralized PRNG

Figure I.14: PRNG strategies for a) software TM; and b) hardware TM.

I.6 Effects of Determinism on Energy Consumption

Energy consumption of all TM implementations can be positively reduced by using ADTM, since random choice is a key mechanism in learning (see Section 4). This effect is especially notable in ASIC implementations aimed at low energy on-chip learning applications, where energy overheads are low (compared to a personal computer, for example).

While software implementations of the TM use centralized pseudorandom number generators (PRNGs) to facilitate the random choices (Figure I.14a), the ASIC implementation uses many smaller PRNGs localized to individual TAs to maximize parallelism (Figure I.14b). In the ASIC implementation of TM, linear feedback shift registers (LFSRs) are used as PRNGs due to their small size and simplicity [6]. Power is consumed by the PRNGs in the process of generating a new random number. This is referred to as *switching power*. In the TM, every TA update is randomized, and switching power is consumed by the PRNGs on every cycle. Additionally, power is also consumed by the PRNGs whilst idle. We term this *leakage power*. Leakage power is always consumed by the PRNGs whilst they are powered up, even when not generating new numbers.

In the ADTM with hybrid TA where the determinism parameter d is introduced, $d = 1$ would be equivalent to a TM where every TA update is randomized. $d = \infty$ means the ADTM is fully deterministic, and no random numbers are required from the PRNG. If a TA update is randomized only on the d^{th} cycle, the PRNGs need only be actively switched

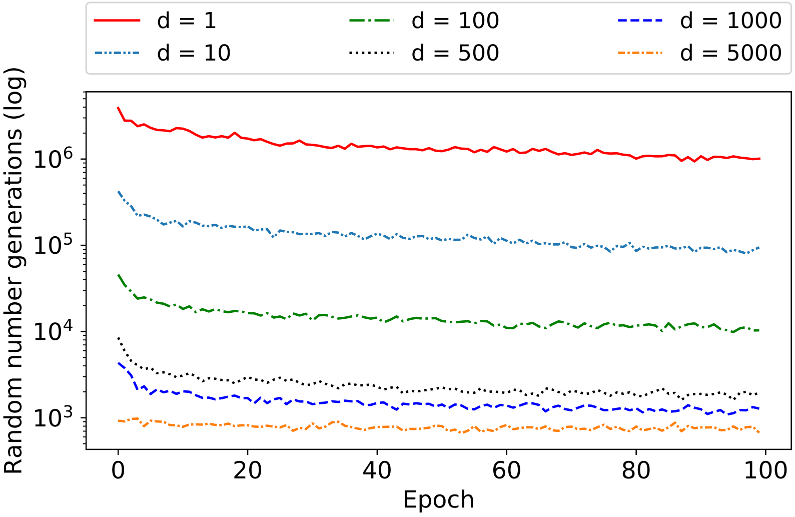


Figure I.15: Number of randomisation events per epoch for the Heart Disease dataset.

(and therefore consume *switching power*) for $\frac{1}{d}$ portion of the entire training procedure. The switching power consumed by the PRNGs accounts for 7% of the total system power when using a traditional TA (equivalent to $d = 1$). With $d = 100$ this is reduced to 0.07% of the system power, and with $d = 5000$ this is reduced further to 0.001% of the same. It can be seen that as d increases in the ADTM, the switching power consumed by the PRNGs tends to zero.

In the special case of $d = \infty$ the PRNGs are no longer required for TA updates since the TAs are fully deterministic – we can omit these PRNGs from the design and prevent their *leakage power* from being consumed. The leakage power of the PRNGs accounts for 32% of the total system power. On top of the switching power savings this equates to 39% of system power, meaning large power and therefore energy savings can be made in the ADTM.

Figure I.15 shows the number of randomisation events for different d values in the case of Heart Disease dataset. As expected, for lower d values, the number of events reduces drastically. For example, in the first iteration this number reduces by 4219X from d=1 to d=5000. Notice, how the number of these events also reduces further for both cases as the number of iterations increase [5]. The reduced number of events can be positively leveraged towards power minimization.

Table I.7 shows comparative training power consumption per datapoint (i.e. all TAs being updated concurrently) for two different d values: d=1 and d=5000. Typically, the overall power is higher for bigger datasets as they require increased number of concurrent TAs as well as PRNGs. As can be seen, the increase in d value reduces the power consumption by 11 mW in the case of Heart Disease dataset. This saving is made by reducing the switching activity in the PRNGs as explained above. More savings are made by larger d values as the PRNG concurrent switching activities are reduced.

Table I.7: Comparative power per datapoint with two different d values.

Dataset	Bankruptcy	Breast Can.	Balance Sca.	Liver Dis.	Heart Dise.
Power (d=1)	6.94 mW	15.8 mW	7.7 mW	12.6 mW	148.0 mW
Power (d=5000)	6.45 mW	14.7 mW	7.2 mW	11.8 mW	137.6 mW

I.7 Conclusion

In this paper, we proposed a novel finite-state learning automaton (MFV-LA) that can replace the Tsetlin Automaton in TM learning, for increased determinism, and thus reduced energy usage. The new automaton uses multi-step deterministic state jumps to reinforce sub-patterns. Simultaneously, flipping a coin to skip every d'th state update ensures diversification by randomization. The new d-parameter thus allows the degree of randomization to be finely controlled. E.g., $d = 1$ makes every update random and $d = \infty$ makes the automaton fully deterministic. First, theoretically, using Markov chain properties, we showed that MVF-LA is able to select the action which has the lowest penalty probability almost surely when both the number of training iterations and memory states are set to infinity. Then, we simulated the MVF-LA and analyzed its convergence empirically to support our theoretical inferences. Further, used together with TM, empirical results on five real-world datasets show that overall, only substantial degrees of determinism reduces accuracy. Energy-wise, the pseudorandom number generator contributes to switching energy consumption within the TM, which can be completely eliminated with $d = \infty$. We can thus use the new d-parameter to trade off accuracy against energy consumption, to facilitate low-energy machine learning.

Bibliography

[1] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP". In: *ACL*. 2019.

[2] J. Chen and X. Ran. "Deep Learning With Edge Computing: A Review". In: *Proc. of the IEEE* 107.8 (2019), pp. 1655–1674.

[3] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. "Estimation of Energy Consumption in Machine Learning". In: *Journal of Parallel and Distributed Computing* 134 (2019), pp. 75–88. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2019.07.007.

[4] Rishad Shafik, Alex Yakovlev, and Shidhartha Das. "Real-Power Computing". In: *IEEE Transactions on Computers* 67.10 (2018), pp. 1445–1461.

[5] Ole-Christoffer Granmo. "The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic". In: *arXiv preprint arXiv:1804.01508* (2018).

[6] Adrian Wheeldon, Rishad Shafik, Tousif Rahman, Jie Lei, Alex Yakovlev, and Ole-Christoffer Granmo. "Learning Automata Based Energy-efficient AI Hardware Design for IoT". In: *Philosophical Transactions of the Royal Society A* (2020). URL: https://eprints.ncl.ac.uk/268038.

[7] Jie Lei, Adrian Wheeldon, Rishad Shafik, Alex Yakovlev, and Ole-Christoffer Granmo. "From Arithmetic to Logic based AI: A Comparative Analysis of Neural Networks and Tsetlin Machine". In: *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE. 2020, pp. 1–4.

[8] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications". In: *IEEE Access* 7 (2019), pp. 115134–115146.

[9] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. "The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression". In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

[10] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. "The Convolutional Tsetlin Machine". In: *arXiv preprint:1905.09688* (2019).

[11] Rohan Kumar Yadav, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. "Interpretability in Word Sense Disambiguation using Tsetlin Machine". In: *Proceedings of ICAART, Vienna, Austria.* 2021.

[12] Rohan Kumar Yadav, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. "Human-Level Interpretable Learning for Aspect-Based Sentiment Analysis". In: *Proceedings of AAAI, Vancouver, Canada.* AAAI. 2021.

[13] Adrian Phoulady, Ole-Christoffer Granmo, Saeed Rahimi Gorji, and Hady Ahmady Phoulady. "The Weighted Tsetlin Machine: Compressed Representations with Clause Weighting". In: *Ninth International Workshop on Statistical Relational AI (StarAI 2020).* 2020.

[14] Saeed Rahimi Gorji, Ole-Christoffer Granmo, Adrian Phoulady, and Morten Goodwin. "A Tsetlin Machine with Multigranular Clauses". In: *Lecture Notes in Computer Science: Proceedings of the Thirty-ninth International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI-2019).* Vol. 11927. Springer International Publishing, 2019.

[15] Saeed Gorji, Ole Christoffer Granmo, Sondre Glimsdal, Jonathan Edwards, and Morten Goodwin. "Increasing the Inference and Learning Speed of Tsetlin Machines with Clause Indexing". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer. 2020.

[16] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "Extending the Tsetlin Machine with Integer-Weighted Clauses for Increased Interpretability". In: *IEEE Access* 9 (2021), pp. 8233–8248.

[17] B John Oommen. "Stochastic Searching On the Line and Its Applications to Parameter Learning in Nonlinear Optimization". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27.4 (1997), pp. 733–739.

[18] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. "A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer. 2019, pp. 564–578.

[19] Rishad Shafik, Adrian Wheeldon, and Alex Yakovlev. "Explainability and Dependability Analysis of Learning Automata based AI Hardware". In: *IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS).* IEEE. 2020.

[20] Lei Jiao, Xuan Zhang, Ole-Christoffer Granmo, and K. Darshana Abeyrathna. "On the Convergence of Tsetlin Machines for the XOR Operator". In: *arXiv preprint arXiv:2101.02547* (2021).

[21] Xuan Zhang, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. "On the Convergence of Tsetlin Machines for the IDENTITY- and NOT Operators". In: *arXiv preprint arXiv:2007.14268* (2020).

[22] Kumpati S Narendra and Mandayam AL Thathachar. *Learning Automata: An Introduction*. Courier corporation, 2012.

[23] Michael Lvovitch Tsetlin. "On Behaviour of Finite Automata in Random Medium". In: *Avtomat. i Telemekh* 22.10 (1961), pp. 1345–1354.

[24] M A L Thathachar and P S Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004.

[25] Lei Jiao. "Markov Chain and Stationary Distribution". In: *Channel Aggregation and Fragmentation for Traffic Flows*. Springer, 2020, pp. 17–28.

[26] Tianqi Chen and Carlos Guestrin. "Xgboost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[27] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. "InterpretML: A Unified Framework for Machine Learning Interpretability". In: *arXiv preprint arXiv:1909.09223* (2019).

[28] James McDermott and Richard S Forsyth. "Diagnosing a Disorder in a Classification Benchmark". In: *Pattern Recognition Letters* 73 (2016), pp. 41–43.

Paper J

Intrusion Detection with Interpretable Rules Generated Using the Tsetlin Machine

The rapid deployment in information and communication technologies and internet-based services have made anomaly based network intrusion detection ever so important for safeguarding systems from novel attack vectors. To this date, various machine learning mechanisms have been considered to build intrusion detection systems. However, achieving an acceptable level of classification accuracy while preserving the interpretability of the classification has always been a challenge. In this paper, we propose an efficient anomaly based intrusion detection mechanism based on the Tsetlin Machine (TM). We have evaluated the proposed mechanism over the Knowledge Discovery and Data Mining 1999 (KDD'99) dataset and the experimental results demonstrate that the proposed TM based approach is capable of achieving superior classification performance in comparison to several simple Multi-Layered Artificial Neural Networks, Support Vector Machines, Decision Trees, Random Forest, and K-Nearest Neighbor machine learning algorithms while preserving the interpretability.

J.1 Introduction

Ensuring security of information systems is of significant interest due to the escalating number of attacks mounted on systems for breaching confidentiality, integrity and availability of protected data. We refer an intrusion as an unlawful attempt which is targeted on a system to steal or manipulate data for gaining various advantages and the mechanisms which are capable of detecting such intrusions are considered as intrusion detection systems. Given the upsurge in attention directed towards the security and privacy, it is fair to say that the integration of sophisticated intrusion detection mechanisms are vital for any information system.

Primarily, there are two types of intrusion detection mechanisms known as - signature based intrusion detection and anomaly based detection. A signature based detection system is equipped with a database which consists of known attack signatures. This allows the system to recognize whether a particular access has the characteristics of an intrusion via comparing it with the known signatures in the database [1, 2, 3, 4, 5]. However, such an approach is not capable of providing protection against new attack patterns given the non-existence of attack signatures. In contrast, anomaly detection deals with differentiating normal behaviors from anomalous behaviors. An anomaly detection algorithm will learn the normal behaviors and contrast the current events with normal activities to make decisions on the current events [6, 7, 8, 9, 10, 11]. Hence, anomaly detection systems are build upon classification algorithms. Thus, in comparison to signature based detection, anomaly detection is capable of identifying novel attack patterns. However the main challenge that we have in-developing efficient anomaly detection methods is that how we can effectively train the system to restrict the number of false alarms or false positives [12].

In order to develop intrusion detection systems with anomaly based detection, machine learning algorithms such as Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), Decision Trees (DTs), k-nearest neighbors (KNN) and Random Forest (RF) classifiers have been widely considered. Among these techniques ANNs has received significant attention due to its ability of classifying data with very high accuracy. However, on the contrary ANNs suffer from interpretability which is essential in understanding machine learning models as well as comprehend on why certain decisions or predictions have been made.

Interpretable machine learning refers to machine learning models that obtain transparency by providing the reasons behind their output. Linear Regression, Logistic Regression, DTs, and Decision Rules are some of the traditional interpretable machine learning approaches. However, it is important to note that the degree of interpretability of these algorithms vary significantly [13]. More importantly, accuracy of these interpretable models for more complex problems is typically low in comparison to deep learning. Deep learning inference, on the other hand, cannot easily be interpreted [14] and is thus less suitable for high-stakes application domains such as network intrusion detection. Therefore, developing machine learning algorithms for intrusion detection that can achieve a better trade-off between interpretability and accuracy continues to be an active area of research. As a solution, in this paper, we propose a novel, efficient anomaly based intrusion detection algorithm using the Tsetlin Machine (TM) [15] which outshines ANN when considering interpretability while achieving a competitive classification performance.

TM is a propositional logic based approach to interpretable machine learning and it produces decision rules similar to the branches in a DT (e.g., **if** X **satisfies** condition A **and not** condition B **then** Y = 1) [16]. However, the accuracy of the predictions made out of these interpretable rules are competitive compared to the state-of-the-art machine learning algorithms. This fact is supported by the original study of TM in [15], where Granmo discusses the competitive accuracy of TM on Binary Iris Dataset, Binary Digits Dataset, Axis and Allies board game dataset, noisy XOR dataset, and MNIST dataset compared to SVMs, DTs, RF, Naive Bayes Classifier, Logistic Regression, and simple ANNs. We can also find further evidence in [16] where the authors have shown the ability of the TM in categorizing natural language text simply based on a document word presence vector.

The TM has been expanded into many different directions to apply on many other application domains and improved in many ways to achieve better interpretable and accurate outputs, i.e, Convolutional Tsetlin Machine [17], Multi-Layered Tsetlin Machine [18], Tsetlin Machine with multi-granular clauses[19] and Regression Tsetlin Machine [20, 21].

The rest of the paper is organized as follows. Section II summarizes the most prominent research associated with machine learning techniques used for intrusion detection. We introduce the TM and its operating principle in Section III which provides the base to our intrusion detection algorithm while in Section IV, we present how the TM can be utilized for network intrusion detection. In Section V, we provide evidence on the classification accuracy of the proposed algorithm along with evidence to support our claim regarding the interpretability of the proposed approach. Finally, the paper is concluded in Section VI.

J.2 Related Work

In this section, we introduce the machine learning techniques which have been adopted in intrusion detection systems.

ANNs are inspired by the human biological system, relating to how the human brain process information and communicate them through the nervous system. ANNs consist of a set of highly interconnected elements referred to as neurons. These neurons transform a set of inputs to a set of desired outputs where the transformation is dependent upon the characteristics of the elements as well as the weights associated with the interconnections. Given that weights influences the output, it is necessary to adjust the weights and the thresholds accordingly, which is referred to as the learning process of an ANN [22, 23, 24]. ANN based intrusion detection algorithms are found in [9, 25, 26, 27].

In general, a two-class classification problem deals with finding a separation function $f(x)$, such that $y_i = f(x_i)$ given n data samples $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$. In SVM, this discriminant function formally called as a separating hyperplane. In other words, given labeled training data, the algorithm outputs an optimal hyperplane which classifies new examples. In two dimensional space, this hyperplane is a line dividing a plane in two parts where in each class lay in either side [28]. There can be many hyperplanes that classify the data, however it is necessary to find the hyperplane which maximizes the distance between the nearest data points in each side [29]. The research work found in [8, 30, 31] provides evidence for using SVM for intrusion detection.

DT is another well-known classification method used for classifying intrusions. DTs consist with a set of decision nodes and a set of leaf nodes. Each decision node represents an input parameter and it can have two or more branches depending on the attributes of the input parameter. Furthermore, each leaf node represents the classification decision (i.e intrusion or a normal behavior). In a DT based classification model, training of data relates to identifying the optimum number of branches for each decision node via learning appropriate thresholds [32, 33, 34, 35]. When a new event occurs, it is possible to recognize whether the event is an intrusion or not with the help of the input parameters

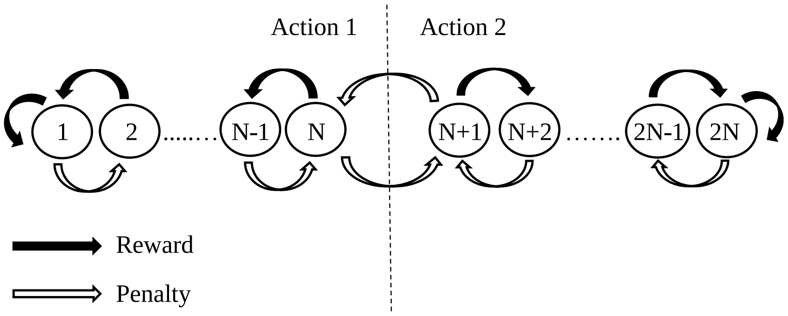


Figure J.1: Transition graph of a two-action TA.

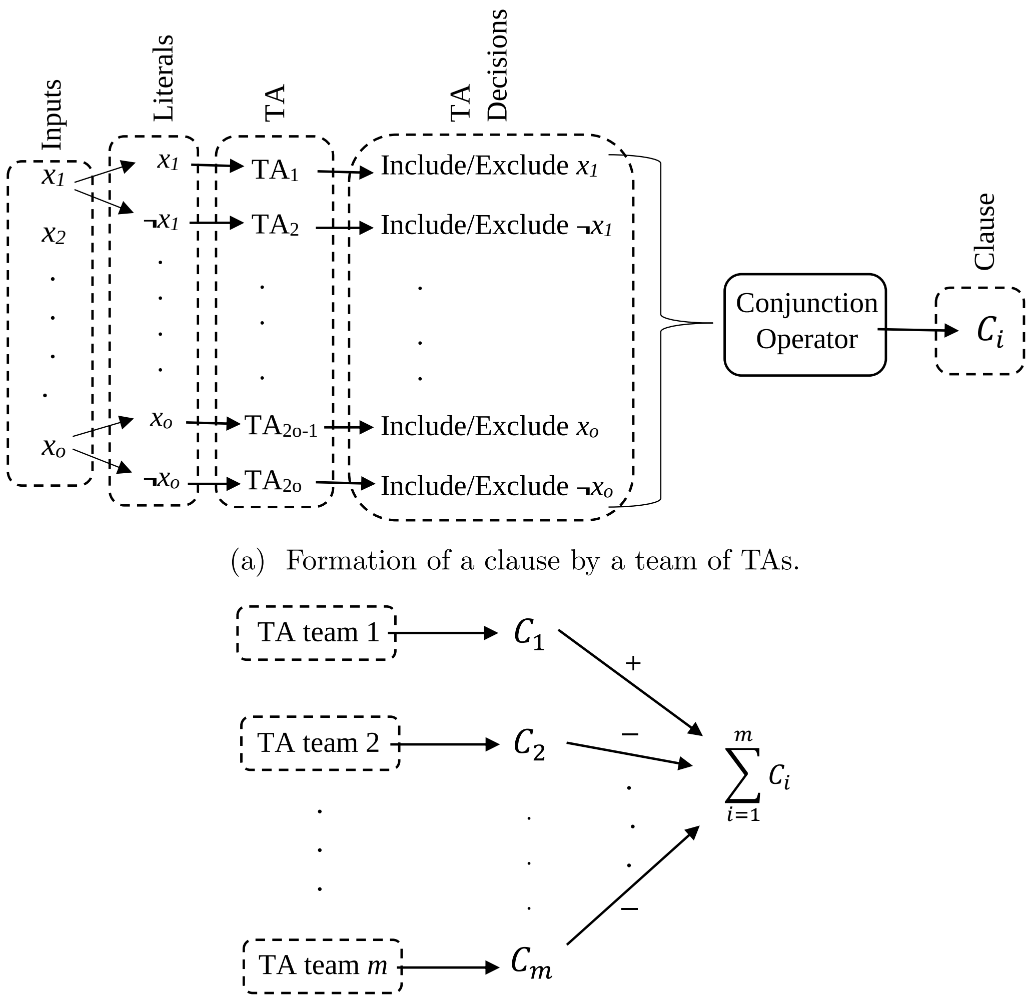
associated with the event and the decision tree established with the training data.

In addition to the above-stated machine learning algorithms, RFs, KNN classifiers and Naive Bayesian classifiers have also shown promise towards constructing intrusion detection systems and the research work in [36, 37, 38, 39] highlights the involvement of the above-mentioned algorithms.

From the related work, it is evident that many classification algorithms have shown immense potential in successfully classifying network intrusions. However, as we have elaborated previously, achieving the balance between classification accuracy and interpretability is an area where the above-stated algorithms fall behind. So, the main contribution of this paper is to address the above-stated issue by proposing an efficient, highly interpretable classification algorithm for intrusion detection using the TM which has never been applied for the purpose of detecting network intrusions. In order to test our algorithm, we have used the Knowledge Discovery and Data Mining 1999 (KDD'99) dataset given that KDD'99 is the most comprehensive and widely used dataset to compare and contrast intrusion detection algorithms.

J.3 The Tsetlin Machine (TM)

The TM is an evolving, cutting-edge classification mechanism introduced by Granmo [15] that manipulates expressions in propositional logic based on a team of Tsetlin Automata (TAs). A TA can be regarded as a fixed structure deterministic automaton which is capable of learning the optimal action among the set of allowable actions offered by an environment. An instance of a simple TA having 2N states with two actions is shown in Figure J.1. Note that the states from 1 to N maps to Action 1, whereas states from N+1 to 2N maps to Action 2. The TA interacts iteratively with its environment while in each of the iteration the TA performs the action associated with its current state (i.e. The subsequent action depends upon the current state). This, in turn, randomly triggers a reward or a penalty from the environment, according to an unknown probability distribution. A reward will reinforce the performed action via stepping into a "deeper" state (i.e. one step closer to one of the ends) whereas a penalty will drive the TA one step towards the middle state, which will ultimately weaken the performed action. Hence, the TA ultimately converges to performing the action with the highest probability of producing rewards (i.e. the optimal action) simply interacting with the environment [40]. The TM which build upon TAs operates as follows. Firstly, propositional formulas in



(a) Formation of a clause by a team of TAs.

(b) Formation of the TM.

Figure J.2: Architecture of the TM.

disjunctive normal form are used to represent the patterns. The propositional formulas are learned using the labeled data with the help of a set of TAs organized in the form of a game. Furthermore, the payoff matrix of the game has been designed so that the Nash Equilibria correspond to the optimal configurations of the TM. That makes the architecture of the TM relatively simple which ultimately helps in achieving transparency and interpretation of both learning and classification phases. Moreover, it is also worth pointing out that TM is designed for bit-wise operation, meaning that, it takes bits as input and uses fast bit manipulation operators for both learning and classification. As a result, TM has an inherent computational advantage over other classification methods.

In the following subsections, we dive into the details of the TM. We start by illustrating the architecture of the TM, thereafter the learning mechanism is presented.

J.3.1 The Tsetlin Machine Architecture

The TM can be used to tackle pattern classification problems where a class is represented with a collection of sub-patterns, each fixing certain features to distinct values. The TM is capable of unearthing these sub-patterns in a fairly simpler manner with the help of a series of clauses where a clause captures a sub-pattern by means of a Boolean statement consisting of literals combined with the logical conjunctive operator. Note that a literal simply is a propositional variable (which can be assigned with logical 0 or 1) or its negation.

To further elaborate the architecture of the TM, let us consider the following example. Suppose $\boldsymbol{X} = [x_1, x_2, x_3, \ldots, x_o]$ be a feature vector having o propositional variables with domain $\{0, 1\}$. Then the resulting pattern classification problem with two classes (class 1 and class 0) can be captured using m conjunctive clauses C_i, $1 \leq i \leq m$, in which

$$C_i = 1 \land \left(\bigwedge_{k \in I_i} x_k \right) \land \left(\bigwedge_{k \in \bar{I}_i} \neg x_k \right). \tag{J.1}$$

Note that, I_i and $\bar{I}_i$ are non-overlapping subsets of the input variable indexes, $I_i, \bar{I}_i \subseteq \{1, ...o\}$, $I_i \cap \bar{I}_i = \emptyset$. The subsets decide which of the propositional variables will be activated in the clause, and also whether they are negated or not.

We have illustrated the architecture of the TM in Figure J.2. The sub-figure Figure J.2a illustrates how a team of TAs forms a clause that processes the input features while the sub-figure Figure J.2b depicts how the TM is formed by a set of TA teams. Now, let us present the details of the above-mentioned to key phases of the TM - formation of a clause by a team of TAs and the formation of the TM with a set of TA teams.

J.3.1.1 Formation of a clause by a team of TAs

Let us continue with the classification example presented in Section IIIA, in which we considered an input with o propositional variables. Thus, for each of the clause C_i, as shown in Figure J.2a, the TM takes the o propositional variables $x_1, x_2, x_3, \ldots, x_o$ as the input. Note that, for each propositional variable x_k, there are two literals, the variable itself and its negation $\neg x_k$. Then, each literal is assigned to a unique TA which takes the decision whether to *include* or *exclude* its assigned literal in the given clause. Therefore, a clause requires $2o$ number of TAs to serve the n input variables. We call this collective arrangement of TAs as a TA team. Then by considering only the literals that TAs have decided to *include* in the clause, the TA team composes a conjunction of the literals. The conjunction will output a logical 1 if and only if the included literals evaluate to logical 1, otherwise, the clause will output a logical 0.

J.3.1.2 Formation of the TM with a set of TA teams

In the example we assumed that we need m clauses to recognize the sub-patterns in the classification problem. Hence, the TM should have m clauses with each clause is associated with a distinct TA team as depicted in Figure J.2b. The output of the TM is driven by a voting scheme where each clause casts its vote. This means that the m clauses jointly compute the output of the TM. Clauses with odd indexes are assigned positive polarity (+) and clauses with even indexes are assigned negative polarity (−). Note that the clauses with positive polarity cast their votes to favor the decision that the input belongs to the class 1, whereas the clauses with negative polarity vote for the input

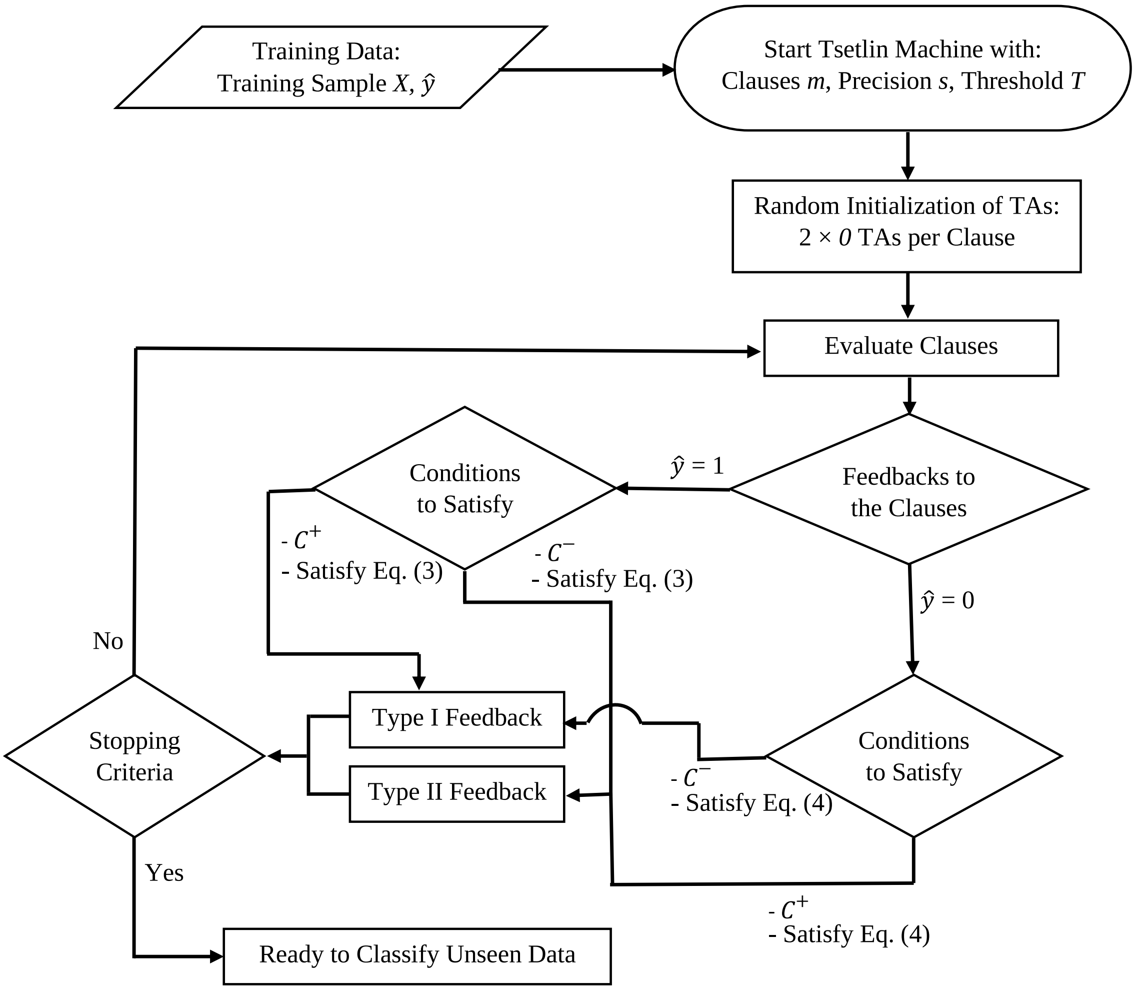


Figure J.3: The learning process of the TM.

belonging to class 0. Then, the summation operator aggregates the votes by subtracting the number of negative votes from the number of positive votes to derive the output of the TM by considering the majority.

J.3.2 The Tsetlin Machine Learning Mechanism

The learning mechanism of the TM is organized as a game played among the TAs. The Nash Equilibria of the game corresponds to the goal state of the TA, providing the final classifier. In the worst case, a single action of any TA has the power to disrupt the whole game. Therefore, the TAs must be guided carefully, so that we can realize optimal pattern recognition.

In order to achieve the aforementioned goal, the learning process of the TM is formed around two kinds of feedback - Type I Feedback and Type II Feedback. The Reward, Inaction, and Penalty probabilities associated with the above-stated two feedback types are summarized in Table J.1. These probabilities are determined based on the clause output (logical 1 or 0), the literal value (logical 1 or 0), and the current action of the TA (include or exclude). Rewards and Penalties are fed to the TA as normal whereas an Inaction means that the state of the TA remains unchanged.

We have illustrated the learning process of the TM highlighting the involvement of the

Table J.1: Tabulation of Reward, Inaction and Penalty probabilities associated with Type I Feedback and Type II Feedback.

Feedback Type			I				II			
Clause Output			1		0		1		0	
Literal Value			1	0	1	0	1	0	1	0
Current State	Include (Probability)	Reward	(s-1)/s	NA	0	0	0	NA	0	0
		Inaction	1/s	NA	(s-1)/s	(s-1)/s	1	NA	1	1
		Penalty	0	NA	1/s	1/s	0	NA	0	0
	Exclude (Probability)	Reward	0	1/s	1/s	1/s	0	0	0	0
		Inaction	1/s	(s-1)/s	(s-1)/s	(s-1)/s	1	0	1	1
		Penalty	(s-1)/s	0	0	0	0	1	0	0

*s is the precision and controls the granularity of the sub-patterns in [15]

two feedback mechanisms with the help of a flow diagram in Figure J.3. Furthermore, in the following, We explain how each of the feedback mechanism is activated and the roles they play upon activation in detail.

J.3.2.1 Type I Feedback

As evident from Figure J.3, Type I Feedback will only be activated if and only if the labeled training output of the TM, $\hat{y} = 1$. To explain how Type I Feedback impacts the learning process, we need to consider two cases - the output of the target clause is 1 and the case in which the output of the target clause is 0. The roles that Type I Feedback will play subjected to the above-stated two cases are summarized below.
Case 1: The output of the target clause is 1. Given that $\hat{y} = 1$, this case represents a true positive output. In this scenario, Type I Feedback has three roles:

- It reinforces the true positive output by assigning a large reward probability $\frac{(s-1)}{s}$ to the action of including literals that evaluate to 1, and thus contributing to the result of the clause output being 1.
- Conversely, exclude actions are penalized with the same magnitude under these conditions. This is to tighten the clause, since it would still produce an output of 1 even when the literal considered is included instead.
- Furthermore, if the value of the literal is 0, excluding the literal is the way to go, and therefore exclude actions are rewarded with a probability $\frac{1}{s}$.

Case 2: The output of the target clause is 0. Note that this case represents a false negative output given that $\hat{y} = 1$. In this scenario, Type I Feedback has the following roles:

- Type I Feedback systematically penalizes include actions with probability $\frac{1}{s}$. This is due to the fact that excluding literals is the only way to invert the output of a clause that outputs a 0.

- With an exclude action, this will be rewarded with a probability $\frac{1}{s}$, given that reinforcing exclude actions will sooner or later invert the output of the clause to 1.

Thus, eventually, Type I Feedback is capable of combating false negative outputs while encouraging the true positive outputs.

J.3.2.2 Type II Feedback

In contrast to Type I Feedback, Type II Feedback is activated when the actual output $\hat{y} = 0$. The idea behind including this type of a feedback mechanism is to eliminate false positive outputs. This means that the clause erroneously evaluates to 1, when the clause output should have been 0. In such a scenario, Type II Feedback is triggered. When Type II Feedback is repeated it will ultimately force the offending clause to evaluate to 0, by simply including a literal that has the value 0 into the clause (i.e. Such an inclusion makes the conjunction of literals evaluate to 0). This is accomplished by penalizing exclude actions with probability 1, for literals that evaluate to 0.

In brief, Type I Feedback reinforces true positive outputs, while simultaneously reducing false negative outputs. To improve the classification accuracy, these dynamics are supported by Type II Feedback, through systematically reducing false positive outputs.

J.3.2.3 The Clause Feedback Activation Function

In addition to the above-stated two feedback mechanisms, an additional function is introduced in [15] with the intention of effectively allocating the sparse pattern representation resources provided by the clauses. This is achieved by introducing a target value T for the number of clauses voting from a specific pattern which will helps in reducing the frequency of feedback for a specific pattern, as the number of votes approaches T. In all brevity, the feedback activation function is basically an activation probability controlled by a threshold T. Thus, the probability of activating Type I Feedback for a specific clause is given by,

$$\frac{T - max(-T, min(T, \sum_{i=1}^{m} C_i))}{2T} \tag{J.2}$$

whereas for Type II Feedback, the probability is given by,

$$\frac{T + max(-T, min(T, \sum_{i=1}^{m} C_i))}{2T}. \tag{J.3}$$

From (J.2), it is easily observable that the activation probability decreases as the number of votes approaches T, and finally when T is reached, the probability reaches 0. Therefore, Type I feedback will not be activated, when enough clauses are producing the correct number of votes. This in turn “freezes” the affected clauses purely because TAs will no longer change the state. Furthermore, this will result in freeing out other clauses to seek other sub-patterns, given that the “frozen” pattern is no longer attractive for the TA. The same rationale holds for Type II feedback which is evident from (J.3). In this way, the pattern representation resources can be allocated more effectively with the adoption of the Clause Feedback Activation Function.

J.4 Empirical Evaluation

In this section, we present how the TM is utilized to differentiate network anomalies from normal behaviors. We start the section by introducing the dataset that we have used to test the proposed approach, then we explain how the dataset is pre-processed and thereafter the adopted experimental setup is presented.

J.4.1 The Dataset

In this study, we used the KDD'99 dataset to demonstrate the feasibility of the TM to be used for the purpose of detecting network intrusions[1]. The KDD'99 dataset has been derived from DARPA datasets (1998) generated in MIT Lincoln Laboratories. It consists of 41 feature attributes. The intrusions in the dataset have been grouped into four different types - Denial of Service (DoS), Probe, Remote to Local (R2L), and User to Root (U2R).

J.4.2 Data Pre-processing

In this study, we are interested in classifying whether a particular access represents a network intrusion or a normal behavior based on the feature attributes instead of identifying the type of the intrusion. Hence, we update the class labels so that the normal data samples are represented by 0s while data samples representing intrusions are denoted with 1s.

We observed that only few features from the total of 41 feature attributes are correlating with the class labels. For instance, 5 out of 41 feature attributes are constant throughout the dataset. Therefore, these features will not contribute towards classifying the patterns into the two classes separately. Accordingly, after a secondary correlation analysis, we selected the most influencing 5 feature attributes to classify the samples from the dataset. These features, starting from the most correlated feature, are *logged_in*, *count*, *dst_host_count*, *protocol_type*, and *dst_bytes*.

It is important to note that the features - *logged_in* and *protocol_type* are *symbolic* features whereas the remaining 3 features are *continuous* features. As explained in Section III, the input to the TM must be in binary form. Therefore, to feed the identified features into the TM, firstly, the symbolic features must be integer coded so that they can be binarized along with the continuous features. To achieve binarization, we used the thresholding method proposed in [41]. For instance, the feature *protocol_type* deals with three protocol types: tcp, udp, and icmp. Hence, the feature *protocol_type* can be binarized as shown in Table J.2. After binarizing all features, the resulting dataset represents 1 original feature vector (consisting of 5 feature attributes) with a vector having 654 binary feature attributes. Table J.3 exhibits which of the features in the resulting feature set belong to the originally selected 5 features.

[1] The dataset is available to download at `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

Table J.2: Binarizing the feature *protocol_type* in the KDD'99 dataset.

Protocol Type	Integer Code	Thresholds		
		≤0	≤1	≤2
icmp	0	1	1	1
tcp	1	0	1	1
udp	2	0	0	1

Table J.3: Features before and after binarization.

Original Feature	Binarized Features
logged_in	From feature 1 to 2
count	From feature 3 to 80
dst_host_count	From feature 81 to 258
protocol_type	From feature 259 to 261
dst_bytes	From feature 262 to 654

J.4.3 The TM on KDD'99 Dataset

We have utilized several TM setups by varying the number of clauses (m = 2, 10, 100, 500, 2000, 8000, 15000, 20000) to analyze the classification capability of the TM with the help of the pre-processed dataset[2]. Since there are 654 binary features, we require 1308 TAs to form a clause. Each TA is given 100 states per action. Half of the allocated clauses will recognize the patterns associated with class 1 (intrusion) and the other half of the clauses will recognize patterns related to class 0 (normal). The other two hyper parameters, T and s are perceived using a binary search for distinct TM setups.

J.5 Results and Discussion

This section is dedicated to analyze the performance of our TM based intrusion detection algorithm with the help of the KDD'99 dataset. Firstly, we evaluate the classification accuracy of the TM by considering the eight TM setups introduced in Section IV to observe the influence of the number of clauses in the TM on the classification accuracy. Thereafter, the classification accuracy of the TM is compared with several other state-of-the-art machine learning algorithms. Towards the end of the section, we provide evidence to support the claim on the interpretability of our machine learning algorithm by generating interpretable rules from the TM.

J.5.1 Classification Accuracy

In the experiments, we used 80% of the data samples to train the TM while the rest is utilized to measure the prediction performance. The performance was measured in-terms of the parameters - precision, recall, F1-score, accuracy, and specificity. Considering the dataset that we have used for the experiments, we consider that F1-score is the best

[2] The source code of the TM can be found at `https://github.com/cair/pyTsetlinMachine`

Table J.4: Performance of different TM Setups.

	Number of Clauses (m)							
	2	10	100	500	2000	8000	15000	20000
Precision	0.7733	0.7843	0.8285	0.8709	0.9178	0.9575	0.9809	0.9828
Recall	0.9978	1.0000	0.9923	0.9916	0.9806	0.9729	0.9803	0.9827
F1-Score	0.8701	0.8788	0.9002	0.9252	0.9456	0.9636	0.9806	0.9827
Accuracy	0.7716	0.7866	0.8275	0.8734	0.9113	0.9419	0.9703	0.9736
Specificity	0.0492	0.0651	0.2928	0.4685	0.6917	0.8392	0.9371	0.9442

performance metric given the imbalance of the data samples belonging to the two classes in the KDD'99 dataset. In addition, we also consider specificity as another important metric considering our application of interest. This is because,

$$\text{specificity} = 1 - \text{false positive rate}$$

and keeping the false positives to a minimum is one of the main obstacles that we need to overcome when developing anomaly based intrusion detection systems.

J.5.1.1 TM classification accuracy with number of clauses

As we have explained in Section IV, we formulated eight TM setups by varying the number of clauses to observe the behavior of the above-stated performance parameters. The obtained results are tabulated in Table J.4. Note that, we conducted each experiment 50 times with random separations of training and testing data; thus the tabulated results correspond to average values.

From Table J.4, it is evident that precision, F1-score, accuracy, and specificity improve with the number of clauses. Hence, the best performance for these 4 metrics is achieved when the TM is configured with 20000 clauses. At this parameter setting, the TM is capable of achieving performance levels of 0.9828, 0.9827, 0.9736, 0.9442 for precision, F1-score, accuracy, and specificity respectively. On the other hand, recall reaches its maximum value of 1.0000 when the number of clauses is 10 and then fluctuates around 0.9800 when the number of clauses is further increased.

As we have explained previously, specificity is crucial for intrusion detection systems. When the TM is configured with 20000 clauses, we have shown that it is possible to achieve a specificity of 0.9442. This means that, with 20000 clauses, the TM is capable of identifying nearly 95 out of 100 normal behaviors correctly. Therefore, a TM with this number of clauses would raise about 5 false alarms. However, according to Table J.4, it is evident that specificity can further be increased by increasing the number of clauses in the TM; but with the expense of a higher computational cost.

J.5.1.2 Performance Comparison

To compare the performance of the proposed TM based intrusion detection algorithm, we selected five state-of-the-art machine learning algorithms - ANNs, SVM, DT, RF,

Table J.5: Performance Comparison between TM and other Machine Learning Algorithms.

	Machine Learning Algorithm							
	ANN-1	ANN-2	ANN-3	SVM	DT	RF	KNN	TM
Precision	0.9656	0.9660	0.9652	1.0000	0.9796	1.0000	0.9797	0.9828
Recall	0.9967	0.9953	0.9977	0.9500	0.9830	0.9472	0.9784	0.9827
F1-Score	0.9808	0.9803	0.9812	0.9743	0.9812	0.9729	0.9788	0.9827
Accuracy	0.9702	0.9694	0.9707	0.9618	0.9711	0.9593	0.9678	0.9736
Specificity	0.8838	0.8854	0.8834	1.0000	0.9311	1.0000	0.9314	0.9442

and KNN. For comprehensiveness, three ANN architectures with the following parameter settings were used.

- ANN-1: 1 hidden layer with 20 neurons
- ANN-2: 3 hidden layers with 20, 150, and 100 neurons
- ANN-3: 5 hidden layers with 20, 200, 150, 100, and 50 neurons

All the other machine learning algorithms were also configured with their best parameter settings found after a secondary parameter exploration to ensure a fair comparison. In Table J.5, we have tabulated the resulting average performance values obtained for the considered performance metrics (i.e. precision, recall, F1-score, accuracy and specificity) after executing 50 runs of each above-stated machine learning algorithm on the KDD'99 dataset along with the performance results of the proposed TM based classification algorithm when configured with 20000 clauses.

From Table J.5, we can observe that, out of the three ANN setups, the best F1-score is realized by ANN-3 (0.9812) (i.e. F1-score increases with the number of hidden parameters in an ANN). However, when the number of hidden parameters in an ANN increase, the specificity fluctuates. It is observable that the specificity at the peak F1-Score for ANNs is merely 0.8834. The other three considered performance metrics, i.e., precision, recall, and accuracy also fluctuate when the number of hidden parameters in ANNs are increased. Hence, selecting the best ANN for intrusion detection from the considered ANN architectures is a strenuous process.

We can also notice that the DTs are capable of outperforming SVM, RF, and KNN algorithms by achieving an F1-score similar to the one achieved with ANN-3. However, DTs achieve this performance level with better precision, accuracy, and specificity in comparison to ANN-3. It is also important to mention that SVM and RF acquire the best possible specificity (1.0000). Therefore, these two algorithms (SVM and RF) do not make false alarms while DT and KNN make nearly 6 false alarms for 100 normal behaviors.

The proposed TM based algorithm with 20000 clauses outperforms all the considered state-of-the-art machine learning algorithms in terms of the F1-score and accuracy. It is nearly impossible for the SVM, DT, RF, and KNN algorithms to further improve their F1-score by changing the hyper-parameters. Despite the mixed performance characteristics

Table J.6: Lower and upper bounds of each feature in each clause when the TM is configured with 2 clauses.

Feature	1		2		3		4		5	
Lower Bound (L) / Upper Bound (U)	L	U	L	U	L	U	L	U	L	U
Clause 1	-	≤ 0	-	-	-	-	-	-	-	-
Clause 2	$0 <$	-	-	-	-	-	-	-	-	-

achieved by the considered ANNs, one might argue that the performance of ANNs can be simply improved by further increasing the number of hidden parameters. However, this incidentally equals to the scenario of increasing the number of clauses in a TM which also show promise in improving classification accuracy as evident from Table J.4.

J.5.2 Interpretability

In this subsection, we intend to provide evidence on the interpretability of the proposed TM based approach through generating rules from the TM outputs.

To understand the process of generating rules from the TM outputs, first of all, it is important to understand the properties of the utilized binarization procedure. So, let us revisit the binarization example given in Table J.2. In this table, threshold ≤ 0 represents the protocol type *icmp* while the threshold ≤ 1 and threshold ≤ 2 appear for the protocol type *tcp* and *udp*, respectively. Thus, in a selected clause, if the only threshold included by its corresponding TA is ≤ 1, then both protocol types *icmp* and *tcp* will be selected to build the classifier rule (e.g. ... **AND** (*icmp* **OR** *tcp*) **AND** ...). Furthermore, if two thresholds ≤ 1 and ≤ 2 are included in the clause by their corresponding TAs, then the two protocol types - *icmp* and *tcp* will again be selected to build the rule due to the fact that **AND** operation of ≤ 1 and ≤ 2 threshold columns in Table J.2 yields the threshold column ≤ 1. Nevertheless, if the negation of the threshold ≤ 0 is the only included threshold in the clause, then it reveals that **NOT** *icmp* should be selected to build the rule (e.g. ... **AND NOT** *icmp* **AND** ...). In other words, given that the negation of ≤ 0 is equivalent to $0 <$, the clause activates for both the other threshold values, i.e., ≤ 1 and ≤ 2 (e.g. ... **AND** (*tcp* **OR** *udp*) **AND** ...).

For continuous features, included original thresholds (i.e. non negated thresholds) define the upper bound of the continuous feature in the rule. As an example, let us assume that Table J.2 represents the binarization of a continuous feature instead of the categorical feature. In this situation, if both ≤ 1 and ≤ 2 thresholds are included in the clause, then only ≤ 1 threshold should be added to the classifier rule. The reason being the clauses compute the conjunction of the included literals to decide the clause outputs (**AND** of ≤ 1 and ≤ 2 or any other higher threshold is ≤ 1). Likewise, the lower bound of the continuous value to be included in the rule can also be computed from the included negated thresholds.

Taking the above properties of the binarization procedure into account, we tabulated the clause outputs in Table J.6, when the TM is configured with merely 2 clauses. As one can notice, the TM has considered only the most correlated feature, i.e., *logged_in* to

Table J.7: Lower and upper bounds of each feature in each clause when the TM is configured with 10 clauses.

Feature	1		2		3		4		5	
Lower Bound (L) / Upper Bound (U)	L	U	L	U	L	U	L	U	L	U
Clause 1	-	≤ 0	-	-	-	-	-	-	-	-
Clause 2	-	-	-	≤ 16	-	-	-	-	-	-
Clause 3	-	≤ 0	-	-	-	-	-	-	-	-
Clause 4	$0 <$	-	-	-	-	-	-	-	-	-
Clause 5	-	≤ 0	-	-	-	-	-	-	-	-
Clause 6	-	-	-	≤ 18	-	-	-	-	-	-
Clause 7	-	≤ 0	-	-	-	-	-	-	-	-
Clause 8	-	≤ 1	-	-	-	-	-	-	-	-
Clause 9	-	≤ 0	-	-	-	-	-	-	-	-
Clause 10	-	≤ 1	-	-	-	-	-	-	-	-

make the classifications. Note that the *logged_in* feature can have one of the two values 1 or 0 where 1 represents a successful login while 0 represents a login failure. Hence, clause 1, which is voting for class 1, says *logged_in* should be 0 whereas the clause 2, which votes for class 0, says *logged_in* should be 1. With the help of the above information on clause outputs, we can construct the following simple rule.

$$\text{Outcome} = \begin{cases} \text{Not an attack} & \textbf{if } \text{Successfully logged in} \\ \text{Attack} & \textbf{otherwise}. \end{cases} \tag{J.4}$$

Now, let us consider the case in which the TM is equipped with 10 clauses. The clause outputs at the end of the training phase are summarized in Table J.7. Out of these 10 clauses, clauses with an odd index vote for class 1 (an attack) while the clauses with an even index vote for the class 0 (not an attack). All clauses with an odd index have recognized the same sub-pattern learned by the clause 1 in Table J.6 (*logged_in* is unsuccessful). Therefore, when *logged_in* is 0, class 1 receives 5 votes. However, when *logged_in* is 0, class 0 also receives 2 easy votes (without needing to satisfy other conditions) from clause 8 and clause 10. With some conditions associated with feature 2, clause 2 and clause 4 could also add votes to the class 0 when *logged_in* is 0. However, the maximum possible votes for class 0 is inadequate to beat class 1. Therefore, regardless of the other conditions, when *logged_in* is 0, that can be categorized as 'an attack'.

On the other hand, when *logged_in* is 1, class 0 receives 3 easy votes and 2 other votes from clause 2 and clause 6, if they satisfy the conditions associated with the feature 2. Class 1 does not receive votes when *logged_in* is 1. Therefore, regardless of the other conditions, when *logged_in* is 1, that can be categorized as 'not an attack'. Hence, for this considered example, the same rule built in (J.4) can be regenerated.

From the above analysis, we can observe that, for the considered example, both the TM configurations (i.e. TM with 2 clauses and 10 clauses) have ended up with the same rule given in (J.4). However, intuitively we know that the probability of getting an error with TM configured with 2 clauses is higher than the TM configured with 10 clauses.

Furthermore, as we can see from Table J.6, the TM configured with 2 clauses does not have enough resources to search for other sub-patterns. It has optimally employed its clauses to recognize the sub-pattern of the most correlated feature, *logged_in*. In contrast, the TM configured with 10 clauses, as evident from Table J.7, starts searching for other sub-patterns of the two classes (e.g. additional sub-patterns associated with feature 2, which is the second most correlated feature). Therefore, when the number of clauses in the TM is increased, it is certain that the TM will learn additional sub-pattern with higher probability and thereby yields better accuracy as we can observe from Table J.4.

Nevertheless, even if we discard the duplicate clauses while generating the rules, the increase in the number of clauses will also increase the number of literals in the rules. Furthermore, when the TM is configured with a large number of clauses, it might make the classifier less interpretable. Hence, it is important to achieve a trade-off between the number of literals in the rule and the prediction accuracy. In Figure J.4, we plot the average prediction accuracy against the average number of literals in unique clauses.

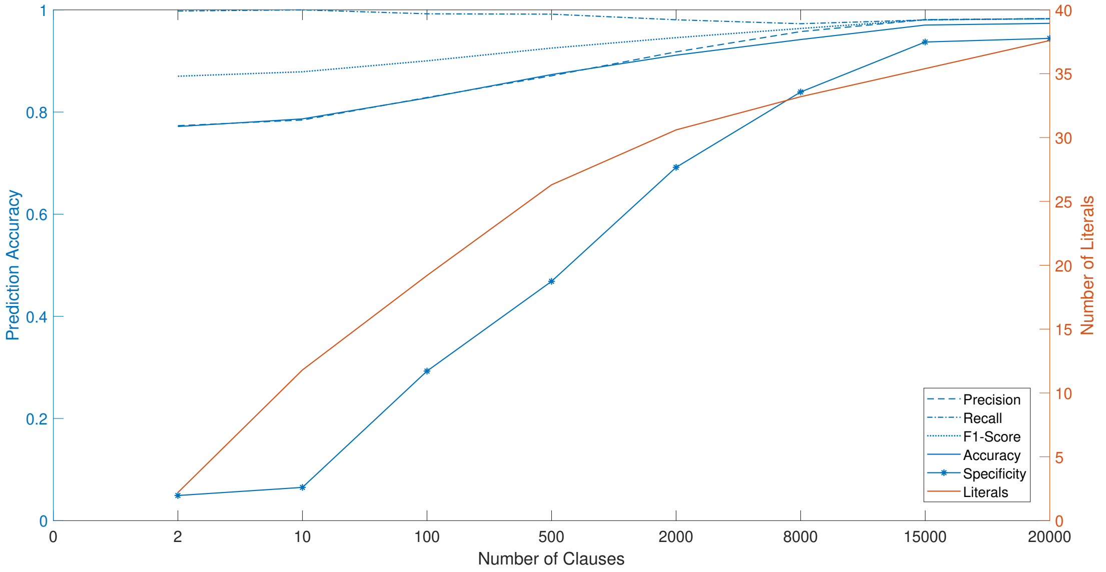


Figure J.4: Variation of the prediction accuracy and the number of literals against the number of clauses .

From Figure J.4, we can clearly observe that the specificity benefits greatly from having a large number of literals in rules. However, the increase of precision, F1-Score, and accuracy compared to the growth of the number of literals is trivial while recall remains steady. Furthermore, depending on the classification objectives or requirements, the number of clauses can be decided upon appropriately. For instance, one might satisfy with a F1-score of 87% when the classifier is easily interpretable (less number of literals) as in (J.4). On the other hand, one might need higher accuracy in the expense of interpretability; which he or she can achieve by allocating a large number of clauses to the TM.

J.6 Conclusion

In this paper, we propose an efficient anomaly based network intrusion detection mechanism using the Tsetlin Machine (TM), to address the problem of achieving the necessary balance between classification accuracy and interpretability which has a significant importance for critical applications such as intrusion detection. We evaluated the proposed TM based mechanism using the KDD'99 dataset, given that it is widely considered as the benchmark dataset for testing intrusion detection algorithms. The classification results show us that the proposed method is capable in outperforming the state-of-the-art interpretable machine-learning algorithms while achieving a competitive classification performance in relation to ANNs. Furthermore, we have presented how classification rules can be derived from the TM output, which supports our claim on interpretability of the proposed approach. In our future work, we intend to investigate the possibility of using the TM to identify the types of the intrusions instead of just classifying intrusions from normal behaviors. In addition, we will also explore the possibilities of using the Integer Weighted Tsetlin Machine [42], which includes fewer literals in classifier rules compared to the regular TM, to detect the anomalies from normal behavior.

Bibliography

[1] B.Santos Kumar, Sk.Dawood Baba T.Chandra Sekhara Phani Raju M.Ratnakar, and N.Sudhakar. "Intrusion Detection System-Types and Prevention". In: *International Journal of Computer Science and Information Technologies* 4.1 (2013), pp. 77–82.

[2] F. Erlacher and F. Dressler. "FIXIDS: A High-Speed Signature-based Flow Intrusion Detection System". In: *Proceedings of IEEE/IFIP Network Operations and Management Symposium.* 2018, pp. 1–8.

[3] A. H. Almutairi and N. T. Abdelmajeed. "Innovative Signature based Intrusion Detection System: Parallel Processing and Minimized Database". In: *Proceedings of International Conference on the Frontiers and Advances in Data Science (FADS).* 2017, pp. 114–119.

[4] Y. Zhang, F. Gao, Y. Guo, and X. Liu. "Research on Intrusion Detection Approach based on Signature Generation". In: *Proceedings of 2nd Pacific-Asia Conference on Circuits, Communications and System.* 2010, pp. 236–240.

[5] *What is IDS. URL:* ***https://searchsecurity.techtarget.com/definition/intrusion-detection-system.*** Accessed July 04, 2018. URL: `https://searchsecurity.techtarget.com/definition/intrusion-detection-system`.

[6] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. "Network Anomaly Detection: Methods, Systems and Tools". In: *IEEE Communications Surveys Tutorials* 16.1 (2014), pp. 303–336.

[7] Fangjun Kuang, Weihong Xu, and Siyang Zhang. "A Novel Hybrid KPCA and SVM with GA Model for Intrusion Detection". In: *Applied Soft Computing* 18 (2014), pp. 178–184.

[8] R. R. Reddy, Y. Ramadevi, and K. V. N. Sunitha. "Effective Discriminant Function for Intrusion Detection using SVM". In: *Proceedings of International Conference on Advances in Computing, Communications and Informatics (ICACCI).* 2016, pp. 1148–1153.

[9] B. Ingre and A. Yadav. "Performance Analysis of NSL-KDD Dataset using ANN". In: *Proceedings of International Conference on Signal Processing and Communication Engineering Systems.* 2015, pp. 92–96.

[10] Nabila Farnaaz and MA Jabbar. "Random forest modeling for network intrusion detection system". In: *Procedia Computer Science* 89 (2016), pp. 213–217.

[11] J. Zhang, M. Zulkernine, and A. Haque. "Random-Forests-Based Network Intrusion Detection Systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.5 (2008), pp. 649–659.

[12] *Types of IDS. URL:* ***http://www.omnisecu.com/security/infrastructure-and-email-security/types-of-intrusion-detection-systems.php***. Accessed July 04, 2018. URL: `http://www.omnisecu.com/security/infrastructure-and-email-security/types-of-intrusion-detection-systems.php`.

[13] Christoph Molnar. *Interpretable Machine Learning*. Lulu. com, 2019.

[14] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. "Deep Learning for Healthcare: Review, Opportunities and Challenges". In: *Briefings in bioinformatics* 19.6 (2018), pp. 1236–1246.

[15] Ole-Christoffer Granmo. "The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic". In: *arXiv preprint arXiv:1804.01508* (2018).

[16] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. "Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization With Medical Applications". In: *IEEE Access* 7 (2019), pp. 115134–115146.

[17] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. "The Convolutional Tsetlin Machine". In: *arXiv preprint:1905.09688* (2019).

[18] Ole-Christoffer Granmo. "The Multi-Layered Tsetlin Machine". In: *Preparation* (2020).

[19] Saeed Rahimi Gorji, Ole-Christoffer Granmo, Adrian Phoulady, and Morten Goodwin. "A Tsetlin Machine with Multigranular Clauses". In: *Lecture Notes in Computer Science: Proceedings of the Thirty-ninth International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI-2019)*. Vol. 11927. Springer International Publishing, 2019.

[20] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Lei Jiao, and Morten Goodwin. "The regression Tsetlin Machine: A Tsetlin Machine for Continuous Output Problems". In: *EPIA Conference on Artificial Intelligence*. Springer. 2019, pp. 268–280.

[21] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. "The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression". In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

[22] D. Niu, Q. Wanq, and J. Li. "Short Term Load Forecasting Model Using Support Vector Machine based on Artificial Neural Network". In: *Proceedings of International Conference on Machine Learning and Cybernetics*. 2005, pp. 4260–4265.

[23] *Using Neural Nets to Recognize Handwritten Digits. URL:* ***http://neuralnetworksanddeeplear*** Accessed July 10, 2018. URL: `http://neuralnetworksanddeeplearning.com/chap1.html`.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[25] H. Deng and Y. Wang. “An Artificial-Neural-Network-based Multiple Classifiers Intrusion Detection System”. In: *Proceedings of International Conference on Wavelet Analysis and Pattern Recognition.* 2007, pp. 683–686.

[26] M. U. ÖNEY and S. PEKER. “The Use of Artificial Neural Networks in Network Intrusion Detection: A Systematic Review”. In: *Proceedings of International Conference on Artificial Intelligence and Data Processing (IDAP).* 2018, pp. 1–6.

[27] B. Subba, S. Biswas, and S. Karmakar. “A Neural Network based system for Intrusion Detection and Attack Classification”. In: *Proceedings of 22nd National Conference on Communication (NCC).* 2016, pp. 1–6.

[28] *Chapter 2-SVM Theory. URL:* ***https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72***. Accessed July 04, 2018. URL: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

[29] *SVM-Understanding the Math. URL:* ***https://www.svm-tutorial.com/2014/11/svm-understanding-math-part-1/***. Accessed Julyl 04, 2018. URL: https://www.svm-tutorial.com/2014/11/svm-understanding-math-part-1/.

[30] B. Senthilnayaki, K. Venkatalakshmi, and A. Kannan. “Intrusion Detection Using Optimal Genetic Feature Selection and SVM based Classifier”. In: *Proceedings of 3rd International Conference on Signal Processing, Communication and Networking (ICSCN).* 2015, pp. 1–4.

[31] X. Tang, S. X. -. Tan, and H. Chen. “SVM Based Intrusion Detection Using Nonlinear Scaling Scheme”. In: *Proceedings of 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT).* 2018, pp. 1–4.

[32] H. Elaidi, Y. Elhaddar, Z. Benabbou, and H. Abbar. “An Idea of a Clustering Algorithm Using Support Vector Machines based on Binary Decision Tree”. In: *2018 International Conference on Intelligent Systems and Computer Vision (ISCV).* 2018, pp. 1–5.

[33] M. A. Jabbar and S. Samreen. “Intelligent Network Intrusion Detection Using Alternating Decision Trees”. In: *Proceedings of International Conference on Circuits, Controls, Communications and Computing (I4C).* 2016, pp. 1–6.

[34] J. Wang, Q. Yang, and D. Ren. “An Intrusion Detection Algorithm Based on Decision Tree Technology”. In: *Proceedings of Asia-Pacific Conference on Information Processing.* 2009, pp. 333–335.

[35] S. Sahu and B. M. Mehtre. “Network Intrusion Detection System Using J48 Decision Tree”. In: *Proceedings of International Conference on Advances in Computing, Communications and Informatics (ICACCI).* 2015, pp. 2023–2026.

[36] J. Zhang, M. Zulkernine, and A. Haque. “Random-Forests-Based Network Intrusion Detection Systems”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.5 (2008), pp. 649–659.

[37] A. Tesfahun and D. L. Bhaskari. “Intrusion Detection Using Random Forests Classifier with SMOTE and Feature Reduction”. In: *Proceedings of International Conference on Cloud Ubiquitous Computing Emerging Technologies.* 2013, pp. 127–132.

[38] H. Xu, C. Fang, Q. Cao, C. Fu, L. Yan, and S. Wei. “Application of a Distance-weighted KNN Algorithm Improved by Moth-Flame Optimization in Network Intrusion Detection”. In: *Proceedings of 4th IEEE International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS).* 2018, pp. 166–170.

[39] X. Han, L. Xu, M. Ren, and W. Gu. “A Naive Bayesian Network Intrusion Detection Algorithm Based on Principal Component Analysis”. In: *Proceedings of 7th International Conference on Information Technology in Medicine and Education (ITME).* 2015, pp. 325–328.

[40] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. “A Novel Tsetlin Automata Scheme to Forecast Dengue Outbreaks in the Philippines”. In: *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI).* IEEE. 2018, pp. 680–685.

[41] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. “A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks”. In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer. 2019, pp. 564–578.

[42] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. “Extending the Tsetlin Machine with Integer-Weighted Clauses for Increased Interpretability”. In: *IEEE Access* 9 (2021), pp. 8233–8248.

Paper K

Public Transport Passenger Count Forecasting in Pandemic Scenarios Using Regression Tsetlin Machine. Case Study of Agder, Norway

Challenged by the effects of the COVID-19 pandemic, *public transport is suffering from low ridership and staggering economic losses. One of the factors which triggered such losses was the lack of preparedness among governments and public transport providers. The present paper explores the use of a novel machine learning algorithm, namely* Regression Tsetlin Machine, *in using historical passenger transport data from the current COVID-19 pandemic and pre-pandemic period to forecast pandemic-scenarios for public transport patronage variations. Results show that the* Regression Tsetlin Machine *has the best accuracy of forecasts compared to four other models usually employed in the field.*

K.1 Introduction

K.1.1 The world-wide situation

The spread of the COVID-19 virus has shed new light on the way urban societies react to pandemic situations. In this new daily reality, public transport (PT) has an ambivalent character: maintaining the continuation of critical services (hospitals, supermarkets etc.) and being a potential high contamination risk environment due to the enclosed and crowded conditions typical to PT. The significant patronage losses triggered by the societal reaction to the pandemic and slow recovery process in ridership point in the direction of a lingering fear of contagion for the population and the need for preparedness for similar future global events.

K.1.2 Case study of Agder

The region of Agder (approx. 300 000 inhabitants, 80% concentrated in the coastal area) in southern Norway is used as a case study in the present research due to the following reasons:

- data availability- the PT provider in Agder has granted access to the transport data.
- replicability- (inter-)national replication potential for spatial and population features.
- choice of bus line 100- limitation of scope for the model development and initial testing.

K.1.3 Motivation of present research

We cannot predict the likelihood of pandemics. Therefore, we need to prepare the PT network for similar events by using forecasting models.

A promising way to identify patterns in lockdown effects on the PT network is to employ machine learning (ML) algorithms in analyzing big data sets for forecasting purposes. In this study, we use the Regression Tsetlin Machine (RTM)[1], a variant of the binary Tsetlin Machine (TM)[2], to predict the travel behavior in future pandemic scenarios. The main advantage of using TMs is the competitive accuracy of predictions, despite the fact that they are made of interpretable rule-based classifiers, memory footprint, and inference speed.

The RTM has been previously used to predict e.g. the dengue incidences in the Philippines [3]. Abeyrathna et al. in [4], [5], and [6] discussed the interpretability of the binary TM on distinct applications. In our study, this algorithm is applied for the first time to the transport domain, where we predict the PT ridership variation during the pandemic period providing evidence on the interpretability of the TM based approach through generating rules, which can be used for travel behavior predictions in future pandemic scenarios.

K.2 State of the art

K.2.1 Role of public transport in a pandemic

Browne et al. [7] studied the relationship between spread and PT. They revealed that the duration of travel and seating proximity influences the risk of infection for ground transport. In terms of patronage, a study of the 2002/03 SARS pandemic [8] showed that reported new cases caused immediate losses in the underground ridership, despite no lockdown being in place ("fresh fear") [9]. Looking at mobility data within Europe, Santamaria et al. [10] showed that "confinement measures explain up to 90% of the mobility patterns".

K.2.2 Machine learning in public transport research

Currently, applications of ML in PT research are popular in the domains of:

- Travel mode choice modeling- analyzing user data to accurately predict mode choice. Previous research indicates ML models, i.e. random forest or different artificial neural nets, as the best performers in the field [11, 12, 13].
- Travel demand modelling and forecasting- models such as Autoregressive Integrated Moving Average, dynamic Partial Adjustment Model have been used by Chi-Hong et al. [14] to predict PT demand. Mozolin et al. used neural networks (NN) for trip distribution forecasting [15]. Koushik et al. discuss that the results of applying NN for travel demand models are not in favour of NN due to the "black-box" effect [16].
- Forecasting passenger flows- Toque et al. used gated recurrent unit and recurrent NN for short term prediction of passenger flows, comparing the results with the ones of Random Forest and long-term forecasting models [17]. Wei and Chen employed the empirical mode decomposition and NN [18] for the same purpose. Toque et al. extended their previous study to long-term forecasting as well [19].

K.2.3 Machine learning in the COVID-19 pandemic

In a 2020 review, Lalmuanawma et al. conclude that "the ongoing development in AI and ML has significantly improved treatment, medication, screening, prediction, forecasting, contact tracing, and drug/vaccine development process for the Covid-19 pandemic and reduced the human intervention in medical practice. However, most of the models are not deployed enough to show their real-world operation" [20]. Another example is predicting the growth and trend of COVID-19 number of cases [21]. A recent interpretable ML algorithm, the Regression Tsetlin Machine based mobile application, has also been developed for the same purpose [22].

K.3 Methodology

In this section, we will briefly introduce the theory of the ML algorithm used in this study, data pre-processing, and model validation. The diagram of the planned work-flow is illustrated in Fig. K.1.

K.3.1 The Regression Tsetlin Machine algorithm

The RTM is a variation of TMs and a novel approach to interpretable non-linear regression [1]. The RTM takes o propositional input features, i.e., $\mathbf{X} = [x_1, x_2, x_3, \dots, x_o,]$ and sends them along with their negations (collectively called literals), i.e., $\mathbf{X}' = [x_1, x_2, x_3, \dots, x_o, \neg x_1, \neg x_2, \neg x_3, \dots, \neg x_o]$ to each of the clauses, c_j, $j = 1, 2, 3, \dots, m$. Each clause comprises of a team of Tsetlin Automata (TAs) which decides the composition of the clause. Individual TAs attached to each literal decide to include or exclude their corresponding

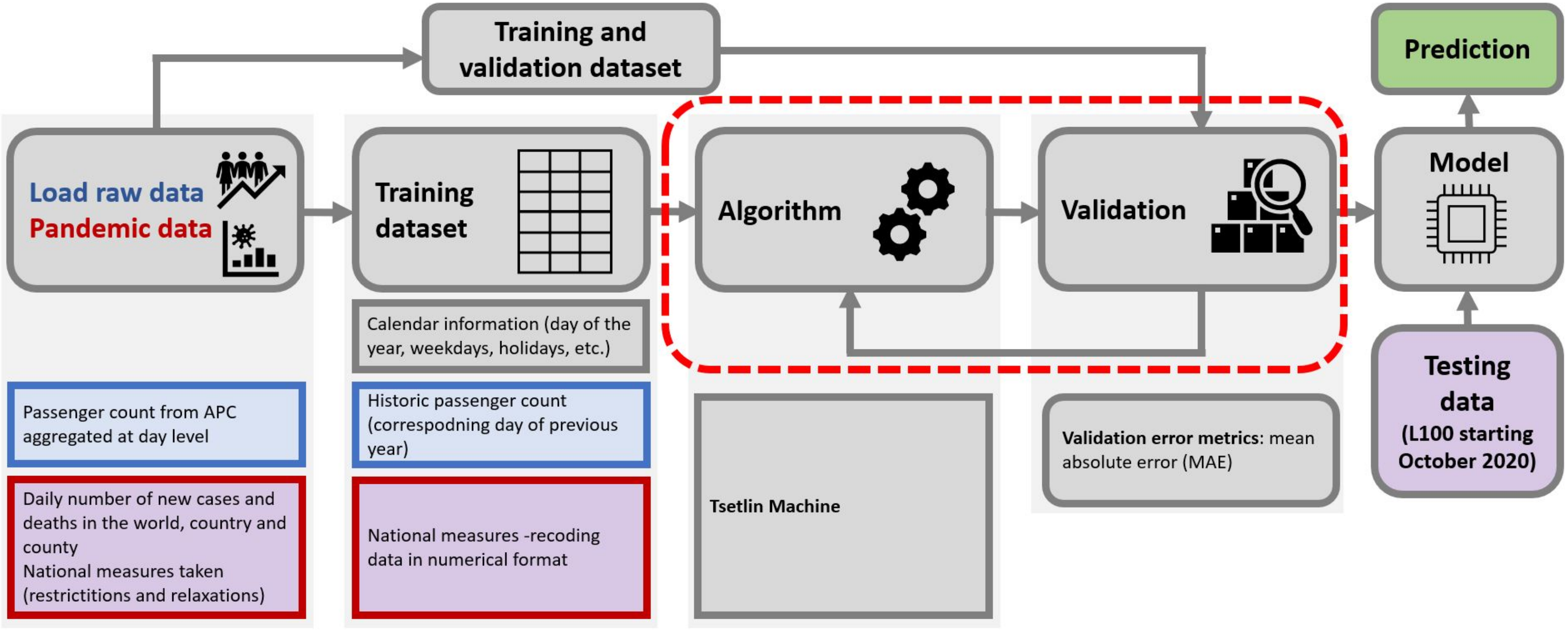

Training and validation dataset
Load raw data
Pandemic data
Training dataset
Algorithm
Validation
Model
Prediction
Calendar information (day of the year, weekdays, holidays, etc.)
Passenger count from APC aggregated at day level
Historic passenger count (correspodning day of previous year)
Daily number of new cases and deaths in the world, country and county
National measures taken (restrictitions and relaxations)
National measures -recoding data in numerical format
Tsetlin Machine
Validation error metrics: mean absolute error (MAE)
Testing data (L100 starting October 2020)

Figure K.1: Method concept.

literals in the clause. The clause then computes the conjunction of only the included literals in the clause. By carefully guiding these TAs to make correct decisions as a team, individual clauses recognize the sub-patterns in data.

Once a sub-pattern is recognized by a clause, the clause outputs 1. The resulting sum of the individual clause outputs is then mapped into a continuous value between 0 and $\hat{y}_{max}$, where $\hat{y}_{max}$ is the maximum training output.

During the training phase, the predicted daily passenger count, y, is compared against the actual passenger count output, $\hat{y}$. Depending on whether y (predicted value) is higher or lower than $\hat{y}$, the individual clauses are systematically guided to reduce the prediction error.

K.3.2 Features, Predictions, and Data Preprocessing

What we expect in this study is to train the RTM algorithm using historical PT ridership data from bus line 100 and the calendar of pandemic related events to predict the PT daily ridership variation in future pandemic situations. The model requires the input of different features to be able to estimate the future passenger count for a similar situation.

In our case, the passenger count, PC of day d for the bus line 100, $PC(d)$ is predicted using the previous day passenger count, $PC(d-1)$ previous year-same-day passenger count of bus line 100, $PC(d-365)$, previous day passenger count of the Agder province, $PC_A(d-1)$, previous year-same-day passenger count of the Agder province, $PC_A(d-365)$, number of new corona cases of day $d-1$ worldwide, $NCC_w(d-1)$ number of corona cases of day $d-1$ in Agder, $CC_A(d-1)$, number of corona cases of day $d-1$ in Norway, $CC_N(d-1)$, number of corona cases of day $d-1$ worldwide, $CC_w(d-1)$, number of corona deaths of day $d-1$ in Agder, $CD_A(d-1)$, number of corona deaths of day $d-1$ in Norway, $CD_N(d-1)$, number of corona deaths of day $d-1$ worldwide, $CD_w(d-1)$, holiday information related to day d, and different pandemic related measures related to day d. In the first phase of preprocessing, the non-numerical features (e.g. pandemic measures) are encoded to numerical values without losing the true meaning of the feature. In the second phase, the complete set of numerical features are binarized as the RTM accepts only binary form features. For feature binarization, we use the thresholding approach proposed in [3].

K.3.3 Training and Validation

The RTM model is trained on different training samples by varying the day d between 1st of January, 2020 and day t. We, in this study, focus on medium-term prediction of passenger count: up to two weeks. Hence, once the model has been trained on the data from 1st of January, 2020 to day t, this model is used to predict the passenger count for the days from $t+1$ to $t+14$.

However, as noticed, for real-life situations, predictions of passenger count for the above days have to be made based on the predictions of $PC(d-1)$, PC_A, CC_A, CC_N, NCC_w, CC_w, CD_A, CD_N, CD_w, and pandemic related measures. For instance, to predict the passenger count of day $t+7$, $PC(t+7)$ the model will require the feature values of some of the features of day $t + 6$, which are not available by day t. Hence, we use simple moving average approach to predict those input features and then send them to the model to make predictions for the future.

The pandemic related measures on the other hand have to be also estimated to make prediction for the next two weeks. Since numerical analysis can not be used to estimate the future pandemic related measures, we consider three cases where we assume that 1). no pandemic measure in the next 14 days, 2). one restrictive pandemic measure in the next 14 days, and 3). two restrictive pandemic measures in the next 14 days. For the cases where we assume to have a measure, the day of the measure is randomly selected. Considering the above conditions, the RTM is used to make two predictions: Case 1 and Case 2. For Case 1, the RTM is trained on data from 1/1/2020 to 9/15/2020 and predictions made for 9/16/2020 to 9/29/2020. In Case 2, the RTM is trained on the data from 1/1/2020 to 10/8/2020 and predictions made for 10/9/2020 to 10/22/2020. The accuracy values of the predictions are summarized in the next section.

K.4 Results and discussion

The performance of the RTM in relation to the above experiment is measured in terms of the mean absolute error (MAE) between actual and predicted passenger counts. We also contrast the performance of the RTM with a classic statistical model, Moving Average (MA), and three other widely used machine learning models: Random Forest (RF), Regression Trees (RT), and Support Vector Machine (SVM). Their performance on Cases 1 and 2 is summarized in Table K.1 and Table K.2.

Regardless of the number of measures used in testing, the RTM obtains the lowest MAE in both cases. On average, RT exhibits second best performance, closely followed by RF, while SVM and MA struggle to make competitive predictions.

The data shows that the error is higher for the Case 1 testing (Fig. K.2 and Fig. K.3). Figure K.2 also shows that all machine learning models barely recognize the ridership reduction in the weekends.

A possible reason for the above observation could be the lesser number of training samples the Case 1 has compared to Case 2, as Case 1 consists of 259 training sample while Case 2 contains 282. In our case, the missing samples in Case 1 are highly important since this seems to be the period where the impact of the second pandemic wave starts to be visible on the ridership of Line 100.

Since the drastic pandemic control measures in the first wave had been different in relation to the number of new daily cases in Norway, the available 259 training samples did

Table K.1: MAE between actual and predicted passenger counts (testing for Case 1).

		Method				
		RTM	RF	RT	SVM	MA
No. of measures	0	338.41	376.82	370.76	490.78	495.88
	1	337.81	375.50	370.62	490.79	495.88
	2	337.13	375.41	354.74	490.80	495.88

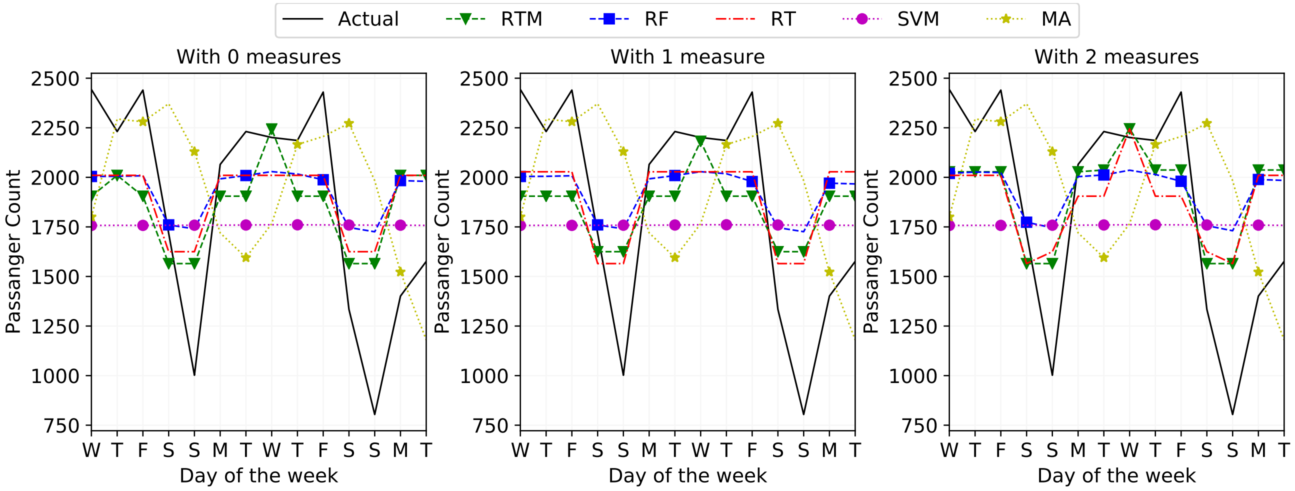


Figure K.2: Actual vs Predicted passenger count for the testing in Case 1.

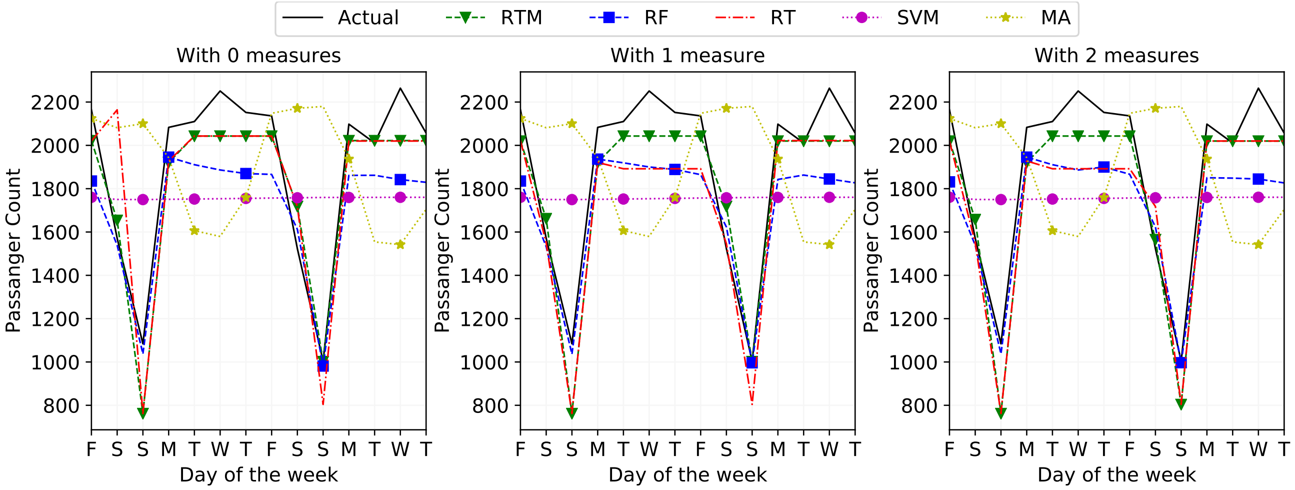


Figure K.3: Actual vs Predicted passenger count for the testing in Case 2.

not provide enough similar samples for the ML training in this situation. In the lockdown period of the first wave (March-April), the passenger count difference between weekdays and weekends is much smaller than for the two cases analyzed. Therefore, in Case 1 the ML models predict similar patterns to the lockdown period, with smaller passenger count differences between weekdays and weekends. For Case 2, the models learn to recognize the differences with better accuracy due to the increased number of training samples available.

Another possible reason could be the predicted input data for the testing period. For Case 1 these have been mainly derived from data previous to the second pandemic wave. For Case 2 the data used for predictions is already overlapping the second pandemic wave,

Table K.2: MAE between actual and predicted passenger counts (testing for Case 2).

		Method				
		RTM	RF	RT	SVM	MA
No. of measures	0	192.48	217.66	193.75	401.13	445.64
	1	187.16	237.29	193.08	401.10	445.64
	2	186.89	234.26	192.79	401.09	445.64

hence containing more realistic values as testing inputs than in Case 1.

A crucial result of the study is generating applicable rules for ridership variation predictions by interpreting the logical outputs given by the TM results. Hence, we changed the output in our dataset from regression to categorical and then applied the binary TM on it. The output is categorized as "positive trend" with a passenger count higher than the average of last 14 days, and "negative trend" otherwise. This way the clauses in the TM identify the reasons for positive and negative trends.

Using a similar approach to the ones in [4], [5], and [6], we identify the reasons for a positive trend using the complete set of features as,

IF $CD_A(d-1) \leq 7390$ **AND** $CD_N(d-1) \leq 9$ **AND** $PC(d-365) > 2220$ **AND** $PC(d-1) > 955$ **AND** $PC_A(d-1) > 6908$ **AND** $PC_A(d-365) > 39401$ **AND** day d is **NOT** a holiday **AND** Days to enforce a measure ≤ 5 **AND** day d is a week day **THEN** a positive trend.

Similarly, the reasons for a negative trend is identified as,

IF $PC(d-365) \leq 1997$ **AND** $PC_A(d-365) \leq 33703$ **AND** No pandemic control relaxation measures taken **AND** No public transport restrictive measures taken **AND** day d is a weekend day **THEN** a negative trend.

Using the above rule, the correct trend can be predicted with an accuracy of over 86%.

The next step is to remove the number of cases and deaths worldwide, and the historical ridership data from the features set and re-run the model. The resulting rule generated from the remaining features still predicts the trend with an accuracy close to 85%. Thus, the reasons for a positive trend using the filtered set of features are identified as,

IF $CC_N(d-1) \leq 100$ **AND** the number of days after a positive announcement ≤ 38 **AND** day d is a week day **THEN** a positive trend.

Similarly, the reasons for a negative trend is identified as,

IF day d is a weekend day **THEN** a negative trend.

K.5 Conclusions and future research

Our research concentrates on the application of RTM on forecasting ridership variation in PT in the specific conditions of a pandemic where the virus spreads similarly to COVID-19. The results in Tables K.1 and K.2 show that RTM obtains the lowest mean absolute error for forecasting the variation in PT ridership in comparison to all other ML models tested (RF, RT, SVM, MA). They give evidence on the interpretability of the TM, allowing for the formulation of forecasting rules.

Therefore, the method presents good potential for supporting PT providers and decision makers in their response to pandemic scenarios that affect PT ridership. This enables continuous learning by implementing current data on patronage, timetable alterations and local restrictions, allowing for minimal financial and patronage losses.

When discussing the accuracy of prediction, we observe variations in relation to the period of the pandemic for which simulations are being run. For the second wave, the accuracy of the prediction depends on how far within the second wave the simulation is being run. It may be necessary to correct the data in future pandemic scenarios (i.e. correct input data set to train model on similar number of daily cases) to ensure accuracy.

From the generated rules, we observe that only marking a pandemic control measure as positive or negative does not correctly estimate the impact strength of the measure itself. Therefore, further research is necessary on their level of impact.

Bibliography

[1] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, Lei Jiao, and Morten Goodwin. "The Regression Tsetlin Machine - A Novel Approach to Interpretable Non-Linear Regression". In: *Philosophical Transactions of the Royal Society A* 378 (2164 2019).

[2] Ole-Christoffer Granmo. "The Tsetlin Machine - A game Theoretic Bandit Driven Approach to Optimal Pattern Recognition With Propositional Logic". In: *arXiv preprint arXiv:1804.01508* (2018).

[3] K. Darshana Abeyrathna, Ole-Christoffer Granmo, Xuan Zhang, and Morten Goodwin. "A Scheme for Continuous Input to the Tsetlin Machine With Applications to Forecasting Disease Outbreaks". In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer. 2019, pp. 564–578.

[4] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "Extending the Tsetlin Machine with Integer-Weighted Clauses for Increased Interpretability". In: *IEEE Access* 9 (2021), pp. 8233–8248.

[5] K. Darshana Abeyrathna, Harsha S. Gardiyawasam Pussewalage, Sasanka N. Ranasinghe, Vladimir A. Oleshchuk, and Ole-Christoffer Granmo. "Intrusion Detection with Interpretable Rules Generated Using the Tsetlin Machine". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI).* IEEE. 2020.

[6] K. Darshana Abeyrathna, Ole-Christoffer Granmo, and Morten Goodwin. "On Obtaining Classification Confidence, Ranked Predictions and AUC with Tsetlin Machines". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI).* IEEE. 2020.

[7] Annie Browne, Sacha St-Onge Ahmad, Charles R Beck, and Jonathan S Nguyen-Van-Tam. "The Roles of Transportation and Transportation Hubs in the Propagation of Influenza and Coronaviruses: A Systematic Review". In: *Journal of travel medicine* 23.1 (2016), tav002.

[8] Kow-Tong Chen, Shiing-Jer Twu, Hsiao-Ling Chang, Yi-Chun Wu, Chu-Tzu Chen, Ting-Hsiang Lin, Sonja J Olsen, Scott F Dowell, Ih-Jen Su, and Taiwan SARS Response Team. "SARS in Taiwan: An Overview and Lessons Learned". In: *International Journal of Infectious Diseases* 9.2 (2005), pp. 77–85.

[9] Kuo-Ying Wang. "How Change of Public Transportation Usage Reveals Fear of the SARS Virus in a City". In: *PloS one* 9.3 (2014), e89405.

[10] Carlos Santamaria, Francesco Sermi, Spyridon Spyratos, Stefano Maria Iacus, Alessandro Annunziato, Dario Tarchi, and Michele Vespe. "Measuring the Impact of COVID-19 Confinement Measures on Human Mobility Using Mobile Positioning Data. A European Regional Analysis". In: *Safety Science* 132 (2020), p. 104925.

[11] Xilei Zhao, Xiang Yan, Alan Yu, and Pascal Van Hentenryck. "Prediction and Behavioral Analysis of Travel Mode Choice: A Comparison of Machine Learning and Logit Models". In: *Travel behaviour and society* 20 (2020), pp. 22–35.

[12] Long Cheng, Xuewu Chen, Jonas De Vos, Xinjun Lai, and Frank Witlox. "Applying a Random Forest Method Approach to Model Travel Mode Choice Behavior". In: *Travel behaviour and society* 14 (2019), pp. 1–10.

[13] Julian Hagenauer and Marco Helbich. "A Comparative Study of Machine Learning Classifiers for Modeling Travel Mode Choice". In: *Expert Systems with Applications* 78 (2017), pp. 273–282.

[14] Chi-Hong Tsai, Corinne Mulley, and Geoffrey Clifton. "Forecasting Public Transport Demand for the Sydney Greater Metropolitan Area: A Comparison of Univariate and Multivariate Methods". In: *Road & Transport Research: A Journal of Australian and New Zealand Research and Practice* 23.1 (2014), p. 51.

[15] Mikhail Mozolin, J-C Thill, and E Lynn Usery. "Trip Distribution Forecasting with Multilayer Perceptron Neural Networks: A Critical Evaluation". In: *Transportation Research Part B: Methodological* 34.1 (2000), pp. 53–73.

[16] Anil NP Koushik, M Manoj, and N Nezamuddin. "Machine Learning Applications in Activity-Travel Behaviour Research: A Review". In: *Transport reviews* 40.3 (2020), pp. 288–311.

[17] Florian Toqué, Etienne Côme, Latifa Oukhellou, and Martin Trépanier. "Short-Term Multi-Step Ahead Forecasting of Railway Passenger Flows During Special Events with Machine Learning Methods". In: *CASPT 2018, Conference on Advanced Systems in Public Transport and TransitData 2018.* 2018, 15p.

[18] Yu Wei and Mu-Chen Chen. "Forecasting the Short-Term Metro Passenger Flow with Empirical Mode Decomposition and Neural Networks". In: *Transportation Research Part C: Emerging Technologies* 21.1 (2012), pp. 148–162.

[19] Florian Toqué, Mostepha Khouadjia, Etienne Come, Martin Trepanier, and Latifa Oukhellou. "Short & Long Term Forecasting of Multimodal Transport Passenger Flows with Machine Learning Methods". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC).* IEEE. 2017, pp. 560–566.

[20] Samuel Lalmuanawma, Jamal Hussain, and Lalrinfela Chhakchhuak. "Applications of Machine Learning and Artificial Intelligence for Covid-19 (SARS-CoV-2) Pandemic: A Review". In: *Chaos, Solitons & Fractals* (2020), p. 110059.

[21] Shreshth Tuli, Shikhar Tuli, Rakesh Tuli, and Sukhpal Singh Gill. "Predicting the Growth and Trend of COVID-19 Pandemic using Machine Learning and Cloud Computing". In: *Internet of Things* (2020), p. 100222.

[22] K. Darshana Abeyrathna. *The Regression Tsetlin Machine Mased AI Enabled Mobile App for Forecasting the Number of Corona Patients for the Next Day in Different Countries. URL:* ***https://github.com/DarshanaAbeyrathna/Tsetlin-Machine-Based-AI-Enabled-Mobile-App-for-Forecasting-the-Number-of-Corona-Patients***. 2019.