Learning k-Variable Pattern Languages Efficiently Stochastically Finite on Average from Positive Data

Peter Rossmanith[1] and Thomas Zeugmann[2]

[1] Institut für Informatik, Technische Universität München, 80290 München, Germany
rossmani@in.tum.de
[2] Department of Informatics, Kyushu University, Kasuga 816-8580, Japan
thomas@i.kyushu-u.ac.jp

Abstract. The present paper presents a new approach of how to convert Gold-style [4] learning in the limit into *stochastically finite* learning with *high confidence*. We illustrate this approach on the concept class of all pattern languages. The transformation of learning in the limit into stochastically finite learning with high confidence is achieved by first analyzing the Lange–Wiehagen [7] algorithm with respect to its average-case time behavior until convergence. This algorithm learns the class of all pattern languages in the limit from positive data. The expectation of the total learning time is analyzed and *exponentially* small tail bounds are established for a large class of probability distributions. For patterns containing k different variables Lange and Wiehagen's algorithm possesses an expected total learning time of $O\big((1/\alpha)^k(\log_{1/\beta}(k))E[\Lambda]\big)$, where α and β are two easily computable parameters from the underlying probability distribution, and $E[\Lambda]$ is the expected example string length.
Finally, we show how to arrive at stochastically finite learning with high confidence.

1 Introduction

Suppose you have to deal with a learning problem of the following kind. On the one hand, it is known that the problem is not solvable within the PAC model unless you achieve the needed breakthrough in complexity theory. On the other hand, your learning problem has been proved to be learnable within Gold's [4] learning in the limit model. Here, a learner is successively fed data about the concept to be learned and is computing a sequence of hypotheses about the target object. However, the only knowledge you have about this sequence is its convergence in the limit to a hypothesis correctly describing the target concept. Therefore, you never know whether the learner has already converged. Such an uncertainty may not be tolerable in many applications. In general, there may be no way to overcome this uncertainty. But part of the problem is caused by the fact that learning in the limit has to be successful from all possible data sequences. It is intuitively clear that there are data sequences that contain huge amounts of redundant data before successful learning can take place. But such sequences may be rare in practice.

What we would like to present in this paper is a rather general method to overcome the difficulties described above. This method is based on an average-case analysis of known limit learners with respect to their time complexity including tail bounds. Assuming a certain amount of knowledge concerning the underlying probability distributions, we can put it all together and arrive at *stochastically finite learning with high confidence.* This learning model may be considered as a variant of PAC-learning. The major differences are easily explained. First, stochastically finite learning with high confidence is not completely distribution independent. Thus, from that perspective, this variant is weaker than the PAC-model. But the hypothesis computed is *probably exactly correct.* Moreover, the learner is fed positive data only, while the correctness of its final hypothesis is measured with respect to both positive and negative data.

Second, it should be emphasized that our approach is applicable in a rather general context. Suppose you have a PAC-learner for the concept class you want to learn. In that case, additional knowledge about the underlying probability distributions directly yields better hypotheses, i.e., *probably correct* ones instead of *probably approximately correct* ones. But its main advantage is achieved when dealing with situations as described above, i.e., in those cases where it is highly unlikely to obtain a PAC-learner. Now, instead of facing all the disadvantages of limit learning additional knowledge about the underlying probability distributions nicely buys a learner that is even more reliable than a PAC-learner.

In the following, we exemplify this approach by dealing with the learnability of the well-known pattern languages (PAT for short), a prominent and important language family that can be learned from positive data (cf. [1, 10, 12]). There are also numerous interesting applications for pattern language learners (cf., e.g., [12] and the references therein).

Nevertheless, despite its importance there is still a bottleneck concerning efficient learning algorithms. Kearns and Pitt [5], Ko, Marron and Tzeng [6] and Schapire [11] intensively studied the learnability of pattern languages in the PAC–learning model. In particular, Schapire [11] proved that the class PAT is not PAC-learnable regardless of the representation used by the learning algorithm, provided only that the learner is requested to output a polynomial-size hypothesis that can be evaluated in polynomial time, unless $\mathcal{P}_{/poly} = \mathcal{NP}_{/poly}$. However, the class Pat of all patterns is not a polynomial time representation for PAT, since the membership problem for PAT with respect to Pat is $\mathcal{NP}$-complete [1]. In contrast, we show Pat to be stochastically finite learnable with high confidence with respect to Pat (cf. Theorem 9). On the other hand, Kearns and Pitt [5] designed a polynomial-time PAC-learner for the set of all k-variable pattern languages (k arbitrarily fixed) if only product distributions are allowed. But the constant in the running time achieved depends *doubly exponentially* on k, and thus, their algorithm becomes rapidly impractical when k increases.

In contrast, our stochastically finite learner achieves a running time whose constant depends only exponentially on the number k of different variables occurring in the target pattern and is otherwise *linearly* bounded in the expected length of sample strings fed to the learner (cf. Corollary 1). The price paid is rather small. We restrict the class of all probability distributions to a subclass that has an arbitrary but fixed bound on two parameters arising naturally. In essence, that means at least two letters from the underlying probability distribution have a *known* lower bound on their probability.

We use the Lange–Wiehagen [7] algorithm (LWA for short) as the basic ingredient for achieving our goal. The LWA learns the class of all pattern languages in the limit from positive data. That means the learner is fed successively example strings and its previously made hypothesis, and it computes from these input data a new pattern as its hypothesis. The sequence of all hypotheses stabilizes to a single pattern which generates the target pattern language. First, we *generalize* and *improve* the average-case analysis of the same algorithm performed by Zeugmann [14] for its expected *total learning time*. The time taken by a learner for computing a single hypothesis from its input data is usually called *update time*. The total learning time is the time taken by the learner until convergence, i.e., the sum of all update times until successful learning. Note that it is a highly non-trivial task to define an appropriate complexity measure for learning in the limit (cf. [8]). The total learning time has been introduced by Daley and Smith [3]. As Pitt [8] pointed out, allowing the total learning time to depend on the length of the examples seen so far is unsatisfactory, since the learner may delay convergence until a sufficiently long example appears so that the algorithm may meet the wanted polynomial time bound. We therefore measure the total learning time *only* in dependence on the length of the target pattern.

Second, we show how the improved analysis can be used to establish *stochastically finite learnability*. The basic idea can be described as follows. Based on exponentially shrinking tail bounds obtained from our average case analysis for the expected *total learning time*, the new learner takes as input a randomly generated text and a *confidence parameter* δ. It then computes internally hypotheses until the confidence bound is met and outputs exclusively its last guess (cf. Section 4, Definition 2).

Owing to lack of space some results and most proofs could not be included into this extended abstract; they can be found in the full paper (cf. [9]).

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ be the set of all natural numbers, and let $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$.

Following Angluin [1] we define patterns and pattern languages as follows. Let $\mathcal{A} = \{0, 1, \ldots\}$ be any non-empty finite alphabet containing at least two elements. By $\mathcal{A}^*$ we denote the set of all strings over $\mathcal{A}$ and by $\mathcal{A}^+ = \mathcal{A}^* \setminus \varepsilon$ all non-null strings. By $|\mathcal{A}|$ we denote the cardinality of $\mathcal{A}$. Furthermore, let $X = \{\, x_i \mid i \in \mathbb{N} \,\}$ be an infinite set of variables such that $\mathcal{A} \cap X = \emptyset$. *Patterns* are non-empty strings over $\mathcal{A} \cup X$, e.g., 01, $0x_0111$, $1x_0x_00x_1x_2x_0$ are patterns. The length of a string $s \in \mathcal{A}^*$ and of a pattern π is denoted by $|s|$ and $|\pi|$, respectively. A pattern π is in *canonical form* provided that if k is the number of different variables in π

then the variables occurring in π are precisely $x_0, \ldots, x_{k-1}$. Moreover, for every j with $0 \le j < k-1$, the leftmost occurrence of x_j in π is left to the leftmost occurrence of x_{j+1} in π. The examples given above are patterns in canonical form. In the sequel we assume, without loss of generality, that all patterns are in canonical form. By *Pat* we denote the set of all patterns in canonical form.

Let $\pi \in Pat$, $1 \le i \le |\pi|$; we use $\pi(i)$ to denote the i-th symbol in π. If $\pi(i) \in \mathcal{A}$, then $\pi(i)$ is a *constant*; otherwise $\pi(i) \in X$ is a *variable*. Analogously, by $s(i)$ we denote the i-th symbol in s for $s \in \mathcal{A}^+$. By $\#\mathrm{var}(\pi)$ we denote the number of different variables occurring in π, and by $\#_{x_i}(\pi)$ we denote the number of occurrences of variable x_i in π. If $\#\mathrm{var}(\pi) = k$, then we refer to π as to a *k-variable pattern*. Let $k \in \mathbb{N}$, by Pat_k we denote the set of all *k-variable patterns*. Furthermore, let $\pi \in Pat_k$, and let $u_0, \ldots, u_{k-1} \in \mathcal{A}^+$; then we denote by $\pi[x_0/u_0, \ldots, x_{k-1}/u_{k-1}]$ the string $w \in \mathcal{A}^+$ obtained by substituting u_j for each occurrence of x_j, $j = 0, \ldots, k-1$, in the pattern π. For example, let $\pi = 0x_0 1x_1 x_0$. Then $\pi[x_0/10, x_1/01] = 01010110$. The tuple $(u_0, \ldots, u_{k-1})$ is called a *substitution*. Furthermore, if $|u_0| = \cdots = |u_{k-1}| = 1$, then we refer to $(u_0, \ldots, u_{k-1})$ as to a *shortest substitution*. Now, let $\pi \in Pat_k$, and let $S = \{(u_0, \ldots, u_{k-1}) \mid u_j \in \mathcal{A}^+,\ j = 0, \ldots, k-1\}$ be any finite set of substitutions. Then we set $S(\pi) = \{\pi[x_0/u_0, \ldots, x_{k-1}/u_{k-1}] \mid (u_0, \ldots, u_{k-1}) \in S\}$, i.e., $S(\pi)$ is the set of all strings obtained from pattern π by applying all the substitutions from S to it. For every $\pi \in Pat_k$ we define the *language generated by pattern π* by $L(\pi) = \{\pi[x_0/u_0, \ldots, x_{k-1}/u_{k-1}] \mid u_0, \ldots, u_{k-1} \in \mathcal{A}^+\}$. By PAT_k we denote the set of all *k-variable pattern languages*. Finally, $PAT = \bigcup_{k \in \mathbb{N}} PAT_k$ denotes the set of all pattern languages over $\mathcal{A}$. Note that for every $L \in PAT$ there is precisely one pattern $\pi \in Pat$ such that $L = L(\pi)$ (cf. [1]).

We are interested in *inductive inference*, which means to gradually learn a concept from successively growing sequences of examples. If L is a language to be identified, a sequence $(s_1,\ s_2,\ s_3,\ \ldots)$ is called a *text* for L iff $L = \{s_1,\ s_2,\ s_3,\ \ldots\}$ (cf. [4]). However, in practical applications, the requirement to exhaust the language to be learned will be hardly fulfilled. We therefore *omit* this restriction here. Instead, we assume that the sequence $t = s_1,\ s_2,\ s_3,\ \ldots$ contains "enough" information to recognize the target pattern. As for the LWA, "enough" can be made precise by requesting that sufficiently many shortest strings appear in the text. We shall come back to this point when defining admissible probability distributions.

An *inductive inference machine* (IIM) is an algorithm that takes as input larger and larger initial segments of a text and outputs, after each input, a hypothesis from a prespecified *hypothesis space* (cf. [4]). In the case of pattern languages the hypothesis space is *Pat*.

Definition 1. *PAT* is called learnable in the limit from text iff there is an IIM M such that for every $L \in PAT$ and every text for L,

(1) for all $n \in \mathbb{N}^+$, $M(t_n)$ is defined,
(2) there is a pattern $\pi \in Pat$ such that $L(\pi) = L$ and for almost all $n \in \mathbb{N}^+$, $M(t_n) = \pi$.

It is well-known that pattern languages are learnable in the limit from text (cf. [1]).

Whenever one deals with the average case analysis of algorithms one has to consider probability distributions over the relevant input domain. For learning from text, we have the following scenario. Every string of a particular pattern language is generated by a substitution. Therefore, it is convenient to consider probability distributions over the set of all possible substitutions. That is, if $\pi \in Pat_k$, then it suffices to consider any probability distribution D over $\mathcal{A}^+ \times \cdots \times \mathcal{A}^+$ (k times). For $(u_0, \ldots, u_{k-1}) \in \mathcal{A}^+ \times \cdots \times \mathcal{A}^+$ we denote by $D(u_0, \ldots, u_{k-1})$ the probability that variable x_0 is substituted by u_0, variable x_1 is substituted by u_1, …, and variable x_{k-1} is substituted by u_{k-1}.

In particular, we mainly consider a special class of distributions, i.e., *product distributions*. Let $k \in \mathbb{N}^+$, then the class of all product distributions for Pat_k is defined as follows. For each variable x_j, $0 \le j \le k-1$, we assume an arbitrary probability distribution D_j over $\mathcal{A}^+$ on substitution strings. Then we call $D = D_0 \times \cdots \times D_{k-1}$ product distribution over $\mathcal{A}^+ \times \cdots \times \mathcal{A}^+$, i.e., $D(u_0, \ldots, u_{k-1}) = \prod_{j=0}^{k-1} D_j(u_j)$. Moreover, we call a product distribution *regular* if $D_0 = \cdots = D_{k-1}$. Throughout this paper, we restrict ourselves to deal with *regular* distributions. We therefore use d to denote the distribution over $\mathcal{A}^+$ on substitution strings, i.e, $D(u_0, \ldots, u_{k-1}) = \prod_{j=0}^{k-1} d(u_j)$. As a special case of a regular product distribution we sometimes consider the *uniform* distribution over $\mathcal{A}^+$, i.e., $d(u) = 1/(2 \cdot |\mathcal{A}|)^\ell$ for all strings $u \in \mathcal{A}^+$ with $|u| = \ell$.

Note, however, that most of our results can be generalized to larger classes of distributions. Finally, we can provide the announced specification of what is meant by "enough" information. We call a regular distribution *admissible* provided $d(a) > 0$ for at least two different elements $a \in \mathcal{A}$.

Following Daley and Smith [3] we define the total learning time as follows. Let M be any IIM that learns all the pattern languages. Then, for every $L \in PAT$ and every text t for L, let $Conv(M, t)$ be the least number $m \in \mathbb{N}^+$ such that $M(t_n) = M(t_m)$ for all $n \ge m$, denotes the *stage of convergence* of M on t. Moreover, by $T_M(t_n)$ we denote the time to compute $M(t_n)$. We measure this time as a function of the length of the input and call it *update time*. Finally, the total learning time taken by the IIM M on input t, one string at a time, is defined as

$$TT(M, t) =_{df} \sum_{n=1}^{Conv(M,t)} T_M(t_n).$$

Assuming any fixed admissible probability distribution D as described above, we aim to evaluate the *expectation* of $TT(M, t)$ with respect to D which we call *total average learning time*.

The model of computation as well as the representation of patterns we assume is the same as in Angluin [1]. We assume a random access machine that performs a reasonable menu of operations each in unit time on registers of length $O(\log n)$ bits, where n is the input length.

Finally, we recall the LWA. The LWA works as follows. Let h_n be the hypothesis computed after reading $s_1, \ldots, s_n$, i.e., $h_n = M(s_1, \ldots, s_n)$. Then $h_1 = s_1$

and for all $n > 1$:

$$h_n = \begin{cases} h_{n-1}, & \text{if } |h_{n-1}| < |s_n| \\ s_n, & \text{if } |h_{n-1}| > |s_n| \\ h_{n-1} \cup s_n, & \text{if } |h_{n-1}| = |s_n| \end{cases}$$

The algorithm computes the new hypothesis only from the latest example and the old hypothesis. If the latest example is longer than the old hypothesis, the example is ignored, i.e., the hypothesis does not change. If the latest example is shorter than the old hypothesis, the old hypothesis is ignored and the new example becomes the new hypothesis. Hence, the LWA is quite simple and the update time will be very fast for these two possibilities.

If, however, $|h_{n-1}| = |s_n|$ the new hypothesis is the *union* of h_{n-1} and s_n. The union $\varrho = \pi \cup s$ of a canonical pattern π and a string s of the same length is defined as

$$\varrho(i) = \begin{cases} \pi(i), & \text{if } \pi(i) = s(i) \\ x_j, & \text{if } \pi(i) \neq s(i) \ \& \ \exists k < i : [\varrho(k) = x_j,\ s(k) = s(i),\ \pi(k) = \pi(i)] \\ x_m, & \text{otherwise, where } m = \#var(\varrho(1)\dots\varrho(i-1)), \end{cases}$$

where $\varrho(0) = \varepsilon$ for notational convenience. The resulting pattern is canonical.

Obviously, the union operation can be computed in quadratic time. We finish this section by providing a linear-time algorithm computing the union operation. The only crucial part is to determine whether there is some $k < i$ with $\varrho(k) = x_j$, $s(k) = s(i)$, and $\pi(k) = \pi(i)$. The new algorithm uses an array $I = \{1, \dots, |s|\}^{\mathcal{A} \times (\mathcal{A} \cup \{x_0, \dots, x_{|\pi|-1}\})}$ for finding the correct k, if any, in *constant* time. The array I is *partially* initialized by writing the first position into it at which $s(i), \pi(i)$ occurs. Then, for each position i, the algorithm checks whether $I_{s(i),\pi(i)} = i$. Suppose it is, thus $s(i), \pi(i)$ did not occur left to i. Hence, it remains to check whether or not $\pi(i) = s(i)$ and $\varrho(i)$ can be immediately output. If $I_{s(i),\pi(i)} \neq i$, then $s(i), \pi(i)$ did occur left to i. Hence, in this case it suffices to output $\varrho(j)$ where $j = I_{s(i),\pi(i)}$.

Theorem 1. *The union operation can be computed in linear time by Algorithm* 1.

Algorithm 1

```
    Input: A pattern $\pi$ and a string $s \in \mathcal{A}^+$ such that $|\pi| = |s|$.
    Output: $\pi \cup s$
    for $i = 1, \dots, |s|$ do $I_{s(i),\pi(i)} \leftarrow i$ od; $m \leftarrow 0$;
    for $i = 1, \dots, |s|$ do $j \leftarrow I_{s(i),\pi(i)}$;
        if $i = j$ then
            if $\pi(i) = s(i)$ then $\varrho(i) = \pi(i)$
            else $\varrho(i) \leftarrow x_m$; $m \leftarrow m + 1$ fi
        else $\varrho(i) = \varrho(j)$ fi
    od
```

The correctness of this algorithm can be easily proved inductively by formalizing the argument given above. We omit the details.

3 Results of the Analysis

Following [14] we perform the desired analysis in dependence on the number k of different variables occurring in the target pattern π. If $k = 0$, then the LWA immediately converges. Therefore, in the following we assume $k \in \mathbb{N}^+$, and $\pi \in Pat_k$. For analyzing the *average-case* behavior of the LWA, in the following we let $t = s_1, s_2, s_3, \ldots$ range over all randomly generated texts with respect to some arbitrarily fixed admissible probability distribution D. Then the stage of convergence is a random variable which we denote by C. Note that the distribution of C depends on π and on D. We introduce several more random variables. By Λ_i we denote the length of the example string s_i, i.e., $\Lambda_i = |s_i|$. Since all Λ_i are independent and identically distributed, we can assume that the random variable has the same distribution as Λ. We shall use Λ when talking about the length of an example when the number of the example is not important. Particularly, we will often use the expected length of a random example $E[\Lambda]$.

Let $T = \Lambda_1 + \Lambda_1 + \ldots + \Lambda_C$ be the *total length* of examples processed until convergence. Whether the LWA converges on $s_1, \ldots, s_r$ depends only on those examples s_i with $s_i \in L(\pi)_{min} = \{\, w \mid w \in L(\pi),\ |w| = |\pi| \,\}$. It should be mentioned that *without* seeing a single *shortest* string, k-variable pattern languages *cannot* be *learned* provided $k > 1$. This is easily seen if one looks at patterns $x_0 \cdots x_k$ and $x_0 \cdots x_k x_{k+1}$. The languages they generate are identical except for strings of length k. Using a result from Zeugmann [14], this negative result extends to arbitrary k and $k+1$ variable patterns, respectively. Moreover, as we shall see, waiting for one shortest strings takes almost the same time as waiting for all the shortest strings needed to converge.

Let $r \in \mathbb{N}^+$; by M_r we denote the number of minimum length examples among the first r strings, i.e., $M_r = |\{\, i \mid 1 \le i \le r \text{ and } \Lambda_i = |\pi| \,\}|$. In particular, M_C is the number of minimum length examples read until convergence. We assume that computing $\varrho \cup s$ takes at most $c \cdot |\varrho|$ steps, where c is a constant that depends on the implementation of the union operation.

We express all estimates with the help of the following parameters: $E[\Lambda]$, c, α and β. To get concrete bounds for a concrete implementation one has to obtain c from the algorithm and has to compute $E[\Lambda]$, α, and β from the admissible probability distribution D. Let $u_0, \ldots, u_{k-1}$ be independent random variables with distribution d for substitution strings. Whenever the index i of u_i does not matter, we simply write u or u'.

The two parameters α and β are now defined via d. First, α is simply the probability that u has length 1, i.e., $\alpha = \Pr(|u| = 1) = \sum_{a \in \mathcal{A}} d(a)$. Second, β is the conditional probability that two random words that get substituted into π are identical under the condition that both their length are 1, i.e.,

$$\beta = \Pr\big(u = u' \mid |u| = |u'| = 1\big) = \textstyle\sum_{a \in \mathcal{A}} d(a)^2 \Big/ \Big(\sum_{a \in \mathcal{A}} d(a)\Big)^2 .$$

The parameters α and β are therefore quite easy to compute even for complicated distributions since they depend only on $|\mathcal{A}|$ point probabilities. We can also compute $E[\Lambda]$ for a pattern π from d quite easily. Let $\hat{\alpha} = 1/\alpha$.

Theorem 2. $E[TT] = O\big(\hat{\alpha}^k(\log_{1/\beta}(k))E[\Lambda]\big)$.

Next, we insert the parameter for the uniform distribution into Theorem 2. For the uniform distribution we get $\hat{\alpha} = 2$, $\beta = 1/|\mathcal{A}|$, and $E[\Lambda] \leq 2|\pi|$.

Theorem 3. $E[TT] = O(2^k|\pi|\log_{|\mathcal{A}|}(k))$ *for the uniform distribution.*

Often time is the most precious resource, but the number of examples until convergence can also be interesting, if the gathering of examples is expensive.

Theorem 4. $E[C] = O(\hat{\alpha}^k \cdot \log_{1/\beta}(k))$.

We can even get a better understanding of the behavior if we examine the union operations by themselves. Is it worthwhile to optimize the computation of $w \cup \pi$? Let U be the number of union operations and V be the time spent in union operations.

Theorem 5.

(1) $E[U] = O(\hat{\alpha}k + \log_{1/\beta}(k))$
(2) $E[V] = O(\hat{\alpha}kE[\Lambda] + \log_{1/\beta}(k)|\pi|)$ *if the union operation is performed by Algorithm* 1*,*
(3) $E[V] = O(\hat{\alpha}kE^2[\Lambda] + \log_{1/\beta}(k)|\pi|^2)$ *if the union operation is performed by the naïve algorithm.*

Consequently, in most cases we can use the simple, quadratic algorithm for union operations, since they make only a small contribution to the overall running time. The proof of this counterintuitive result is unfortunately very long and can be found in the full paper (cf. [9]).

3.1 Tail Bounds

Finally we have to ask whether the expected value of the total learning time is sufficient for judging the LWA. The expected value of a random variable is only one aspect of its distribution. In general we might also be interested on how often the learning time exceeds the average substantially. Again this is a question motivated mainly by practical considerations. Equivalently we can ask, how good the distribution is concentrated around its expected value. Often this question is answered by estimating the *variance*, which enables the use of Chebyshev's inequality. If the variance is not available, Markov's inequality provides us with (worse) tail bounds: $\Pr(X \geq t \cdot E[X]) \leq 1/t$. The Markov inequality is quite general but produces only weak bounds. The next theorem gives better tail bounds for a large class of learning algorithms including the LWA. A learner is set-driven, if its outputs depends only on the range of its input (cf. [13]). Conservative learners maintain their actual hypotheses at least as long as they have not seen data contradicting them (cf. [2]).

Theorem 6. *Let X be the sample complexity of a conservative and set-driven learning algorithm. Then $V[X] \leq 20E[X]^2$ and $\Pr(X \geq t \cdot E[X]) \leq 2^{-t/2}$ for all $t \in \mathbb{N}$.*

Theorem 6 holds also for conservative, *rearrangement independent* learners, which means that each hypothesis must not depend on the order of the examples.

3.2 The Sample Complexity

In this section we estimate the sample complexity. While being of interest itself, whenever acquiring examples is expensive, $E[C]$ is also an important ingredient in the estimation of the total learning time.

Lemma 1. $\Pr(M_C > m) = \Pr(C > r \mid M_r = m) \leq \binom{k}{2}\beta^m + k\beta^{m/2}$ *for all* $m, r \in \mathbb{N}^+$ *with* $r \geq m$.

Proof. Without loss of generality, let $S_r = \{s_1, \ldots, s_m\}$, i.e., $m = r$. Additionally, we can make the assumption that all strings $s_i \in S_r$ have length k, since we need to consider only shortest words for M_C and we can assume that $\pi = x_0 x_1 \ldots x_{k-1}$ (cf. [14]). For $1 \leq j \leq k$ let $c_j = s_0(j)s_1(j)\ldots s_{m-1}(j)$ be the jth *column* of a matrix whose rows are $s_1, \ldots, s_m$.

The algorithm computes the hypothesis π on input S_r iff no column is constant and there are no identical columns. The probability that c_j is constant is at most $\beta^{m/2}$, since $m/2$ pairs have to be identical, but this short argument works only for even m. A slightly more complicated proof shows that the same bound holds also for odd m. The probability that at least one of the k columns is constant is then at most $k\beta^{m/2}$.

The probability that $c_i = c_j$ is β^m if $i \neq j$. The probability that some columns are equal is therefore at most $\binom{k}{2}\beta^m$. The probability that there is a constant column *or* that there are identical columns is at most $\binom{k}{2}\beta^m + k\beta^{m/2}$.

Inserting the above tail bounds into the definition of the expected value yields an upper bound on $E[M_C]$.

Lemma 2. $E[M_C] \leq (2\ln(k)+3)/(\ln(1/\beta))+2 \leq 7\log_{1/\beta}(k)+2 = O(\log_{1/\beta}(k))$.

Proof. M_C is the number of shortest words read until convergence. By Lemma 1 we have $\Pr(M_C > m) \leq \binom{k}{2}\beta^m + k\beta^{m/2}$ and thus $E[M_C]$ is

$$\sum_{m=0}^{\infty} \Pr(M_C > m) \leq \ell + \sum_{m=\ell}^{\infty}\left(\binom{k}{2}\beta^m + k\beta^{m/2}\right) = \ell + \binom{k}{2}\frac{\beta^\ell}{1-\beta} + k\frac{\sqrt{\beta}^{\,\ell}}{1-\sqrt{\beta}}$$

for each natural number ℓ. We choose $\ell = \lceil 2\log_{1/\beta}(k)\rceil + 1$, which yields when inserted in above inequality $E[M_C] \leq \ell + \frac{\beta}{1-\beta} + \frac{\sqrt{\beta}}{1-\sqrt{\beta}}$. The lemma now follows from the inequality $\frac{\beta}{1-\beta} + \frac{\sqrt{\beta}}{1-\sqrt{\beta}} \leq 3/\ln(1/\beta)$, which can be proved by standard methods from calculus.

Our next major goal is to establish an upper bound on the overall number of examples to be read on average by the LWA until convergence.

Theorem 7. $E[C] = \hat{\alpha}^k E[M_C] \leq \hat{\alpha}^k(7\log_{1/\beta}(k)+2) = O(\hat{\alpha}^k\log_{1/\beta}(k))$.

Proof. The LWA converges after reading exactly C example strings. Among these examples are M_C many of minimum length. Prior to these minimum length words come M_C possibly empty blocks of words whose length is bigger than

$|\pi|$. Let us call the numbers of those words in the ith block G_i. Then $C = G_1 + G_2 + \cdots + G_{M_C} + M_C$. It is easy to compute the distribution of G_i:

$$\Pr(G_i = m) = \Pr(\Lambda > |\pi|)^m \Pr(\Lambda = |\pi|) = (1-\alpha^k)^m \alpha^k \tag{1}$$

Of course, all G_i are identically distributed and independent. The expected value of C is therefore

$$\begin{aligned} E[C] &= E[M_C] + E[G_1 + \cdots + G_{M_C}] \\ &= E[M_C] + \sum_{m=0}^{\infty} E[G_1 + \cdots + G_m \mid M_C = m] \cdot \Pr(M_C = m) \\ &= E[M_C] + \sum_{m=0}^{\infty} m \cdot E[G_1] \cdot \Pr(M_C = m) = E[M_C] + E[M_C] \cdot E[G_1] \qquad (2) \end{aligned}$$

The expected value of G_1 is

$$E[G_1] = \sum_{m=0}^{\infty} m \cdot \Pr(\Lambda > |\pi|)^m \cdot \Pr(\Lambda = |\pi|) = \frac{\Pr(\Lambda > |\pi|)}{\Pr(\Lambda = |\pi|)} = \frac{1-\alpha^k}{\alpha^k} \tag{3}$$

Now combine (2) and (3) with $E[M_C] \leq 7\log_{1/\beta}(k) + 2$ from Lemma 2.

3.3 The Length of the Text until Convergence

When we use the linear time algorithm for union operations, then the total learning time is $O(T)$, so the length of the text until convergence is an important number. In the following we analyze its expected value.

Lemma 3. *Let $m \geq 1$. Then $E[\Lambda_1 \mid G_1 = m] = (E[\Lambda] - \alpha^k)/(1-\alpha^k)$.*

Theorem 8. $E[T] = E[M_C] \cdot \big(|\pi| + \hat{\alpha}^k(E[\Lambda] - 1)\big)$
$\leq (7\log_{1/\beta}(k) + 2)\big(|\pi| + \hat{\alpha}^k(E[\Lambda]-1)\big) = O\big(\hat{\alpha}^k(\log_{1/\beta}(k))E[\Lambda]\big)$.

Proof. We can write the length of text read until convergence as $T = T_1 + T_2 + \cdots + T_{M_C} + |\pi| M_C$. Exactly M_C strings of length $|\pi|$ are read; all other strings are longer and are contained in blocks in front of those minimum length strings. The ith block contains G_i strings and we denote the total length of these G_i strings by T_i (these are different T_i's than in [14]). In order to get $E[T]$ we start by computing $E[T_1]$.

$$\begin{aligned} E[T_1] &= \sum_{m=0}^{\infty} E[\Lambda_1 + \cdots + \Lambda_m \mid G_1 = m] \cdot \Pr(G_1 = m) \\ &= \sum_{m=1}^{\infty} m \cdot E[\Lambda_1 \mid G_1 = m] \cdot \Pr(G_1 = m) \\ &= \sum_{m=1}^{\infty} m \cdot \frac{E[\Lambda] - \alpha^k}{1-\alpha^k} \cdot (1-\alpha^k)^m \alpha^k \quad \text{(by Lemma 3 and (1))} \\ &= (E[\Lambda] - \alpha^k)\alpha^k \sum_{m=1}^{\infty} m(1-\alpha^k)^{m-1} \;=\; \hat{\alpha}^k E[\Lambda] - 1 \end{aligned}$$

Now it is easy to estimate $E[T]$. We use that T_1 and M_C are independent.

$$\begin{aligned} E[T] - |\pi| E[M_C] &= E[T_1 + \cdots + T_{M_C}] \\ &= \sum_{m=0}^{\infty} m \cdot E[T_1] \cdot \Pr(M_C = m) = E[M_C] \cdot E[T_1] \end{aligned}$$

and thus $E[T] = E[M_C]\big(|\pi| + \hat{\alpha}^k E[\Lambda] - 1\big)$. Finally insert the estimation of $E[M_C]$ from Lemma 2.

4 Learning Stochastically Finite with High Confidence

Definition 2. *Let D be an admissible probability distribution. PAT is called stochastically finitely learnable with high confidence from random text iff there is an IIM M such that for every $L \in PAT$ and every number $\delta \in (0, 1)$, M outputs the single hypothesis π, $L(\pi) = L$, with probability at least δ, and stops thereafter, when fed a random text according to D and L.*

Note that the learner in the definition above takes δ as additional input and that the definition immediately generalizes to arbitrary concept classes.

Next, we show how the LWA can be transformed into a stochastically finite learner that identifies all the pattern languages with high confidence provided we have a bit of prior knowledge about the class of admissible distributions that may actually be used to generate the information sequences.

Theorem 9. *Let α_*, $\beta_* \in (0, 1)$. Assume $\mathcal{D}$ to be a class of admissible probability distributions over $\mathcal{A}^+$ such that $\alpha \geq \alpha_*$, $\beta \geq \beta_*$ and $E(d)$ finite for all distributions $d \in \mathcal{D}$. Then PAT is stochastically finitely learnable with high confidence from random text for all distributions D generated by any $d \in \mathcal{D}$.*

Proof. Let $d \in \mathcal{D}$ and $\delta \in (0, 1)$ be arbitrarily fixed. Note that d induces an admissible probability distribution D. Furthermore, let $t = s_1, s_2, s_3, \ldots$ be any randomly generated text with respect to D for the target pattern language. The learner M uses the LWA as a subroutine. Additionally, it has a counter for the number of examples already seen. We exploit the fact that the LWA produces a sequence $(\tau_n)_{n \in \mathbb{N}^+}$ of hypotheses such that $|\tau_n| \geq |\tau_{n+1}|$ for all $n \in \mathbb{N}^+$.

The learner runs the LWA until for the first time C many examples have been processed, where $C = 2\log(1/(1-\delta))(7\hat{\alpha}_*^{|\tau|} + 2)\log_{1/\beta_*}(|\tau|)$ and τ is the current output made by the LWA. By Theorem 6 and 7, it follows that after processing

$$2\log(1/(1-\delta))(7\hat{\alpha}^k + 2)\log_{1/\beta}(k) \qquad \text{(A)}$$

examples the LWA converges with probability at least δ. The number C is bigger since $|\tau| \geq k$. If we are learning PAT_k instead of PAT, we can replace $|\tau|$ in (A) by k to get a better bound.

If we fix k in advance to learn only PAT_k then we arrive at a stochastically finite linear-time learner for PAT_k. This is a major improvement, since the constant depending on k grows only exponentially in k in contrast to the doubly exponentially growing constant in Kearns and Pitt's [5] algorithm.

Corollary 1. *Let $\alpha_*,\ \beta_* \in (0,1)$. Assume $\mathcal{D}$ to be a class of admissible probability distributions over $\mathcal{A}^+$ such that $\alpha \geq \alpha_*$, $\beta \geq \beta_*$ and $E(d)$ finite for all distributions $d \in \mathcal{D}$. Furthermore, let $k \in \mathbb{N}^+$ be arbitrarily fixed. Then there exists a learner M such that*

(1) *M learns PAT_k stochastically finitely with high confidence from random text for all admissible probability distributions D generated by any $d \in \mathcal{D}$, and*
(2) *The running time of M is bounded by $O\big(\hat{\alpha}^k \log(1/(1-\delta)) \log_{1/\beta}(k) E[\Lambda]\big)$.*

References

1. D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21(1):46–62, 1980.
2. D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
3. R. Daley and C.H. Smith. On the complexity of inductive inference. *Information and Control*, 69:12–40, 1986.
4. E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
5. M. Kearns and L. Pitt. A polynomial-time algorithm for learning k–variable pattern languages from examples. In R. Rivest, D. Haussler and M.K. Warmuth, editors, *Proc. 2nd Annual ACM Workshop on Computational Learning Theory* pp. 57–71, 1991, Morgan Kaufmann Publishers Inc., San Mateo.
6. Ker-I Ko, A. Marron and W.G. Tzeng. Learning string patterns and tree patterns from examples. In B.W. Porter and R.J. Mooney, editors, *Proc. 7th International Conference on Machine Learning*, pp. 384–391, 1990, Morgan-Kaufmann Publishers Inc., San Mateo.
7. S. Lange and R. Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.
8. L. Pitt. Inductive inference, DFAs and computational complexity. In K.P. Jantke, editor, *Proc. Analogical and Inductive Inference*, Lecture Notes in Artificial Intelligence 397, pp. 18–44, Berlin, 1989, Springer-Verlag.
9. P. Rossmanith and T. Zeugmann. Learning k-variable pattern languages efficiently stochastically finite on average from positive data, DOI Technical Report DOI-TR-145, Department of Informatics, Kyushu University, January 1998.
10. A. Salomaa. Patterns & Return to patterns. (The Formal Language Theory Column). EATCS Bulletin, 54:46–62 and 55:144–157, 1994.
11. R.E. Schapire. Pattern languages are not learnable. In M.A. Fulk and J. Case, editors, *Proc. 3rd Annual ACM Workshop on Computational Learning Theory*, pp. 122 – 129, 1990. Morgan Kaufmann Publishers Inc., San Mateo.
12. T. Shinohara and S. Arikawa. Pattern inference. In K. P. Jantke and S. Lange, editors, *Algorithmic Learning for Knowledge-Based Systems*, Lecture Notes in Artificial Intelligence 961, pp. 259–291, Berlin, 1995. Springer-Verlag.
13. K. Wexler and P. Culicover. *Formal Principles of Language Acquisition.* MIT Press, Cambridge, MA, 1980.
14. T. Zeugmann. Lange and Wiehagen's pattern learning algorithm: An average-case analysis with respect to its total learning time. *Annals of Mathematics and Artificial Intelligence*, 23:1-2, 117–145, 1998.