ISBN 978-981-11-7861-0
Proceedings of 2018 the 8th International Workshop on Computer Science and Engineering (WCSE 2018)
Bangkok, 28-30 June, 2018, pp. 780-785
doi: 10.18178/wcse.2018.06.129

Probabilistic Regular Grammar Inference Algorithm Using Incremental Technique

Torsak Penpinun [1+] and Athasit Surarerks [2]

[1] pentorja@gmail.com, Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand

[2] athasit.s@chula.ac.th, Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand

Abstract. Grammatical inference has been studied for a long time where grammar is illustrated by a collection of re-writing rules, together with their probabilities. We are interested in regular language model which can be recognized by a finite state machine. The most popular technique is an Alergia algorithm. The objective is to construct a probabilistic finite state machine using only positive examples together with their probabilities (or frequency). In this work, we introduce a probabilistic grammatical inference algorithm in order to construct a prefix tree. The algorithm starts by considering the shortest positive example. Two types of regular grammar rules (productions) are introduced. Our experimental results show that the probabilities obtained from our probabilistic finite state machine can be more accurate than the one obtained from the previous algorithm. We hope that our algorithm will be an alternative way for constructing a probabilistic finite state machine.

Keywords: grammar inference, probabilistic finite state machine, incremental technique.

1. Introduction

Grammatical inference or grammatical induction has been studied for a long time. The theoretical foundations in grammatical inference were first introduced by M.E. Gold [1]. This topic can be considered a problem in natural language processing (NLP) and in human cognition. A grammar is described as a model of sentence-level phenomena in language. Many researches concentrated on how to learn a grammar based on Chomsky hierarchy [2] from some positive examples of words in specific language. The typically classes of grammars can be found [1, 3, 4, and 5]. Grammatical inference can also be applied in many applications such as automatic speech recognition, statistical machine translation and information retrieval, etc.

Considering the structural pattern recognition, and computational linguistic, the important problem in inference is that of dealing with positive and negative examples. In detail, grammar can be illustrated by a collection of re-writing rules, together with their probabilities. Such rules (or production rules) are used for producing words in some specific languages. The research topic is to determine production rules which can generate most of words in the target language.

Many research domains including pattern recognition, the process of learning is a set of production rules for strings in a formal language. The rules used to describe how to generate strings from alphabet that are valid according to the syntax. Normally use for describing a language model which can represent a set of sequences of processes. There are limited literatures recognized a language from examples, a probabilistic finite automaton is usually used for an inference model. Many researches interest in algorithm to inference model but Alergia algorithm [6], the famous one, is more efficient and accurate. After we studied Alergia algorithm, we have found that it is a complex grammatical inference and takes a lot of computing time.

+ Corresponding author.
E-mail address: pentorja@gmail.com.

In this work, we concentrated on how a language can be recognized using some positive examples. We scoped only on the regular languages. Our concept is to construct a grammatical inference algorithm with positive examples. The technique is to represent a language using grammar model together with its probability comparing with the Alergia algorithm. We then show in our work by some examples that the probability obtained from our algorithm are closed to the original comparing to the previous work.

In this paper, we provide some background in language and grammatical inference model in Section 2. Our algorithm is proposed in Section 3. The experimental results are illustrated by Section 4. We conclude our research in Section 5

2. Background Knowledge

This section recalls some formal definitions concerning model of formal languages and their grammar systems. Probabilistic finite state machines are also detailed since they are used for representing the result of the inference model. Finally, the *Hoeffding* bound technique is restated.

Formal Languages and Their Grammar Systems

Formal language is a mathematical model which is represented by a set of symbols or letters. Every element called words in a specific language is defined by certain rules or grammar productions. Each language can be used for representing a decision problem. Each word in a language corresponds to a yes-instance of the associated decision problem. In order to answer an instance of the problem, this is to determine that the associated word is in the language.

Definition 1: Alphabet, denoted by Σ, is a finite set of characters or symbols.

Definition 2: Language is a set of finite strings of characters in the alphabet. An element in a language is called a word. We define length of a word x is the number of characters occurred in x.

Example 1: Consider the following path problem. The problem is to determine the path of a robot starting from "in" and end up at the "out" with respect to the map in Fig.1. To define a language, let Alphabet Σ be a set {0, 1} where 0 and 1 state for "turn right" and "turn left" respectively. The language associated to this problem is the set of all yes-instances. The language is {11, 101110, 1011111110, …}. Given a part, in order to determine that the robot does end up at the "out", it is to check that the part is a word in the language.

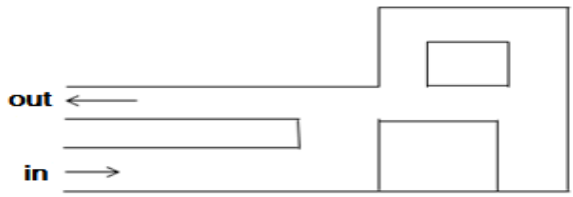

Fig. 1: Path problem as a language model.

Definition 3: Regular grammar G = <∑, V, P, S > is defined by

- ∑ A finite set of characters,
- V A finite set of variables,
- P A finite set of production of the form $X \rightarrow aY$, $X \rightarrow a$, $X \rightarrow \lambda$ where X and Y are variables,
- S A starting variable,

where λ is an empty string. Note that $X \rightarrow aY$ means that X can be substituted by aY. We usually note $X \Rightarrow ab$ for representing X can be substituted by ab using some sequence of substitutions.

A string x is said to be produced from a grammar G if x can be produced from a sequence of productions starting from a starting variable S. A language L is said to be a regular language if and only if every string in L can be produced from a regular grammar.

Example 2: Given a regular grammar G defined by $\{S \rightarrow aS, S \rightarrow bT, T \rightarrow aT, T \rightarrow \lambda\}$. Let L be a language that every string in the language can be produced from G. It can see that a string baa is in L since "baa" can be produced by starting from S,

$S \Rightarrow bT$ using the production $S \rightarrow bT$,

$S \Rightarrow baT$ using the production $T \rightarrow aT$,

$S \Rightarrow baaT$ using the production $T \rightarrow aT$,

$S \Rightarrow baa$ using the production $T \rightarrow \lambda$.

Probabilistic Finite State Machine (PFSM)

Finite state machine is a mathematical model for recognizing a regular language with respects to the input. We will recall now a probabilistic model for a finite state machine as follows:

Definition 4: A probabilistic finite state machine is composed of

- Q A finite set of states where q_0 is the initial state,
- Σ A finite alphabet is a set of all characters,
- T A transition function where $T:(Q \times \Sigma \cup \{\lambda\}) \rightarrow (Q \times p)$ where p is the probability of the transition and λ is an empty string.

The probability that a string $w = x_1x_2\ldots x_n$ (where x_i is in Σ) is in the language associated to the PFSM is the product of p's of every transition for w.

Example 3: Let PFSM be a probabilistic finite state machine as shown in Fig 2. The probability of an empty string (λ) is 1/2. The probability of "ab" can be computed by $1/4 \times 1/4 \times 2/3 = 1/24$.

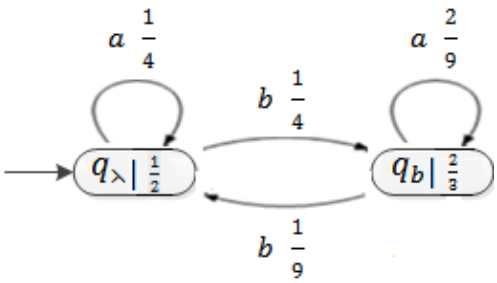

Fig. 2: An example of a probabilistic finite state machine with two states.

Alergia Algorithm and Hoeffding bound

The Alergia algorithm [6, 7], is a learning algorithm which works by merging the states of a generated finite state machine from a probabilistic perspective. The algorithm starts by generating a Prefix Tree Accepter (PTA) from the input (positive) examples and analyses the relative frequency of outgoing transactions at every node. The algorithm compares probabilities between two states using the Hoeffding bound concept [8].

Alergia Algorithm

Let ∑ be the finite set of alphabet {a, b}.

Let S be the samples set of strings

Step 1: Initialize Prefix Tree Acceptor

A is a stochastic prefix tree acceptor from S

Step 2: Compare two successor nodes (start from the root of A)

Using Hoeffding bound between successor (first node(A) to last node(A))

Step 3: Merging techinques

If Hoeffding satisfied,

Merge (firstnode(A), the last node(A))

Determinize(A)

Repeat Step 3 until no successor.

Step 4: The result

Return A

Example 4: *From PFSM defined in Example 7. Given some positive examples as illustrated in Table 1, the PFSM obtained from the Alergia algorithm contains of two states as shown in Fig. 3.*

Table 1: Input positive examples

Strings	Frequency	Probability	Strings	Frequency	Probability
λ	490	0.49	abaa	2	0.002
a	128	0.128	abab	2	0.002
b	170	0.17	abba	2	0.002
aa	31	0.031	abbb	1	0.001
ab	42	0.042	baaa	2	0.002
ba	38	0.038	baab	2	0.002
bb	14	0.014	baba	1	0.001
aaa	8	0.008	babb	1	0.001
aab	10	0.01	bbaa	1	0.001
aba	10	0.01	bbab	1	0.001
abb	4	0.004	bbba	1	0.001
baa	9	0.009	aaaaa	1	0.001
bab	4	0.004	aaaab	1	0.001
bba	3	0.003	aaaba	1	0.001
bbb	6	0.006	aabaa	1	0.001
aaaa	2	0.002	aabab	1	0.001
aaab	2	0.002	aabba	1	0.001
aaba	3	0.003	abbaa	1	0.001
aabb	2	0.002	abbab	1	0.001

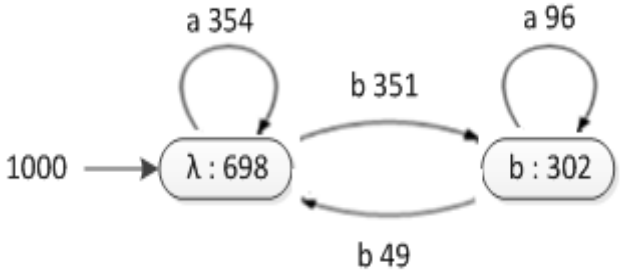

Fig. 3: Probabilistic finite state machine generated by Alergia algorithm.

3. Our Works

Alergia algorithm is always a most popular technique for learning PFSM, but the problem is the number of states of the PFSM are probably large. The study of Xuanyi Qi [9] aimed to analyse and improve the algorithm via minimization of deterministic finite automaton. This can reduce the number of states of the output PFSM. We remark that time complexity of the algorithm is still a problem. In this work, we propose an alternative way to construct a probabilistic grammar using incremental technique. The result of PFSM can be obtained from such productions.

Algorithm: Probabilistic grammar inference

Let ∑ be the finite set of alphabet.

Let V be the set of variables.

Input: A finite set of all positive examples in canonically ordering

Output: A finite set of all probabilistic productions

Step 1: Initialize grammar patterns

Two types of productions can be generated by the algorithm are $A \rightarrow \lambda$ and $A \rightarrow aB$

Where A and B are variables and *a* is a character in the alphabet ∑ and λ be an empty string.

Step 2: Initialize the probability of the λ-production

Compute the probability of λ-production (string with length = 0) from the examples.

Step 3: Determine the probability of string with length = n – 1 (start from n = 1)

Construct production for generating strings with length = n

Consistency checking

Adjust the probability of string with length = n – 1.

Step 4: Increase n by 1

Repeat Step 3 again until the end of examples.

Concept of the algorithm: the algorithm starts by computing the probability of all productions corresponding to generate all strings of length n (where n starts from 0 and increasing by 1). For the first step, only one variable S is introduced. For each step (n > 1), introduce a new variable if the probability of all productions does not satisfy the consistency of probability constraint, and also adjust the probability values. The PFSM can be constructed using the final productions obtained from the algorithm.

4. Experimental Results

In this section, we will give you an example. Using the set of all positive examples described in Example 8, we will show you how the productions can be obtained from our algorithm.

Step 1: Initialize grammar patterns

Generate two productions: S→λ and S→*a*S. (The variable S states for the starting variable.)

Step 2: Initialize the probability of the λ-production

Compute the probability of λ-production from the example, prob(S→λ) = 0.49

Step 3: Determine the probability of string with length = 0

Construct the production S→λ with prob(S→λ) = 0.49 = prob(λ).

Step 4: Increase n by 1

Consider strings *a* and *b*.

Step 3: Determine the probability of string with length = 1

Construct the production S → *a*S with prob(S → *a*S) = 0.2612

Since prob(*a*) = prob(S → *a*S) × prob(S→λ), then

prob(S → *a*S) = 0.128/0.49 = 0.2612

Construct the production S → *b*S with prob(S → *b*S) = 0.3469

Since prob(*b*) = prob(S → *b*S) × prob(S→λ),

prob(S → *b*S) = 0.17/0.49 = 0.3469

Using the consistency checking, since the prob(S→λ) + prob(S → *a*S) + prob(S → *b*S) ≠ 1,

then the production S → *b*S must be replaced by S → *b*T and T → λ, where prob(S → *b*T) = 0.2488

Since prob(b) = prob(S → *b*T) × prob(T → λ), then

prob(T → λ) = 0.17/0.2488 = 0.6832

Step 4: Increase n by 1

Consider strings *aa*, *ab*, *ba* and *bb*.

Repeat until we obtain the final solutions as follows:

prob(S→λ) = 0.49 prob(S → *a*S) = 0.2612 prob(S → *b*T) = 0.2488

prob(T→λ) = 0.6462 prob(T → *a*T) = 0.239 prob(T → *b*S) = 0.1148

The PFSM can be constructed as illustrated by Fig. 7.

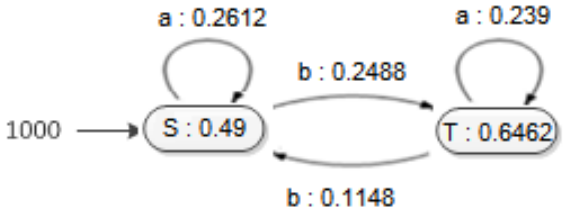

Fig. 7: Probabilistic finite state machine generated by our incremental algorithm.

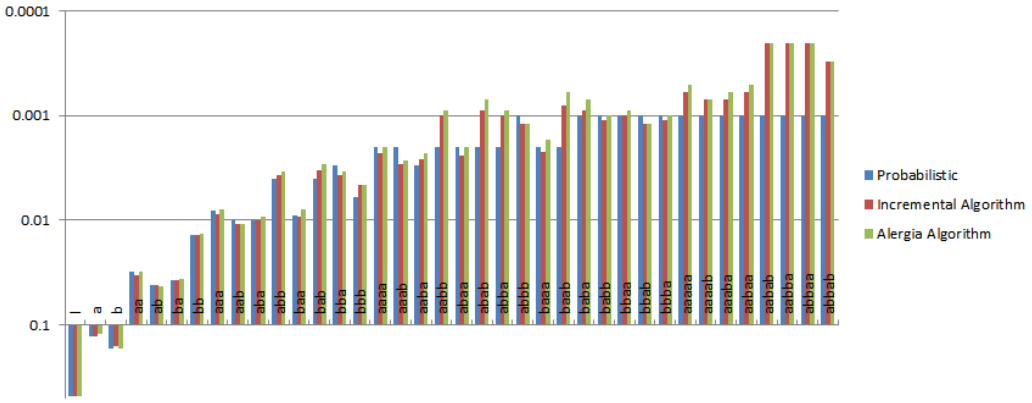

Fig. 8: A comparative bar graph of probabilities between Alergia and our algorithm.

From this comparison graph, we compare probabilities between our incremental algorithm and Alergia algorithm. It is obtained that 20 input examples that their probabilities obtained from our algorithm are closer to the original than those obtained from the Alergia algorithm, 10 input examples that the probabilities obtained from Alergia algorithm are closer than those obtained from our algorithm, and 8 input examples that both algorithms got the same probabilities.

5. Conclusion

We propose an alternative way to construct a probabilistic finite state machine for describing the set of some positive examples of the target language. Our concept is to construct the target probabilistic productions which can fit to the given positive examples. The PFSM can be finally obtained from the obtained productions. We suggest that our algorithm will be an alternative way for constructing a PFSM.

6. References

[1] M.E. Gold. Language identification in the limit. *Information and Control.* 1967, 10 (5): 447-474.

[2] N. Chomsky. *Syntactic Structures.* Berlin: Mouton de Gruyter 1957.

[3] M.V. Zaanen. *Bootstrapping syntax and recursion using alignment-based learning*. Proceedings of the 17th International Conference on Machine Learning, 2000: 1063-1070.

[4] D. Klein and C.D. Manning. Natural language grammar induction with a constituent-context model. *Pattern Recognition*. 2005, 38 (9): 1407-1419.

[5] D. Klein. *The unsupervised learning of natural language structure*, Ph.D. thesis, Standford University, 2005.

[6] R.C. Carrasco, J. Oncina. Learning stochastic regular grammars by means of a state merging method. In: R.C. Carrasco, J. Oncina (eds.). *Grammatical inference and applications*, Springer, Berlin, Heidelberg 1994: pp. 139-152.

[7] C. de la Higuera. Learning probabilistic finite automata. In: C. de la Higuera (eds.). *Grammatical Inference: learning automata and grammars*. Cambridge: The United Kingdom at the University Press, 2010: pp. 331-354.

[8] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal* 1963, 58: 13-30.

[9] X. Qi. *Analysis on Alergia Algorithm: pattern recognition by automata theory*. Master's Projects, 2016: 491.