

Report on MTL Learning

Nabin Kumar Sahoo

July 2022

1 Introduction

The Problem We are interested in developing an algorithm for learning MTL formulae from data. Given a finite set of positive timed words, and a finite set of negative timed words, we want to generate an MTL formula which is satisfied by all the positive timed words and not satisfied by any of the negative timed words; that is, it perfectly separates the two given sets.

Note that, for now, we are not following the traditional learning mechanism where the objective is to separate some hidden dataset as well as possible; instead, we are only trying to separate a given visible dataset and there's no hidden (test) dataset.

Motivation for the problem There has been some recent research on inference of temporal logic properties from data, specifically for Signal Temporal Logic (STL) [1] and Linear Temporal Logic (LTL) [3].

We wanted to come up with a similar method for MTL.

The paper for STL follows the traditional learning mechanism and though they do not have a separate test dataset, they use k -fold cross-validation to validate their algorithm [1]. Whereas the paper for LTL tries to perfectly separate the given dataset [3].

2 Preliminaries

We follow the definitions of *Metric Temporal Logic* (MTL) and *timed words* from [4].

Definition 1. A timed word over finite alphabet Σ is a pair $\rho = (\omega, \tau)$, where $\omega = w_0 w_1 w_2 \dots$ is a word over Σ and $\tau = t_0 t_1 t_2 \dots$ is a time sequence of the same length with $t_i \in \mathbb{R}_{\geq 0}$ and $t_0 = 0$.

Definition 2. Given an alphabet Σ of atomic events, the formulas of MTL are built up from Σ using the following grammar.

$$\varphi ::= \top \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \varphi_1 \mathcal{U}_I \varphi_2$$

where $a \in \Sigma$, and \mathcal{U}_I is the time-constrained version of the until operator \mathcal{U} with $I \subseteq \mathbb{R}_{\geq 0}$ with endpoints in $\mathbb{N} \cup \{\infty\}$.

Definition 3. Given a timed word $\rho = (\omega, \tau)$ and an MTL formula φ , the satisfaction relation $(\rho, i) \models \varphi$ is defined inductively as follows:

- $(\rho, i) \models \top$
- $(\rho, i) \models a$ iff $i < |\rho|$ and $w_i = a$.
- $(\rho, i) \models \varphi_1 \wedge \varphi_2$ iff $(\rho, i) \models \varphi_1$ and $(\rho, i) \models \varphi_2$.
- $(\rho, i) \models \neg \varphi$ iff $(\rho, i) \not\models \varphi$.
- $(\rho, i) \models \varphi_1 \mathcal{U}_I \varphi_2$ iff there exists j such that $i < j < |\rho|$, $(\rho, j) \models \varphi_2$, $t_j - t_i \in I$, and $(\rho, k) \models \varphi_1$ for all k with $i < k < j$.

We say $\rho \models \varphi$ if $(\rho, 0) \models \varphi$.

Similar to LTL, we also have *eventually* and *always* operators. These are the *constrained eventually* operator $\mathbf{F}_I \varphi \equiv \top \mathcal{U}_I \varphi$ and the *constrained always* operator $\mathbf{G}_I \varphi \equiv \neg \mathbf{F}_I \neg \varphi$.

3 Learning MTL Formula using Decision Trees

Motivated by [1], where they use decision trees to classify signals, we wanted to classify timed words using decision trees.

3.1 Decision Trees

A decision tree is a tree-like model of *decisions* (about the data) and their consequences.

For example, suppose you have a data with two labels, ‘Yes’ and ‘No’. Start with “asking a Yes/No question” to the data and then partition the data based on the answer. Now you’d have two partitions - one would consist of data elements that answered Yes and the other, No. For each partition, repeat the same procedure until you get a partition with uniform label.

Naturally, the question arises: how do we decide when to ask which question? Sadly, the problem of learning the optimal decision tree is NP-complete, for various natural optimality criteria [2]. Therefore, we have to rely on greedy approaches, making locally optimal choices.

3.2 Primitives

In order to build the decision tree, we need “questions” that can be asked to the data. We use a finite set of parameterized MTL formulae called *primitives* for that. For the initial experiments, we considered the following simple formulae.

- $\mathbf{F}_I a$,
- $\mathbf{G}_I a$,
- $\mathbf{F}_I \mathbf{G} a$,
- $\mathbf{G}_I \mathbf{F} a$,
- $\mathbf{F}_I(a \implies \mathbf{G} b)$,
- $\mathbf{G}_I(a \implies \mathbf{F} b)$

For this set of primitives, I first checked if they (and their logical negations) could be converted to equivalent Timed Automata. This check turned out to be affirmative with the Timed Automata having only one clock without any reset. This implies that their union and intersection would also only have one clock. So any Boolean combination of the primitives can be converted to an equivalent Timed Automaton with one clock without reset.

3.3 Set-up

For implementing the decision tree I needed to come up with a method for checking whether a timed word satisfies a primitive or not. For this I used the equivalent Timed Automata of the primitives.

I created a class (in Python 3) for simulating an arbitrary Timed Automaton and running timed words on it to check acceptance. In the said class, I also made it possible to define Timed Automata with abstract bounds. With this it became easy to define Timed Automata for all parameters while searching the space for optimal solution, by assigning values to the abstract bound.

3.3.1 Impurity measures

Definition 4. Let S be a finite set of timed words, φ an MTL formula and $S_{\top}, S_{\perp} = \text{partition}(S, \varphi)$; that is, $S_{\top} := \{\rho \in S \mid \rho \models \varphi\}$ and $S_{\perp} := \{\rho \in S \mid \rho \not\models \varphi\}$. Define

$$p_{\top} = \frac{|S_{\top}|}{|S|}, \quad p_{\perp} = \frac{|S_{\perp}|}{|S|}, \quad p(S, c) = \frac{|\{(\rho_i, l_i) \mid l_i = c\}|}{|S|}$$

The term $p(S, c)$ represents the fraction of timed words in S that belong to class c , where c is one of the two classes – positive or negative.

The impurity measures we work with are as follows [5]:

- *Information gain (IG)*

$$IG(S, \{S_{\top}, S_{\perp}\}) = H(S) - \sum_{k \in \{\top, \perp\}} p_k \cdot H(S_k)$$

$$H(S) = - \sum_{c \in \{+, -\}} p(S, c) \log p(S, c)$$

- *Gini gain (GG)*

$$GG(S, \{S_{\top}, S_{\perp}\}) = Gini(S) - \sum_{k \in \{\top, \perp\}} p_k \cdot Gini(S_k)$$

$$Gini(S) = \sum_{c \in \{+, -\}} p(S, c)(1 - p(S, c))$$

3.3.2 Local Optimization

While building the decision tree, at each node, we have to make the locally optimal choice to choose the MTL formula from among the primitives. That is, we choose the MTL formula which maximizes the impurity measure.

The optimization is performed over the set of Parameterized MTL primitives and their valuations.

For now we are either using Differential Evolution [6] or manual iteration to find the best values for the parameters of the primitives.

3.4 Experiments

For all the experiments we considered an alphabet of size 3 – $\Sigma = \{a, b, c\}$.

3.4.1 Generating Dataset

For the experiments I needed (to generate) datasets – timed words. I tried to build a general procedure for generating the timed words – that is, given an arbitrary timed automaton, generate some accepting words and some rejecting ones. But this turned out to be very costly – in terms of both space and time. So, for generating the words I used procedures specific to the given automaton.

3.4.2 Experiment 1

Dataset We first considered the automaton equivalent of $\mathbf{F}(a \wedge \mathbf{F}_I a)$ where $I = [1, 1]$ or an open interval around 1, for example $(0.8, 1.2)$. For this experiment I generated 500 positive and 500 negative words. Positive words were from the language of the automaton and negative words from the complement of the language.

Obstacles I had to make sure the lengths of the timed words in both the positive and negative sets varied similarly and the time stamps were also similar as these could be exploited to differentiate between the two sets *without* recognizing the crucial pattern(s).

This was the case in the very first experiment when the best split was given by a primitive of the form $\mathbf{F}_I(b)$, even though the actual pattern has nothing to do with the letter b . This happened because timestamps of some of the negative words differed drastically from the positive ones.

To combat the above situation, I generated positive and negative data with exact same timestamps – all the words had the same timestamp. This gave better results as for all the splits (at all heights), the algorithm (majorly) chose a primitives. But even for this simple example, the depth of the tree reached 7. However, this experiment clearly isn't comprehensive. So, next, I worked on generating timed words that kept the above points in mind while still having diverse timestamps.

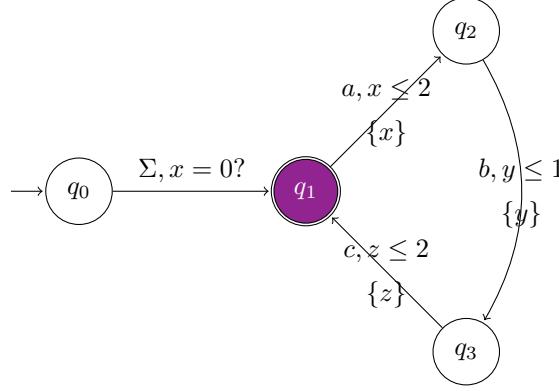


Figure 1: Timed Automaton for Experiment 2

Local optimization For this particular experiment we used Differential Evolution [6] (DE) for local optimization instead of iterating over positive integers as the latter was time consuming. However, one cannot restrict the search space to integers in the SciPy¹ implementation of DE, and so the results we got were positive real numbers. So the MTL formula we get as a result of the algorithm is not, by definition, an MTL formula as the intervals in it need not be in \mathbb{N} .

However, a formula with intervals having real numbers should at least be as good (at separating the given dataset) as a formula with intervals having integers. So, we went ahead with this experiment as it would give us an idea of how good the results we can expect.

Result The result – the decision tree – of the experiment can be found as a separate pdf (as it was too big to attach here).

3.4.3 Experiment 2

Dataset We considered the timed automaton in Figure 1. As before, the (500) positive words were the words accepted by the automaton and the (500) negative words were the rejecting ones.

Local optimization From this experiment onward, we worked with integers instead of reals for the values of the parameters. For this experiment we iterated over the positive integers instead of using DE. The largest timestamp for the dataset was not too large so iterating over the (bounded set of) integers was fast.

Obstacles The resulting decision tree we got failed to perfectly separate the two – positive and negative – sets. This happened because there were positive-negative pairs which could not be distinguished by any primitive for any value of the parameters.

To overcome this, we added new primitives of the form $\mathbf{F}(a \wedge \mathbf{F}_I b)$ and $\mathbf{F}_I(a \wedge \mathbf{F} b)$ for all $a, b \in \Sigma$.

3.4.4 Experiment 3

Next, we worked with discrete-time timed words.

Dataset We considered the timed automaton in Figure 2, with 300 positive words and 300 negative words.

Local optimization For this experiment we used DE, but we mapped its resulting real numbers to integers. This wouldn't change the partitions as we are working with discrete time.

First, we ran the decision tree algorithm with just the original primitives. We got perfect separation but the height of the decision tree was 12.

We then ran the algorithm again, this time with the new primitives we had added in Experiment 2. For this, we got a smaller decision tree – with height 9.

¹https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html

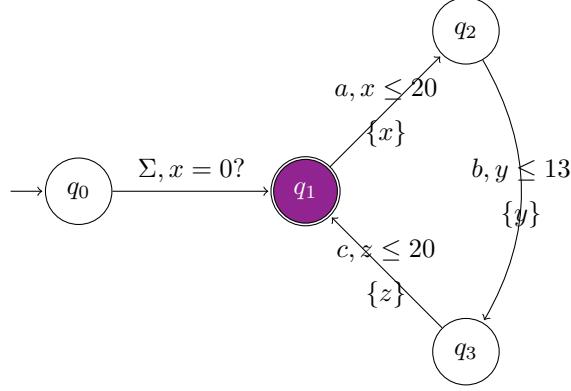


Figure 2: Timed Automata for experiment 3

4 Future work

Finding a faster method for local optimization Since we are following a greedy approach, even if we do find the locally optimal MTL formula, splitting the dataset with that formula will probably not lead us to a globally optimal decision tree. And since finding the locally optimal MTL formula is expensive (in terms of time) we are thinking of taking a (more) randomized approach for local optimization in the future.

Experimenting with new primitives As we saw in experiments 2 and 3, with fewer primitives it might not be possible to separate the dataset, and adding new primitives gave us (comparatively) smaller decision trees. If we add more primitives, we might get even smaller ones.

Experiment in a traditional learning setting For now we have only been trying to perfectly separate a given dataset. We could also try following the traditional learning mechanism and try to build a general classifier.

References

- [1] Giuseppe Bombara, Cristian-Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. A decision tree approach to data classification using signal temporal logic. *HSCC '16*, page 1–10, New York, NY, USA, 2016. Association for Computing Machinery.
- [2] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Inf. Process. Lett.*, 5:15–17, 1976.
- [3] Daniel Neider and Ivan Gavran. Learning linear temporal properties, 2018.
- [4] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 188–197, 2005.
- [5] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Elsevier, 2014.
- [6] Rainer Storn and Kenneth V. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.