Grammatical Inference Algorithms in MATLAB

Hasan Ibne Akram[1], Colin de la Higuera[2], Huang Xiao[1], and Claudia Eckert[1]

[1] Technische Universität München, Munich, Germany
{*hasan.akram, huang.xiao, claudia.eckert*}*@sec.in.tum.de*
[2] Nantes University, Nantes, France, *cdlh@univ-nantes.fr*

Abstract. Although MATLAB[1] has become one of the mainstream languages for the machine learning community, there is still skepticism among the Grammatical Inference (GI) community regarding the suitability of MATLAB for implementing and running GI algorithms. In this paper we will present implementation results of several GI algorithms, e.g., RPNI (Regular Positive and Negative Inference), EDSM (Evidence Driven State Merging), and k-testable machine. We show experimentally based on our MATLAB implementation that state merging algorithms can successfully be implemented and manipulated using MATLAB in the similar fashion as other machine learning tools. Moreover, we also show that MATLAB provides a range of toolboxes that can be leveraged to gain parallelism, speedup etc.

1 Introduction

In this paper we focus on two important tasks for GI - learning regular languages from an informant and learning *k-testable* languages [1] from text. We have implemented the RPNI [2] and EDSM [3] algorithms to investigate the feasibility of running such classes of algorithm in MATLAB. We have also implemented algorithms to learn *k-testable* languages which corresponds to a subset of regular languages.

We have followed the notations and algorithms given in Colin de la Higuera's [4] book. The details of RPNI algorithm can be found in chapter 12, EDSM in chapter 14 and *k-testable* language in chapter 11 of the book. In this section we briefly introduce notations and definitions.

1.1 Preliminaries

Definition 1 *A* ***Deterministic Finite Automaton*** *is defined as a 6-tuple* $\mathcal{A} = \langle \Sigma, Q, q_0, \mathbb{F}_{\mathbb{A}}, \mathbb{F}_{\mathbb{R}}, \delta \rangle$*, where* Σ *is the set of alphabets,* Q *is the set of finite states,* $q_0 \in Q$ *is the initial state,* $\mathbb{F}_{\mathbb{A}} \subseteq Q$ *is the set of final accepting states,* $\mathbb{F}_{\mathbb{R}} \subseteq Q$ *is set of final rejecting states,* δ *is the transition function.*

Definition 2 *A* ***Prefix Tree Acceptor (PTA)*** *is a tree-like DFA generated by extracting all the prefixes of the samples as states that only accepts the samples it is built from. A* ***prefix tree*** *is also known as* ***trie*** *which is an ordered data structure and it is expressed as a DFA to form a PTA. Let* S *be the sample from which we build a PTA.* $\mathcal{A}_{\mathcal{PTA}} = PTA(S)$ *is a DFA that contains a path from the initial state to a final accepting state for each strings in* S*.*

[1] MATLAB is a registered trademark of The MathWorks, Inc.

To recognize *k-testable* languages we would require a special machine called *k-testable machine* from which we can build an equivalent DFA.

Definition 3 *Given $k > 0$, a* ***k-testable machine*** $k - \mathcal{TSS}$ *is a 5-tuple* $Z_k = \langle \Sigma, I, F, T, C \rangle$ *where* Σ *is the set of alphabets,* $I \subseteq \Sigma^{k-1}$ *set of prefixes of length* $k-1$*,* $F \subseteq \Sigma^{k-1}$ *suffixes of length* $k-1$*,* $C \subseteq \Sigma^{k}$ *set of short strings, and* $T \subseteq \Sigma^{k}$ *set of allowed segments.*

A *k-testable machine* $k - \mathcal{TSS}$ will recognize strings only either exactly in C, or those whose prefix of length $k-1$ is in I, suffix of length $k-1$ is in F, and where all substrings of length k is in T.

2 State Merging Algorithms

The basic idea of state merging algorithm to infer a DFA is to build a PTA from the positive sample (S_+), conduct the state-merging iteratively, each intermediate DFA is verified by examining the negative samples. Only the merge resulting in a DFA that rejects all the negative samples is kept as current status, otherwise the merge is discarded. This process is repeated until the target automaton is found. Examples of such state merging algorithms are RPNI, EDSM, Blue-Fringe etc.

2.1 RPNI

The RPNI version given in [4] uses two labels for the states in the automaton: *red* states and *blue* states. After a series of merges between the *red* states and *blue* states, promoting the states (e.g., *blue* to *red*), the target DFA is produced. For the details of RPNI please consult chapter 12 of [4].

By consideration of overhead of computing, we initiated another version of RPNI, Parallel RPNI, which opens multiple sessions for states in PTA and runs those sessions concurrently so that if the merge at state i is failed, then the merges at states $\{i+1, i+2, i+3, \cdots\}$ might be already prepared to be taken into execution.

2.2 EDSM

RPNI basically performs a greedy search to find out the target DFA, meaning whenever two states are mergable, they are merged. Obviously there could be other option, e.g., choosing a *better* or even *best* merge by means of some heuristics. EDSM algorithm introduced by Lang and et. al., [3] which takes such heuristics into consideration.

3 MATLAB GI Toolbox

In this section we present the MATLAB GI Toolbox, an open source implementation of a set of GI algorithms in MATLAB. The Toolbox offers out-of-the-box implementations of a range of GI algorithms that can be used like every other machine learning tool provided in MATLAB.

The fundamental data structures in the MATLAB [5] platform are matrices. Therefore, in our implementation we have used matrices as primary data structure to represent a DFA. The DFA object contains eight different sets represented as matrices: ***FiniteSetOfStates:*** the set of finite states (Q) of the DFA, it is stored as an integer vector in MATLAB. ***Alphabets***: the set of symbols. It is stored as a cell array where each cell contains a character. ***Transition-Matrix***: each column of the matrix corresponds to a symbol $a \in \Sigma$ *(Alphabets)* and each row corresponds to a state $q \in Q$ *(FiniteSetOfStates)*. Each cell of the matrix is a transition δ, e.g., if there is a transition from a state q_i to q_j via a symbol a, then the corresponding cell for q_i and a is marked as q_j. No transition cells are marked with -1. ***InitialState:*** the set of initial states, an integer vector which contains only state. ***FinalAcceptedStates:*** the set of final accepted states, an integer vector which is a subset of *FiniteSetOfStates*. ***FinalRejectStates:*** the set of final rejecting states, an integer vector which is a subset of *FiniteSetOfStates*. ***RED:*** the set of *red* states, an integer vector which is a subset of *FiniteSetOfStates*. ***BLUE:*** the set of *blue* states, an integer vector which is a subset of *FiniteSetOfStates*.

The input file is given in the similar format as the Abbadingo[2] format. However, internally in MATLAB the training dataset is represented as cell arrays [5], each cell containing a character, which is independent of the input file format.

3.1 Features

The MATLAB GI Toolbox provides the GI algorithms in a modular fashion so that they can be reused to enhance or improve the existing GI algorithms. The DFA data structure and the built-in methods can be used for RPNI, Blue-Fringe, EDSM and can be extended to incorporate other methods such as Genetic Algorithm techniques to optimize the search strategy. Moreover, this toolbox has been made absolutely compatible to other MATLAB features and toolboxes which enables ways of trying out new experiments using other MATLAB Toolboxes in an extremely easy manner. It is simple to use the toolbox to do a *k-fold cross-validation*, ROC analysis etc. using internal MATLAB classes to obtain fast results.

4 Experimental Results

In this section we present our experiments which are conducted on the Gowachin[3] dataset varying the size of the target DFA and the sample size Table 1 for RPNI and EDSM. The results obtained from the MATLAB GI Toolbox are tested also with Gowachin. Table 2 shows experiments on learning k-testable machines. The experiments were run in a machine having two CPUs, each with 4 core Quad-Core AMD Opteron^{TM}Processor 2384, cache size: 512 KB, memory 66175292

[2] Abbadingo is a DFA learning competition held in 1998. The details about the competition and data format can be found at: http://www-bcl.cs.may.ie/

[3] Gowachin: DFA Learning Competition is a test version of a follow-on to Abbadingo One. Artificial datasets for training and testing can be generated using this website: http://www.irisa.fr/Gowachin/

KB. In these experiments, the accuracy results are as expected, bad when an insufficient amount of data is provided.

sample size →	*200*		*500*		*1000*		*5000*		
DFA size ↓	RPNI	EDSM	RPNI	EDSM	RPNI	EDSM	RPNI	EDSM	↓
3	99.17	99	100	99.28	99.33	100	99.17	100	***a***
	0.49	0.44	0.48	2.25	1.03	1221.6	43.12	23.09	***t***
5	100	77.94	92.28	100	99.78	99.67	100	99.72	***a***
	0.50	639.59	511.67	31.84	2.28	26.06	11.10	245.95	***t***
10	67.39	66.78	100	99.89	99.61	100	100	100	***a***
	16.09	736.31	2.9	257.79	4.57	464.95	31.47	1421.8	***t***
20	63.94	52.5	61.67	60.77	99.06	99.33	100	100	***a***
	13.10	2043.6	180.71	1873.8	21.46	3677.99	99.20	3989.71	***t***

Table 1. MATLAB GI ToolBox executions of RPNI and EDSM with different target DFA sizes and sample sizes. Row ***a*** shows the accuracy (%) and row ***t*** shows the time cost in seconds.

5 Conclusion & Future Work

The experimental results shown above clearly indicate that MATLAB is perfectly suitable for GI algorithms and experiments, at least for reasonable sizes of datasets. Besides the two state merging algorithms, we have also implemented learning algorithm for *k-testable machine*. Moreover, we have implemented a parallel version of RPNI using the MATLAB Parallel Toobox [5], where we have been able to gain 10-15% speedup for each additional CPU. Our future plan is to incorporate other GI algorithms such as L*, OSTIA (for learning *transducers*) etc. in the toolbox.

sample size →	*200*			*500*			*1000*			*5000*		
k ↓	***t***	***p***	***r***	***t***	***p***	***r***	***t***	***p***	***r***	***t***	***p***	***r***
2	1.16	0.12	0.14	2.98	1	0.58	6.03	1	0.408	31.77	0.5	1
3	1.16	1	0.14	2.84	1	0.579	5.83	1	0.407	31.30	1	1
5	1.17	1	0.04	2.86	1	0.142	5.89	1	0.023	30.78	1	0.543
10	4.58	1	0.002	9.19	NaN	0	16.08	1	0.007	65.06	1	0.002

Table 2. MATLAB GI ToolBox executions of learning ***k-testable machine***. Column ***t*** shows the time cost in seconds, ***p*** the precision and ***r*** the recall.

To the best of our knowledge this is the first open source implementation of GI algorithms in MATLAB. We plan to publish the MATLAB GI ToolBox as an open source library under the MIT License for open source software. The beta version of MATLAB GI ToolBox can be downloaded from the following link: `http://www.sec.in.tum.de/~hasan/matlab/gi_toolbox/`.

References

1. García, P., Vidal, E.: Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. IEEE Trans. Pattern Anal. Mach. Intell. **12**(9) (1990) 920–925
2. Oncina, J., Garcia, P.: Identifying regular languages in polynomial time. In: Advances in Structural and Syntactic Pattern Recognition, volume 5 of Series in Machine Perception and Artificial Intelligence, World Scientific (1992) 99–108
3. Lang, K.J., Pearlmutter, B.A., Price, R.A.: Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In: ICGI '98: Proceedings of the 4th International Colloquium on Grammatical Inference, London, UK, Springer-Verlag (1998) 1–12
4. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press (2010)
5. MathWorks: Matlab - the language of technical computing. (2010) http://www.mathworks.com/products/matlab.