Robust Learning with Infinite Additional Information

Susanne Kaufmann
Interactive Systems Laboratories
Am Fasanengarten 5
Universität Karlsruhe
76128 Karlsruhe, Germany
kaufmann@ira.uka.de

Frank Stephan [*]
Mathematisches Institut
Im Neuenheimer Feld 294
Universität Heidelberg
69120 Heidelberg, Germany
fstephan@math.uni-heidelberg.de

Abstract
The present work investigates Gold style algorithmic learning from input-output examples whereby the learner has access to oracles as additional information. Furthermore this access has to be robust, that means that a single learning algorithm has to succeed with every oracle which meets a given specification. The first main result considers oracles of the same Turing degree: Robust learning with any oracle from a given degree does not achieve more than learning without any additional information.

The further work considers learning from function oracles which describe the whole class of functions to be learned in one of the following four ways: the oracle is a list of all functions in this class or a predictor for this class or a one-sided classifier accepting just the functions in this class or a martingale succeeding on this class.

It is shown that for learning in the limit (Ex), lists are the most powerful additional information, the powers of predictors and classifiers are incomparable and martingales are of no help at all. Similar results are obtained for the criteria of predicting the next value, finite, Popperian and finite Popperian learning. Lists are omniscient for the criterion of predicting the next value but some classes can not be Ex-learned with any of these types of additional information. The class REC of all recursive functions is Ex-learnable with the help of a list, a predictor or a classifier.

[*]Supported by the Deutsche Forschungsgemeinschaft (DFG) grant Am 60/9-1

1 Introduction

Gold style inductive inference [7, 13] is an abstract model for learning: the learner receives the course of values $f(0), f(1), \ldots$ of a recursive function to be learned and synthesizes from this information a program for f. This synthesis has to meet certain convergence requirements. The special model considered in the present paper is that the learner has in addition access to nonrecursive information on the class S from which the function f is taken. This information is provided as a function oracle. But the access to this oracle has to be robust, i.e., ignorant of the actual coding of this information. So the learner has to cope with every oracle which meets a given specification. Four types of such specifications are used in this paper: (1) the oracle is a list of all functions in S; (2) the oracle is a predictor which predicts every $f \in S$ under the model "next value"; (3) the oracle is a one-sided classifier which converges on a function f to 1 iff $f \in S$; (4) the oracle is a martingale which succeeds on every function in S. Learning with additional information has several roots in the literature which are presented now.

Adleman and Blum [1] as well as Gasarch and Pleszkoch [12] transferred the concept of using nonrecursive oracles to inductive inference. Such oracles can be very helpful, for example every high oracle allows to learn all recursive functions in the limit [1]. Also every nonrecursive oracle allows to learn some class finitely which can not be finitely learned without any oracle. But in these models, the machines always depend on the actual form of the oracle. Indeed Theorem 2.1 shows the following: if a class S can be learned via a fixed machine succeeding with any oracle inside a given Turing degree then S can be learned without any help of an oracle. So it is in this context more suitable to specify the oracles by some structural properties which allow to derive some information in a uniform way than by their Turing degree.

A second root is the notion of learning with additional information in the way as introduced by Freivalds and Wiehagen [11]. They presented a model where the additional information is just a number (and not an infinite object as an oracle) which depends on the function f (and not only on the class S). One important result is the following: they presented in addition to the values of the function an upper bound of the size of some program of f. This finite information is already sufficient to learn the whole class of all recursive functions, REC, in the limit. Jain and Sharma [15] extended this work.

Baliga and Case [4] modified this setting such that the learner receives as additional information an index of a higher-order program instead of this upper bound of the program size. This concept is not so powerful as that of Freivalds and Wiehagen [11], as it does not allow the inference of REC. But it still permits inference of larger classes than without any additional information. Jain and Sharma [14] gave as additional information programs which are defined on a "sufficiently large" domain and coincide with the function f to be learned on their domain.

Case, Kaufmann, Kinber and Kummer [9] considered as additional information an index of a certain tree such that among other requirements the function to be learned is an infinite branch of it. These trees had to fulfill certain requirements as having bounded width; they showed that – depending on the parameters of the tree – the class REC of all recursive functions is learnable via a team of machines using this additional information. Furthermore, Merkle and Stephan [19] showed, that there is a class S which can be learned in the limit only if as additional information an index of such a tree is provided, on which the function to be learned is an isolated infinite branch.

Finally Osherson, Stob and Weinstein [23] already went in the direction of the present paper by synthesizing learner for a whole class from additional information on the class S to be learned.

The third root is the work of Angluin [3] whose notion of "minimal adequate teacher" is some kind of infinite additional information. The infinity is given by the fact that the teacher has to answer each query from a given infinite query-language correctly. The answers to the queries are not always unique; e.g., there may be several ways to select counterexamples to a learner's hypothesis. So the learner has in her model to be robust in the sense that learning has to succeed with every teacher which meets the specification. Similarly robust learning in the present paper is modelled by the infinite concept of a "minimal adequate oracle".

Main recursion theoretic notions follow the books of Odifreddi [21] and Soare [26]. $\mathbb{N}$ is the set of natural numbers. A, B, C denote subsets of $\mathbb{N}$ and are identified with their characteristic function: $A(x) = 1$ for $x \in A$ and $A(x) = 0$ for $x \notin A$. f and g denote total recursive functions from $\mathbb{N}$ to $\mathbb{N}$. REC denotes this class of all total recursive functions and $\mathrm{REC}_{0,1} = \{f \in \mathrm{REC} : (\forall x)\,[f(x) \leq 1]\}$. Strings $\sigma, \tau, \eta \in \mathbb{N}^*$ are finite sequences of natural numbers and binary strings α, β, γ range over $\{0,1\}^*$. Strings are also identified with a partial function: If $\sigma = abcc$ then $\sigma(x)$ equals a for $x = 0$, b for $x = 1$, c for $x = 2, 3$ and is undefined for $x > 3$. A string σ is prefix of some other string τ (or function f or set A) iff $\sigma(x) = \tau(x)\downarrow$ ($f(x)$ or $A(x)$, respectively) for all x in the domain of σ (which is denoted by $dom(\sigma)$). $\preceq$ denotes the prefix-relation ($\sigma \preceq \tau$). φ_e is the e-th partial recursive function w.r.t. some fixed acceptable numbering. This numbering is also always used as hypotheses space unless explicitly stated otherwise.

Now an overview on the most important definitions from learning theory [7, 13, 22] is included for the readers' convenience.

Learning functions and classes: A machine learns a class S iff it learns every function $f \in S$ according to the given criterion. The classes S contain always only recursive functions.

Finite Learning (Fin): M learns a function f finitely if $M(\sigma) \in \{?, e\}$ for all $\sigma \preceq f$, $M(\sigma) = e$ for some $\sigma \preceq f$ and $\varphi_e = f$. That means M first outputs the symbol "?" to indicate that it wants to see more data on f and then eventually decides to make a guess e which has to be correct.

Explanatory Learning (Ex): M learns a function f explanatorily if $M(\sigma) = e$ for almost all $\sigma \preceq f$ and e is a fixed program for f. That means that M

first outputs some arbitrary guesses and then converges eventually to a correct program for f.

Popperian Finite Learning (PFin): This is finite learning combined with the additional constraint that any output (also for input not belonging to any function in S) is either the symbol "?" or a program for a total function.

Popperian Explanatory Learning (PEx): This is explanatory learning combined with the additional constraint that any output is either the symbol "?" or a program for a total function.

Predicting the Next Value (NV): A machine M predicts a function f iff M is defined everywhere and $M(f(0)f(1)\dots f(x)) = f(x+1)$ for almost all x.

Now these five concepts (Ex, Fin, PEx, PFin, NV) are combined with different types of oracles. The definitions are stated for Ex but it is easy to see how they are adapted to the other four learning criteria. The general model is, that the learner M receives the course of values of the function f and in addition has access to a function oracle describing a certain information on the class S (and so very indirectly also on the single function f). M accesses the oracle O via queries for $O(x)$ at certain numbers or strings x. M has to learn every $f \in S$ with any "minimal adequate" oracle meeting the specification. Together with the definition an overview on the results is given.

List: A class S is in Ex[List] iff there is a machine M such that M equipped with a function oracle F Ex-learns every $f \in S$ whenever F is a list of S, i.e., $S = \{F_0, F_1, \dots\}$ where F_x is the function given by $F_x(y) = F(x,y)$. The most common inference classes as the class REC of all recursive functions and the class $\text{REC}_{0,1}$ of all $\{0,1\}$-valued recursive functions are in Ex[List], but there is also some $S \notin$ Ex[List]. Furthermore every class is in NV[List] but PEx[List], Fin[List] and PFin[List] are weaker than PEx[List].

Predictor: A class S is in Ex[Predictor] iff there is a machine M such that M equipped with a function oracle P Ex-learns every $f \in S$ whenever P is a device which NV-learns all $f \in S$. Predictors are strictly weaker than lists, e.g., $\text{REC}_{0,1} \in$ Ex[List] $\Leftrightarrow$Ex[Predictor]. Interestingly this is one of the few cases in inductive inference where a criterion fails for $\text{REC}_{0,1}$ but succeeds for REC, i.e., REC $\in$ Ex[Predictor]. By definition, predictors are omniscient for the criterion NV but on the other hand they are useless for the criteria PFin, Fin and PEx, i.e., anything learned with a predictor under one of these criteria can also be learned without any additional help under the same criterion.

Classifier: A class S is in Ex[Classifier] iff there is a machine M such that M equipped with a function oracle C Ex-learns every $f \in S$ whenever C is a one-sided classifier for S. A one-sided classifier C converges on all $f \in S$ to 1, i.e., $(\forall^\infty \sigma \preceq f)\,[C(\sigma) = 1]$, and does not converge to 1 on every (also nonrecursive) $f \notin S$, i.e. $(\exists^\infty \sigma \preceq f)\,[C(\sigma) = 0]$. Classifiers allow to Ex-learn and NV-learn the classes REC and $\text{REC}_{0,1}$ but they are not omniscient for these criteria. For Fin, PEx and PFin they are useless.

Martingale: A class S is in Ex[Martingale] iff there is a machine M such that M equipped with a function oracle m Ex-learns every $f \in S$ whenever m is a martingale succeeding on S. A martingale is a total function with positive rational values such that $m(\lambda) = 1$ and for each σ there is a rational number q with $0 \leq q < m(\sigma)$ and a prediction a such that $m(\sigma a) = m(\sigma) + q$ and $m(\sigma b) = m(\sigma) - q$ for all $b \neq a$. It turns out that martingales are useless for all considered learning criteria.

Inside Given Degrees: The oracles in this notion are (other than the previous ones) independent of S. A class S is robust learnable inside a given degree $\mathbf{a}$ of oracles iff there is a machine M which Ex-learns every $f \in S$ with any oracle $A \in \mathbf{a}$. It is shown that for all common notions of degrees (Turing, tt, wtt, btt, m) except the notion of 1-degrees this kind of additional information allows only to learn classes which can already be learned without access to any oracle.

Now the concepts are presented in detail each in one section starting with the notion of learning inside given degrees.

2 Robust Learning inside given Degrees

For a given oracle A, the Turing degree of A is the collection of all oracles B which have the same computational complexity as A, i.e., which are Turing equivalent to A. There are refinements of the notion of a Turing degree such as m-degree and 1-degree: A set A is m-reducible to B iff there is a recursive function f such that $A(x) = B(f(x))$ for all x. If this f is furthermore one-to-one, then A is 1-reducible to B. A and B are called m-equivalent, i.e., A and B have the same m-degree, if A is m-reducible to B and B is m-reducible to A. Similarly 1-equivalence and 1-degrees are defined. Odifreddi [21, Chapter VI] gives an overview on these and other degrees. The following theorem states that robust learning from an m-degree does not help. The same result also holds for the degrees given by the reductions btt, tt, wtt and Turing as defined in [21] since each such degree is the union of several m-degrees.

Theorem 2.1 *Assume that a single machine M* Ex[B]*-learns* (NV*-learns*) *S via access to oracle B for any B in the m-degree of A. Then S can be* Ex*-learned* (NV*-learned*) *without any oracle.*

Proof (a) for Ex-Learning Let $S \in \text{Ex}[B]$ via M^B for all oracles B in the m-degree of A. W.l.o.g. M is total also for the oracles outside the m-degree of A and $M(\sigma)$ is computed with oracle access only below $|\sigma|$ – these conditions can be satisfied via delaying mind changes [10, Note 2.14]. For any function f let $M^\alpha(f)$ abbreviate $M^\alpha(f(0)f(1)\ldots f(|\alpha|))$. There are two cases:

(I) There is a function $f \in S$ with $(\forall \alpha)\,(\exists \beta \succeq \alpha)\,[M^{\beta}(f) \neq M^{\alpha}(f)]$. Now it is possible to compute inductively binary strings $\alpha_0, \alpha_1, \ldots$ such that for each n and $a_0, a_1, \ldots, a_n \in \{0,1\}$ there are α, β with $a_0\alpha_0 a_1\alpha_1 \ldots a_n \preceq \alpha \preceq \beta \preceq a_0\alpha_0 a_1\alpha_1 \ldots a_n\alpha_n$ and $M^{\alpha}(f) \neq M^{\beta}(f)$. Here α_n is produced by concatenating strings γ_k for $k = 0, 1, \ldots, 2^{n+1} \Leftrightarrow 1$ where the γ_k are defined inductively: if $a_0 a_1 \ldots a_n$ is the binary representation of k and if $\alpha = a_0\alpha_0 a_1\alpha_1 \ldots a_n\gamma_0\gamma_1 \ldots \gamma_{k-1}$ then γ_k is the first string found which enforces $M^{\alpha}(f) \neq M^{\beta}(f)$ for $\beta = \alpha\gamma_k$.

It follows that M does not converge for any oracle of the form $a_0\alpha_0 a_1\alpha_1 \ldots$, in particular not for $B = A(0)\alpha_0 A(1)\alpha_1 \ldots$ which is m-equivalent to A. So the case (I) does not hold.

(II) For each function $f \in S$ there is an α with $M^{\beta}(f) = M^{\alpha}(f)$ for all $\beta \succeq \alpha$. Now the Ex-learner N for S works as follows: On input $f(0)f(1)\ldots f(n)$, N searches for the first string α (according to some enumeration of all strings) such that $M^{\beta}(f(0)f(1)\ldots f(|\beta|)) = M^{\alpha}(f(0)f(1)\ldots f(|\alpha|))$ for all strings $\beta \succeq \alpha$ of length up to n and outputs $M^{\alpha}(f(0)f(1)\ldots f(|\alpha|))$.

Some first string α satisfies the condition at (II) for the given function f and thus N converges to the value $M^{\alpha}(f)$. Since there is some oracle B in the m-degree of A with $B \succeq \alpha$, M^{B} converges also to $M^{\alpha}(f)$ and $M^{\alpha}(f)$ is the correct value. So N infers S without the help of any oracle.

(b) for NV-Learning Here $M^{\alpha}(f(0)f(1)\ldots f(n))\downarrow = y$ means that the oracle Turing machine queries only within *dom*(α) and converges to the output y. Again there is a case-distinction.

(I) $(\exists f \in S)\,(\forall \alpha)\,(\forall n)\,(\exists m > n)\,(\exists \beta \succ \alpha)\,[M^{\beta}(f(0)f(1)\ldots f(m))\downarrow \neq f(m+1)]$. As in the Ex-case it is possible to construct a computable sequence $\alpha_0, \alpha_1, \ldots$ such that M makes infinitely many mistakes during the attempt to NV-learn f for any oracle of the form $a_0\alpha_0 a_1\alpha_1 \ldots$ and thus there is an oracle which is m-equivalent to A on which M does not succeed to NV-learn f. So this case does not hold.

(II) $(\forall f \in S)\,(\exists \alpha)\,(\exists n)\,(\forall m > n)\,(\forall B \succ \alpha)\,[M^{B}(f(0)f(1)\ldots f(m))\downarrow = f(m+1)]$. Here the learning-algorithm is a bit different to that of the Ex-case but has the same basic idea. Note that for every input $f(0)f(1)\ldots f(n)$ and for every α there is some $\beta \succeq \alpha$ such that $M^{\beta}(f(0)f(1)\ldots f(n))\downarrow$ since some oracle in the m-degree of A extends β. The new inference machine N tries always to extrapolate B from a finite amount of information α in the just indicated way and crosses out every α which once produced an error via moving it into a book-keeping set C.

> Let $\alpha \notin C$ be the first binary string within a given enumeration which is not already crossed out. Now let
>
> $$N(f(0)f(1)\ldots f(n)) = M^{\beta}(f(0)f(1)\ldots f(n))$$
>
> for the first $\beta \succeq \alpha$ where this computation terminates within $|\beta|$ computation steps. If it turns out later (when the input $f(0)f(1)\ldots f(n)f(n+1)$ is processed) that this prediction was wrong then α is crossed out and C is replaced by $C \cup \{\alpha\}$.

Note that at every stage of the algorithm only finitely many strings are crossed out and that the α in the algorithm will either converge to some α satisfying (II) and therefore make almost always correct predictions or remain at some α before which hazardedly abstains from making wrong predictions and so keep this wrong α (which then of course does not matter). ▮

For the criteria Fin, PEx and PFin the same result holds also with 1-degrees in place of m-degrees.

Theorem 2.2 *Assume that a single machine M* PEx$[B]$*-learns S via access to oracle B for any B in the 1-degree of A. Then S can be* PEx*-learned without any oracle. The same result holds also for the criteria* Fin *and* PFin*.*

Proof First it is necessary to note that each finite binary string can be extended to a set in the 1-degree of A since A is not recursive and therefore infinite and coinfinite – otherwise one could fix A and replace queries to the oracle by computations. Now let S be PEx-learnable via uniform access to some oracle in the 1-degree of A via a machine M. The set

$$E = \{M^B(\sigma) : B \equiv_1 A \text{ and } \sigma \in \mathbb{N}^*\} = \{e : (\exists \beta \in \{0,1\}^*)\,(\exists \sigma \in \mathbb{N}^*)\,[M^\beta(\sigma) = e]\}$$

is an enumerable set of indices: since M uses for any output only a finite prefix of B, the search can go over all binary strings instead over all oracles 1-equivalent to A. Since any such string can be extended to an oracle 1-equivalent to A, each index in E is an index of a total recursive function. On the other hand, E contains all guesses $M^A(f(0)f(1)\dots f(n))$ for each $f \in S$. Since M learns S from oracle A, E contains for each $f \in S$ and index. Thus E is an enumerable set containing only indices of total recursive functions and for each function in S there is an index in E. It follows that S is PEx-learnable.

The proofs for the criteria Fin and PFin are based on the same idea. For each σ define – similarly to above – the sets

$$\begin{aligned} E(\sigma) &= \{M^\beta(\sigma) : \beta \in \{0,1\}^*\} \\ G(\sigma) &= \cup_{\tau \preceq \sigma} E_{|\sigma|}(\tau) \end{aligned}$$

where the $E_s(\tau)$ are a recursive enumeration of the $E(\tau)$ uniform in τ. The algorithm outputs "?" until it reaches some $\sigma \preceq f$ such that $G(\sigma)$ is not empty. Then the algorithm outputs some $e \in G(\sigma)$ and abstains from any mind change. This first guess is computed relative some finite binary string β and since some $B \equiv_1 A$ extends β, the output must be a correct index for f provided that $f \in S$. Furthermore in the case PFin e has to be a total index, also if σ does not belong to any $f \in S$. So again it follows that uniform access to 1-degrees does not support learning for the criteria Fin and PFin. ▮

Some 1-degrees are also trivial for Ex-learning and NV-learning. For example the 1-degree of a cylinder A (which satisfies $A(\langle x,y\rangle) = A(\langle x,0\rangle)$ for all pairs $\langle x,y\rangle$). But if A is sufficiently thin then the class REC of all recursive functions can be Ex-learned and NV-learned uniformly relative to every $B \equiv_1 A$ by a single machine M.

Theorem 2.3 *If the principal function p_A of A dominates every recursive function $f \in$ REC then there is a machine M which* Ex*-learns* REC *relative to any oracle B in the 1-degree of A. The same holds for* NV*.*

Proof Recall that the principal function p_A of A assigns to each x the x-th element of A. It can be shown that for each $B \equiv_1 A$, the principal function p_B of B also dominates every recursive function: $B = \{f(x) : x \in A\}$ for some recursive bijection f and if p_B would not dominate the recursive function g then p_A would also not dominate the recursive function $n \to \max\{f(m) : m \leq g(n)\}$.

Now the following algorithm M^B Ex-learns all recursive functions g: On input $\sigma \preceq g$ of length n, M^B first computes $x = p_B(n)$. Then M^B searches for the least e such that $\varphi_e(y)\downarrow = \sigma(y)$ within x computation steps for all $y \in dom(\sigma)$. If M^B finds such an e below n then M^B outputs this program e. Otherwise M^B outputs the symbol "?" to indicate that M^B could not make up its mind because of either too few data or too few computation time.

M^B converges to the minimal index e of g: Since the principal function p_B dominates the computation time of φ_e, the learner M^B outputs almost always either e or an index below e. The second case only occurs finitely often because there are only finitely many indices $i < e$ and each φ_i either diverges or computes a value different form f on some x_i. So for all $x > x_0 + x_1 + \ldots + x_{i-1}$, M^B does no longer output a value below e and thus converges to e.

The modification from Ex-learning to NV-learning is that M^B in place of outputting e simulates $\varphi_e(n+1)$ for x computation steps and outputs the result if it is found within x steps. Otherwise it outputs 0. Since p_B dominates the computation-time of φ_e whenever φ_e is total (and in particular equals f) the procedure predicts almost always every recursive function. Note that M must be total only for oracles $B \equiv_1 A$ and may diverge on others, in particular on oracles represented by finite sets. ∎

This proof gives the nice (and already well-known) fact that whenever a dominating function can be computed from the oracle then REC can be learned under the criterion Ex using this dominating function. This function needs not to be the same for all permitted oracles but each permitted oracle must give a dominating function via the same algorithm. The construction will be used in several proofs below.

3 Lists

Angluin [2] discovered that it is very much easier to learn a class of languages if it is a uniformly recursive family of functions (in her case: sets) whose index is known to the learner. Jantke [16] introduced within this model the intensively studied notions of monotonic inference and Zeugmann [28] gives an overview on these studies.

In the present work such a uniformly recursive computation procedure is replaced by an oracle which consists of a list of all functions in S. It is investigated how much such an oracle supports learning.

For a given array F let F_x denote the function $F_x(y) = F(x,y)$. Such an array F is a list for a class S iff $S = \{F_x : x \in \mathbb{N}\}$, so a list for S contains just all functions in S (but no nonmembers of S). First it is shown that some famous classes can be learned using a list.

A folklore result is that every uniformly recursive class can be learned w.r.t. its enumeration as hypothesis space. Some anonymous referee of the European Conference on Computational Learning Theory 1997 pointed out to the authors that this proof transfers to the setting of learning lists: if the entries to the rows of the list are used as hypothesis space then every class S can be learned in the limit with help of a list. Furthermore Case, Jain and Sharma [8] introduced learning w.r.t. limiting programs as a space of hypothesis and showed that they increase the learning power. This is still true for learning with lists as additional information. Nevertheless in the present work only the restricted version is considered where the learner still has to use the given acceptable numbering φ_e as hypothesis space for learning from lists under the criteria Ex, PEx, Fin and PFin.

Theorem 3.1 *The classes* REC *of all recursive functions,* $\mathrm{REC}_{0,1}$ *of all* $\{0,1\}$*-valued recursive functions,* $S_0 = \{f : (\exists e)\,[\varphi_e = f \wedge 0^e 1 \preceq f]\}$ *of all self describing functions,* $S_1 = \{f : (\forall^\infty x)\,[f(x) = 0]\}$ *of all functions with "finite support" and* $S_0 \cup S_1$ *are in* Ex[List]*, i.e., they are learnable in the limit from a list.*

Proof S_0 and S_1 are in Ex, so it remains to show the other three results. The proof for REC is based on the fact that every high oracle allows to infer all recursive functions [1] and the result for $\mathrm{REC}_{0,1}$ uses a construction of Jockusch [17].

If F is a list for REC then the function $h(x) = F_0(x)+F_1(x)+\ldots+F_x(x)$ dominates each F_y and therefore all recursive functions. Arguing as in Theorem 2.3, REC can be Ex-learned using the dominating function h obtained from the given list F of REC.

Similarly it is shown that $S_0 \cup S_1 \in$ Ex[List]. As above, a function h is constructed which dominates every function in $S_0 \cup S_1$. This function indeed dominates all recursive functions and thus enables to learn every subset of REC, in particular $S_0 \cup S_1$. So the domination property remains to be shown:

Note that a self describing function codes its index, i.e., there is an e such that $0^e 1 \preceq f$ and $f = \varphi_e$. For any total recursive function φ_e let $\varphi_{s(e,a)}(x) = 0$ for $x < a$, $\varphi_{s(e,a)}(a) = 1$ and $\varphi_{s(e,a)}(x) = \varphi_e(x)$ for $x > a$. The recursion-theorem states that there is an a with $\varphi_{s(e,a)} = \varphi_a$, so one of these functions is self describing and in $S_0 \cup S_1$. Thus h dominates this φ_a and also the finite variant φ_e of φ_a.

The case $\mathrm{REC}_{0,1}$ is more difficult, since no dominant function can be computed from a list of $\mathrm{REC}_{0,1}$. But the following method from [17] can be applied: Let ψ be a

$\{0,1\}$-valued function which has no recursive extension. Now define via dovetailing

$$\varphi_{g(i)}(x) = \begin{cases} 0 & \text{if } \varphi_i(y) \text{ converges for all } y \leq x; \\ \psi(x) & \text{if } \psi(x) \text{ converges before the condition above is satisfied;} \\ \uparrow & \text{otherwise.} \end{cases}$$

The function $\varphi_{g(i)}$ has a recursive $\{0,1\}$-valued extension iff φ_i is total. Now the inference-algorithm always outputs the i from the least pair $\langle i,j \rangle$ such that the input $f(0)f(1)\ldots f(x)$ is compatible with $\varphi_{i,x}$ and that $\varphi_{g(i),x}$ is extended by F_j. Such a pair $\langle i,j \rangle$ exists, since each $\{0,1\}$-valued total recursive function has an index i and then the function $\varphi_{g(i)}$ is also total and recursive and equals some F_j. Furthermore all false pairs $\langle i,j \rangle$ are thrown out since either $\varphi_i(x)\downarrow \neq f(x)$ for some x or $\varphi_{g(i)}$ has no extension within the list of all $\{0,1\}$-valued recursive functions and in particular differs from F_j. ∎

Theorem 3.2 *There are two classes $S_2, S_3 \in$ PFin[List] such that their union $S_2 \cup S_3$ and their difference $S_2 \Leftrightarrow S_3$ are not in* Ex[List].

Proof Let S_3 contain all constant functions. The class S_2 is defined using a construction from [18, Theorem 7.1]. This theorem shows that there is a family $\varphi_{g(i)}$ and an array A of low Turing degree such that

- $range(\varphi_{g(i)}) = \{0,1\}$ and $0^i1 \preceq \varphi_{g(i)}$;
- For all i there is at most one x with $\varphi_{g(i)}(x)\uparrow$;
- A_i extends $\varphi_{g(i)}$ and is recursive;
- The class $S_4 = \{A_i : i \in \mathbb{N}\}$ is not Ex-learnable relative to A.

Now let S_2 contain all functions in S_4 plus all constant functions of the form $f(x) = \langle i,j \rangle + 2$ where $(\forall j \geq i)\,[\varphi_{g(i)}(j)\downarrow]$. The following four observations hold:

- $S_2 \in$ PFin[List]: Learning a function $f \in S_2$, the learner M checks whether $f(0) > 1$. If so, the function is a constant function and M outputs a total index for it. If not, M outputs "?" until an i is known with $0^i1 \preceq f$ and a j is found such that the function h with $h(0) = \langle i,j \rangle + 2$ is in the list. Then the function $\varphi_{g(i)}$ is total beyond j and M outputs the index $g'(i,j)$ of

$$\varphi_{g'(i,j)}(x) = \begin{cases} f(x) & \text{if } x \leq j; \\ \varphi_{g(i)}(x) & \text{otherwise;} \end{cases}$$

where f is the function on the input. Only its first j values are necessary, but the j can depend on the concrete form of the list. By the choice of the constant functions in S_2 and the fact that the list contains exactly those functions which belong to S_2, the algorithm always outputs exactly one total program and this one is correct if the data belongs to some $f \in S_2$.

- $S_3 \in$ PFin[List]: This follows directly from the fact, that a constant function f is known after seeing the value $f(0)$.
- $S_2 \Leftrightarrow S_3 \notin$ Ex[List]: $S_4 = S_2 \Leftrightarrow S_3$ and A is a list for S_4. By the choice of A and S_4, the class S_4 can not be learned with A-oracle, in particular not under the criterion Ex[List] since the list presented can be exactly A.
- $S_2 \cup S_3 \notin$ Ex[List]: There is also an A-recursive array for $S_4 \cup S_3 = S_2 \cup S_3$. Since $S_4 \notin$ Ex[A], the same holds for the superclass $S_2 \cup S_3$ and so this class can also not be learned with the help of a list. Indeed the point is that by the union the particular information, from where on a function $\varphi_{g(i)}$ is total, is overwritten.

These observations give directly the theorem. ∎

A direct corollary is, that the class S_2 can be learned under the criteria PFin[List], PEx[List], Fin[List] and Ex[List], but not under the criteria PFin, PEx, Fin or Ex. So lists are really a help for several learning criteria.

4 Predictors

Barzdins [5] and Blum and Blum [7] introduced the learning criterion NV where the learner has to interpolate the next value from the previous ones. In this section it is investigated to which extent such a predicting device can be uniformly translated into a learner for one of the other four criteria. Formally, a total device P is called a predictor for S iff

$$(\forall f \in S)\,(\exists x)\,(\forall y > x)\,[P(f(0)f(1)\dots f(y)) = f(y+1)],$$

i.e., if it predicts each function $f \in S$ at almost all places $y+1$ from the data $f(0), f(1), \dots, f(y)$. Any list can be turned into a predictor as follows: Let F be a list and define

$$P(a_0 a_1 \dots a_y) = \begin{cases} F_x(y+1) & \text{for the first } x \leq y \text{ with} \\ & F_x(0) = a_0, F_x(1) = a_1, \dots, F_x(y) = a_y; \\ 0 & \text{if there is no such } x \leq y. \end{cases}$$

This translation is not reversible: a predictor may also predict functions outside the class S to be learned and so hide the information which functions belong to S and which not. The translation from lists to predictors has the following immediate application.

Theorem 4.1 $S \in$ NV[List] *for all* $S \subseteq$ REC*, i.e., lists are omniscient for* NV.

While lists help under all inference-criteria, predictors are no longer helpful for PFin, Fin and PEx. This is due to the fact, that every finite modification of a predictor is again a predictor and so the inference machine has to fulfill the requirements for these three learning criteria also under all finite modifications of the predictors. Then it follows by an easy adaption of the proof of Theorem 2.2 that the criteria are not supported by predictors as additional information.

Theorem 4.2 PEx[Predictor] = PEx, Fin[Predictor] = Fin *and* PFin[Predictor] = PFin.

While predictors are omniscient for NV-learning (by definition) and trivial for Fin, PFin and PEx, they are intermediate for Ex-learning. In particular the natural class REC is learnable by predictor while the also natural class $\mathrm{REC}_{0,1}$ is not.

Theorem 4.3 REC $\in$ Ex[Predictor] *and* $\mathrm{REC}_{0,1} \notin$ Ex[Predictor].

Proof The first result is due to the fact that a dominating function can be computed using a predictor. For each σ the predictor P defines inductively a total function f_σ via extending the string σ by P:

$$f_\sigma(n) = \begin{cases} \sigma(n) & \text{for } n \in dom(\sigma). \\ P(f_\sigma(0) f_\sigma(1) \dots f_\sigma(n-1)) & \text{for } n \notin dom(\sigma). \end{cases}$$

Let $\sigma_0, \sigma_1, \dots$ be an enumeration of all strings. Now

$$g(x) = f_{\sigma_0}(x) + f_{\sigma_1}(x) + \dots + f_{\sigma_x}(x)$$

is uniformly recursive in the given predictor P and dominates every recursive function. As in Theorem 2.3 it follows that REC can be learned in the limit using this g obtained from P.

The construction fails in the case of $\mathrm{REC}_{0,1}$. Indeed there is a low oracle predicting all $\{0,1\}$-valued functions. This oracle gives a predictor, but the predictor is not sufficiently powerful to learn $\mathrm{REC}_{0,1}$ in the limit since this requires a high oracle [1, 10]. ∎

5 Classifiers

A one-sided classifier C [27] assigns to every string σ a binary value. C classifies S iff

$$(\forall f)\,[f \in S \Leftrightarrow (\forall^\infty \sigma \preceq f)\,[C(\sigma) = 1]\,].$$

Note that the quantifier also ranges over nonrecursive functions, i.e., C must not converge to 1 on any nonrecursive function. Two-sided classification requires in addition, that C converges on the functions outside S to 0. So one-sided classes are the $\Sigma_2^{(s)}$-classes and two-sided classes are the $\Delta_2^{(s)}$-classes according to the notation of Rogers [20, Chapter 15.1]; they are also called Σ_2^0-classes and Δ_2^0-classes. There are classes of recursive functions, which have no two-sided classifier, even not relative to any oracle, e.g., REC and S_1 [6, 20, 27]. On the other hand, every countable class has a (not necessarily recursive) one-sided classifier. So the concept of one-sided classification is more suitable. One-sided classifiers still do not help the criteria PEx, Fin and PFin via the same argument as in the case of 1-degrees and predictors. So the following theorem is stated without proof, since the one for Theorem 2.2 could be adapted with minor changes.

Theorem 5.1 PEx[Classifier] = PEx, Fin[Classifier] = Fin *and* PFin[Classifier] = PFin.

Reliable inference means, that a machine converges on a function f iff it learns this function. The next theorem shows, that every class S learnable in the limit using a classifier can even be learned reliably using this classifier.

Theorem 5.2 Ex[Classifier] = REx[Classifier].

Proof The criterion Ex is more general than REx, thus it is sufficient to show only the direction Ex[Classifier] $\rightarrow$ REx[Classifier]. Let $S \in$ Ex[Classifier] via a classifier C and an inference-machine M. Furthermore let *pad* be an injective padding-function such that $\varphi_{pad(i,j)} = \varphi_i$ for all i and j. The new REx-learner N uses *pad* to enforce a mind change whenever C takes the value 0 (let $M(\lambda) = a$ for some $\varphi_a \notin S$):

$$N(\sigma) = pad(M(\sigma), \tau) \text{ for the longest } \tau \preceq \sigma \text{ with } C(\tau) = 0 \vee M(\tau) \neq M(\sigma).$$

N is a reliable inference algorithm for S: If $f \in S$, then C converges on f to 1 and M converges to some index e with $\varphi_e = f$. There is a smallest $\tau \preceq f$ which satisfies $C(\sigma) = 1$ and $M(\tau) = M(\sigma) = e$ for all σ with $\tau \preceq \sigma \preceq f$. Thus the N converges to $pad(e, \tau)$. If f is not in S then there are infinitely many $\tau \preceq f$ with $C(\tau) = 0$. For all these τ, N takes the value $pad(M(\tau), \tau)$ and all these values are different, i.e., N does not converge. It follows that S is learned via the reliable machine N. ▮

The next Theorem uses – as the corresponding Theorem 3.1 for lists – Jockusch's construction [17] in order to show that every class containing all $\{0,1\}$-valued self describing functions is learnable using a classifier.

Theorem 5.3 *If* $\text{REC}_{0,1} \cap S_0 \subseteq S$ *then* $S \in$ Ex[Classifier].

Proof Let ψ be a $\{0,1\}$-valued partial recursive function without any total recursive extension. By Jockusch's construction [17] there is a recursive function g such that the functions $\varphi_{g(n,m)}$ satisfy the following requirements:

- $0^m 1 \preceq \varphi_{g(n,m)}$ and $range(\varphi_{g(n,m)}) = \{0,1\}$;
- If φ_n is total so are all functions $\varphi_{g(n,m)}$;
- If φ_n is partial then $\varphi_{g(n,m)}(x)\downarrow = \psi(x)$ for almost all $x \in dom(\psi)$.

By the recursion theorem with parameters [26, II.3.5] there is a recursive function h such that $\varphi_{h(n)} = \varphi_{g(n,h(n))}$ for all n. Note that every function $\varphi_{h(n)}$ is self describing and that $\varphi_{h(n)}$ is total iff φ_n is. Furthermore, $\varphi_{h(n)}$ is either total or has no total recursive extension at all.

It can be computed effectively in the limit from any classifier C for S whether the function $\varphi_{h(n)}$ is total or not: To see this let σ_s be the longest prefix of the function $\varphi_{h(n)}(0)\varphi_{h(n)}(1)\ldots$ such that all its values are calculated within s stages. An extension $\tau \succeq \sigma_s$ is said to be consistent with $\varphi_{h(n)}$ $(\varphi_{h(n),s})$ iff for every $x \in dom(\tau) \cap dom(\varphi_{h(n)})$ $(x \in dom(\tau) \cap dom(\varphi_{h(n),s}))$, the values $\tau(x)$ and $\varphi_{h(n)}(x)$ coincide. Consider the following sequence, which is uniformly recursive in the parameters s and n.

$$a_s = \begin{cases} 1 & \text{if there is an extension } \tau \in \{0,1\}^{s+1} \text{ of } \sigma_s \\ & \text{which is consistent with } \varphi_{h(n),s} \text{ and which satisfies} \\ & C(\eta) = 1 \text{ for all } \eta \text{ with } \sigma_s \preceq \eta \preceq \tau; \\ 0 & \text{otherwise.} \end{cases}$$

If φ_n is total, then $\varphi_{h(n)}$ is total and C converges on $\varphi_{h(n)}$ to 1. If s is sufficiently large, then σ_s is sufficiently long and the string $\tau = \varphi_{h(n)}(0)\varphi_{h(n)}(1)\ldots\varphi_{h(n)}(s)$ satisfies the requirements. τ extends σ_s. τ is obviously consistent with $\varphi_{h(n),s}$. $C(\eta) = 1$ for all η between σ_s and τ. So the a_s converge to 1.

If φ_n is partial, then $\varphi_{h(n)}$ has no recursive extension and C does not converge to 1 on any f extending $\varphi_{h(n)}$. Let σ be the longest prefix of $\varphi_{h(n)}$ such that all its values are defined. Now consider the following binary tree T_σ:

> A binary string τ is in T_σ either if $\tau \preceq \sigma$ or if τ extends σ, τ is consistent with $\varphi_{h(n)}$ and $M(\eta) = 1$ for all η between σ and τ.

Since C does not converge to 1 on any f extending $\varphi_{h(n)}$, the binary tree T_σ does not have any infinite branch f. So the tree T_σ is finite and there is some x bounding the length of every string in T_σ. Let $s > x$ be a stage such that for all $y \leq x$ the value $\varphi_{h(n)}(y)$ is calculated within s steps whenever it is defined. Now $\sigma_s = \sigma$. Furthermore whenever $\tau \notin T_\sigma$ there is either some η between σ_s and τ with $C(\eta) = 0$ or there is some y with $\tau(y)\downarrow \neq \varphi_{h(n)}(y)\downarrow$. If the first case does not hold, then it follows by the construction of T_σ that the second case holds for some $y \leq x$. So τ is also inconsistent with $\varphi_{h(n),s}$. It follows that $a_s = 0$ since a_s is not 1 via any $\tau \in \{0,1\}^{s+1}$. The a_s converge to 0 in this second case.

So it can be computed in the limit using C which functions φ_n are total and this computation does not depend on the particular form of C. The learner M uses this information for the following construction: At every stage M outputs the first e which is at stage $|\sigma|$ assumed to be total and for which $\varphi_{e,|\sigma|}$ is consistent with the data σ seen so far, i.e., which satisfies $\varphi_{e,|\sigma|}(x) = \sigma(x)$ for all $x \in dom(\varphi_{e,|\sigma|}) \cap dom(\sigma)$. ∎

This result also holds for NV-learning (after modifying the last part of the proof above).

Any list can be transferred into a one-sided classifier: The classifier determines for every σ the smallest index $e \leq |\sigma|$ such that $\sigma \preceq F_e$. If this index for σa is greater than that for σ or if σa does not have such an index, then the classifier outputs 0. Otherwise it outputs 1. The algorithm can be easily verified. So everything which can be learned from a classifier can also be learned from a list.

Theorem 5.4 Ex[Classifier] $\subseteq$ Ex[List].

So both concepts Ex[Classifier] and Ex[Predictor] are weaker than Ex[List]. The next theorem shows that they are incomparable and thus both concepts are strictly weaker than Ex-learning from a list.

Theorem 5.5 Ex[Classifier] *and* Ex[Predictor] *are incomparable.*

Proof Since $\mathrm{REC}_{0,1} \in$ Ex[Classifier] $\Leftrightarrow$Ex[Predictor], only the other noninclusion remains to be shown: Ex[Predictor] $\not\subseteq$ Ex[Classifier]. The class to witness this noninclusion is the union of the following two classes:

- The class S_4 from Theorem 3.2.
- The class $S_5 = \{\Phi_e : e \geq 0\} \cap REC$ of all total step-counting functions. Thereby $\Phi_e(x)$ is defined as the time to compute $\varphi_e(x)$ if $\varphi_e(x) \downarrow$; otherwise $\Phi_e(x)$ is undefined.

The class S_4 has a list relative to some low oracle A and therefore it also has a classifier relative to A. S_5 even has a recursive one-sided classifier C: The uniform graph $G = \{(x, y, e) : \Phi_e(x)\downarrow = y\}$ of all step-counting functions is decidable. There is a one-sided computable classifier C such that $C(f(0)f(1)\dots f(n)) = 0$ iff $n = 0$ or

$$a_n = \max\{i \leq n : (\forall j \leq i)\,(\exists x < n)\,[(x, f(x), j) \notin G]\} > a_{n-1}$$

where a_{n-1} is defined analogously; $C(f(0)f(1)\dots f(n)) = 1$ otherwise. So the union of $S_4 \cup S_5$ has a classifier of degree A, but as already mentioned in Theorem 3.2, S_4 and every superclass can only be learned from oracles of high degree. Therefore $S_4 \cup S_5 \notin$ Ex[Classifier].

On the other hand, if M is a predictor for S_5 then M must predict the computation-time for each function φ_e almost everywhere. So uniformly in M some function dominating all computation-times can be calculated and using this function it is possible to infer every recursive function – in particular every function in $S_4 \cup S_5$. ▮

A direct corollary is, that whenever M is a predictor for S_5, then a dominating and therefore nonrecursive function can be computed relative to M. In particular S_5 has no predictor which uses only the computable above constructed classifier as oracle and thus $S_5 \notin$ NV[Classifier].

Theorem 5.6 *The class S_5 of all total step-counting functions is not in* NV[Classifier].

6 Martingales

A martingale calculates the gambling-account of someone who always tries to predict the next value of a function. In each round the gambler places an amount q on some number a, i.e., for each string σ there is a rational number q, $0 \leq q < m(\sigma)$, such that $m(\sigma a) = m(\sigma) + q$ for some a and $m(\sigma b) = m(\sigma) \Leftrightarrow q$ for all $b \neq a$. The gambler wins on a function f iff the martingale takes on prefixes of f arbitrary large amounts of money. m is a martingale for S iff m wins on every function $f \in S$. The interested reader finds more on martingales in Schnorr's book [24].

Theorem 6.1 *If* $S \in$ Ex[Martingale] *then* $S \in$ Ex. *The same holds for all other inference criteria. In short: martingales do not help.*

Proof There is a martingale $m \leq_T A$ for some 1-generic set $A \leq_T K$ which wins on every recursive function – indeed every set A of hyperimmune degree is suitable. Let $g \leq_T A$ be a monotone function which is not dominated by any recursive function. Now the strategy of m is the following:

> Let σ be the input, $x = |\sigma|$ and $a = f(x)$ be the value to be predicted. Now look for the least $e \leq x$ such that $\varphi_e(y)$ converges to $\sigma(y)$ for $y = 0, 1, \ldots, x \Leftrightarrow 1$ and $\varphi_e(y)$ also converges to some value a within $g(x)$ steps. If there are such an e and a then bet $q = \frac{m(\sigma)}{2}$ on a and otherwise do not bet $(q = 0)$.

This martingale succeeds: Let e be the least index of f. g is not dominated by h where $h(x)$ is the time to compute all values $\varphi_e(0), \ldots, \varphi_e(x)$. There are even infinitely many x with $g(x) > h(3x + 3e)$. For these x, the martingale m bets for $y = x, x + 1, \ldots, 3x + 3e$ on either $\varphi_e(y)$ or $\varphi_j(y)$ for some $j < e$. It happens for each $j < e$ at most once that m bets on $\varphi_j(y)$ and $\varphi_j(y) \neq \varphi_e(y)$, so this phenomenon produces in total at most e wrong bets. On the other hand, $\varphi_e(y)$ is computed within $g(x) \leq g(y)$ steps and so whenever m takes no value $\varphi_j(y)$ with $j < e$ then it predicts the value $\varphi_e(y)$. So at least $2x + 2e$ of the predictions between x and $3x + 3e$ are correct and $m(f(0)f(1)\ldots f(3x + 3e)) \geq (\frac{9}{8})^{x+e}$. Since this holds for infinitely many x, m succeeds on g and so m succeeds on every recursive function.

If now $S \in$ Ex[Martingale] then S can also be learned via any oracle relative to which such a martingale exists. In particular S can be inferred relative to a low 1-generic oracle and thus S can be learned in the limit without any oracle [25]. So martingales do not help for learning in the limit. The same holds for learning under the criterion NV.

As in the case of predictors and classifiers, each finite part of any martingale can be extended to a martingale for S. The set of all such finite parts is enumerable and therefore the arguments from Theorem 2.2 can be used to show that martingales do also not help to learn under the criteria Fin, PFin and PEx. ∎

So martingales are on the bottom of the inclusion-structure of these four types of additional information as it is summarized in the following theorem.

Theorem 6.2 *The inclusion-structure of the four types of additional information with respect to the learning criteria* Ex *and* NV *are given by the following diagrams.*

Ex-learning

$$\begin{tikzcd}
& \text{List} & \\
\text{Predictor} \arrow[ur] & & \text{Classifier} \arrow[ul] \\
& \text{Martingale} = \text{Trivial} \arrow[ul] \arrow[uu] \arrow[ur] &
\end{tikzcd}$$

NV-learning

$$\begin{tikzcd}
\text{List} = \text{Predictor} = \text{Omniscient} \\
\text{Classifier} \arrow[u] \\
\text{Martingale} = \text{Trivial} \arrow[u]
\end{tikzcd}$$

For the criteria Fin*,* PFin *and* PEx *only lists provide some help while the other three types of additional information are trivial, i.e., do not increase the learning-power.*

Acknowledgment We would like to thank Arun Sharma and the anonymous referees of the European Conference on Computational Learning Theory 1997 for useful suggestions and comments.

References

[1] Lenny Adleman and Manuel Blum (1991): Inductive Inference and Unsolvability. Journal of Symbolic Logic 56:891–900.

[2] Dana Angluin (1980): Inductive Inference of Formal Languages from Positive Data. Information and Control 45:117–135.

[3] Dana Angluin (1987): Learning Regular Sets From Queries and Counterexamples. Information and Computation 75:87–106.

[4] Ganesh Baliga and John Case (1994): Learning with Higher Order Additional Information. Proceedings of the Fifth Workshop on Algorithmic Learning Theory (ALT) 64–75.

[5] Janis Barzdins (1971): Prognostication of automata and functions. Information Processing '71 (1) 81–84. Edited by C. P. Freiman, North-Holland, Amsterdam.

[6] Shai Ben-David (1992): Can Finite Samples Detect Singularities of Real-Valued Functions? Proceedings of the 24th Annual ACM Symposium on the Theory of Computer Science, Victoria, B.C., 390–399.

[7] Leonard Blum and Manuel Blum (1975): Towards a Mathematical Theory of Inductive Inference. Information and Control 28:125–155.

[8] John Case, Sanjay Jain and Arun Sharma (1992): On learning limiting programs. International Journal of Foundations of Computer Science 1:93–115.

[9] John Case, Susanne Kaufmann, Efim Kinber and Martin Kummer (1995): Learning Recursive Functions From Approximations. Proceedings of the second European Conference on Computational Learning Theory (EuroCOLT) 140-153.

[10] Lance Fortnow, William Gasarch, Sanjay Jain, Efim Kinber, Martin Kummer, Steven Kurtz, Mark Pleszkoch, Theodore Slaman, Robert Solovay and Frank

Stephan (1994): Extremes in the Degrees of Inferability. Annals of Pure and Applied Logic 66:231–276.

[11] Rusins Freivalds and Rolf Wiehagen (1979): Inductive inference with additional information. Elektronische Informationsverarbeitung und Kybernetik 15:179–185.

[12] William Gasarch and Mark Pleszkoch (1989): Learning via queries to an oracle. Proceedings of the Second Annual Conference on Computational Learning Theory (COLT) 214–229.

[13] Mark Gold (1967): Language Identification in the Limit. Information and Control 10:447–474.

[14] Sanjay Jain and Arun Sharma (1991): Learning in the Presence of Partial Explanations. Information and Computation 95:162–191.

[15] Sanjay Jain and Arun Sharma (1993): Learning with the Knowledge of an Upper Bound on Program Size. Information and Computation 102:118–166.

[16] Klaus-Peter Jantke (1991): Monotonic and Non-Monotonic Inductive Inference. New Generation Computing 8:349–360.

[17] Carl Jockusch (1972): Degrees in which recursive sets are uniformly recursive. Canadian Journal of Mathematics, 24:1092–1099.

[18] Martin Kummer and Frank Stephan (1993): On the Structure of Degrees of Inferability. Proceedings of the Sixth Conference on Computational Learning Theory (COLT) 117–126.

[19] Wolfgang Merkle and Frank Stephan (1996): Trees and Learning. Proceedings of the Ninth Conference on Computational Learning Theory (COLT) ACM-Press, pp. 270–279.

[20] Hartley Rogers, Jr. (1967): Theory of Recursive Functions and Effective Computability. McGraw-Hill Book Company, New York.

[21] Piergiorgio Odifreddi (1989): Classical Recursion Theory. North-Holland.

[22] Daniel Osherson, Michael Stob and Scott Weinstein (1986): Systems that Learn. Bradford – The MIT Press, Cambridge, Massachusetts.

[23] Daniel Osherson, Michael Stob and Scott Weinstein (1988): Synthesizing inductive expertise, Information and Computation 77(2):138–161.

[24] Claus Peter Schnorr (1971): Zufälligkeit und Wahrscheinlichkeit. Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1971.

[25] Theodore Slaman and Robert Solovay (1991): When Oracles Do Not Help. Proceedings of the Fourth Conference on Computational Learning Theory (COLT) 379–383.

[26] Robert Soare (1987): Recursively Enumerable Sets and Degrees. Springer-Verlag Heidelberg.

[27] Frank Stephan (1996): On One-Sided Versus Two-Sided Classification. Forschungsberichte Mathematische Logik 25 / 1996, Mathematisches Institut, Universität Heidelberg.

[28] Thomas Zeugmann (1993): Algorithmisches Lernen von Funktionen und Sprachen. Habilitationsschrift an der Technischen Hochschule Darmstadt.