Team Learning of Formal Languages

Sanjay Jain
Dept. of Info. Systems & Computer Science
National University of Singapore
Singapore 0511, Republic of Singapore
sanjay@iscs.nus.sg

Arun Sharma
School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
arun@cse.unsw.edu.au

Abstract

A team of learning machines is a multiset of learning machines. A team is said to successfully learn a concept just in case each member of some nonempty subset, of predetermined size, of the team learns the concept.

Team learning of computer programs for computable functions from their graphs has been studied extensively. However, team learning of languages turns out to be a more suitable theoretical model for studying computational limits on multi-agent machine learning. The main reason for this is that language learning can model both learning from positive data and learning from positive and negative data, whereas function learning models only learning from positive and negative data.

Some theoretical results about learnability of formal languages by teams of algorithmic machines are surveyed. Some new results about restricted classes of languages are presented. These results are mainly about two issues: redundancy and aggregation. The issue of redundancy deals with the impact of increasing the size of a team and increasing the number of machines required to be successful. The issue of aggregation deals with conditions under which a team may be replaced by a single machine without any loss in learning ability.

Scenarios which can be modeled by team learning are also presented.

1 INTRODUCTION

Algorithmic identification in the limit of two concept classes, computable functions and recursively enumerable languages, have been studied extensively in the computational learning theory literature.

We first describe the learning of a computable function. A learning machine is fed the graph of a computable function, and the machine, from time to time, conjectures a sequence of computer programs. The machine is said to learn the function just in case its conjectures converge to a program for the function. Recently, learning of functions by teams of learning machines has become a very active area of research and has been suggested as a theoretical model for multi-agent learning (for example, see [17, 16, 7, 4, 3, 12]).

We argue that the utility of function learning as a model for machine learning is somewhat limited. Data available to most learning systems are of two kinds: positive data and complete (both positive and negative) data. Function learning models only the later. Since the input to a function learning machine is the graph of a function, the negative data is implicitly available to the learning machine. To see this: if the ordered pair $(2, 5)$ is encountered in the graph, then a learning machine can safely assume that each pair $(2, x)$, $x \neq 5$, does not belong to the function.

However, this problem does not arise in the case of identification in the limit of recursively enumerable languages (described in the next section), as both learning from positive data and complete data can be modeled. For this reason, we suggest that team learning of languages is a more suitable model of multi-agent learning.

In what follows, we proceed formally. In Section 2, we introduce the preliminary notions about identification in the limit of languages by single machines. In Section 3, we motivate and describe identification of languages by teams of machines. Section 4 contains a survey of fundamental results about team learning of languages. In Section 5, we summarize new results about team learning of restricted classes of languages.

In Section 6, we describe two hypothetical scenarios that may be modeled using team learning. Finally, in an appendix, we sketch one of the proof techniques.

2 LEARNING BY A SINGLE MACHINE

Let N denote the set of natural numbers, $\{0, 1, 2, \ldots\}$. As already noted our domain is the collection of recursively enumerable languages over N. A grammar for a recursively enumerable language L is a computer program that accepts L (or, equivalently, generates L [6]). For any recursively enumerable language L, the elements of L constitute its positive data and the elements of the complement, $N - L$, constitute its negative data. We next describe notions that capture the presentation of positive data and presentation of both positive and negative data.

Definition 1 A *text* for the language L is an infinite sequence (repetitions allowed) consisting of all and only the elements of L. T denotes a typical variable for texts.

So, a text for L represents an instance of positive data presentation for L. The next definition introduces a notion that represents an instance of both positive and negative data presentation for L.

Definition 2 An informant for L is an infinite sequence (repetitions allowed) of ordered pairs such that for each $n \in N$ either $(n, 1)$ or $(n, 0)$ (but not both) appear in the sequence and $(n, 1)$ appears only if $n \in L$ and $(n, 0)$ appears only if $n \notin L$.

A learning machine may be thought of as an algorithmic device that takes as input finite initial sequences of texts and informants and that from time to time conjectures computer programs as hypotheses. $\mathbf{M}$ denotes a typical variable for learning machines. We now consider what it means for a learning machine to successfully learn languages. The criterion of success considered in the present paper is Gold's [5] *identification in the limit*. We first introduce it for learning from positive data.

Definition 3 [5]

(a) $\mathbf{M}$ **TxtEx**-*identifies* an r.e. language L just in case $\mathbf{M}$, fed any text for L, converges to a grammar for L. In this case we say that $L \in \mathbf{TxtEx}(\mathbf{M})$.

(b) $\mathbf{M}$ **TxtEx**-*identifies* a collection of languages, $\mathcal{L}$, just in case $\mathbf{M}$ **TxtEx**-identifies each language in $\mathcal{L}$.

(c) **TxtEx** denotes all such collections $\mathcal{L}$ of r.e. languages such that some machine **TxtEx**-identifies $\mathcal{L}$.

The class **TxtEx** is a set theoretic summary of the capability of machines to **TxtEx**-identify collections of r.e. languages. Intuitively, if a collection $\mathcal{L} \in \mathbf{TxtEx}$, then there exists a machine that **TxtEx**-identifies each language in the collection $\mathcal{L}$.

It is easy to see that any singleton class of languages is identifiable because a "dumb" machine that ignores its input and keeps on emitting a grammar for the only language in the class is successful on that language; however, such a machine is unsuccessful on every other language. It is precisely for this reason, that we introduced Part (b) in the above definition because machines that learn only one language are not very interesting. Also, **FIN**, the collection of finite languages, belongs to **TxtEx** because a machine employing the heuristic of emitting a grammar for all the elements it has seen at any given time will suffice because eventually it will see all the elements in the finite language.

We now define identification from both positive and negative data.

Definition 4 [5]

(a) $\mathbf{M}$ **InfEx**-*identifies* an r.e. language L just in case $\mathbf{M}$, fed any informant for L, converges to a grammar for L. In this case we say that $L \in \mathbf{InfEx}(\mathbf{M})$.

(b) $\mathbf{M}$ **InfEx**-*identifies* a collection of languages, $\mathcal{L}$, just in case $\mathbf{M}$ **InfEx**-identifies each language in $\mathcal{L}$.

(c) **InfEx** denotes all such collections $\mathcal{L}$ of r.e. languages such that some machine **InfEx**-identifies $\mathcal{L}$.

3 LEARNING BY A TEAM

A team of learning machines is a multiset[1] of learning machines. Before we formally define learning by a team, it is worth considering the origins of team learning. Consider the following theorem for **TxtEx**-identification.

Theorem 1 [2] *There are collections of languages $\mathcal{L}_1$ and $\mathcal{L}_2$ such that*

(a) $\mathcal{L}_1 \in \mathbf{TxtEx}$,

(b) $\mathcal{L}_2 \in \mathbf{TxtEx}$, *but*

(c) $(\mathcal{L}_1 \cup \mathcal{L}_2) \notin \mathbf{TxtEx}$.

The above result[2], popularly referred to as the "non-union theorem," says that the class **TxtEx** is not

[1] A *multiset* is like a set, but elements need not be distinct. For example, the cardinality of set $\{2, 2, 3\}$ is 2, but the cardinality of the multiset $\{2, 2, 3\}$ is 3.

[2] Taking $\mathcal{L}_1 = \{N\}$ and $\mathcal{L}_2 = \mathbf{FIN}$ yields a proof because of Gold's [5] result that no collection of languages

closed under union. In other words, there are collections of languages that are identifiable, but the union of these collections is not identifiable. This result may be viewed as a fundamental limitation on building a general purpose device for machine learning, and, to an extent, justifies the use of heuristic methods in Artificial Intelligence. However, this result also suggests a more general criterion of identification in which a team of learning machines is employed and success of the team is the success of any member in the team. We illustrate this idea next.

Consider the collections of languages $\mathcal{L}_1$ and $\mathcal{L}_2$ in Theorem 1. Let $\mathbf{M}_1$ **TxtEx**-identify $\mathcal{L}_1$ and $\mathbf{M}_2$ **TxtEx**-identify $\mathcal{L}_2$. Now, if we employed a team of $\mathbf{M}_1$ and $\mathbf{M}_2$ to identify $\mathcal{L}_1 \cup \mathcal{L}_2$ and weakened the criterion of success to the requirement that success is achieved just in case any one member in the team is successful, then the collection $\mathcal{L}_1 \cup \mathcal{L}_2$ becomes identifiable by the team consisting of $\mathbf{M}_1$ and $\mathbf{M}_2$ under this new criterion of success. This idea can be extended to teams of n machines out of which at least m ($m \leq n$) are required to be successful. The formal definitions for team identification of languages are presented next. J. Case first suggested the notion of team identification for functions based on the non-union theorem of the Blums [2], and it was extensively investigated by C. Smith [17]. The general case of m out of n teams is due to Osherson, Stob, and Wienstein [13]. Jain and Sharma [8] first investigated team learning for languages.

Definition 5 Let $m, n \in N$ and $0 < m \leq n$.

(a) A team of n machines $\{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n\}$ is said to $\mathbf{Team}_n^m\mathbf{TxtEx}$-identify a language L just in case at least m members in the team **TxtEx**-identify L. In this case we write $L \in \mathbf{Team}_n^m\mathbf{TxtEx}(\{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n\})$.

(b) A team of n machines $\{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_n\}$ is said to $\mathbf{Team}_n^m\mathbf{TxtEx}$-identify a collection of languages $\mathcal{L}$ just in case the team $\mathbf{Team}_n^m\mathbf{TxtEx}$-identifies each language in $\mathcal{L}$.

(c) $\mathbf{Team}_n^m\mathbf{TxtEx}$ is defined to be the class of sets $\mathcal{L}$ of r.e. languages such that some team of n machines $\mathbf{Team}_n^m\mathbf{TxtEx}$-identifies $\mathcal{L}$.

We can similarly define the class $\mathbf{Team}_n^m\mathbf{InfEx}$ for team learning from both positive and negative data.

that contains all the finite languages and an infinite language can be identified in the limit from only positive data.

4 RESULTS

We now survey some of the results about team learning of languages. The results that we present here are about redundancy and aggregation. We direct the reader to [8, 10, 9, 11] for additional results.

First, it is easy to see the following proposition.

Proposition 1 *Let $k, m, n \in N$ such that $0 < m \leq n$ and $k \geq 1$.*

(a) $\mathbf{Team}_n^m\mathbf{TxtEx} \subseteq \mathbf{Team}_{n\cdot k}^{m\cdot k}\mathbf{TxtEx}$.

(b) $\mathbf{Team}_n^m\mathbf{InfEx} \subseteq \mathbf{Team}_{n\cdot k}^{m\cdot k}\mathbf{InfEx}$.

The above proposition says that for both texts and informants, all the collections of languages that can be learned by a given team can also be learned if we multiply the size of the team and the number of machines required to be successful by the same factor. In other words, introducing redundancy does not hurt. The question is: Does it help ? We consider team learning from informants first, followed by team learning from texts.

4.1 TEAM LEARNING FROM INFORMANTS

For identification from both positive and negative data, introducing redundancy in the team does not yield any extra learning ability.

Theorem 2 *Let $k, m, n \in N$ such that $0 < m \leq n$ and $k \geq 1$. Then* $\mathbf{Team}_n^m\mathbf{InfEx} = \mathbf{Team}_{n\cdot k}^{m\cdot k}\mathbf{InfEx}$.

The above result says that the collections of languages that can be identified by teams employing n machines and requiring at least m to be successful are exactly the same as those collections which can be identified by teams employing $n \cdot k$ machines and requiring at least $m \cdot k$ to be successful.

We next consider the question of aggregation, that is, under what conditions can a team be replaced by a single machine without any loss in learning ability. Part (a) of the next result says that if a majority of the members in the team are required to be successful, then employing a team does not yield any extra learning ability. Part (b) of the result says that $\frac{1}{2}$ is indeed the cutoff. In the sequel, we refer to such cutoff points as *aggregation ratios*.

Theorem 3 *(a)* $(\forall m, n \mid \frac{m}{n} > \frac{1}{2})[\mathbf{Team}_n^m\mathbf{InfEx} = \mathbf{InfEx}]$.

(b) $\mathbf{InfEx} \subset \mathbf{Team}_2^1\mathbf{InfEx}$.

A proof of the above result can be worked out using techniques from Pitt [15].

4.2 TEAM LEARNING FROM TEXTS

Surprisingly, introducing redundancy in the team does help sometimes in the context of learning from only positive data. The following result says that there are collections of languages that can be **TxtEx**-identified by teams employing 4 machines and requiring at least 2 to be successful, but cannot be **TxtEx**-identified by any team employing 2 machines and requiring at least 1 to be successful. $\supset$ denotes proper superset.

Theorem 4 $\mathbf{Team}_4^2\mathbf{TxtEx} \supset \mathbf{Team}_2^1\mathbf{TxtEx}$.

Even more surprising is the next theorem which implies that the collections of languages that can be **TxtEx**-identified by teams employing 6 machines and requiring at least 3 to be successful are exactly the same as those collections that that can be **TxtEx**-identified by teams employing 2 machines and requiring at least 1 to be successful.

Theorem 5
$(\forall j)[\mathbf{Team}_{4j+2}^{2j+1}\mathbf{TxtEx} = \mathbf{Team}_2^1\mathbf{TxtEx}]$.

The complete picture is actually quite complicated. The status of teams with success ratio $\frac{1}{2}$ is completely known, but only partial results are known for other team ratios ($\frac{1}{k}, k > 2$); we direct the reader to [9].

The next result sheds light on when a team learning languages from texts can be aggregated into a single machine without loss in learning ability. Part (a) of the result says that if *more than two-thirds* of the members in the team are required to be successful, then employing a team for learning languages from texts does not yield any extra learning ability. Part (b) of the result says that $\frac{2}{3}$ is indeed the cutoff.

Theorem 6 *(a)* $(\forall m, n \mid \frac{m}{n} > \frac{2}{3})[\mathbf{Team}_n^m\mathbf{TxtEx} = \mathbf{TxtEx}]$.

(b) $\mathbf{TxtEx} \subset \mathbf{Team}_3^2\mathbf{TxtEx}$.

5 TEAM LEARNING OF RESTRICTED CLASSES OF LANGUAGES

It may justifiably be argued that recursively enumerable languages are too general to usefully model concepts of practical interest. For this reason, it is worth considering the effects of team learning on restricted collections of languages. In this section, we summarize results about redundancy and aggregation for the following two restrictions on r.e. languages.

- *Recursive languages*: These are languages that have an algorithmic decision procedure. We denote the class of recursive languages by **REC**.
- *Indexed families of recursive languages*: A sequence of nonempty languages $L_0, L_1, \ldots$ is an indexed family just in case there exists a computable function f such that for each $i \in N$ and for each $x \in N$,

  $$f(i,x) = \begin{cases} 1 & \text{if } x \in L_i, \\ 0 & \text{otherwise.} \end{cases}$$

  In other words, there is a uniform decision procedure for languages in the class. Angluin [1] was the first researcher to restrict investigations to indexed families of recursive languages; she was motivated by the fact that most language families of practical interest are indexed families (e.g., the collection of pattern languages).

We now report on redundancy and aggregation issues for these classes. We restrict ourselves to texts, as informants do not yield any new insight.

5.0.1 Identification of Recursive Languages

It turns out that even for recursive languages, redundancy does help sometimes. Our proof of Theorem 4 also implies the following because the language class constructed as the witness for $\mathbf{Team}_4^2\mathbf{TxtEx}$ being a strict super set of $\mathbf{Team}_2^1\mathbf{TxtEx}$ consist only of recursive languages. (*Notation*: The power set of a set A is denoted 2^A.)

Theorem 7 $(\mathbf{Team}_4^2\mathbf{TxtEx} \cap 2^{\mathbf{REC}}) \supset (\mathbf{Team}_4^2\mathbf{TxtEx} \cap 2^{\mathbf{REC}})$.

For similar reasons, the aggregation ratio for team identification of recursive languages turns out to be $\frac{2}{3}$, as recorded in the following theorem.

Theorem 8 *(a)* $(\forall m, n \mid \frac{m}{n} > \frac{2}{3})$ $[(\mathbf{Team}_n^m\mathbf{TxtEx} \cap 2^{\mathbf{REC}}) = (\mathbf{TxtEx} \cap 2^{\mathbf{REC}})]$.

(b) $(\mathbf{TxtEx} \cap 2^{\mathbf{REC}}) \subset (\mathbf{Team}_3^2\mathbf{TxtEx} \cap 2^{\mathbf{REC}})$.

It may be argued that if we are restricting ourselves to learning of recursive languages then we should consider identifying decision procedures instead of grammars (accepting procedures). The following definition formalizes this notion. (*Notation*: A *characteristic function* of a language A is the function which is 1 on elements of A and 0 on non-elements of A.)

Definition 6 [5]

(a) $\mathbf{M}$ **TxtExCI**-*identifies* a recursive language L just in case $\mathbf{M}$, fed any text for L, converges to a program that computes the characteristic function of L. In this case we say that $L \in \mathbf{TxtExCI}(\mathbf{M})$.

(b) $\mathbf{M}$ **TxtExCI**-*identifies* a collection of languages, $\mathcal{L}$, just in case $\mathbf{M}$ **TxtExCI**-identifies each language in $\mathcal{L}$.

(c) **TxtExCI** denotes all such collections $\mathcal{L}$ of recursive languages such that some machine **TxtExCI**-identifies $\mathcal{L}$.

It should be noted that the hypothesis space of the learner in the above definition is still the set of all programs; it is only required that the final converged program compute the characteristic function of the language being learned. Osherson and Weinstein [14] observed the following fact which implies that there are collections of recursive languages for which a grammar can be identified from texts, but for which a decision procedure cannot be identified from texts.

Theorem 9 [14] $\mathbf{TxtExCI} \subset (\mathbf{TxtEx} \cap 2^{\mathbf{REC}})$.

We next consider team identification of decision procedures for recursive languages from texts. Clearly, the class $\mathbf{Team}_n^m\mathbf{TxtExCI}$ can be defined. Until now, we have seen that in the case of learning from only positive data (texts), redundancy sometimes results in increased learning ability. Surprisingly, the following theorem shows that redundancy does not pay when the team is learning decision procedures.

Theorem 10 *Let $k, m, n \in N$ such that $0 < m \leq n$ and $k \geq 1$. Then* $\mathbf{Team}_n^m\mathbf{TxtExCI} = \mathbf{Team}_{n\cdot k}^{m\cdot k}\mathbf{TxtExCI}$.

We sketch a proof of the above result in the Appendix. The aggregation ratio for **TxtExCI** turns out to be $\frac{1}{2}$ as shown by the following theorem.

Theorem 11 *(a)*
$(\forall m, n \mid \frac{m}{n} > \frac{1}{2})[\mathbf{Team}_n^m\mathbf{TxtExCI} = \mathbf{TxtExCI}]$.

(b) $\mathbf{TxtExCI} \subset \mathbf{Team}_2^1\mathbf{TxtExCI}$.

5.0.2 Indexed Families of Recursive Languages

Until now, we have considered learning criteria involving the following concept classes.

- recursively enumerable languages;
- recursive languages.

For both the concept classes, the hypothesis space was the set of all computer programs.

We now consider learning scenarios where the concepts classes are indexed families of recursive languages. We introduce some notation. Let Σ be a fixed terminal alphabet. A grammar G over Σ defines an accepting grammar. $\mathbf{Lang}(G)$ is the language accepted by G.

Since we now exclusively deal with indexed families $\mathcal{L} = \{L_i \mid i \in N\}$, we take as hypothesis space some enumerable family of grammars $G_0, G_1, G_2, \ldots$ over Σ such that $\mathcal{L} \subseteq \{\mathbf{Lang}(G_i) \mid i \in N\}$. We also require that the membership in $\mathbf{Lang}(G_i)$ is uniformly decidable over all $i \in N$ and all strings $s \in \Sigma^*$. When a learning machine emits i, we interpret it to mean that it is conjecturing the grammar G_i.

Gold's criterion of identification in the limit can be adapted to the identification of indexed families as follows:

Definition 7 *Let $\mathcal{L}$ be an indexed family and let $\mathcal{G} = \{G_i \mid i \in N\}$ be a hypothesis space.*

(a) Let $L \in \mathcal{L}$. A machine $\mathbf{M}$ **TxtEx**-identifies *with respect to $\mathcal{G}$ just in case $\mathbf{M}$, fed any text for L, converges to j and $L = \mathbf{Lang}(G_j)$.*

(b) A machine $\mathbf{M}$ **TxtEx**-identifies $\mathcal{L}$ *with respect to $\mathcal{G}$ just in case for each $L \in \mathcal{L}$,* $\mathbf{M}$ **TxtEx**-*identifies L with respect to $\mathcal{G}$.*

(c) $[\mathbf{TxtEx}]_{\mathbf{Index}}$ *denotes the collection of all indexed families $\mathcal{L}$ for which there is a machine* $\mathbf{M}$ *and a hypothesis space $\mathcal{G}$ such that* $\mathbf{M}$ **TxtEx**-*identifies $\mathcal{L}$ with respect to $\mathcal{G}$.*

Similarly, we can define a hypothesis space of decision procedures and define the class $[\mathbf{TxtExCI}]_{\mathbf{Index}}$. It turns out that for indexed families of recursive languages, learning grammars and decision procedures are equivalent.

Proposition 2 $[\mathbf{TxtEx}]_{\mathbf{Index}} = [\mathbf{TxtExCI}]_{\mathbf{Index}}$.

Hence we only consider grammar identification in our investigation of team learning for indexed families. One can define the class $[\mathbf{Team}_n^m\mathbf{TxtEx}]_{\mathbf{Index}}$ in a natural way.

The following two theorems summarize that in the case of learning indexed families of recursive languages from texts, redundancy does not pay and the aggregation ratio is $\frac{1}{2}$.

Theorem 12 *Let $k, m, n \in N$ such that $0 < m \leq n$ and $k \geq 1$. Then* $[\mathbf{Team}_n^m\mathbf{TxtEx}]_{\mathbf{Index}} = [\mathbf{Team}_{n\cdot k}^{m\cdot k}\mathbf{TxtEx}]_{\mathbf{Index}}$.

Theorem 13 *(a)* $(\forall m,n \mid \frac{m}{n} > \frac{1}{2})[[\mathbf{Team}_n^m\mathbf{TxtEx}]_{\mathbf{Index}} = [\mathbf{TxtEx}]_{\mathbf{Index}}]$.

(b) $[\mathbf{TxtEx}]_{\mathbf{Index}} \subset [\mathbf{Team}_2^1\mathbf{TxtEx}]_{\mathbf{Index}}$.

Finally we would like to note that if considered learning of indexed families where the hypothesis space was not restricted, that is we allowed the learning machine to emit any computer program, then it can be shown the redundancy pays in some cases, and the aggregation ratio is $\frac{2}{3}$.

We are also able to study the effects of team learning on various strategies of identification in which successive conjectures of a learning machine satisfy certain constraint (for example, successive conjectures are generalizations or specializations). These results will be presented in a fuller version of this paper.

6 SETTINGS FOR TEAM LEARNING

Finally, it is worth noting an aspect of team identification that cannot be overlooked, namely, it may not always be possible to determine which members in the team are successful. This property seems to rob team identification of any possible utility. However, we present below scenarios in which the knowledge of which machines are successful is of no consequence, all that matters is some are.

First, consider a hypothetical situation in which an intelligent species, somewhere in outer space, is attempting to contact other intelligent species (such as humans on earth) by transmitting radio signals in some language (most likely alien to humans). Being a curious species ourselves, we would like to establish a communication link with such a species that is trying to reach out. For this purpose, we could employ a team of language learners each of which perform the following three tasks in a loop:

(a) receive and examine strings of a language (eg., from a radio telescope);

(b) guess a grammar for the language whose strings are being received;

(c) transmit messages back to outer space based on the grammar guessed in Step (b).

If one or more of the learners in the team is actually, but, possibly unknowingly, successful in learning a grammar for the alien language, a correct communication link would be established between the two species.

Consider another scenario in which two countries, A and B, are at war with each other. Country B uses a secret language to transmit movement orders to its troops. Country A, with an intention to confuse the troops of country B, wants to learn a grammar for country B's secret language so that it can transmit conflicting troop movement instructions in that secret language. To accomplish this task, country A employs a team of language learners, each of which perform the following three tasks in a loop:

(a) receive and examine strings of country B's secret language;

(b) guess a grammar for the language whose strings are being received;

(c) transmit conflicting messages based on the grammar guessed in Step (b) (so that B's troops think that these messages are from B's Generals).

If one or more of the learners in the team is actually, but possibly unknowingly, successful in correctly learning a grammar for country B's secret language, then country A achieves its purpose of confusing the troops of country B.

It should be noted that the notion of team learning models only part of the above scenario, as we ignore in our mathematical model the aspect of learners transmitting messages back. We also mathematically ignore possible detrimental effects of a learner guessing an incorrect grammar and transmitting messages that could interfere with messages from a learner that infers a correct grammar (for example, the string 'baby milk powder factory' in one language could mean the string 'ammunition storage' in another!). In no way are these issues trivial; we simply don't have a formal handle on them at this stage.

Acknowledgements

We thank the referees for useful suggestions. Research of Arun Sharma has been partially supported by a grant from the Australian Research Council.

References

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

[2] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[3] R. P. Daley, B. Kalyanasundaram, and M. Velauthapillai. Breaking the probability 1/2 barrier in

fin-type learning. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, Pennsylvania*, pages 203–217. A. C. M. Press, 1992.

[4] R. P. Daley, L. Pitt, M. Velauthapillai, and T. Will. Relations between probabilistic and team one-shot learners. In L. Valiant and M. Warmuth, editors, *Proceedings of the Workshop on Computational Learning Theory*, pages 228–239. Morgan Kaufmann Publishers, Inc., 1991.

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

[6] J. Hopcroft and J. Ullman. *Introduction to Automata Theory Languages and Computation*. Addison-Wesley Publishing Company, 1979.

[7] S. Jain and A. Sharma. Finite learning by a team. In M. Fulk and J. Case, editors, *Proceedings of the Third Annual Workshop on Computational Learning Theory, Rochester, New York*, pages 163–177. Morgan Kaufmann Publishers, Inc., August 1990.

[8] S. Jain and A. Sharma. Language learning by a team. In M. S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, pages 153–166. Springer-Verlag, July 1990. Lecture Notes in Computer Science, 443.

[9] S. Jain and A. Sharma. Computational limits on team identification of languages. Technical Report 9301, School of Computer Science and Engineering; University of New South Wales, 1993.

[10] S. Jain and A. Sharma. Probability is more powerful than team for language identification. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory, Santa Cruz, California*, pages 192–198. ACM Press, July 1993.

[11] S. Jain and A. Sharma. On aggregating teams of learning machines. *Theoretical Computer Science A*, 137(1):85–108, January 1995.

[12] S. Jain, A. Sharma, and M. Velauthapillai. Finite identification of function by teams with success ratio 1/2 and above. *Information and Computation*, 1995. To Appear.

[13] D. Osherson, M. Stob, and S. Weinstein. Aggregating inductive expertise. *Information and Control*, 70:69–95, 1986.

[14] D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists.* MIT Press, Cambridge, Mass., 1986.

[15] L. Pitt. Probabilistic inductive inference. *Journal of the ACM*, 36:383–433, 1989.

[16] L. Pitt and C. Smith. Probability and plurality for aggregations of learning machines. *Information and Computation*, 77:77–92, 1988.

[17] C. Smith. The power of pluralism for automatic program synthesis. *Journal of the ACM*, 29:1144–1165, 1982.

APPENDIX

We now sketch a proof technique that is crucial in showing when a team of machines can be aggregated into a single machine. The technique originates in the work of Pitt [15]. To facilitate the description, we first introduce some notation.

Notation: We let φ denote an acceptable programming system. Since there are countably infinite number of programs in the programming system φ, we refer to each program with its index (or, its number). We let φ_i stand for the partial computable function computed by the program with index i in the φ-system. We denote $\varphi_i(x)\downarrow$ to mean that the program with index i in the φ-system on input x is defined. We write $\varphi_i(x)\downarrow = y$, or simply $\varphi_i(x) = y$, to mean that the program with index i in the φ-system outputs y. We write $\varphi_i(x)\uparrow$ to denote that the program with index i in the φ-system on input x does not halt.

We refer to i as a grammar (acceptor) for L just in case L is the domain of φ_i. That is, L is the recursively enumerable language accepted by the φ-program with index i. A recursive language has a computable decision procedure. We refer to i as a decision procedure (or, the chracteristic index) for a recursive language L just in case $\varphi_i(x) = 1$ if $x \in L$ and $\varphi_i(x) = 0$ if $x \notin L$. Suppose i is not a decision procedure for L, then we consider two kinds of errors that i can make in deciding if a element belongs to L. Suppose $\varphi_i(x)\downarrow$ and either $\varphi_i(x) \neq 1$ when $x \in L$ or $\varphi_i(x) \neq 0$ when $x \notin L$, then we say that i makes an *error of commission* at x. On the other hand if $\varphi_(x)\uparrow$, then we say that i makes an *error of omission* at x.

Finally, for a finite set S of programs, let unify(S) be a program defined as follows:

$\varphi_{\text{unify}(S)}(x)$

Search for $i \in S$ such that $\varphi_i(x)\downarrow$.

If and when such an i is found, let $\varphi_{\text{unify}(S)}(x) = \varphi_i(x)$.

End

Intuitively, unify(S) just computes the union of functions computed by programs in S (on inputs where more than one program in S converge but to different values, unify(S) can arbitrarily choose one of the converging programs). It is easy to observe that if S contains either decision procedures for L or programs that make only errors of omission in deciding membership in L, then unify(S) is a decision procedure for L. This observation will be useful in extracting (in the limit) a decision procedure for a recursive language L from a set of programs F, at least one of which is a decision procedure for L, and a text for L. This is the subject of the next claim.

Claim 1 *Given a finite set of programs, F, and a text T for L, such that at least one of the programs in F is a decision procedure for L, one can find, in the limit, on F and T a decision procedure for L.*

PROOF. (Sketch) Suppose F, and a text T for L be given. We show how to construct a decision procedure for L in the limit.

Let $S_1 = \{i \in F \mid (\exists x \in L)[\varphi_i(x)\downarrow = 0]\}$.

So, programs in S_1 are not decision procedures for L. S_2 below tries to search for programs which accept elements in the complement of L.

Let $S_2 = \{i \in F - S_1 \mid (\exists j \in F - S_1)(\exists x)[\varphi_i(x)\downarrow \neq 0 \wedge \varphi_j(x)\downarrow = 0]\}$. Note that if $i \in F - S_1$ as witnessed by x and j, then $x \notin L$ (since otherwise j would be in S_1).

It should be noted that both S_1 and S_2 will be constructed from F and T in the limit.

We now claim that unify$(F - (S_1 \cup S_2))$ is a decision procedure for L. To see this first note that all programs in F which reject an element of L are in S_1. Thus all elements in $F - S_1$ either accept each element of L or diverge on elements of L. Also since there is a decision procedure for L in F (there exists one such by the assumption), it follows that there is a decision procedure for L in $F - S_1$. Now for any element i in $F - S_1$ such that for some $x \notin L$, $\varphi_i(x)\downarrow = 1$, we have that $i \in S_2$. This follows by the definition of S_2 and the fact that there exists a decision procedure for L in $F - S_1$. Now it is straightforward to see that for each $j \in F - (S_1 \cup S_2)$, for each x, either $\varphi_j(x)\uparrow$ or $\varphi_j(x)$ correctly determines the membership of x in L; that is, j is either a decision procedure for L or only makes errors of omission. It follows that unify$(F - (S_1 \cup S_2))$ is a decision procedure for L. $\square$

We now sketch how the above claim can be used to establish Theorem 10. We will first show that for arbitrary m and n, $\mathbf{Team}_n^m\mathbf{TxtExCI} \subseteq \mathbf{Team}^1_{\lfloor n/m \rfloor}\mathbf{TxtExCI}$. From this it follows that, $\mathbf{Team}_{k\cdot n}^{k\cdot m}\mathbf{TxtExCI} \subseteq \mathbf{Team}^1_{\lfloor n/m \rfloor}\mathbf{TxtExCI}$. It is also easy to see that $\mathbf{Team}^1_{\lfloor n/m \rfloor}\mathbf{TxtExCI} \subseteq \mathbf{Team}_n^m\mathbf{TxtExCI}$. The theorem follows.

We now show that

$$\mathbf{Team}_n^m\mathbf{TxtExCI} \subseteq \mathbf{Team}^1_{\lfloor n/m \rfloor}\mathbf{TxtExCI}.$$

Suppose $\mathbf{M}_1, \mathbf{M}_2, ..., \mathbf{M}_n$ are given. Suppose further that these machines $\mathbf{Team}_n^m\mathbf{TxtExCI}$ identify L, and T is a text for L. We observe the following:

(a) The number of machines converging to a program (perhaps an incorrect one) among $\mathbf{M}_1, \mathbf{M}_2, ..., \mathbf{M}_n$ on text T lies in one of the intervals, $[i \cdot m, (i+1) \cdot m)$ where $1 \leq i \leq \lfloor n/m \rfloor$ and

(b) if the number of converging machines lies in the interval $[i \cdot m, (i+1) \cdot m)$, then at least one of the first $i \cdot m$ converging machines on T converges to a decision procedure for L.

We now construct $\lfloor n/m \rfloor$ machines as follows: machine $\mathbf{M}'_i$ where $1 \leq i \leq \lfloor m/n \rfloor$, on text T, searches for the first $i \cdot m$ machines converging on T. Let F_i be the set of programs to which these machines converge to. $\mathbf{M}'_i$ then, assuming that F_i contains a decision procedure for L, using the above claim tries to find a decision procedure for L.

Note that the assumption — F_i contains a decision procedure for L — is true for at least one i, and thus at least one of the $\lfloor n/m \rfloor$ machines succeeds in $\mathbf{TxtExCI}$-identifying L.