# Automatic Learning from Repetitive Texts

**Rupert Hölzl** R@HOELZL.FR
*Institute for Theoretical Computer Science, Mathematics, and Operations Research*
*Bundeswehr University Munich*
*Werner-Heisenberg-Weg 39, 85579 Neubiberg, Germany*

**Sanjay Jain** SANJAY@COMP.NUS.EDU.SG
*School of Computing*
*National University of Singapore*
*Singapore 117417, Republic of Singapore*

**Philipp Schlicht** SCHLICHT@MATH.UNI-BONN.DE
*Mathematisches Institut*
*Universität Bonn*
*Endenicher Allee 60, 53115 Bonn, Germany*

**Karen Seidel** KAREN.SEIDEL@HPI.DE
*Research group algorithm engineering*
*Hasso-Plattner-Institut*
*Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany*

**Frank Stephan** FSTEPHAN@COMP.NUS.EDU.SG
*Department of Mathematics*
*National University of Singapore*
*Singapore 119076, Republic of Singapore*

## Abstract

We study the connections between the learnability of automatic families of languages and the types of text used to present them to a learner. More precisely, we study how restrictions on the number of times that a correct datum appears in a text influence what classes of languages are automatically learnable. We show that an automatic family of languages is automatically learnable from fat text iff it is automatically learnable from thick text iff it is verifiable from balanced text iff it satisfies Angluin's tell-tale condition. Furthermore, many automatic families are automatically learnable from exponential text. We also study the relationship between automatic learnability and verifiability and show that all automatic families are automatically partially verifiable from exponential text and automatically learnable from thick text.

**Keywords:** Inductive inference, automatic family, automatic learner, fat text, one-one text, balanced text.

## 1. Introduction

Gold (1967) pioneered the field of inductive inference by proposing the model of learning in the limit. Subsequently, this model and many variations of it were studied by numerous authors, for example, Ambos-Spies, Badaev and Goncharov (2011); Angluin (1980a); Beros (2009); Blum and Blum (1975); Fernau (2003); Geilke and Zilles (2011); Grieser (2008);

Head, Kobayashi and Yokomori (1998); Heinz, Kasprzik and Kötzing (2012); Hölzl, Jain and Stephan (2016); Hutter and Poland (2004); Jain, Kinber and Wiehagen (2000); Jain, Osherson, Royer and Sharma (1999); Jantke (1991); Minicozzi (1976); Mukouchi and Arikawa (1995); Osherson, Stob and Weinstein (1982, 1986); Lange, Zeugmann and Zilles (2008); Luo and Schulte (2006); Schäfer-Richter (1984); Stephan and Ventsov (2001); Wexler and Culicover (1980); Wiehagen and Zeugmann (1994).

One central branch of inductive inference studies the learning of languages from positive data in the limit. In this model, all elements of a target language $L$ are presented to the learner, one by one, in arbitrary order, and possibly with repetitions (such an input is called a text for the language $L$). While processing more and more of this information, the learner makes and refines conjectures about the language it is shown. If the sequence of conjectures produced this way converges to an index for $L$, then the learner is said to learn the target language from the given text. If the learner learns $L$ from all texts for $L$, then it is said to learn the language $L$. If it learns all $L$ in a class $\mathcal{L}$ of languages, then it is said to learn $\mathcal{L}$.

Pitt (1989) used a delaying technique to construct, for learnable families, polynomial time learners, that is, learners which use only polynomial time to update their conjectures. Case and Kötzing (2009) provide evidence that some kind of delaying technique works for many inference criteria and therefore one can have learners with low time complexity. Subsequent research therefore aimed for learning models which are of low complexity with respect to criteria other than time so that similar delaying techniques cannot achieve full learnability.

As such models cannot be found within the realm of standard complexity classes, one has to turn to computationally weak models of learning. One such model stems from automata theory where it is required that both the class to be learnt as well as the learner can be represented using automatic structures. Hodgson (1982) and later Khoussainov and Nerode (1995); Blumensath (1999); Blumensath and Grädel (2000) pioneered the field of automatic structures, and Rubin (2004, 2008) provides an overview of the field. Automatic structures have interesting closure properties. For example, suppose a first-order formula defining a function or relation is given which uses given automatic relations and functions in an automatic structure. Then this new function or relation is also automatic, and an automaton computing it can be obtained effectively from automata for the functions and relations used in the first-order formula giving the definition of the function. This effective closure property also implies that the first-order theory of automatic structures is decidable.

Angluin (1980a) introduced *indexed families* as collections of sets $L_e$ such that the mapping $e, x \mapsto L_e(x)$ is uniformly recursive, where the set $L_e$ is identified with its characteristic function. Using the notion of automatic structures it is possible to introduce an analogue notion in the context of automata theory by requiring that the mapping $e, x \mapsto L_e(x)$ is automatic where the $e$ are drawn from a regular set $E$, the so-called *index set*. Jain, Ong, Pu and Stephan (2012) give an introduction to the notion of automatic families and their usage in learning theory. The learnability of all regular sets or certain classes of regular sets has been studied for a long time, for example, by Angluin (1982, 1987); Kinber (2010); Kearns and Pitt (1989); Pitt (1989). Furthermore, there is related work by Angluin (1980b); Lange and Wiehagen (1991); Shinohara and Arimura (2000) on the learnability of pattern languages.

Automatic learners are an example of learners which have constraints on how much information about the past data they can memorise. More precisely, whenever a limited memory learner processes a datum, it reads out some long term memory and then computes, based on this long term memory and the current datum, an updated content of the long term memory and a new conjecture. Freivalds, Kinber and Smith (1995) laid the foundations of this model and Kinber and Stephan (1995) transferred it from function learning to language learning from positive data. Jain, Luo and Stephan (2012) transferred this model to automata theory by defining that the content of the long term memory is just a word over some alphabet chosen ahead of time and by requiring that the two functions which compute the step-wise updates of the memory and issue the conjecture are automatic functions. Case, Jain, Le, Ong, Semukhin and Stephan (2011); Case, Jain, Seah and Stephan (2013); Jia (2013) extended these studies.

Automatic learners have a severe limitation as they are not able to memorise all data they see due to their computational limitations (except in case where the alphabet for the data is unary). Already Jain, Luo and Stephan (2012) observed that fat texts – where every datum appears infinitely often – enable overcoming these memory limitations in the sense that whenever there is a recursive learner for an automatic family, there is also an automatic learner which learns all languages in the family from fat text. Jia (2013) studied other ways to overcome this memory limitation. In particular he showed that the family of all co-singleton sets (that is, sets which miss out exactly one element from $\Sigma^*$) becomes learnable when the text is a one-one text, that is, a text in which each element of the set to be learnt appears exactly once (where padding the text using infinitely many pause symbols is permitted). This finding is the starting point for the current research which attempts a systematic study of various types of texts and the extent to which these texts support the learnability of automatic families using automatic learners. Note that this specific topic is really bound to the requirement that the learners are automatic, as all the types of texts considered in this article have, in the case of recursive learners without any restrictions, equivalent learning power as the various types of text can be translated into each other in an effective way.

The present work extends the study of Jia (2013) by considering various forms of text which generalise the previously considered types. First balanced texts are considered, that is, texts in which every datum appears exactly $k$ times for some $k \in \{1, 2, \ldots, \infty\}$. These texts are a natural common generalisation of both one-one texts and fat texts and they share many learnability properties with these two. We also consider texts in which the number of occurrences of a word grows quickly with the position of the word in the length-lexicographic order. A "thick text" is a type of text where the amount of occurrences of longer and longer words approaches $\infty$ sufficiently fast but in which, in contrast to fat texts, for each word there is still only a finite number of occurrences in the text. So such a text is similiar to a fat text and it will turn out that a class is automatically learnable from thick text iff it is automatically learnable from fat text. Furthermore, in addition to the task of learning, the task of verification is studied. In verification problems, the learner is given an index $e$ and a text for some language $L$ in the target class. Then, the learner has to converge to the answer of the decision problem of whether $e$ is an index for $L$.

The main results of this article concern automatic families. For these, the following conditions turn out to be equivalent: (a) A family is recursively learnable from arbitrary text; (b) The family is verifiable from balanced text; (c) The family is verifiable from thick text (see Definition 2.3); (d) The family is automatically learnable from fat text; (e) The family is automatically learnable from thick text. Jain, Luo and Stephan (2012) have already observed the equivalence of (a) and (d); Jia (2013) has furthermore shown that certain families which are not automatically learnable from arbitrary text are nevertheless automatically learnable from one-one text and these examples generalise to learnability from balanced text.

Gold (1967) already discovered that, in the recursion-theoretic setting, the class of all r.e. languages is learnable from primitive recursive text. This suggests to ask the question whether a similar result holds for inductive inference in the automatic setting. To study it, one could introduce the notion of an automatic text which is given by an automatic function from some automatic presentation of the natural numbers to the set of words and the pause symbol; such texts are always primitive recursive texts. However, since they are closed under finite variants, one can show as in Theorem 3.5 and Example 5.1 that such texts are too complicated for learning with automatic learners in the sense that there are automatic families which are learnable from one-one text using such learners but not from automatic text. Since this implies that texts of the lowest possible computational complexity can already be insufficient for automatic learnability, the emphasis in the present work is on the number of times a datum appears in a text instead.

## 2. Definitions and notations

Let $\mathbb{N}$ denote the set of natural numbers $\{0, 1, 2, \ldots\}$. Let $\Sigma$ denote the alphabet set used for languages. For a set $A$, we write $A^+$ for finite words of length at least 1 of elements of $A$ and $A^*$ for $A^+ \cup \{\varepsilon\}$, where $\varepsilon$ is the word of length 0. By $|x|$ we denote the length of the word $x$. We write $x <_{\text{lex}} y$ to express that $x$ is lexicographically smaller than $y$ and $x <_{\text{ll}} y$ to express that $x$ appears before $y$ in the length-lexicographic ordering of words, that is, either $|x| < |y|$ holds or it holds that both $|x| = |y|$ and $x <_{\text{lex}} y$.

We use the additional symbol $\Diamond$ to pad words when we need multiple words to have the same length. Let the convolution $\text{conv}(u, v)$ of two words $u, v \in \Sigma^*$ be defined as follows. Suppose $u = u(0)u(1)\ldots u(n-1)$ and $v = v(0)v(1)\ldots v(m-1)$ are two words of length $n$ and $m$, respectively. Let $\text{conv}(u, v)$ be $w(0)w(1)\ldots w(\max(m, n) - 1)$, where

$$w(i) = \begin{cases} \binom{u(i)}{v(i)} & \text{if } i < \min(m, n) \\ \binom{u(i)}{\Diamond} & \text{if } m \leq i < n \\ \binom{\Diamond}{v(i)} & \text{if } n \leq i < m \end{cases}$$

A relation $R = \{(u_1, u_2, \ldots, u_n) \colon u_i \in \Sigma^*\}$ is said to be *automatic* if the set

$$\{\text{conv}(u_1, u_2, \ldots, u_n) \colon (u_1, u_2, \ldots, u_n) \in R\}$$

is regular. Similarly, a function $f$ from $(\Sigma^*)^m$ to $(\Sigma^*)^n$ is said to be automatic if the set

$$\{\text{conv}(u_1, u_2, \ldots, u_m, w_1, w_2, \ldots, w_n) \colon f(u_1, u_2, \ldots, u_m) = (w_1, w_2, \ldots, w_n)\}$$

is regular.

**Definition 2.1.** A set of languages $\mathcal{L} = \{L_e \colon e \in I\}$ is called an *automatic family of languages* if both the *index set* $I$ of $\mathcal{L}$ and $\{\mathrm{conv}(e, x) \colon x \in L_e\}$ are regular.

We now give a short summary of notions for inductive inference as first considered by Gold (1967). We will be mainly concentrating on automatic learners. More detailed discussion about learners and learning criteria is provided by Angluin (1980a); Blum and Blum (1975); Gold (1967); Jain, Osherson, Royer and Sharma (1999); Osherson, Stob and Weinstein (1986). The description below is a modification of the model considered by Gold (1967), to adapt it to automatic learners as first considered by Jain, Luo and Stephan (2012).

A *text* is a mapping from $\mathbb{N}$ to $\Sigma^* \cup \{\#\}$. The content of a text $T$, denoted $\mathrm{cnt}(T)$, is $\{T(i) \colon i \in \mathbb{N} \wedge T(i) \neq \#\}$. $T$ is *a text for* a language $L$ if $\mathrm{cnt}(T) = L$. A *finite sequence* is a finite initial segment of a text, that is, a finite sequence of words. The length of a finite sequence $\sigma$, denoted $|\sigma|$, is the number of elements in its domain. The content of a sequence $\sigma$ is defined as $\{\sigma(i) \colon i < |\sigma|\} - \{\#\}$ and denoted by $\mathrm{cnt}(\sigma)$.

Intuitively speaking, a learner $M$ is presented a text of a language $L$ as input, one word at a time. At any given time, the learner $M$ has some information stored in its working memory which depends on the words it has seen previously. In dependence of its working memory and the word which is currently presented to $M$, $M$ modifies its working memory and forms a new conjecture about $L$. The sequence of conjectures formed this way over time can be interpreted as how $M$'s understanding about $L$ evolves when acquiring more and more information. If the sequence does indeed converge to an index of the language $L$, then $M$ is said to learn $L$.

In this article we will be mainly concerned about automatic learners.

**Definition 2.2** (Gold (1967); Osherson, Stob and Weinstein (1986); Jain, Luo and Stephan (2012))**.** A learner $M$ learning an automatic family $\mathcal{L} = \{L_e \colon e \in I\}$ using a conjecture space $\mathcal{H} = \{H_e \colon e \in J\}$ is defined as follows. In this article, except where it is explictly said to be otherwise, $\mathcal{H}$ and $\mathcal{L}$ will be equal.

$M$ has an initial memory consisting of the empty word and in each cycle, the learner reads a datum, updates its memory and outputs a conjecture from $J$. Formally, the memory is a word over $M$'s memory alphabet $\Gamma$ and the update in each cycle is realised by a recursive function mapping the old memory $\mathrm{mem}_n$ and the current datum $T(n)$ to the new memory $\mathrm{mem}_{n+1}$ and the conjecture $e_n$; if this mapping is even automatic then $M$ is called an *automatic learner*.

$M$ *learns* a language $L$ from text $T$, if the sequence of the $e_n$, as defined above, converges to an $e$ such that $H_e = L$.

$M$ *partially learns* a language $L$ from text $T$, if for the sequence of the $e_n$ as defined above, there exists an $e$ such that $H_e = L$ and (i) there exist infinitely many $n$ such that $e_n = e$, and (ii) for all $e' \neq e$, there exist only finitely many $n$ such that $e_n = e'$.

$M$ *automatically learns* $L$ from text $T$ if it learns $L$ from text $T$ and is automatic. $M$ *automatically partially learns* $L$ from text $T$ if it partially learns $L$ from text $T$ and is automatic.

$M$ *(automatically) learns* a language $L$ if for all texts $T$ for $L$, $M$ (automatically) learns $L$ from $T$. $M$ *(automatically) learns* $\mathcal{L}$ if it (automatically) learns each $L_e \in \mathcal{L}$. We say that $\mathcal{L}$ is *(automatically) learnable* if there is an (automatic) learner that learns $\mathcal{L}$.

(Automatic) partial learning is similarly defined.

In order to define the notion of the complement of a set of words we need to fix a *base set* inside which the complement is taken. Without explicitly mentioning it in the remainder of the article we fix $\Sigma^*$ as the base set where $\Sigma$ is the smallest alphabet with $\bigcup_{e \in I} L_e \subseteq \Sigma^*$. With the base set fixed we can also talk about the position of a word inside the base set by letting ll be an order-isomorphism from the base set with length-lexicographical ordering to $\mathbb{N}$ with the usual ordering $<$.

Furthermore, we let $\mathrm{succ}_e(x)$ denote the length-lexicographically least element of the set $\{y \in L_e : x <_{\mathrm{ll}} y\}$; if this is empty then $\mathrm{succ}_e(x)$ is just the successor of $x$ with respect to the base set.

In this article, we restrict our study to the learnability from texts with a specific number of occurrences for each word. Osherson, Stob and Weinstein (1986) introduced the notion of a *fat text* where each word appears infinitely often. We extend their study to the following notions.

**Definition 2.3.** Suppose that $f : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ is a function. A text $T$ is called an *$f$-text* for a language $L$ if every $x \in L$ appears in $T$ exactly $f(\mathrm{ll}(x))$ times.

If $f$ is the constant function with value $k \in \mathbb{N} \cup \{\infty\}$, we call $T$ a *$k$-text.* To maintain consistency with existing literature, 1-texts will be referred to as one-one-texts. Similarly, $\infty$-texts will be called fat texts.

Moreover, a text that is a $k$-text for some $k \in \mathbb{N} \cup \{\infty\}$ is called a *balanced text.* A $2^{\mathrm{ll}(x)}$-text is called an *exponential text.* An $f$-text with all values of $f$ being nonzero natural numbers and $f(\mathrm{ll}(w)+1) > 6 \cdot |\mathrm{succ}(w)| \cdot f(\mathrm{ll}(w))$ for all $w$ is called a *thick text.*

If $\mathcal{R}$ is one of these text types then we say that a learner $M$ *(automatically) learns* a language $L$ *from* $\mathcal{R}$ if for all texts $T \in \mathcal{R}$ for $L$, $M$ (automatically) learns $L$ from $T$. We say that $\mathcal{L}$ is *(automatically) learnable from* $\mathcal{R}$ if there is an (automatic) learner that learns each $L \in \mathcal{L}$ from $\mathcal{R}$.

Intuitively, thick texts are texts in which each element of the text appears "many more" times than elements length-lexicographically smaller than it.

The following important condition due to Angluin (1980a) will be used several times below.

**Definition 2.4.** A family $\mathcal{L} = \{L_e : e \in I\}$ satisfies the *tell-tale condition* if and only if for all $e \in I$ there are finite sets $D_e \subseteq L_e$ such that for all $d \in I$ with $D_e \subseteq L_d \subseteq L_e$, we have $L_d = L_e$. We call such a set $D_e$ *tell-tale set for the language* $L_e$.

For learnable automatic families, the *tell-tale cut-off word* for $L_e$ is the length-lexicographically first word $c_e$ for which $\{x \in L_e : x <_{\mathrm{ll}} c_e\}$ is a tell-tale set for $L_e$. The mapping $e \mapsto c_e$ is automatic by a first-order definability argument.

## 3. Balanced text

In this section we extend the following characterisation of Jain, Luo and Stephan (2012).

**Theorem 3.1 (Jain, Luo and Stephan (2012), Theorem 28).** *An automatic family is automatically learnable from fat text if and only it satisfies the tell-tale condition.*

**Definition 3.2.** A *verifier* $M$ for $\mathcal{L} = \{L_e : e \in I\}$ is defined similarly to an automatic learner except that (a) as input it first receives an index $e \in I$ and then some text and (b) its possible conjectures are *yes* and *no*.

$M$ is said to *partially verify* an input $e, T$ when the conjectures of $M$ are infinitely often *yes* iff $T$ is a text for $L_e$; $M$ is said to *verify* an input $e, T$ iff it partially verifies $e, T$ and, furthermore, the conjectures of $M$ converge either to *yes* or to *no*.

*$M$ (partially) verifies $\mathcal{L}$* if $M$ (partially) verifies every input $e, T$ where $e$ is an index of a language in $\mathcal{L}$ and $T$ is a text for a language in $\mathcal{L}$.

Let $\mathcal{R}$ be a class of texts. *$M$ (partially) verifies $\mathcal{L}$ from $\mathcal{R}$* if $M$ (partially) verifies every input $e, T$ where $e$ is an index of a language in $\mathcal{L}$ and $T$ is a text in $\mathcal{R}$ for a language in $\mathcal{L}$. We say that $\mathcal{L}$ is *(partially) verifiable from $\mathcal{R}$* if there is a learner that (partially) verifies $\mathcal{L}$ from $\mathcal{R}$.

**Example 3.3.** Even without any restrictions on texts, verifiers can be more powerful than automatic learners. For example, Jain, Luo and Stephan (2012) showed that the family of all co-singleton sets is not automatically learnable; however, it is verifiable as upon receipt of the index $x$ of $L_x = \{y \in \Sigma^* : y \neq x\}$, the verifier just conjectures *yes* until $x$ appears in the text and in that case, the verifier makes a mind change to *no* and keeps that conjecture forever. Jia (2013) showed that this family is automatically learnable from one-one text.

While it is unknown whether all automatic families satisfying the tell-tale condition are automatically learnable from balanced text or from one-one text, one can show that they are verifiable from balanced text; recall that verifiers are by definition automatic.

**Theorem 3.4.** *An automatic family is verifiable from balanced text if and only if it satisfies the tell-tale condition.*

**Proof.** ($\Rightarrow$): Suppose that $\mathcal{L} = \{L_e : e \in I\}$ is an given automatic family which satisfies the tell-tale condition. Recall that the mapping $e \mapsto c_e$ is automatic, where $c_e$ is the tell-tale cut-off word which is the length-lexicographically first word $w$ satisfying that $D_e = \{x \in L_e : x <_{\text{ll}} w\}$ is a tell-tale set for $L_e$.

Suppose the verifier is given an index $e$ and a balanced text $T$ for $L_d$ as input. It needs to verify whether $L_e = L_d$. Without loss of generality assume that $L_e \neq \emptyset$, as otherwise the verification is easy. Intuitively, if the verifier ever sees an element in the text outside $L_e$, then it immediately knows that $L_d \neq L_e$. Otherwise, the verifier runs two subalgorithms in parallel. One algorithm works correctly if the input is a $k$-text for $L_e$ where $1 \leq k < \infty$, while the other works correctly if the input is a fat text. The main aim of the following construction is to combine both algorithms so that they do not interfere with each other while ensuring that the resulting algorithm is automatic.

**Algorithm.** The verifier keeps track of the following conditions and numbers, which can be stored in the memory using convolution:

- Whether any element not in $L_e$ has appeared in the input text;
- The number $k$ denoting the number of occurrences so far of the length-lexicographically least word in $L_e$ in the input text;

- A number $\ell$ denoting the number of occurrences so far of elements strictly below $c_e$ in the input text;
- A word $x$, which starts as being the length-lexicographically least word in $D_e$; this word $x$ is updated to $\mathrm{succ}_e(x)$, whenever it appears in the input text;
- A number $m$, which denotes the value of $\ell$ had when $x$ got updated the last time, that is, got its current value;
- A check whether $\ell/k = |D_e|$; note that this check is not automatic, its computations are done over several cycles of reading the input words with one step per cycle and the check restarts every time $\ell$ or $k$ is updated.

Now the verifier, after reading any input, outputs *no* if it has seen some word not in $L_e$ in the past and remains so. If this has not happened and $x \geq_{\mathrm{ll}} c_e$, then the verifier outputs *yes*, as it has clearly seen all the elements of $D_e$ in the input text and remains so. Otherwise, the verifier checks whether $\ell/k = |D_e|$ and $k \leq m$. If the computation of $\ell/k = |D_e|$ has terminated and is still valid, that is, $\ell, k$ did not change after the start of the computation, and also the condition $k \leq m$ is true, then the verifier outputs *yes*, else it outputs *no*.

**Verification.** To show that this algorithm succeeds, first assume that $L_d = L_e$. In this case the verifier never sees an element outside $L_e$. If the text is fat, then eventually the value of $x$ will be length-lexicographically larger than all elements of $D_e$ and thus the verifier will almost always output *yes*. If the input is an $n$-text for some $n \in \mathbb{N}$, the above may also happen. So assume otherwise, that is, the value of $x$ stabilises on some element of $D_e$. Note that eventually the value of $k$ converges to $n$ and $m \geq n$ (as otherwise a further occurrence of $x$ in the text will happen after the point when $x$ got its current value and therefore the value of $x$ will become larger). Thus, the values $\ell$ and $k = n$ will eventually converge, $\ell/k = |D_e|$ and $k \leq m$. Therefore the verifier will almost always output *yes*.

Now assume that $L_d \neq L_e$. If $L_d \not\subseteq L_e$, then the output is eventually *no* as the verifier will eventually see an input element outside $L_e$. The other case is that $L_d \subset L_e$. In that case, $D_e \not\subseteq L_d$ and $D_e$ is nonempty by the tell-tale condition. Furthermore, the value of $x$ is always an element of $D_e$. Thus, the verifier never outputs *yes* due to it having seen all the elements of $D_e$. If the input text is an $n$-text, for some $n \in \mathbb{N}$, then the value of $k$ eventually reaches $n$. Furthermore, $\ell < n \cdot |D_e|$ holds, as the input text does not contain all the elements of $D_e$. Hence the verifier outputs *no* almost always. If the input text is fat, then the value of $k$ is unbounded, whereas the value of $m$ is fixed (as it does not change after the last time $x$ is updated). Thus, $k \leq m$ almost always fails and the verifier outputs *no* almost always.

($\Leftarrow$): Let $N$ be a verifier for $\mathcal{L} = \{L_e \colon e \in I\}$ and assume that $N$ verifies $\mathcal{L}$ from $k$-text. Assume that an arbitrary text $T$ for some $L \in \mathcal{L}$ is given. We build a recursive learner $M$ which first transforms $T$ into a $k$-text $T'$ for $L$. $M$ simulates the operation $N$ on input $e, T'$ for length-lexicographically increasing values of $e \in I$ and the $n$-th conjecture of $M$ is the length-lexicographically least $e \in I$ such that $\mathrm{ll}(e) \leq n$ and the verifier outputs *yes*. If there is currently no such $e$, then $M$ abstains from a conjecture and outputs ?. It is immediate that $M$'s conjectures converge to the least $e \in I$ with $L_e = L$. Angluin (1980a) showed that $\mathcal{L}$ then satisfies the tell-tale condition. $\square$

The following result shows that the last result is close to optimal.

**Theorem 3.5.** *There is an automatic family $\mathcal{L} = \{L_e \colon e \in I\}$ as follows.*

1. *There is an automatic learner that learns $\mathcal{L}$ from balanced text.*
2. *There is no automatic verifier that verifies $\mathcal{L}$ from texts in which every word in the input language appears exactly $1$ or $2$ times.*
3. *There is no automatic verifier that verifies $\mathcal{L}$ from texts that contain every word exactly $k$ times for a $k$ with $1 \leq k \leq \infty$, with the possible exception of a single word that only needs to appear at least once.*

**Proof.** Consider the automatic family $\mathcal{L}$ given by the index set $I = \{0,1\}^+ \cup \{2\}^+$ and the languages $L_e = \{0,1\}^{|e|} \setminus \{e\}$ for $e \in I$.

The proof of the first claim is similar to the proof of Theorem 3.4. We assume that the input text is a text for some $L_e$ with $|e| = n$, where $n$ can be obtained from the first input word in $\{0,1\}^*$. The learner keeps track of the following information.

- The first datum $z$ of the input text and the number $k$ of its occurrences.
- A number $\ell$ denoting the overall number of words read so far.
- A word $x$, which starts as being the length-lexicographically least word in $L_{2^n}$. The word $x$ is updated to $\mathrm{succ}_{2^n}(x)$ when the verifier sees the word $x$ in its input. For ease of notation, we take $\mathrm{succ}_{2^n}(1^n) = 2^n$.
- The binary sum $s$ of all data observed so far, as well as the binary sum $s' = \sum_{y \in \{0,1\}^n} y$.
- The value $m$ of $\ell$ at the first time when $x$ reaches its current value (after the update of the current value of $\ell$).

The memory takes the value ? until the first datum $z$ is observed, when the values are initialised as $x = \min_{\mathrm{ll}}(\{0^n, 0^{n-1}1\} \setminus \{z\})$, $k = 1$, $\ell = 1$ and $m = 1$. The conjecture $d$ is defined as follows.

- If $x = 2^n$ then $d = x$ else the conjecture is computed in the following steps.
- If $k \leq m$ and $\ell/k$ is an even integer then $d = 2^n$.
- If $k \leq m$ and $\ell/k$ is an odd integer and $s' \cdot k = s + y \cdot k$ for some $y \in \{0,1\}^n$ then $d = y$.
- If $k > m$ then $d = x$.
- If none of these conditions holds then $d = ?$.

The conditions in the second and third bullet points need to be checked by the learner using computations that run over multiple learning steps. This allows calculating the divisions stepwise. Whenever the computations have not yet terminated, or if they have been invalidated by new data, the conditions are considered not satisfied.

To show that this algorithm has the required properties, note that for $x = 2^n$ we have $L_e = L_{2^n}$ and hence the conjecture is correct by the first condition. Thus we now assume that the eventual value of $x$ is not $2^n$.

We first assume that the input is a $k$-text for some $k \in \mathbb{N}$. Assume that every datum different from $\#$ has already been processed by the learner. If $x \neq e$ at the time when $x$ first takes its final value, then all $k$ occurrences of $x$ have already passed, $k \leq m$ and the conjecture is correct by second and third cases. Otherwise $x = e$ at the time when $x$ takes its final value and then the conjecture is correct by the second, third and fourth cases.

We now assume that the input is a fat text. Then we will eventually have $x = e$. Since $m$ is fixed from this step onwards but $k$ converges to $\infty$, we eventually have $k > m$ and hence the fourth condition implies that the conjecture $d = e$ is correct.

The proofs of the remaining claims are modifications of a proof from Jain, Luo and Stephan (2012). To prove the second claim, we consider sequences $\sigma$ containing exactly $n$ distinct words of length $n$ from $\{0,1\}^n$ in length-lexicographic order for any $n \in \mathbb{N}$. There are $\binom{2^n}{n}$ many such sequences with different content. When $n$ is sufficiently large, we have $\frac{2^n - n}{n} \geq n$ and hence $\binom{2^n}{n} = \frac{2^n \cdot (2^n-1) \cdot \ldots \cdot (2^n-n)}{1 \cdot 2 \cdot \ldots \cdot n} \geq n^n$.

If $M$ is an automatic verifier for this family, let $\mathrm{mem}_M(e,\sigma)$ denote the memory contents after reading the index $e$ and the prefix $\sigma$ of some text. If the automaton for the update function of $M$ has $c$ states, then due to the pumping lemma, the length of the memory after reading $\sigma$ is at most $(n+1)c$. If the alphabet for the memory has $b$ letters, then for any index $e$ of length at most $n$, there are $b^{(n+1)c}$ many possible values of $\mathrm{mem}_M(e,\sigma)$, thus the number of possible values of the memory after reading $n$ words of length $n$ is an exponential function.

As $n^n$ grows faster than any exponential function, one has for sufficiently large $n$ that there are two sequences $\sigma, \rho$ with $\mathrm{cnt}(\sigma) \neq \mathrm{cnt}(\rho)$, each of $\sigma$, $\rho$ containing exactly $n$ distinct words from $\{0,1\}^n$ and $\mathrm{mem}_M(e,\sigma) = \mathrm{mem}_M(e,\rho)$, where $e$ is the index $2^n$ of $\{0,1\}^n$. As the content of the two sequences are different, there is one word $x$ which occurs in only one of the sequences, say $x \in \mathrm{cnt}(\sigma) \setminus \mathrm{cnt}(\rho)$. Let $T$ be a one-one text for $L_x$. Now $\sigma T$ is a text for $\{0,1\}^n$ while $\rho T$ is a text for $L_x$. However, the verifier has to behave in the limit on both texts the same way, as its memory cannot retain whether the text begins with $\sigma$ or $\rho$. So $M$ converges to the same conjecture for both texts, contradicting the assumption that $M$ is a verifier for the family $\mathcal{L}$.

The proof of the last claim is obtained from the previous argument by replacing the one-one text $T$ with a fat text for $L_x$. $\square$

The previous result also answers a question posed by one of the anonymous referees of ALT 2017 who asked whether texts where each word occurs at most $k$ times are as powerful as texts where each word occurs exactly $k$ times; the answer is negative for all $k \geq 2$. Furthermore, the referee posed the question of what happens if texts are considered in which every word of the language to be learnt occurs at least $k$ times. This notion is equivalent to the learnability from texts without any restriction for any $k \geq 1$.

**Proposition 3.6.** *If a class can be learnt by an automatic learner from all texts in which every datum appears at least k times, then the class can also be learnt by another automatic learner from all texts without any restrictions.*

**Proof.** Consider the case where an automatic learner $M$ learns from all texts in which every word occur at least $k$ times. Now convert $M$ into a new learner $N$ which simulates $M$ in such a way that $M$ immediately processes each received datum $k$ times. For instance, if $k = 3$ and $M$ on memory $y$ and datum $x$ updates the memory to $mem(x, y)$ and outputs the hypothesis $hyp(x, y)$ then consider $N$ which updates the memory to $mem(x, mem(x, mem(x, y)))$ and outputs the hypothesis $hyp(x, mem(x, mem(x, y)))$. This learner $N$ behaves on a text $T$ as $M$ would behave on the text $T(0)\ T(0)\ T(0)\ T(1)\ T(1)\ T(1)\ T(2)\ T(2)\ T(2)\ \ldots$ and thus when $M$ learns on the so translated text then $N$ learns on $T$. Thus whatever $M$ learns from texts in which every word occurs at least $k$ times, $N$ learns from all texts. Furthermore, if $M$ is automatic, so is $N$, as automatic functions are closed under composition. □

All automatic families are partially verifiable from balanced text. The proof uses some ideas from the proof of Theorem 3.4, but also requires some new tricks since without the tell-tale condition the quantities $c_e, D_e$ as used in the proof of Theorem 3.4 do not exist.

**Theorem 3.7.** *Every automatic family is partially verifiable from balanced text.*

**Proof.** Recall that, given an automatic family $\mathcal{L} = \{L_e \colon e \in I\}$, a partial verifier needs to infinitely often output *yes* on an input consisting of $e \in I$ and a balanced text for $L_e$, and it must almost always output *no* on an input consisting of an $e \in I$ and a balanced text for some $L_d \neq L_e$ with $d \in I$. On all other inputs, the learner can show arbitrary behaviour.

Assume without loss of generality that $L_e \neq \emptyset$, as $\emptyset$ is easy to verify. Intuitively, the idea of the algorithm is the following. If ever an element not in $L_e$ is observed in the input text, then clearly the input text is not for $L_e$. To verify whether the input is properly contained in $L_e$ or not we again use an algorithm composed of two subalgorithms similar in spirit to those appearing in the proof of Theorem 3.4.

At the beginning of the algorithm, $b, b'$ are initialised as $\varepsilon$ and $k, \ell, \ell', m$ are initialised as 0. Furthermore, $y$ is initialised as the length-lexicographically least element of $L_e$ (unless $L_e$ is empty in which case the value of $y$ does not matter). For each newly observed word $x$ in the text the learner acts as follows; we only explicitly mention it when the verifier outputs *yes* and assume implicitly that at all other times it outputs *no*.

1. If $x \notin L_e \cup \{\#\}$ then the verifier stores in its memory that it has a seen a non-element of $L_e$. If this is not the case, and has never happened so far, proceed with step 2.

2. If $L_e = \emptyset$ and $k = 0$ then the verifier outputs *yes* and terminates this round; if $L_e \neq \emptyset$ and $x = \#$ the algorithm goes to step 5; if $L_e \neq \emptyset$ and $x \neq \#$ then the algorithm continues in step 3; the remaining case is already captured in Step 1.

3. If $x \leq_{\mathrm{ll}} b$ then $\ell = \ell + 1$ else $\ell' = \ell' + 1$;
   let $b' = \max_{\mathrm{ll}}\{b', x\}$;
   if $x = \min_{\mathrm{ll}}(L_e)$ then $k = k + 1$.

4. If $x = y$ or $y$ is length-lexicographically larger than all elements of $L_e$,
   then conjecture *yes*, let $m = \ell + \ell'$ and let $y = \mathrm{succ}_e(y)$.

5. Carry out one step of the computation of $n = |\{x \in L_e \colon e \leq_{\mathrm{ll}} b\}|$ and the product $n \cdot k$ (these computations run over several cycles).

If these computations have terminated without any change of $b$, $k$ and $\ell$ during the computations and $\ell = k \cdot n$ and $1 \leq k \leq m$,
then let $b' = \mathrm{succ}_e(b')$, let $b = b'$, let $\ell = \ell + \ell'$, let $\ell' = 0$ and output *yes*.

The following case-distinction establishes the correctness of the algorithm.

1. If the input text contains some $x \notin L_e$ then the verifier only finitely often outputs *yes* by Step 1 of the algorithm. So assume that the text is for a language contained in $L_e$.

2. If $L_e = \emptyset$ and the text is for $\emptyset$, then Step 2 ensure that *yes* is infinitely often conjectured, since $k$ remains 0 forever. On the other hand, when the text is for a nonempty set, then as needed the verifier outputs *yes* only finitely often by the previous case.

3. If the text is for $L_e$ and contains every element of $L_e$ exactly $k$ times for some $k \in \mathbb{N}$ then the verifier outputs *yes* infinitely often: The reason is that for each current value of $b$, the learner will eventually have computed the number $n = |\{x \in L_e : x \leq b\}|$. Furthermore, for each value of $y$, if $m$ is initialised below the final value of $k$, then $y$ will occur in the text at least once after this initialisation and therefore $y$ will be updated to another value; hence whenever $y$ stabilises on some value, then the $m$ for this value of $y$ will be initialised as at least the final value of $k$. Now for each bound $b$ it will eventually happen that all elements of $L_e$ up to $b$ have appeared $k$ times in the text and no other elements below $b$ have appeared. Then the equality $n \cdot k = \ell$ will eventually hold and $b$ will be updated to $b'$ and *yes* will be output.

4. If the text is a $k$-text for a $k \in \mathbb{N}$ and if the text is for a strict subset of $L_e$ then the verifier outputs a *yes* only finitely often which means that it converges to a *no*: Assume that enough of the text has been seen that $k$ has reached its final value and that $b$ bounds an element of $L_e$ which is not in the text. From now onwards the equality $k \cdot n = \ell$ will never hold again and therefore only finitely often a *yes* is output by Step 3. Furthermore, $y$ will never move beyond the gap of the element in $L_e$ which is not in the text and so there will be only finitely many *yes*'s output by Step 4. Hence the verifier converges to a *no*.

5. If the text is a fat text for $L_e$ then the verifier will output a *yes* infinitely often by Step 4: The reason is that for all elements $y$ of $L_e$ and each occurrence of $y$ in the text, there will be a further occurrence of $y$ later in the text, since the text is fat. When this further occurrence is found, $y$ will advance to the next element of $L_e$ and *yes* will be output. For texts of finite $L_e$ and $y$ has advanced through all elements of $L_e$, *yes* will also be infinitely often output as this case is captured by Step 4.

6. If the text is a fat text for a proper subset of $L_e$, then the verifier will only finitely often output *yes* and therefore its conjecture converges to *no*: The reason is that there is a minimal $y$ which appears in $L_e$, but not in the text. If the verifier reaches $y$ and reinitialises the value $m$ then $y$ will remain unchanged, as it does not occur in the text, while $k$ will become larger and larger until $k > m$. From then onwards, the verifier will no longer output a *yes* as in Step 4 the $y$ does no longer advance upward and in Step 5 the condition $k \leq m$ is no longer satisfied.

In summary, the verifier converges in all cases to the correct answer. □

## 4. Exponential and thick text

We now consider texts where the number of occurrences of an individual word is fast-growing with the position of that word in the length-lexicographical order. Exponential texts and thick texts are nearly fat texts, however, in these texts each word occurs only finitely often. The main idea for learning exponential text is to do some form of counting which allows constructing in the limit an informant for the input language in the memory. However, this construction only converges in the limit and therefore a learner operating like this cannot learn all learnable automatic families using it, but only those which are learnable finitely from informant.

Here, we say that $T$ is an informant for a language $L$, if $T(\mathrm{ll}(x)) = L_e(x)$ for all $x$ in the base set, that is, $T$ is a sequence of bits. We say that a family of languages is *finitely learnable from informant*, if there is a learner which given any informant for $L$ outputs only one distinct conjecture besides ? and this conjecture is an index for the language $L$. For an automatic family $\{L_e \colon e \in I\}$ finite learnability from informant is equivalent to

$$\forall e \in I\, \exists b_e\, \forall d \in I\, [\forall x <_{\mathrm{ll}} b_e\, [L_d(x) = L_e(x)] \to \forall x\, [L_d(x) = L_e(x)]]$$

where the mapping $e \mapsto b_e$ is automatic as it is first-order definable.

The idea for learning from thick text is to buffer a reduced copy of the text in the working memory and then to simulate a recursive learner using this reduced copy as input text. This strategy works for all learnable families. The construction can also easily be modified to produce an informant in the limit which never contains false positive information.

**Theorem 4.1.** *Every automatic family is automatically partially learnable from exponential text. Furthermore, if an automatic family $\{L_e \colon e \in I\}$ is finitely learnable from informant then the family is also automatically learnable from exponential text.*

**Proof.** The learner maintains a natural number $a$ which is the sum of $2^{|x|}$ for all words $x$ observed so far. Note that the $(\mathrm{ll}(x) + |x|)$-th bit of $a$ is $L(x)$ after the learner has seen all words $y$ with $|y| \leq \mathrm{ll}(x) + |x| + 1$ in the $2^{\mathrm{ll}(x)}$-text; all subsequent words $y$ will be represented in $a$ as multiples of $2^{\mathrm{ll}(x)+|x|+1}$ and do not influence the lower bits of the binary representation of $a$. Consequently, the number $a$ converges to an infinite binary sequence.

In parallel to building up $a$, the learner cycles through all $e$ so that each $e$ is accessed infinitely often and maintains for each $e$ a variable $p_e$ which stores that at the last recent checking the bits of the stored word $a$ at $\mathrm{ll}(x) + |x|$ coincides with $L_e(x)$ for all $x$ up to $p_e$. Now the learner carries out, distributed over multiple but finitely many learning cycles, a new computation of $p_e$. When this new value is larger than the old then the learner outputs $e$ one more time. Now one can see that an index $e$ is conjectured infinitely often if the bits of $a$, in the limit, code $L_e$; furthermore, if the bit at $x$ is wrong then the learner will eventually always see a wrong bit for $x$ and $p_e$ will never go beyond $x$ but converge to the smallest $y$ where $a(\mathrm{ll}(y) + |y|) \neq L_e(y)$. Thus $e$ will only be conjectured finitely often. Furthermore, we can assume that $I$ is a one-one indexing; if not, replace $I$ by a suitable subset, as shown by Jain, Ong, Pu and Stephan (2012). Therefore there will only be one correct index that is conjectured infinitely often while all other indices are conjectured only finitely often. Thus the so described automatic learner is a partial learner.

For the last result, note that when the family is finitely learnable from informant, then there is an automatic mapping $e \mapsto b_e$ such that whenever $L_d \neq L_e$ then there is an $x \leq b_e$ with $L_d(x) \neq L_e(x)$. This property is exploited by making the learner conjecture $e$ as long as $p_e >_{ll} b_e$. As each bit of $a$ converges in the limit, the learner will only finitely often oscillate between conjecturing $e$ and conjecturing something else. □

**Theorem 4.2.** *Every automatic family $\{L_e \colon e \in I\}$ is automatically partially learnable from thick text. Furthermore, the family is automatically learnable from thick text iff it satisfies the tell-tale condition.*

**Proof.** The main idea of the proof is to construct an automatic learner $N$ which simulates a recursive learner $M$ and also manages its inputs and outputs. $N$ tries to write the words into a buffer which then serves as input for $M$; however, due to the slow down caused by the fact that concatenation operations require more than one cycle, there is a loss of data. Now if a word appears sufficiently more often than all length-lexicographically lower words (which will be copied into the buffer with priority), then it will eventually be copied into the buffer itself. Thick texts have this property for all words.

More precisely, the simulated recursive learner $M$ is given as a three tape Turing machine having a one-way infinite read-only input tape and write-only output tape and which uses the work tape also as an internal memory for everything it wants to remember. The Turing machine $M$ is programmed such that it reads the buffer representing the text from the back to the front and $N$ inserts new data-items at the front end of the buffer. The new learner $N$ maintains as its memory a convolution of a copy of the current or a very recent datum $x$ to be added to the buffer; a word $u$ representing the contents of the buffer (reading tape) of $M$; a word $w$ representing the output tape of $M$; a word $v$ representing the content of the work tape of $M$, the internal states of $M$ and other similar data such as a special symbol indicating whether the current output $w$ is complete or not; the current conjecture $e$ of $N$ which will be repeated until a new conjecture of $M$ is available in $w$. As the concatenation of automatic functions is automatic, we can describe the update function of the learner $N$ in multiple separate steps which are in actuality carried out by a single automatic update function. The algorithm is as follows for every new datum $y$:

1. If $x$ is void or if $y <_{ll} x$ then replace $x$ by $y$.

2. Shift $u$ by inserting two blanks at the start.

3. If there are $|x| + 1$ or more blanks at the beginning of $u$ then replace the first $|x|$ blanks by $x$ and then void $x$.

4. Simulate one step of the Turing machine $M$; if $M$ requests to read a new word from the input, read the last symbol of the buffer $u$ and remove it from $u$; in case that $u$ consists only of blanks, append a pause symbol # prior to reading at the end of $u$ which is then read.

5. If after the simulation step a valid conjecture has been completely written into $w$ then update $e$ to $w$ and then void $w$.

6. $N$ outputs $e$ as its current conjecture.

To summarise, $N$ is buffering the observed data in $u$ using the protocol of a queue. This buffering process is a bottle neck and conflicts are resolved by defining that the length-lexicographically smaller words have higher priority. The simulated Turing machine $M$ is delayed, as suggested by Pitt (1989) for many complexity-theoretic settings. $M$ reads pause symbols if it overtakes the buffering process in speed. The output of $M$ is stored in $w$ and is then moved into $e$ as soon as it has been completely written. This way $N$ obtains delayed versions of $M$'s conjectures.

Consider now a thick text $T$ for a language to be learnt. For the verification, the most critical point is that for every $x$ occurring in the text, some copy of $x$ in the thick text is eventually buffered in $u$ and then processed by $M$, that is, $M$ reads from the input a translated text for the language to be learnt. As $6 \cdot f(n) < f(n+1)$ implies $\sum_{m \leq n} f(m) \leq 2 \cdot f(n)$, it follows immediately from $f(\mathrm{ll}(z)+1) > 6 \cdot |\mathrm{succ}(z)| \cdot f(\mathrm{ll}(z))$ that $f(\mathrm{ll}(z)+1) > 3 \cdot |\mathrm{succ}(z)| \sum_{y \leq_{\mathrm{ll}} z} f(\mathrm{ll}(y))$. Thus there is an $i$ with $T(i) = z$ such that there is no $j$ satisfying $i - |z| \leq j \leq i + |z|$ and $T(j) <_{\mathrm{ll}} T(i)$. When $N$ processes $T(i)$, the component $x$ of $N$'s memory is either void or contains a value $z' \geq_{\mathrm{ll}} z$; so $x$ will have the value $T(i)$ after the $i$-th cycle. Now, during the next $|z|$ cycles, none of the $T(j)$ will force the replacement of the value of $x$ by something different from $z$ and therefore $z$ will be copied into $u$ during these cycles; once $z$ is in $u$ it will stay there until $M$ reads this copy of $z$. Thus, $M$ sees, during the simulation, copies of all words occurring in the thick text; $M$ therefore works correctly and its conjectures are copied eventually into those of $N$.

Osherson, Stob and Weinstein (1986) showed that every uniformly r.e. family and thus also every automatic family is partially learnable by a recursive learner; if we let $M$ be this learner then $N$ will be an automatic partial learner $N$ for this family.

Similarly, Angluin (1980a) showed that every uniformly recursive family that satisfies the tell-tale condition (and thus every automatic family that satisfies the tell-tale condition) is learnable by a partial learner; so if we let $M$ be this learner then $N$ will be an automatic learner learning this family. Inversely, if an automatic family can be learnt by an automatic learner, then the family must satisfy the tell-tale condition by Angluin (1980a), as every automatic learner is also recursive. $\square$

## 5. Learning by counting the number of words

Jia (2013) constructed automatic families that are automatically learnable from one-one text, but not from arbitrary text; one such family is that of the co-singleton sets in Example 3.3. The next example shows that it is possible to for an automatic family to be automatically unlearnable from arbitrary text, while there is very simple learning algorithm from one-one text. Unlike the algorithms for exponential text in the previous section, which added the values $2^{\mathrm{ll}(x)}$ for each datum $x$ observed, this algorithm simply counts words.

**Example 5.1.** *For a binary number $h = 2^n - 2^{m+1} + k$ with $k < 2^m$ and $m + 2 \leq n$, let $L_h$ contain all binary words of length $n$ which either do not start with $0^{n-m-1}1$ or which are of the form $0^{n-m-1}1x$ with $|x| = m$ and the binary value of $x$ being strictly below $k$; for $h = 2^n - 1$ with $n > 0$, let $L_h = \{0,1\}^n \setminus \{0^{n-1}1\}$. The family of the languages $L_h$ as above is automatically learnable from one-one text by counting the number of words, but it is not learnable from arbitrary text.*

**Proof.** In order to obtain an automatic family, we use a coding of $(\mathbb{N}, n \mapsto n+1)$ where each number $n$ is identified with the binary representation without leading zeroes in the usual way. For every $h > 0$, $L_h$ represents a set with $h$ elements. To see this, note that $L_h = \{x \in \{0,1\}^n : x <_{\text{lex}} v_h \vee x >_{\text{lex}} w_h\}$, where for $h = 2^n - 1$ the values of $v_h, w_h$ are both $0^{n-1}1$ and for $2^n - 2^{m+1} + k$ with $m+1 < n$ and $k < 2^m$ the values of $v_h$ and $w_h$ are $0^{n-m-1}1u$ and $0^{n-m-1}1^{m+1}$, respectively, where $u$ is the binary word of length $m$ with binary value $k$. Now $L_h$ contains all words of $\{0,1\}^n$ except those between $v_h$ and $w_h$, inclusively. There are $2^{m+1} - k$ words in the closed interval from $v_h$ to $w_h$ of length $n$ and therefore $L_h$ has $2^n - 2^{m+1} + k$ many values.

For $h, u, m, n$ as above, as the input $h$ is either $1^n$ in case that $h = 2^n - 1$ or $1^{n-m-1}0u$ in case that $h = 2^n - 2^{m+1} + k$, the mappings $h \mapsto v_h$ and $h \mapsto w_h$ are automatic. Therefore the family $\{L_h : h > 0\}$ is automatic. The automatic family has the following straightforward learner from one-one text:

> The learner for this family simply counts the number $h$ of words seen so far (not counting #) in the one-one text. If $h > 0$ then the learner conjectures $h$ else the learner outputs ? in order to signal that there is no conjecture.

This learner is clearly automatic and correct.

It remains to show that the family is not automatically learnable from arbitrary text. For any $n$, consider the behaviour of an arbitrary learner after seeing a sequence containing $n-3$ inputs $(x_1, x_2, \ldots, x_{n-3})$ where $x_m$ is from the set $\{0^{n-m-2}10\} \cdot (\{0,1\}^m \setminus \{1\}^m)$. For given $n$, the number of possible such sequences is

$$\prod_{m=1}^{n-3} (2^m - 1) \geq \prod_{m=1}^{n-3} 2^{m-1} = 2^{(n-4)(n-3)/2};$$

however, the size of the possible memory of the learner after seeing such a sequence has at most $c \cdot (n+1)$ symbols, for some constant $c$. So the memory can only take exponentially many values while the number of the sequences of the given form is an exponential of a quadratic function. Thus, for large enough $n$, there exist two distinct such sequences $\sigma = (x_1, x_2, \ldots, x_{n-3})$ and $\rho = (y_1, y_2, \ldots, y_{n-3})$ with $x_m$, $y_m$ from the set $0^{n-m-2}10\{0,1\}^m$ such that the learner has the same memory after seeing $\sigma$ and after seeing $\rho$. There is an $m \in \{1, 2, \ldots, n-3\}$ for which $x_m \neq y_m$, say $x_m < y_m$. One can view $y_m$ as a binary number $h$ and $y_m \in L_{h+1}$ but $y_m \notin L_h$. Now let $\tau$ be a sequence containing all the words in $L_h$. In the limit the learner behaves the same way on $\sigma\tau\#^\infty$ as on $\rho\tau\#^\infty$ although these are texts for the two distinct languages $L_h$ and $L_{h+1}$. Thus, no automatic learner can learn the given family from arbitrary text. □

**Proposition 5.2.** The family of Example 5.1 is also automatically learnable from $k$-text, fat text, balanced text, exponential text and thick text. Furthermore, the family is verifiable from arbitrary text.

**Proof.** The $k$-text learner would just have two counters, a first counter which counts the data modulo $k$ and a second counter $h$ which increments whenever the first counter goes from $k-1$ modulo $k$ to 0 modulo $k$. The value $h$ of the second counter converges to the index of the language to be learnt as in Example 5.1.

The fat text learner can be implemented easily by starting with $h = 1$ and always incrementing $h$ when an example outside $L_h$ has been seen. When learning $L_{h'}$ with $h' > 0$, as long as $h < h'$, there is an element $x \in L_{h'} - L_h$, which will be observed eventually in the fat text and therefore $h$ will be incremented again; however, once $h$ has reached $h'$ it will stay with this value forever.

The balanced text learner maintains several variables: The number $k$ of times the first word in the text was seen, the number $\ell$ of words (different from #) seen so far, an index $h$ which is initialised as 1 and is incremented whenever an element outside $L_h$ appears in the data. Also number $m$ is maintained which indicates the number of elements seen when $h$ was last updated. If $k > m$ then the learner conjectures $h$ else the learner conjectures the most recent value of $\ell/k$ that was computed and is an integer. On a $k'$-text with $1 \leq k' \leq \infty$ for $L_{h'}$, the following four cases can arise.

- Case $k' > m$ and $h < h'$: The learner will eventually see an element outside $L_h$ as some $x \in L_{h'} - L_h$ has not been seen $k'$ times. Therefore $h$ will be increased eventually;
- Case $k' \leq m$, $h < h'$ and the learner sees eventually again a datum outside $L_h$: as in the previous case, $h$ will be increased eventually;
- Case $k' \leq m$ and the learner does not see again a datum outside $L_h$: In this case, $k$ will stay below $m$ and $\ell, k$ will eventually take their final values and their quotient $\ell/k$ will converge to $h'$ so that the learner learns in this case.
- Case $h = h'$ and $k' > m$: In this case, $k$ will be eventually above $m$ and from then onwards the learner will conjecture $h$, so the learner learns also in this case.

Since eventually only the third or fourth case applies the learner learns the family.

For exponential text, note that there is a finite learner from informant which waits for the first word $v$ to be found such that the informant evaluates $v$ to 0 but its length-lexicographic predecessor to 1. For this $v$ there is a unique $h$ with $v_h = v$ and the learner conjectures this $h$. By Theorem 4.1 the family is also automatically learnable from exponential text.

For thick text, the usual algorithm for learning automatic families satisfying the tell-tale condition can be used.

The verifier for $L_h$ with $h > 0$ would automatically compute $v_h, w_h$ and then monitor whether (a) some word of length $|v_h|$ has been observed, whether (b) neither $v_h$ nor $w_h$ have not yet been observed and whether (c) the length-lexicographic predecessor of $v_h$ has been observed. If all three conditions (a), (b), (c) are currently true then the verifier also conjectures *yes* else the verifier conjectures *no*. □

## 6. Conclusion

Gold (1967) observed already that the class of all recursively enumerable languages is learnable from primitive recursive text and subsequent studies have investigated the influence of various types of text on learning. For example, it is known that learning from fat text is helpful for the case of iterative learning. This transfers to automatic learning, as Jain, Luo and Stephan (2012) have proven that fat text overcomes the limitations of memorisation in automatic learning. Furthermore, Jia (2013) has shown that certain automatic families are

| Type of Text | Automatic learnability of all automatic families | Automatic learnability of automatic families with tell-tale condition | Reference |
|---|---|---|---|
| Fat | Partially learnable | Learnable | Jain, Luo and Stephan (2012) |
| One-one | Partially verifiable | Verifiable | Theorems 3.4, 3.7 |
| Balanced | Partially verifiable | Verifiable | Theorems 3.4, 3.7 |
| Exponential | Partially learnable | Learnable if finitely learnable from informant by a recursive learner | Theorem 4.1 |
| Thick | Partially Learnable | Learnable | Theorem 4.2 |

Table 1: Overview of results

automatically learnable from one-one text although they are not automatically learnable from arbitrary text. The present work expands on these results by studying in detail what bearing the level of repetitiveness in texts has on learnability using automatons. Table 1 summarises the main results; recall that (partial) learnability implies (partial) verifiability.

The task of verification is easier than learning. For example, the family of all languages $L_e = \{0,1\}^{|e|} \setminus \{e\}$ with $e \in \{0,1\}^+$ is verifiable from arbitrary text but not automatically learnable from arbitrary text. However it is an open problem whether all automatic families satisfying Angluin's tell-tale condition are automatically learnable from one-one texts, balanced texts and exponential texts.

Nonetheless, texts of these types are helpful; for example, it is possible to learn the family of all sets of the form $L_e = \{0,1\}^{|e|} \setminus \{e\}$ with $e \in \{0,1\}^+ \cup \{2\}^+$ from such texts while it is impossible to verify it from arbitrary text, see Theorem 3.5.

**Acknowledgments.** Karen Seidel was partially supported by the German Research Foundation (DFG) through grant KO 4635/1-1 (SCL). Sanjay Jain was supported in part by NUS grant C252-000-087-001. Furthermore, Sanjay Jain and Frank Stephan have been supported in part by the Singapore Ministry of Education Academic Research Fund Tier 2 grant MOE2016-T2-1-019 / R146-000-234-112. Part of this work was done while Frank Stephan was on sabbatical leave to the Universität Bonn in April 2017 and to the Bundeswehr University Munich in June 2017.

## References

Klaus Ambos-Spies, Serikzhan Badaev and Sergey Goncharov. Inductive inference and computable numberings. *Theoretical Computer Science*, 412(18):1652–1668, 2011.

Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control* 45:117–135, 1980.

Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21(1):46–62, 1980.

Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29:741–765, 1982.

Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

Achilles A. Beros. Anomalous vacillatory learning. *The Journal of Symbolic Logic*, 78(4):1183–1188, 2009.

Lenore Blum and Manuel Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

Achim Blumensath. *Automatic structures.* Diploma thesis, RWTH Aachen, 1999.

Achim Blumensath and Erich Grädel. Automatic structures. *15th Annual IEEE Symposium on Logic in Computer Science* (LICS), pages 51–62, IEEE Computer Society, 2000.

John Case, Sanjay Jain, Trong Dao Le, Yuh Shin Ong, Pavel Semukhin and Frank Stephan. Automatic Learning of Subclasses of Pattern Languages. *Information and Computation*, 218:17–35, 2012.

John Case, Sanjay Jain, Yuh Shin Ong, Pavel Semukhin and Frank Stephan. Automatic learners with feedback queries. *Journal of Computer and System Sciences*, 80(4):806–820, 2014.

John Case, Sanjay Jain, Samuel Seah and Frank Stephan. Automatic functions, linear time and learning. *Logical Methods in Computer Science*, 9(3), 2013.

John Case and Timo Kötzing. Difficulties in forcing fairness of polynomial time inductive inference. *Algorithmic Learning Theory*, Twentieth International Conference, ALT 2009, Porto, Portugal, 3–5 October 2009. Proceedings. Springer LNAI, 5809:263–277, 2009.

Henning Fernau. Identification of function distinguishable languages. *Theoretical Computer Science*, 290:1679–1711, 2003.

Rusins Freivalds, Efim Kinber and Carl H. Smith. On the impact of forgetting on learning machines. *Journal of the ACM*, 42:1146–1168, 1995.

William I. Gasarch and Carl H. Smith. Learning via queries. *Journal of the ACM*, 39(3), 649–674, 1992.

Michael Geilke and Sandra Zilles. Learning Relational Patterns. *Algorithmic Learning Theory, Twentysecond International Conference* (ALT 2011), Springer LNAI 6925:84–98, 2011.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

Gunter Grieser. Reflective inductive inference of recursive functions. *Theoretical Computer Science*, 397(1-3):57–69, 2008.

Tom Head, Satoshi Kobayashi and Takashi Yokomori. Locality, reversibility, and beyond: learning languages from positive data. *Algorithmic Learning Theory, Ninth International Conference*, (ALT 1998). Springer LNAI 1501:191–204, 1998.

Jeffrey Heinz, Anna Kasprzik and Timo Kötzing. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457:111–127, 2012.

Bernard R. Hodgson. On direct products of automaton decidable theories. *Theoretical Computer Science*, 19:331–335, 1982.

Rupert Hölzl, Sanjay Jain and Frank Stephan. Inductive inference and reverse mathematics. *Annals of Pure and Applied Logic* 167:1242–1266, 2016.

Marcus Hutter and Jan Poland. *Algorithmic Learning Theory*, Fifteenth International Conference, ALT 2004, Padova, Italy, 2–5 October 2004, Proceedings. Springer LNAI, 3244:279–293, 2004.

Sanjay Jain, Efim Kinber and Rolf Wiehagen. Language learning from texts: degrees of intrinsic complexity and their characterizations. *Journal of Computer and System Sciences*, 63(3):305–354, 2000.

Sanjay Jain, Qinglong Luo and Frank Stephan. Learnability of automatic classes. *Journal of Computer and System Sciences*, 78(6):1910–1927, 2012. Special issue on LATA 2010.

Sanjay Jain, Yuh Shin Ong, Shi Pu and Frank Stephan. On automatic families. *Proceedings of the eleventh Asian Logic Conference* in honour of Professor Chong Chi Tat on his sixtieth birthday, pages 94–113, World Scientific, 2012.

Sanjay Jain, Daniel N. Osherson, James S. Royer and Arun Sharma. *Systems That Learn.* MIT Press, 2nd Edition, 1999.

Klaus Peter Jantke. Monotonic and non-monotonic inductive inference. *New Generation Computing*, 8(4):349–360, 1991.

Mengchi Jia. *Automatic Learning from Specific Text.* Bachelor of Science and Bachelor of Computing Dissertation (Double Degree Programme), National University of Singapore, 2013.

Michael Kearns and Leonard Pitt. A polynomial-time algorithm for learning $k$-variable pattern languages from examples. *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 57–71, Morgan Kaufmann Publishers Inc., 1989.

Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. *Logical and Computational Complexity*, (International Workshop LCC 1994). Springer LNCS 960:367–392, 1995.

Efim Kinber. Learning regular expressions from representative examples and membership queries. *International Colloquium on Grammatical Inference*, ICGI 2010, Springer LNCS 6339:94–108, 2010.

Efim Kinber and Frank Stephan. Language learning from texts: mind changes, limited memory and monotonicity. *Information and Computation*, 123:224–241, 1995.

Steffen Lange and Rolf Wiehagen. Polynomial time inference of arbitrary pattern languages. *New Generation Computing*, 8:361–370, 1991.

Learning indexed families of recursive languages from positive data: a survey. *Theoretical Computer Science*, 397(1):194–232, 2008.

Eliana Minicozzi. Some natural properties of strong-identification in inductive inference. *Theoretical Computer Science*, 2:345–360, 1976.

Wei Luo and Oliver Schulte. Mind change efficient learning. *Information and Computation*, 204(6):989–1011, 2006.

Yasuhito Mukouchi and Setsuo Arikawa. Towards a mathematical theory of machine discovery from facts. *Theoretical Computer Science*, 137(1):53–84, 1995.

Daniel Osherson, Michael Stob and Scott Weinstein, Learning strategies. *Information and Control*, 53:32–51, 1982.

Daniel Osherson, Michael Stob and Scott Weinstein, *Systems That Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists.* Bradford — The MIT Press, Cambridge, Massachusetts, 1986.

Leonard Pitt. Inductive inference, DFAs, and computational complexity. *Analogical and Inductive Inference, Proceedings of the Second International Workshop*, AII 1989. Springer LNAI 397:18–44, 1989.

Sasha Rubin. *Automatic Structures.* Ph.D. Thesis, University of Auckland, 2004.

Sasha Rubin. Automata presenting structures: a survey of the finite string case. *The Bulletin of Symbolic Logic*, 14:169–209, 2008.

Gisela Schäfer-Richter. *Über Eingabeabhängigkeit und Komplexität von Inferenzstrategien.* Dissertation. Rheinisch-Westfälische Technische Hochschule Aachen, 1984.

Takeshi Shinohara and Hiroki Arimura. Inductive inference of unbounded unions of pattern languages from positive data. *Theoretical Computer Science*, 241(1–2):191–209, 2000.

Frank Stephan and Yuri Ventsov. Learning algebraic structures from text. *Theoretical Computer Science*, 268(2):221–273, 2001.

Kenneth Wexler and Peter W. Culicover. *Formal Principles of Language Acquisition.* MIT Press, 1980.

Rolf Wiehagen, Thomas Zeugmann: Ignoring data may be the only way to learn efficiently. *Journal of Experimental and Theoretical Artificial Intelligence*, 6(1):131–144, 1994.