A Congruence-based Approach to Active Automata Learning from Neural Language Models

Franz Mayr MAYR@ORT.EDU.UY
Sergio Yovine YOVINE@ORT.EDU.UY
Matías Carrasco MATIAS.CARRASCO@FI365.ORT.EDU.UY
Federico Pan
Federico Vilensky
Facultad de Ingeniería, Universidad ORT Uruguay, Montevideo, Uruguay

Editors: François Coste, Faissal Ouardi and Guillaume Rabusseau

Abstract

The paper proposes an approach for probably approximately correct active learning of probabilistic automata (PDFA) from neural language models. It is based on a congruence over strings which is parameterized by an equivalence relation over probability distributions. The learning algorithm is implemented using a tree data structure of arbitrary (possibly unbounded) degree. The implementation is evaluated with several equivalences on LSTM and Transformer-based neural language models from different application domains.

Keywords: MAT-PAC active learning. Neural language models. PDFA.

1. Introduction

In this paper we are interested in providing formal algorithmic means to help verifying the behavior of neural language models. The main motivation is that those models are expected to be progressively used for solving tasks of increasing criticality, in domains such as control (Scheiner et al. (2019)), cybersecurity (Du et al. (2017)), genetics (Oubounyt et al. (2019)), etc. Their failure may cause more or less serious damage depending on the application. It is therefore important to ensure they work correctly (Seshia et al. (2022)).

For binary predictors, several works have already addressed this issue through a *model, then verify* approach by learning some kind of automata that approximates the formal language defined by the network and is amenable for automated verification (see Bollig et al. (2022) and references there in). For general neural language models, the literature on learning automata is more scarce. To the best of our knowledge we could mention Weiss et al. (2019) and Mayr et al. (2022). The former extends L* to learn a probabilistic deterministic finite automaton (PDFA) (Clark and Thollard (2004)) through a non-transitive similarity relation between next-symbol probability distributions. The latter proposes to overcome the algorithmic issues due to non-transitivity by relying on an equivalence relation defined by quantizing the next-symbol probability simplex and on a tree-based learning algorithm.

This paper generalizes these two works. The theoretical contributions are twofold: 1) it defines a parameterized congruence $\equiv_{\mathcal{S}}$ over strings where $\mathcal{S}$ is an equivalence relation over probability distributions and shows it results in an irreducible quotient for PDFA (Sec. 2); 2) it extends the algorithm of Mayr et al. (2022) for learning a quotient PDFA modulo $\equiv_{\mathcal{S}}$ from any language model and proves its partial correctness, in general, and its total correctness for so-called $\mathcal{S}$-regular languages (Sec. 3). Moreover, it discusses the experimental results obtained with various neural language models and congruences $\equiv_{\mathcal{S}}$ on several case studies from different application domains (Sec. 4). Further related work is analyzed in Sec. 5.

2. Language models

Let Σ be a finite *alphabet* and $\Sigma_{\$} \triangleq \Sigma \cup \{\$\}$, where $\$$ is a special *terminal* symbol not in Σ. Also let $\Delta(\Sigma_{\$}) \triangleq \{\rho : \Sigma_{\$} \to \mathbb{R}_+ \mid \sum_{\sigma \in \Sigma_{\$}} \rho(\sigma) = 1\}$ be the probability simplex over $\Sigma_{\$}$. A *language model* is a total function $\mathcal{L} : \Sigma^* \to \Delta(\Sigma_{\$})$ where $\mathcal{L}(u)$ is interpreted as the *next-symbol* probability distribution, that is, $P_{\mathcal{L}}[\sigma|u] \triangleq \mathcal{L}(u)(\sigma)$.

2.1. Equivalence between language models

Let $\mathcal{S} \subseteq \Delta(\Sigma_{\$}) \times \Delta(\Sigma_{\$})$ be an equivalence relation. We write $\rho =_{\mathcal{S}} \rho'$ instead of $\mathcal{S}(\rho, \rho')$, and $[\![\Delta(\Sigma_{\$})]\!]_{\mathcal{S}}$, respectively $[\![\rho]\!]_{\mathcal{S}}$, to denote the quotient of $\Delta(\Sigma_{\$})$ induced by $\mathcal{S}$, and the class of ρ. Given $\mathcal{L}$, $\mathcal{S}$ induces the equivalence relation $\equiv_{\mathcal{S}} \subseteq \Sigma^* \times \Sigma^*$ defined as:

$$u \equiv_{\mathcal{S}} u' \overset{\Delta}{\iff} \forall w \in \Sigma^*.\ \mathcal{L}(uw) =_{\mathcal{S}} \mathcal{L}(u'w) \tag{1}$$

We write $[\![\Sigma^*]\!]_{\mathcal{S}}$ for the set of equivalence classes defined by $\equiv_{\mathcal{S}}$ and $[\![u]\!]_{\mathcal{S}}$ for the class of $u \in \Sigma^*$. Indeed, $\equiv_{\mathcal{S}}$ is a *right congruence* with respect to concatenation of a symbol:

Proposition 2.1 $\forall u, u' \in \Sigma^*.\ u \equiv_{\mathcal{S}} u' \implies \forall \sigma \in \Sigma.\ u\sigma \equiv_{\mathcal{S}} u'\sigma.$
Proof Let $u \equiv_{\mathcal{S}} u'$ and $\sigma \in \Sigma$. Then for any $w \in \Sigma^*$ we have $\mathcal{L}((u\sigma)w) = \mathcal{L}(u(\sigma w)) =_{\mathcal{S}} \mathcal{L}(u'(\sigma w)) = \mathcal{L}((u'\sigma)w)$. $\square$

Naturally, we can say that two language models $\mathcal{L}_1$ and $\mathcal{L}_2$ are equivalent modulo $\mathcal{S}$ if $\mathcal{L}_1(u) =_{\mathcal{S}} \mathcal{L}_2(u)$ for all $u \in \Sigma^*$. This implies that $\mathcal{L}_1$ and $\mathcal{L}_2$ induce the same equivalence relation over Σ^*. Therefore, we write $\mathcal{L}_1 \equiv_{\mathcal{S}} \mathcal{L}_2$. $\mathcal{L}$ is $\mathcal{S}$-*regular* if $[\![\Sigma^*]\!]$ is finite.

Several equivalence relations $\mathcal{S}$ are of interest. One such equivalence is equality modulo *quantization* (Mayr et al. (2022)), denoted $=_\kappa$, where $\kappa \in \mathbb{N}$, $\kappa \geq 1$, is the quantization parameter. This equivalence is motivated by the need to cope with small discrepancies between distributions. For $n \in \mathbb{N}$, $0 \leq n < \kappa - 1$, we define the quantization *interval* I_κ^n to be the left-closed right-open interval $\left[n\kappa^{-1}, (n+1)\kappa^{-1}\right)$, and for $n = \kappa - 1$, to be the closed interval $\left[n\kappa^{-1}, 1\right]$. For $x \in \mathbb{R}$, $[\![x]\!]_\kappa$ is the interval I_κ^n such that $x \in I_\kappa^n$. For $x, y \in \mathbb{R}$, $x =_\kappa y$ if $[\![x]\!]_\kappa = [\![y]\!]_\kappa$. For $\rho, \rho' \in \Delta(\Sigma_{\$})$, $\rho =_\kappa \rho'$ if $[\![\rho(\sigma)]\!]_\kappa = [\![\rho'(\sigma)]\!]_\kappa$ for all $\sigma \in \Sigma_{\$}$.

Other equivalences are related to the purpose a language model is used for. For instance, for anomaly detection, the classification approach proposed by Du et al. (2017) relies on comparing each symbol in the sequence with the top-r most likely symbols predicted by the model. Let $\mathit{rank}(\rho) : \Sigma_\$ \to \mathbb{N}$ be the ranking of symbols $\sigma \in \Sigma_\$$ induced by their probability $\rho(\sigma)$: the symbol with the greatest probability has rank 1, the second best has rank 2, and so on, where symbols with equal probability have the same rank. We denote rank_r the restriction to the first r ranked symbols. The top-r ranked symbols are given by the function: $\mathit{top}_r(\rho) \triangleq \{\sigma \in \Sigma_\$ \mid \mathit{rank}(\rho)(\sigma) \leq r\}$. These functions induce the following equivalences between distributions: $\rho =_{\mathit{rank}_r} \rho'$ if $\mathit{rank}_r(\rho) = \mathit{rank}_r(\rho')$, and $\rho =_{\mathit{top}_r} \rho'$ if $\mathit{top}_r(\rho) = \mathit{top}_r(\rho')$. A string $u = u_1 \dots u_n \in \Sigma^*$ is classified as anomalous by $\mathcal{L}$ if there exists $i \in [1 \dots n]$ such that $u_i \notin \mathit{top}_r(\mathcal{L}(u[i-1]))$, where for $i \geq 1$, $u[i] \triangleq u_1 \dots u_i$ and $u[0] \triangleq \lambda$, otherwise it is classified as normal. Therefore, two language models which are $\equiv_{\mathit{top}_r}$-equivalent will classify sequences the same way. Moreover, $\equiv_{\mathit{rank}}$-equivalence implies $\equiv_{\mathit{top}_r}$-equivalence for every cutoff threshold r.

	ρ_0	ρ_1	ρ_2	ρ_3	ρ_4
0	7/16	6/16	6/16	4/16	2/16
1	6/16	7/16	3/16	2/16	4/16
\$	3/16	3/16	7/16	10/16	10/16

Table 1: Probability distributions

To illustrate the above equivalences, let us consider the distributions in Tab. 1. Quantization with $\kappa = 2$ results in two classes corresponding to $\rho_0 =_\kappa \rho_1 =_\kappa \rho_2$ and $\rho_3 =_\kappa \rho_4$. For rank there are four classes since ρ_0, ρ_1, ρ_2 and ρ_4 have all different rankings, and $\rho_2 =_{\mathit{rank}} \rho_3$. In the case of top_1 we obtain three classes: for ρ_0 and ρ_1, the top ranked symbol is 0 and 1, resp., while for ρ_2, ρ_3 and ρ_4 it is \$. In Fig. 1 the triangle represents the simplex of distributions $\Delta(\Sigma_\$)$ and the partitions the respective equivalence classes in the example of Tab. 1. For a distribution (a point in the triangle) the probability of each symbol is given by the distance to the side opposite to the vertex representing that symbol.

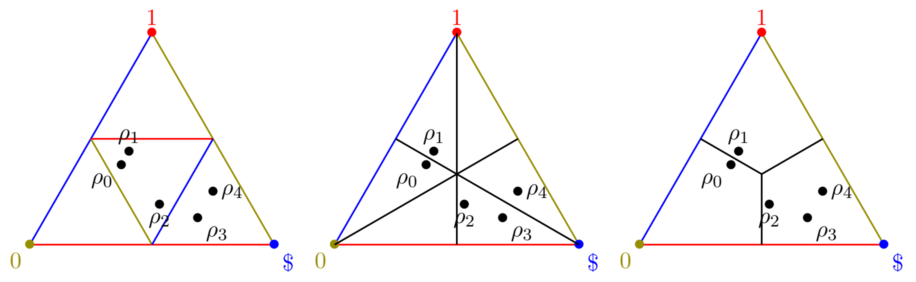

Figure 1: From left to right: $=_\kappa$, rank and top_1 equivalence classes, respectively.

2.2. Probabilistic Deterministic Finte Automata

A *probabilistic deterministic finite automaton* (PDFA) (Clark and Thollard (2004); Weiss et al. (2019)) is an instance of a language model. A PDFA A over Σ is a tuple $(Q, q_{\text{in}}, \pi, \tau)$, where Q is a finite set of states, $q_{\text{in}} \in Q$ is an initial state, $\pi : Q \to \Delta(\Sigma_\$)$ maps each state to a probability distribution over $\Sigma_{\$}$, and $\tau : Q \times \Sigma \to Q$ is the transition function. Both π and τ are total functions.

We define τ^* and π^* to be the extensions of τ and π to Σ^*, respectively: $\tau^*(q, \lambda) \triangleq q$ and $\tau^*(q, \sigma u) \triangleq \tau^*(\tau(q, \sigma), u)$, and $\pi^*(q, u) \triangleq \pi(\tau^*(q, u))$. When the state is q_{in}, we simply write $\tau^*(u)$ and $\pi^*(u)$. A defines the language model such that $A(u) \triangleq \pi^*(u)$. Now, Def. 1 can be rephrased over Q:

$$q \equiv_{\mathcal{S}} q' \overset{\triangle}{\iff} \forall w \in \Sigma^*.\ \pi^*(q, w) =_{\mathcal{S}} \pi^*(q', w) \tag{2}$$

This implies the following relationship between states and strings:

Proposition 2.2 $\forall u, u' \in \Sigma^*.\ u \equiv_{\mathcal{S}} u' \iff \tau^*(u) \equiv_{\mathcal{S}} \tau^*(u')$.
Proof Let $u, u' \in \Sigma^*$:

$$\begin{aligned}
u \equiv_{\mathcal{S}} u' &\iff \forall w \in \Sigma^*.\ A(uw) =_{\mathcal{S}} A(u'w) && \text{Def. 1}\\
&\iff \forall w \in \Sigma^*.\ \pi^*(uw) =_{\mathcal{S}} \pi^*(u'w) && A(\cdot) \triangleq \pi^*(\cdot)\\
&\iff \forall w \in \Sigma^*.\ \pi(\tau^*(uw)) =_{\mathcal{S}} \pi(\tau^*(u'w)) && \text{Def. of } \pi^*\\
&\iff \forall w \in \Sigma^*.\ \pi(\tau^*(\tau^*(u), w)) =_{\mathcal{S}} \pi(\tau^*(\tau^*(u'), w)) && \text{Def. of } \tau^*\\
&\iff \forall w \in \Sigma^*.\ \pi^*(\tau^*(u), w) =_{\mathcal{S}} \pi^*(\tau^*(u'), w) && \text{Def. of } \pi^*\\
&\iff \tau^*(u) \equiv_{\mathcal{S}} \tau^*(u') && \text{Def. 2}
\end{aligned}$$

□

Moreover, $\equiv_{\mathcal{S}}$ is a right congruence over states with respect to the transition function:

Proposition 2.3 $\forall q, q' \in Q.\ q \equiv_{\mathcal{S}} q' \implies \forall \sigma \in \Sigma.\ \tau(q, \sigma) \equiv_{\mathcal{S}} \tau(q', \sigma)$.
Proof Follows from Prop. 2.1 and Prop. 2.2. □

Let A_1 and A_2 be PDFA. Since they are language models, $A_1 \equiv_{\mathcal{S}} A_2 \iff \pi_1^*(u) =_{\mathcal{S}} \pi_2^*(u)$ for all $u \in \Sigma^*$. By Def. 2, it means their initial states are equivalent: $q_{\text{in}}^1 \equiv_{\mathcal{S}} q_{\text{in}}^2$.

A state q is *reachable* if $q = \tau^*(u)$ for some string $u \in \Sigma^*$. Any such u is called an *access* string of q. We denote *reach*(Q) the set of reachable states. Let $\overline{Q} \triangleq [\![\mathit{reach}(Q)]\!]$ be the set of equivalence classes of reachable states. We write $\overline{q}_{\text{in}}$ for $[\![q_{\text{in}}]\!]$. From Prop. 2.2, it follows that each $\overline{q} \in \overline{Q}$ can be represented by an access string u of any state in the class, i.e., $\overline{q} = [\![\tau^*(u)]\!]$. We denote $\alpha(\overline{q})$ the designated access string of $\overline{q}$. W.l.o.g., $\alpha(\overline{q}_{\text{in}}) \triangleq \lambda$.

We define the *quotient* PDFA $\overline{A}_\alpha \triangleq (\overline{Q}, \overline{q}_{\text{in}}, \overline{\pi}, \overline{\tau})$, where for all $\overline{q} \in \overline{Q}$, $\overline{\pi}(\overline{q}) \triangleq \pi^*(\alpha(\overline{q}))$, and $\overline{\tau}(\overline{q}, \sigma) \triangleq [\![\tau^*(\alpha(\overline{q})\sigma)]\!]$ for all $\sigma \in \Sigma$. It is worth noticing that all choices of α lead to isomorphic $\equiv_{\mathcal{S}}$-equivalent PDFA. Therefore, we omit α and simply write $\overline{A}$.

We illustrate the construction of the quotient with the following example. Consider the PDFA in Fig. 2(a), with $\Sigma = \{0, 1\}$. For each state q_i, $i = 0, \ldots, 4$, $\pi(q_i) = \rho_i$ defined in Tab. 1. Let us describe the quotient state set $\overline{Q}$ in each case. For $=_\kappa$, with $\kappa = 2$, the quotient $\overline{Q}$ has five elements. In fact, for $j = 3, 4$, q_j cannot be in the same class as q_i, for $i = 0, 1, 2$, since $\pi^*(q_i, \lambda) = \pi(q_i) \neq_\kappa \pi(q_j) = \pi^*(q_j, \lambda)$ (Fig. 1). The following table shows

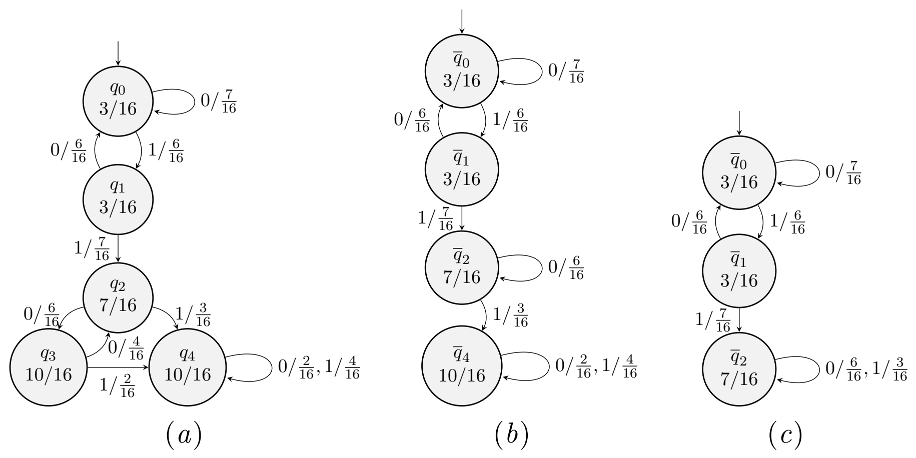

Figure 2: (a) Original PDFA (b) *rank*-Quotient PDFA (c) top_1-Quotient PDFA.

that $q_0 \not\equiv_\kappa q_i$, for $i = 1, 2$, $q_1 \not\equiv_\kappa q_2$, and $q_3 \not\equiv_\kappa q_4$:

$$\pi^*(q_0, 11) = \rho_2 \neq_\kappa \rho_4 = \pi^*(q_1, 11) \qquad \pi^*(q_0, 1) = \rho_1 \neq_\kappa \rho_4 = \pi^*(q_2, 1)$$
$$\pi^*(q_1, 1) = \rho_2 \neq_\kappa \rho_4 = \pi^*(q_2, 1) \qquad \pi^*(q_3, 0) = \rho_2 \neq_\kappa \rho_4 = \pi^*(q_4, 0)$$

In this case the quotient PDFA coincides with the original automaton.

For *rank* the quotient $\overline{Q}$ has four elements. Considering the empty word we see that q_0, q_1 and q_4 are the unique elements of their own classes. Let us show that q_2 and q_3 are equivalent. By definition we must show that $\pi^*(q_2, w) =_{rank} \pi^*(q_3, w)$ for any string w. The proof is by induction. It holds for $w = \lambda$ since $\pi(q_2) =_{rank} \pi(q_3)$. Let $w = 0u$:

$$\pi^*(q_2, 0u) \triangleq \pi^*(\tau(q_2, 0), u) = \pi^*(q_3, u) \qquad \pi^*(q_3, 0u) \triangleq \pi^*(\tau(q_3, 0), u) = \pi^*(q_2, u)$$

By induction hypothesis $\pi^*(q_2, u) =_{rank} \pi^*(q_3, u)$, so $\pi^*(q_2, 0u) =_{rank} \pi^*(q_3, 0u)$. If $w = 1u$, transitions by 1 from both q_2 and q_3 go to q_4, so in this case the equality is trivial. The quotient PDFA is shown in Fig. 2(b), for $\alpha(\overline{q}_0) = \lambda$, $\alpha(\overline{q}_1) = 1$, $\alpha(\overline{q}_2) = 11$, $\alpha(\overline{q}_4) = 111$. For top_1 a similar argument shows that the quotient $\overline{Q}$ is the one shown in Fig. 2(c).

Proposition 2.4 *For all PDFA A, $\overline{A}$ is the smallest PDFA which is $\equiv_{\mathcal{S}}$-equivalent to A.*
Proof Let $B = (Q_B, q_{\text{in}}^B, \pi_B, \tau_B)$ be another PDFA $\equiv_{\mathcal{S}}$-equivalent to A. The composition of $\alpha : \overline{Q} \to \Sigma^*$ and $\tau_B^* : \Sigma^* \to Q_B$ defines a map from $\overline{Q}$ to Q_B. We show it is one-to-one: if $\overline{q}, \overline{q}' \in \overline{Q}$ with $\tau_B^*(\alpha(\overline{q})) = \tau_B^*(\alpha(\overline{q}'))$, then for any $w \in \Sigma^*$, we have $\pi^*(\alpha(\overline{q})w) =_{\mathcal{S}} \pi_B^*(\alpha(\overline{q})w) = \pi_B^*(\alpha(\overline{q}')w) =_{\mathcal{S}} \pi^*(\alpha(\overline{q}')w)$ and therefore $\overline{q} \equiv_{\mathcal{S}} \overline{q}'$. Hence $|\overline{Q}| \leq |Q_B|$. $\square$

We say that a PDFA is *irreducible* if every pair of distinct states are not $\equiv_{\mathcal{S}}$-equivalent. In other words, a PDFA A is irreducible if it is isomorphic to its quotient $\overline{A}$.

3. Learning language models

We are interested in learning an irreducible PDFA A that approximates an unknown language model $\mathcal{L}$ through queries. For such purpose, we make use of the minimally adequate teacher (MAT) framework proposed by Angluin (1987). Here, the learner's goal is to discover the language model $\mathcal{L}$ concealed by a teacher, via two types of questions. First, a *membership* query, denoted **MQ**, returns the model's output for a given input $u \in \Sigma^*$: $\mathbf{MQ}(u) \triangleq \mathcal{L}(u)$. Second, an *equivalence* query, denoted **EQ**, takes as input a hypothesis A and an equivalence relation $\mathcal{S}$ and responds whether $\mathcal{L} \equiv_{\mathcal{S}} A$. In case they are not, it provides as evidence a counterexample $\gamma \in \Sigma^*$ such that $\mathcal{L}(\gamma) \not\equiv_{\mathcal{S}} A(\gamma)$.

For checking $\mathcal{L} \equiv_{\mathcal{S}} A$, we fall back on the probably approximately correct (PAC) framework devised by Valiant (1984). To answer an **EQ**, an *oracle* samples a subset $W \subset \Sigma^*$ using a distribution $\mathcal{D}$. If there is a string $\gamma \in W$ such that $\mathcal{L}(\gamma) \not\equiv_{\mathcal{S}} A(\gamma)$, **EQ** returns γ, otherwise it responds they are equivalent. The objective of the learner is to output a hypothesis A such that with confidence at least $1 - \delta$ (over the choice of a set W of m samples from $\mathcal{D}$, that is, $W \sim \mathcal{D}^m$) the probability of A to be a *misleading* one, that is, it passes the **EQ** while in fact it is not equivalent to $\mathcal{L}$, is at most ε. For that matter, the empirical error is $|W|^{-1} \sum_{w \in W} \mathbb{1}_{A(w) \not\equiv_{\mathcal{S}} \mathcal{L}(w)}$ and the sample W must be chosen of an appropriate size m. Following Angluin (1987), the i-th time the learner calls **EQ**, a set of size at least $\varepsilon^{-1}\big(i \ln 2 - \ln \delta\big)$ is sampled.

3.1. Tree-based PDFA learning

Here, we present a tree-based algorithm for learning PDFA, extending work in Mayr et al. (2022) where it has already been empirically shown that building a tree positively impacts efficiency in terms of computation time and structure size compared to table-based approaches. In contrast to Kearns and Vazirani (1994) and Isberner et al. (2014) which build a binary tree, our algorithm handles general trees with arbitrary degree, similar to Isberner (2015) for Mealy machines and Drewes et al. (2011) for weighted tree automata. A tree T maintains a set $Acc \subset \Sigma^*$ of access strings and a set $Dis \subset \Sigma^*$ of distinguishing strings. Each $u \in Acc$ is bound to a unique leaf in T labeled with $\mathbf{MQ}(u)$. Every inner node is labeled with a string $w \in Dis$. The empty word λ belongs to both Acc and Dis. The root of T is labeled with λ. Also, there is a leaf for λ. Arcs in T are labeled with classes in $[\![\Delta(\Sigma_\$)]\!]$. Every outgoing arc from an inner node is labeled with a different class. Strings in Acc and Dis are connected as follows: for every pair of distinct strings $u, u' \in Acc$, the string $w \in Dis$ which is the lowest common ancestor of u and u' in T, denoted $lca(u, u')$, is such that $\mathcal{L}(uw) \not\equiv_{\mathcal{S}} \mathcal{L}(u'w)$. Therefore, $u \not\equiv_{\mathcal{S}} u'$. We define $\phi_T : Acc \to [\![\Sigma^*]\!]$, with $\phi_T(u) = [\![u]\!]$. Hence, ϕ_T is one-to-one.

If $\mathcal{S}$ induces an infinite partition of $\Delta(\Sigma_\$)$, the degree of a node may be unbounded. Indeed, it may be infinite if $[\![\Sigma^*]\!]$ is infinite. Notice that this does not occur for non-binary trees used for Mealy machines (Isberner (2015)). We will come back to this issue later (in 3.2) after we discuss the learning algorithm.

Fig. 3 shows the trees corresponding to the examples in Fig. 2. For $=_\kappa$, with $\kappa = 2$, the leafs of the tree in Fig. 3(a) correspond to the states of the PDFA of Fig. 2(a), identified with their associated access strings: $Acc = \{\lambda, 1, 11, 110, 111\}$. Every leaf is labeled with the probability distribution of the state. Tree arcs are labeled with quantization classes, represented with their partition indexes. For instance, $(1,0,0)$ corresponds to the quantization class (I^1_2, I^0_2, I^0_2), where the first coordinate corresponds to the symbol 0, the second to 1, and the third to \$. The root λ of the tree has an arc for each one of the classes in which quantization partitions the set of probability distributions of the states of the PDFA. In the example, there are two, namely $(0,0,0)$ and $(0,0,1)$.

The tree explains that the five states are not $\equiv_\kappa$-equivalent. Indeed, even if there are states with $=_\kappa$-equivalent distributions, like for example q_0 and q_1, the tree shows how the set $Dis = \{\lambda, 0, 1, 11\}$ distinguishes them: $\pi^*(q_0, 11) \in (0,0,0)$ and $\pi^*(q_1, 11) \in (0,0,1)$. Fig. 3(b) corresponds to the PDFA of Fig. 2(b) obtained with *rank*. In this case we label the classes by their ranking, for example ρ_0 ranks first the symbol 0, second the symbol 1 and third the symbol \$, so it belongs to the class labeled $(0,1,\$)$. Here, $Acc = \{\lambda, 1, 11, 111\}$ and $Dis = \{\lambda\}$, respectively. Fig. 3(c) corresponds to the PDFA of Fig. 2(c) obtained with top_1. In this case we label the classes by their top symbol, for example ρ_2 ranks first the symbol \$, so it belongs to the class labeled \$. Also, $Acc = \{\lambda, 1, 11\}$ and $Dis = \{\lambda\}$.

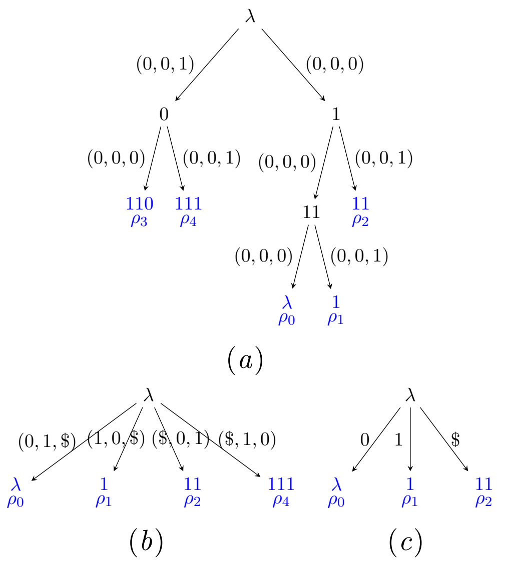

Figure 3: (a) Tree for Fig. 2(a); (b) Tree for Fig. 2(b); (c) Tree for Fig. 2(c).

T defines an equivalence $=_T \subseteq \Sigma^* \times \Sigma^*$. Let $\zeta_u \subseteq Dis$ be the set of distinguishing strings in the path from the root to $u \in Acc$:

$$u_1 =_T u_2 \overset{\triangle}{\iff} \exists u \in Acc.\ \forall w \in \zeta_u.\ \mathcal{L}(u_1 w) =_\mathcal{S} \mathcal{L}(uw) =_\mathcal{S} \mathcal{L}(u_2 w) \tag{3}$$

If $v \in \Sigma^*$ is $=_T$-equivalent to some $u \in Acc$ then $v \not\equiv_\mathcal{S} u'$ for any other access string $u' \neq u$. As a corollary, v can be $=_T$-equivalent to at most one access string in T.

Proposition 3.1 *Let* $u \in Acc,\ v \in \Sigma^*,\ u =_T v.\ \forall u' \in Acc.\ u' \neq u \implies [\![v]\!] \neq [\![u']\!]$.

Proof Let $u' \in Acc$, $u' \neq u$. Then, $\mathcal{L}(u'w) \neq_\mathcal{S} \mathcal{L}(uw)$ for $w = lca(u', u)$. Since w is an ancestor of u, it follows that $w \in \zeta_u$. So, by Def. 3, $u =_T v$ implies $\mathcal{L}(uw) =_\mathcal{S} \mathcal{L}(vw)$. Therefore, $\mathcal{L}(u'w) \neq_\mathcal{S} \mathcal{L}(vw)$. Hence, $[\![v]\!] \neq [\![u']\!]$. □

To find the $=_T$-equivalent leaf (if exists) of v, we define the function $sift(v)$ as follows. *sift* starts at the root of T and proceeds recursively. If the current node is a leaf $u \in Acc$, it returns u. Otherwise, let $w \in Dis$ be the distinguishing string at the current inner node. If there is an arc labeled $[\![\mathbf{MQ}(vw)]\!]$, *sift* recursively descends through the arc to the subtree. Otherwise, it means $\mathbf{MQ}(vw) \neq_{\mathcal{S}} \mathbf{MQ}(uw)$ for all descendant leafs u of w. In such case, *sift* updates the tree as follows: it adds v to Acc labeled with $\mathbf{MQ}(v)$ and a new arc from w to v labeled with $[\![\mathbf{MQ}(vw)]\!]$, and it returns v.[1] It is important to remark that, in case *sift* modifies T, only its degree can grow but never its depth.

Proposition 3.2 *For all $v \in \Sigma^*$, sift(v) returns a leaf u such that $v =_T u$ and $[\![v]\!] \neq [\![u']\!]$ for all leafs $u' \neq u$. Also, sift maintains the properties of the (possibly updated) tree.*
Proof Suppose *sift*(v) returns an existing leaf u. It means it recursively traversed the path ζ_u from the root to u. Then, by Def. 3, $v =_T u$, and by Prop. 3.1, $\forall u' \in Acc.\ u' \neq u \implies [\![v]\!] \neq [\![u']\!]$. Suppose *sift*$(v)$ returns v after updating the tree. Obviously, $v =_T v$. Now, this case occurs because *sift* arrives at an inner node with label $w \in Dis$ for which there is no outgoing arc $=_{\mathcal{S}}$-equivalent to $\mathcal{L}(vw)$. Then, $[\![v]\!] \neq [\![u']\!]$ for any descendant leaf u' of the current inner node because $\mathcal{L}(vw) \neq_{\mathcal{S}} \mathcal{L}(u'w)$, and *sift* ensures $w = lca(u', v)$. Also, $[\![v]\!] \neq [\![u']\!]$ for any other leaf u' in the tree because $\mathcal{L}(vw') \neq_{\mathcal{S}} \mathcal{L}(u'w')$ for $w' = lca(u', w)$, and *sift* guarantees $w' = lca(u', v)$. Therefore, $[\![v]\!] \neq [\![u']\!]$ for all already existing leafs u'. Moreover, the newly updated tree is such that every pair of distinct leafs is distinguished by its lowest common ancestor and so ϕ is one-to-one. □

Given T, *build* constructs a PDFA $A \triangleq (Q, q_{\text{in}}, \pi, \tau)$ where: $Q \triangleq \{q_u \mid u \in Acc\}$ with $\alpha(q_u) \triangleq u$; $q_{\text{in}} \triangleq q_\lambda$; and for all $q_u \in Q$, $\pi(q_u) \triangleq \mathcal{L}(u)$, and for all $\sigma \in \Sigma_\$$, $\tau(q_u, \sigma) \triangleq q_{u'}$ with $u' = sift(u\sigma)$. Whenever *sift* adds a new leaf, it is restarted with the updated tree.

Proposition 3.3 *1) If build terminates, the output A is irreducible. 2) If $[\![\Sigma^*]\!]$ is finite, $|Q|$ is bounded by the size of $[\![\Sigma^*]\!]$. 3) If one of $[\![\Delta(\Sigma_\$)]\!]$ or $[\![\Sigma^*]\!]$ is finite, build terminates.*
Proof Since ϕ_T is one-to-one, $q \neq q' \implies q \not\equiv_{\mathcal{S}} q'$. This implies that, if *build* terminates, A is irreducible, and if $[\![\Sigma^*]\!]$ is finite, $|Q|$ is at most the size of $[\![\Sigma^*]\!]$ and also, by Prop. 3.2, *build* terminates. If $[\![\Delta(\Sigma_\$)]\!]$ is finite, the degree of T is bounded by the size of $[\![\Delta(\Sigma_\$)]\!]$. Since *sift* can only make T to grow in width, it follows *build* terminates. □

If either $[\![\Delta(\Sigma_\$)]\!]$ or $[\![\Sigma^*]\!]$ is infinite, T could grow without bound. We address this issue in 3.2. Assume A is constructed. If T and A agree for some $v \in \Sigma^*$ but differ in $v\sigma$ for some $\sigma \in \Sigma_\$$ then $[\![v]\!]$ is not a leaf. Moreover, v and *sift*(v) are not $\equiv_{\mathcal{S}}$-equivalent. Formally:

Proposition 3.4 *Let $v \in \Sigma^*, \sigma \in \Sigma$. If sift$(v) = \alpha(\tau^*(v))$ and sift$(v\sigma) \neq \alpha(\tau^*(v\sigma))$ then (1) $v \notin Acc$, and (2) sift$(v) \not\equiv_{\mathcal{S}} v$.*
Proof Let $u_1 = sift(v)$, $u_2 = sift(v\sigma)$, $u_2' = \alpha(\tau^*(v\sigma))$, and $w = lca(u_2, u_2')$.
(1) Suppose $v \in Acc$. By Prop. 3.2, $u_1 = v$, and by construction, $\tau(q_v, \sigma) = q_{u_2}$. Besides, $\tau^*(v\sigma) = \tau(\tau^*(v), \sigma) = \tau(q_v, \sigma)$. Then, $\tau^*(v\sigma) = q_{u_2}$. So, $u_2' = \alpha(\tau^*(v\sigma)) = \alpha(q_{u_2}) = u_2$ which contradicts the hypothesis $u_2 \neq u_2'$. Hence, $v \notin Acc$.

1. *sift* updates are necessary since arcs are discovered on-the-fly because $[\![\Delta(\Sigma_\$)]\!]$ may be unbounded.

(2) (i) By Prop. 3.2, $u_2 = \mathit{sift}(v\sigma)$ implies $v\sigma =_T u_2$. So, $\mathcal{L}(v\sigma w) =_{\mathcal{S}} \mathcal{L}(u_2 w)$ since $w \in \zeta_{u_2}$.
(ii) From $u_1 = \alpha(\tau^*(v))$ and $u'_2 = \alpha(\tau^*(v\sigma))$, it follows by construction of A that $u'_2 = \mathit{sift}(u_1\sigma)$. Then, by Prop. 3.2, $u_1\sigma =_T u'_2$. So, $\mathcal{L}(u_1\sigma w) =_{\mathcal{S}} \mathcal{L}(u'_2 w)$ since $w \in \zeta_{u'_2}$.
Therefore, from $\mathcal{L}(u_2 w) \not\equiv_{\mathcal{S}} \mathcal{L}(u'_2 w)$, (i) and (ii) it follows that $\mathcal{L}(v\sigma w) \not\equiv_{\mathcal{S}} \mathcal{L}(u_1\sigma w)$. So, $u_1 \not\equiv_{\mathcal{S}} v$. Hence, $\mathit{sift}(v) \not\equiv_{\mathcal{S}} v$. □

Assume $\mathbf{EQ}(A)$ returns a counterexample γ. Let $u_i = \mathit{sift}(\gamma[i])$ and $u'_i = \alpha(\tau^*(\gamma[i]))$. Since $\mathcal{L}(\gamma) \not\equiv_{\mathcal{S}} \pi^*(\gamma)$, there is some i such that $u_i \neq u'_i$. Let j be the first such index. Therefore, Prop. 3.4 implies $\gamma[j-1] \notin \mathit{Acc}$ and $u_{j-1} \not\equiv_{\mathcal{S}} \gamma[j-1]$. Moreover, $\gamma_j w$ distinguishes u_{j-1} and $\gamma[j-1]$ with $w = \mathit{lca}(u_j, u'_j)$. The procedure *update* modifies T with this new evidence as follows: $\gamma[j-1]$ is added to Acc, and the leaf u_{j-1} is replaced by an inner node $\gamma_j w$ and two children, namely u_{j-1} and $\gamma[j-1]$. *update* maintains the properties of the tree.

Algorithm 1 sketches the code. It starts creating an initial A, with a single state q_λ with a loop for each symbol and probability distribution $\mathbf{MQ}(\lambda)$. Then, it calls $\mathbf{EQ}(A)$, which either returns $\bot$ and terminates or a counterexample γ triggering the initialization of T. The first instance of T has a root λ and two children, one labeled λ and the other γ. In the main loop, Algorithm 1 uses T to build a PDFA A, then calls $\mathbf{EQ}(A)$. If a counterexample is returned, T is updated and the loop restarts. Otherwise, it terminates. Algorithm 1 learns a PAC-equivalent PDFA.

Algorithm 1 Tree-based learning algorithm.

```
Parameter: Equivalence relation $\mathcal{S}$
Output   : PDFA $A$
$A \leftarrow$ CreateInitialHypothesis($\mathcal{S}$);   $\gamma \leftarrow \mathbf{EQ}(A, \mathcal{S})$
if $\gamma = \bot$ then
    return $A$
end
$T \leftarrow$ InitializeTree($\gamma, \mathcal{S}$)
while $\gamma \neq \bot$ do
    $A \leftarrow \mathit{build}(T)$;   $\gamma \leftarrow \mathbf{EQ}(A, \mathcal{S})$
    if $\gamma \neq \bot$ then
        $T \leftarrow \mathit{update}(T, \gamma, \mathcal{S})$
    end
end
return $A$
```

Proposition 3.5 (Correctness) *If Algorithm 1 terminates, it outputs an irreducible PDFA which is $\equiv_{\mathcal{S}}$-equivalent (in the sense of PAC) to $\mathcal{L}$.*
Proof Let A be the output of Algorithm 1. Then, $A \equiv_{\mathcal{S}} \mathcal{L}$ (in the sense of PAC) since $\mathbf{EQ}(A)$ does not return a counterexample. By Prop. 3.3(1), A is irreducible. □

Proposition 3.6 (Strict progress) *Let A_i be the PDFA built by Algorithm 1 at iteration i. If $\mathbf{EQ}(A_i) \neq \bot$ then A_{i+1}, if built, has strictly more states than A_i.*
Proof If $\mathbf{EQ}(A_i) \neq \bot$, *update* adds a new leaf to T. Hence, $|Q_{i+1}| > |Q_i|$. □

Corollary 3.1 *Algorithm 1 always terminates for $\mathcal{S}$-regular languages.*
Proof From Prop. 3.6, Prop. 3.3, Prop. 2.4, and Prop. 2.2.

3.2. Addressing infinite equivalence classes

If either $[\![\Delta(\Sigma_\$)]\!]$ or $[\![\Sigma^*]\!]$ is infinite, Algorithm 1 is not guaranteed to terminate. Indeed, even in the case that termination is theoretically certain, it may take too long to finish. The learning algorithm could be forced to stop by imposing some kind of bound to its execution. Mayr and Yovine (2018) proposes finishing whenever the hypothesis automaton exceeds a maximum number of states or the string passed to a membership query $\mathbf{MQ}$ (wherever it happens) is longer than a certain length. If the length bound is reached, it may occur that some transitions are not properly defined because the corresponding destination state has not yet been discovered. In such case, the last completely constructed automaton is returned. The size condition is checked after the hypothesis automaton is constructed, therefore it can only happen if the length one did not occur before. Weiss et al. (2019) resorts to stopping conditions that depend on the algorithm's data structure, together with an execution time bound. As for length cutoff, when time is exhausted, some transitions may have a missing destination. In this case, their algorithm uses the state whose probability distribution is the closest one, with respect to an appropriate distance. Here, we adopt the above three stopping criteria. When any of them occurs, an exception is launched and the algorithm proceeds to constructing a hypothesis PDFA. Possibly, a missing destination of a transition labeled σ from a state q could be found during the execution of $\mathit{sift}(\alpha(q)\sigma)$ if there is no outgoing arc $[\![\mathbf{MQ}(\alpha(q)\sigma w)]\!]$ for some inner node w. In normal mode, this situation results in T to be updated with a new leaf and the construction to be restarted. In exception mode, however, state q is connected by σ to a dummy *unknown* state which acts as a sink. A hypothesis with a reachable unknown state is called a *partial* PDFA to stress the fact that the learner would have produced a larger (possibly complete) PDFA provided more resources were given. Actually, by Prop. 3.3(3), if $[\![\Delta(\Sigma_\$)]\!]$ is finite, as it is the case for quantization, *rank* and top_r, for instance, it could be possible to let *build* run until termination, and so produce a PDFA with no *unknown* state. However, this may be too costly.

3.3. Checking equivalence between language models by learning

Indeed, the idea of partial PDFA can be used for other purposes. Consider the partial function f defined as follows: $f(w) \triangleq \mathcal{L}_1(w)$ if $\mathcal{L}_1(w) =_{\mathcal{S}} \mathcal{L}_2(w)$, otherwise $f(w) \triangleq \perp$ (undefined). Clearly, f is total and equal to $\mathcal{L}_1$ if and only if $\mathcal{L}_1 \equiv_{\mathcal{S}} \mathcal{L}_2$. As before, Algorithm 1 can be adapted to produce PDFA with a state for $\perp$. Therefore, learning a PDFA from f will either give the quotient of $\mathcal{L}_1$ and prove the equivalence, or return a partial PDFA having a $\perp$ state and so proving they are not equivalent, or return a partial PDFA with no $\perp$-state which results in an inconclusive verdict.

4. Experiments

We analyze case studies from genetics and cybersecurity application domains, using several equivalences[2]. Similar to Craven and Shavlik (1995), $\mathcal{D}$ is the empirical distribution of the available network training data. Specifically, to evaluate an **EQ** the oracle draws a set $W \sim \mathcal{D}^m$ which corresponds to a bootstrap sample (Efron and Tibshirani (1993)) of the given training data.

4.1. TATA-boxes in DNA promoter sequences

DNA promoter sequences control the activation or repression of genes. It is a sequence of length 6 characterized by totalling more A's and T's than C's and G's.

Several works have applied deep learning for classification tasks involving TATA-boxes, e.g., Oubounyt et al. (2019). Here, we use an LSTM-based language model trained on 1400 TATA-boxes of human DNA from EPDnew[3]. Table 2 shows the results for top_1, $rank$, $=_{10}$ and a TATA-specific equivalence induced by $P_{\mathcal{L}}[T|w] + P_{\mathcal{L}}[A|w] \geq P_{\mathcal{L}}[G|w] + P_{\mathcal{L}}[C|w]$. PAC parameters are $\varepsilon = \delta = 0.05$. All experiments terminated with a successful **EQ**. Thus, $A \equiv_{\mathcal{S}} \mathcal{L}$ in the PAC sense. Column (d, n) shows the depth d and the number of internal nodes n of T. Column $|[\![\Delta(\Sigma_\$)]\!]|$ gives an approximation of the number of classes defined by $\mathcal{S}$. Note that coarser $\mathcal{S}$ tend to generate smaller Q.

$\mathcal{S}$	$\lvert Q\rvert$	(d,n)	$\lvert[\![\Delta(\Sigma_\$)]\!]\rvert$	**EQ**	Sec.
$=_{10}$	811	(6, 17)	5×10^4	102	328.0
rank	299	(6, 19)	120	86	130.3
top_1	79	(10, 23)	5	55	65.6
TATA	54	(13, 32)	2	54	71.1

Table 2: TATA-Box results

4.2. Language model for generating normal HDFS traces

We trained an LSTM-based language model on a dataset of 4800 normal Hadoop File System (HDFS) logs from Du et al. (2017), which are sequences from a set of 30 symbols including \$. There, the language model is used to classify logs: if there is a prefix for which its next symbol is not one of the top_r symbols, the log is consider abnormal. Otherwise, it is classified as normal. Table 3 shows results for $rank$, top and $=_\kappa$ for several values of r and κ, with $\varepsilon = \delta = 0.05$. All experiments exhausted a time bound of 10K seconds and generated partial PDFA. To assess the error of the outputs, we sampled a test set of size 1K from available data. "Unk" shows the part of the error corresponding to the fraction of logs in the test set for which A reaches the *unknown* state. This error is included in the total test error shown in column "Err". We observe that coarser equivalences generated

2. Code available at: https://github.com/neuralchecker/pymodelextractor_congruence_approach_to_active_automata_learning_from_neural_LM
3. https://epd.epfl.ch//index.php

$\mathcal{S}$	$\lvert Q \rvert$	(d, n)	$\lvert [\![\Delta(\Sigma_\$)]\!] \rvert$	**EQ**	Unk.	Err.
top_3	1616	(9, 56)	4.1×10^3	153	0.00	0.23
$rank_3$	2008	(6, 43)	2.4×10^4	105	0.02	0.39
top_6	2087	(8, 40)	5.9×10^5	87	0.00	0.66
$rank_6$	8837	(2, 1)	4.3×10^8	1	0.91	1.00
$=_3$	660	(14, 148)	1.0×10^{14}	326	0.00	0.03
$=_5$	3014	(6, 20)	4.7×10^{20}	78	0.01	0.80
$=_{10}$	7289	(2, 1)	5.0×10^{29}	1	0.01	0.62

Table 3: HDFS results

automata with fewer states and, generally, smaller total errors. It is of interest to analyze how $|Q|$, **EQ**, "Unk" and "Err" evolve as the execution time augments.

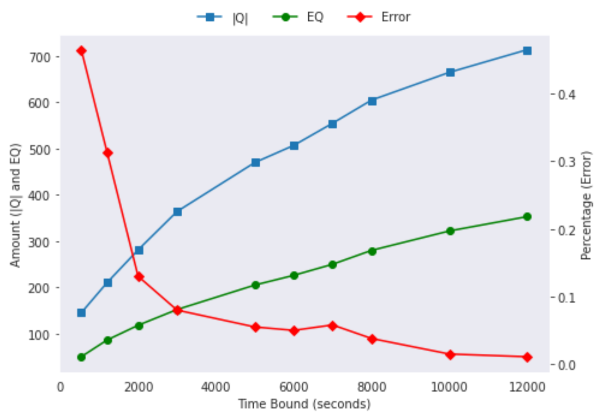

Figure 4: HDFS results by time

For that purpose, the algorithm was run for equivalence $=_3$ with increasing time bounds ranging from 120 to 12K seconds and measure them. In these experiments, "Unk" was always 0. The other obtained results are depicted in Fig. 4. The curves show that $|Q|$ (as stated in Prop. 3.6) monotonically increases. "Err" tends to decrease but there are some spikes.

4.3. Detection of malicious web requests

Here, we study the equivalence between two character-level Transformer language models presented in Martínez (2022), pre-trained with malicious web requests from Li et al. (2020). Σ consists of 256 ASCII characters. $\mathcal{L}_1$ has $27K$ parameters and $\mathcal{L}_2$ has $6.5M$ but both have an accuracy greater than 0.97 for predicting the next character given the last 10 observed ones. We ran the algorithm for 60 seconds with the goal of learning whether they were equivalent for top_1.

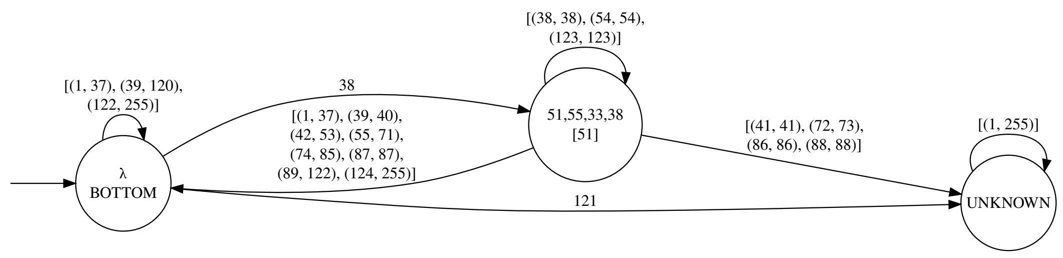

Figure 5: Automaton showing both Transformers are not top_1-equivalent

Fig. 5 depicts a simplified visualization of the returned PDFA where probability distributions are omitted and multiple transitions with same end points are collapsed into a single one labeled with a set of intervals of symbols. It shows that both Transformers are not top_1-equivalent since $\mathcal{L}_1(\lambda) \neq_{top_1} \mathcal{L}_2(\lambda)$. However, they agree in the class of access string $51, 55, 33, 38$ corresponding to 51 as the most probable symbol. It is worth noting that attempting to learn $\mathcal{L}_1$ and $\mathcal{L}_2$ resulted in high Unk and Err even after a 12K secs.

5. Related Work

Previous works have used different techniques to relate models by the similarity of their outputs in the context of probabilistic and weighted automata over sequences and trees. In Clark and Thollard (2004), models are related through their probability over strings instead of their next-symbol probability by using a non-transitive relation based on the L_∞-norm and a *tolerance* parameter t. Weiss et al. (2019) adapts this relation to next-symbol probability distributions and calls it t-equality. Mayr et al. (2022) uses quantization to define an equivalence relation over next-symbol probabilities which leads to a congruence over strings. It is also shown that it is always possible to come up with a quantization that implies t-equality. Drewes et al. (2011) uses an equivalence relation based on the output weight of the model for the input tree. Their equivalence is based on proportionality of outputs defined over the product operation and equality on the output space. When interpreted over language models it corresponds to equality of next-symbol distributions. Moreover, all these algorithms strictly rely on a fixed relation over models. On the contrary, our learning algorithm is parameterized by an externally-defined equivalence relation over the output space. Besides, our approach is completely agnostic to the way the equivalence is actually defined.

6. Conclusions

This work investigated the problem of learning quotient language models modulo arbitrary equivalence relations on distributions. Similar to Myhill-Nerode congruence for regular languages, for any equivalence on distributions $\mathcal{S}$ a language model induces a congruence over strings, and leads to correspondences between language models and automata, parameterized by $\mathcal{S}$. A practical motivation for this is that $\mathcal{S}$ can be chosen to fit a desired performance metric on language models, such as *word error rate* (which corresponds to top_1-equivalence, or problem-specific ones, such as the one for the TATA-box case study. This leads to quotient models that abstract away details which are unnecessary for the purpose of the analysis. The size of such models is likely to be smaller than a full fledged one. Besides, we developed a learning algorithm which uses a tree of finite but unbounded width to learn an irreducible PDFA which is PAC-equivalent to the target language model. The algorithm is guaranteed to terminate whenever the congruence induced by the language model for the given equivalence on distributions is finite. We used the algorithm on several case studies from cybersecurity and genetics, including pre-trained neural language models. The experimental results showcased that parameterizing the learning process would help in some cases to overcome the so-called state explosion problem when building state-based models, by choosing appropriate equivalence relations that lead to property-preserving abstractions. Moreover, if the state-space still remains too large, learning can be used to directly verify properties such as the equivalence between language models. An interesting direction for future work is to extend our approach to learning automata-based transducers (e.g., Akram et al. (2012) and references therein) from neural ones (e.g., Graves (2012)).

Acknowledgements Research reported in this article has been partially funded by ANII-Agencia Nacional de Investigación e Innovación under grant IA_1_2022_1_173516.

References

H. I. Akram, C. de la Higuera, and C. Eckert. Actively learning probabilistic subsequential transducers. In *ICGI*, volume 21 of *PMLR*, pages 19–33, 2012.

D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Com.*, 75, 1987.

B. Bollig, M. Leucker, and D. Neider. A survey of model learning techniques for recurrent neural networks. In *LNCS 13560*, pages 81–97. Springer Nature Switzerland, 2022.

A. Clark and F. Thollard. PAC-learnability of probabilistic deterministic finite state automata. *J. Machine Learning Research*, 5:473–497, 2004.

M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In *NIPS'95*, page 24–30, Cambridge, MA, USA, 1995. MIT Press.

F. Drewes, J. Högberg, and A. Maletti. MAT learners for tree series: an abstract data type and two realizations. *Acta Informatica*, 48:165–189, 2011.

M. Du, F. Li, G. Zheng, and V. Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *SIGSAC CCS*, page 1285–1298. ACM, 2017.

B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 1993.

A. Graves. Sequence transduction with recurrent neural networks. *CoRR*, abs/1211.3711, 2012.

M. Isberner. *Foundations of Active Automata Learning: An Algorithmic Perspective*. PhD thesis, TU Dortmund University, 2015.

M. Isberner, F. Howar, and B. Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In *RV'14*, pages 307–322. LNCS 8734, 2014.

M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994.

J. Li, H. Zhang, and Z. Wei. The weighted word2vec paragraph vectors for anomaly detection over http traffic. *IEEE Access*, 8:141787–141798, 2020.

N. Martínez. Comparison of lstm and transformer neural network on multiple approaches for weblogs attack detection. Master's thesis, Universidad ORT Uruguay, 2022.

F. Mayr and S. Yovine. Regular inference on artificial neural networks. In *Machine Learning and Knowledge Extraction*, pages 350–369, Cham, 2018. Springer.

F. Mayr, S. Yovine, F. Pan, N. Basset, and T. Dang. Towards efficient active learning of PDFA. *CoRR*, abs/2206.09004, 2022.

M. Oubounyt, Z. Louadi, H. Tayara, and K.-T. Chong. Deepromoter: Robust promoter predictor using deep learning. *Frontiers in genetics*, 10, 2019.

N. Scheiner, N. Appenrodt, J. Dickmann, and B. Sick. Radar-based road user classification and novelty detection with RNN ensembles. In *IEEE IVS*, pages 722–729, 2019.

S. A. Seshia, D. Sadigh, and S. Sastry. Toward verified artificial intelligence. *Communications of the ACM*, 65(7):46–55, july 2022.

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.

G. Weiss, Y. Goldberg, and E. Yahav. Learning deterministic weighted automata with queries and counterexamples. In *NeurIPS*, volume 32, 2019.