JMLR: Workshop and Conference Proceedings 57:69–72, 2016 Proceedings of the 13th ICGI

Learning Deterministic Finite Automata from Infinite Alphabets

Gaetano Pellegrino G.PELLEGRINO@TUDELFT.NL
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands

Christian Albert Hammerschmidt CHRISTIAN.HAMMERSCHMIDT@UNI.LU
Interdisciplinary Institute for Security, Reliability, and Trust
University of Luxembourg
Rue Alphonse Weicker 4, L-2127, Luxembourg

Qin Lin Q.LIN@TUDELFT.NL
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands

Sicco Verwer S.E.VERWER@TUDELFT.NL
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD, Delft, The Netherlands

Abstract

We proposes an algorithm to learn automata infinite alphabets, or at least too large to enumerate. We apply it to define a generic model intended for regression, with transitions constrained by intervals over the alphabet. The algorithm is based on the Red & Blue framework for learning from an input sample. We show two small case studies where the alphabets are respectively the natural and real numbers, and show how nice properties of automata models like interpretability and graphical representation transfer to regression where typical models are hard to interpret.

Keywords: Passive Learning, Deterministic Finite Automata, Regression

1. Introduction

Automata have been studied in depth, e.g. in **?** (**?**), and successfully applied in a wide range of fields, ranging from biology over linguistics to computer science itself. Especially in the field of software specification and verification, these models are appreciated for their balance between expressive power and decidability of model and language class properties. In reverse, inferring an automaton from observations of software interactions, is an important step when analyzing and reverse engineering software and protocols, e.g. **?** (**?**), **?**. Here, the benefit of automata are easy to interpret and graphically representable models. Because most work on learning these models has been done under the assumption of small alphabet sizes, it is hard to transfer the nice properties to situations where the alphabet is large, or even uncountable. In this work, our goal is to provide a general method to infer

automaton models on infinite totally ordered alphabets, meant for the task of regression. Many regression methods, like ARIMA (**?**, **?**) and neural LSTM models (**?**, **?**), don't offer a good way of interpreting and understanding the learned model. We propose a *bottom-up approach* to learn finite automata from infinite alphabets and apply to deterministic finite automata intended for regression: each transition is constrained by intervals over the infinite alphabet. For this reason we call then Deterministic Regression Automata with Guards (RAGs).

Finite automata with potentially infinite alphabets have been studied in theory, e.g. in **?** (**?**), **?**, and shown to be conservative extensions. Often, automata are also extended with additional memory such as registers or variables **?**, **?** (**?**). Learning these variants of automata from input samples has not been researched frequently. In the context of alphabet refinement, **?** (**?**) permits infinite alphabets and uses an active learning approach to learn automata. **?** (**?**) proposes a modified Angluin's L* algorithm to learn from large alphabets. The method is *top-down*: Initially, the largest possible label is taken for each transition. Upon queries to the oracle, a label can be partitioned. Partitions of the infinite alphabet are used to label the transitions in the final automaton. For state-merging learning approaches, **?**, **?** (**?**), propose a clustering-based algorithm to infer real-time automata on multivariate timed events, where the events may contain real-valued components. The data is used to cluster states globally in a merge step, without evaluating the equivalence of future continuations of the states.

A very close related work is **?**, where regression automata without guards are learned for predicting wind speed data, overcoming the limitless of the alphabet by translating it in a bounded symbolic domain.

The rest of the paper is organized as follows. In Section 2 we provide basic notation and definitions, furthermore we introduce Regression Automata with Guards. In Section 3 we describe our algorithm for learning RAGs from a data sample, and we show how it works in two case studies. In Section **??** we conclude by a discussion about the current and future work.

2. Deterministic Regression Automata with Guards

This section uses basic notation from grammar inference theory, for an introduction we refer to **?**. In several contexts, i.e. time series forecasting, data are sequences of points made over a continuous time interval, out of consecutive and equally spaced measurements. We model such data with sequences of symbols taken from a given totally ordered alphabet Σ, where the time is indirectly represented by the position within the sequence. This is sufficient because in practice we always deal with a finite precision of time intervals, e.g. milliseconds, minutes, hours. In this paper we introduce a new type of finite state automaton exclusively meant for dealing with large or infinite alphabets. In Regression Automata with Guards (RAGs), every transition is decorated with constraint guards. We represent a constraint guard by a closed interval in Σ, and we say that $[l, r]$ is satisfied by a symbol $s \in \Sigma$ if $s \in [l, r]$. A RAG is defined as follows:

Definition 1 (RAG) *A Regression Automaton with Guards (RAG) is a 5-tuple* $\langle \Sigma, Q, q_0, \Delta, P \rangle$ *where* Σ *is the alphabet,* Q *is a finite set of states,* $q_0 \in Q$ *is the start state,* Δ *is a*

finite set of transitions, and $P : Q \to \Sigma$ *is a prediction function that assigns a prediction value to every state in* Q*. A transition* $\delta \in \Delta$ *is a triple* $\langle q, q', [l, r]\rangle$ *where* $q, q' \in Q$ *are respectively the source and target states, and* $[l, r]$*,* $l, r \in \Sigma$*, is a guard.*

We will also use functional notation for transitions, so for $q \in Q$ and $s \in \Sigma$, $\delta(q, s) = q' \in Q$ iff $\exists\, \langle q, q', [l, r]\rangle \in \Delta$ s.t. $s \in [l, r]$. We only focus on deterministic regression automata because of the complexity of learning non-deterministic automata (**?**). A RAG is called deterministic if it does not contain two transitions with the same source state and any overlap between the guards. In a RAG, a state transition is possible only if its constraint guard is satisfied by a coming value. Hence a transition $\delta = \langle q, q', [l, r]\rangle$ is interpreted as follows: whenever the automaton is in state q, by reading an incoming value v such that $v \in [l, r]$, then the automaton changes state moving to q'.

In order to define a computation of a RAG, we introduce the notion of closest transition:

Definition 2 (closest transition) *The closest transition for a given state* $q \in Q$ *of a RAG* $A = \langle \Sigma, Q, q_0, \Delta, P\rangle$*, and given a symbol* $s \in \Sigma$*, is the transition* $\pi_q(s)$ *such that:*

$$\pi_q(s) = \begin{cases} \langle q, q_0, [s, s]\rangle & \text{if } \nexists\, \langle q, q', [l, r]\rangle \in \Delta \\ \langle q, q', [l, r]\rangle & \text{if } \exists\, \langle q, q', [l, r]\rangle \in \Delta \text{ s.t. } s \in [l, r] \\ \langle q, q', [l, r]\rangle & \text{if } \exists!\, \langle q, q', [l, r]\rangle \in \Delta \text{ s.t. } s < l \text{ or } s > r \\ \underset{\delta_{left}, \delta_{right} \in \Delta}{\operatorname{argmin}} \{|s - r_{left}|, |s - l_{right}|\} & \text{otherwise.} \end{cases}$$

The first branch defines a default transition to the start state when no transitions are available in q. The second branch defines the closest transition as the only one, if exists, that contains s. Since we are restricting to deterministic RAGs, at most one transition which includes the value can be present in Δ. The third branch addresses the special case when there exists only one transition in Δ, with initial state q, and it does not contain s. The last branch occurs when the value is located in between two consecutive transitions ($\delta_{left} = \left\langle q, q'_{left}, [l_{left}, r_{left}]\right\rangle$ and $\delta_{right} = \left\langle q, q'_{right}, [l_{right}, r_{right}]\right\rangle$). In this case the one with the closest edge is chosen.

The behavior of a RAG is defined by its computation:

Definition 3 (RAG computation) *A finite computation of a RAG* $A = \langle \Sigma, Q, q_0, \Delta, P\rangle$ *over a finite sequence of symbols* $s = s_1, s_2, \ldots, s_n$ *is a finite sequence*

$$q_0 \xrightarrow{s_1} q_1 \xrightarrow{s_2} q_2, \ldots, q_{n-1} \xrightarrow{s_n} q_n$$

such that for all $1 \leq i \leq n$ $\langle q_{i-1}, q'_i, [l_i, r_i]\rangle = \pi_{q_{i-1}}(s_i)$*. It is also called close-computation.*

Let $S \subseteq \Sigma^+$ denote a sample of non-empty sequences with symbols in Σ. We call $SUFF(S) \subseteq \Sigma^+$ the set of all non-empty suffixes of sequences in S. Hereby we introduce the notion of future continuations set of a state q, as the set of all suffixes of a sample inducing a computation in A that starts with q:

Definition 4 (future continuations set) *The future set for a state $q \in Q$ of a RAG $A = \langle \Sigma, Q, q_0, \Delta, P \rangle$, given a sample $S \subseteq \Sigma^+$, is the set $\phi_{A,S}(q) = \{s = s_1, s_2, \ldots, s_n \in SUFF(S) \mid \exists q_1, q_2, \ldots, q_n \in Q^+, q_0 = q$, and $\delta(q_{i-1}, s_i) = q_i$, $i = 1, 2, \ldots, n\}$. $\phi_A(q)$ denotes the future continuations set of state q given the sample used for learning A.*

We also introduce the notion of transition centroid given a sample as the mean of all values of the sequences, included in the sample, that get caught by this transition:

Definition 5 (transition centroid) *The centroid of a transition $\delta = \langle q, q', [l, r] \rangle$ of a RAG A, given a sample $S \subseteq \Sigma^+$, is $\mu_S(\delta) = \frac{\sum_{s \in \phi_{A,S}(q) \wedge I(s)=1} s}{\sum_{s \in \phi_{A,S}} I(s)}$, where $I(s) = \begin{cases} 1 \text{ if } |s| = 1, \\ 0 \text{ otherwise.} \end{cases}$*

When clear from the context, we will omit the sample subscript from the transition centroid.

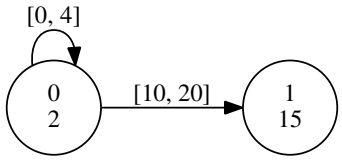

Figure 1: An example of a RAG. The leftmost state is the start state. Every state transition contains a guard. Missing transitions lead to the start state. Every state contains an identification number (above) and a prediction value (below).

Example 1 *Figure 1 shows an example of RAG. This RAG computes sequences of real values. For instance, given the sequence* $0.88, 15.07$*, it crosses the states* $0, 0, 1$*. Given the sequence* $0.88, 9.05$ *the RAG crosses the states* $0, 0, 1$ *because in state* 0 *the closest transition to value* 9.05 *leads to state* 1*.*

3. Learning Regression Automata with Guards

The problem of learning RAGs is a specialization of the more general problem of learning deterministic finite state automata (DFAs), with the additional task of learning guards over transitions. Unfortunately identifying transition guards is already an NP-Complete problem, as demonstrated in **?** (**?**) for time guards in real time automata. In addition, the more general problem of learning DFAs is again NP-Complete, as proved in **?**. Hence we will not be able to solve this task efficiently unless P = NP. However, we can still design an efficient algorithm that learns both the structure and the guards and converge to the correct underlying RAG when more and more data are provided, in the limit. It has been done for deterministic finite state automata with RPNI algorithm (**?**), and for real time automata with RTI algorithm (**?**, **?**). For real time automata there exists a polynomial time algorithm, able to identify time guards over transitions and structure of the automaton in the same way, such that it converges in the limit to the correct target.

Algorithm 1 Regression Automata Identifier (RAI)

Data: a sample S of real value sequences, a threshold τ

$A :=$ BUILD-PT(S) ; `// construct the prefix tree`

$RED := \emptyset$; `// core set of red states`

$BLUE := \emptyset$; `// fringe of blue states`

`// make the root of the prefix tree red`

PROMOTE(PROMOTE($A, RED, BLUE, q_0, \tau$)) **while** $BLUE \neq \emptyset$ **do**

  CHOOSE($q_b \in BLUE$) ; `// select the best red/blue merge`

  ; `// if` q_b `and` q_r `are compatible`

  if $\exists q_r \in RED \mid$ COMPATIBLE(A, q_r, q_b, τ) **then**

    MERGE(A, q_r, q_b) ; `// perform the merge`

  else

    ▷ if no compatible merges are available for q_b PROMOTE($A, RED, BLUE, q_b, \tau$) ▷ make q_b red

return A