Interpretable Sequence Classification via Discrete Optimization*

Maayan Shvo,[1,2,3] **Andrew C. Li,**[1,2] **Rodrigo Toro Icarte,**[1,2] **Sheila A. McIlraith**[1,2,3]
[1] Department of Computer Science, University of Toronto, Toronto, Canada
[2] Vector Institute, Toronto, Canada
[3] Schwartz Reisman Institute for Technology and Society, Toronto, Canada
{maayanshvo,andrewli,rntoro,sheila}@cs.toronto.edu

Abstract

Sequence classification is the task of predicting a class label given a sequence of observations. In many applications such as healthcare monitoring or intrusion detection, early classification is crucial to prompt intervention. In this work, we learn sequence classifiers that favour early classification from an evolving observation trace. While many state-of-the-art sequence classifiers are neural networks, and in particular LSTMs, our classifiers take the form of finite state automata and are learned via discrete optimization. Our automata-based classifiers are interpretable—supporting explanation, counterfactual reasoning, and human-in-the-loop modification—and have strong empirical performance. Experiments over a suite of goal recognition and behaviour classification datasets show our learned automata-based classifiers to have comparable test performance to LSTM-based classifiers, with the added advantage of being interpretable.

1 Introduction

Sequence classification—the task of predicting a class label given a sequence of observations—has a myriad of applications including biological sequence classification (e.g., Deshpande and Karypis 2002), document classification (e.g., Sebastiani 2002), and intrusion detection (e.g., Lane and Brodley 1999). In many settings, early classification is crucial to timely intervention. For example, in hospital neonatal intensive care units, early diagnosis of infants with sepsis (based on the classification of sequence data) can be life-saving (Griffin and Moorman 2001).

Neural networks such as LSTMs (Hochreiter and Schmidhuber 1997), learned via gradient descent, are natural and powerful sequence classifiers (e.g., Zhou et al. 2015; Karim et al. 2019), but the rationale for classification can be difficult for a human to discern. This is problematic in many domains, where machine decision-making requires a degree of accountability in the form of verifiable guarantees and explanation for decisions (Doshi-Velez and Kim 2017).

In this work, we use discrete optimization to learn interpretable classifiers that favour early classification. In particular, we learn a suite of binary classifiers that take the form of finite state automata, each capturing the rationale for classification in a compact extractable form. To classify a sequence of observations, we then employ Bayesian inference to produce a posterior probability distribution over the set of class labels. Importantly, our automata-based classifiers, by virtue of their connection to formal language theory, are both generators and recognizers of the pattern language that describes each behavior or sequence class. We leverage this property in support of a variety of interpretability services, including explanation, counterfactual reasoning, verification of properties, and human modification.

Previous work on learning automata from data has focused on learning minimum-sized automata that perfectly classify the training data (e.g., Gold 1967; Angluin 1987; Oncina and Garcia 1992; Ulyantsev, Zakirzyanov, and Shalyto 2015; Angluin, Eisenstat, and Fisman 2015; Giantamidis and Tripakis 2016; Smetsers, Fiterău-Broştean, and Vaandrager 2018). Nonetheless, such approaches learn large, overfitted models in noisy domains that generalize poorly to unseen data. We propose novel forms of regularization to improve robustness to noise and introduce an efficient mixed integer linear programming model to learn these automata-based classifiers. Furthermore, to the best of our knowledge, this is the first work that proposes automata for early classification. Experiments on a collection of synthetic and real-world goal recognition and behaviour classification problems demonstrate that our learned classifiers are robust to noisy sequence data, are well-suited to early prediction, and achieve comparable performance to an LSTM, with the added advantage of being interpretable.

In Section 2, we provide necessary background and introduce our running example. In Section 3, we discuss our method for learning DFA sequence classification models and elaborate on the interpretability services afforded by these models in Section 4. In Section 5, we discuss the experimental evaluation of our approach on a number of goal recognition and behaviour classification domains, and in Section 6 we situate our work within the body of related work, followed by concluding remarks. For detailed discussion of the data we use in experiments, background on Linear Temporal Logic, examples of learned automata classifiers, and details of our experimental setup, the reader is directed to the technical appendix associated with this work (Shvo et al. 2020).

*Associated technical appendix appears in (Shvo et al. 2020).

2 Background and Running Example

The class of problems we address comprises symbolic time-series classification problems that require discrimination of a set of potential classes, where early classification may be favored, data may be noisy, and an interpretable, and ideally queryable, classifier is either necessary or desirable.

We define the sequence classification problem as follows.

Definition 2.1 (Sequence Classification) *Given a trace $\tau = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, $\sigma_i \in \Sigma$, where Σ is a finite set of symbols, and $\mathcal{C}$ is a set of class labels, sequence classification is the task of predicting the class label $c \in \mathcal{C}$ that corresponds to τ.*

The observation trace, τ, is typically assumed to encode the entire trace. However, we also examine the *early classification* setting, where it is desirable to produce a high confidence classification with a small prefix of the entire trace (e.g., Griffin and Moorman 2001; Xing, Pei, and Keogh 2010; Ghalwash, Radosavljevic, and Obradovic 2013).

We propose the use of Deterministic Finite Automata (DFA) as sequence classifiers.

Definition 2.2 (Deterministic Finite Automaton) *A Deterministic Finite Automaton is a tuple $\mathcal{M} = \langle Q, q_0, \Sigma, \delta, F \rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite set of symbols, $\delta : Q \times \Sigma \to Q$ is the state-transition function, and $F \subseteq Q$ is a set of accepting states.*

Given a sequence of input symbols $\tau = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, $\sigma_i \in \Sigma$, a DFA $\mathcal{M} = \langle Q, q_0, \Sigma, \delta, F \rangle$ transitions through the sequence of states $s_0, s_1, \ldots, s_n$ where $s_0 = q_0$, $s_i = \delta(q_{i-1}, \sigma_i)$ for all $1 \le i \le n$. $\mathcal{M}$ *accepts* τ if $s_n \in F$, otherwise, $\mathcal{M}$ *rejects* τ.

A DFA provides a compact graphical encoding of a *language*, a set of (potentially infinite) traces accepted by the DFA. The class of languages recognized by DFAs is known collectively as the *regular languages*. In Section 4 we employ formal language theory to straightforwardly propose a set of interpretability services over our DFA classifiers.

We use the following goal recognition problem as a running example to help illustrate concepts.

Example 2.1 (The office domain) *Consider the environment shown in Figure 1. We observe an agent that starts at one of **A**, **B**, or **E** with the goal of reaching one of the other coloured regions, $\mathcal{C} = \{$**A**, **B**, **E**, ☕, 🚺, 🚹$\}$, using* only *the hallways **H1**, **H2**, and **H3**. The agent always takes the shortest Manhattan distance path to the goal, choosing uniformly at random if multiple shortest paths exist. E.g., an agent starting at **B** with goal ☕ will pursue paths (**B**, **H2**, **H1** ☕) and (**B**, **H1**, ☕). We wish to predict the agent's goal as early as possible, given a sequence of observed locations.*

Figure 1 shows a binary DFA classifier that predicts if an agent is trying to achieve the ☕ goal in the office domain. This DFA was **learned** from the set of all *valid* traces and corresponding goals (as stipulated in Example 2.1) using the method discussed in Section 3. In contrast to the rich body of work on goal recognition that leverages models of the domain to recognize an actor's goal, here we do not have a model and instead learn a classifier directly from a given set of traces. The DFA predicts whether the agent would achieve the ☕ goal by keeping track of the agent's locations over time. Its input symbols are $\Sigma = \{$**A**, **B**, **H1**, **H2**, **H3**, **E**, 🚺, 🚹, ☕ $\}$ and the only accepting state is $q_2 \in F$. A decision is provided after each incoming observation based on the current state: **yes** for the blue accepting state, and **no** for red, non-accepting states. For example, on the trace (**B**, **H2**, **H1**, ☕) the DFA would transition through the states (q_0, q_3, q_0, q_2), predicting that the goal is *not* ☕ after the first three observations, then predicting the goal *is* ☕ after the fourth observation.

Note that this learned DFA leverages biases in the data—namely, that in the training data the agent only pursues optimal paths. However, in addition to correctly classifying all optimal paths, the DFA also generalizes to some unseen traces. For example, the DFA correctly classifies the trace corresponding to the suboptimal path (**B**, **H3**, **H2**, **H1**, ☕).

Finally, the DFA in Figure 1 only predicts the goal ☕ once ☕ is observed. Another DFA trained to detect goal **E** (not shown) highlights early detection by predicting goal **E** once **H3** is observed, unless the agent started at **E**. This is correct since when the agent starts at **A** or **B**, the observation **H3** only appears on optimal paths to **E**.

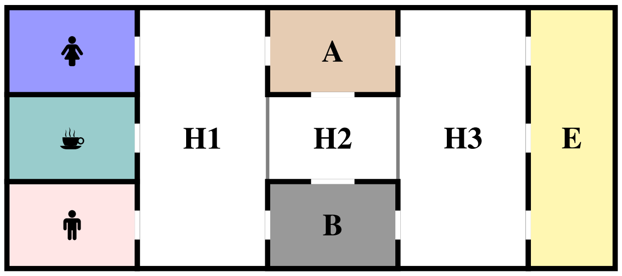
3 Learning DFAs for Sequence Classification

In this section, we describe our method for learning DFA sequence classification models from a set of training traces and corresponding class labels $\{(\tau_1, c_1), \ldots, (\tau_N, c_N)\}$. We adopt the standard supervised learning assumption that each $(\tau_i, c_i) \overset{iid}{\sim} p(\tau, c)$, and the objective of maximizing the predictive accuracy of the model over $p(\tau, c)$. For each possible label $c \in \mathcal{C}$, we train a separate DFA $\mathcal{M}_c$ responsible for recognizing traces with label c (see Section 3.1). At test time, given a trace (or a partial trace) τ, all $|\mathcal{C}|$ DFAs are evaluated independently on τ, and the collective decisions of the DFAs are used to produce a posterior probability distribution over $\mathcal{C}$ (see Section 3.2).

3.1 Learning One-vs-Rest Binary Classifiers

We now describe our *Mixed Integer Linear Programming (MILP)* model to learn DFAs. We rely on MILP solvers because they are the state of the art for solving a wide range of discrete optimization problems and they are guaranteed to find optimal solutions given sufficient resources (Jünger et al. 2009).

Given a training set $\{(\tau_1, c_1), \ldots, (\tau_N, c_N)\}$, we learn one DFA $\mathcal{M}_c$ per each label $c \in \mathcal{C}$ responsible for discriminating traces in class c from traces not in class c. We start by representing the whole training set as a *Prefix Tree (PT)* (De la Higuera 2010)—this is a common preprocessing step in the automata learning literature (Giantamidis and Tripakis 2016). PTs are incomplete DFAs with no accepting states. They are incomplete in the sense that some of their transitions are unspecified. Given a training set $\{(\tau_1, c_1), \ldots, (\tau_N, c_N)\}$, we can construct (in poly-



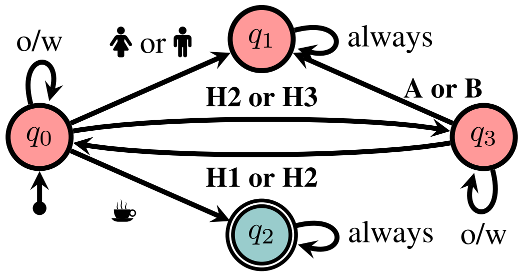


Figure 1: Left - Goal recognition environment where the possible goals of the agent are going to an office (A or B), leaving the building (E), going to the restroom (🚺 or 🚹), or getting coffee (☕). Right - a DFA classifier, learned from traces, that detects whether or not the agent is trying to reach the goal ☕. A decision is provided after each new observation based on the current state: yes for the blue accepting state, and no for the red, non-accepting states. "o/w" (otherwise) stands for all symbols that do not appear on outgoing edges from a state. "always" stands for all symbols. The DFA is guaranteed to correctly classify traces from an agent starting in A, B, or E that pursues an optimal path using only the hallways, measured by Manhattan distance. It also learns to generalize to some traces not seen in training. E.g., the trace (B, H3, H2, H1, ☕) is accepted and (B, H2, H1, 🚹) is rejected.

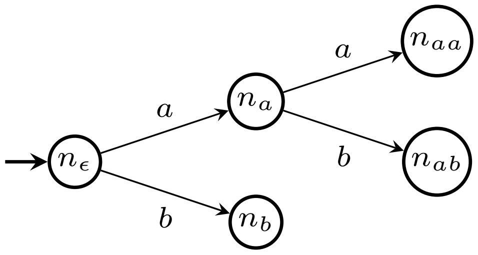


Figure 2: A PT for $\{b, aa, ab\}$.

nomial time) a PT that compactly represents all the prefixes from the training traces. In particular, the PT will be $\mathcal{P} = \langle N, n_\epsilon, \Sigma, \delta, \emptyset\rangle$ such that Σ is the set of symbols in the training set and, for every training trace $\tau_i = (\sigma_1, \ldots, \sigma_n)$, 1) there is a node $n_{(\sigma_1,\ldots,\sigma_j)} \in N$ for all $j \in \{0 \ldots n\}$ (i.e., all prefixes of τ_i) and 2) the transition function is constrained to have $\delta(n_{(\sigma_1,\ldots,\sigma_j)}, \sigma_{j+1}) = n_{(\sigma_1,\ldots,\sigma_{j+1})}$ for all $j \in \{2, n-1\}$ and $\delta(n_\epsilon, \sigma_1) = n_{\sigma_1}$. Intuitively, this PT defines a tree where all training traces are branches. As an example, Figure 2 shows the PT for a set of training traces $\{b, aa, ab\}$ (from Giantamidis and Tripakis (2016)).

Each node in the PT represents a prefix that appears in one or more training traces. After constructing the PT, we label its nodes with the number of positive n^+ and negative n^- training traces that start with the node's prefix. Positive traces are those belonging to the target class c for learning DFA $\mathcal{M}_c$ and all other traces are negative traces. In the example, if traces b and aa are positive and ab is negative, then $n_\epsilon^+ = 2$, $n_\epsilon^- = 1$, $n_b^+ = 1$, $n_b^- = 0$, $n_a^+ = 1$, $n_a^- = 1$, $n_{aa}^+ = 1$, $n_{aa}^- = 0$, $n_{ab}^+ = 0$, and $n_{ab}^- = 1$. In practice, we divide the contribution of each τ_i to n^+ or n^- by its length so that longer traces are not overrepresented. We then use these values to compute the training error in our MILP model.

Our MILP model takes the PT $\mathcal{P} = \langle N, n_\epsilon, \Sigma, \delta, \emptyset\rangle$, its nodes counters ($n^+$ and n^-), and a positive number $q_{\max}$ to learn a DFA $\mathcal{M}_c = \langle Q, q_0, \Sigma, F\rangle$ for class c with at most $q_{\max}$ states. The main idea is to assign the DFA state reached by each node in the tree (which represents a sequence of observations). A binary decision variable x_{nq}, which is equal to 1 if DFA state q is assigned to node n (and zero otherwise) encodes this assignment based on the transition function δ. Our model searches for an assignment of DFA states to the tree nodes that is feasible (there exists a transition function δ that generates such an assignment) and has low early prediction error. We predefine a set of accepting states F and add the error $x_{nq}n^-$ if $q \in F$ and by $x_{nq}n^+$ if $q \notin F$, for all $n \in N$.

To reduce overfitting, we limit the maximum number of DFA states, a common form of regularization in automata learning (Gold 1967, 1978; Giantamidis and Tripakis 2016). Additionally, we designate two DFA states—one accepting, and one non-accepting—as *absorbing states* which can only self-transition. These states prevent the classifier from changing decisions once reached. We found that rewarding nodes for reaching these absorbing states and penalizing the number of transitions between different DFA states acted as effective regularizers, significantly improving generalization. Further details about the MILP model can be found in the Technical Appendix § A.

In principle, the binary DFA classifiers learned by the MILP model can be used directly. However, they would have deterministic outcomes (i.e., the trace is either accepted or rejected), might contradict each other (multiple DFAs could accept the same trace), or might all reject the trace. In the next section, we address these issues by computing a probability distribution from the DFAs' predictions.

3.2 Posterior Inference of the Class Label

Given a trace (or partial trace) τ and the decisions of the one-vs-rest classifiers $\{\mathrm{D}_c(\tau) : c \in \mathcal{C}\}$, we use an approximate Bayesian method to infer a posterior probability distribution over the true label c^*. Each $D_c(\tau)$ is treated as a discrete random variable with possible outcomes $\{\text{accept}, \text{reject}\}$.

We make the following assumptions: (1) the classification decisions D_c for $c \in \mathcal{C}$ are conditionally independent given the true label c^* and (2) $p(D_c|c^*)$ only depends on whether $c = c^*$.

For each c', we compute the posterior probability of $c^* = c'$ to be

$$p(c^* = c'|\{\mathrm{D}_c : c \in \mathcal{C}\}) \propto \frac{p(c^* = c') * p(D_{c'}|c^* = c')}{p(D_{c'}|c^* \neq c')}$$

The full derivation can be found in the Technical Appendix § A.3. The probabilities on the right-hand side are estimated using a held-out validation set. We normalize the posterior probabilities $p(c^* = c'|\{\mathrm{D}_c : c \in \mathcal{C}\})$ such that their sum over $c' \in \mathcal{C}$ is 1 to obtain a valid probability distribution.

While we present a simple Bayesian inference approach suitable for low-data settings, more sophisticated ensemble methods may be leveraged to relax assumption (1). Additionally, conditioning on the DFA state rather than only the classifier decision can improve early classification performance. We leave this investigation to future work.

3.3 Discussion

In this section, we described an approach to learning DFAs for sequence classification based on mixed integer linear programming. Our model includes a set of constraints to enforce DFA structure, but in general, constraints can also be added to incorporate domain knowledge. Furthermore, our formulation is compatible with off-the-shelf optimizers and as such can benefit from advances in discrete optimization solvers. While our approach does not scale as well as gradient-based optimization, our use of prefix trees significantly reduces the size of the discrete optimization problem, allowing us to tackle real-world datasets with nearly 100,000 observation tokens, as demonstrated in our experiments.

We further show how the decisions of the DFA classifiers can be used to predict a probability distribution over labels. In our experiments we demonstrate how these probabilities are useful in an early classification setting. Furthermore, we can flexibly handle many important settings in sequence classification, including returning the k most probable class labels and multi-label sequence classification (see Technical Appendix § C.5).

4 Classifier Interpretability

An important property of our learned classifiers is that they are interpretable insofar as that the rationale leading to a classification is captured explicitly in the structure of the DFA. DFAs can be queried and manipulated to provide a set of interpretability services including explanation, verification of classifier properties, and (human) modification, as we demonstrate below. Our purpose here is to highlight the breadth of interpretability services afforded by DFA classifiers via their relationship to formal language theory. The effectiveness of a particular interpretability service is user-, domain-, and even task-specific and is best evaluated in the context of individual domains. We leave detailed study of this question to a separate paper.

As noted in Section 2, DFAs provide a compact, graphical representation of a (potentially infinite) set of traces the DFA positively classifies. Collectively, each DFA defines a regular language, the simplest form of language in the Chomsky hierarchy (Chomsky 1956). While many people will find the DFA structure highly interpretable, the DFA classifier can be transformed into a variety of different language-preserving representations including regular expressions, context-free grammars (CFGs), and variants of Linear Temporal Logic (LTL) (Pnueli 1977) (see also Technical Appendix § B.3). These transformations are automatic and can be decorated with natural language to further enhance human interpretation.

Example 4.1 *The following regular expression compactly describes the set of traces that are classified as belonging to the DFA classifier depicted in Figure 1:* $[(\Sigma - \{🚺, 🚹, H2, H3\})^*(H2 \cup H3)(\Sigma - \{A, B, H1, H2\})^*(H1 \cup H2)]^*(\Sigma - \{🚺, 🚹, H2, H3\})^* ☕ \Sigma^*$

Of course, this regular expression is only decipherable to a subset of computer scientists. We include it in order to illustrate/demonstrate the multiple avenues for interpretation afforded by our DFA classifiers. In particular, the regular expression can be further transformed into a more human-readable form as illustrated in Example 4.2 or transformed into a CFG that is augmented with natural language in order to provide an enumeration, or if abstracted, a compact description of the traces accepted by the DFA classifier.

Example 4.2 *The regular expression can be transformed into a more readable form such as:*
"Without first doing 🚺 or 🚹, repeat the following zero or more times: eventually do H2 or H3, then without doing A or B, eventually do H1 or H2, followed optionally by other events, excluding 🚺 and 🚹. Finally do ☕, followed by anything."

For others, it may be informative to extract path properties of a DFA as LTL formulae, perhaps over a subset of Σ or with preference for particular syntactic structures (e.g., (Camacho and McIlraith 2019)).

Example 4.3 *DFA classifier* $\mathcal{M} \models \Box\Diamond ☕$*, the LTL property "always eventually do ☕ ".*

These transformations and entailments utilize well studied techniques from formal language theory (e.g., Rozenberg and Salomaa 2012). Which delivery form is most effective is generally user- and/or task-specific and should be evaluated in situ via a usability study.

4.1 Explanation

An important service in support of interpretability is explanation. In the context of classification, given classifier $\mathcal{M}$ and trace τ, we wish to query $\mathcal{M}$, seeking explanation for the classification of τ.

In many real-world applications, traces comprise extraneous symbols that are of no interest and play no role in the classifier (such as the agent scratching their nose en route to coffee). It often makes sense to define an *explanation vocabulary*, $\Sigma_e \subseteq \Sigma$, a set of distinguished symbols of interest for explaining classifications that are pertinent to the explanation of traces such as τ, i.e. $\Sigma_e \cap \Sigma_\tau \neq \{\}$. Explanations for a positive classification can be extracted from a DFA classifier over an explanation vocabulary following the techniques described above.

In cases where a classifier does not return a positive classification for a trace, a useful explanation can take the form of a so-called *counterfactual explanation* (e.g., Miller 2019).

Let α and β be strings over Σ. The *edit distance* between α and β, $d(\alpha, \beta)$, is equal to the minimum number of edit operations required to transform α to β. We take the edit distance between two strings to be their Levenshtein distance where the set of edit operations comprises *insertion*, *deletion*, and *substitution*, and where each of these operations has unit cost (Levenshtein 1966).

Definition 4.1 (Counterfactual Explanation) *Let $\mathcal{M}$ be a DFA classifier that accepts the regular language $\mathcal{L}$ defined over Σ and let τ be some string over Σ. A counterfactual explanation for τ is the sequence of edit operations transforming τ to a string $\tau' = argmin_{\omega \in \mathcal{L}}(d(\tau, \omega))$.*

Wagner (1974) proposed an algorithm that computes, for some string τ and regular language $\mathcal{L}$, a string $\tau' \in \mathcal{L}$ with minimal edit distance from τ. The algorithm has a time complexity that is quadratic in the number of states of the DFA that accepts the language $\mathcal{L}$ in question.

Example 4.4 *Given the DFA depicted in Figure 1 and a trace τ = (**A, H2, H1,** 🚹), a possible counterfactual explanation is the edit operation (informally specified)* REPLACE 🚹 WITH ☕ *which transforms (**A, H2, H1,** 🚹) to (**A, H2, H1,** ☕). This explanation can then be transformed into a natural language sentence: "The binary classifier would have accepted the trace had ☕ been observed instead of 🚹". A simple approach that generates such natural language sentences from counterfactual explanations can be found in the Technical Appendix § B.1.*

4.2 Classifier Verification and Modification

Explanation encourages human trust in a classification system, but it can also expose rationale that prompts a human (or automated system) to further question or to wish to modify the classifier. Temporal properties of the DFA classifier $\mathcal{M}$, such as *"Neither 🚺 nor 🚹 occur before ☕"* can be straightforwardly specified in LTL and verified against $\mathcal{M}$ using standard formal methods verification techniques (e.g., Vardi and Wolper 1986). In the case where the property is false, a witness can be be returned.

Our learned classifiers are also amenable to the inclusion of additional classification criteria, and the modification to the DFA classifier can be realized via a standard product computation.

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be regular languages over Σ. Their intersection is defined as $\mathcal{L}_1 \cap \mathcal{L}_2 = \{x \mid x \in \mathcal{L}_1 \text{ and } x \in \mathcal{L}_2\}$. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be the DFAs that accept $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. The *product* of $\mathcal{M}_1$ and $\mathcal{M}_2$ is $\mathcal{M}_1 \times \mathcal{M}_2$ where the language accepted by the DFA $\mathcal{M}_1 \times \mathcal{M}_2$ is equal to $\mathcal{L}_1 \cap \mathcal{L}_2$ (i.e., $\mathcal{L}(\mathcal{M}_1 \times \mathcal{M}_2) = \mathcal{L}_1 \cap \mathcal{L}_2$).

Definition 4.2 (Classifier Modification) *Given a DFA encoding some classification criterion $\mathcal{M}_c$ and a DFA classifier $\mathcal{M}$, the modified classifier $\mathcal{M}'$ is the product of $\mathcal{M}$ and $\mathcal{M}_c$.*

Classifier modification ensures the enforcement of criterion $\mathcal{M}_c$ in $\mathcal{M}'$. However, such post-training modification could result in rejection of traces in the dataset that are labelled as positive examples of the class. Such modification can (and should) be verified against the dataset. Finally, modification criteria can be expressed directly in a DFA, or specified in a more natural form such as LTL.

5 Experimental Evaluation

In this section we describe the results of an evaluation of our approach, Discrete Optimization for Interpretable Sequence Classification (DISC), on a suite of goal recognition and behaviour classification domains. DISC is the implementation of the MILP model and Bayesian inference method described in Section 3. We compare against LSTM (Hochreiter and Schmidhuber 1997), a state-of-the-art neural network architecture for sequence classification; Hidden Markov Model (HMM), a probabilistic generative model which has been extensively applied to sequence tasks (Kupiec 1992; Sonnhammer et al. 1998); n-gram (Dunning 1994) for $n = 1, 2$, which perform inference under the simplifying assumption that each observation only depends on the last $n - 1$ observations; and a DFA-learning approach (DFA-FT) that maximizes training accuracy (minimizing the number of DFA states only as a secondary objective), representative of existing work in learning DFAs.

DISC, DFA-FT, and HMM learn a separate model for each label while LSTM and n-gram directly model a probability distribution over labels. Each classifier predicts the label with highest probability and all datasets consist of at least 7 labels. The LSTM optimized average accuracy over all prefixes of an observation trace in order to encourage early prediction.

Table 1 contains a summary of results for all datasets. Additional results (including examples of DFA classifiers learned from the data) and details of the experiments can be found in the Technical Appendix. The code for DISC is available online[1].

5.1 Experimental Setup

Performance is measured as follows. Cumulative convergence accuracy (CCA) at time t is defined as the percentage of traces τ that are correctly classified given $\min(t, |\tau|)$ observations. Percentage convergence accuracy (PCA) at $X\%$ is defined as the percentage of traces τ that are correctly classified given the first $X\%$ of observations from τ. All results are averaged over 30 runs, unless otherwise specified.

The datasets we used for evaluation were selected to be representative of the diversity of this class, both with respect to data properties such as noise, complexity of classification patterns, and to be somewhat suggestive of the diversity of the tasks for which this work is applicable. We con-

[1]https://github.com/andrewli77/DISC

sidered three goal recognition domains: Crystal Island (Ha et al. 2011; Min et al. 2016), a narrative-based game where players solve a science mystery; ALFRED (Shridhar et al. 2020), a virtual-home environment where an agent can interact with various household items and perform a myriad of tasks; and MIT Activity Recognition (MIT-AR) (Tapia, Intille, and Larson 2004), comprised of noisy, real-world sensor data with labelled activities in a home setting. Given a trace the classifier attempts to predict the goal the agent is pursuing. Crystal Island and MIT-AR are particularly challenging as subjects may pursue goals non-deterministically.

Experiments for behaviour classification were conducted on a dataset comprising replays of different types of scripted agents in the real-time strategy game StarCraft (Kantharaju, Ontañón, and Geib 2019), and on two real-world malware datasets comprising 'actions' taken by different malware applications in response to various Android system events (BootCompleted and BatteryLow) (Bernardi et al. 2019). The behaviour classification task involves predicting the type of StarCraft agent and malware family, respectively, that generated a given behaviour trace.

5.2 Results

Detailed results for StarCraft, MIT-AR, Crystal Island, and BatteryLow are shown in Figure 3 while a summary of results from all domains is provided in Table 1.

DISC generally outperformed n-gram, HMM, and DFA-FT, achieving near-LSTM performance on most domains. LSTM displayed an advantage over DISC on datasets with long traces. n-gram models excelled in some low-data settings (see MIT-AR) but perform poorly overall as they fail to model long-term dependencies. Surprisingly, DFA-FT was able to outperform all other methods on the malware datasets, but tested poorly on other noisy datasets (e.g. MIT-AR, Crystal Island) due to overfitting.

In realistic early classification scenarios, a single, irrevocable classification must be made, therefore it is common to report a prediction accompanied by the classifier's confidence at various times (Xing et al. 2008). If the confidence values closely approximate the true predictive accuracy, this allows an agent to appropriately trade-off between earliness and accuracy. We conducted an experiment where a classifier receives higher utility for making a correct prediction with fewer observations and uses its confidences to choose the time of prediction. We note that LSTM is the state of the art and is regularly used when earliness is a factor (e.g., Ma, Sigal, and Sclaroff 2016; Liu et al. 2016). The results (presented in the Technical Appendix § C.4) show that DISC has strong performance on each domain, only comparable by LSTM. This demonstrates that DISC produces robust confidences in its predictions.

5.3 Discussion and Limitations

We experimentally demonstrated a number of merits of our model: we achieve near-LSTM performance in most goal recognition and behaviour classification domains, as well as in an early classification task. We note that we claim no advantage over LSTMs in sequence classification and early prediction and it is not the objective of this work to demonstrate superior classifier accuracy to the LSTM baseline.

One feature of our learned classifiers is that they can encode simple long-term dependencies, while n-gram classifiers cannot; in the simplest case, a unigram model does not consider the order of observations at all. DISC makes similar Markov assumptions to HMM – that the information from any prefix of a trace can be captured by a single state – however, DISC only considers discrete state transitions, does not model an observation emission distribution, and regularizes the size of the model. We believe these were important factors in handling noise in the data.

A common approach to DFA-learning is to maximize training accuracy with the minimum number of DFA states. DFA-FT, which is based on this approach, excelled on the malware domains (suggesting that the DFA structure is a strong inductive bias for some real-world tasks), however, it performed poorly on many noisy datasets. The novel regularization techniques based on (non-self-loop) transitions and absorbing states introduced by DISC were crucial to learning robust DFAs which generalized to unseen data. Qualitatively, the DFAs learned by DISC were orders of magnitude smaller than those learned by DFA-FT (see Technical Appendix § C).

Finally, DISC assumes the traces for each label can be recognized by a DFA (or equivalently, form a regular language), which does not always hold true. In particular, DISC has limited model capacity, struggling on tasks that require large or unbounded memories, or involve counting occurrences. While this can be ameliorated by increasing the number of DFA states, such models may be less interpretable and require more computation to train. DISC also requires an appropriately chosen penalty on state transitions that depends on the amount of noise in the data, however, the reward for absorbing states did not require tuning in our experiments. A direction for future work is to extend DISC to handle real-valued or multi-dimensional data.

6 Related Work

We build on the large body of work concerned with learning automata from sets of traces (e.g., Gold 1967; Angluin 1987; Oncina and Garcia 1992; Heule and Verwer 2010; Ulyantsev, Zakirzyanov, and Shalyto 2015; Angluin, Eisenstat, and Fisman 2015; Giantamidis and Tripakis 2016; Smetsers, Fiterău-Broştean, and Vaandrager 2018). Previous approaches to learning such automata have typically constructed the prefix tree from a set of traces and employed heuristic methods or SAT solvers to minimize the resulting automaton. Here we follow a similar approach, but instead specify and realize a MILP model that is guaranteed to find optimal solutions given enough time; optimizes for a different objective function than those commonly used by previous work (see Section 3); does not assume noise-free traces or prior knowledge of the problem (e.g., a set of DFA templates); and introduces new forms of regularization.

Some work in automata learning has also shown (some) robustness to noisy data. For instance, Xue et al. (2015) combine domain-specific knowledge with domain-independent automata learning techniques and learn min-

			Percent Accuracy given full observation traces					
Dataset	N	$\|\tau\|$	**DISC**	DFA-FT	LSTM	HMM	1-gram	2-gram
Crystal Island	893	52.9	78 (±1.2)	46 (±1.0)	**87** (±1.1)	57 (±1.2)	69 (±0.9)	57 (±1.1)
StarCraft	3872	14.8	43 (±0.4)	38 (±0.4)	**44** (±0.4)	38 (±0.6)	29 (±0.4)	37 (±0.4)
ALFRED	2520	7.5	**99** (±0.1)	94 (±0.3)	**99** (±0.1)	97 (±0.7)	83 (±0.3)	94 (±0.2)
MIT-AR	283	9.3	57 (±1.9)	36 (±1.9)	56 (±1.9)	45 (±2.1)	**66** (±2.0)	55 (±1.5)
BootCompleted	477	206.0	59 (±2.2)	**69** (±1.5)	65 (±1.3)	54 (±2.4)	46 (±2.8)	55 (±1.6)
BatteryLow	283	216.2	60 (±1.4)	**73** (±1.4)	70 (±1.4)	52 (±2.2)	35 (±1.3)	54 (±1.5)

Table 1: A summary of results from all domains (DISC is our approach). With respect to the full dataset, N is the total number of traces, and $|\tau|$ is the average length of a trace. Reported are the percentages of traces correctly classified given the full observation trace, with 90% confidence error in parentheses. Highest accuracy is bolded.

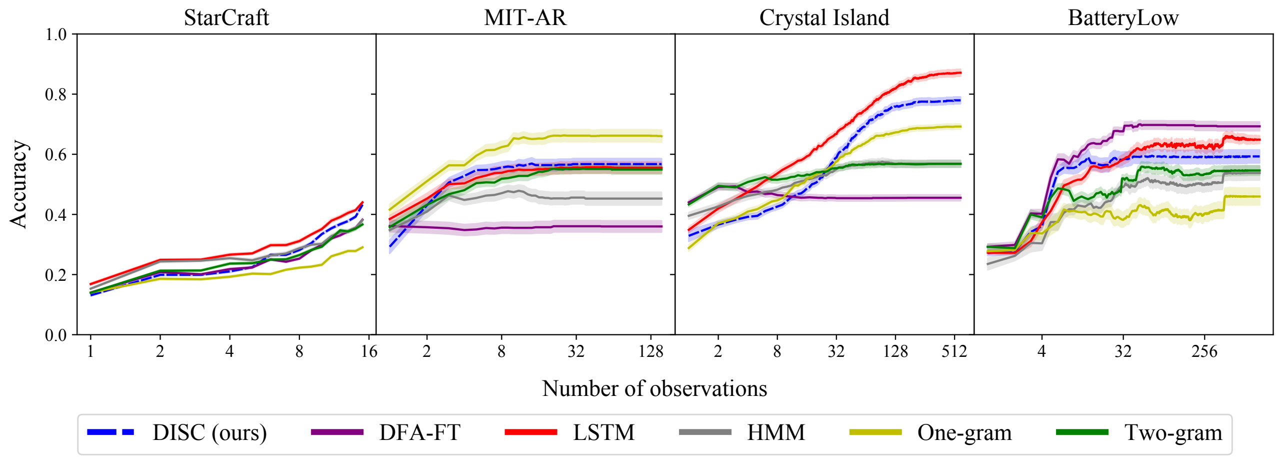


Figure 3: Test accuracy of DISC and all baselines as a function of earliness (number of observations seen so far) on one synthetic dataset (left) and three real-world datasets (three right). We report Cumulative Convergence Accuracy up to the maximum length of a trace. Error bars correspond to a 90% confidence interval. Further results appear in the Technical Appendix § C.

imal DFAs that capture malware behaviour, with empirical results suggesting a degree of robustness to noisy data. While we eschew domain knowledge in this work, our approach allows for domain knowledge to be incorporated during the learning process. Ulyantsev, Zakirzyanov, and Shalyto (2015) also work with noisy data, but their SAT-based model assumes that at-most k training instances have wrong labels, which is not a natural hyperparameter in machine learning, and does not support regularization.

Our work directly learns DFAs from data. However, it is also possible to follow a two-step approach. The idea is to first train a recurrent neural network (RNN) to solve the classification problem and then to learn a DFA that attempts to mimic the behaviour of the RNN (e.g., Jacobsson 2005; Wang et al. 2018; Weiss, Goldberg, and Yahav 2018; Koul, Fern, and Greydanus 2019). To learn the DFA, the main approaches consist of clustering the set of potential hidden states of the RNN into a finite set of DFA states (e.g., Jacobsson 2005; Wang et al. 2018) or to use the L^* algorithm (Angluin 1987) by utilizing the RNN as a *minimally adequate teacher* (Weiss, Goldberg, and Yahav 2018).

Our work shares some of its motivation with previous work that has proposed to learn interpretable classifiers which favour early prediction (e.g., Xing et al. 2011; Ghalwash, Radosavljevic, and Obradovic 2013; Wang et al. 2016; Hsu, Liu, and Tseng 2019). Some work in this space appealed to discrete optimization, as we do. For instance, Ghalwash, Radosavljevic, and Obradovic (2013) leverage discrete optimization to extract salient features from time series and use those for early classification. Chang, Bertsimas, and Rudin (2012) propose to learn binary classifiers that take the form of interpretable association rules; however, their approach does not consider sequential data. In our work, we specify a discrete optimization problem that yields highly structured DFA classifiers that support explanation and modification.

Relatedly, Bernardi et al. (2019) leverage process mining techniques and perform malware detection by learning declarative models representing different families of malware. However, they only consider the binary classification task and do not consider early prediction. Additionally, there exists a body of work that learns LTL formulae capable of discriminating between sets of traces (e.g., Neider and Gavran 2018; Camacho and McIlraith 2019; Kim et al. 2019). These formulae can in turn be used to perform classification tasks (Camacho and McIlraith 2019). However, these works learn formulae from full traces and do not consider early prediction.

While our work was originally motivated by the goal recognition task, we have developed a general learning approach for sequence classification. Previous work in goal and plan recognition has typically relied on rich domain knowledge (e.g., Kautz and Allen 1986; Ramírez and Geffner 2011), thus limiting the applicability of this body of work. To leverage the existence of large datasets and machine learning techniques, some approaches to goal recognition eschew assumptions about domain knowledge and instead propose to learn models from data and use these models to predict an agent's goal given a sequence of observations (e.g., Geib and Kantharaju 2018; Amado et al. 2018; Polyvyanyy et al. 2020). Such approaches either learn models of the dynamics that govern the environment which are then used in goal recognition, or directly learn classifiers that are given a sequence of observations and predict the goal. Our work partially shares its motivation with this body of work and proposes to directly learn classifiers from a given set of traces. Our learned classifiers offer a set of interpretability services, are optimized for early prediction, and demonstrate a capacity to generalize in noisy sequence classification settings.

There is also a body of work which applied automated planning tools to the malware detection task (e.g., Geib and Goldman 2001; Sohrabi, Udrea, and Riabov 2013; Riabov et al. 2015). In particular, Sohrabi, Udrea, and Riabov (2013) emphasize the importance of the robustness of a malware detection system to unreliable observations derived from network traffic, and demonstrate the robustness of their system to such observations. Riabov et al. (2015) show how robustness to noisy data can be enhanced by leveraging expert knowledge. Our learned classifiers demonstrate robustness to noisy sequence data in malware datasets and can be modified by experts to incorporate domain knowledge. Riabov et al. (2015) develop techniques which allow domain experts with no technical expertise in planning to construct models which reflect their knowledge of the domain. Such techniques could inspire methods by which domain experts can intuitively modify our learned DFA classifiers.

Finally, it is worth mentioning that DFA learning has recently been used to learn high-level models of memories for RL agents (e.g., Toro Icarte et al. 2019a,b; Xu et al. 2020a,b; Furelos-Blanco et al. 2020a,b; Rens and Raskin 2020). As such, our method might also provide interesting avenues for future work in reinforcement learning.

7 Concluding Remarks

The classification of (noisy) symbolic time-series data represents a significant class of real-world problems that includes malware detection, transaction auditing, fraud detection, and a diversity of goal and behavior recognition tasks. The ability to interpret and troubleshoot these models is critical in most real-world settings. In this paper we proposed a method to address this class of problems by combining the learning of DFA sequence classifiers via MILP with Bayesian inference. Our approach introduced novel automata-learning techniques crucial to addressing regularization, efficiency, and early classification. Critically, the resulting DFA classifiers offer a set of interpretability services that include explanation, counterfactual reasoning, verification of properties, and human modification. Our implemented system, DISC, achieves similar performance to LSTMs and superior performance to HMMs and n-grams on a set of synthetic and real-world datasets, with the important advantage of being interpretable.

Acknowledgements

We thank our anonymous reviewers for their constructive feedback, helpful questions, and technical suggestions. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, and Microsoft Research. The third author also acknowledges funding from ANID (Becas Chile). Resources used in preparing this research were provided by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence (www.vectorinstitute.ai/partners). Finally, we thank the Schwartz Reisman Institute for Technology and Society for providing a rich multi-disciplinary research environment.

References

Amado, L.; Pereira, R. F.; Aires, J.; Magnaguagno, M.; Granada, R.; and Meneguzzi, F. 2018. Goal recognition in latent space. In *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75(2): 87–106.

Angluin, D.; Eisenstat, S.; and Fisman, D. 2015. Learning regular languages via alternating automata. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*.

Bernardi, M. L.; Cimitile, M.; Distante, D.; Martinelli, F.; and Mercaldo, F. 2019. Dynamic malware detection and phylogeny analysis using process mining. *International Journal of Information Security* 18(3): 257–284.

Camacho, A.; and McIlraith, S. A. 2019. Learning interpretable models expressed in linear temporal logic. In *Proceedings of the 29th International Conference on Automated Planning and Sched. (ICAPS)*, 621–630.

Chang, A.; Bertsimas, D.; and Rudin, C. 2012. An Integer Optimization Approach to Associative Classification. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012.*, 269–277.

Chomsky, N. 1956. Three models for the description of language. *IRE Transactions on information theory* 2(3): 113–124.

De la Higuera, C. 2010. *Grammatical inference: learning automata and grammars*. Cambridge University Press.

Deshpande, M.; and Karypis, G. 2002. Evaluation of techniques for classifying biological sequences. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 417–431. Springer.

Doshi-Velez, F.; and Kim, B. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* .

Dunning, T. 1994. *Statistical identification of language*. Computing Research Laboratory, New Mexico State University Las Cruces, NM, USA.

Furelos-Blanco, D.; Law, M.; Jonsson, A.; Broda, K.; and Russo, A. 2020a. Induction and Exploitation of Subgoal Automata for Reinforcement Learning. *arXiv preprint arXiv:2009.03855* .

Furelos-Blanco, D.; Law, M.; Russo, A.; Broda, K.; and Jonsson, A. 2020b. Induction of Subgoal Automata for Reinforcement Learning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, 3890–3897.

Geib, C. W.; and Goldman, R. P. 2001. Plan recognition in intrusion detection systems. In *Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01*, volume 1, 46–55. IEEE.

Geib, C. W.; and Kantharaju, P. 2018. Learning Combinatory Categorial Grammars for Plan Recognition. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 3007–3014.

Ghalwash, M. F.; Radosavljevic, V.; and Obradovic, Z. 2013. Extraction of interpretable multivariate patterns for early diagnostics. In *2013 IEEE 13th International Conference on Data Mining*, 201–210. IEEE.

Giantamidis, G.; and Tripakis, S. 2016. Learning Moore machines from input-output traces. In *International Symposium on Formal Methods*, 291–309. Springer.

Gold, E. M. 1967. Language identification in the limit. *Information and control* 10(5): 447–474.

Gold, E. M. 1978. Complexity of automaton identification from given data. *Information and control* 37(3): 302–320.

Griffin, M. P.; and Moorman, J. R. 2001. Toward the early diagnosis of neonatal sepsis and sepsis-like illness using novel heart rate analysis. *Pediatrics* 107(1): 97–104.

Ha, E. Y.; Rowe, J. P.; Mott, B. W.; and Lester, J. C. 2011. Goal recognition with Markov logic networks for player-adaptive games. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Heule, M. J.; and Verwer, S. 2010. Exact DFA identification using SAT solvers. In *International Colloquium on Grammatical Inference*, 66–79. Springer.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8): 1735–1780.

Hsu, E.-Y.; Liu, C.-L.; and Tseng, V. S. 2019. Multivariate Time Series Early Classification with Interpretability Using Deep Learning and Attention Mechanism. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 541–553. Springer.

Jacobsson, H. 2005. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation* 17(6): 1223–1263.

Jünger, M.; Liebling, T. M.; Naddef, D.; Nemhauser, G. L.; Pulleyblank, W. R.; Reinelt, G.; Rinaldi, G.; and Wolsey, L. A. 2009. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media.

Kantharaju, P.; Ontañón, S.; and Geib, C. W. 2019. Scaling up CCG-Based Plan Recognition via Monte-Carlo Tree Search. In *Proc. of the IEEE Conference on Games 2019*.

Karim, F.; Majumdar, S.; Darabi, H.; and Harford, S. 2019. Multivariate lstm-fcns for time series classification. *Neural Networks* 116: 237–245.

Kautz, H. A.; and Allen, J. F. 1986. Generalized Plan Recognition. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)*, 32–37.

Kim, J.; Muise, C.; Shah, A.; Agarwal, S.; and Shah, J. 2019. Bayesian inference of linear temporal logic specifications for contrastive explanations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 776.

Koul, A.; Fern, A.; and Greydanus, S. 2019. Learning Finite State Representations of Recurrent Policy Networks. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

Kupiec, J. 1992. Robust part-of-speech tagging using a hidden Markov model. *Computer speech & language* 6(3): 225–242.

Lane, T.; and Brodley, C. E. 1999. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security (TISSEC)* 2(3): 295–331.

Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, 707–710.

Liu, J.; Shahroudy, A.; Xu, D.; and Wang, G. 2016. Spatio-temporal LSTM with trust gates for 3d human action recognition. In *Proceedings of the 14th European Conference on Computer Vision (ECCV)*, 816–833. Springer.

Ma, S.; Sigal, L.; and Sclaroff, S. 2016. Learning activity progression in LSTMs for activity detection and early detection. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1942–1950.

Miller, T. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence* 267: 1–38.

Min, W.; Mott, B. W.; Rowe, J. P.; Liu, B.; and Lester, J. C. 2016. Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 2590–2596.

Neider, D.; and Gavran, I. 2018. Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design (FMCAD)*, 1–10. IEEE.

Oncina, J.; and Garcia, P. 1992. Identifying regular languages in polynomial time. In *Advances in structural and syntactic pattern recognition*, 99–108. World Scientific.

Pnueli, A. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 46–57. IEEE.

Polyvyanyy, A.; Su, Z.; Lipovetzky, N.; and Sardina, S. 2020. Goal Recognition Using Off-the-Shelf Process Mining Techniques. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1072–1080.

Ramírez, M.; and Geffner, H. 2011. Goal recognition over POMDPs: Inferring the intention of a POMDP agent. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*.

Rens, G.; and Raskin, J.-F. 2020. Learning Non-Markovian Reward Models in MDPs. *arXiv preprint arXiv:2001.09293* .

Riabov, A.; Sohrabi, S.; Sow, D.; Turaga, D.; Udrea, O.; and Vu, L. 2015. Planning-based reasoning for automated large-scale data analysis. In *Proceedings of the 25th International Conference on Automated Planning and Sched. (ICAPS)*, 282–290.

Rozenberg, G.; and Salomaa, A. 2012. *Handbook of Formal Languages*. Springer Science & Business Media.

Sebastiani, F. 2002. Machine learning in automated text categorization. *ACM computing surveys (CSUR)* 34(1): 1–47.

Shridhar, M.; Thomason, J.; Gordon, D.; Bisk, Y.; Han, W.; Mottaghi, R.; Zettlemoyer, L.; and Fox, D. 2020. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *Proceedings of the 2020 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Shvo, M.; Li, A. C.; Toro Icarte, R.; and McIlraith, S. A. 2020. Interpretable Sequence Classification via Discrete Optimization (Technical Appendix). Technical Report CSRG-637, Department of Computer Science, University of Toronto. URL http://ftp.cs.toronto.edu/csrg-technical-reports/637/shvo-litoroicart-mci-tr20.pdf.

Smetsers, R.; Fiterău-Broştean, P.; and Vaandrager, F. 2018. Model learning as a satisfiability modulo theories problem. In *International Conference on Language and Automata Theory and Applications*, 182–194. Springer.

Sohrabi, S.; Udrea, O.; and Riabov, A. V. 2013. Hypothesis exploration for malware detection using planning. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence (AAAI)*, 883–889.

Sonnhammer, E. L.; Von Heijne, G.; Krogh, A.; et al. 1998. A hidden Markov model for predicting transmembrane helices in protein sequences. In *Ismb*, volume 6, 175–182.

Tapia, E. M.; Intille, S. S.; and Larson, K. 2004. Activity recognition in the home using simple and ubiquitous sensors. In *International conference on pervasive computing*, 158–175. Springer.

Toro Icarte, R.; Waldie, E.; Klassen, T. Q.; Valenzano, R.; Castro, M. P.; and McIlraith, S. A. 2019a. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Proceedings of the 32nd Conference on Advances in Neural Information Processing Systems (NeurIPS)*, 15497–15508.

Toro Icarte, R.; Waldie, E.; Klassen, T. Q.; Valenzano, R.; Castro, M. P.; and McIlraith, S. A. 2019b. Searching for Markovian Subproblems to Address Partially Observable Reinforcement Learning. In *Proceedings of the 4th Multi-disciplinary Conference on Reinforcement Learning and Decision (RLDM)*, 22–26.

Ulyantsev, V.; Zakirzyanov, I.; and Shalyto, A. 2015. BFS-based symmetry breaking predicates for DFA identification. In *International Conference on Language and Automata Theory and Applications*, 611–622. Springer.

Vardi, M. Y.; and Wolper, P. 1986. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, 322–331. IEEE Computer Society.

Wagner, R. A. 1974. Order-n correction for regular languages. *Communications of the ACM* 17(5): 265–268.

Wang, Q.; Zhang, K.; Ororbia II, A. G.; Xing, X.; Liu, X.; and Giles, C. L. 2018. An empirical evaluation of rule extraction from recurrent neural networks. *Neural computation* 30(9): 2568–2591.

Wang, W.; Chen, C.; Wang, W.; Rai, P.; and Carin, L. 2016. Earliness-aware deep convolutional networks for early time series classification. *arXiv preprint arXiv:1611.04578* .

Weiss, G.; Goldberg, Y.; and Yahav, E. 2018. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 5247–5256.

Xing, Z.; Pei, J.; Dong, G.; and Yu, P. S. 2008. Mining sequence classifiers for early prediction. In *Proceedings of the 2008 SIAM international conference on data mining*, 644–655. SIAM.

Xing, Z.; Pei, J.; and Keogh, E. 2010. A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter* 12(1): 40–48.

Xing, Z.; Pei, J.; Yu, P. S.; and Wang, K. 2011. Extracting interpretable features for early classification on time series. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, 247–258. SIAM.

Xu, Z.; Gavran, I.; Ahmad, Y.; Majumdar, R.; Neider, D.; Topcu, U.; and Wu, B. 2020a. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the 30th International Conference on Automated Planning and Sched. (ICAPS)*, volume 30, 590–598.

Xu, Z.; Wu, B.; Neider, D.; and Topcu, U. 2020b. Active Finite Reward Automaton Inference and Reinforcement Learning Using Queries and Counterexamples. *arXiv preprint arXiv:2006.15714* .

Xue, Y.; Wang, J.; Liu, Y.; Xiao, H.; Sun, J.; and Chandramohan, M. 2015. Detection and classification of malicious JavaScript via attack behavior modelling. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 48–59.

Zhou, C.; Sun, C.; Liu, Z.; and Lau, F. 2015. A C-LSTM neural network for text classification. *arXiv preprint arXiv:1511.08630* .