Norwegian University
of Life Sciences

Master's Thesis 2021 30 ECTS
Faculty of Science and Technology

Tsetlin Machine for Analysis of Healthcare Data: A Comparison With Standard Machine Learning Algorithms

Petter Sunde Nymark
Master of Science in Data Science

CONTENTS

Abstract

In recent years, the use of machine learning within healthcare has increased as a result of a growing amount of data being produced. This data needs analyzing, and there exists many standard machine learning methods to do so. Many of these methods do however have problems when it comes to interpreting the models. Therefore, it is essential to find methods that provide good predictive performance and are interpretable in high-stake decision domains.

The Tsetlin Machine has shown up as a new and promising candidate regarding performance and interpretability. The Tsetlin Machine can be seen as a new branch within machine learning with propositional algebra-based logic to create models. The main aim of this thesis is to consider whether the Tsetlin Machine is applicable within the field of healthcare data science where the data considered as wide. It is also of interest to determine if the Tsetlin Machine can compete with standard algorithms to predict the survival of patients with colorectal cancer. Also, the interpretability of the Tsetlin Machine was explored, and the usefulness of this concept was evaluated.

This thesis used data from the OxyTarget study, a Norwegian study where MR images and blood sample markers were collected to analyze rectal cancer survival. There have also been other efforts, such as extracting radiomic features from the MRI. The analysis will try to predict a response variable called Progression Free Survival (PFS). The data used for predicting are radiomic features extracted from MRI images from the OxyTarget study. RENT was used to remove the features without information and thus increase predicative performance.

Dataset 1 combined version had F1-scores over 0.6 for all the models and had the highest MCCs. The best F1-score was 0.65 with SVM. The best MCC was from the Tsetlin Machine on the same version, achieving 0.33. Dataset 2 combined had the best overall score, with the Tsetlin Machine scoring 0.68 for F1-score and 0.42 for MCC. The Tsetlin Machine provided clauses in the form of propositional formulas.

The results indicate that the Tsetlin Machine can compete with the other models. For practical use in healthcare, the results are however not quite good enough. The Tsetlin Machine can also produce clauses for interpretability. These still need an automated process to display them in the original continuous values from the dataset.

Acknowledgment

This thesis was written at the Faculty of Science and Technology at the Norwegian University of Life Sciences (NMBU) in 2021, and marks the end of my five-year master's degree. I would like to thank my supervisors Oliver Tomic and Kristian Liland for being both positive and motivating through this process. Also, thank you to Ole-Christoffer Granmo for taking the time, giving me great input on the Tsetlin Machine. Finally, thank you to my friends and family for the support. Navigating the process of writing a master thesis in a pandemic has been both puzzling and intense.

Petter Sunde Nymark

Oslo, 01.06.21

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

1.1.1 Healthcare Background

In 2020 it was estimated to occur 19.3 million new cancer cases globally [1]. The same year, there were approximately 10 million deaths by cancer, and the global cancer burden is assumed to increase to 28.4 million cases by 2040. Out of all newly diagnosed cancers, nearly 10% were colorectal cancer, making it one of the most common forms and thus an important form to explore and research. Further on, colorectal cancer has the second-highest mortality rate of cancers with both men and women in Europe [2]. It is the second most common type of cancer for women after breast cancer, and for men, it is the third after prostate and lung cancer.

Because of the high number of cases and the complexity of the therapy, colorectal cancer significantly impacts health services in Norway and worldwide [2]. In Norway alone, more than 4300 new cases of colorectal cancer were diagnosed in 2016, and 24 000 people were living with it. This makes it one of the most widespread cancers in Norway. The number of colorectal cancer cases is also projected to grow in the future due to the average lifespan increasing and its growing proportion.
This thesis used data from the OxyTarget study [23], a Norwegian study where MR images and blood sample markers were collected to be analyzed. There have also been other efforts, such as extracting statistical features from the MRI to improve the effectiveness of the treatments given the data need analyzing. There are many standard machine learning methods to do this, but there are often problems when interpreting the models. The Tsetlin Machine has shown up as a new and promising candidate regarding performance and interpretability.

1.1.2 Tsetlin Machine

Rudin (2019) stated that high-stake decisions in domains such as healthcare should be made by models that are interpretable to avoid biased and dangerous outcomes [3]. The Tsetlin Machine performs on the same level as other models for classic benchmark problems such as the Binary Iris Dataset classification. It also has the potential added benefit of being interpretable. The Tsetlin Machine further has the advantage of being more interpretable than other machine learning algorithms. Keeping this in mind, it seems plausible to investigate how useful the Tsetlin Machine could be in the field of medicine, as well as considering if the model can compete or even outperform the more common machine learning models. This thesis is therefore, to a certain degree, an exploration of whether the Tsetlin machine could be a valuable tool in machine learning and AI.

1.2 Structure of Thesis

The structure of this thesis is as follows. Chapter 2 explores the theory around the Tsetlin Machine, starting with machine learning, then working towards the essential parts of the algorithm and how it is constructed to work as a classifier. The chapter also has a short description of the algorithms for training models for comparison to models based on the Tsetlin Machine. In chapter 3, the materials used in the analysis are listed and explained. Further, chapter 4 contains the method, explaining how the theory and materials were set up to produce the results. The results are listed and explained in chapter 5. Chapter 6 contains the discussion of the results in relation to the project's aim and other important implications. Chapter 7 contains the conclusion of the thesis, as well as proposing potential further work.

1.3 Project Aim

The Tsetlin Machine can be seen as a new branch within machine learning with propositional algebra-based logic to create models. The main aim of this thesis is to consider whether an algorithm within machine learning, namely the Tsetlin Machine, is applicable within the field of healthcare data science where the data considered as wide, that is, relatively few rows relative to the number of features. More specifically, it is of interest to determine if the Tsetlin Machine can compete with standard algorithms within machine learning to predict the survival of patients with colorectal cancer. Also, the interpretability of the Tsetlin Machine was explored, and the usefulness of this concept was evaluated. Though there might be medical advances resulting from the thesis, the main concern is to explore the use of algorithms to improve data-driven predictions.

The analysis will try to predict a response variable called Progression Free Survival (PFS). This is a binary classification problem where the positive value 1 represents cancer recurrence, metastases, or death within three years, and the negative value 0 is when the patients survive cancer-free. The data used for predicting are radiomic features extracted from MRI images from the OxyTarget study [23].

There was potential to cover more on the Tsetlin Machine, but due to the scope of this thesis with the timeframe, limitations must be set as the project aim states.

Chapter 2 Theory

The following chapter contains the theoretical background, where the goal is to put a theoretical context to the issue at hand. First, machine learning as a general concept is introduced. Later classification and reinforcement learning is explained as these are both crucial concepts to understand the theory behind the Tsetlin Machine and answer the central questions of this thesis. Further on, the Tsetlin Machine is described as this is the main focus. Finally, as the thesis compares the results of the Tsetlin Machine with other models', the other models are briefly explained. This part is not as comprehensive because the focus is to explore the potential of the Tsetlin Machine.

2.1 Machine Learning

Raschka (2017) explained that Machine Learning is a subfield of artificial intelligence that concerns self-learning algorithms that extract knowledge from a set of data to make predictions [4]. The basic premise of machine learning is capturing knowledge in data without requiring humans to explicitly program rules and building models from analyzing large quantities of data. Thus, the models can gradually improve the performance of the predictive capabilities and make data-driven decisions [4].

When referring to training data and targets, it can be helpful to use a matrix representation of the data. For the data, as seen in formula (1), $\boldsymbol{X}_{\boldsymbol{j}}^{(\boldsymbol{i})}$ is used. Here the subscript i is the number of training samples: for example, the number of patients. The second subscript j is the number of features, making it a feature space in the j-th dimension. These features can hold describing values such as age, height, gender, and other informational characteristics. The values can be, but are not limited to, continuous, discrete, categorical, and so on.

$$\boldsymbol{X}_{\boldsymbol{j}}^{(\boldsymbol{i})} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_j^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_j^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(i)} & x_2^{(i)} & \cdots & x_j^{(i)} \end{bmatrix} \qquad (1)$$

Like the data, the target variables can be represented as seen in formula (2), with the i-subscript corresponding with ones in the $\boldsymbol{X}$ matrix. In this example, the variable can be 1 for a positive class and 0 for a negative class. This is similar to the targets used later in the analysis.

$$\boldsymbol{y}^{(\boldsymbol{i})} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(i)} \end{bmatrix}, \qquad y \in \{1,0\} \qquad (2)$$

There are three types of machine learning: unsupervised learning, supervised learning, and reinforcement learning. Each has its different learning methods [4], and each subfield has its characteristics as seen in figure 2.1.

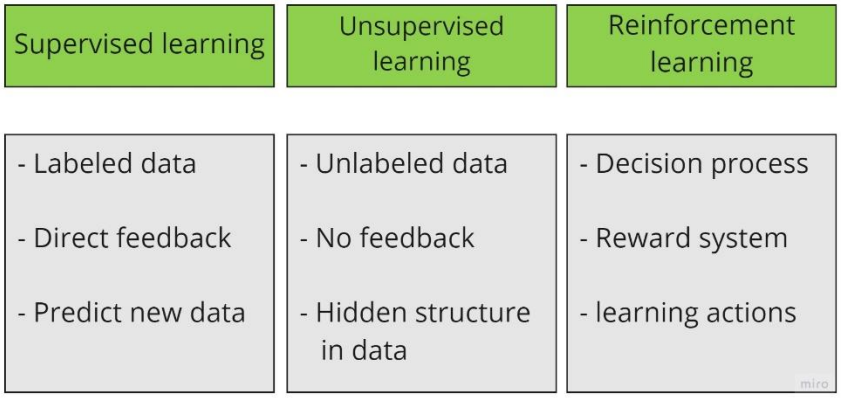

Figure 2.1: The types of ML and their characteristics

Supervised learning is machine learning where the label of the training data is known [4]. Here, the algorithms will use the data $\boldsymbol{X}_{\boldsymbol{j}}^{(\boldsymbol{i})}$ to map it to the target variables in $\boldsymbol{y}^{(\boldsymbol{i})}$. This allows algorithms to predict unseen or future data that do not have labels. An example of supervised learning is if one trains a model on a dataset of cancer patients, where the label is whether the patient has cancer or not. The model will learn the patterns in the data $\boldsymbol{X}_{\boldsymbol{j}}^{(\boldsymbol{i})}$, that is

characteristic for each of the labels. Supervised learning is further subdivided into regression problems and classification problems. Regression is when the algorithm is trying to predict a continuous variable. On the other hand, classification is when one tries to predict distinct classes, meaning trying to predict a discrete value.

Unsupervised learning handles unlabelled data and data of unknown structure [4]. This type of machine learning is about using algorithms to explore and extract information from the data without having the correct answers to guide the learning process. Typically, this information is used to group data points into subgroup clusters. Another use is to reduce the dimensionality of the data to remove noise and decrease the computational cost of the ML algorithm.

As reinforcement learning will be the main focus, the concept will be explored in the next section.

2.1.1 Reinforcement Learning and Game Theory

In order to properly understand the Tsetlin Machine and how it can be used to predict classes, it is important to understand how the field of reinforcement learning and game theory intersect and thus can be used in machine learning. Reinforcement learning is the third subfield of machine learning alongside supervised and unsupervised learning. This part of machine learning focuses on learning models with agents operating in some given environment and learns through trial and error. The agent typically has an end goal and will perform actions in the environment to try to reach the goal. If the action is favourable, the agent will get rewarded, and if it is not, it will be ignored or given a penalty. Trying to optimize the actions taken to maximize the rewards and minimize the penalties can be considered the learning process. How the model solves the optimization problem to find the best actions will differ from algorithm to algorithm. Some algorithms will learn to ignore immediate small rewards to get.

An example of this could be a simulation of a mouse in a maze. The mouse is the agent, the maze the environment, and what turn to take is the action. The goal is to reach a piece of food placed in the maze. At first, it will take actions at random, but the mouse will remember the best actions to get it to the goal based on rewards and penalties over many iterations.

Game theory is a field of mathematics studying the theory behind the phenomena observed when decision-maker interact (Osborne & Rubinstein, 1994, p. 1) [5]. It is applied in many research fields, such as economics, biology, social sciences, and computer science. It has its roots in the zero-sum games and its proofs in the book "Theory of games and Economic Behaviour" by John Von Neuman and Oskar Morgenstern [6]. Game theory looks at games and how to take the best decision toward a given goal. It is crucial to understand what the Nash equilibrium is to understand the fundamentals of the Tsetlin Machine. The Nash equilibrium is a solution or set of solutions in a non-cooperative game [5]. When the equilibrium is reached, there is nothing to gain for any of the agents or players by changing their strategy. In machine learning, reinforcement learning can be used to find the Nash equilibrium, as will be discussed later in this thesis. Section 2.2.5 will show how the Tsetlin Machine sets up a game so that the Nash equilibrium coincides with the patterns in the data to get optimal patterns recognition [7].

Al, within game theory, an essential concept for understanding the Tsetlin Machine is the multi-armed bandit problem [8]. In this problem, one has N-arms on classical bandit machines giving N actions to take. Each arm has a certain probability of giving a reward, with the probabilities being unknown. The gambler or agent is trying to maximize the gain from the machines. The problem is then to find the balance between exploration and exploitation. This means that the gambler is trying to maximize the sum of rewards by finding a trade-off between trying new arms or continue pulling an arm that has a seemingly good probability of giving a reward. Section 2.2 will explore how the Tsetlin Automata solves a two-armed version of the bandit problem to find the explore-exploit trade-off.

2.1.2 Learning Automata

The Tsetlin Automaton belongs to the subfield of reinforcement called Learning Automata [7]. Narendra, K. S., & Thathachar, M. A. (1974) coins Learning Automata as automata operating in an unknown random environment. For each action of the Automata, the environment it operates in will have a probability distribution of rewarding or penalizing. An unknown random environment has unknown probability distributions for the responses to the actions of the Learning Automata, where the distributions may change as a result of the actions. The automata update their action probabilities as they interact with the environment

and learn to improve their performance during its operation [11]. For example, take an automaton with a finite number of actions to take in a random environment. Doing an action will result in the random environment responding with either a favourable or non-favourable response. The random response will follow a probabilistic distribution, and these probabilities are unknown. The goal of the Automaton is then to try to learn the probabilities associated with a given action based on earlier actions and interactions with the environment.

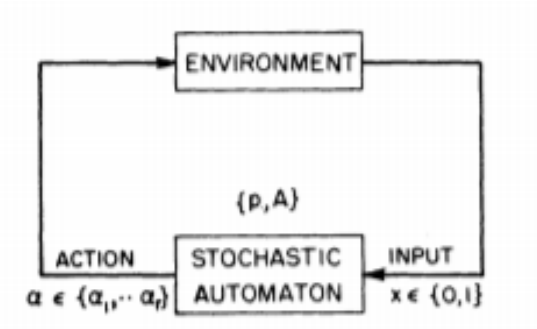

Figure 2.2:: A Learning Automaton with input from the environment and actions in the environment. Adapted from Narendra, K. S., & Thathachar, M. A. (1974).

Figure 2 shows an example Learning Automaton, with a binary input set $x \in \{0,1\}$ from the environment and output (action) set $\alpha \in \{\alpha_1, \dots, \alpha_r\}$ [11]. Narendra, K. S., & Thathachar, M. A. (1974) describe the Stochastic Automaton by the sextuple $\{x, \phi, \alpha, p, A, G\}$. In this sextuple, $\phi = \{\phi_1, \phi_2, \dots, \phi_s\}$ is the set of internal states of the Automaton with $r \leq s$ so that each action has at least one state. p is the probability vector that by each step n decides the choice of state $p(n)$, meaning that each state has a corresponding probability. A is an algorithm that generates the probabilities at the next step $p(n+1)$ from $p(n)$. G is the output vector that outputs an action α from ϕ. These steps create a feedback loop between the environment and the Automaton, where the action is the input for the environment, and it in turn outputs the binary response that is the input of the Automaton. The response then influences the updating of the action probabilities. It should be noted that the environment that the Automaton operates in can vary greatly and can have many different types of outputs. Binary is used in this example for simplicity.

Two or more Learning Automata can be set up to play games similar to the games described in game theory [11]. For simplicity, consider two Automata, both operating in a random

stochastic environment without any knowledge about each other or the environment. Both automata will perform an action, and the environment will respond randomly. The automata will then update their action probability according to their reinforcement scheme. This is a round of the game and is repeated over a number of iterations. The game's goal is to figure out a set of actions (also called a strategy) that maximize pay-off. The automata can play against each other, which is called a competitive game, or they can play together called a cooperative game [6]. Because the environment is unknown with the pay-off function having a random distribution, the automates must learn as they play to find the optimal strategy.

2.1.3 Classification

Classification methods are used within machine learning when the goal is to determine what category a sample belongs to. It is advantageous to use when the goal is a discrete class. The number of categories depends on the problem one is trying to solve. The cancer example mentioned in supervised learning is a typical binary classification problem with only two classes; the patient either has cancer or not. However, if the problem is what type of cancer, multiclass classification is necessary. The algorithm will use pattern recognition to separate the data into the different classes of cancer.

For the classifiers to make sense, one needs methods to evaluate their performance. There are many different metrics for evaluating the classifier, and selecting the right one is essential to getting the most reliable results. Which one is the best will depend on the question and the data. The most basic one is accuracy, where only the number of correct predictions is considered. However, this can be misleading, for instance, in a dataset with an unbalanced number of classes. If the target value is 0 and 1, but only 5 percent of the total number of values is 1, the classifier can guess every output to be 0. An accuracy metric would give a 95 percent accuracy, which seems like a good result, but it got none of the critical values correct. For cases like these, other evaluation methods need to be used to get better insight.

2.1.4 Model Evaluation

When evaluating the model in the analysis pipeline, using evaluation metrics suited for the task at hand is essential. If one only uses accuracy, one can get a wrong accuracy for an imbalanced classification problem impression of the score if it is unbalanced. It does not necessarily provide a realistic picture of the situation, leading to incorrect conclusions and poorly performing models. Therefore, several different metrics were used for the analysis.

F1-score

One of the evaluation metrics used was the F1-score. It is widely used in machine learning applications for imbalanced class distribution [15]. It is a metric based on precision and recall. As seen in figure 2.3, the prediction of a sample in a binary classification problem may be True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). From this $Recall = \frac{TP}{TP+FN}$ and $Precision = \frac{TP}{TP+FP}$ [16].

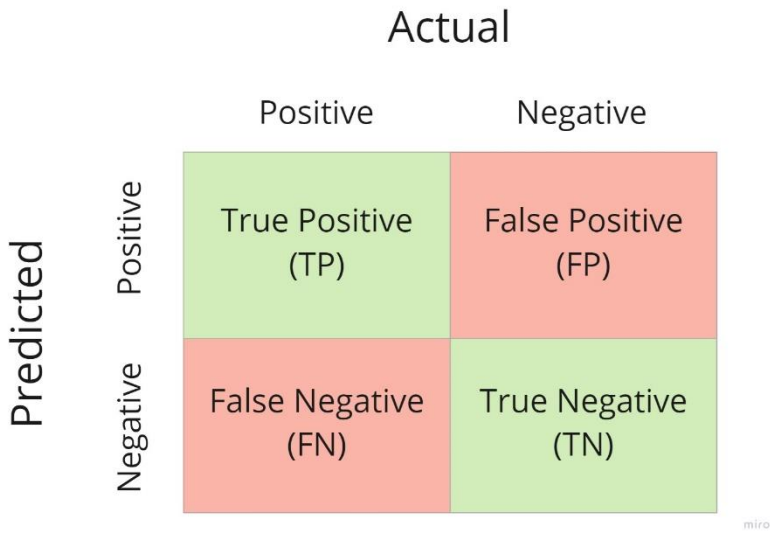

Figure 2.3:Confusion matrix of binary classification

F1 has a value between 0 and 1, where F1 = 1 is perfect prediction, and F1 = 0 is equivalent to random guessing and is defined:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + TP)}$$

The F1 score has an advantage over just accuracy, as it can deal with unbalanced classes. It was used because it has become an industry standard and makes for a good general comparison method [15]. As Chicco, D., & Jurman, G. (2020) states, the F1 measure has flaws, so other metrics were introduced as well.

Matthews correlation coefficient

Matthews correlation coefficient is a measure also unaffected by dataset imbalance. It is a method of calculating the *Pearson product-moment correlation coefficient* [15] between the predicted and actual values. Using the terms from figure 2.3, Matthews correlation coefficient is defined as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}$$

It has the ability of only giving a high score if the positive classification instances, as well as the negative instances, were correctly predicted [15]. The score has a range of $[-1,1]$, where MCC = 0 is the same as random guessing, MCC = 1 is perfect prediction, and -1 is perfect misclassification.

2.1.5 The Curse of Dimensionality and Interpretable models

For datasets with high-dimensional feature space, it can be difficult for machine learning models to find patterns if the number of training samples is few. When the feature space is high, there may be many possible combinations of the various ranges of the values. In order to capture this, it is desirable to have many training samples. Typically, there will be a peak of performance, then a drop as the number of features in the space increase due to overfitting when not using some form of regularisation. This is called the peaking phenomenon [9].

Depending on the situation, gathering new samples to prevent this might not be possible. In these situations, one must use methods to reduce dimensionality to get better results.

One type of dimension reduction is feature selection. The focus of this field is to remove the redundant features that contribute unwanted noise to the data. By finding the best features, one can avoid the Curse of Dimensionality and produce smaller and simpler models that are more interpretable, which in turn gives shorter runtimes.

As a result of increasing demands for machine learning, the need for developing better methods has pushed the field to new heights, with algorithms reaching astonishing results in terms of performance [11]. However, this has come at a price. The best methods have many parameters, making it difficult to understand how they reached a given decision. This is often referred to as the model being a "black box"; the data is fed to the model, and a prediction is made, but one often has difficulty understanding what happens in the process of learning and making predictions like visualized in figure 2.4. Opening this "black box" could be very useful for many fields, such as medicine. Understanding why the algorithm predicts a prognosis is essential information and could lead to discovering new biomarkers of the disease.

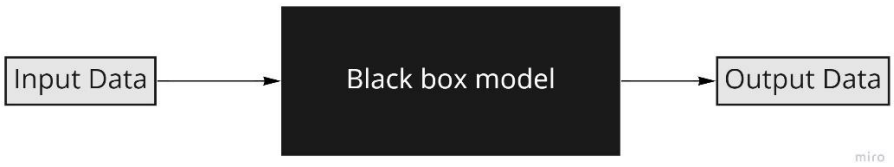

Figure 2.4: Black Box model visualization

2.1.6 Validation

When training a model, it often good practice to set aside a portion of the available data for testing in the later stages of training. This is done to validate the model's performance on unseen data, to see how well it generalizes. It is important to do a step like this to avoid overfitting the model. How much data one sets a side depends on how much data is available; the bigger the dataset, the more can be used as test data. Usually, this is done by splitting a given percentage of the data, as seen in figure 2.5. However, in cases like this thesis, where

few samples are available, it is not very informative to do a standard split. Because there is little data left to train on and the test data is even more limited, it can be susceptible to how the samples are split. This means that the performance can vary greatly depending on what samples are in the training split and what samples are in the test split. In cases like these, one can use other methods to validate the data.

Figure 2.5: Splitting data into a training set and a test set

One way to validate the performance is through cross-validation, where one trains k model using k folds of the training data. One of the folds is held as validation each iteration meaning the models are trained on $k - 1$ folds as seen in figure 2.6. Each iteration, a different fold is used. This way, it can be worked around being sensitive to how it is partitioned. When using stratified K-fold, the distribution of classes are preserved in each fold.

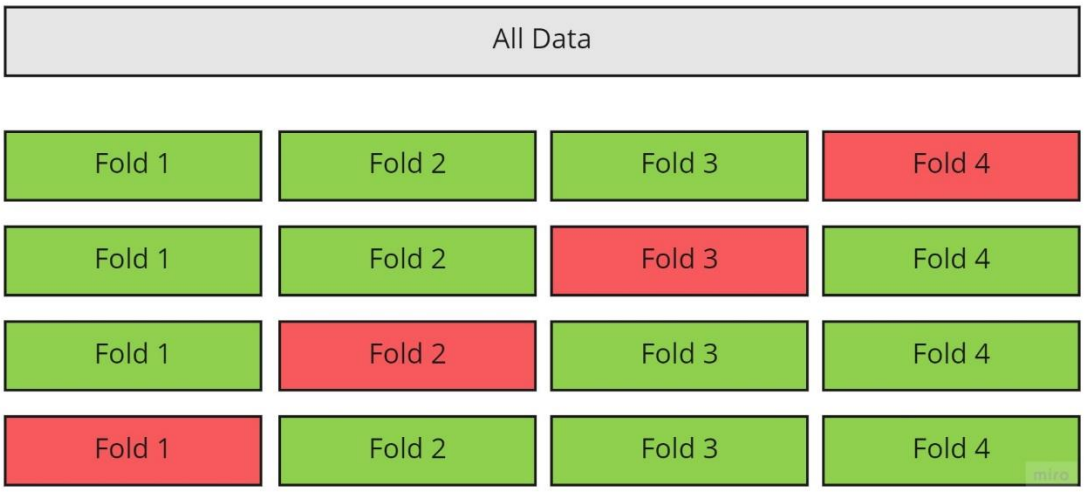

Figure 2.6: Splitting data into four folds

2.2 Tsetlin Machine

The Tsetlin Machine is a pattern recognition algorithm introduced by Ole-Christoffer Granmo in his 2018 paper [7]. The paper explains how the Tsetlin Machine can perform complex pattern recognition using a collective of Tsetlin Automata. The Tsetlin Automaton is a method of solving the multi-armed bandit problem [8] from game theory. It identifies these patterns using propositional logic [7], which can also be used for interpretation. Section 2.2 will explain how the Tsetlin Machine works. It is structured in sub-sections to make each part of the Tsetlin Machine more understandable, stating with the necessary basics and progressing to the more complicated components.

2.2.1 Propositional logic

The Tsetlin Machine is based on propositional logic and thus is central to understanding it. Propositional logic, also called propositional algebra, is a branch of logic dealing with true and false propositions. Due to the binary nature, it is an influential subject in the logics of computer science. It has a role as its formal language and theoretical basis [10]. The syntax of propositional logic follows a set of rules to create statements (propositional formulas) that are either true or false. The statements are built up by propositional variables that are combined using logical operators. In this way, complex formulas can be created. The standard notation for the logical operators is conjunctive denoted $\wedge$ (logical *and*), disjunctive denoted $\vee$ (logical *or*), and the logical *not* is denoted $\neg$. Further, this thesis will denote the propositional variables with x_i, as done in Büning & Lettmann (1999), where $i = 1, 2, \dots, n$. Büning & Lettmann (1999) describe in their book the following set of rules:

1. Every propositional variable is a formula.
2. If $x_1$is a formula, then $\neg x_1$ is also a formula.
3. If x_1 and x_2 are formulas, then $x_1 \wedge x_2$ and $x_1 \vee x_2$ are formulas.
4. Only the expression forms given by 1. - 3. are formulas.

These rules can create highly complex formulas and are the language used by the Tsetlin Machine to describe complex patterns. A more detailed explanation of how the Tsetlin Machine does this will be shown in section 2.2.2. This gives the Tsetlin Machine its interpretability as its patterns can be explained through propositional statements [7], which are readable by humans.

Suppose one uses this logic on natural language and wants to make a statement if an object is, for example, a tree or not. Then one can create a statement such as: IF an object is OVER 3 meter AND has left. If both conditions of the statements are true, then the whole statement is true. This is, of course, not a very detailed description of a tree. Adding more conditions could help make the statement more refined, and thus the description more detailed.

2.2.2 The Tsetlin Automaton

The basic building block of the Tsetlin Machine is the Tsetlin Automaton. The Automaton is a solution to a two-armed bandit problem explained in section 2.1.2, where it is trying to learn the best action. Functionally, it belongs to reinforcement learning, where it is trying to reinforce the best action in an environment.

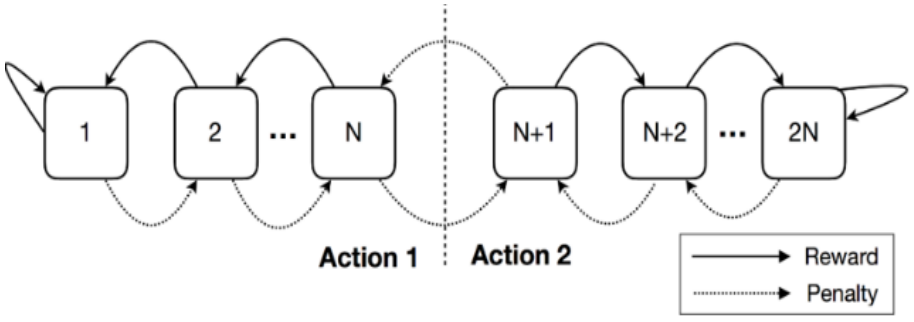

Figure 2.7: "A Tsetlin Automaton for two-action environments." From O.C Granmo (2018)

There are two possible actions for the Tsetlin Automaton, and performing an action will result in either reward or penalty [7]. The agent of the Automaton will have a state in range 1 to 2N. If the agent is in state range 1 to N, it will perform action 1. Consequently, it will perform action 2 when in range N + 1 to 2N, as seen in figure 2.7. It will then check if the action

performed was correct in relation to the ground truth. If it was, it gets a reward, and the agent will update its state such that it moves further towards state 1 if the state is in the range 1 to N or towards 2N if the state is in the range N + 1 to 2N. This reward and penalty system will show how strongly the automata prefer one action. The further towards one side the agent is, the more secure it is that the given action will give the overall best yield. When transitioning from action 1 to action 2 or reverse, it is always the result of a penalty. This is the way the Tsetlin Automata tries to solve the explore and exploit problem.

2.2.3 From Input Data to Literals and Clauses

As input data, the Tsetlin Machine will use some given binary data $X = [x_1, x_2, \dots, x_n], x_p \in \{0,1\}$. The data will be used to create propositional patterns and thus must be binary so that it is equivalent to a vector of propositional variables. We then also must consider the counterpart to each of the propositional variables [7], $\neg x_k = 1 - x_k$ With $k \in [1, \dots, 2n]$.Together these make up the literal set $L = [l_1, l_2, \dots, l_{2n}] = [x_1, x_2, \dots, x_n, \neg x_1, \neg x_2, \dots, \neg x_n]$, meaning L is double the length of X. The Tsetlin Machine learns by creating patterns through ANDing ($\bigwedge$-notation) a subset of the literals $L_j \subseteq L$ into conjunctive clauses denoted as C_j, with j as the index of the clause. This results in [7]:

$$C_j(X) = \bigwedge_{l_k \in L_j} L_j = \prod_{l_k \in L_j} L_j$$

As an example, if one have the input data $X = [x_1, x_2, x_3]$ and the full literal set $L = [x_1, x_2, x_3, \neg x_1, \neg x_2, \neg x_3]$, one possible clause could be $C_j(X) = x_1 \wedge x_3 \wedge \neg x_2 = x_1 \times x_3 \times \neg x_2$. This clause is created from the subset $L_j = [x_1, x_3, \neg x_2]$. If the values then are $X = [1,0,1]$ and $L = [1,0,1,0,1,0]$, the clause would output $C_j(X) = 1 \wedge 1 \wedge 1 = 1$.

As stated in Granmo (2018), the conjunctive clauses can express no less than 2^{2^n} unique propositional patterns when *n* is the number of input variables. Because of this, the conjunctive clauses are great for expressing non-linear patterns pattern recognition [7].

2.2.4 Tsetlin Automata Teams for Creating Clauses

The next step to understanding the Tsetlin Machine is using the Tsetlin Automaton with the literals and propositional logic to give it the ability to learn patterns. When the Tsetlin Machine initializes, there is one Tsetlin Automaton per literal, which oversees deciding whether the literal is included in the resulting conjunctive clause or excluded. Figure 2.8 shows a visualization of a Tsetlin Machine with two clauses, one positive and one negative. It depicts input data with two binary features. This leads to a literal set of 4 features, and thus 4 Tsetlin Automata per clause, giving a total of 8. Each TA has six action states, and the black dots in figure 2.8 is the current state. If the dot is over the dotted line, the TA is included (action 1) and conversely excluded (action 2) when under the line. If a TA and its literal are included, the figure depicts them brighter than the excluded. The figure also shows the outputs $W_+$and W_- from each of the clauses that are used in the majority voting. The majority voting will be further explained in the next section. Lastly, figure 2.8 shows the feedback, which updates the states in each of the TA. The mechanic behind the feedback will be explained in section 2.2.5. As explained in section 2.2.1, the TAs have two actions, which is to decide whether the literal is included in the conjunctive clause. The TM will use the mechanics of the TA to learn what action they will take. If we look at the two clauses in figure 2.8, the positive and the negative has learned different patterns. These patterns will determine the clause output based on the input X and what it will vote towards as the final prediction.

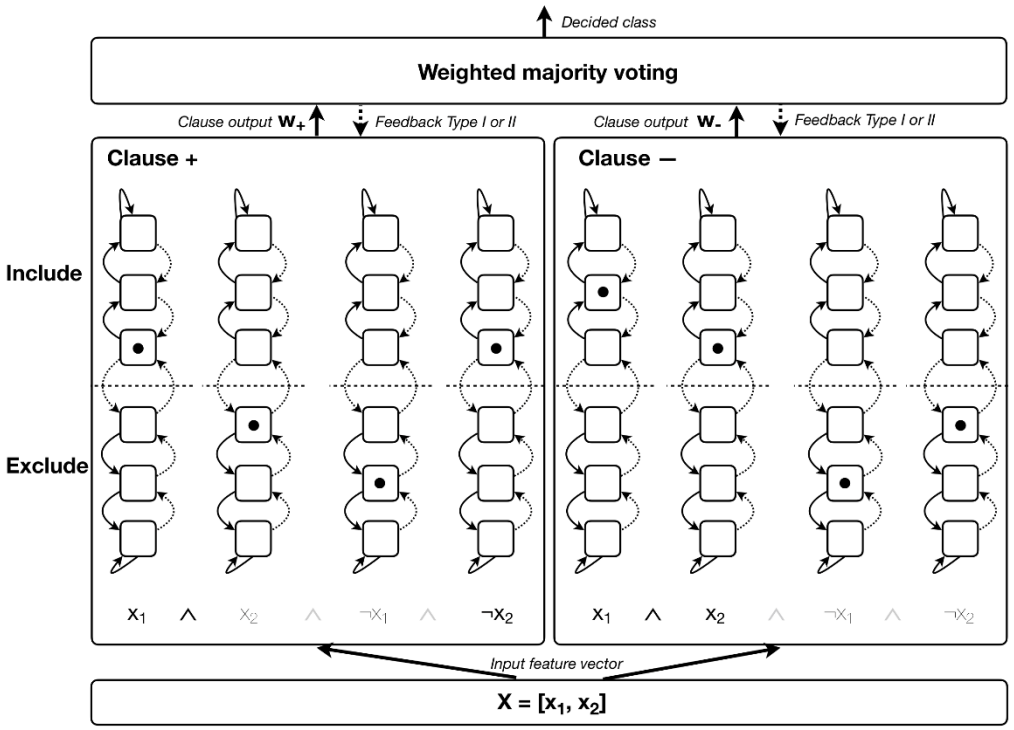

Figure 2.8:: "Two Tsetlin Automata teams, each producing a conjunctive clause. The overall output is based on majority voting." From O.C. Granmo (2018)

2.2.5 Majority Voting

The Tsetlin Machine has a parameter m that decides the number of clauses. The input X will be fed to each of these clauses. Half of the clauses will be assigned positive polarity, and the other half will be assigned negative polarity. The positive clauses are in Granmo (2018) denoted with upper index 1: C_j^1, with $j \in \{1,2,\ldots,m/2\}$. The negative clauses with the upper index 0: C_j^0, with $j \in \{1,2,\ldots,m/2\}$. The role of the clauses is to learn different sub-pattern, and during classification, each of the clauses will individually try to predict the class of the input sample.

When deciding on the prediction output $\hat{y}$ the Tsetlin Machine uses majority voting. Using a step unit $u(v) = 1\ if\ v \geq 0\ else\ 0$, the outputs of both positive and negative clauses are summed and checked on which side of the threshold they fall. When voting, the positive

clauses votes for $y = 1$ and the negative clauses votes for $y = 0$. The resulting formula for majority voting is:

$$\hat{y} = u\left(\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X)\right)$$

As Granmo (2018) stated, the purpose of the clauses in the majority voting is to get a balanced decision, weighting negative and positive evidence found in the input. An example used in the paper are clauses that capture the classic XOR-relation [?]: $\hat{y} = u(x_1 \neg x_2 + \neg x_1 x_2 - x_1 x_2 - \neg x_1 \neg x_2)$.

2.2.6 Tsetlin Automata Game as Learning Process

As described earlier, each clause $C_j(X)$ consists of a team of Tsetlin Automata, with each Tsetlin Automaton representing a literal l_k. The Tsetlin Automaton is deciding to include or exclude the literal clause, as seen in figure 5. They learn if a literal should be included or not through a game of automata to reinforce them. Because of this game, the Tsetlin Machine reaches a global optimum when learning [7].

Granmo (2018) explains the game-theoretic logic behind the learning mechanism and how it solves the pattern recognition problem. The game is set up between the Tsetlin Automata and is played over multiple rounds. Each Tsetlin Automaton decides separately to perform either the include or action exclude depending on the current state. Since the number of actions available to each Automaton is two, the number of available action configuration is 2^N (N is the number Tsetlin Automata). This gives the game a high potential for complexity. After each round, every Automaton is either penalized or rewarded, thus changing the state of every Tsetlin Automaton according to the internal rules, as mentioned in section 2.2.1. For the game to be fully specified, each of the Tsetlin Automata must be assigned a reward probability for each unique configuration of actions. In the paper, those probabilities are stated as the *pay-off matrix* of the game [7]. Since the Tsetlin Machine uses N two-action Tsetlin Automata, this gives us $N2^N$ probabilities in the pay-off matrix.

Because of how the Tsetlin Automaton together decides the output of the Tsetlin Machine, the game can potentially become very complex. Because of this, Granmo (2018) has carefully designed the *pay-off matrix* so that the Tsetlin Automata jointly is guided towards solving a given pattern recognition problem. Because of the potential size of the problem, the matrix cannot be specifically stated [7]. To solve the problem, the *pay-off matrix* is decomposed to tackle true positive, false positive and false negative outputs of the clauses, therefore treating each clause as individual classifiers. To increase the freedom of the automata, true negatives are ignored. To lead the Tsetlin Automata towards increased pattern recognition accuracy, the false positive and false negative clause outputs are progressively suppressed. The Granmo (2018) paper refers to the guiding as Type I and Type II feedback [7].

Type I feedback is responsible for increasing true positive outputs and preventing false negative outputs [7]. It is given when a clause is of positive polarity $C_j^1(X)$ when $y = 1$ and to clauses of negative polarity when $C_j^0(X)$ when $y = 0$. In addition, as seen in table 2.1, the action α (include or exclude) and the value of the literal l_k decides the probability. So, for example, if a clause evaluates $C_j^1(X) = 1$, and the literal value is $l_j = 1$ and the action state has decided *include*, then the probability of rewarding the Tsetlin Automaton is $\frac{s-1}{s}$, the probability of inaction is $\frac{1}{s}$ and the probability of penalty is 0. The inaction is an extension of the Tsetlin Automatons two actions and leaves its state unchanged [7].

Input	*Clause*	1		0	
	Literal	1	0	1	0
Include Literal	$P(Reward)$	$\frac{s-1}{s}$	NA	0	0
	$P(Inaction)$	$\frac{1}{s}$	NA	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	$P(Penalty)$	0	NA	$\frac{1}{s}$	$\frac{1}{s}$
Exclude Literal	$P(Reward)$	0	$\frac{1}{s}$	$\frac{1}{s}$	$\frac{1}{s}$
	$P(Inaction)$	$\frac{1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	$\boldsymbol{P(Penalty)}$	$\boldsymbol{\frac{s-1}{s}}$	**0**	**0**	**0**

Table 2.1: Type 1 Feedback

Type II feedback is responsible for preventing false positive outputs [7]. Similar to type I, it follows a set of rules to create its "*pay-off matrix*". It is given when the class polarity of the clause does not match the *y*. The feedback is produced when $C_j^0(X)$ and $y = 1$ or $C_j^1(X)$ and $y = 0$. The purpose is to increase the discriminant power of the Tsetlin Machine. Table 2.2 shows that if a clause is supposed to output 0 when $C_j^0(X)$ and $y = 1$ or $C_j^1(X)$ and $y = 0$ to achieve this [7]. However, if the clause evaluates as 1 by error, then a zero-valued literal will be given a penalty to include it in the clause. Because of the conjunctive logic of the Tsetlin Automata teams, the clause will change its evaluation to 0. In the situations besides this, as one can see in table 2.2, type II feedback will only cause inaction. Because the *include* reinforcement is left to the type I feedback, the Tsetlin Machine avoids local minima and work together to minimize the expected output error and go towards the global optimum. In the paper Granmo (2018), it is shown analytically that the optimal configuration of sub-patterns is the Nash equilibrium of the Tsetlin Machine game.

Input	*Clause*	1		0	
	Literal	1	0	1	0
Include Literal	$P(Reward)$	0	NA	0	0
	$P(Inaction)$	1	NA	1	1
	$P(Penalty)$	0	NA	0	0
Exclude Literal	$P(Reward)$	0	0	0	0
	$P(Inaction)$	1	0	1	1
	$P(Penalty)$	**0**	**1**	**0**	**0**

Table 2.2: Type II Feedback

2.2.7 Parameters: s, T, and Number of Clauses

The hyperparameter s from the type I feedback probabilities is set by the user [7]. As can be deduced from table 2.1 $s \geq 1$ and decides how much to favour the inclusion of literals by the Tsetlin Automata. Another way to word it is, the *s* decides how finely detailed the clauses will be and how frequent they are going to be produced. This means that the bigger the *s* is the more the Tsetlin Machine will favour the inclusion of the literals. Consequently, the opposite is true for *s* closer to 1. Finding the correct s for a given pattern recognition problem is a part of hyperparameter-tuning.

Another hyperparameter is T, which is a target for the summation of clauses. It is enough for the Tsetlin Machine output to have the correct sign when summing the output the clauses $\sum_{j=1}^{n/2} C_j^1(X) - \sum_{j=1}^{n/2} C_j^0(X)$, as this will be put through a step unit function. T is introduced to help with noisy data [7]. Granmo (2018) states that the intention of this hyperparameter is to make the available clauses distribute themselves between sub-patterns in the data [7]. It also potentially makes for interplay between the clauses, including rectification of special cases [7]. T also comes into play with the resource allocation mechanism [7] that the Tsetlin Machine uses. This mechanism ensures that not too many resources are used on each of the specific sub-patterns. It does this by randomly selecting clauses to receive feedback so that the intensity is reduced when getting closer to the summation target T.

The last hyperparameter to be set by the user is the number of clauses to be created. As mentioned in section 2.2.4, half of the clauses are given positive polarity, and the other half is given negative. This gives the user control over how many potential sub-patterns the Tsetlin Machine can potentially find in the data.

2.2.8 Binarizing of Continuous Data

The Tsetlin Machine only takes binary variables as input. Therefore continuous data needs to be binarized before being fed to the Tsetlin Machine. In the Darshana (2019) paper, a pre-processing method for converting continuous variables into binary ones is described. First, the method selects a feature to work on, and the procedure can be replicated on all the features. The unique values of the continuous feature are found $\{v_1, v_2, \dots, v_u\}$ [12], These unique values are then potential thresholds for the transformation. If one takes some threshold value v_w from the set, then the condition for all the other values in the complete continuous feature set is $\leq v_w$. Because of this, the continuous values will then either pass or fail the condition and output 1 or 0, respectively.

Table 2.3 shows an example of a continuous feature being transformed into three binary features using the thresholds. The unique values in the continuous feature are $\{4.21, 7.48, 22.92\}$. The first value, (7.48), is greater than the first threshold (4.21), but the same as the second threshold (7.48) and smaller than the third (22.92). This results in the binary feature 011. When applying the same procedure, the second continuous feature becomes 001 and the fourth 111.

Continous Data	Thershold		
	≤ 4.21	≤ 7.48	≤ 22.92
7.48	0	1	1
22.92	0	0	1
7.48	0	1	1
4.21	1	1	1

Table 2.3: Thersholding continous features

2.3 Models for Comparison

For comparing the results of the Tsetlin Machine as a classifier for this thesis, a handful of other standard classifier algorithms were selected. They are some of the most common algorithms used in data science and machine learning. The following sections are some short summaries of the logic behind each of them.

2.3.1 Logistic Regression

Here we look at the basic form of logistic regression, a linear model that performs binary classification. It is a probabilistic model that outputs the probability of a sample being a positive event. A positive event, in this case, means desired output, not necessarily a good event. Raschka & Mirjaili (2017) describe the odds, which is "the odds in favour of a particular event" (p. 59). They use the form $\frac{p}{(1-p)}$ where p is the probability of the event being positive and has the label $y = 1$ [4]. The *logit* function is then defined as the logarithm of the odds:

$$logit(p) = \log \frac{p}{(1-p)}$$

If we then have a training set $\{(x_1, y_1), \dots, (x_m, y_m)\}$, we can establish the relationship between the feature values and log-odds. Because the *logit* function can transform a value in the range 0 to 1 into a real-number value, we can write [4][13]:

$$logit(p(y = 1|x)) = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{i=0}^{m} w_ix_i = \boldsymbol{w}^T\boldsymbol{x}$$

Here w is the weights of each features x. Since we are interested in the conditional probability $p(y_i = 1|x_i)$, we want the inverse form called *logistic sigmoid function.* It is this outputs the probabilities of the class of a sample given its features:

$$p(y_i = 1|x_i) = \Phi(z) = \frac{1}{1 + e^{-z}}$$

z is the linear combination of the weights and sample features, $\boldsymbol{z = w^T x}$**.** During training, it is these weights that are updated.

2.3.2 Naïve Bayes Classifier

Naïve Bayes belongs to the class of Bayesian classifiers, meaning it is a probabilistic classifier [14]. Although perhaps unrealistic, it assumes independence between the classes it is trying to predict. This simplifies the classifier greatly, and we get:

$$P(\boldsymbol{X}|y) = \prod_{i=1}^{n} P(x_i|y)$$

Where $\boldsymbol{X} = [x_1, x_2, \dots x_n]$ is the feature vector, and y is the class of the problem. To arrive at the classifier, we take Bayes theorem with the previous assumption:

$$P(y|\boldsymbol{X}) = \frac{P(y)\sum_{i=1}^{n} P(x_1, x_2, \dots, x_n|y)}{P(x_1, x_2, \dots, x_n)}$$

Because $P(x_1, x_2, \dots, x_n)$ is identical for all the classes, it can be ignored and gives the Bayes discriminant function:

$$P(y|\boldsymbol{X}) \propto P(y)\sum_{i=1}^{n} P(x_1, x_2, \dots, x_n|y)$$

Then an *argmax* function can be applied:

$$\hat{y} = argmax\, P(y) \sum_{i=1}^{n} P(x_1, x_2, \dots, x_n|y)$$

To estimate $P(y)$ and $P(x_1, x_2, \dots, x_n|y)$ a maximum a posterior probability is found. One of the main differences between the Naïve Bayes Classifiers is the assumptions of $P(\boldsymbol{X}|y)$. In the Gaussian Naïve Bayes it follows a Gaussian distribution:

$$P(\boldsymbol{X}|y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_1-\mu)^2}{2\sigma^2}}$$

2.3.3 Support Vector Machine

The support vector machine is an algorithm that seeks to maximize the margin between classes [4]. The margin is a hyperplane defined as the distance between the decision boundary and the training samples closest to the hyperplane. These samples are called support vectors. Figure 2.9 shows an example of this. This logic can be extended to many-dimensional data, and the SVM is often used for that.

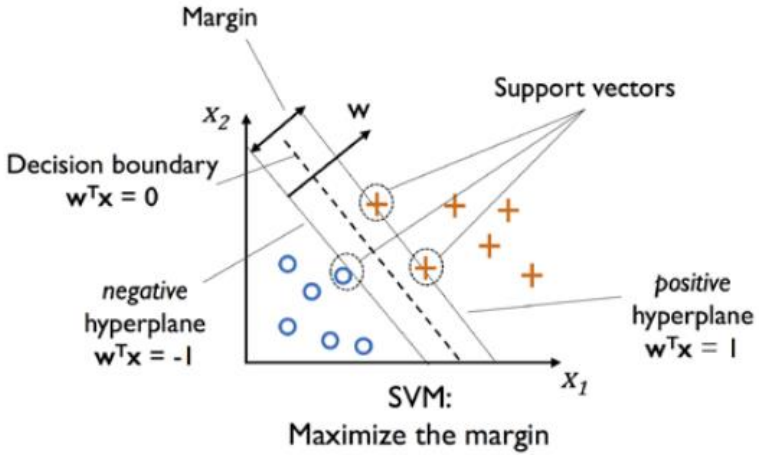

Figure 2.9: SVM maximizing the margin between the hyperplanes and decision boundary. Adapded from Raschka, S., & Mirjalili, V. (2017).

The idea of maximizing the distance from decision boundary is to increase the generalization capabilities of the model, thus decreasing overfitting. To find the best hyperplane the positive and negative parallel to the decision boundary can be written as:

$$w_0 + \boldsymbol{w^T x^+} = \mathbf{1}$$

$$w_0 + \boldsymbol{w^T x^-} = \mathbf{-1}$$

Subtracted from each other, they are:

$$\boldsymbol{w^T}(\boldsymbol{x^+} - \boldsymbol{x^-}) = \mathbf{2}$$

To find the length of $\boldsymbol{w}$, the equation is normalized:

$$\|\boldsymbol{w}\| = \sqrt{\sum_{\boldsymbol{j=1}}^{\boldsymbol{m}} \boldsymbol{w_j}}$$

We then arrive at the equation:

$$\frac{\boldsymbol{w^T}(\boldsymbol{x^+} - \boldsymbol{x^-})}{\|\boldsymbol{w}\|} = \frac{\mathbf{2}}{\|\boldsymbol{w}\|}$$

The left side of the equation can be seen as the distance between the positive and negative hyperplane. It is this distance we are trying to maximize. The task of the SVM is then to use the training samples under the restraint that they are correctly classified to maximize the margin to create the optimal decision boundary for the data.

Chapter 3 Materials

The structure of this section may seem somewhat counterintuitive. Because the analysis is at the core of the thesis, collecting new data is secondary to assessing the Tsetlin Machine. The data is rather a benchmark for the algorithm to compare with other machine learning methods. Thus, in this context, the data is a tool for comparing the different machine learning algorithms. This makes the role of the data different but not less important.

The need for substantial materials is somewhat obsolete in this set of circumstances. The dataset has a limited sample size and does not require comprehensive resources to process. Because of this, the hardware needed is somewhat standard. Also, the Tsetlin Machine has open-source code and is readily available.

This chapter describes the tools used for the thesis. Section 3.1 is a description of where the dataset was accumulated and a short description of the features created from MRI images using radiomics. After that, in section 3.2, the hardware is listed. Lastly, section 3.3 is about the software used for the thesis from API to Python libraries. It also contains a short overview of what they are used for.

3.1 Dataset

The data used in this analysis was originally part of *The OxyTarget study – Functional MRI of Hypoxia-Mediated Rectal Cancer Aggressiveness [ClinicalTrials NCT01816607]* for Akershus University hospital in 2013 [23] and was collected before this thesis. It has a limited number of patients, making the set relatively small in terms of sample size. Because of this, no high computational performance hardware should be required for the analysis, and it can be performed on most modern personal computers and laptops.

The raw MRI image data from the study were not used, but rather a set created for Langan (2020) master thesis [24] by extracting radiomic features from the MRI images. In the thesis, Langan uses the PyRadiomics [25] Python package to create these features. The data comes in two unique sets (set1 and set2). Set1 has the features extracted without any changes to the

voxels of the original images, while set2 has its voxels resampled to give a resolution of $1x1x1\ mm^2$. Both sets have three versions, T2 containing only T2-weigthet images, $T2_{b5}$ containing both T2-weigted and diffusion-weighted images (b5) and one combined of the two with all the diffusion-weighted images (b0-b6). All the files have radiomic features of three different classes: shape features, First-order statistical features and texture features. The pyRadiomic can extract many features, and for some perspective, the combined set has 772. For a more in-depth description of the features, see Langan (2020) [24].

The thesis explores several datasets. In addition to the feature sets, one additional dataset contains the target variable for the analysis. This variable is called progression-free survival (PFS) and can be true or false (Boolean 1 or 0). If the variable is true, the given patient has cancer recurrence, metastases, or death within 3 years after being included in the OxyTarget study. If it is false, the patient has survived cancer-free. Thus, the thesis seeks to use the features from the above paragraph to predict the PFS. While the original study has 110 patients, the total number of patients used was 81. This is because some of the patients withdrew their consent, and some of the PFS values are missing.

3.2 Hardware

As mentioned previously, the analysis in this thesis can be done on most modern personal computers. The experiments were run on a laptop with Intel Core i5-6300HQ processor at 2.30 GHz. The RAM of the laptop was 8.00 GB and running Windows 10 as the operating system.

3.3 Software

As the Tsetlin Machine python package requires a Linux operating system, installing a sub-system on the laptop mentioned in the previous section was necessary. Specifically, Ubuntu release 20.04.1 was used. This enabled it to run all the necessary programs for the analysis. The thesis used Python version 3.8.2, with Jupyter Notebook version 6.1.4. As for libraries

used, they were Numpy [17] version 1.19.2, Pandas [18] version 1.1.3, Matplotlib [19] version 3.3.2, Scikit-learn [20] version 0.23.2 and pyTsetlinMachine [21]. All these packages are, as mentioned previously, open-source and can be accessed by anyone using pip3 or installing a distribution package such as Anaconda [22]. The packages were installed separately using pip3 inside the Ubuntu terminal to create a separate environment to work within.

These packages combined are used to perform the different operation needed to do the analysis. They handle things such as holding the data in arrays and doing mathematical computation. It enables Python to do the calculations with greater efficiency. Matplotlib handles the plots so that the data and the results can be visualized. Scikit-learn and pyTsetlinMachine are packages explicitly used for data analysis and machine learning. The packages will be described in more detail in chapter 4: methods. There it will be explained how they are used for the analysis.

Chapter 4 Methods

This chapter will describe the methodology of the analysis. Section 4.1 describes the pre-processing. It includes data handling and RENT used for feature selection. Section 4.1 covers the training of models and finding the optimal hyperparameters.

4.1 Pre-processing

The Tsetlin Machine is the basis for how the pre-processing is done. In the case of the other models requiring additional pre-processing, this is specified.

4.1.1 Data Handling

To contain and handle the data, the Python package *Pandas* [26] was used. *Pandas* is a library for handling and creating data structures for statistical computing for scientific use. When having structured data such as CSV-file, Pandas is advantageous for manipulating and working with data in Python. It can open the file directly from the directory and convert it into a DataFrame, a type of array that has additional functionality. One example is allowing the naming of the columns, which is practical when working with named features.

Because the data is, as mentioned in section 3.1, separated into different files, the target values and features had to be merged. The total number of samples in the features and target sets did not match up because some of the patients did not have PFS value assigned, so this had to be matched to proceed. This was done by matching the patient's ID in the different files, which was done using Pandas. The new datasets now consist of only the samples with both the radiomic features and progression-free survival values. For example, one of the DataFrames with the combined data contained 81 rows of patients and 773 columns of radiomic features.

4.1.2 RENT for Feature Selection

Jenul, A. (2020) explains that for reducing the complexity of models when training on datasets with many features, it is essential to perform feature selection [27]. Repeated elastic net technique (RENT) is the feature selection method used in this thesis as part of the analysis. RENT can be used for both regression and binary classification problems. It is not only focused on improving the predictive performance but also gives information about the stability of the process of selecting features. The feature selection is an assemble of elastic net regularized models trained on unique subsets of the original data. The regularization is done using a combination of L1 and L2 regularization. Because of this, two of the parameters that must be set prior to the feature selection is the ratio between L1 and L2 (called L1 ratio in the package) and the regularization strength (called C in the package). If given multiple values, RENT will perform a cross-validated grid search with these parameters. As seen in figure 4.1, RENT trains K models on unique subsets of the training data, where K is an input parameter that the user can set.

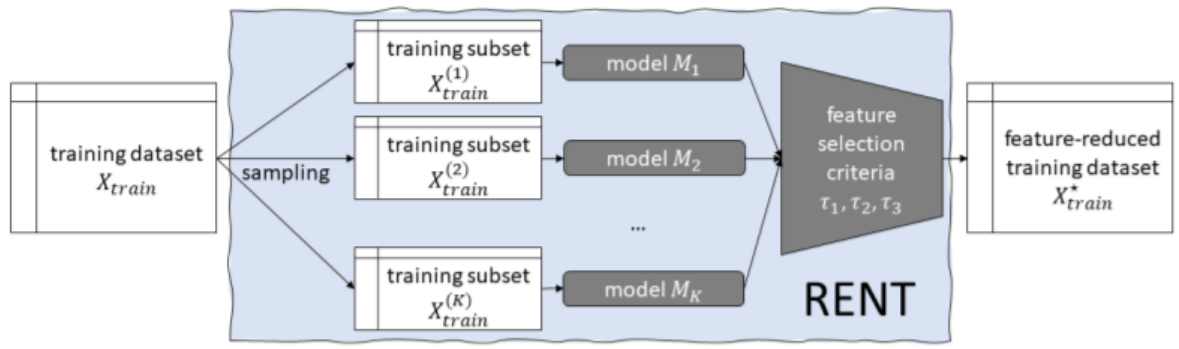

Figure 4.1: "The scheme depicts the feature selection pipeline suggested by RENT, represented by the blue frame." From Jenul, A. (2020)

For the feature selection, the relevant information of the importance of the features are gained across all the models and saved as a matrix [27]. There are then three criteria that can be set, which all must be met for a feature to be selected, τ_1, τ_2, τ_3. For each feature f_n $(n = 1, \dots, N)$ from the training set, there are weights $\beta_{k,n}$ that are trained per model M_k $(k = 1, \dots, \mathrm{K})$. Every feature f_n thus has a feature importance vector $\beta_n = (\beta_{1,n}, \dots, \beta_{K,n})$.The $\tau_1$is

then a measure of relevancy of the feature taking the average frequency $c(\beta_n)$ across the models by finding the percentage of non-zero parameters for f_n:

$$c(\beta_n) = \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{[\beta_{k,n}\neq 0]}$$

Two other summary statistics are calculated, which are the feature-specific mean and variance of the feature weights:

$$\mu(\beta_n) = \frac{1}{K}\sum_{k=1}^{K} \beta_{k,n}$$

$$\sigma^2(\beta_n) = \frac{1}{K}\sum_{k=1}^{K} (\beta_{k,n} - \mu(\beta_n))^2$$

Jenul, A. (2020) describes that a feature f_n is selected if:

1. It has a high score $c(\beta_n)$ that beats a set threshold.
2. The feature does not alternate between positive and negative sign more than a set threshold.
3. It has high non-zero model parameter estimates consistently over the K models and that they have low variance.

Mathematically the criteria are defined as:

$$\tau_1(\beta_n) = c(\beta_n)$$

$$\tau_2(\beta_n) = \frac{1}{K}\left|\sum_{k=1}^{K} sign(\beta_{k,n} \neq 0)\right|$$

$$\tau_3(\beta_n) = t_{K-1}\left(\frac{\mu(\beta_n)}{\sqrt{\frac{\sigma^2(\beta_n)}{K}}}\right)$$

Where the function t_{K-1} is the cumulative density of the Student's t-distribution with K − 1 degrees of freedom. These criteria can be thresholded on a scale of [0,1] so that the user decides how strict the demands are for a feature to be selected. The Python package also has other functions, such as the function called object summary that gives information about how many times the samples has been part of the test set, as well as the number of times it was incorrectly classified by RENT while part of the test set and percentage incorrectly predicted.

4.2 Training Models

For the Tsetlin Machine, the pyTsetlinMachine [21] was used. Logistic Regression, SVM and Naïve Bayes were used as models for comparison. They were from the Python library scikit-learn. However, since the Tsetlin Machine was not implemented in scikit-learn, the methodology usually used when training could not be used, such as automated grid search and other similar tools. To get the most comparable results, all the models were trained using the same framework.

4.2.1 Scikit-learn Stratified 4-fold

The stratified K-fold method from Scikit-learn is a type of K-fold data splitting used to validate the performance of both the RENT analysis and the models used for comparison. Each fold is created such that they contain approximately the same percentage of each class in the complete dataset. When running it, four folds was selected, and the method provides the indices for each fold.

4.2.2 Hyperparameter Searching

The hyperparameter searching was done using a grid search by looping hyperparameters for each algorithm to find the best performance. For the Tsetlin Machine *T*, *s* and *number of clauses* were explored, the logistic regression searched *solver* and *C* hyperparameter, SVM searched *kernel* and *gamma* and lastly, Naïve Bayes did not have hyperparameters to search. The searches were organized in a grid where the intervals for the values of the hyperparameters could be set. For each setting of hyperparameters per algorithm, a 4-fold validation taking an average of the scores was used. This was done so that the performance would be more robust. Because of the small number of samples, these best results were used to compare the models.

Chapter 5 Results

Figure 5.1 shows a visualization of the workflow for producing the results. This flow was used for both set1 and set2, as well as their respective versions. The workflow was set up as shown in figure 5.1 to work around the low number of samples and get validated results from RENT and the models. In the step to produce the F1 and MCC, each algorithm was trained four times using stratified 4-fold to validate the metrics.

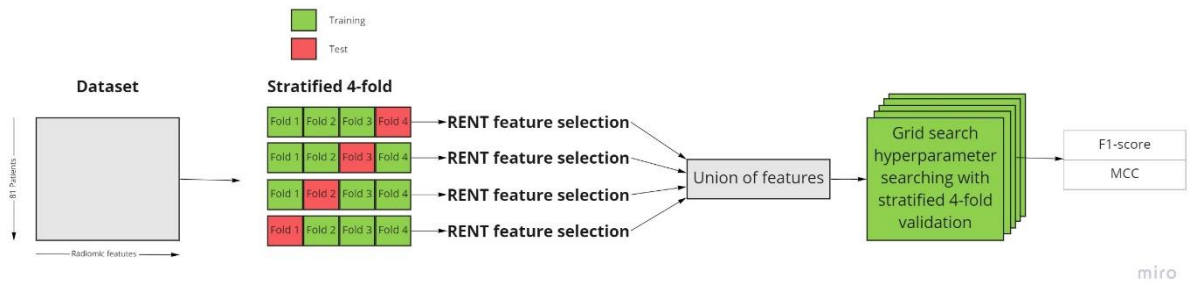

Figure 5.1: The workflow for producing the results. The same workflow was used for all the datasets.

This chapter is structured by first looking at set1 and following the workflow. Results from RENT first and then the results from the training and hyperparameter search of the models. This set is described more in-depth as it was the focus of the analysis, while set2 was mainly used for additional information to potentially get more information about PFS from RENT and the models. Set2 follows the same structure but not as detailed. The chapter also has a section about the clauses and literals that the Tsetlin Machine learns.

5.1 Dataset 1: T2, T2b5 and combined

5.1.1 RENT feature selection

For set 1, all three versions were run through the same pipeline to identify what MCC and F1-score they could achieve. The first step was feature selection using RENT, described in 4.2.2. When selecting features, RENT was run on 3 out of 4 stratified folds and tested on the remaining fold to get more robust predictive performance. In practice, that meant running RENT four times on each set to get variations in training and test splits and get a more robust selection. The C parameter, l1-ratio and cut-off was set to the same for every time RENT was run. Table 5.1 shows what they were set to.

C parameter	0.9
L1 ratio	0.5
Cut-off: τ_1, τ_2, τ_3	0.5, 0.5, 0.5

Table 5.1: RENT parameter settings

When using splits, there is a choice between using the intersection of features where the features selected over all the splits are used or the union of features, where using the features that have been selected at least once. Due to the timeframe of the thesis, union was used, but the intersection of features can be explored in future work. Table 5.2 shows the original number of features as well as the selected number with the setting mentioned above.

Set 1 version	Original feature number	Selected feature number
T2	107	46
T2b5	214	57
Combined	772	103

Table 5.2: Number of selected features on the 81 patients

5.1.2 Scores from Hyperparameter tuning

The models were all trained and tested on every version of set1. A hyperparameter search per version of the set with the selected features for each version was also performed. The best score of each model was used to compare. All the models were run through a grid search, except for Naïve Bayes, which do not have hyperparameters to tune. For more robust results, each combination was also tested on a 4-fold stratified split taking the average of the folds. This was done to try to get a more robust estimate of the performance across 4 splits.

Model	F1-score	MCC
Tsetlin Machine	0.49	0.08
Logistic Regression	**0.56**	**0.23**
SVM	0.55	0.20
Naïve Bayes	0.38	-0.09

Table 5.3: F1-score and MCC for set 1 T2

Table 5.3 shows the results of all the classifiers on the T2 set. Something important to note is that some of the models, in this case, did not get the best F1-score and MCC with the same hyperparameter settings. For example, the results from the Tsetlin Machine got the F1 = 0.49 with 850 clauses, T = 20 and s = 10, but got MCC = 0.08 with 100 clauses T = 140 and s = 1.1. What this could mean will be discussed in chapter 6.

Model	F1-score	MCC
Tsetlin Machine	0.59	0.29
Logistic Regression	0.58	0.22
SVM	**0.63**	**0.33**
Naïve Bayes	0.53	0.07

Table 5.4: F1-score and MCC for set 1 T2b2

For the T2b5 version, table 5.4 contains the results. It has the same issue as the results of T2. Some of the models do not have the same parameters for both metrics. For the Tsetlin Machine, it is only the s that varies, with F1 scoring best with s = 3.3 and MCC best at s = 7.8. For this set version, the logistic regression also does this, with C = 10 for the best MCC and C = 4.3 for the best F1.

Table 5.5 shows the results for the combined version of set 1. For this set version, the MCC and F1-scores had the same hyperparameters. One interesting detail to note is that for the logistic regression, C could range from 0.8 to 9.7 and still get the same results as displayed in table 5.5.

Model	F1-score	MCC
Tsetlin Machine	0.62	**0.39**
Logistic Regression	0.64	0.32
SVM	**0.65**	0.38
Naïve Bayes	0.63	0.25

Table 5.5: F1-score and MCC for set 1 combined

5.2 Dataset 2: T2, T2b5 and combined

5.2.1 RENT feature selection

The same workflow for set1 was used for set2 because they had the same features with only different resolutions. The second set was used to find out if there were additional information to be gathered to predict the response variable. For the RENT feature selection, the parameters as in table 5.1 were used, resulting in the dimension reduction as in table 5.6. As one can see, the number of selected features is similar to the feature selection on dataset 1.

Set 2 version	Original feature number	Selected feature number
T2	107	49
T2b5	214	53
Combined	772	107

Table 5.6: Number of selected features on the 81 patients

5.2.2 Scores from Hyperparameter tuning

Using the number of features from table 5.6 feature selection and using the same grid search method as set1 provided the results shown in table 5.7-5.9.

Model	F1-score	MCC
Tsetlin Machine	**0.48**	**0.12**
Logistic Regression	0.42	0.01
SVM	0.47	0.06
Naïve Bayes	0.42	-0.06

Table 5.7: F1-score and MCC for set 2 T2

Model	F1-score	MCC
Tsetlin Machine	**0.55**	**0.20**
Logistic Regression	0.53	0.18
SVM	0.52	0.17
Naïve Bayes	0.54	0.05

Table 5.8: F1-score and MCC for set 2 T2b5

Model	F1-score	MCC
Tsetlin Machine	**0.68**	**0.42**
Logistic Regression	0.66	0.34
SVM	0.63	0.37
Naïve Bayes	0.63	0.25

Table 5.9: F1-score and MCC for set 2 combined

5.3 Printing Clauses for Interpretability

With some additional code, it was possible to print the clauses of the Tsetlin Machine model. These are the sub-patterns it has learned during training. Figure 5.1 shows some of these clauses. They are literal form from the binarized input data, being displayed as conjunctive literal statements. They still need work to be able to interpret them regarding the continuous features from the datasets. Since the literals are based on binary features, it is necessary to identify distinct values which the literals represent in the continuous features to return them to the original features.

```
Class 0 Positive Clauses:

Clause #0:   x3027 ∧  x4710 ∧ ¬x0 ∧ ¬x1 ∧ ¬x2 ∧ ¬x3 ∧ ¬x4 ∧ ¬x5 ∧ ¬x6 ∧ ¬x7 ∧ ¬x8 ∧ ¬x9 ∧ ¬x1212 ∧ ¬x1343 ∧ ¬x2235 ∧ ¬x3918 ∧ ¬
x5568
Clause #2:   x2944 ∧ ¬x0 ∧ ¬x1 ∧ ¬x2 ∧ ¬x3 ∧ ¬x4 ∧ ¬x5 ∧ ¬x6 ∧ ¬x7 ∧ ¬x8 ∧ ¬x9 ∧ ¬x114 ∧ ¬x149 ∧ ¬x755 ∧ ¬x993 ∧ ¬x1018 ∧ ¬x102
4 ∧ ¬x1289 ∧ ¬x1512 ∧ ¬x1597 ∧ ¬x1801 ∧ ¬x1866 ∧ ¬x2165 ∧ ¬x2192 ∧ ¬x2701 ∧ ¬x2816 ∧ ¬x3061 ∧ ¬x3275 ∧ ¬x3927 ∧ ¬x3992 ∧ ¬x4050
∧ ¬x4342 ∧ ¬x4534 ∧ ¬x4891 ∧ ¬x4912 ∧ ¬x4913 ∧ ¬x5071 ∧ ¬x5163 ∧ ¬x5334 ∧ ¬x5459 ∧ ¬x5571
Clause #4:   x4677 ∧ ¬x0 ∧ ¬x1 ∧ ¬x2 ∧ ¬x3 ∧ ¬x4 ∧ ¬x5 ∧ ¬x6 ∧ ¬x7 ∧ ¬x8 ∧ ¬x9 ∧ ¬x422 ∧ ¬x1156 ∧ ¬x1228 ∧ ¬x1571 ∧ ¬x1687 ∧ ¬x
1842 ∧ ¬x3532 ∧ ¬x3770
Clause #6:   x1167 ∧  x1260 ∧  x3084 ∧  x4808 ∧  x4902 ∧  x5114 ∧  x5562 ∧  x5918 ∧ ¬x0 ∧ ¬x1 ∧ ¬x2 ∧ ¬x3 ∧ ¬x4 ∧ ¬x5 ∧ ¬x6 ∧ ¬
x7 ∧ ¬x8 ∧ ¬x9 ∧ ¬x18 ∧ ¬x105 ∧ ¬x367 ∧ ¬x609 ∧ ¬x1083 ∧ ¬x1294 ∧ ¬x2407 ∧ ¬x3112 ∧ ¬x3146 ∧ ¬x3247 ∧ ¬x3512 ∧ ¬x3595 ∧ ¬x4097
∧ ¬x4653 ∧ ¬x4654 ∧ ¬x5807
```

Figure 5.2: Four of the Class 0 Positive clauses generated from the set 1 combined dataset

5.4 RENT object summary

Figure 5.2 displays RENT object summary of 7 of the 81 samples from one of the RENT analyses. The patient is in numbered order, and the missing patients are in the fold used for testing. It shows how many times a sample has been in the test split when running RENT for 100 repeated models. Looking at patient 3 in table 5.10, the first column shows that it has been part of the test set during the training of the models in RENT. The second show that the actual class of the patient is 0. The two last columns show that the number of times the class has been incorrectly predicted is 40 and that it was mispredicted 97.6% of the time. Patient 10 has done considerably better, with only five times being incorrect out of the 39 times it has been part of the test set. What this might imply for the results will be discussed in chapter 6.

	# test	class	# incorrect	% incorrect
2	42	0.0	32	76.190476
3	41	0.0	40	97.560976
5	42	0.0	18	42.857143
6	42	0.0	17	40.476190
7	34	0.0	17	50.000000
9	44	1.0	39	88.636364
10	39	0.0	5	12.820513

Table 5.10: Excerpt of some of the samples and how often the models in RENT incorrectly predicted the samples.

Chapter 6 Discussion

6.1 Datasets

The quality of the data was a slight drawback of the project. As were reflected in the results in chapter 5, neither RENT nor the models provided stable high scores. It can seem like the information in the feature are not sufficient to give good predictions of the class. What this meant for the analysis was simply to lower the expectations for the scores and instead look at comparisons of the models. Even though set 2 was not the focus point of the analysis, it was valuable in giving more F1 scores and MCC for making the model comparisons more robust. The differences in the scores from the sets and the version in the sets might come from the differences in features precent from the radiomic features extractions and difference in resolutions.

Perhaps the most significant factor in making pattern recognition difficult with the set is the sample size versus the feature space. As described in chapter 3: materials, even the smallest datasets have more features than samples. The number of usable patients being 81 is also not particularly favourable. When working with datasets like these, the classification results might be statistically questionable even with feature selection [28]. These caveats must be considered when working with the data, especially since they are healthcare data.

6.2 RENT feature Selection

To work around the limited sample size, 4-fold stratified splitting was from Scikit-learn used. Using a normal test and training split, both RENT and the models were sensitive to how the data was split. This could lead to quite varying results, indicating that they could be misleading. For this reason, the average performance of the 4-fold splits was used to get a more robust estimate of the performance. The split sensitivity can also be seen in the validation studies generated by RENT. Figure 6.1 and 6.2 show a validation study from two different folds on set 1 Combined. The validation is done on the testing fold from the 4-fold

splits left out when training. It uses the selected features from the training and tests the predicative performance, comparing it to VS1 and VS2. VS1 selects randomly from all the features and represents an inefficient feature selection that does not take any predive information into account. This is done a default of 100 times, and the blue curve shows the distribution of scores.VS2 trains a logistic regression model on the test data based on the features selected by RENT and randomly permutes the class label. It does this a default of 100 times, and the green curve shows the distribution of the scores. Lastly, the red line is the score of a logistic regression model trained with all the data using the selected features from RENT. As seen in the two figures, the different splits have different performances. This was the reasoning behind using four folds to do four analysis because it gave a more varied look at the feature selection on the data. The potential drawback of using this method is that the generalization of the whole process is unknown.

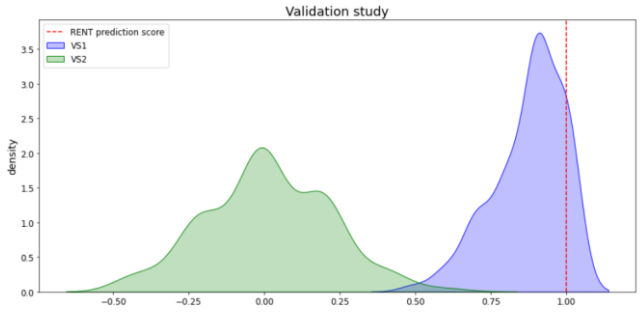

Figure 6.1: RENT Analysis validation study for split 2 of set 1 Combined.

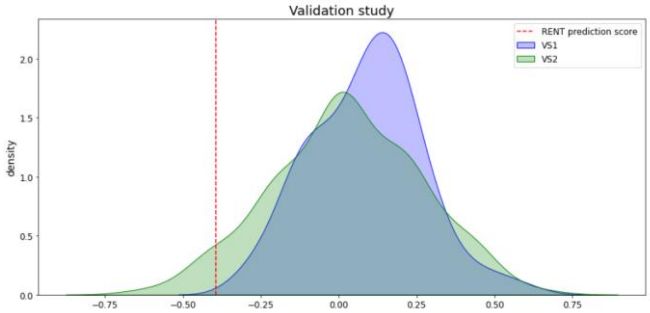

Figure 6.2: RENT Analysis validation study for split 3 of set 1 Combined.

Even though the data has some drawback, it still responded somewhat well to feature selection. When trying the models without RENT feature selection with the complete set of

772 features, the performance from all the models was around the same as guessing (around MCC = 0). This meant that RENT feature selection was able to improve the performance of the models to what they are listed as in the results section. Perhaps this indicates that the RENT could remove some of the noise in the data, revealing patterns. It would seem feature selection was an essential part of the workflow, maybe even the most vital. Trying different methods for feature selection could be interesting to see if it would select the same features as RENT and if there are differences in performance. Maybe another set of features could reveal more. However, this is something for further work.

Something note about RENT is that it is based on logistic regression. One can think that this makes the selected features more tuned to logistic regression as a classifier and maybe do not improve performance for other algorithms. This is speculative though, and more evidence is needed to state something conclusive about it.

6.3 Set1: Comparing Tsetlin Machine to Other Models

From how predictive models are used in healthcare, the overall performances of the models of this analysis must become higher than they are currently. As seen in the results, the best performances for set1 were achieved with the combined version. This version had F1-scores over 0.6 for all the models and had the highest MCCs. The best F1-score was 0.65 from the SVM. The best MCC was from the Tsetlin Machine on the same version, achieving 0.33. It would be fair to say that these are modest numbers at best, however not directly poor. In general, the models scored roughly the same T2, T2b2 and combined. Naïve Bayes was a bit behind on T2 and T2b2. The Tsetlin Machine also had poor performance on the T2 for the MCC. However, perhaps the most interesting result is that the Tsetlin Machine scores around the same as the other models for F1 and MCC (apart from T2). This might indicate that no one algorithm performs overall better or worse. In these cases, other advantages such as lower runtime or better interpretability might be the deciding factor in selecting what algorithm to use ultimately. The interpretability of the results will be discussed in section 6.5.

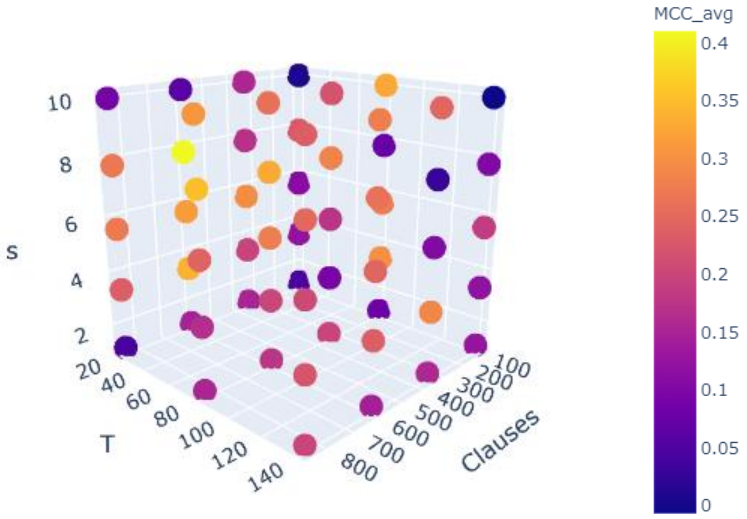

Figure 6.3: Grid showing TM results Set1 combined.

The results come with a caveat, and that is the Tsetlin Machine got its best F1-score and best MCC with different hyperparameters with T2 and T2b5. Figure 6.3 shows a grid of one of the hyperparameters MCC with scores. As one can see, there is not one apparent pattern for when the scores improve or get worse. It can seem that a T = 20 gives somewhat better scores. Since there are so few samples, this can affect the model to make it less robust towards generalization and make it harder to find patterns.

6.4 Set2: Comparing Tsetlin Machine to Other Models

The results for set 2 seem to overall be similar to the results for set 1. The most significant difference is perhaps that the F1 scores for all the models lie close to each other, with no one sticking out. One interesting note is that the Tsetlin Machine outperforms the other models for all the versions of set 2 in both F1-score and MCC. The combined set 2 had the best overall score, with the Tsetlin Machine scoring 0.68 for F1-score and 0.42 for MCC. This adds to the notion that it can compete with the other models.

6.5 Interpretability of the Tsetlin Machine

An important factor of what makes the TM promising is the interpretability it provides with clauses in the form of propositional formulas. Potentially this gives an insight into which patterns are important for predicting the PFS variable. For healthcare data such as those in this analysis, these patterns could give important insight into the disease. As seen in Figure 5.1, it is possible to print the clauses for the trained Tsetlin Machine model with some additional code. This is a good starting point for interpreting the decision-making in the model, but the problem is that it is not quite human-readable. In terms of the original features of the dataset, it can be challenging to understand what the clauses indicate without calculating the literals back to a continuous scale. This is because the clauses are binary, and the data is continuous. What this means for this analysis is that there is a step missing in the automation of this process to take full advantage of the information provided by the Tsetlin Machine. To make the results more human-readable, a step should be included to convert the binary clauses back into continuous values and features. This could give individuals, such as doctors, without the knowledge of the Tsetlin Machine a better understanding of the outputs of potential new patients. One would only have to know the data one is working with, and the Tsetlin Machine would provide the patterns in the data that makes it a given class.

6.6 Other Notions

One thing to note about the data quality is the results of the object summary from RENT seen in figure 5.2, is the number of samples it got wrong a high percentage of the time. During its analysis of the RENT ensemble of models predicted many of the samples wrong, and this might indicate that the amount of relevant systematic information of the data might affect the results of the models. The effect is most likely that the wrongly predicted samples add noise to the data and makes it harder for the models to recognize patterns. Removing these samples would most likely yield improved performance for the models but at the cost of generalization. Because if they are removed, a similar process must be done for potential new samples. Deciding if a sample is relevant for the model when collecting it might be difficult.

Chapter 7 Conclusion

Based on the results from the four algorithms applied in this thesis, it seems that in this project, the Tsetlin Machine can at least compete with the other models when working with the wide dataset of radiomic features from MRI images. However, this is with the note that all the models do not score very high. However, the scores are at least relatively consistent. It must also be added that in order to get better performance from the models on the data, they were dependent on using RENT feature selection to reduce the feature dimension space to remove non-informative features. The low number of samples compared to the high number of features gave rise to results that were difficult to validate with high certainty, so the 4-fold cross-validation used tried to increase the robustness of the validation. Considering these factors, the fact that the Tsetlin Machine generally was able to compete is confirmation of its usefulness.

For the explainable part of the Tsetlin Machine, in terms of continuous data, it still needs a bit of work to be used in a practical way. The literals and clauses produced are still a not straightforward for human interpretability since the propositional formula may be relatively complex. However, it should also be stated that the fact that the Tsetlin Machine can even produce these clauses, which are the sub-patterns it uses for decision making, is promising. This could be reasoning for use in medicine in the near future if the translation of the propositional formula from literals back to values on a continuous scale would be automated.

7.1 Further Work

Most likely, this thesis has not utilized the full potential of the Tsetlin Machine. To increase the performance, one could explore hyperparameter tuning more to see if this can lead to better performance. When exploring this, one can look at the features from the analysis and the number of bits per feature when binarizing the data. There might be a possibility of finding better performance through an extensive search. Additionally, there is an option for weighted clauses. This version of the Tsetlin Machine weighs clauses so that it needs fewer clauses to reinforce patterns [29], which require different settings of hyperparameters s, T and number of clauses. One can also look at different methods for feature selection to see how much impact finding the right features have on the scores. Another option to explore to work around the limited number of samples is trying techniques to synthetically increase the number of samples. This might help the Tsetlin Machine, and the other models enforce the patterns found in the data. The training scores should perhaps have been included to see if the models were overfitting or the information in the data was difficult to predict.

It would also be interesting to look at the running time and memory usage of the Tsetlin Machine and compare it to the other algorithms used in this thesis.

Bibliography

[1] Bray, F., Ferlay, J., Soerjomataram, I., Siegel, R. L., Torre, L. A., & Jemal, A. (2018). Global cancer statistics 2018: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: a cancer journal for clinicians*, *68*(6), 394-424.

[2] The Acredit Network, (n.d.), *About Colorectal Cancer*. https://www.acredit.no/about-colorectal-cancer-2/

[3] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, *1*(5), 206-215.

[4] Raschka, S., & Mirjalili, V. (2017). Python Machine Learning: Machine Learning and Deep Learning with Python. *Scikit-Learn, and TensorFlow. Second edition ed*. Packt Publishing Ltd.

[5] Osborne, M. J., & Rubinstein, A. (1994). *A course in game theory*. MIT press.

[6] Morgenstern, O., & Von Neumann, J. (1953). *Theory of games and economic behavior*. Princeton university press.

[7] Granmo, O. C. (2018). The Tsetlin machine-a game theoretic bandit driven approach to optimal pattern recognition with propositional logic. *arXiv preprint arXiv:1804.01508*.

[8] J. C. Gittins, 1979, *Bandit Processes and Dynamic Allocation Indices,* Journal of the Royal Statistical Society. Series B (Methodological), Vol. 41, No. 2, pp. 148-177

[9] Sima, C., & Dougherty, E. R. (2008). The peaking phenomenon in the presence of feature-selection. *Pattern Recognition Letters*, *29*(11), 1667-1674.

[10] Büning, H. K., & Lettmann, T. (1999). *Propositional logic: deduction and algorithms* (Vol. 48). Cambridge University Press.

[11] Narendra, K. S., & Thathachar, M. A. (1974). Learning automata-a survey. *IEEE Transactions on systems, man, and cybernetics*, (4), 323-334.

[12] Darshana Abeyrathna, K., Granmo, O. C., Zhang, X., & Goodwin, M. (2019). A Scheme for Continuous Input to the Tsetlin Machine with Applications to Forecasting Disease Outbreaks. *arXiv e-prints*, arXiv-1905.

[13] Bai, Y. Q., & Shen, K. J. (2016). Alternating Direction Method of Multipliers for $\ell_1 - \ell_2$-Regularized Logistic Regression Model. *Journal of the Operations Research Society of China*, *4*(2), 243-253.

[14] Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, No. 22, pp. 41-46).

[15] Chicco, D., & Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics*, *21*(1), 1-13.

[16] Powers, D. M. (2020). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*.

[17] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357-362.

[18] McKinney, W. (2010, June). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51-56).

[19] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *IEEE Annals of the History of Computing*, *9*(03), 90-95.

[20] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, *12*, 2825-2830.

[21] O.-C. Granmo, 2020, pyTsetlinMachine [internet], Grimstad: Centre for Artificial Intelligence Research; 2020 [retrieved 02.12.2020], Available from: https://github.com/cair/pyTsetlinMachine

[22] *Anaconda Software Distribution*. Computer software. Vers. 2-2.4.0. Anaconda, Nov. 2016. Web. <https://anaconda.com>.

[23] Kathrine R. Redalen. Functional MRI of Hypoxia-mediated Rectal Cancer Aggressiveness (OxyTarget). (2013). url: https://clinicaltrials. gov/ct2/show/NCT01816607 (visited 05.04.2021).

[24] Aase M. L., (2018), MRI-Based Radiomics Analysis for Predicting Treatment Outcome in Rectal Cancer, NTNU, Faculty of Natural Sciences

[25] Pyradiomics Community. Pyradiomics Documentation. Release v3.0.post2+g896682d. 2020

[26] McKinney, W. (2010, June). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51-56).

[27] Jenul, A., Schrunner, S., Liland, K. H., Indahl, U. G., Futsaether, C. M., & Tomic, O. (2020). RENT--Repeated Elastic Net Technique for Feature Selection. *arXiv preprint arXiv:2009.12780*.

[28] Somorjai, R. L., Dolenko, B., & Baumgartner, R. (2003). Class prediction and discovery using gene microarray and proteomics mass spectroscopy data: curses, caveats, cautions. *Bioinformatics*, *19*(12), 1484-1491.

[29] Phoulady, A., Granmo, O. C., Gorji, S. R., & Phoulady, H. A. (2019). The weighted tsetlin machine: compressed representations with weighted clauses. *arXiv preprint arXiv:1911.12607*.

Appendix

	# test	class	# incorrect	% incorrect
2	39	0.0	31	79.487179
3	44	0.0	43	97.727273
5	39	0.0	14	35.897436
6	36	0.0	15	41.666667
7	33	0.0	17	51.515152
9	41	1.0	36	87.804878
10	38	0.0	6	15.789474
11	34	0.0	22	64.705882
13	34	1.0	13	38.235294
15	42	0.0	15	35.714286
16	41	0.0	6	14.634146
17	43	0.0	11	25.581395
19	39	0.0	16	41.025641
20	32	1.0	4	12.500000
21	43	1.0	0	0.000000
22	38	1.0	29	76.315789
23	41	0.0	18	43.902439
24	39	0.0	6	15.384615
25	49	1.0	14	28.571429
26	42	1.0	33	78.571429
27	36	0.0	7	19.444444
28	26	1.0	9	34.615385
29	39	0.0	6	15.384615
30	37	0.0	15	40.540541
34	39	1.0	24	61.538462
35	39	0.0	17	43.589744
36	29	0.0	9	31.034483
37	45	1.0	2	4.444444
38	29	0.0	13	44.827586
39	38	1.0	21	55.263158
40	31	1.0	22	70.967742
42	36	0.0	27	75.000000
43	25	1.0	0	0.000000
44	37	1.0	3	8.108108
46	28	1.0	5	17.857143
47	45	1.0	9	20.000000
48	35	0.0	19	54.285714
49	36	0.0	21	58.333333
50	30	1.0	7	23.333333
51	31	0.0	9	29.032258
52	39	1.0	39	100.000000
54	30	0.0	13	43.333333
55	40	1.0	8	20.000000
56	39	1.0	23	58.974359
63	49	1.0	49	100.000000
64	34	1.0	7	20.588235
66	37	0.0	37	100.000000
67	36	1.0	18	50.000000
68	37	0.0	14	37.837838
69	39	0.0	5	12.820513
70	40	0.0	16	40.000000
71	44	1.0	16	36.363636
72	36	1.0	17	47.222222
73	31	0.0	16	51.612903
74	40	0.0	19	47.500000
75	52	0.0	3	5.769231
76	38	1.0	17	44.736842
77	40	1.0	39	97.500000
78	38	0.0	33	86.842105
80	40	0.0	18	45.000000

Figure 0.1:Full object summary from RENT on split 1.

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway