On sample complexity for computational classification problems

Daniil Ryabko*
IDSIA, Galleria 2, CH-6928 Manno-Lugano, Switzerland
daniil@ryabko.net

April 25, 2006

Keywords: pattern recognition, classification, sample complexity, Kolmogorov complexity, computability analysis

Abstract

[1] In the statistical setting of the classification (pattern recognition) problem the number of examples required to approximate an unknown labelling function is linear in the VC dimension of the target learning class. In this work we consider the question of whether such bounds exist if we restrict our attention to computable classification methods, assuming that the unknown labelling function is also computable. We find that in this case the number of examples required for a computable method to approximate the labelling function not only is not linear, but grows faster (in the VC dimension of the class) than any computable function. No time or space constraints are put on the predictors or target functions; the only resource we consider is the training examples.

The task of classification is considered in conjunction with another learning problem — data compression. An impossibility result for the task of data compression allows us to estimate the sample complexity for pattern recognition.

*tel: +41 58 666 6660 fax: +41 58 666 6661
[1]This research was supported by the Swiss NSF grant 200020-107616
The main results were reported at ALT'05, [9].

1 Introduction

The task of classification (pattern recognition) consists in predicting an unknown label of some observation (or object). For instance, the object can be an image of a hand-written letter, in which case the label is the actual letter represented by this image. Other examples include DNA sequence identification, recognition of an illness based on a set of symptoms, speech recognition, and many others.

More formally, the objects are drawn independently from the object space X (usually $X = [0,1]^d$ or $\mathbb{R}^d$) according to some unknown but fixed probability distribution P on X, and labels are defined according to some function $\eta : X \to Y$, where $Y = \{0,1\}$. The task is to construct a function $\varphi : X \to Y$ which approximates η, i.e. for which $P\{x : \eta(x) \neq \varphi(x)\}$ is small, where P and η are unknown but examples $x_1, y_1, \dots, x_n, y_n$ are given; $y_i := \eta(x_i)$. In such a general setting finite-step performance guarantees are not possible; however, good error estimates can be obtained if η is known to belong to some (small) class $\mathcal{C}$. Thus, in the framework of statistical learning theory [10] it is assumed that the function η belongs to some known class of functions $\mathcal{C}$. The number of examples required to obtain a certain level of accuracy (or the *sample complexity* of $\mathcal{C}$) is linear in the VC-dimension of $\mathcal{C}$. How to select a class $\mathcal{C}$ is left to be specified for each learning problem separately.

In this work we investigate the question of whether finite-step performance guarantees can be obtained if we consider the class of computable (on some Turing machine) classification methods. To make the problem more realistic, we assume that the target function η is also computable. Two definitions of target functions are considered: they are either of the form $\{0,1\}^\infty \to \{0,1\}$ or $\{0,1\}^t \to \{0,1\}$ for some t (which can be different for different target functions).

We show that there are classes $\mathcal{C}_k$ of functions for which the number of examples needed to approximate the classification problem to a certain accuracy grows faster in the VC dimension of the class than any computable function (rather than being linear as in the statistical setting). In particular this holds if $\mathcal{C}_k$ is the class of all computable functions of length not greater than k, in which case k is a (trivial) upper bound of the VC dimension.

Importantly, the same negative result holds even if we allow the data to be generated "actively", e.g. by some algorithm, rather than just by some fixed probability distribution.

To obtain this negative result we consider the task of data compression:

an impossibility result for the task of data compression allows us to estimate the sample complexity for classification. We also analyze how tight the negative result is, and show that for some simple computable rule (based on the nearest neighbour estimate) the sample complexity is finite in k, under different definitions of computational patterning recognition task.

In comparison to the vast literature on classification relatively little attention had been paid to the "computable" version of the task. There is a track of research in which different concepts of computable learnability of functions on countable domains are studied, see [2]. A link between this framework and statistical learning theory is proposed in [7], where it is shown that for a uniform learnability finite VC dimension is required.

Another approach is to consider classification methods as functions computable in polynomial time, or under other resource constraints. This approach leads to many interesting results, but it usually considers more specific settings of a learning problem, such as learning DNFs, finite automata, etc. See [4] for an introduction to this theory and for references.

It may be interesting to observe the connection of the results for pattern recognition with another learning problem, sequence prediction. In one of its simplest forms this task is as follows: it is required to predict the next outcome of a deterministic sequence of symbols, where the sequence is assumed to be computable (is generated by some program). There is a predictor which can solve any such problem and the number of errors it makes is at most linear in the length of the program generating the sequence (see, e.g. [3], Section 3.2.3). Such a predictor is not computable. Trivially, there is no computable predictor for all computable sequences, since for any computable predictor a computable sequence can be constructed on which it errs at every trial, simply by reversing the predictions. Thus we have linear number of errors for non-computable predictor versus infinitely many errors for any computable one; whereas in pattern recognition, as we show, it is linear for a non-computable predictor versus growing faster than any computable function for any computable predictor.

2 Notation and definitions

A *(binary) string* is a member of the set $\{0,1\}^* = \bigcup_{i=0}^{\infty}\{0,1\}^n$. The length of a string x will be denoted by $|x|$, while x^i is the ith element of x, $1 \le i \le |x|$. For a set A the symbols $|A|$ and $\#A$ are used for the number of elements in

A. We will assume the lexicographical order on the set of strings, and when necessary will identify $\{0,1\}^*$ and $\mathbb{N}$ via this ordering, where $\mathbb{N}$ is the sets of natural numbers. The symbol log is used for $\log_2$. For a real number α the symbol $\ulcorner \alpha \urcorner$ is the least natural number not smaller than α. By computable functions we mean functions computable on a Turing machine with an input tape, output tape, and some working tapes, the number of which is supposed to be fixed throughout the paper.

All computable functions can be encoded (in a canonical way) and thus the set of computable functions can be effectively enumerated. Fix some canonical enumeration and define the *length* of a computable function η as $l(\eta) := |n|$ where n is the minimal number of η in such enumeration. For an introduction to the computability theory see, for example, [8].

From the set of all computable functions we are interested in labelling functions, that is, in functions which represent pattern recognition problems. In pattern recognition a labelling function is usually a function from the interval $[0,1]$ or $[0,1]^d$ (sometimes more general spaces are considered) to a finite space $Y := \{0,1\}$. As we are interested in computable functions, we should consider instead total computable functions of the form $\{0,1\}^\infty \to \{0,1\}$. However, since we require that labelling functions are total (defined on all inputs) and computable, it can be easily shown (e.g. with König's lemma [5]) that any such function never scans its input tape further than a certain position independent of the input. Thus apparently the smallest meaningful class of computable labelling functions that we can consider is the class of functions of the form $\{0,1\}^t \to \{0,1\}$ for some t. So, we call a partial recursive function (or program) η a *labelling function* if there exists such $t =: t(\eta) \in \mathbb{N}$ that η accepts all strings from $X_t := \{0,1\}^t$ and only such strings. (It is not essential for this definition that η is not a total function. An equivalent for our purposes definition would be as follows: a labelling function is any total function which outputs the string 00 on all inputs except on the strings of some length $t =: t(\eta)$, on each of which it outputs either 0 or 1.)

It can be argued that this definition of a labelling function is too restrictive to approximate well the notion of a real function. However, as we are after negative results (for the class of all labelling functions), it is not a disadvantage. Other possible definitions are discussed in Section 5, where we are concerned with tightness of our negative results. In particular, all the results hold true if a target function is any total computable function of the form $\{0,1\}^\infty \to \{0,1\}$.

Define the task of computational classification as follows. An (unknown) labelling function η is fixed. The objects $x_1, \ldots, x_n \in X$ are drawn according to some distribution P on $X_{t(\eta)}$. The labels y_i are defined according to η, that is $y_i := \eta(x_i)$.

A *predictor* is a family of functions $\varphi_n(x_1, y_1, \ldots, x_n, y_n, x)$ (indexed by n) taking values in Y, such that for any n and any $t \in \mathbb{N}$, if $x_i \in X_t$ for each i, $1 \leq i \leq n$, then the marginal $\varphi(x)$ is a total function on X_t. We will often identify φ_n with its marginal $\varphi_n(x)$ when the values of other variables are clear. Thus, given a *sample* $x_1, y_1, \ldots, x_n, y_n$ of labelled objects of the same size t a predictor produces a labelling function on X_t which is supposed to approximate η.

A *computable predictor* is a total computable function from $X_t \times Y \times \cdots \times X_t \times Y \times X_t$ to $\{0, 1\}$, where the arguments are assumed to be encoded into a single input in a certain fixed (simple canonical) way.

3 Setup

We are interested in what sample size is required to approximate a labelling function η.

For a (computable) predictor φ, a labelling function η and $0 < \varepsilon \in \mathbb{R}$ define

$$\delta_n(\varphi, \eta, \varepsilon) := \sup_{P_t} P_t\Big\{x_1, \ldots, x_n \in X_t : \\ P_t\big\{x \in X_t : \varphi_n(x_1, y_1, \ldots, x_n, y_n, x) \neq \eta(x)\big\} > \varepsilon\Big\},$$

where $t = t(\eta)$ and P_t ranges over all distributions on X_t (i.i.d. on X_t^n). As usual in PAC theory we have two probabilities here: consider the P_t-probability over a training sample of size n that the P_t-probability of error of a predictor φ exceeds ε; then take the supremum over all possible distributions P_t.

For $\delta \in \mathbb{R}$, $\delta > 0$ define the *sample complexity* of η with respect to φ as

$$N(\varphi, \eta, \delta, \varepsilon) := \min\{n \in \mathbb{N} : \delta_n(\varphi, \eta, \varepsilon) \leq \delta\}.$$

The number $N(\varphi, \eta, \delta, \varepsilon)$ is the minimal sample size required for a predictor φ to achieve ε-accuracy with probability $1 - \delta$ when the (unknown) labelling function is η, under all probability distributions.

With the use of statistical learning theory [10] we can easily derive the following statement

Proposition 1. *There exists a predictor φ such that*

$$N(\varphi, \eta, \delta, \varepsilon) \leq \frac{const}{\varepsilon} l(\eta) \log \frac{1}{\delta}$$

for any labelling function η and any $\varepsilon, \delta > 0$.

Observe that the bound is linear in the length of η.

In the next section we investigate the question of whether any such bounds exist if we restrict our attention to computable predictors.

Proof. The predictor φ is defined as follows. For each sample $x_1, y_1, \dots, x_n, y_n$ it finds a shortest program $\bar{\eta}$ such that $\bar{\eta}(x_i) = y_i$ for all $i \leq n$. Clearly, $l(\bar{\eta}) \leq l(\eta)$. Observe that the VC-dimension of the class of all computable functions of length not greater than $l(\eta)$ is bounded from above by $l(\eta)$, as there are not more than $2^{l(\eta)}$ such functions. Moreover, φ minimizes empirical risk over this class of functions. It remains to use the bound (see e.g. [1], Corollary 12.4) $\sup_{\eta \in \mathcal{C}} N(\varphi, \eta, \delta, \varepsilon) \leq \max\Big(V(\mathcal{C})\frac{8}{\varepsilon}\log\frac{13}{\delta}, \frac{4}{\varepsilon}\log\frac{2}{\delta}\Big)$, where $V(\mathcal{C})$ is the VC-dimension of the class $\mathcal{C}$. $\square$

4 Main results

The main result of this work is that for any computable predictor φ there is no computable upper bound in terms of $l(\eta)$ on the sample complexity of the function η with respect to φ:

Theorem 1. *For every computable predictor φ and every partial computable function $\beta : \mathbb{N} \to \mathbb{N}$ that has infinite domain and goes to infinity, there are infinitely many functions η, such that for some $n > \beta(l(\eta))$*

$$P\{x \in X_{t(\eta)} : \varphi(x_1, y_1, \dots, x_n, y_n, x) \neq \eta(x)\} > 0.05,$$

for any $x_1, \dots, x_n \in X_{t(\eta)}$, where $y_i = \eta(x_i)$ and P is the uniform distribution on $X_{t(\eta)}$.

For example, we can take $\beta(n) = 2^n$, or 2^{2^n}.

Corollary 1. *For any computable predictor φ, any total computable function $\beta : \mathbb{N} \to \mathbb{N}$ and any $\delta < 1$*

$$\sup_{\eta : l(\eta) \le k} N(\varphi, \eta, \delta, 0.05) > \beta(k)$$

from some k on.

Observe that there is no δ in the formulation of Theorem 1. Moreover, it is not important how the objects $(x_1, \dots, x_n)$ are generated — it can be any individual sample. In fact, we can assume that the sample is chosen in any manner, for example by some algorithm. This means that no computable upper bound on sample complexity exists even for *active learning algorithms*.

It appears that the task of classification is closely related to another learning task — data compression. Moreover, to prove Theorem 1 we need a similar negative result for this task. Thus before proceeding with the proof of the theorem, we introduce the task of data compression and derive a negative result for it. We call a total computable function $\psi : \{0,1\}^* \to \{0,1\}^*$ a *data compressor* if it is an injection (i.e. $x_1 \ne x_2$ implies $\psi(x_1) \ne \psi(x_2)$). We say that a data compressor *compresses* the string x if $|\psi(x)| < |x|$. Clearly, for any natural n any data compressor compresses not more than half of the strings of size up to n.

Next we present a definition of Kolmogorov complexity; for fine details see [11, 6]. The complexity of a string $x \in \{0,1\}^*$ with respect to a Turing machine ζ is defined as

$$C_\zeta(x) = \min_p \{l(p) : \zeta(p) = x\},$$

where p ranges over all binary strings (interpreted as partial computable computable functions; minimum over empty set is defined as ∞). There exists such a machine ζ that $C_\zeta(x) \le C_{\zeta'}(x) + c_{\zeta'}$ for any x and any machine ζ' (the constant $c_{\zeta'}$ depends on ζ' but not on x). Fix any such ζ and define the *Kolmogorov complexity* of a string $x \in \{0,1\}^*$ as

$$C(x) := C_\zeta(x).$$

Clearly, $C(x) \le |x| + b$ for any x and for some b depending only on ζ. A string is called c-incompressible if $C(x) \ge |x| - c$. Obviously, any data compressor can not compresses many c-incompressible strings, for any c. However, highly compressible strings (that is, strings with Kolmogorov complexity low relatively to their length) might be expected to be compressed well by some sensible data compressor. The following lemma shows that this cannot be always the case, no matter what we mean by "relatively low".

The lemma is proven using the fact that there are no non-trivial computable lower bounds on Kolmogorov complexity; the lemma itself can be considered as a different formulation of this statement. The proof of the lemma is followed by the proof of Theorem 1.

Lemma 1. *For every data compressor ψ and every partial computable function $\gamma : \mathbb{N} \to \mathbb{N}$ which has an infinite domain and goes to infinity there exist infinitely many strings x such that $C(x) \leq \gamma(|x|)$ and $|\psi(x)| \geq |x|$.*

For example, we can take $\gamma(n) = \log \log n$.

Proof. Suppose the contrary, i.e. that there exist a data compressor ψ and some function $\gamma : \mathbb{N} \to \mathbb{N}$ monotonically increasing to infinity such that if $C(x) \leq \gamma(|x|)$ then $\psi(x) < |x|$ except for finitely many x. Let T be the set of all strings which are not compressed by ψ

$$T := \{x : |\psi(x)| \geq |x|\}.$$

Define the function τ on the set T as follows: $\tau(x)$ is the number of the element x in T

$$\tau(x) := \#\{x' \in T : x' \leq x\}$$

for each $x \in T$. Obviously, the set T is infinite. Moreover, $\tau(x) \leq x$ for any $x \in T$ (recall that we identify $\{0,1\}^*$ and $\mathbb{N}$ via length-lexicographical ordering). Observe that τ is a total computable function on T and onto $\mathbb{N}$. Thus $\tau^{-1} : \mathbb{N} \to \{0,1\}^*$ is a total computable function on $\mathbb{N}$. Hence, for any $x \in T$ for which $\gamma(|x|)$ is defined we have, except for finitely many x:

$$C(\tau(x)) \geq C(\tau^{-1}(\tau(x)) - c = C(x) - c > \gamma(|x|) - c, \tag{1}$$

for a constant c depending only on τ, where the first inequality follows from computability of τ^{-1} and the last from the definition of T. Since τ is computable we also have $C(\tau(x)) \leq C(x) + c'$ for some constant c'.

It is a well-known result (see e.g. [11]) that for any unbounded partial computable function δ with infinite domain there are infinitely many $x \in \{0,1\}^*$ such that $C(x) \leq \delta(|x|)$. In particular, allowing $\delta(|x|) = \gamma(|x|) - c' - 2c$, we conclude that there are infinitely many $x \in T$ such that

$$C(\tau(x)) \leq C(x) + c' \leq \gamma(|\tau(x)|) - 2c \leq \gamma(|x|) - 2c,$$

which contradicts (1). □

Proof of Theorem 1. Suppose the contrary, that is that there exists such a computable predictor φ and a partial computable function $\beta : \mathbb{N} \to \mathbb{N}$ such that for any except finitely many labelling functions η for which $\beta(l(\eta))$ is defined and all $n > \beta(l(\eta))$ we have

$$P\{x : \varphi(x_1, y_1, \dots, x_n, y_n, x) \neq \eta(x)\} \leq 0.05,$$

for some $x_i \in X_{t(\eta)}$, $y_i = \eta(x_i)$, $i \in \mathbb{N}$, where P is the uniform distribution on $X_{t(\eta)}$.

Define $\varepsilon := 0.05$. We will construct a data compressor ψ which contradicts Lemma 1. For each $y \in \{0,1\}^*$ define $m := |y|$, $t := \ulcorner \log m \urcorner$. Generate (lexicographically) first m strings of length t and denote them by x_i, $1 \leq i \leq m$. Define the labelling function η_y as follows: $\eta_y(x) = y^i$, if x starts with x_i, where $1 \leq i \leq m$. Clearly, $C(\eta_y) \geq C(y) - c$, where c is some universal constant capturing the above description. Let the distribution P be uniform on X_t.

Set $n := \sqrt{m}$. Next we run the predictor φ on all possible tuples $\mathbf{x} = (x_1, \dots, x_n) \in {X_t}^n$ and each time count the errors that φ makes on all elements of X_t:

$$E(\mathbf{x}) := \{x \in X_t : \varphi(x_1, y^1, \dots, x_n, y^n, x) \neq \eta_y(x)\}.$$

Thus $E(\mathbf{x})$ is the set of all objects on which φ errs after being trained on $\mathbf{x}$. If $|E(\mathbf{x})| > \varepsilon m$ for all $\mathbf{x} \in X_t$ then $\psi(y) := 0y$.

Otherwise proceed as follows. Fix some tuple $\mathbf{x} = (x'_1, \dots, x'_n)$ such that $|E(\mathbf{x})| \leq \varepsilon m$, and let $H := \{x'_1, \dots, x'_n\}$ be the unordered tuple $\mathbf{x}$. Define

$$\kappa^i := \begin{cases} e & \text{if } \ x_i \in E(\mathbf{x}) \backslash H \\ c_0 & \text{if } \ x_i \in H, y^i = 0 \\ c_1 & \text{if } \ x_i \in H, y^i = 1 \\ * & \text{otherwise} \end{cases}$$

for $1 \leq i \leq m$. Thus, each κ^i is a member of a five-letter alphabet (a four-element set) $\{e, c_0, c_1, *\}$. Denote the string $\kappa^1 \dots \kappa^m$ by K.

So K contains the information about the (unordered) training set and the elements on which φ errs after being trained on this training set. Hence the string K, the predictor φ and the order of $(x'_1, \dots, x'_n)$ (which is not contained in K) are sufficient to restore the string y. Furthermore, the n-tuple $(x'_1, \dots, x'_n)$ can be obtained from H (the un-ordered tuple) by the appropriate permutation; let r be the number of this permutation in some fixed ordering of all $n!$ such permutations. Using Stirling's formula, we have $|r| \leq 2n \log n = \sqrt{m} \log m$; moreover, to encode r with some self-delimiting code we need not more than $2\sqrt{m} \log m$ symbols (for $m > 3$). Denote such an encoding of r by ρ.

Next, as there are at least $(1 - \varepsilon - \frac{1}{\sqrt{m}})m$ symbols $*$ in the m-element string K (at most εm symbols e_0 and e_1, and $n = \sqrt{m}$ symbols c_0 and c_1), it can be encoded by some simple binary code σ in such a way that

$$|\sigma(K)| \leq \frac{1}{2}m + 8(\varepsilon m + n). \tag{2}$$

Indeed, construct σ as follows. First replace all occurrences of the string $**$ with 0. Encode the rest of the symbols with any fixed 3-bit encoding such that the code of each letter starts with 1. Clearly, $\sigma(K)$ is uniquely decodable. Moreover, it is easy to check that (2) is satisfied, as there are not less than $\frac{1}{2}(m - 2(\varepsilon m + n))$ occurrences of the string $**$. We also need to write m in a self-delimiting way (denote it by s); clearly, $|s| \leq 2 \log m$.

We can define a monotone increasing function β' with an infinite domain on which it coincides with β. Indeed, this can be done by executing in a quasi-parallel fashion β on all inputs and defining $\beta'(k) = \beta(k)$ if $\beta(k)$ was found and $\beta'(l) < \beta'(k)$ for all l on which β' is already defined. Next we can define a function $\beta^{-1}(n)$ with infinite domain such that β^{-1} goes monotonically to infinity and such that $\beta^{-1}(\beta'(n)) = n$. This can be done by running in a quasi-parallel fashion β on all inputs m and stopping when $\beta(m) = n$ with m as an output.

Finally, $\psi(y) = 1s\rho\sigma(K)$ and $|\psi(y)| \leq |y|$, for $m > 2^{10}$. Thus, ψ compresses any (except finitely many) y such that $n > \beta'(C(\eta_y))$; i.e. such that $\sqrt{m} > \beta'(C(\eta_y)) \geq \beta'(C(y) - c)$. This contradicts Lemma 1 with $\gamma(k) := \beta^{-1}(\sqrt{k}) + c$. $\square$

5 Different settings and tightness of the negative results

In this section we discuss how tight the conditions of the statements are and to what extend they depend on the definitions.

Let us consider the question of whether there exists some (not necessarily computable) total sample-complexity function

$$\mathcal{N}_\varphi(k,\delta,\varepsilon) := \sup_{\eta: l(\eta)\le k} N(\varphi,\eta,\delta,\varepsilon),$$

at least for some predictor φ.

Proposition 2. *There exists a predictor φ such that $\mathcal{N}_\varphi(k,\delta,\varepsilon) < \infty$ for any $\varepsilon,\delta > 0$ and any $k \in \mathbb{N}$.*

Indeed it is easy to see that the "pointwise" predictor

$$\varphi(x_1,y_1,\dots,x_n,y_n,x) = \begin{cases} y_i & \text{if } x = x_i, 1 \le i \le n \\ 0 & x \notin \{x_1,\dots,x_n\} \end{cases} \tag{3}$$

satisfies the conditions of the proposition.

It can be argued that probably this statement is due to our definition of a labelling function. Next we will discuss some other variants of this definition.

First, observe that if we define a labelling function as any total computable function on $\{0,1\}^*$ then some labelling functions will not approximate any function on $[0,1]$; for example the function η_+ which counts bitwise sum of its input: $\eta_+(x) := \sum_{i=1}^{|x|} x_i \mod 2$. That is why we require a labelling function to be defined only on X_t for some t.

Another way to define a labelling function (which perhaps makes labelling functions most close to real functions) is as a function which accepts any *infinite* binary string. Let us call an *i-labelling function* any total recursive function $\eta : \{0,1\}^\infty \to \{0,1\}$. That is, η is computable on a Turing machine with an input tape on which one way infinite input is written, an output tape and possibly some working tapes. The program η is required to halt on any input. As it was mentioned earlier, in this case the situation essentially does not change, since (as it is easy to show) for any i-labelling function η there exist $n_\eta \in \mathbb{N}$ such that η does not scan its input tape beyond position n_η. In particular, $\eta(x) = \eta(x')$ as soon as $x_i = x'_i$ for any $i \le n_\eta$. Moreover, it is easy to check that Theorem 1 holds for i-labelling functions as well. Finally, it can be easily verified that Proposition 2 holds true if we consider i-labelling functions instead of labelling functions, constructing the required predictor based on the nearest neighbour predictor. Indeed, it suffices to replace the "pointwise" predictor in the proof of Proposition 2 by the predictor φ, which assigns to the object x the label of that object among $x_1,\dots,x_n$ with whom x has longest mutual prefix (where the prefixes are compared up to some growing horizon).

6 Discussion

The main result of the paper can be interpreted as that the task of computational classification is not feasible if the target labelling function is (only) known to be computable. In fact this means that the task of finding such a function η in a (finite) class $\mathcal{C}$ that η fits the given data can be algorithmically a very complex problem. It is also important to note here that we did not impose any resource constraints on the computation.

Perhaps the proposed approach, that is, the analysis of complexity of a computable learning problem as related to the complexity of the solution, can be applied to learning problems other than classification and data compression. In the present paper the complexity of a learning problem means sample size complexity, whereas complexity of the solution is the length (or Kolmogorov complexity) of the program which describes it. It can be conjectured that whatever a learning problem is, its complexity in terms of the complexity of a solution is very high, if both complexities are reasonably defined. This further supports the view that “universal” learners are not feasible and each specific learning problem should be solved by a specially designed algorithm.

References

[1] L. Devroye, L. Györfi, G. Lugosi, A probabilistic theory of pattern recognition. New York: Springer, 1996.

[2] S. Jain, D. Osherson, J. Royer, and A. Sharma. Systems That Learn: An Introduction to Learning Theory, 2nd edition. The MIT Press, Cambridge, MA, 1999.

[3] M. Hutter, Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability, Springer, 2004.

[4] M. Kearns and U. Vazirani. An Introduction to Computational Learning Theory The MIT Press, Cambridge, Massachusetts, 1994.

[5] S. Kleene, Mathematical Logic, New York: Wiley, 1967.

[6] M. Li, P. Vitányi. An introduction to Kolmogorov complexity and its applications. Second edition, Springer, 1997.

[7] W. Menzel, F. Stephan. Inductive versus approximative learning. In: Perspectives of adaptivity and learning, edited by R. Kuehn et al., pp. 187–209, Springer, 2003.

[8] H. Rogers. Theory of recursive functions and effective computability, McGraw-Hill Book Company, 1967.

[9] D. Ryabko. On computability of pattern recognition problems, in Proceedings of The 16th International Conference on Algorithmic Learning Theory Singapore, pp. 148–156, 2005

[10] V. Vapnik, Statistical Learning Theory: New York etc.: John Wiley & Sons, Inc. 1998

[11] N. Vereshchagin, A. Shen and V. Uspensky. Lecture Notes on Kolmogorov Complexity, 2004, Unpublished, http://lpcs.math.msu.su/∼ver/kolm-book .