Chapter 1

Your First Tsetlin Machine

In this chapter, you build your first Tsetlin machine. Using toy data on vehicles and real-life data on breast cancer, you learn how Tsetlin machines unify three powerful strategies for pattern recognition:

1. First, you explore how to use simple if-then rules to recognize regularities in data – so-called frequent patterns. Frequent patterns capture knowledge like: "most cars have four wheels".
2. Next, you discover how to distinguish between different types of objects. For instance, both cars and planes may have four wheels. However, planes have wings, while cars do not. You will find out how Tsetlin machines learn patterns that are both frequent and able to differentiate, demonstrated on the vehicle- and breast cancer data.
3. Finally, you uncover how a Tsetlin machine dissects the data to construct multiple rules, making them interact to predict breast cancer.

At the end of this chapter, you know how a standard Tsetlin machine works. You have also learned how to implement one.

The rest of the book introduces several variants of the above mechanisms, like regression and convolution. After mastering Chapter 1, those variations will be easy to pick up because they all build on the standard Tsetlin machine.

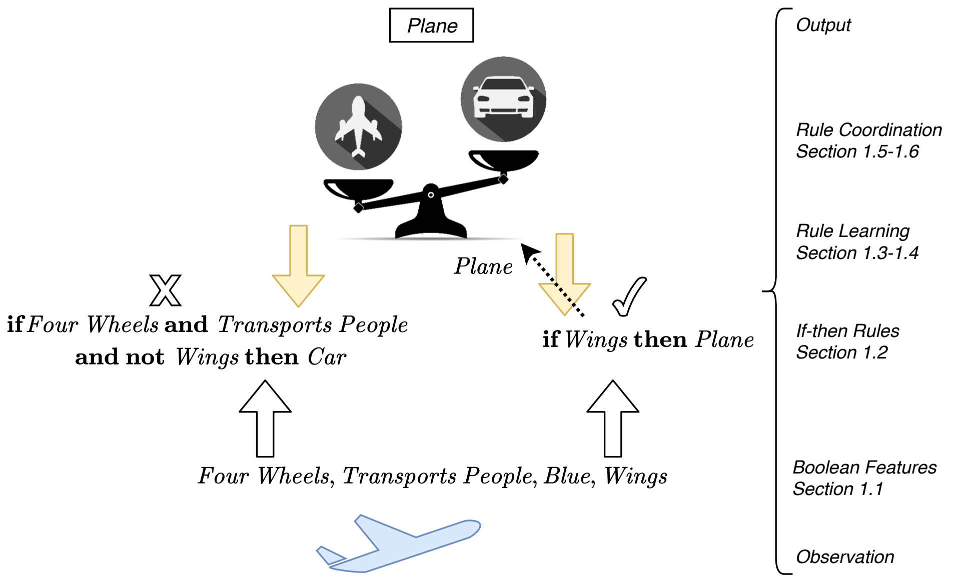
Plane
Output
Rule Coordination Section 1.5-1.6
Rule Learning Section 1.3-1.4
Plane
if *Four Wheels* and *Transports People* and not *Wings* then *Car*
if *Wings* then *Plane*
If-then Rules Section 1.2
Four Wheels, *Transports People*, *Blue*, *Wings*
Boolean Features Section 1.1
Observation

Figure 1.1: Overview of Tsetlin machine and chapter structure.

Figure 1.1 shows the overall structure of a Tsetlin machine, which you explore in Chapter 1. First, in Section 1.1, you learn how to prepare the input to a Tsetlin machine using so-called *Boolean features.* Then you discover how to describe patterns with if-then rules in Section 1.2. Using the if-then rules, Section 1.3 explains how you uncover frequent patterns in data, while Section 1.4 covers how to make the if-then rules discriminative. Finally, you study the complete Tsetlin machine algorithm in Section 1.5 and Section 1.6, investigating how the Tsetlin machine combines multiple if-then rules.

1.1 Boolean Features

A Tsetlin machine observes objects (or phenomena), which it recognizes based on the objects' features. Each feature is an object property that either is *True* or *False*, making it *Boolean.* Consider, for example, a database on vehicles of different types, like cars, boats, trains, and planes. Vehicles of the same kind will naturally have some properties in common. The cars depicted in Figure 1.2, for instance, all have four wheels, lack wings, and transport people. At the same time, they are different in various ways. One is a sedan, the others are a pickup, an SUV, and a van. They also appear in different colors. All these properties are examples of Boolean features. For instance, a vehicle is either yellow or not yellow.

Table 1.1 lists six example vehicles and their five Boolean features: *Four Wheels*, *Transports People*, *Wings*, *Yellow*, and *Blue*. There is one column per feature, each entry taking the value *True* or *False*, denoted '•' and '·',

Tsetlin Machine Origins

The Tsetlin machine is named after Michael Lvovitch Tsetlin (22 September 1924 – 30 May 1966), a Soviet mathematician and physicist. Tsetlin invented a learning algorithm that optimizes its behavior online, embedded in a random environment. Tsetlin's algorithm is particularly fascinating because it keeps track of previous experience only using a single integer. Further, it is adaptive, capable of "changing its mind". Based on this algorithm, Tsetlin introduced algorithm collectives for solving complex problems. The Tsetlin machine builds upon the work of Tsetlin, dealing with advanced pattern recognition tasks.

Machine Learning Basics

Machine learning deals with how computers can improve from experience and what constitutes the fundamental laws of learning. A typical machine learning system consists of:

- *A **task** such as classification or regression, for instance, photo classification or breast cancer recurrence prediction.*
- ***Training data** with individual cases, which is the experience.*
- *An **algorithm** that learns **supervised** from labelled training data, **unsupervised** from unlabelled data, or from **reinforcement** such as penalties and rewards.*
- ***Testing data** for evaluating the performance of the algorithm after training.*

#	**Four Wheels**	**Transports People**	**Wings**	**Yellow**	**Blue**	***Car***
1.	●	●	·	·	●	●
2.	●	●	·	●	·	●
3.	●	●	·	●	·	●
4.	●	●	●	·	●	·
5.	●	·	●	●	·	·
6.	·	●	●	·	●	·

Table 1.1: A table of three cars and three planes, with five Boolean features. There is one column per feature, each entry taking the value *True* (●) or *False* (·). The final column decides type of vehicle, *Car* (●) or *Plane* (·).

respectively. The last column contains type of vehicle: *Car* (●) or *Plane* (·). Vehicle #1, for instance, is of type *Car*, has *Four Wheels*, *Transports People*, does not have *Wings*, is *Blue*, but not *Yellow*. Some of the vehicle features are essential for recognition, such as *Four Wheels*. Others are peripheral,

Tsetlin Machines vs Black Boxes

Recent research has brought increasingly accurate learning algorithms and powerful computation platforms. However, the accuracy gains come with escalating computation costs, and models are getting too complicated for humans to comprehend. Mounting computation costs make AI an asset for the few and impact the environment. Simultaneously, the obscurity of AI-driven decision-making raises ethical concerns. We are risking unfair, erroneous, and, in high-stakes domains, fatal decisions. Tsetlin machines address the following key challenges:

- *They are universal function approximators, like neural networks.*
- *They are rule-based, like decision trees.*
- *They are summation-based, like Naive Bayes classifier and logistic regression.*
- *They are hardware-near, with low energy- and memory footprint.*

As such, the Tsetlin machine is a general-purpose, interpretable, and low-energy machine learning approach.

Figure 1.2: Four different vehicles - van, pickup, sedan, and SUV.

Property	**Values**	**Description**
Menopause	lt40, ge40, premeno	Pre- or postmenopausal status at time of diagnosis
Inv-nodes	0-2, 3-5, 6-8	Axillary lymph nodes containing visible metastatic breast cancer
Deg-malig	1, 2, 3	Degree of tumor malignancy
Recurrence	yes, no	Patient recurrence status

Table 1.2: Four properties and their values from the breast cancer data.

#	**Menopause**	**Inv-nodes**	**Deg-malig**	***Recurrence***
1.	ge40	3-5	3	*yes*
2.	lt40	0-2	3	*no*
3.	ge40	6-8	3	*yes*
4.	ge40	0-2	2	*no*
5.	premeno	0-2	3	*yes*
6.	premeno	0-2	1	*no*

Table 1.3: A table of six patients with information on *Menopause*, *Inv-nodes*, *Deg-malig*, and *Recurrence*.

such as *Blue*. Later, you discover how a Tsetlin machine distinguishes essential features from more peripheral ones.

	Menopause			Inv-nodes			Deg-malig			
#	**lt40**	**ge40**	**premeno**	**0-2**	**3-5**	**6-8**	**1**	**2**	**3**	***Recurrence***
1.	·	●	·	·	●	·	·	·	●	●
2.	●	·	·	●	·	·	·	·	●	·
3.	·	●	·	·	·	●	·	·	●	●
4.	·	●	·	●	·	·	·	●	·	·
5.	·	·	●	●	·	·	·	·	●	●
6.	·	·	●	●	·	·	●	·	·	·

Table 1.4: A Booleanized version of Table 1.3 with one Boolean feature per value of *Menopause*, *Inv-nodes*, *Deg-malig*, and *Recurrence*. Each column is a Boolean feature, each entry taking the value True (●) or False (·).

Breast Cancer Recurrence. In your first task in this book, you predict breast cancer recurrence using a real-life dataset. The dataset contains information on breast cancer patients five years after surgery.[1] You will focus on six patients and four (out of nine) multi-valued properties, which is enough data to lay out how a Tsetlin machine operates.

The four properties are: *Menopause*, *Inv-nodes*, *Deg-malig*, and *Recurrence*. You find a description of these properties and the values they take in Table 1.2. For instance, *Menopause* takes the values *lt40*, *ge40*, or *premeno*, describing menopausal status at the time of diagnosis.

Table 1.3 contains the six selected patients, our objects of study. The first object, marked #1, is a patient with *Recurrence*. You obtain the class from the last column, where *Recurrence* is *yes* for patient #1. Moreover, this patient has *Menopause* type *ge40*, *Inv-nodes* value *3-5*, and *Deg-malig* value *3*. By being multi-valued, the patient properties are not yet ready for use by the Tsetlin machine and need to be transformed by Booleanizing the data.

Data Booleanization. The first step of building a Tsetlin machine is to Booleanize your data by making suitable Boolean features. The breast cancer data demonstrates the starting point. You start by inspecting the

[1]This breast cancer dataset was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Slovenia. Thanks go to M. Zwitter and M. Soklic for providing the data, found at https://archive.ics.uci.edu/ml/datasets/Breast+Cancer.

values in Table 1.2. For each property, you create one Boolean feature per value. *Menopause*, for instance, gives the Boolean features *lt40*, *ge40*, and *premeno*. After Booleanization, you obtain Table 1.4, which encompasses nine Boolean features. You get nine features because the three original properties each takes three values. The table provides the same information on each patient as the information in Table 1.3. The difference is that Table 1.4 only contains values that are either *True* or *False*. Notice how the *Recurrence* cases all have a *Deg-malig* value of *3*. You exploit this and other data regularities later in this chapter.

Advantages of Boolean Features

Using Boolean features has several advantages:

- *Boolean features fit well with a computer because you can store them individually as bits.* True *becomes* 1*, and* False *becomes* 0*. Hardware-near bitwise operators then dramatically increase inference speed. Further, the resulting small energy- and memory footprint is ideal for edge computing and Internet of Things (IoT).*
- *You can use Boolean algebra – the language of computers – to build patterns. If you know a programming language, you have probably used Boolean algebra. You write Boolean expressions to control the flow of a program, for example, with if-then constructs.*
- *Boolean algebra is the same as propositional logic, which models logical reasoning with* True *and* False *statements. Such logical reasoning is easy for humans to understand, making the Tsetlin machine* transparent*. In comparison, so-called* black box *machine learning models are too complicated for human comprehension.*

1.2 Creating Patterns with AND and NOT

The Tsetlin machine solves pattern recognition problems by building *if-then rules* from object observations. Each rule has the form

if *condition* **then** *class*.

The *condition* part is a placeholder for a Boolean expression that describes a pattern in the data, to be learnt by the Tsetlin machine. Let us now return to our vehicle data from Table 1.1. As you can see from the table, the condition

Four Wheels **and** *Transports People*

characterises the three cars (object #1, #2, and #3) and one of the planes (object #4).

Notice how a Tsetlin machine uses the **and**-operator to combine several features. Using **and** means that all of the features must be *True* for the overall condition to be *True*. In this instance, the rule *matches* the properties of the observed object. On the other hand, if one or more of the features are *False*, the overall condition becomes *False* because the rule no longer matches the object's properties.

The *class* part is a placeholder for a type of object, such as *Car*. This is the type of object that the rule predicts when the rule's condition matches the object's properties. For example, the rule

if *Four Wheels* **and** *Transports People* **then** *Car*

predicts *Car* when it sees an object with *Four Wheels* that *Transports People*.

Negation. The Tsetlin machine also uses the **not**-operator to tell which features a class *does not* have. One says that **not** *negates* the feature. For instance, a car does not have *Wings*. You can include this fact in the rule for recognizing cars by using **not**:

if *Four Wheels* **and** *Transports People* **and not** *Wings* **then** *Car*.

The negated feature **not** *Wings* makes the rule able to distinguish between class *Car* and *Plane*. Correspondingly, class *Plane* gets its own rule:

if *Wings* **then** *Plane*.

Literals. Taken together, features and negated features are called *literals*, as customary in Boolean algebra. The three literals of our *Car* example rule are: *Four Wheels*, *Transports People*, and **not** *Wings*. Using the word *literal* is convenient because Tsetlin machines treat features and negated features the same way.

Remark. Note that a Tsetlin machine only uses the above kind of **and**-expressions. Using pure **and**-expressions makes it easier to understand the rules that the Tsetlin machine builds, increasing transparency.

1.3 Learning Frequent Patterns with Recognize and Erase Feedback

A Tsetlin machine learns by observing examples of objects of different types. It also gets to know the type of each object. It sees a plane, then a car, and so on. However, the Tsetlin machine perceives the object's features, not the object itself. For example, it observes a *Blue* vehicle of type *Car* that *Transports People*, has *Four Wheels* and does not have *Wings*. The task of the Tsetlin machine is then to learn to recognize and distinguish between the different types of objects. Learning from such labeled examples goes under the name *supervised learning*.

A Tsetlin machine learns by memorizing features. The learning is inspired by how humans remember. If you observe something multiple times, you remember it better. However, without observations, forgetting sets in after some time. Also, general properties tend to stick while fine details disappear. You remember that your neighbor's car is blue and has four wheels but may forget the kind of tire. After some time, you may forget the color as well. It appears as if everything memorized is in some state of forgetting.

Rule Memory

In traditional computer memory, everything sticks. Figure 1.3 visualizes memory that stores rule

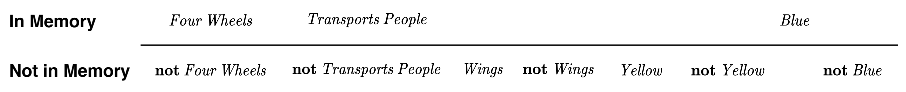

Figure 1.3: Traditional memory for the rule: **if** *Four Wheels* **and** *Transports People* **and** *Blue* **then** *Car*.

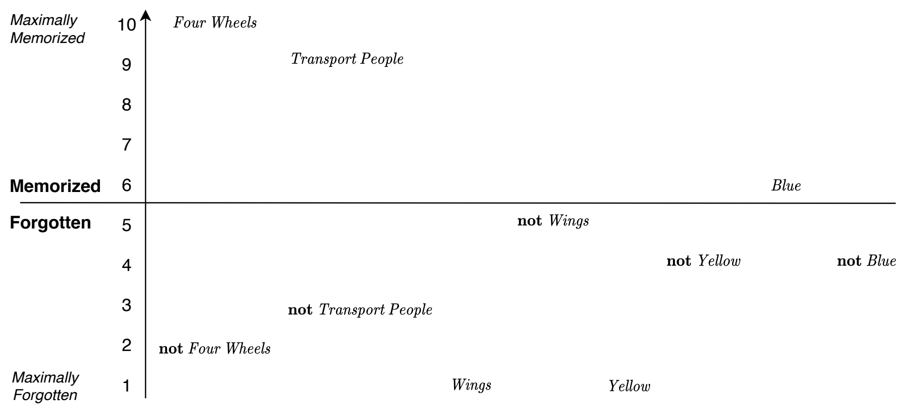

Figure 1.4: Example of Tsetlin machine memory for the rule: **if** *Four Wheels* **and** *Transports People* **and** *Blue* **then** *Car*.

if *Four Wheels* **and** *Transports People* **and** *Blue* **then** *Car*

in the traditional way. The literals *Four Wheels*, *Transports People*, and *Blue* stay in memory until you delete them. Notice that the example rule for predicting *Car* includes the literal *Blue* in its condition. If a *Car* of another color appears, the rule will not recognize the object as *Car*. You will discover how Tsetlin machines fix too specialized rules below.

A Tsetlin machine simulates forgetting and memorization. Figure 1.4 shows the Tsetlin machine memory of our example rule. The y-axis ranges from 1 to 10 and measures how deeply the memory stores each literal:

- Values 1 to 5 stand for *Forgotten*. Value 1 means maximally forgotten while value 5 means almost memorized. Literals in this range *do not* take part in the rule's condition.

- Values 6 to 10 mean *Memorized.* Value 6 stands for lightly retained, and value 10 represents maximally memorized. Literals in this range take part in the condition of the rule.

In Figure 1.4, *Four Wheels* and *Transports People* are deeply memorized, while *Blue* is lightly memorized. The other literals are forgotten to varying degrees, measured by their position in memory (from 1 to 5).

Remark. Observe how the negated and non-negated versions of each feature co-exist in memory. Both versions are candidates for being memorized, however, memorizing both gives a contradiction. A vehicle cannot both have *Four Wheels* and **not** have *Four Wheels* at the same time. That is, if *Four Wheels* is *True* then **not** *Four Wheels* becomes *False*, and vice versa. Accordingly, the condition *Four Wheels* **and not** *Four Wheels* is always *False.* As explored below, Tsetlin machines avoid contradictions by forgetting literal combinations that evaluate to *False.*

Learning of a Single Rule

The key to understanding how a complete Tsetlin machine learns is to study how a single if-then rule learns by itself. By learning independently, each rule becomes more self-contained and simpler to understand. As a side benefit, independent learning also accommodates parallel processing. Later in this chapter, you will see how multiple rules coordinate indirectly without being aware of each other, providing a simple and efficient algorithm for dissecting the complete dataset.

Rule Initialization. A rule starts up with all the literals in memory position 5. That is, the literals are about to be *Memorized* but currently *Forgotten.* Figure 1.5 shows the initial memory for your example rule. Note that you can initialize the rule any way you like. The initialization does not affect the end result of learning because the Tsetlin machine is self-correcting. This is in contrast to so-called neural network learning, which is more sensitive to initialisation.

Algorithm – Learning Steps for Single Rule. Let us first look at what happens when a rule faces an object of its own class. That would

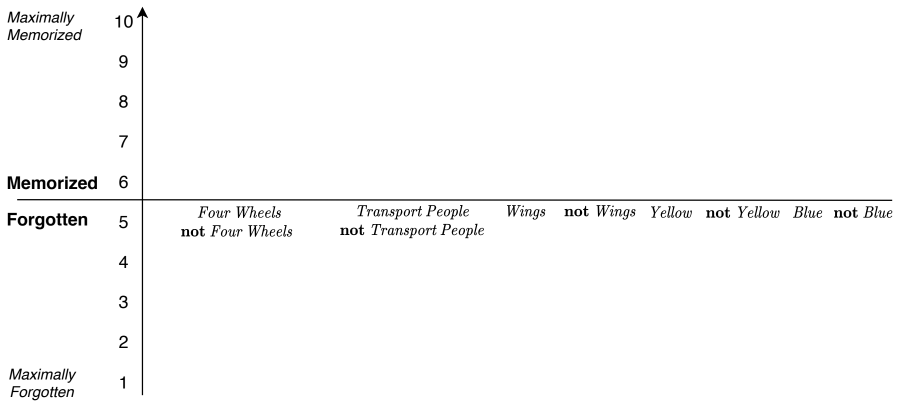

Figure 1.5: Initialization of example rule for predicting class *Car*.

be the case when our example rule for recognizing *Car* observes a vehicle labeled as *Car*. Learning then proceeds in three steps:

1. **Rule Evaluation.** Evaluate the condition part of the rule by assessing the object's literals.

2. **Recognize Feedback.** Perform this step if the condition part is *True*.

   a) Memorize the literals that are *True* for the object. You memorize a literal by incrementing its position in memory unless already *Maximally Memorized* in position 10.

   b) Forget the *False* literals by pushing them towards being *Maximally Forgotten*. You forget a literal by decrementing its position in memory unless already *Maximally Forgotten* in position 1.

   The name of this kind of feedback is Recognize Feedback because it makes the condition part of the rule mimic the observed object.

3. **Erase Feedback.** Perform this step if the condition part is *False*. Then forget all of the literals, independently of their truth values. Again, you forget literals by pushing them towards being *Maximally*

Forgotten. The name of this type of feedback is Erase Feedback because it erases the condition part of the rule.[2]

Randomization. Because coincidences happen and events sometimes occur by chance, learning should be flexible. A simple way to achieve such flexibility is randomization. Then no coincidence or chance event can make the learning get stuck. The Tsetlin machine, therefore, randomizes the increments and decrements. Before performing an increment, you draw a random floating-point value between 0.0 and 1.0. Skip the increment if the value is above 0.5. The value 0.5 gets the name *Memorize Value.* You also randomize the decrements. Again, you draw a random floating-point value before decrementing. A value above 0.5 leaves out the decrement. This second 0.5 value is coined *Forget Value.* Note that the randomized updating further diversifies the rules and boosts exploration.

Memorization and Forgetting Speed. The Memorize Value decides how quickly the rule memorizes literals. Conversely, the Forget Value decides how quickly the rule forgets in the absence of observations. If you increase the Memorize Value and reduce the Forget Value, the rule will remember literals longer. Remembering literals longer allows the rule to retain more details. In the extreme, a Memorize Value of 1.0 and a Forget Value of 0.0 make the rule memorize *every True* literal of one single object. Conversely, Memorize Value 0.0 and Forget Value 1.0 render the rule incapable of memorizing anything at all. It is usual to keep the relationship: Forget Value + Memorize Value = 1.0. The reason is that you then can obtain one from the other, and therefore you only need to specify one of them. Alternatively, you can fix the Memorize Value to 1.0.

Examples

Updating of Memory – Example 1. The first example traverses the data from Table 1.1, using Memorize Value 0.1 and Forget Value 0.9. Consider a rule for predicting *Car.* After initialization, the rule first faces vehicle #1 in the table. The vehicle is of type *Car*, hence either Recognize or

[2]In the research literature, Recognize and Erase Feedback are typically called Type Ia and Type Ib Feedback. In this book, I use the former names to make it easier to remember the roles of the feedback.

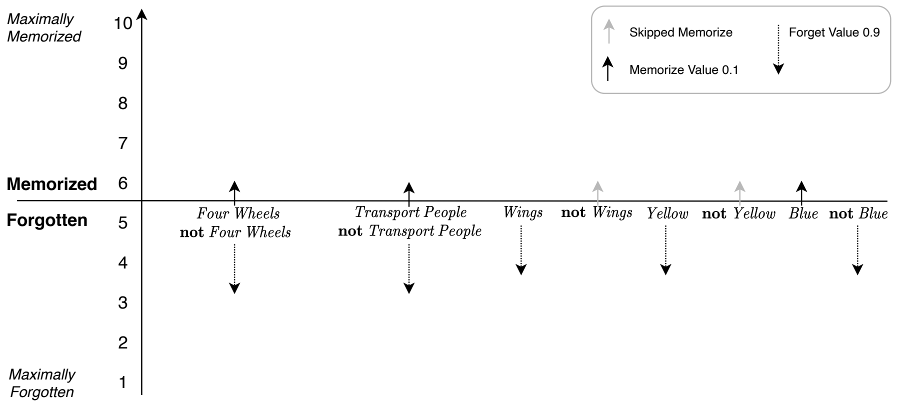

Figure 1.6: Memory updating for newly initialized rule when facing a *Blue* car that *Transports People* and does **not** have *Wings*.

Erase Feedback applies. It has five *True* literals: *Four Wheels*, *Transports People*, **not** *Wings*, **not** *Yellow*, and *Blue*. The remaining literals are *False.* Learning proceeds as follows:

- All the literals of the rule start *Forgotten*, as shown in Figure 1.5. So, the rule's condition is empty. An empty condition does not specify any literal requirements so it is always *True.* Accordingly, you proceed with Recognize Feedback.
- The solid arrows in Figure 1.6 visualize how Recognize Feedback increments the *True* literals randomly. Because of the small Memorize Value of 0.1, only a few increments will happen, hence the short arrows. Recall how each increment is skipped whenever a random floating-point value between 0.0 and 1.0 surpasses the Memorize Value. For the sake of the example, execute the increments for *Four Wheels*, *Transports People*, and *Blue*. Then **not** *Wings* and **not** *Yellow* stay in place, signified by the greyed-out arrows in the figure.
- Conversely, the dotted arrows in the figure visualize the random decrements for the *False* literals. These arrows are visualized as being longer because of the higher Forget Value of 0.9. Again, you draw a random floating-point value per pinpointed decrement. However,

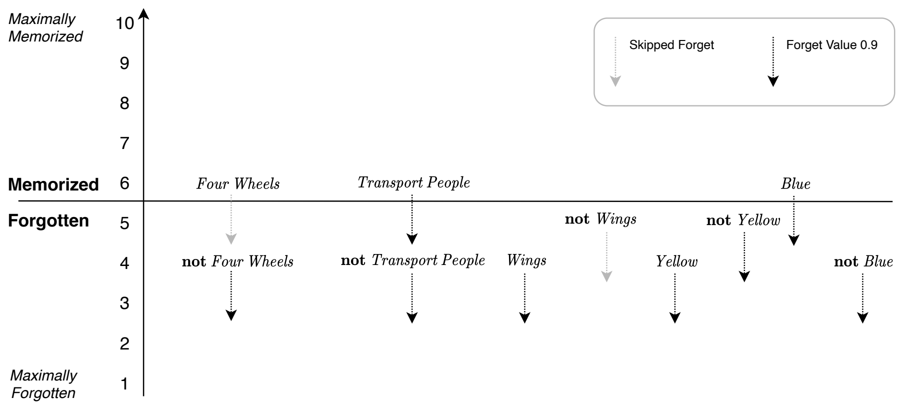

Figure 1.7: Memory updating for rule '**if** *Four Wheels* **and** *Transports People* **and** *Blue* **then** *Car*' when facing a *Yellow* car.

now a value equal to or below 0.9 executes the decrement. For this particular example, assume that all of the decrements happen.

The updated rule then becomes:

if *Four Wheels* **and** *Transports People* **and** *Blue* **then** *Car*.

Notice how the rule now is more similar to the object it observed.

Updating of Memory – Example 2. The updated rule then faces vehicle #2 in Table 1.1 – a *Yellow* car. Figure 1.7 shows how to update the rule. This time, the rule's condition does not match the car's literals because of the mismatching color. Accordingly, you use Erase Feedback. As visualized, Erase Feedback randomly decrements the memory position of all of the literals. Execute these decrements, except for *Four Wheels* and **not** *Wings* to simulate the randomization (indicated by the greyed out arrows). The new rule becomes:

if *Four Wheels* **then** *Car*.

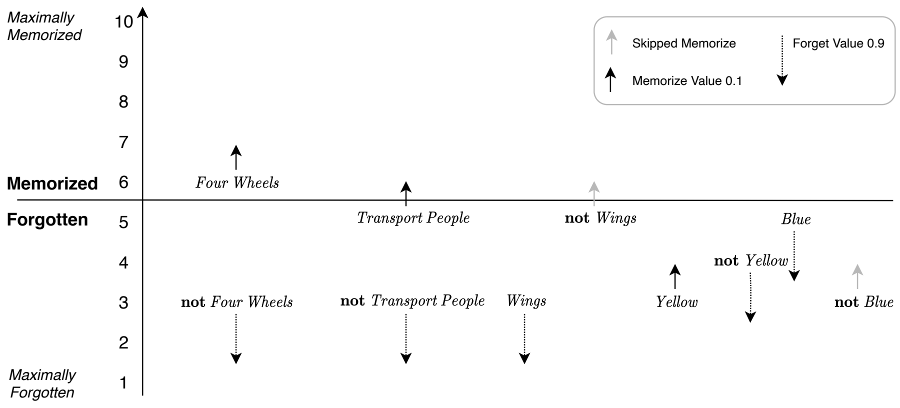

Figure 1.8: Memory updating for rule '**if** *Four Wheels* **then** *Car*' when facing a *Yellow* car.

Updating of Memory – Example 3. The rule now faces vehicle #3, which also happens to be *Yellow*. This time the rule's condition matches the vehicle's features because *Blue* is no longer included. Figure 1.8 depicts how the resulting Recognize Feedback acts on the literals. First, it increments the *True* literals *Four Wheels*, *Transports People*, **not** *Wings*, *Yellow*, and **not** *Blue*. Again, the increments are random, based on the Memorize Value. To simulate randomization, execute only the increments for *Four Wheels*, *Transports People*, and *Yellow*. The figure also shows the decrementing of the *False* literals: **not** *Four Wheels*, **not** *Transports People*, *Wings*, **not** *Yellow*, and *Blue*. To simulate randomization with Forget Value 0.9, perform all of these decrements. The resulting rule after observing vehicle #3 becomes:

if *Four Wheels* **and** *Transports People* **then** *Car*.

Figure 1.9 shows the memory of the updated rule. Observe how the rule now memorizes *Four Wheels* and *Transports People* more strongly, while getting ready to memorize **not** *Wings*. Also observe how the color literals *Yellow*, **not** *Yellow*, *Blue*, and **not** *Blue* are gradually forgotten. Varying from car to car, the frequency of each color is too low to be memorized with Memorize Value 0.1 and Forget Value 0.9. Finally, notice how the literals

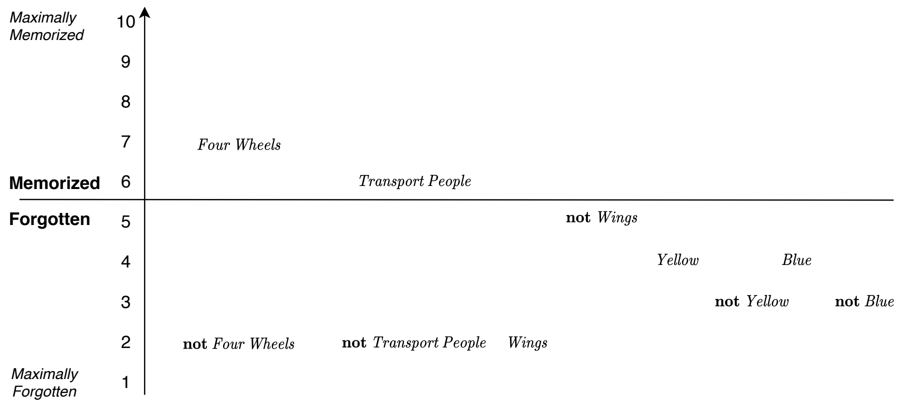

Figure 1.9: Memory for rule '**if** *Four Wheels* **and** *Transports People* **then** *Car*' after updating.

	Menopause			Inv-nodes			Deg-malig		
#	**lt40**	**ge40**	**premeno**	**0-2**	**3-5**	**6-8**	**1**	**2**	**3**
1.	·	●	·	·	●	·	·	·	●
3.	·	●	·	·	·	●	·	·	●
5.	·	·	●	●	·	·	·	·	●

Table 1.5: Three *Recurrence* patients with nine Boolean features. Each entry takes the value *True* (●) or *False* (·).

	Menopause			Inv-nodes			Deg-malig		
#	**¬lt40**	**¬ge40**	**¬premeno**	**¬0-2**	**¬3-5**	**¬6-8**	**¬1**	**¬2**	**¬3**
1.	●	·	●	●	·	●	●	●	·
3.	●	·	●	●	●	·	●	●	·
5.	●	●	·	·	●	●	●	●	·

Table 1.6: Three *Recurrence* patients with nine negated Boolean features, signified with '¬'. Each entry takes the value *True* (●) or *False* (·).

not *Four Wheels*, **not** *Transports People*, and *Wings* are quickly forgotten, not representing any car observed thus far.

High-Frequency Breast Cancer Pattern. Let us return to the prognosis of breast cancer recurrence, examining the effects of Memorize Value

0.2 and Forget Value 0.8. Consider a rule for predicting *Recurrence*. The rule learns its condition by traversing the *Recurrence* patients in the data. There is no need for you to consider the *Non-Recurrence* patients at this point because they do not influence Recognize and Erase Feedback. Table 1.5 compiles the features of the *Recurrence* patients for ease of reference. The Tsetlin machine also uses the negated features to make rules. Table 1.6 contains the truth values of these. You get the complete set of literals that the Tsetlin machine uses by combining the two tables.

With Memorize Value 0.2 and Forget Value 0.8, *True* literals that match all three *Recurrence* patients increment $3 \times 0.2 = 0.6$ times on average. They never decrement because every *Recurrence* patient matches. In contrast, consider the literals that are *False* for one of the patients, such as *Menopause ge40*. On average, this literal increments $2 \times 0.2 = 0.4$ times, while it decrements $1 \times 0.8 = 0.8$ times. So, the literal decrements more than it increments, moving towards being *Maximally Forgotten*. The only possible rule then becomes:

if *Deg-malig 3* **and not** *Menopause lt40* **and not** *Deg-malig 1* **and not** *Deg-malig 2* **then** *Recurrence*.

Note that since *Deg-malig 3* implies **not** *Deg-malig 1* and **not** *Deg-malig 2*, the rule simplifies to:

if *Deg-malig 3* **and not** *Menopause lt40* **then** *Recurrence*.

Mid-Frequency Breast Cancer Pattern. You capture less frequent patterns by diminishing forgetting and boosting memorization. For instance, let us see how Memorize Value 0.5 and Forget Value 0.5 affect learning of *Menopause ge40*. On average, this literal now increments $2 \times 0.5 = 1.0$ times and decrements $1 \times 0.5 = 0.5$ times. Accordingly, the new Memorize and Forget Values encourage rules that match two out of tree *Recurrence* patients. Then you, for instance, could get the rule

if *Deg-malig 3* **and** *Menopause ge40* **and not** *Menopause lt40* **and not** *Menopause premeno* **and not** *Deg-malig 1* **and not** *Deg-malig 2* **then**

Recurrence,

which simplifies to

if *Deg-malig 3* **and** *Menopause ge40* **then** *Recurrence*.

You could also get another rule than the one above when other regularities occur two out of three times. However, which one you get does not matter. You will later use multiple rules that together capture many different patterns in the data.

1.4 Increasing Discrimination Power with Reject Feedback

Frequent patterns that describe what is typical can be powerful in their own right. They express how the world appears, for instance, that the world contains vehicles that have four wheels and transport people. However, sometimes you want to classify objects into predefined classes such as *Car* and *Plane*. Then it is crucial to make sure that the frequent patterns distinguish between them. For instance, the condition of rule

if *Four Wheels* **and** *Transports People* **then** *Car*

characterizes both cars and planes, so it cannot see the difference between the two.

You make a rule more discriminative with the following third and final feedback type.

Algorithm – Increasing Discrimination Power. A rule increases its discrimination power when it faces an object of a class different from its own. That would be the case when our example rule for recognizing *Car* observes a vehicle labeled *Plane*. Learning then skips Recognize and Erase Feedback, going directly to the fourth step:

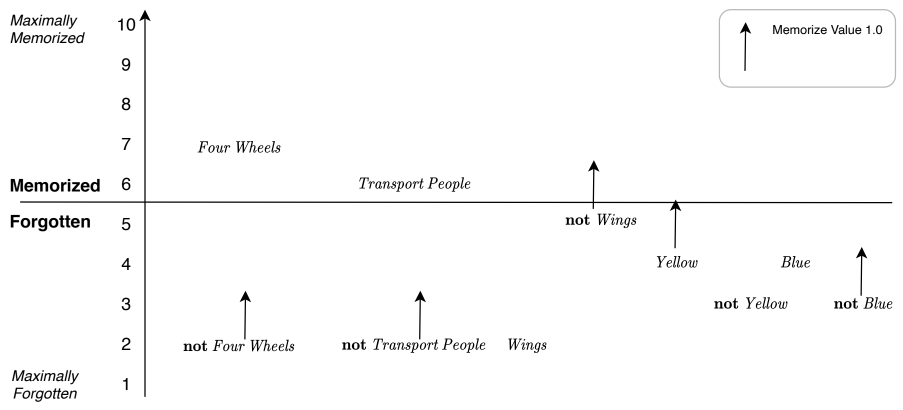

Figure 1.10: Memory updating for rule '**if** *Four Wheels* **and** *Transports People* **then** *Car*' when facing a *Blue* plane with *Four Wheels* that *Transports People*.

4. **Reject Feedback.** Check if the condition part of the rule is *True* by assessing the object's literals. If the condition part is *True*, then memorize all *Forgotten* literals that are *False* for the object. Again, you memorize a literal by incrementing its position in memory. However, this time there is no randomization – the increment is always performed. In effect, the memorization pushes the literals that are in memory position 1 to 5, and at the same time are *False*, towards being *Memorized.*

The above learning step has the name Reject Feedback because it makes the rule reject the observed object by memorizing the *False* literals.

Example with Vehicle Data. Figure 1.10 shows how our example rule for predicting *Car* updates when observing vehicle #4 in Table 1.1 – a *Plane* with *Four Wheels* that *Transports People*, is *Blue* (**not** *Yellow*), and has *Wings*. Look at how all the literals that are *False* and on the *Forgotten* side all move towards being *Memorized.* These are: **not** *Four Wheels*, **not** *Transports People*, **not** *Wings*, *Yellow*, and **not** *Blue*. In particular, **not** *Wings* goes from *Forgotten* to *Memorized* by switching from memory position 5 to memory position 6. The other literals are all *True*,

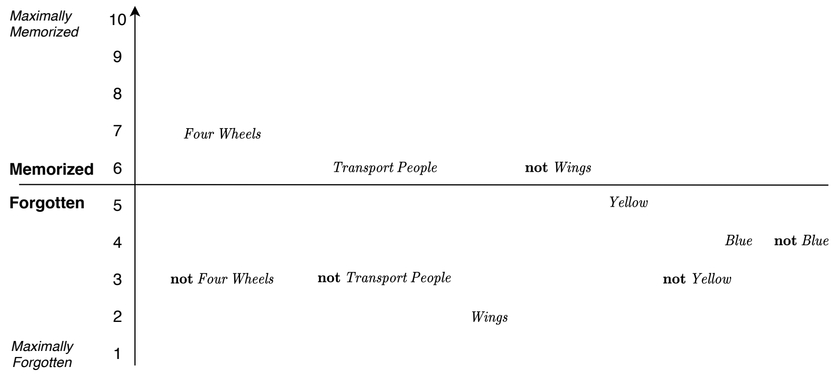

Figure 1.11: Memory for rule '**if** *Four Wheels* **and** *Transports People* **and not** *Wings* **then** *Car*' after updating.

	Menopause			Inv-nodes			Deg-malig		
#	**lt40**	**ge40**	**premeno**	**0-2**	**3-5**	**6-8**	**1**	**2**	**3**
2.	●	·	·	●	·	·	·	·	●
4.	·	●	·	●	·	·	·	●	·
6.	·	·	●	●	·	·	●	·	·

Table 1.7: Three *Non-Recurrence* patients with nine Boolean features. Each entry takes the value *True* (●) or *False* (·).

hence Reject Feedback leaves them alone. Figure 1.11 depicts the updated memory. The rule then becomes

if *Four Wheels* **and** *Transports People* **and not** *Wings* **then** *Car*.

At this point, the condition is *False* for the observed plane because the condition now contains the literal **not** *Wings*. The next time the rule faces a plane, it no longer needs to update because its condition will then be *False*. It can now distinguish between *Car* and *Plane*.

Example with Breast Cancer Data. Consider a rule that votes for *Recurrence* if the literal *Deg-malig 3* is *True*:

	Menopause			Inv-nodes			Deg-malig		
#	**¬lt40**	**¬ge40**	**¬premeno**	**¬0-2**	**¬3-5**	**¬6-8**	**¬1**	**¬2**	**¬3**
2.	·	●	●	·	●	●	●	●	·
4.	●	·	●	·	●	●	●	·	●
6.	●	●	·	·	●	●	·	●	●

Table 1.8: Three *Non-Recurrence* patients with nine negated Boolean features, signified with '¬'. Each entry takes the value *True* (●) or *False* (·).

if *Deg-malig 3* **then** *Recurrence.*

Now, take a look at the literals of the *Non-Recurrence* patients in Table 1.7 and Table 1.8. *Deg-malig 3* is *True* for patient #2, despite being a *Non-Recurrence* patient. When the above rule faces patient #2, it thus votes for the wrong class, i.e., *Recurrence.* Reject Feedback corrects this error by memorizing the *Forgotten* literals that are *False.* For patient #2, these are:

not *Menopause lt40*, *Menopause ge40*, *Menopause premeno*, **not** *Inv-nodes 0-2*, *Inv-nodes 3-5*, *Inv-nodes 6-8*, *Deg-malig 1*, *Deg-malig 2*, and **not** *Deg-malig 3.*

Facing patient #2 several times, Reject Feedback eventually pushes one of the *Forgotten False* literals to memory position 6, making it *Memorized.* From all of the literals listed above, Recognize and Erase Feedback make sure that the rule eventually memorizes literal **not** *Menopause lt40*. The resulting rule becomes

if *Deg-malign 3* **and not** *Menopause lt40* **then** *Recurrence.*

From this point, Reject Feedback no longer applies, and Recognize and Erase Feedback operate alone.

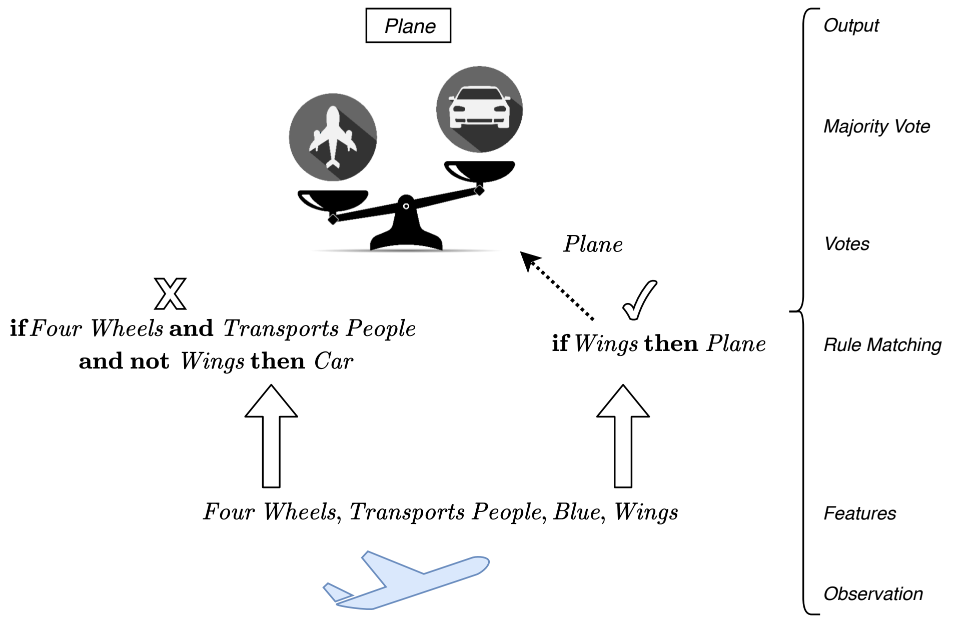
Plane
Output
Majority Vote
Plane
Votes
if Four Wheels and Transports People and not Wings then Car
if Wings then Plane
Rule Matching
Four Wheels, Transports People, Blue, Wings
Features
Observation

Figure 1.12: A majority vote among the rules decides the output. Here, *Plane* obtains more votes than *Car* (1 vs. 0 votes) and is thus output by the Tsetlin machine.

1.5 Putting the Pieces Together – How Several Rules Coordinate

Tsetlin machines use multiple rules that interact. You will now explore how this interaction takes place by dissecting the toy data on vehicles and the real-life data on breast cancer.

Classification Procedure. The Tsetlin machine classifies input by voting. That is, a single rule does not get the final say on the class by itself. Instead, it casts a vote for its class. To classify, you simply count how many rules vote for each class. The Tsetlin machine then outputs the class with the most votes. *In other words, a majority vote among the rules decides the output.*

#	**Menop.**	**Inv-nodes**	**Deg-malig**	***Recur.***	**R1**	**R2**	**R3**	**v**
1.	ge40	3-5	3	*yes*	●	●	·	+2
2.	lt40	0-2	3	*no*	·	·	●	−1
3.	ge40	6-8	3	*yes*	●	●	·	+2
4.	ge40	0-2	2	*no*	·	·	●	−1
5.	premeno	0-2	3	*yes*	●	●	●	+1
6.	premeno	0-2	1	*no*	·	·	●	−1

Table 1.9: A table of six patients with information on *Menopause*, *Inv-nodes*, *Deg-malig*, and *Recurrence*. Columns **R1**, **R2**, and **R3** list the truth value of each rule while column **v** contains the resulting vote sums.

Demonstration of Vehicle Classification. Figure 1.12 illustrates the classification of vehicle #4 in Table 1.1 – a *Blue Plane* that *Transports People* and has *Four Wheels* and *Wings*. The depicted Tsetlin machine employs two rules:

R1 **if** *Four Wheels* **and** *Transports People* **and not** *Wings* **then** *Car*.

R2 **if** *Wings* **then** *Plane*.

Rule R1 does not match the vehicle because it specifies **not** *Wings*. Hence, *Car* gets no votes. Rule R2, on the other hand, matches. So, *Plane* gets a vote and wins the majority vote. The Tsetlin machine then outputs *Plane*.

Demonstration of Breast Cancer Recurrence Prediction. Recall the breast cancer dataset from Table 1.4. Here are three rules that demonstrate the different ways rules can cooperate:

R1 **if** *Deg-malign 3* **and not** *Menopause lt40* **then** *Recurrence*,

R2 **if** *Deg-malign 3* **and not** *Menopause lt40* **then** *Recurrence*,

R3 **if** *Inv-nodes 0-2* **then** *Non-Recurrence*.

Notice that rules R1 and R2 for predicting *Recurrence* are duplicates. By voting together, R1 and R2 can win majority votes that otherwise would be tied, demonstrated below.

Columns **R1**, **R2**, and **R3** in Table 1.9 contain the truth value of each rule's condition, per patient from Table 1.4. Again, '●' means *True* and '·' means *False*. For your convenience, column **v** contains the outcome of the majority vote. Each value in the column is simply the sum of matching rules that votes for *Recurrence* minus the sum of matching rules that vote for *Non-Recurrence*. Note that to fit all the information in one table, the table only lists *True* literals. All non-listed literals are *False*.

Breast Cancer Classification Example 1. Now consider patient #1 in Table 1.9. Rules R1 and R2 match this patient, while rule R3 does not. This means that class *Recurrence* gets two votes for patient #1. The class *Non-Recurrence* gets zero votes. Accordingly, the vote sum becomes +2 and *Recurrence* wins the majority vote. This is the class that the Tsetlin machine predicts for patient #1. The prediction matches the patient's label in column ***Recur***.

Breast Cancer Classification Example 2. Sometimes a rule is not sufficiently precise by itself. Then the majority vote arbitrates disagreement. Patient #5 matches all three rules. While the correct class is *Recurrence*, rule R3 still votes for *Non-Recurrence*. The reason is that patient #5 has *True Inv-nodes 0-2*. Because *Menopause lt40* is *False* (not listed) and *Deg-malig 3* is *True*, rules R1 and R2 matches as well. As a result, the voting sum becomes +1. *Recurrence* then wins the majority vote.

Remark. Note how the last example uses detailed rules with several literals. At the same time, the output is decided by summation. One can say that the Tsetlin machine integrates rule-based and summation-based decision making. Sometimes it is useful to describe objects in detail with rules, and sometimes it is useful to add up evidence from multiple rules using summation. For comparison, so-called *decision trees* are rule-based, while so-called *logistic regression* is summation-based.

1.6 Learning to Coordinate

Starting Up. The Tsetlin machine uses multiple rules. You start up by assembling how many rules you like. Then you assign one class to each of them. In the breast cancer example, you used two *Recurrence* rules and one

Non-Recurrence rule, for instance. The Tsetlin machine then initialises each rule by placing all the literals in memory position 5, i.e., barely *Forgotten.* From there on, each rule learns its condition from observing objects one by one, together with each object's class. You already know how a single rule learns. You will now discover how several rules coordinate the learning, filling different roles in the classification of objects.

Vote Margin. The Tsetlin machine coordinates learning of multiple rules with a *Vote Margin.* A Vote Margin is an integer number that you use to create a margin between the winning and the losing class. If you set the Vote Margin to 2, you tell the Tsetlin machine that the winning class must have *exactly* two more votes than the losing class. It must then attempt to fulfill this requirement for every object it observes during learning. In this manner, the Tsetlin machine creates complementary rules. In the following, you can use a Vote Margin of 2 for illustration purposes.

Complete Learning Algorithm. The Tsetlin machine learns complementary rules as follows:

1. *Observe a new object and its class. The observation consists of the object's literals (see Section 1.1).*
2. *Evaluate each rule's condition using the truth values of the literals (see Section 1.2).*
3. *Calculate the vote sum (see Section 1.5):*
   a) *Identify the rules whose condition is* True. *Use these for voting.*
   b) *Add up the votes in favour of the object's class.*
   c) *Subtract the votes in favour of the other class.*
   d) *Refer to the summation outcome as v.*
   e) *Set v to the Vote Margin 2 if larger than 2 and to -2 if smaller than -2.*
4. *Go through each rule and give it feedback if $Rand() \leq \frac{2-v}{4}$, drawn randomly per rule:*

#	**Menop.**	**Inv-nodes**	**Deg-malig**	***Recur.***	**R1**	**R2**	**R3**	**v**
1.	ge40	3-5	3	*yes*	●	●	·	$+2$
2.	lt40	0-2	3	*no*	·	●	●	$+0$
3.	ge40	6-8	3	*yes*	●	●	·	$+2$
4.	ge40	0-2	2	*no*	·	·	●	-1
5.	premeno	0-2	3	*yes*	●	●	●	$+1$
6.	premeno	0-2	1	*no*	·	·	●	-1

Table 1.10: A table of six patients with information on *Menopause*, *Inv-nodes*, *Deg-malig*, and *Recurrence.*

   a) Give the rule Recognize or Erase Feedback if the rule belongs to the object's class (see Section 1.3).

   b) Give the rule Reject Feedback if it belongs to another class (see Section 1.4).

5. Goto 1.

Above, *Rand*() provides a random floating-point value between 0.0 and 1.0. Observe how the random updating of rules according to $Rand() \leq \frac{2-v}{4}$ is crucial for coordination. If you are far from achieving the Vote Margin for a particular object, you update the rules more aggressively. In the extreme, if the vote sum is -2 or smaller, you update all of the rules. This is because $\frac{2-v}{4}$ then becomes $\frac{2-(-2)}{4}$, which is equal to 1.0.

If the vote sum is zero, you randomly update each rule with probability 0.5. This happens because $\frac{2-v}{4}$ then becomes $\frac{2-0}{4}$, which is equal to 0.5.

If you are close to 2, updating calms down. Indeed, if the voting sum is 2 or larger, you update none of the rules. This is because $\frac{2-v}{4}$ becomes $\frac{2-2}{4}$, which is equal to 0.0.

Because of the above random updating of the rules, they individually and gradually assign themselves to classify the different kinds of objects they face. When doing so, they prioritize objects that are further away from the Vote Margin. In this manner, you achieve a resource allocation effect.

Prioritization Example. Consider a situation where you have the following three rules:

R1 **if** *Deg-malign 3* **and not** *Menopause lt40* **then** *Recurrence*,

R2 **if** *Deg-malign 3* **then** *Recurrence*,

R3 **if** *Inv-nodes 0-2* **then** *Non-Recurrence*.

Table 1.10 shows how each rule votes per patient. For convenience, column **v** again includes the resulting vote sums. Observe how the vote sums of patients #1, #3, #4, #5, and #6 give the correct classification. However, patients #4, #5, and #6 do not reach the Vote Margin of 2. As a result, every time these patients appear you update each rule randomly if $Rand() \leq \frac{2-1}{4}$, that is, one fourth of the time. Accordingly, patients #4, #5, and #6 are prioritized over patients #1 and #3. The vote sum of patient #2 gives the wrong classification and is even further away from closing the margin. Every time this patient appears, you update each rule when $Rand()$ is less than $\frac{2-0}{4}$, which is half of the time. So, patient #2 is prioritized over all the other patients.

Coordination Example. Complementary dynamics drive the coordination of rules. First of all, only rules that vote for the wrong class are affected by Reject Feedback, shielding the rules that cast their votes correctly. For instance, rule R2 votes for the wrong class when facing patient #2, however, rule R1 does not. Accordingly, only rule R2 receives the feedback. In brief, Reject Feedback singles out exactly those rules that can reduce the vote sum deviance.

Further, a Tsetlin machine memorizes precise and frequent rules quicker and more deeply. A rule memorizes the fastest when it matches all the objects of its class and never casts a wrong vote. Rule R1 is such an ideal rule. In effect, rule R1 receives the maximum amount of Recognize Feedback and never experiences Erase or Reject Feedback. One can say that the more useful a rule is, the quicker and more deeply it is memorized. So, rule R1 ends up deeply memorizing *Deg-malign 3* and **not** *Menopause lt40.*

On the other side of the spectrum, infrequently matching rules that vote on the wrong class forget quicker and memorize slower. It is Recognize and Erase Feedback that ensure forgetting of rare patterns. Recognize and Erase Feedback also slow down memorization when the matching frequency drops. Simultaneously, when voting on the wrong class, Reject Feedback introduces *False* literals into the rule's condition, disrupting learning. Rule R2 learns, for instance, slower than R1 because the Reject Feedback pushes the *False* literals of patient #5 upwards in the rule's memory, interrupting the memorization. Slower learning leads to more lightly memorized literals. This is helpful because a rule with lightly memorized literals can aptly shift to more powerful patterns when found.

Finally, observe that patient #5 will relentlessly trigger Recognize and Erase Feedback because the voting sum never becomes +2. However, this feedback only makes rule R1 more robust and helps rule R2 become more precise.

Accordingly, the Tsetlin machine quickly settles in the goal state:

R1 **if** *Deg-malign 3* **and not** *Menopause lt40* **then** *Recurrence*,

R2 **if** *Deg-malign 3* **and not** *Menopause lt40* **then** *Recurrence*,

R3 **if** *Inv-nodes 0-2* **then** *Non-Recurrence*.

1.7 Summary

Here are the key points from this chapter:

- The Tsetlin machine recognizes objects by coordinating multiple if-then rules.
- Each rule belongs to a class and learns by itself to recognize objects of that class.
- The condition part of the rule consists of **and**- and **not**-operators. These form a Boolean expression that describes a pattern. The pattern is frequent and unique for the rule's class.

- A majority vote among the rules decides the output. The Tsetlin machine thus unifies rule-based and summation-based decision making.
- There are three kinds of feedback that help produce rules that are both frequent and unique. Recognize and Erase Feedback produce frequent patterns, while Reject Feedback makes the pattern more unique for the class.
- A Vote Margin helps coordinate multiple rules. In brief, each rule acquires a suitable role by itself, supporting the other rules.
- Tsetlin machines thus unify three powerful strategies for learning patterns:
    1. Frequent pattern mining with *Recognize* and Erase Feedback.
    2. Pattern discrimination with *Reject Feedback*.
    3. Data dissection by means of the *Vote Margin*.
- Being based on pure **and**-rules in Boolean algebra, Tsetlin machine rules are transparent and easy to comprehend by humans. This is in contrast to so-called black box machine learning models.
- **And**-rules in Boolean algebra support hardware-near bitwise operators that dramatically increase inference speed. Further, the resulting small energy- and memory footprint is ideal for edge computing and Internet of Things (IoT).

1.8 Exercises

1. What is the structure of a Tsetlin machine rule?
2. How are data features prepared as input to a Tsetlin machine?
3. What are the three learning steps of a Tsetlin machine?
4. When and why are *False* and *Forgotten* literals memorized?
5. What part of Tsetlin machine learning coordinates the learning of multiple rules and how is the coordination done?

6. When is Recognize Feedback triggered and how is this feedback related to the Tsetlin machine learning steps?
7. When is Reject Feedback triggered and how is this feedback related to the Tsetlin machine learning steps?
8. What happens if we set the Forget Value closer to 0.0?
9. Implement learning of a single rule with Recognize and Erase Feedback, using the dataset in Table 1.1.
10. Implement learning of a single rule combining Recognize, Erase, and Reject Feedback, using the dataset in Table 1.4.
11. Implement learning of three rules using Vote Margin 1 first and then Vote Margin 2. Investigate any differences in the rules produced. Use the dataset in Table 1.4.