Learning k-Piecewise Testable Languages from Positive Data *

José Ruiz and Pedro García

Depto. de Sistemas Informáticos y Computación.
Universidad Politécnica de Valencia. Valencia (Spain)
e-mail: jruiz@dsic.upv.es pgarcia@dsic.upv.es

Abstract. A k-piecewise testable language ($k\text{-PWT}$) is defined by the
subwords (sequences of symbols which are not necessarily consecutive)
no longer than $k$ that are contained in its words. We propose an algorithm
that identifies in the limit the class of $k\text{-PWT}$ languages from positive
data. The proposed algorithm has polynomial time complexity on the
length of the received data. As the class of $k\text{-PTW}$ languages is finite, the
algorithm can be used for PAC-learning.

1 Introduction.

The central problem of grammatical inference is that of obtaining a representa-
tion of a formal language from examples. It is well known [8] that the problem
of identifying a language from positive examples only has strong limitations.
In fact, the simplest family of languages in Chomsky’s hierarchy, the family of
regular languages, is not identifiable from positive data in the limit. Inference
from positive data seemed to be condemned to obtaining generalizations of the
training sample by means of heuristics, without having the possibility of char-
acterizing the obtained results [4], [9], [6]. However, there are nontrivial families
of languages identifiable in the limit from positive presentation, and a character-
ization of those families has been proposed in [1], [16]. Afterwards, algorithms
that identify in the limit certain interesting subclasses of regular languages have
been proposed in [2], [15], [7] and [12]. Even a general framework to construct
characterizable inference methods has been proposed in [10].

The family of Piecewise Testable Languages ($PWT$) has been previously stud-
i ed by Simon [17, 11, 14]. It is a subclass of regular languages somewhat similar
to the better known family of locally testable languages [5], [7]. The role played
there by segments of a certain length is played here by subwords, also of certain
length. By means of subword we understand, in this context, a sequence of non
necessarily consecutive symbols taken from a word.

A language is said to be $PWT$ if, for some integer $k$, if a word belongs to the
language, any other word with exactly the same set of subwords of length $k$ also
belongs to the language. The family of $PWT$ languages is a subclass of the class

* Work partially supported by the Spanish CICYT under grant TIC93-0633-C02
of star-free languages neither comparable to the family of reversible languages [2] nor to the family of locally-testable languages.

For any positive integer k, the family of k-PWT languages is a finite class of languages and so, it is identifiable from positive data in the limit. For the same reason it is also PAC learnable.

We propose in this work an algorithm that identifies in the limit the family of k-PWT languages whose time complexity is polynomial in the sum of the lengths of the input words.

2 Definitions and notation.

Let \( \Sigma \) be a finite alphabet and \( \Sigma^* \) the free monoid generated by \( \Sigma \) with concatenation as the binary operation and \( \lambda \) as neutral element. The length of a word \( x \) will be denoted as \( |x| \). Let \( \Sigma^k = \{ x \in \Sigma^* : |x| = k \} \) and \( \Sigma^{\leq k} = \bigcup_{i=0}^{k} \Sigma^i \). The prefix \( a_1a_2...a_i \) of a word \( x = a_1a_2...a_n \) will be denoted as \( x_{\leq i} \).

Given \( x, y \in \Sigma^* \), we say that \( x = a_1a_2...a_n \), with \( a_i \in \Sigma \), \( i = 1, 2...n \) is a subword of \( y \) if \( y = z_0a_1z_1a_2...a_nz_n \) with \( z_i \in \Sigma^*, i = 0, 1, 2...n \). We call this relationship division and denote it by \( x \mid y \). This relationship is compatible with the concatenation, that is, \( x \mid y \land x' \mid y' \Rightarrow xx' \mid yy' \).

Given a word \( y \in \Sigma^* \), the set of subwords of \( y \) and their multiplicity (the number of ways each subword can be obtained from \( y \)) can be evaluated by using either Pascal's formula or Magnus transformation (see [11] for details). A problem with both methods is that they are exponential with the length of the considered word.

The inverse problem is to determine if for a given value of \( k \) and a set of words \( A \) there exists a word \( y \) such that \( A \) is the set of \( k \)-subwords of \( y \). As far as we know, the problem has not been characterized yet (for example the set \( \{ab, ba\} \) cannot be the set of 2-subwords of any word \( y \) for it will also contain either the subword \( aa \) or \( bb \)).

Following [8], we say that an algorithm \( A \) identifies a class \( H \) of DFAs in the limit iff for any \( M \in H \), on input of any presentation of \( L(M) \), the infinite sequence of DFAs output by \( A \) converges to a DFA \( M' \) such that \( L(M) = L(M') \).

A class of \( H \) of DFAs is PAC-identifiable by an algorithm \( A \) iff on input of any parameters \( \epsilon, \delta \) \((0 < \epsilon, \delta < 1)\), for any DFA \( M \in H \) of size \( n \), for any positive integer \( m \) and for any probability distribution \( D \) on strings of \( \Sigma^{\leq m} \), if \( A \) obtains examples generated according to distribution \( D \), \( A \) produces a DFA \( M' \) such that, with probability at least \( 1 - \delta \), the probability of the set \( L(M) \oplus L(M') \) (where \( \oplus \) means the symmetric difference) is at most \( \epsilon \). Any finite class of languages is potentially PAC-learnable, and any consistent algorithm for a potentially learnable class, PAC learns the class [3].
3 K-Piecewise Testable Languages.

For a word \( x \), we define \( S(x, k) \) and \( NS(x, k) \) as follows:

\[
S(x, k) = \begin{cases} 
\{x\} & \text{if } |x| < k \\
\{z \in \Sigma^k : x \} & \text{if } |x| \geq k 
\end{cases}
\]

and

\[
NS(x, k) = \Sigma^\leq k - S(x, k)
\]

Let \( \equiv_k \) be the equivalence relation defined in \( \Sigma^* \) as follows:

\[
\forall x, y \in \Sigma^* (x \equiv_k y \iff S(x, k) = S(y, k)).
\]

The above relationship is a congruence of finite index. We denote as \( [x]_k \) the equivalence class of the word \( x \), that is,

\[
[x]_k = \left( \bigcap_{a_1 a_2 \ldots a_k \in S(x, k)} \Sigma^* a_1 \Sigma^* a_2 \ldots \Sigma^* a_k \Sigma^* \right) - \left( \bigcup_{a_1 a_2 \ldots a_k \in NS(x, k)} \Sigma^* a_1 \Sigma^* a_2 \ldots \Sigma^* a_k \Sigma^* \right).
\]

Properties [11]:

1. \( \equiv_{k+1} \) is a refinement of \( \equiv_k \).
2. Any equivalence class modulo \( \equiv_k \) is either a singleton or has infinite words.
3. If \( x = uv \) is such that \( \exists a \in \Sigma \) with \( S(ua, k) = S(u, k) \) then \( \forall n > 0, ua^n v \in [x]_k \).

A language \( L \) is said to be \( k \)-piecewise testable iff it is the union of equivalence classes modulo \( \equiv_k \). A language \( L \) is piecewise testable if there exists a value of \( k \) such that \( L \) is \( k \)-piecewise testable.

Following [17], a language \( L \subseteq \Sigma^* \) is piecewise testable if and only if \( L \) belongs to the Boole Algebra generated by the languages of the form \( \Sigma^* a_1 \Sigma^* a_2 \ldots \Sigma^* a_k \Sigma^* \) with \( k \geq 0 \), \( a_i \in \Sigma \), \( i = 1, \ldots, k \).

Given \( x, y \in \Sigma^* \), we write that \( x \leq_k y \) if and only if \( S(x, j) \subseteq S(y, j), \forall j \leq k \). This relationship is a partial order in \( \Sigma^* \). It is easily seen that \( x \mid y \Rightarrow x \leq_k y \), although the converse is not true (take, for example, \( k = 2, x = abab, y = baba \)).

The quotient set \( (\Sigma^*/\equiv_k)_{\leq k} \), being \( \leq_k \) the above relationship extended to equivalent classes, is a finite lattice with an absolute maximum (the class of words \( x \) such that \( S(x, k) = \Sigma^k \), which is always feasible) and an absolute minimum (the class of the empty word). With this order relation, the sequence \( \{x_i\}_{i=1}^n \) of the prefixes of a word \( a_1 a_2 \ldots a_n \) always form an ascending chain in the lattice.

Given a DFA \( A = (Q, \Sigma, \delta, q_0, F) \), we say that \( q \in Q \) is at level \( i \) iff \( i = \min\{ |x| : x \in \Sigma^*, \delta(q_0, x) = q \} \).
The following properties come out directly from the above definitions:

1. If $L$ is a $k$-piecewise testable language, then $L$ is $j$-piecewise testable language, $\forall j \geq k$.
2. If $L$ is $k$-piecewise testable then $L$ is regular.
3. If $A$ is an automaton accepting a $k$-piecewise testable language $L$, there are no transitions in $A$ from a state at level $i$ to states at level $j$, $j < i$.

---

Input: $k \in N$, $S$ set of words over $\Sigma^*$
Output: DFA $A = (Q, \Sigma, \delta, q_1, F)$, consistent with the data:
$L(A) \subseteq L_k(S)$

Method:

1. $new = 1;
2. Q = \{q_1\};
3. \delta = \emptyset;
4. F = \emptyset;
5. \forall x = a_1a_2...a_n \in S$ do
6. \hspace{1cm} $i = 1$
7. \hspace{1cm} $state = q_1$
8. \hspace{1cm} While ($i \leq n \land \exists j : (state, a_i, q_j) \in \delta$) do
9. \hspace{2cm} $state = q_j$
10. \hspace{2cm} If $i = n$
11. \hspace{3cm} then $F = F \cup \{state\}$
12. \hspace{3cm} endif;
13. \hspace{2cm} $i = i + 1$
14. EndWhile;
15. While ($i \leq n$) do
16. \hspace{1cm} If $(q_j, a_i, q_j) \in \delta$
17. \hspace{2cm} then $i = i + 1;$
18. else
19. \hspace{2cm} $new = new + 1;
20. \hspace{2cm} j = new;
21. \hspace{2cm} Q = Q \cup \{q_j\};
22. \hspace{2cm} \delta = \delta \cup \{(state, a_i, q_j)\};
23. \hspace{2cm} st = S(a_1a_2...a_i, k);
24. \hspace{2cm} If $\exists a \in \Sigma : S(a_1a_2...a_i, k) = st$
25. \hspace{3cm} then $\delta = \delta \cup \{(q_j, a, q_j)\}$
26. \hspace{3cm} endif;
27. \hspace{2cm} state = q_j;
28. \hspace{2cm} $i = i + 1;$
29. EndIf;
30. EndWhile
31. $F = F \cup \{q_j\}$;
32. End $\forall$
33. $A = (Q, \Sigma, \delta, q_1, F)$
34. EndMethod.

Fig. 1. Algorithm $k$-PWTI.
4. The class of k-piecewise testable languages is a finite class, bounded above by $|\mathcal{P}(\Sigma^k)| = 2^{2^{|\Sigma|^k}}$ (where $\mathcal{P}$ means the power set).

4 Smallest K-Piecewise Testable Language that contains a positive sample $S$. Learning Algorithm.

Given $k > 0$ we can associate to a positive sample $S = \{x_1, x_2, ... x_n\}$ a $k$-piecewise testable language $L_k(S)$ as follows:

$$x \in L_k(S) \iff \exists i, 1 \leq i \leq n : S(x, k) = S(x_i, k).$$

Properties:

It is easy to prove that:

1. $S \subseteq L_k(S)$
2. $L_k(S)$ is the smallest k-piecewise testable language that contains $S$.
3. $S' \subseteq S$ implies that $L_k(S') \subseteq L_k(S)$.
4. $L_{k+1}(S) \subseteq L_k(S)$.
5. If $k > \max_{x_i \in S} |x_i|$ then $L_k(S) = S$.

Based on the above definitions and properties, we propose a learning algorithm: the k-Piecewise Testable Inference algorithm (k-PWTI) that, with input of a positive sample $S$, outputs a DFA consistent with $S$ that accepts a subset of the smallest k-PWT language that contains $S$. The algorithm is shown in Figure 1.

4.1 Example of run

Let $k = 2$ and $S = \{aba, aaba, aa\}$.

While, for better understanding, in figures 2, 3 and 4 each state of the shown automata is labelled with the set $S(a_1a_2...a_i, k)$, in the algorithm we only need to maintain the set for the most recently created state.

The loop (1) is used when the algorithm has constructed part of the automaton, to try to analyze the longest prefix of new incoming word, so it is never used with the first word of the data. If intending to do so, the whole word is accepted, a new state is possibly added to the list of finals in (2).

![Output automata when K-PWTI is executed on input aba](image)

**Fig. 2.** Output automata when K-PWTI is executed on input aba.
Fig. 3. Output automata when K-PWTI is executed on input \{aba, aaba\}.

So with the arrival of the first word "aba" the algorithm directly enters the loop (3) resulting the automaton shown in figure 2. Transitions of type \( (q, a, q) \) are created for all the prefixes of the word that meet the condition (4).

With the word "aaba", the algorithm executes (1) accepting the prefix "a". As the transition \( \delta(q_2, a) \) does not exist, the algorithm enters the loop (2) creating the transition \( (q_2, a, q_5) \). The prefix "aa" meets condition (4) so \( (q_5, a, q_5) \) is added to the list of transitions. When it finishes evaluating the word, the resulting automaton is shown in figure 3. Observe that this automaton recognizes the language of all the words \( x \) such that \( S(x, 2) = \{aa, ab, ba\} \).

At the arrival of the word "aa", which belongs to a different class modulo \( \equiv_k \) as it meets condition (2), the algorithm establishes the already created \( q_5 \) as final state, outputting the algorithm shown in figure 4.

Fig. 4. Output automata when K-PWTI is executed on input \{aba, aaba, aa\}. 

5 Convergence and time complexity

5.1 Convergence of the algorithm

Theorem 1. The algorithm KWPTI identifies the class of $k$-Piecewise Testable languages in the limit.

Proof: It is easily seen that the algorithm is consistent with the received data as for each set $S$ outputs an automaton accepting the set \(\{ua^n v, \forall uv \in S : S(u a^n k) = S(u, k), |u| \geq k, n \geq 0\} \subseteq L_k(S)\).

Convergence in the limit follows from the finiteness of the lattice established in section 3.

A characteristic sample (a set of input words that make the algorithm to converge) for a class $[x]_k$ can be obtained as follows:

Let $l = \min \{ |y| : y \in [x]_k \}$ (a way of calculating $l$ is shown later). The set \(\{z \in \Sigma^{l+k-1} : S(z, k) = S(x, k)\}\) is a characteristic sample for the class $[x]_k$ and algorithm $k$-PWTLI as with that sample we have every possible ascending chain in the lattice. The value $l + k - 1$ is necessary to obtain the loops at level $k + 1$ in the automaton (i.e.: transitions of the form $(S(a_1...a_k, k), a, S(a_1...a_k, k))$, $a \in \Sigma$).

The number $l$ in the above definition can be iteratively evaluated from any word $y \in [x]_k$ as follows:

\[ y_1 = y \]

Repeat

If $\exists a \in \Sigma : y_i = u a v, S(u a, k) = S(u, k)$ then $y_{i+1} = u v$

else $y_{i+1} = y_i$

endif

until $y_{i+1} = y_i$

\[ l = |y_i| \]

Theorem 2. The algorithm $k$-PWTI is PAC-learnable for the class of $k$-Piecewise Testable languages

Proof: It follows immediately from the finiteness of the class and the consistency of the algorithm.

5.2 Time complexity

Every state in the automaton can be represented by a set of subwords of length $k$. If $m$ is the size of such a set, $m \leq |\Sigma|^k$. As every subword of length $k + 1$ contains $k + 1$ subwords of length $k$, the number of operations needed to create a new state (concatenating a symbol to the subwords of length $k$) is $m \cdot k$. As we have to test if every word of length $k$ belongs to the new created state and this operation costs $k \cdot \lg m$, the total cost of creating a new state is $m \cdot k^2 \cdot \lg m$. 

The operation described has to be done at most $n$ times, being $n = \sum_{x \in S} |x|$, so the total cost is $(mk^2 \lg m)n \leq |\Sigma|^k k^2 \lg |\Sigma|^k n = k^3 |\Sigma|^k \lg |\Sigma| n$ in the worst case. Since $|\Sigma|$ and $k$ can be considered as constants, the above algorithm is $O(n)$, where $n$ is the sum of the lengths of the input data.

Acknowledgements

The authors gratefully acknowledge the thoughtful suggestions and corrections of anonymous referees.

References