

# On Universal Transfer Learning

M. M. Hassan Mahmud

*Department of Computer Science  
University of Illinois at Urbana Champaign  
201 N. Goodwin Avenue, Urbana, IL 61801, USA*

---

## Abstract

In transfer learning the aim is to solve new learning tasks using fewer examples by using information gained from solving related tasks. Existing transfer learning methods have been used successfully in practice and PAC analysis of these methods have been developed. But the key notion of relatedness between tasks has not yet been defined clearly, which makes it difficult to understand, let alone answer, questions that naturally arise in the context of transfer, such as, how much information to transfer, whether to transfer information, and how to transfer information across tasks. In this paper we look at transfer learning from the perspective of Algorithmic Information Theory/Kolmogorov complexity theory, and formally solve these problems in the same sense Solomonoff Induction solves the problem of inductive inference. We define universal measures of relatedness between tasks, and use these measures to develop universally optimal Bayesian transfer learning methods. We also derive results in AIT that are interesting by themselves. To address a concern that arises from the theory, we also briefly look at the notion of Kolmogorov complexity of probability measures. Finally, we present a simple practical approximation to the theory to do transfer learning and show that even these are quite effective, allowing us to transfer across tasks that are superficially unrelated. The latter is an experimental feat which has not been achieved before, and thus shows the theory is also useful in constructing practical transfer algorithms.

---

## 1 Introduction

In Transfer Learning (TL) (Pratt, 1992; Singh, 1992; Caruana, 1993, 1997; Schmidhuber, 1994; Thrun and Mitchell, 1995), we are concerned with reducing sample complexity required to learn a particular task by using information gained from solving *related tasks* (see Thrun and Pratt (1998); Vilalta

---

*Email address:* hmahmud42@gmail.com (M. M. Hassan Mahmud).

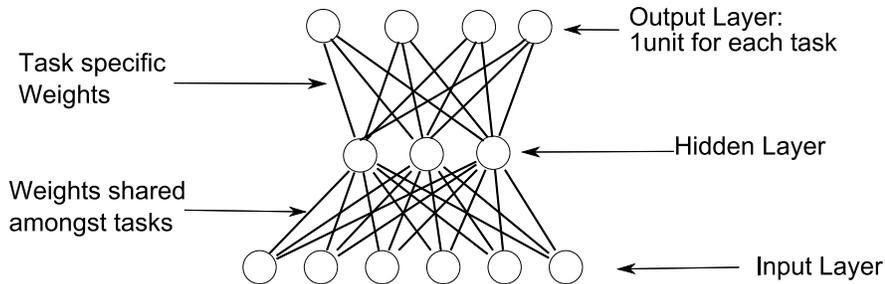


Fig. 1. A typical transfer learning method.

and Drissi (2002) for reviews). Each task in TL corresponds to a particular probability measure generating the data for the task. Transfer learning has in general been inspired by noting that to solve a problem at hand, people almost always use knowledge gained from solving related problems previously. This motivation has been borne out by practical successes; TL was used to recognize related parts of a visual scene in robot navigation tasks (Caruana, 1997), predict rewards in related regions in reinforcement learning based robot navigation problems (Thrun and Mitchell, 1995), predict results of related medical tests for the same group of patients etc. (Caruana, 1997), transfer information across relational/structured data sets, (Mihalkova et al., 2007), transfer in difficult reinforcement learning problems (Taylor and Stone, 2007), and even transfer across *superficially* unrelated classification tasks (Mahmud and Ray, 2007). A key concept in transfer learning, then, is this notion of relatedness between tasks. As we will see, in the work preceding this paper and its predecessors, it was not clear what a proper way to define this notion is (see also Ben-David and Schuller (2003)), and in addition to being conceptually troubling, this problem has also hampered development of more powerful and principled transfer algorithms.

Many current TL methods are in essence based on the method developed by Caruana (1997). The basic idea is to learn  $m$  related tasks in *parallel* using neural networks, with all the tasks defined on the same input space (Fig. 1). Different tasks are related by virtue of requiring the same set of good ‘high level features’ encoded by the hidden units. The hope is that by training with alternating training samples from different tasks, these common high level features will be learned quicker. The same idea has been used for sequential transfer – i.e. input-to-hidden layer weights from previously learned related tasks were used to speed up learning of new tasks. So tasks are considered related if they can be learned faster together than individually, that is there exists a sub-hypothesis space/inductive bias which contains the best hypothesis for each task, and is strictly smaller than the complete space (ideally much smaller). The optimal (and trivial!) inductive bias for a learning problem is, of course, when the bias consists of just the best hypothesis in the space. So one succinct way to state the central idea in multitask learning is to say that tasks are considered related if they have a *common near-optimal inductive*

*bias* with respect to a given hypothesis space (e.g. the common hidden units in Fig. 1). We refer the reader to the introductory sections by Baxter (2000) for discussion of these issues.

Indeed, the above setup for transfer learning was analyzed extensively in a PAC framework by Baxter (2000). Here a probability distribution  $P$  was assumed over the space of tasks, and bounds were derived on the sample complexity required to estimate the expected error (with respect to  $P$ ) of the  $m$  tasks when the tasks were learned using a sub-space of the hypothesis space. That is bounds were derived for sample complexity for estimating fitness of inductive biases. Most work done on TL is subsumed by this analysis, and they all begin with the assumption that tasks have a common, near optimal inductive bias. So no actual measure of similarity between tasks is prescribed, and hence it becomes difficult to understand, let alone answer, questions such as ‘how and when should we transfer information between tasks?’ and ‘how much information should we transfer?’.

There has been some work which attempt to solve these problems. Ben-David and Schuller (2003) give a more explicit measure of relatedness in which two tasks  $P$  and  $Q$  are said to be similar with respect to a given set of functions  $F$  if  $\exists f \in F$  such that  $P(a) = Q(f(a))$  for all events  $a$ . Using  $F$ , the authors derive PAC sample complexity bounds for the error of each task (as opposed to expected error in Baxter (2000)), which can be smaller than single task bounds under certain conditions. So the measure of similarity used is *binary* in that the tasks are either related or they are not. So this does not help solve the problems of measuring *how much* information to transfer and so forth.

More interesting is the approach by Juba (2006) who extends Baxter (2000). Juba (2006) deals only with finite sample spaces, and computable tasks and hypothesis spaces, and gives PAC bounds, where the sample complexity required to bound the expected error is proportional to the *joint Kolmogorov complexity* (Li and Vitanyi, 1997) of the  $m$  hypothesis being considered. The number of tasks required for the bounds to hold is  $\geq 8192$  (Theorem 3). Use of joint Kolmogorov complexity to measure relatedness is a step in the right direction as it measures how well the tasks compress together and hence the *total* absolute information content of the tasks. However what we actually want is the amount of information tasks contain *about* each other, and for this we need to use the *conditional Kolmogorov complexity* and the *Information Distance* (Bennett et al., 1998). Indeed, this is the direction we take in this paper and it leads to an elegant formulation of a transfer learning distance. In addition, through the Bayesian convergence results by Solomonoff (1978); Hutter (2003), we also derive Bayesian transfer learning methods which can be approximated to construct powerful practical transfer learning algorithms (Sect. 6 (Mahmud and Ray, 2007)). Juba (2006) does not provide guidance in actually constructing transfer learning methods.

More specifically, in this paper we address the problems with transfer learning mentioned above in the framework of Algorithmic Information Theory. Our aim will be to look at what is the best we can do (most amount of similarity between tasks we can uncover, most amount of information we can transfer etc.) given unlimited amount of computational time and space. We use and extend the theory of Information Distance (Bennett et al., 1998) to measure relatedness between tasks, transfer the right amount of information etc. For our task space we restrict ourselves to probability measures that are lower semicomputable, which is reasonable as it covers all situations where we learn using computers. In this space the Information Distance is a universal measure of relatedness between tasks. We give a sharp characterization of Information Distance by showing it is, up to a constant, equal to the Cognitive Distance (Theorems 3.10 and 3.14, which are quite interesting results by themselves).

Based on our transfer learning distance, we develop universally optimal Bayesian transfer learning methods for doing sequential transfer (Theorem 4.6). We show that sequential transfer is always justified from a formal perspective (Theorem 4.7). We also briefly discuss several extensions to this theory in Sect. 4.3. In Sect. 4.3.1 we show via a qualitative discussion that parallel or multitask learning algorithms used in practice in a batch setting are just sequential transfer methods in disguise. We also mention that parallel transfer methods in the online setting are no better than sequential transfer in the general case. In Sect. 4.3.2 we discuss a new interpretation of universal optimality of our methods which is in a sense more robust than the ‘classical’ interpretation of universal optimality. And then in section 4.3.3, we briefly discuss how our approach to transfer gives an elegant formulation to the problems of (1) transfer for arbitrary loss functions in sequence prediction (2) reinforcement learning agents ( (1) and (2) via Hutter (2004)) and (3) to the problem of cross-domain transfer (Swarup and Ray, 2006).

Additionally, we also briefly investigate if it is appropriate to define (the way we do in this paper) the Kolmogorov complexity of a probability measure (a function) as the complexity of a program computing it (one of the intensional representations of the function). We show via Lemma 5.6 that under certain reasonable conditions on the complexity, it is appropriate to do so. Finally, in Sect. 6 we derive a very simple and crude approximation to our transfer learning mechanism in Sect. 4, and perform an extensive set of transfer learning experiments using Bayesian decision trees. The experiments turn out to be the most general transfer experiments at the time of these experiments were performed, in the sense that we were able to transfer between data sets that are superficially unrelated. So the experiments establish that the theory can also provide good guidance on constructing practical transfer algorithms.

## 2 Preliminaries

We use  $a := b$  to mean expression  $a$  is defined by expression  $b$ . We use  $\mathbb{N}_m$  to denote the numbers  $1, 2, \dots, m$ . For any finite alphabet  $A$ , let  $A^*, A^n, A^\infty$  be the set of all finite strings, length  $n$  strings and infinite sequences in  $A$  respectively. Let  $\varepsilon$  be the empty string. For  $x, y \in A^*$ ,  $xy$  denotes  $y$  concatenated to the end of  $x$ . Let  $l(x)$  denote the length of a finite string  $x$ . We will use  $x_{1:t}$  to denote the first  $t$  elements of a sequence  $x$  and  $x_{<t}$  the elements  $x_{1:t-1}$ . We use  $x_{t:t}$  to refer to the  $t^{\text{th}}$  letter of the sequence  $x$ , and reserve single digit indices  $x_i$  to refer to different sequences. We will use  $x_{1,n}$  as a shorthand for  $x_1, x_2, \dots, x_n$ .

We use  $\langle \cdot, \cdot \rangle$  to denote a standard bijective mapping from  $A^* \times A^* \rightarrow A^*$ .  $\langle \rangle^m$  denotes the  $m$ -arity version of this, and  $\rangle_i^m$  denotes the  $i^{\text{th}}$  component of the inverse of  $\langle \rangle^m$ . We assume the standard ‘lexicographical’ correspondence between  $A^*$  and  $\mathbb{N}$  – e.g. for  $A := \{0, 1\}$ , this is  $(\varepsilon, 0), (0, 1), (1, 2), (00, 3), (01, 4), \dots$ . Depending on the context, elements of each pair will be used interchangeably (so 01 (and 4) may mean either 01 or 4). A rational number  $a/b$  is represented by  $\langle a, b \rangle$ . We use  $\stackrel{+}{\leq}$  to denote  $\leq$  up to an additive constant independent of the variables in the relation i.e.  $f(x) \stackrel{+}{\leq} g(x)$  means  $f(x) \leq g(x) + c$ . We use the same convention for all the usual binary (in)equality relations. Let  $2^{-\infty} := 0$ ,  $\log := \log_2$  and  $\bar{m}$  the self delimiting encoding of  $m \in \mathbb{N}$  using  $l(m) + 2l(l(m)) + 1$  bits where  $l(m) = \lfloor \log(m + 1) \rfloor$  (Li and Vitanyi, 1997).

We fix a reference prefix universal Turing machine  $U : \mathcal{B}^* \times \mathcal{A}^* \rightarrow \mathcal{A}^*$ , where  $\mathcal{B} := \{0, 1\}$  is the alphabet for programs, and  $\mathcal{A}, \mathcal{A} \supset \mathcal{B}$ , is an arbitrary alphabet for inputs and outputs. The prefix property means that programs are self-delimiting and the lengths of programs satisfy the Kraft inequality:  $\sum_p 2^{-l(p)} \leq 1$ .  $U(p, x)$  denotes the output of program  $p$  when run on input  $x$ . When it is clear from the context that  $p$  is a program, we will denote  $U(p, x)$  simply by  $p(x)$ .

We need some notions of computability of real functions. A real function  $f$  is *upper semicomputable* if there is a program  $p$  such that for  $x, t \in \mathbb{N}$ , 1)  $p(\langle x, t \rangle)$  halts in finite time 2)  $p(\langle x, t \rangle) \geq p(\langle x, t + 1 \rangle)$  3)  $\lim_{t \rightarrow \infty} p(\langle x, t \rangle) = f(x)$ . A real function  $f$  is *lower semicomputable* if  $-f$  is upper semicomputable. A function  $f$  is *computable/recursive* if there is a  $p$  such that for  $n, x \in \mathbb{N}$ ,  $|p(\langle x, n \rangle) - f(x)| < 2^{-n}$ , and  $p(\langle x, n \rangle)$  halts in finite time. We use  $p(x) \uparrow q(x)$  to denote that at  $x$ ,  $p$  and  $q$  lower semicompute the same function.

### 3 Universal Transfer Learning Distances

In this section we will first describe our task space and the learning problem we consider. Then we will discuss our universal transfer learning distances.

#### 3.1 Task Space $\mathcal{V}$ and the Learning Problem

Recall that in transfer learning literature, a learning problem or task is identified with the distribution that generates the samples for that problem (Baxter, 2000) – i.e. the distribution we need to learn or predict against. The reasoning behind this identification is that the only thing that distinguishes one learning problem from another is this generating distribution. While we consider a Bayesian setting (unlike Baxter (2000), who considers a PAC setting), we subscribe to this definition of a task as well. The precise nature of the generation and prediction process that defines a task in our context must necessarily follow the description of the probability measure space we use, and we do this now. We consider as our probability measure space a particular subset of the set of all *semimeasures*.

**Definition 3.1** *A semimeasure is a function  $f : \mathcal{A}^* \rightarrow [0, 1]$  such that*

$$\forall x \in \mathcal{A}^*, f(x) \geq \sum_{a \in \mathcal{A}} f(xa).$$

$f(x)$  is the ‘defective probability’ that a particular infinite sequence starts with the prefix string  $x$  ( $f$  is a probability measure if  $f(\varepsilon) = 1$  and the inequality is an equality). So  $f$  is equivalent to a probability measure  $p$  defined on  $[0, 1]$  such that  $f(x) = p([0.x, 0.x + |\mathcal{A}|^{-l(x)}))$  where  $0.x$  is in base  $|\mathcal{A}|$ . The conditional probability of the next letter being  $a$  given the string  $x$  observed so far is  $f(a|x) := f(xa)/f(x)$ .

Zvonkin and Levin (1970) showed that the set of all lower semicomputable semimeasures is recursively enumerable. That is, there is a Turing machine  $T$  such that  $T(\langle i, \cdot \rangle)$  lower semicomputes  $f_i(\cdot)$ , the  $i^{\text{th}}$  semimeasure in this effective enumeration. Since  $U$  is universal, for each  $i \in \mathbb{N}$ , there is a program  $p_i$  such that  $p_i(x) = T(\langle i, x \rangle)$ . Let  $\mathcal{V}$  be the enumeration of these programs so that each  $\mu \in \mathcal{V}$  lower semicomputes some lower semi-computable semimeasure  $f$ , and each lower semicomputable semimeasure  $f_j$  is computed by at least one  $\mu \in \mathcal{V}$ . We use lowercase Greek letters to denote elements of  $\mathcal{V}$ , which are programs, while we used Roman letters to denote the actual semimeasures (i.e. their extensional representations).

We will consider enumerable subsets  $\mathcal{V}'$  of  $\mathcal{V}$  as the space of probability measures we use in our tasks, as any probability measure that we may expect to be able to learn must either be computable itself, or have a reasonable approximation (however it may be defined) that is computable. Now  $\mathcal{V}$  is the largest superset of the set of computable semimeasures that contains any Bayes mixture of its own elements (Hutter, 2003, Chap. 2), which is important in Sect. 4 (see also Li and Vitanyi (1997)). We can now define our tasks.

**Definition 3.2** *A learning problem or task in the online Bayesian sequence prediction setting is a game played with the predictor which operates as follows. Each task has an associated generating semimeasure  $\mu \in \mathcal{V}$ . At each step  $t$ , a letter  $a \in \mathcal{A}$  is sampled according to  $\mu(\cdot|x_{<t})$ , where  $x_{<t}$  is the string sampled by  $\mu$  in the previous  $t - 1$  steps. The objective in a task is to predict the letter  $a$  at each step  $t$  before it has been sampled according to  $\mu$  (see e.g. Hutter (2003, Sect. 6.2) for how i.i.d. learning problems are a special case of this setting).*

Since the only thing that distinguishes two tasks is the generating semimeasure, for convenience we will refer to elements  $\mu \in \mathcal{V}$  as tasks, and  $\mathcal{V}$  as the task space.

### 3.2 Universal Transfer Learning Distance

We want our transfer learning distance to measure the amount of constructive information  $\mu, \varphi \in \mathcal{V}$  contain about each other. Elements of  $\mathcal{V}$  are strings (interpreted as programs computing measures), and the following defines the amount of constructive information any string  $y$  contains about another string  $x$ .

**Definition 3.3** *The conditional Kolmogorov complexity of  $x$  given  $y$ ,  $x, y \in \mathcal{A}^*$  is the length of the shortest program that outputs  $x$  given  $y$  :*

$$K(x|y) := \min_p \{l(p) : p(y) = x\} .$$

Conditional Kolmogorov complexity measures absolute information content of individual objects, and is a sharper version of information-theoretic entropy which measures information content of ensemble of objects relative to a distribution over them. When  $y = \varepsilon$ , the above is just called Kolmogorov complexity and denoted by  $K(x)$ . For  $m$  strings we use  $\langle \rangle^m$  – e.g.  $K(x, y|z, w) := K(\langle x, y \rangle | \langle z, w \rangle)$  etc. We note that fixing  $U$  as a reference machine is fine because by the *Invariance Theorem* (Kolmogorov, 1965) given any two universal Turing machines  $U_i$  and  $U_j$

$$|K_{U_i}(x|y) - K_{U_j}(x|y)| \stackrel{\pm}{=} 0$$

where  $K_Z$  is the conditional Kolmogorov complexity where the programs are for universal Turing machine  $Z$ . We list some fundamental properties of  $K$ :

**Lemma 3.4**  $\forall x, y, y_{1,m} \in \mathcal{A}^*$ :

- (1)  $K(x|y) \stackrel{+}{\leq} K(x)$ .
- (2)  $K(x) \stackrel{+}{\leq} K(\langle x, y_{1,m-1} \rangle^m)$ .
- (3) The function  $K(x|y)$  is upper semicomputable.
- (4)  $K(x|\arg K(y)) + K(y) \stackrel{\pm}{=} K(x, y)$  (Gacs, 1974; Chaitin, 1975)
- (5)  $K(x, y|z) \stackrel{\pm}{=} K(x|\arg K(y|z), z) + K(y|z) \stackrel{\pm}{=} K(x|y, K(y|z), z) + K(y|z)$  .

**PROOF.** (*Sketch*) The first two properties follow from the definition of the  $K$  functions and the following. The first property follows from the fact that any program that computes  $x$ , with a constant length modification to ignore any input, is also a program to output  $x$  given  $y$ . The second property follows because any program that outputs  $\langle x, y_{1,m-1} \rangle^m$ , with a constant length modification to output  $\langle x, y_{1,m-1} \rangle^m \langle 1 \rangle^m$ , also outputs  $x$ .

For the third property, we note that the following program  $p$  upper semicomputes  $K(x|y)$ :  $p(\langle \langle x, y \rangle, t \rangle)$  runs all programs  $q$  on  $y$  with  $l(q) \leq 2l(x)$  (a loose upper bound on  $K(x|y)$ ), in parallel by ‘dovetailing’, for  $t$  steps each.  $p$  then outputs the length of the shortest program found thus far.

The fourth property, discovered first by Gacs (1974), and then independently by Chaitin (1975), is one of the deepest and fundamental results in Kolmogorov complexity theory. The proof is quite long and complex, and we refer the reader to Li and Vitanyi (1997). The fifth property is a conditional version of the fourth property.

The function  $K(x|y)$  is upper semicomputable which is in agreement with our goal to investigate what type of transfer is possible given infinite resources. We will also make extensive use of the following minimality property of  $K(x|y)$ :

**Lemma 3.5** For any partial, non-negative, upper semicomputable function  $f : \mathcal{A}^* \times \mathcal{A}^* \rightarrow \mathbb{R}$ , with  $f(x, y) = \infty$  when it is undefined, we have:

$$K(x|y) \stackrel{+}{\leq} f(x, y) \quad \text{if} \quad \sum_x 2^{-f(x,y)} \leq 1 \quad .$$

where the constant in  $\stackrel{+}{\leq}$  is equal to  $K(f) + O(1)$  (see Li and Vitanyi (1997)).

In the above lemma the dependence of the constant on  $K(f)$  can be ignored in this work for two reasons. First, in our applications  $f$  will either be symmetric distance functions (see Definition 3.8) and Bayesian priors (see Definition

4.4). We assume that all such distance functions and probability measures are *reasonable* – i.e. that they have short  $O(1)$  length. That this is an acceptable assumption to make can be seen by contemplating the distance functions and priors used in practice. Second, should the reader find the first assumption onerous, we refer them to Sect. 4.3.2, where we dispense with even this very reasonable assumption and induce a different and arguably more robust interpretation of our universal methods.

$K(x|y)$  measures the amount of information  $y$  contains about  $x$ . To measure the amount of information two strings contain about each other, Bennett et al. (1998) defined the following function:

**Definition 3.6** *The Information Distance between  $x, y \in \mathcal{A}^*$  is the length of the shortest program that given  $x$  outputs  $y$ , and vice versa:*

$$E_0(x, y) := \min_p \{l(p) : p(x) = y, p(y) = x\} .$$

$E_0$  is upper semicomputable<sup>1</sup>, which is again in agreement with our desire to investigate transfer in the limit. So for  $\mu, \varphi \in \mathcal{V}$ ,  $E_0(\mu, \varphi)$  measures the amount of information  $\mu$  and  $\varphi$  contain about each other. Hence  $E_0$  is the natural candidate for a transfer learning distance. We will however use a sharper characterization of  $E_0$ :

**Definition 3.7** *The Cognitive Distance between  $x, y \in \mathcal{A}^*$  is given by*

$$E_1(x, y) := \max\{K(x|y), K(y|x)\} .$$

$E_1$  is sharper than  $E_0$  because it is defined in terms of a 1-way conversion program (instead of 2-way) and because it is defined in terms of a better understood and investigated quantity (i.e.  $K(x|y)$ ).  $E_1$  is upper semicomputable because both terms in its definition are also upper semicomputable. Bennett et al. (1998) proved:

$$E_0(x, y) = E_1(x, y) + O[\log(E_1(x, y))] . \tag{3.1}$$

We will actually prove a sharper version of the above where the log term is replaced by a constant. Now, we need:

**Definition 3.8** *An admissible distance  $D$  is a partial, upper semicomputable, non-negative, symmetric function on  $\mathcal{A}^* \times \mathcal{A}^*$  with  $\forall y \sum_x 2^{-D(x,y)} \leq 1$*

<sup>1</sup> Consider  $p: p(\langle\langle x, y \rangle, t \rangle)$  runs all programs  $q$  on  $y$  and  $x$ , with  $l(q) \leq 2 \max\{l(x), l(y)\}$  (a loose upper bound on  $E_0(x, y)$ ), in parallel by ‘dovetailing’ for  $t$  steps each.  $p$  then outputs the length of the shortest program found.

(we will assume  $D(x, y) = \infty$  when it is undefined). Let  $\mathcal{D}$  be the set of admissible distances. A  $D \in \mathcal{D}$  is **universal** in  $\mathcal{D}$  if  $\forall D' \in \mathcal{D}, \forall x, y \in \mathcal{A}^*$ ,  $D(x, y) \stackrel{+}{\leq} D'(x, y)$ .

Bennett et al. (1998) proved the following theorem:

**Theorem 3.9**  $\forall D \in \mathcal{D}, \forall x, y \in \mathcal{A}^*$

$$E_1(x, y) \stackrel{+}{\leq} D(x, y) . \quad (3.2)$$

That is,  $E_1$  is universal in  $\mathcal{D}$  (this was proved via Lemma 3.5 with  $f = D$ , as  $D$  satisfies the requisite conditions due to its admissibility). Note that  $E_1$  satisfies the Kraft inequality as

$$\sum_x 2^{-E_1(x, y)} \leq \sum_x 2^{-K(x|y)} \leq 1$$

as  $K(\mu_i|\varphi)$  are lengths of programs.

We note that Bennett et al. (1998) showed that the above holds for admissible *metrics*, but as pointed out by Li et al. (2004), this holds for admissible distances as well. Admissible distances include admissible versions of Hamming, Edit, Euclidean, Lempel-Ziv etc. distances (Bennett et al., 1998; Li et al., 2004; Cilibrasi and Vitanyi, 2005). See Bennett et al. (1998) for an eloquent account of why admissible distances (and distances satisfying the Kraft Inequality) are interesting for strings. Normalized, practical versions of  $E_1$  has been applied very successfully in various clustering tasks – see Li et al. (2004) and especially Cilibrasi and Vitanyi (2005). We now state a sharper version of (3.1) (the proof is in Sect. 3.3).

**Theorem 3.10**

$$E_0(x, y) \stackrel{\pm}{=} E_1(x, y) .$$

Given Theorem 3.10, we now define:

**Definition 3.11** *The transfer learning distance between two tasks  $\mu, \varphi \in \mathcal{V}$  is defined as  $E_1(\mu, \varphi)$ .*

So from the above, we immediately get that transfer learning distance is universal in the class of admissible distances that may be used for measuring task similarity. This formally solves the conceptual problem of how one measures task similarity. We will use this distance function in Sect. 4 to formally solve other problems in transfer learning mentioned in the Introduction and give more reasons why it is sufficient to consider only admissible distances (see discussion following the proof of Theorem 4.6).

### 3.3 Proof of Theorem 3.10

To prove:

$$E_0(x, y) \stackrel{\pm}{=} E_1(x, y) .$$

**PROOF.** Let  $p$  be a program such that  $p(x) = y$  and  $p(y) = x$ . So by definition  $E_1(x, y) \leq l(p)$  for all such  $p$ . Since  $\arg E_0(x, y)$  is a such a  $p$ , we have  $E_1(x, y) \stackrel{\pm}{\leq} E_0(x, y)$ . Now we prove the inequality in the other direction. Fix any two strings  $\alpha, \beta$  and set  $E_1(\alpha, \beta) = E1$ . Now we will derive a program  $q_{E1}$  with  $l(q_{E1}) \stackrel{\pm}{=} E1$  which given  $\alpha$  outputs  $\beta$  and given  $\beta$  outputs  $\alpha$ . We will do so by constructing a graph  $G$  that assigns a unique color/code of length  $\leq E1 + 1$  to each pair of strings  $x, y$  with  $E_1(x, y) \leq E1$ , and the code will turn out to be more or less the program  $q_{E1}$  we need to convert  $\alpha$  to  $\beta$  and vice versa. We note that the proof of (3.1) also uses a similar graph construction method. Define  $G := (V, E)$  with vertices  $V$  and undirected edges  $E$ :

$$\begin{aligned} V &:= \{x : x \in A\} \text{ and } E := \{\{x, y\} : x \in A, y \in A_x\}, \text{ where,} \\ A &:= \{x : \exists y, E_1(x, y) \leq E1\} \text{ and } \forall x \in A, A_x := \{y : E_1(x, y) \leq E1\} . \end{aligned}$$

The degree of  $x \in V$  is  $|A_x|$  by construction. Hence the maximum degree of  $G$  is  $\Delta_G = \max_{x \in A} |A_x|$ . We define the set of colors/code  $\mathcal{C}_{E1}$  as:

$$\begin{aligned} \mathcal{C}_{E1} &:= \{p0 : p \in B\} \cup \{p1 : p \in B\}, \text{ where,} \\ B &:= \{p : p(x) = y, x \in A, y \in A_x, l(p) \leq E1\} . \end{aligned}$$

$q_{E1}$  will need to dynamically construct  $G$  and  $\mathcal{C}_{E1}$ , and assign a *valid* coloring to the edges in  $G$  using  $\mathcal{C}_{E1}$ . For this, all we need is  $E1$ . We run all programs  $p$  with  $l(p) \leq E1$  on all  $x \in A^*$  in ‘parallel’ by dovetailing and record triples  $(p, x, y)$  such that  $p(x) = y$ . Whenever we record  $(p, x, y)$  we check to see if we have previously recorded  $(q, y, x)$ . If so, we add  $p0, p1, q0, q1$  to  $\mathcal{C}_{E1}$ ,  $x, y$  to  $V$  and  $\{x, y\}$  to  $E$ . Of course, if any of these already exist in the respective sets, we do not add it again. We color a newly added edge  $\{x, y\}$  using a color from  $\mathcal{C}_{E1}$  using the First-Fit algorithm - i.e. the first color that has not been assigned to any other  $\{x, w\}$  or  $\{y, z\}$ . So, by dynamically reconstructing  $G$ , given  $x$  ( $y$ ) and the color for  $\{x, y\}$ ,  $q_{E1}$  can use the color to recognize and output  $y$  ( $x$ ).

That  $\mathcal{C}_{E1}$  has sufficient colors to allow valid coloring can be seen as follows.  $p \in B$  iff  $l(p) \leq E1$  and for some  $A_x, y \in A_x, p(x) = y$ . So for each  $A_x$ , for each  $y \in A_x$ ,  $\exists p_y \in B$ , and  $p_y \neq p_{y'} \forall y' \in A_x, y' \neq y$  since  $p_y(x) \neq y'$ . This means, for each  $A_x$ ,  $|\mathcal{C}_{E1}| \geq 2|A_x|$ , or  $|\mathcal{C}_{E1}| \geq 2\Delta_G$ . By the same reasoning and the construction procedure above, as we dynamically construct  $G$  and  $\mathcal{C}_{E1}$ , the estimates  $\mathcal{C}_{E1}^t$  and  $\Delta_G^t$  at step  $t$  of the construction process also satisfies

$|\mathcal{C}_{E1}^t| \geq 2\Delta_G^t$ . Now at step  $t$  First-Fit requires at most  $2\Delta_G^t - 1$  colors to assign a valid color, as two vertices could have exhausted at most  $2\Delta_G^t - 2$  colors between them. Therefore First-Fit always has sufficient colors to assign a valid coloring.

Each color/code in  $\mathcal{C}_{E1}$  is at most  $E1 + 1$  in length by construction. So, as we construct  $G$ ,  $\alpha$  and  $\beta$  shows up in the graph at some point with code/color (say)  $\gamma$ , and  $l(\gamma) \leq E1 + 1$ . From construction of  $\mathcal{C}_{E1}$ ,  $\gamma$  is a self-delimiting string  $p$ , followed by 0 or 1.  $\gamma$  and  $E1$  can be encoded by a string  $pa0^{E1-l(p)}1$ , where  $a$  is 0 if  $\gamma = p0$ , or 1 if  $\gamma = p1$ , and  $0^{E1-l(p)}$  is 0 repeated  $E1 - l(p)$  times.

The desired program  $q_{E1}$  has encoded in it the string  $pa0^{E1-l(p)}1$  at some fixed position, and  $q_{E1}(z)$  works as follows.  $q_{E1}$  decodes  $p$  (which is possible as it is self-delimiting) and then reads the next bit, which is  $a$ , to get  $\gamma$ . It computes  $E1$  from counting the number of 0s after  $a$  and  $l(p)$ . When  $a = 0$ , it is not confused with the 0s following it because it is the bit that appears immediately after  $p$ , and  $p$  can be decoded by itself.  $q_{E1}$  then reconstructs  $G$  using  $E1$ , and finds the edge  $\{z, w\}$  with color  $\gamma$ , and outputs  $w$ . By construction, if  $z = \alpha$  then  $w = \beta$  and if  $z = \beta$  then  $w = \alpha$ . Since  $l(q_{E1}) \stackrel{\pm}{=} E1$  (the constant being for the extra bits in  $pa0^{E1-l(p)}1$  and other program code in  $q$ ), we have  $E_0(\alpha, \beta) \leq l(q_{E1}) \stackrel{\pm}{=} E1(\alpha, \beta)$ , and therefore  $E_0(\alpha, \beta) \stackrel{\pm}{=} E1(\alpha, \beta)$ .  $\square$

### 3.4 Universal Transfer Learning Distance for $m$ Tasks

The material in this section may be skipped as it is not used below, but we include it here for completeness and because the results are interesting by themselves. We also hope that the functions here will find application in task clustering problems which are important for designing ‘Long Lived’ transfer learning agents (Thrun and Pratt, 1998), and in clustering problems in general (similar to Cilibrasi and Vitanyi (2005)). The distance functions in this section apply to arbitrary strings in addition to elements of  $\mathcal{V}$ .

Let  $X := \{x_{1,m}\}, x_j \in \mathcal{A}^*$ ,  $X_i^{m_1}$  the  $i^{th}$  subset of  $X$  of size  $m_1$ ,  $0 < m_1 < m$ ,  $0 < i \leq \binom{m}{m_1}$ . Let  $\sigma(X_i^{m_1})$  be the set of permutations of elements of  $X_i^{m_1}$ . We now define a generalization of  $E_0$  that measures how much information each group of  $m_1$   $x_j$ s,  $0 < m_1 < m$ , contain about the remaining  $m - m_1$   $x_j$ s:

**Definition 3.12** *The  $m$  fold information distance  $E_0^m(x_{1,m})$  between  $x_{1,m} \in \mathcal{A}^*$ , is the length of the shortest program that given any permutation of  $m_1$   $x_j$ s,*

$1 < m_1 < m$ , outputs a permutation of the other  $m - m_1$   $x_j$ s. That is:

$$E_0^m(x_{1,m}) := \min_p \{l(p) : \forall m_1, i, x, 0 < m_1 < m, 1 \leq i \leq \binom{m}{m_1}, \\ x \in \sigma(X_i^{m_1}), p(\langle x \rangle^{m_1}, m_1) = \langle y \rangle^{m-m_1}, \text{ where } y \in \sigma(X \setminus X_i^{m_1})\} .$$

In contrast to  $E_0$  the additional information  $m_1$  is included in the definition for  $E_0^m$  to determine how to interpret the input, – i.e. which  $\langle \cdot \rangle^{m_1}$  to use to decode the input.  $E_0^m$  is upper semicomputable by the same reasoning  $E_0$  is (Bennett et al., 1998). To give a sharper characterization of  $E_0^m$ , we define:

**Definition 3.13** *The  $m$  fold Cognitive Distance for  $x_{1,m} \in \mathcal{A}^*$  is:*

$$E_1^m(x_{1,m}) := \max_i \max_{y \in \sigma(X \setminus \{x_i\})} E_1(x_i, \langle y \rangle^{m-1}) .$$

$E_1^m$  is upper semicomputable by the same reasoning  $E_1$  is. We can now state the analogue of Theorem 3.10 for  $m$  strings (the proof is in Sect. 3.5):

**Theorem 3.14**

$$E_0^m(x_{1,m}) \stackrel{\pm}{=} E_1^m(x_{1,m}) .$$

**Definition 3.15** *The  $m$ -fold transfer learning distance between  $m$  tasks  $\mu_{1,m} \in \mathcal{V}$  is defined as  $E_1^m(\mu_{1,m})$ .*

We can also define admissible distances:

**Definition 3.16** *The  $m$  fold admissible distances between  $m$  tasks  $\mu_{1,m} \in \mathcal{V}$  are defined as functions  $D_m : \times_m \mathcal{A}^* \rightarrow \mathbb{R}$  that are non-negative, upper semicomputable,  $m$ -wise symmetric, and satisfy the following version of the Kraft inequality:  $\forall x, y_{1,m-1} \in \mathcal{A}^*$*

$$\sum_{z_{1,m-1} \in \mathcal{A}^*} 2^{-D_m(x, z_{1,m-1})} \leq 1 \text{ and } \sum_{w \in \mathcal{A}^*} 2^{-D_m(w, y_{1,m-1})} \leq 1 .$$

Let  $\mathcal{D}_m$  be the set of  $m$  fold admissible distances. A  $D \in \mathcal{D}$  is **universal** in  $\mathcal{D}$  if  $\forall D' \in \mathcal{D}, \forall x_{1,m} \in \mathcal{A}^*, D(x_{1,m}) \stackrel{\pm}{\leq} D'(x_{1,m})$ .

**Theorem 3.17** *1)  $E_1^m$  satisfies the above version of the Kraft inequality and 2) is universal in the class of admissible distances for  $m$  strings*

**PROOF.** Let  $x, y_{1,m-1} \in \mathcal{A}^*$ . Part 1 follows because by definition

$$E_1(x, \langle y_{1,m-1} \rangle^{m-1}) \leq E_1^m(x, y_{1,m-1}) .$$

and  $E_1(x, \langle y_{1,m-1} \rangle^{m-1})$  satisfies the Kraft inequality. For part 2, by Lemma 3.5 and admissibility of  $D_m$ ,  $K(x|y_{1,m-1}), K(y_{1,m-1}|x) \stackrel{+}{\leq} D_m(x, y_{1,m-1})$ . The desired result now follows from the definition of  $E_1^m$ .  $\square$

### 3.5 Proof of Theorem 3.14

To prove:

$$E_0^m(x_1, x_2, \dots, x_m) \stackrel{\pm}{=} E_1^m(x_1, x_2, \dots, x_m) .$$

**PROOF.** The proof is similar to the proof of Theorem 3.10 - we assume  $m$  is fixed and treat it as a constant. Otherwise the theorem holds up to additive  $m \log m$  terms.

Fix  $\Lambda := \{\lambda_{1,m}\}$ . We will first show

$$E_1^m(\lambda_{1,m}) \stackrel{+}{\leq} E_0^m(\lambda_{1,m}) .$$

Let  $p$  be a program such that  $\forall m_1, i, \lambda, 0 < m_1 < m, 1 \leq i \leq \binom{m}{m_1}, \lambda \in \sigma(\Lambda_i^{m_1}), p(\langle \langle \lambda \rangle^{m_1}, m_1 \rangle) = \langle y \rangle^{m-m_1}$ , where  $y \in \sigma(\Lambda \setminus \Lambda_i^{m_1})$ . Fix and  $i \in \mathbb{N}_m$  and  $\eta \in \sigma(\Lambda \setminus \Lambda_i^{m_1})$ . Then we can construct 1) a program  $q$  that given any  $\lambda_i$  outputs  $\eta$  and 2) a program  $q'$  that given  $\eta$  outputs  $\lambda_i$  and  $l(p) \stackrel{\pm}{=} l(q) \stackrel{\pm}{=} l(q')$ . The program  $q$  operates as follows. Given input  $x$ , it runs  $p(\langle x, 1 \rangle)$  and if  $x = \lambda_i$ , gets a  $y \in \sigma(\Lambda \setminus \Lambda_i^{m_1})$ .  $q$  also has encoded in it as  $2m\bar{m}$  in  $< 4m \log m$  bits the order in which  $\lambda_j, j \neq i$  appears in  $y$ , and in which they should appear in  $\eta$  as (for definition of  $\bar{m}$  see Section 2). It then uses that to decode  $y$ , and output  $\eta$ . The program  $q'$  operates as follows - given input  $x$ , it runs  $p(\langle x, m-1 \rangle)$ , needing  $< 2 \log m$  bits to encode  $m-1$  as  $\bar{m}-1$ . If  $x := \eta$   $q'$  gets  $\lambda_i$  and just outputs it. By construction  $l(p) \stackrel{\pm}{=} l(q) \stackrel{\pm}{=} l(q')$ ; furthermore  $\arg E_0^m(\lambda_{1,m})$  is a program satisfying the properties of  $p$ , while, since  $\lambda_i$  and  $\eta$  were chosen arbitrarily,  $\arg E_1^m(\lambda_{1,m})$  is a program satisfying either the properties of  $q$  or  $q'$ . Hence  $l(\arg E_1^m(\lambda_{1,m}))$  is at most  $l(\arg E_0^m(\lambda_{1,m}))$  and so we have the above inequality.

Now we prove

$$E_0^m(\lambda_{1,m}) \stackrel{+}{\leq} E_1^m(\lambda_{1,m}) .$$

Let  $E1m = E_1^m(\lambda_{1,m})$ . We will construct a program  $q_{E1m}$  with  $l(q_{E1m}) \stackrel{\pm}{=} E1m$  that will have the same outputs as  $\arg E_0^m(\lambda_{1,m})$  on  $\langle \langle y \rangle^{m_1}, m_1 \rangle, y \in \sigma(\Lambda_i^{m_1}), 0 < m_1 < m, 1 \leq i \leq \binom{m}{m_1}$ . For this, we need the set  $L$  the sets  $A_x$

$$\begin{aligned} L &:= \{\{x_{1,m}\} : E_1^m(x_{1,m}) \leq E1m\} \\ A_x &:= \{\{z_{1,m-1}\} : \{x, z_{1,m-1}\} \in L\} . \end{aligned}$$

and colors  $\mathcal{C}_{E1m}$ , defined using the set  $B$ .

$$\begin{aligned} \mathcal{C}_{E1m} &:= \{pj : p \in B, j \leq m\}, \text{ where,} \\ B &:= \{p : p(x) = \langle y_{1,m-1} \rangle^{m-1}, \{y_{1,m-1}\} \in A_x, l(p) \leq E1m\} . \end{aligned}$$

By using  $E1m$  and  $m$ ,  $q_{E1m}$  will construct  $L$  dynamically and color each element of  $L$  using colors from  $\mathcal{C}_{E1m}$ , so that if a string  $x_i$  appears in multiple  $m$  tuples in  $L$ , then each  $m$  tuple will have a different color from the  $m$  tuples - this is stated more precisely below.

To perform the coloring as above, we run all programs  $p$  with  $l(p) \leq E1m$  on all  $x \in \mathcal{A}^*$  in parallel. If we find  $p(x) = y$ , we record the tuples  $(p, (w_{1,m-1}), y)$  and  $(p, x, (z_{1,m-1}))$ , where  $x = \langle w_{1,m-1} \rangle^{m-1}$  and  $y = \langle z_{1,m-1} \rangle^{m-1}$ . If we find a  $x_{1,m}$  such that we have recorded  $(p_{x_i,y}, x_i, y)$  and  $(p_{y,x_i}, y, x_i)$  for each  $x_i$  and  $\forall y \in \sigma(\{x_{1,m}\} \setminus \{x_i\})$ , then we add each of the  $p_{x_i,y}, p_{y,x_i}$ s to  $B$  and add the corresponding colors to  $\mathcal{C}_{E1m}$ . We add  $X := \{x_{1,m}\}$  to  $L$  and color it using a variation of First-Fit in Theorem 3.10 as follows. Denote by  $\mathcal{C}(X)$  the color assigned to  $X$ . Then  $\mathcal{C}(X)$  is set to the first  $\gamma \in \mathcal{C}_{E1m}$  such that  $\forall x \in X$ , if  $x \in X', X' \in L$ , then  $\gamma \neq \mathcal{C}(X')$ . So given any  $x \in X$ , and  $\mathcal{C}(X)$ ,  $q_{E1m}$  can reconstruct and color  $L$  as above and hence find  $X$ .

To see that  $\mathcal{C}_{E1m}$  has enough colors: Let  $\Delta_L := \max_x |A_x|$ . For each  $\kappa \in A_x$ ,  $\exists p_\kappa \in B$ ,  $p_\kappa(x) = \langle y \rangle^{m-1}$ ,  $y \in \sigma(\kappa)$  and  $p_{\kappa'} \neq p_\kappa \forall \kappa' \in A_x, \kappa' \neq \kappa$ . Therefore  $|\mathcal{C}_{E1m}| \geq m\Delta_L$ . Also, from the construction method for  $L$  above,  $|\mathcal{C}_{E1m}^t| \geq m\Delta_L^t$  for the estimates at each step  $t$  of the construction process. When coloring  $X$  at step  $t$ , each  $x \in X$  has used  $\leq \Delta_L^t - 1$  colors previously. So, as  $|X| = m$ , First-Fit will require at most  $m(\Delta_L^t - 1) + 1$  colors to assign a valid color to  $X$ .

Now  $\max_{\gamma \in \mathcal{C}_{E1m}} l(\gamma) \leq E1m + l(m)$  ( $l(m) = \lfloor \log(m+1) \rfloor$  (Li and Vitanyi, 1997)), and with  $m$  as a constant, this becomes  $E1m + c$ . Like  $q_{E1}$  from Theorem 3.10,  $q_{E1m}$  can encode  $E1m$ ,  $m$ , and the color  $\gamma_\Lambda = pj$  for  $\Lambda$  in itself as  $p\bar{j}\bar{m}0^{E1m-l(p)}$ . Using this,  $q_{E1m}$  can dynamically construct  $L$ ,  $\mathcal{C}_{E1m}$  and color  $L$ . For input  $\langle x, m_1 \rangle$ ,  $0 < m_1 < m$ ,  $q_{E1m}$  decodes  $x$  with  $\beta_j := \langle x \rangle_j^{m_1}$ ,  $0 < j < m_1$ . By construction of  $L$ , using any  $\beta_j$  and  $\gamma_\Lambda$ ,  $q_{E1m}$  can find  $\Lambda$  in  $L$ , and output  $\langle y \rangle^{m-m_1}$ ,  $y \in \sigma(\Lambda \setminus \{\beta_{1,m_1}\})$ , which is what is required. This proves, with  $m$  as a constant  $E_0^m(\lambda_{1,m}) \stackrel{\dagger}{\leq} E_1^m(\lambda_{1,m})$  and  $E_0^m(\lambda_{1,m}) \stackrel{\dagger}{\leq} E_1^m(\lambda_{1,m}) + 3\lceil \log m \rceil$  otherwise. This and the first inequality completes the proof.  $\square$

## 4 Universal Bayesian Transfer Learning

In this section we will discuss how to do transfer learning using Bayes mixtures over enumerable subsets  $\mathcal{V}'$  of  $\mathcal{V}$ , which we consider as our task spaces.

That is we will present a transfer learning analogue of Solomonoff Induction (Solomonoff, 1978). First we will discuss relevant error bounds for Bayesian sequence prediction, and then we will present our transfer learning methods.

#### 4.1 Bayesian Convergence Results

A Bayes Mixture  $\mathbf{M}$  over  $\mathcal{V}'$  is defined by:

$$\mathbf{M}(x) := \sum_{\mu_i \in \mathcal{V}'} \mu_i(x) W(\mu_i) . \quad (4.1)$$

where  $W$  is a prior with  $W(\mu_i) \geq 0$  for each  $\mu_i$  and  $\sum_{\mu_i \in \mathcal{V}'} W(\mu_i) \leq 1$ . Then the following impressive result holds true:

**Theorem 4.1 (Solomonoff, Hutter)**  $\forall \mu_j \in \mathcal{V}' :$

$$\sum_{t=0}^{\infty} \sum_{x_{1:t}} \mu_j(x_{1:t}) \left( \sum_{a \in \mathcal{A}} [\mathbf{M}(a|x_{1:t}) - \mu_j(a|x_{1:t})]^2 \right) \leq -\ln W(\mu_j) . \quad (4.2)$$

Note that, for finite  $-\ln W(\mu_j)$ , convergence of  $\mathbf{M}_W$  to the target  $\mu_j$  is rapid in the following sense. (1) The expected number of times  $t$   $|\mathbf{M}(a|x) - \mu_j(a|x)| > \epsilon$  is  $\leq -\ln W(\mu_j)/\epsilon^2$ . (2) The probability that the number of  $\epsilon$  deviations  $> -\ln W(\mu_j)/\epsilon^2 \delta$  is  $< \delta$ . Now define:

**Definition 4.2** For a prior  $W$ , the **error bound** under Theorem 4.1 is defined as  $\text{Eb}_W(\mu) := -\ln W(\mu)$ . A prior  $W$  is said to be **universally optimal** in some class  $C$  if for all priors  $W' \in C$ ,  $\forall \mu \in \mathcal{V}'$ ,  $\text{Eb}_W(\mu) \stackrel{+}{\leq} \text{Eb}_{W'}(\mu)$ .

Recall that we wish to investigate what is the best we can do – i.e. most amount of similarity we can uncover, most amount of information we can transfer etc. – given that we have infinite time and memory. Because of this, we will consider only lower semicomputable priors, as this class contains all the computable priors (i.e. the ones we can actually use in practice), is enumerable and hence helps keep  $\mathbf{M}$  lower semicomputable as well.

Theorem 4.1 was first proved by Solomonoff (1978) for  $\mathcal{V}' = \mathcal{V}$  and  $\mathcal{A} = \mathcal{B}$ , and was then extended to arbitrary finite alphabets,  $\mathcal{V}'$ s and bounded loss functions by Hutter (2003, 2004). Hutter (2003) also showed that Bayes mixtures are Pareto optimal, and that if  $\mu_j \notin \mathcal{V}'$ , but there is a  $\rho \in \mathcal{V}'$  such that  $\forall t \in \mathbb{N}$ , the  $t^{\text{th}}$  order KL divergence between  $\rho$  and  $\mu_j$  is  $\leq k$ , then  $\text{Eb}_W(\mu_j) \leq -\ln W(\rho) + k$ .

We now mention a particularly interesting prior, which is the Solomonoff-Levin prior:  $2^{-K(\mu_i)}$ . In this case, the error bound is  $K(\mu_j) \ln 2$ . This is intuitively

appealing because it shows that the smaller the code for  $\mu_j$ , the smaller the bound, which is an instantiation of Occam’s razor. In addition, for any other lower semicomputable prior  $W$ , the error bound  $-\ln W(\mu_j)$  is upper semicomputable, and  $-\ln W()/\ln 2$  satisfies the conditions for Lemma 3.5 (with  $y = \varepsilon$  and  $W(x)$  undefined if  $x \notin \mathcal{V}'$ ), so:

$$K(\mu_j) \ln 2 \stackrel{+}{\leq} -\ln W(\mu_j) . \quad (4.3)$$

Therefore we have:

**Theorem 4.3** *The Solomonoff-Levin prior is universally optimal in the class of lower semicomputable priors.*

## 4.2 Universal Sequential Transfer Learning

We assume that we are given tasks  $\varphi_1, \varphi_2, \dots, \varphi_{m-1}$ ,  $\varphi_i \in \mathcal{V}$ , as previously learned tasks. We do not care about how these were learned - for instance each  $\varphi_i$  may be a weighted sum of elements of  $\mathcal{V}'$  after having observed a finite sequence  $x^{(i)}$  (Hutter, 2003, Sect. 2.4) or each  $\varphi_i$  may be given by the user. Let  $\varphi := \langle \varphi_1, \varphi_2, \dots, \varphi_{m-1} \rangle^{m-1}$ . The aim of transfer learning is to use  $\varphi$  as prior knowledge when predicting for the  $m^{\text{th}}$  task with some unknown generating semimeasure  $\mu \in \mathcal{V}'$ . Given this, a transfer learning scheme is just a conditional prior over  $\mathcal{V}'$ , and it may or may not be based on a distance function. So,

**Definition 4.4** *A transfer learning scheme is a partial, lower semicomputable prior  $W(\mu|\varphi)$  with  $\forall \mu \in \mathcal{V}', W(\mu|\varphi) \geq 0$ ,  $\sum_{\mu \in \mathcal{V}'} W(\mu|\varphi) \leq 1$ , and  $W(x|\varphi)$  undefined for  $x \notin \mathcal{V}'$ . A symmetric distance  $D$  based transfer learning scheme is a transfer learning scheme  $W_D(\mu_i|\varphi)$  with  $W_D(\mu_i|\varphi) := g(D(\mu_i, \varphi))$  for a symmetric function  $D : \mathcal{A}^* \times \mathcal{A}^* \rightarrow \mathbb{R}$  and  $g : \mathbb{R} \rightarrow [0, 1]$ .*

$W_D$  is defined in terms of  $g$  because we do not want to put restrictions on how the distance function  $D$  may be used to induce a prior, or even what constraints  $D$  must satisfy other than being symmetric.

**Definition 4.5** *Our universal transfer learning scheme is the prior  $\xi_{\text{TL}}(\mu_i|\varphi) := 2^{-K(\mu_i|\varphi)}$ . Our TL distance based universal transfer learning scheme for Bayes mixtures over  $\mathcal{V}'$  is the prior  $\xi_{\text{DTL}}(\mu_i|\varphi) := 2^{-E_1(\mu_i, \varphi)}$ .*

For  $\xi_{\text{DTL}}$  we use  $E_1$  instead of  $E_1^m$  because  $E_1$  measures the amount of information between the  $m^{\text{th}}$  task and previous  $m - 1$  tasks, which is what we want, whereas  $E_1^m$  measures amount of information between all possible disjoint groupings of tasks, and hence it measures more information than we are interested in.  $\xi_{\text{DTL}}$  is a prior since  $\sum_{\mu_i \in \mathcal{V}'} 2^{-E_1(\mu_i, \varphi)} \leq \sum_{\mu_i \in \mathcal{V}'} 2^{-K(\mu_i|\varphi)} \leq 1$

( $K(\mu_i|\varphi)$  being lengths of programs). As  $E_1(\cdot, \varphi)$  and  $K(\cdot|\varphi)$  are upper semicomputable,  $\xi_{\text{DTL}}$  and  $\xi_{\text{TL}}$  are lower semicomputable.

So in the Bayesian framework we consider,  $\xi_{\text{DTL}}$  automatically transfers the right amount of information from previous tasks to a potential new task by weighing it according to how related it is to older tasks.  $\xi_{\text{TL}}$  is less conceptually pleasing as  $K(\mu_i|\varphi)$  is not a distance, and a goal of TL has been to define transfer learning scheme using TL distance functions. But as we see below,  $\xi_{\text{TL}}$  is actually more generally applicable for sequential transfer.

**Theorem 4.6**  $\xi_{\text{TL}}$  and  $\xi_{\text{DTL}}$  are universally optimal in the class of transfer learning schemes and distance based transfer learning schemes respectively.

**PROOF.** Let  $W$  be a transfer learning scheme.  $\text{Eb}_{\xi_{\text{TL}}}(\mu) = K(\mu|\varphi) \ln 2$  and  $\text{Eb}_W(\mu) = -\ln W(\mu|\varphi)$ .  $W$  is lower semicomputable, which implies  $-\ln W$  is upper semicomputable;  $-\ln W()/\ln 2$ , defined only on  $\mathcal{V}'$ , satisfies the requisite conditions for Lemma 3.5 with  $y = \varphi$ , and so  $\text{Eb}_{\xi_{\text{TL}}}(\mu) \stackrel{+}{\leq} \text{Eb}_W(\mu)$ .

Let  $W_D$  be a distance based transfer learning scheme.  $\text{Eb}_{\xi_{\text{DTL}}}(\mu) = E_1(\mu, \varphi) \ln 2$  and  $\text{Eb}_{W_D}(\mu) = -\ln W_D(\mu|\varphi)$ .  $-\ln W_D$  is upper semicomputable as  $W_D$  is lower semicomputable;  $-\ln W_D$  is symmetric, and restricted to  $\mathcal{V}'$ ,  $-\ln W_D()/\ln 2$  satisfies the Kraft inequality condition in Definition 3.8; therefore  $-\ln W_D()/\ln 2 \in \mathcal{D}$ . Now by (3.2)  $\text{Eb}_{\xi_{\text{DTL}}}(\mu) \stackrel{+}{\leq} \text{Eb}_{W_D}(\mu)$ .  $\square$

We now make two interesting observations. First, for any distance based prior  $W_D$ , the error bound for a  $\mu \in \mathcal{V}$  is given by  $-\ln W_D(\mu) / \ln 2$ . This function is an admissible distance as it is both lower semicomputable and satisfies the Kraft inequality as required in Definition 3.8. Now recall that in Definition 4.4 we did not require that  $D$  be an admissible distance, while in Theorem 3.9 and the subsequent discussion we ignored the fact that  $E_1$  may or may not minorize non-admissible distances. This seems to detract from using  $E_1$  as the ideal transfer learning distance. The above result now allays this concern because while  $D$  may not be admissible, the error bound when  $D$  is used to do transfer is an admissible distance. And so universality of  $E_1$  over only admissible distances suffice.

Second,  $\xi_{\text{TL}}$  does not transfer information when the tasks are not related in the following sense. By (4.3), the non-transfer universally optimal prior is  $2^{-K(\cdot)}$ , with error bound  $K(\mu) \ln 2$ . As  $K(\mu|\varphi) \stackrel{+}{\leq} K(\mu)$  (Lemma 3.4 part 1), we have

**Theorem 4.7**  $\xi_{\text{TL}}$  is universally optimal in the class of non-transfer priors.

So the theorem implies that when tasks are not related, the universal prior does not perform much worse than the universally optimal non-transfer prior. Hence  $\xi_{\text{TL}}$  does not transfer information when tasks are not related.

The above results formally solve the problems with transfer learning methods that we have mentioned before. These problems are, it is unclear how much information to transfer, when to transfer information and when not to transfer and if there is an optimal way to transfer. Theorem 4.6 and the subsequent discussion shows that there exists a universally optimal transfer scheme, which does almost as well as any reasonable transfer learning algorithm. It automatically determines how much information to transfer by virtue of using universally optimal distance functions to measure task relatedness. Theorem 4.7 further implies that, from a *formal perspective*, sequential transfer is always justified - i.e. it never hurts to transfer.

### 4.3 Extensions

#### 4.3.1 Parallel Transfer

Almost all practical applications of multitask learning methods have been in the batch setting rather than the online setting we consider in this paper. Furthermore, the majority of such methods are considered to be performing transfer *in parallel*, as in Caruana (1997) mentioned in the Introduction. Now we will give a qualitative analysis of these methods and how they relate to the work in the preceding sections.

In the Bayesian setting, current parallel transfer methods (Caruana, 1997; Baxter, 2000) may be interpreted as follows. There are  $m$  different learning problems and the  $i^{\text{th}}$  learning problem is to learn the distribution  $p_i$  that generated the *training sample*  $D_{n_i}^i$  for the  $i^{\text{th}}$  learning problem (see Sect. 6). For a fixed set of  $m$  training samples  $D_{n_1}^1, D_{n_2}^2, \dots, D_{n_m}^m$ , a Bayes mixture is used as the solution for each learning problem (just as above), and when solving the  $i^{\text{th}}$  learning problem, the prior used is  $W(\mu|D_{n_i}^i, 1 \leq i \leq m, i \neq j)$ . So this looks very similar to the priors in Def. 4.4, except that it is ‘conditioned’ on the training samples of the other  $m - 1$  tasks instead of  $\varphi$ .

Furthermore, recall that we considered  $\varphi$  to be encoding  $m - 1$  programs computing probability measures, and each  $D_{n_i}^i$ , along with an appropriate learning algorithm  $A$  (Bayesian or otherwise), also corresponds to a probability measure (i.e. the one learned using  $A$  applied to  $D_{n_i}^i$ ). Hence  $W(\mu|D_{n_i}^i, 1 \leq i \leq m, i \neq j)$  can be thought of as having the same form as the priors we have dealt with so far. This means that current parallel transfer methods used in practice are in fact clever ways of performing  $m$  sequential transfers in parallel.

Finally, we can also consider parallel transfer in the online setting of this paper and in that case there are in fact two distinct ways to interpret parallel transfer. However our analysis shows that neither of these two are interesting – the first turns out to be merely single task learning in a product space and the second is more or less the same as sequential transfer in the general case. So at this juncture we do not expound on this further and refer the curious reader to Mahmud (2008) for details.

#### 4.3.2 *Competitive Optimality of Universal Priors*

We now make a note regarding the additive constants that appear via  $\overset{+}{\leq}$  symbols in Theorems 4.3 and 4.6. These constants depend on  $K(W)$ , the Kolmogorov complexity of the prior  $W$  (conditional or otherwise) that our universal priors are competing against. This dependence appears via use of Lemma 3.5 in the proofs of these theorems. So if the complexity of  $W$  is large, then these optimality results are a bit suspect. There are two possible responses to this issue. The first is to observe that the priors used in practice almost always have very low Kolmogorov complexity (i.e. the priors are ‘reasonable’ (Hutter, 2004)). The second is to consider a slightly different and reasonable ‘competitive’ setup where it is quite easy to shift this dependence to  $U$ , the reference universal Turing machine.

The latter can be achieved by proving a conditional version of Lemma 3.5 that states  $K(x|y, f) \overset{+}{\leq} f(x, y)$  (instead of  $K(x|y) \overset{+}{\leq} f(x, y)$ ) and where the constant of inequality now depends only on  $U$ , the reference universal Turing machine, (instead of  $K(f)$ ). If we now define conditional versions of the universal priors, then we can use this new result and prove competitive versions of the above theorems such that the additive constants of inequality now depend only on the reference universal Turing machine  $U$ . For instance if we define  $\xi_{\text{TL}}$  as  $2^{-K(\mu|\varphi, W)}$  instead of  $2^{-K(\mu|\varphi)}$  (Def. 4.5), where  $W$  is the prior  $\xi_{\text{TL}}$  is competing against, we can use the modification of Lemma 3.5 to show that  $\text{Eb}_{\xi_{\text{TL}}}(\mu) \overset{+}{\leq} \text{Eb}_W(\mu)$  where the constant of inequality now depends only on  $U$ . Similar comments also apply to Theorem 3.9 and we refer the interested reader to Mahmud (2008) for a fuller description of the above ideas.

So the use of these competitive versions of the universal priors induces a different interpretation of exactly what the universal methods are achieving. Now instead of being optimal, these methods are viewed as ones that are powerful ‘base’ methods that may be used any time, and are also ways to enhance any other high complexity method  $W$  that we may choose to use for a particular problem. So even if our prior knowledge  $W$  is wrong, the universal priors are guaranteed to not do too badly.

### 4.3.3 Further Extensions

We now mention some other extensions to this work that are also quite easy to derive. Hutter (2004) extends Theorem 4.1 to arbitrary bounded loss functions where the expected loss bound has the form  $-\ln W + \text{loss of the target measure}$ . So we can easily extend the transfer learning results to these cases by simple restatement of the results, and therefore do not expound on this further.

Hutter (2004) has also extended Theorem 4.1 to develop a Bayesian reinforcement learning agent (Dearden et al., 1998) framework called AIXI. In this case, when predicting the result of agent-environment interaction the expected squared error between the prediction of AIXI and the target *chronological semimeasure*  $\mu$  that ‘generate’ the behavior is also bounded by  $-\ln W(\mu)$ . So the transfer learning results also translate to this type of prediction readily. Unfortunately, the error between value functions of AIXI and any optimal policy that may exist is bounded by a multiplicative constant  $1/W(\mu)$ . So the convergence results are not as strong, but still very interesting given that active prediction is a much more difficult learning setup than passive prediction. Hence transfer learning results are also optimal up to such multiplicative constants.

A sub-branch of transfer learning that has recently received attention from researchers is cross-domain transfer (Swarup and Ray, 2006; Mihalkova et al., 2007; Taylor and Stone, 2007). In this, the goal is to transfer across tasks that are in different domains – i.e. defined over different input, output and hypothesis spaces. Current methods transfer across different domains by finding ‘structural similarity’ or defining ‘transform functions’ between different hypotheses from the different spaces. By representing each hypothesis in the different spaces as programs, we can measure similarity between these tasks using our methods quite easily. Indeed, structural similarity or transform functions are just ways to approximate Kolmogorov complexity and hence our work here gives a theory for cross-domain transfer in the Bayesian setting.

## 5 Kolmogorov Complexity of Functions

One natural definition of Kolmogorov complexity of a function  $f$  given string  $q$  is  $K'(f|q)$ , the length of the shortest program that computes  $f$  given  $q$  as extra information (Hutter, 2002, Sect. 7), (Grunwald and Vitanyi, 2004, Sect 2.2.3). So one objection to the definitions in Sect. 3 may be that, since we are interested in  $\mu \in \mathcal{V}$  as semimeasures (i.e. functions), perhaps we should define the complexity of  $\mu \in \mathcal{V}$  as  $K'(\mu|q)$ . However  $K'$  is not computable in the limit, so to address this concern, we establish another reasonable definition

of complexity of elements of  $\mathcal{V}$ ,  $K_{\mathcal{P}}$ . We then deflate this possible objection by showing that  $K_{\mathcal{P}}$  is in fact, up to a constant, equal to  $K$ . To motivate the definition of  $K_{\mathcal{P}}$ , we will begin by looking at a slight adaptation of a definition of Kolmogorov complexity of functions,  $K''$ . This was introduced and used by Hutter (2002), and was defined primarily to counter the noncomputability in the limit of  $K'$ .

To define  $K''$ , we need a formal system  $\mathcal{F}$  (Shoenfield, 1967) with axioms and inference rules in which we can formalize the notions of provability and Turing machines. We enclose formulas in  $\mathcal{F}$  in  $\S$   $\S$  and the proof of a formula  $s$  is a sequence of formulas such that, each formula in the sequence is either an axiom or derived from a previous formula in the sequence via the inference rules, and the last formula in sequence is  $\S$   $s$   $\S$ . The properties of  $\mathcal{F}$  we use are:

- The set of correct proofs in  $\mathcal{F}$  is enumerable.
- We can formulate the formula  $\S \forall x : \mu(x) \uparrow \alpha(x) \S$  which is true if and only if  $\forall x, U(\mu, x) \uparrow U(\alpha, x)$ .

Now we define  $K''(\mu|q)$

**Definition 5.1** For  $\mu \in \mathcal{V}, q \in \mathcal{A}^*$ ,

$$K''(\mu|q) := \min_r \{l(r) : r(q) = \alpha \text{ and } \exists \text{ proof } \S \forall x : \mu(x) \uparrow \alpha(x) \S\} .$$

The above definition means  $K''(\mu|q)$  is the length of the shortest program that given  $q$  as input, outputs a program  $\alpha$  that *provably* lower semicomputes (denoted by  $\uparrow$ ) the same semimeasure as  $\mu$ . This definition is slightly different from the one used by Hutter (2002), which is:

$$K''_H(\mu) := \min_r \{l(r) : \exists \text{ proof } \S \forall x : \mu(x) \uparrow r(x) \S\} .$$

This is the length of the shortest program that provably lower semicomputes  $\mu$ . However, now it is not entirely clear how the conditional should be defined. Intuition suggests we define it as

$$K''_H(\mu|q) := \min_r \{l(r) : \exists \text{ proof } \S \forall x : \mu(x) \uparrow r(\langle x, q \rangle) \S\} .$$

which is a little awkward. Hence, we adapt Hutter (2002)'s definition to our  $K''$  given above. It is easy to show, using standard methods, that  $K''_H \stackrel{\pm}{=} K''$  for a small constant of equality. That is: Given  $r$  that is a witness in Def. 5.1 we can construct  $r'(x) := (r(q))(x)$  to get  $K''_H(\mu|q) \stackrel{+}{\leq} K''(\mu|q)$ . Similarly, given  $r$  that is a witness in the definition of  $K''_H$ , we can define  $r'(q) := r(\langle q, \cdot \rangle)$  to show  $K''(\mu|q) \stackrel{+}{\leq} K''_H(\mu|q)$ , which proves the equality. The constant of equality is small because the definition of  $r'$  in each case requires just a little bit of extra code.

Note that both  $K''$  and  $K''_H$  are upper semicomputable because  $K$  is upper semicomputable and the set of correct proofs is enumerable. Now we have:

**Lemma 5.2** *Let  $\arg K''(\mu|q)$  denote the  $\alpha$  that is the witness in Definition 5.1. Then,*

$$\begin{aligned}
1) \quad & \forall \mu \in \mathcal{V}, q \in \mathcal{A}^*, K''(\mu|q) \leq K(\mu|q) \\
2) \quad & \forall n \in \mathbb{N}, q \in \mathcal{A}^* \exists \mu \in \mathcal{V} \text{ such that } K(\mu|q) - K''(\mu|q) \stackrel{+}{\geq} n \\
3) \quad & \forall \mu \in \mathcal{V}, q \in \mathcal{A}^*, K(\arg K''(\mu|q)) \stackrel{\pm}{=} K''(\arg K''(\mu|q))
\end{aligned} \tag{5.1}$$

**PROOF.**

**Part 1.** This follows from definition since each  $\mu \in \mathcal{V}$  provably computes the same function as itself.

**Part 2.** Fix  $q \in \mathcal{A}^*$ ,  $\varphi \in \mathcal{V}$ , and  $n \in \mathbb{N}$ . Now by the theory of random strings (see Li and Vitanyi (1997)), there exists infinitely many incompressible strings - i.e. strings  $s$  such that  $K(s|\varphi, K(\varphi|q), q) \geq l(s)$ . Let  $l(s) = n$ , and construct a  $\mu \in \mathcal{V}$  which is just  $\varphi$  with  $s$  encoded in it at a fixed position  $t$ . Now  $K(\mu|q) \stackrel{\pm}{=} K(s, \varphi|q)$ , since, using  $t$ , given a program to generate  $\mu$  given  $q$ , we can recover  $\varphi$  and  $s$  from it, and given a program to generate  $\langle s, \varphi \rangle$  given  $q$ , we can construct  $\mu$ . By Lemma 3.4 part 5, we get

$$K(s, \varphi|q) \stackrel{\pm}{=} K(s|\varphi, K(\varphi|q), q) + K(\varphi|q) .$$

By definition  $K''(\mu|q) \leq K(\varphi|q)$ , so we get:

$$\begin{aligned}
K(\mu|q) - K''(\mu|q) & \stackrel{\pm}{=} K(\varphi, s|q) - K''(\mu|q) \stackrel{\pm}{=} K(s|\varphi, K(\varphi|q), q) + K(\varphi|q) - \\
K''(\mu|q) & \geq n + K(\varphi|q) - K''(\mu|q) \geq n + K(\varphi|q) - K(\varphi|q) = n .
\end{aligned}$$

**Part 3.** Follows from definition.

□

Parts 1 and 2 in the lemma show that the  $K''$ s can uncover much more similarity between tasks than  $K$ . However, there is no advantage to using  $K''$  for Bayesian transfer learning, as for any enumerable set  $\mathcal{V}'$ , the set of programs  $\mathcal{V}'_{proof}$  that are provably equal to the elements of  $\mathcal{V}'$  is also enumerable (because the set of correct proofs in  $\mathcal{F}$  are enumerable). Therefore we get that for any  $\mu \in \mathcal{V}'$ ,  $\arg K''(\mu|q)$  is in  $\mathcal{V}'_{proof}$ . Since the error bound in Bayes mixtures

depends only on the weight assigned to the generating semimeasure , from part 3 of the above lemma, substituting  $\mathcal{V}'$  with  $\mathcal{V}'_{proof}$  counteracts the benefit of using  $K''$ . Part 2 in the lemma seems to suggest that  $K''$  deserves further study and this will be done in future.

However for now, we note that in the definition of  $K''$  we require the witness to output a program that provably lower semicomputes the target function, but we do not require it to actually output the proof. This makes the witness nonconstructive in behavior, and to fix this we will now look at a slightly altered version of  $K''$  where the witness is also required to output this proof. It will then turn out that this new function and  $K$  are in fact equal upto a constant. We first define this altered version of  $K''$ :

**Definition 5.3** *The provable Kolmogorov complexity  $K_{\mathcal{P}}$  of  $\mu \in \mathcal{V}$  is defined as follows:*

$$K_{\mathcal{P}}(\mu) := \min_p \{l(p) : p(\varepsilon) = \langle \gamma, \pi \rangle \text{ where } \pi \text{ is a proof for } \S \forall x : \gamma(x) \uparrow \mu(x) \S\} .$$

So now, in addition to  $\gamma$  that provably computes  $\mu$ , we also require that the program output the corresponding proof. We can now define the conditional  $K_{\mathcal{P}}$  and the information distances. But first we need:

**Definition 5.4** *Let  $\mathcal{J}_{\mu}$  be the enumerable set of all  $\langle \gamma, \pi \rangle$  such that  $\pi$  is a proof of  $\S \forall x : \gamma(x) \uparrow \mu(x) \S$  ( $\mathcal{J}_{\mu}$  is enumerable because the set of all correct proofs in  $\mathcal{F}$  is enumerable).*

So  $\mathcal{J}_{\mu}$  is the equivalence class of all elements of  $\mathcal{V}$  that are provably equivalent to  $\mu$  (in terms of the relation  $\uparrow$ ), along with the proof of their equivalence. Now define:

**Definition 5.5** *The conditional  $K_{\mathcal{P}}$  is defined as:*

$$K_{\mathcal{P}}(\mu|\mu') := \min_p \{l(p) : \forall \tau' \in \mathcal{J}_{\mu'}, p(\tau') \in \mathcal{J}_{\mu}\} .$$

the **Information Distance**  $E_{\mathcal{P}0}$  is defined as:

$$E_{\mathcal{P}0}(\mu, \mu') := \min_p \{l(p) : \forall \tau \in \mathcal{J}_{\mu}, \forall \tau' \in \mathcal{J}_{\mu'}, p(\tau) \in \mathcal{J}_{\mu'}, p(\tau') \in \mathcal{J}_{\mu}\} .$$

and **Cognitive Distance**  $E_{\mathcal{P}1}$  is now defined as:

$$E_{\mathcal{P}1}(\mu, \mu') := \max\{K_{\mathcal{P}}(\mu|\mu'), K_{\mathcal{P}}(\mu'|\mu)\} .$$

The conditional  $K_{\mathcal{P}}(\mu|\mu')$  is the length of the shortest program that given any program  $\tau'$  that provably lower semicomputes the same semimeasure as  $\mu'$ , and a proof that it does so, outputs a program  $\tau$  that provably lower

semicomputes the same semimeasure as  $\mu$  and a proof that it does so. So the difference between this and standard definitions of conditional Kolmogorov complexity is that this function outputs lengths of programs that deals with equivalent classes of programs. The definitions of  $E_{\mathcal{P}_0}$  and  $E_{\mathcal{P}_1}$  are also similar.

$K_{\mathcal{P}}(\cdot)$  is upper semicomputable by the same reasoning  $K''(\cdot)$  is. However, our definition of conditional  $K_{\mathcal{P}}(\cdot|\cdot)$  is a lot stronger than definition of  $K''(\cdot|\cdot)$  as we require that the  $\arg K_{\mathcal{P}}(\mu|\mu')$  output an element in  $\mathcal{J}_{\mu}$  given *any* element of  $\mathcal{J}_{\mu'}$ . Because of this last condition  $K_{\mathcal{P}}(\cdot|\cdot)$  (and by similar reasoning  $E_{\mathcal{P}_1}$  and  $E_{\mathcal{P}_0}$ ) is not upper semicomputable. However, we can show the following:

**Lemma 5.6** *The following equalities now hold:*

- (1)  $K_{\mathcal{P}}(\mu) \stackrel{\pm}{=} K(\mu)$ .
- (2)  $K_{\mathcal{P}}(\mu_{1,n}) \stackrel{\pm}{=} K(\mu_{1,n})$ .
- (3)  $K_{\mathcal{P}}(\mu|\mu') \stackrel{\pm}{=} K(\mu|\mu')$ .
- (4)  $E_1(\mu, \mu') \stackrel{\pm}{=} E_{\mathcal{P}_1}(\mu, \mu')$ .
- (5)  $E_0(\mu, \mu') \stackrel{\pm}{=} E_{\mathcal{P}_0}(\mu, \mu')$ .
- (6)  $E_{\mathcal{P}_0}(\mu, \mu') \stackrel{\pm}{=} E_{\mathcal{P}_1}(\mu, \mu')$ .
- (7)  $K_{\mathcal{P}}(\mu|\arg K_{\mathcal{P}}(\rho)) \stackrel{\pm}{=} K(\mu|\arg K_{\mathcal{P}}(\rho))$ .
- (8)  $K_{\mathcal{P}}(\mu|\arg K_{\mathcal{P}}(\rho)) + K_{\mathcal{P}}(\rho) \stackrel{\pm}{=} K_{\mathcal{P}}(\mu, \rho) \stackrel{\pm}{=} K_{\mathcal{P}}(\rho|\arg K_{\mathcal{P}}(\mu)) + K_{\mathcal{P}}(\mu)$ .

That is the  $K_{\mathcal{P}}$  and  $K$  etc. are equal upto a constant, and hence  $K_{\mathcal{P}}$  satisfies some of the most interesting inequalities in Kolmogorov complexity theory. This shows that there exists a reasonable definition of Kolmogorov complexity of elements of  $\mathcal{V}$  for which it is equivalent to the Kolmogorov complexity of strings. That is, the objection stated in the beginning of the section, that since we are interested in  $\mu \in \mathcal{V}$  as semimeasures (i.e. functions), perhaps we should define the complexity of  $\mu \in \mathcal{V}$  as  $K'(\mu|q)$ , is not justified.

Now we prove Lemma 5.6, and we will do so using the following lemma:

**Lemma 5.7** *Let  $\mu_i \in \mathcal{V}, i \in \mathbb{N}_N$  and  $\rho_j \in \mathcal{V}, j \in \mathbb{N}_M$  for finite  $N$  and  $M$ . Then,*

- (1) *For all programs  $p$  such that*

$$\forall X, X := \langle \tau_{1,N} : \tau_i \in \mathcal{J}_{\mu_i} \rangle, p(X) = Y \text{ where } Y = \langle \kappa_{1,M} : \kappa_i \in \mathcal{J}_{\rho_i} \rangle .$$

*there exists a program  $q$  with  $l(p) \stackrel{\pm}{=} l(q)$  such that*

$$q(\langle \mu_{1,M} \rangle) = \langle \rho_{1,N} \rangle .$$

- (2) *Similarly given any program  $q'$  such that,*

$$q'(\langle \mu_{1,M} \rangle) = \langle \rho_{1,N} \rangle .$$

there exists a program  $p'$  with  $l(p') \stackrel{\pm}{=} l(q')$  such that

$$\forall X, X := \langle \tau_{1,N} : \tau_i \in \mathcal{J}_{\mu_i} \rangle, p'(X) = Y \text{ where } Y = \langle \kappa_{1,M} : \kappa_i \in \mathcal{J}_{\rho_i} \rangle .$$

**PROOF.** Let  $p$  be a program as above. Then we can construct the requisite program  $q$  as follows. Program  $q$  given any  $\langle \mu_{1,N} \rangle$ , constructs  $X := \langle \tau_{1,N} : \tau_i \in \mathcal{J}_{\mu_i} \rangle$  by setting  $\tau_i := \langle \mu_i, \pi_i \rangle$  where  $\pi_i$  is simply the statement  $\S \forall x : \mu_i(x) \uparrow \mu_i(x) \S$ .  $q$  then runs  $p(X)$  to get  $Y := \langle \kappa_{1,M} : \kappa_i \in \mathcal{J}_{\rho_i} \rangle$ . We extract  $\rho_i$  from  $\kappa_i := \langle \gamma, \pi \rangle$  decoding  $\rho_i$  in  $\pi$  as the last statement in  $\pi$  is of the form  $\S \forall x : \gamma(x) \uparrow \rho_i(x) \S$ . Then program  $q$  outputs  $\langle \rho_{1,M} \rangle$ . As  $q$  has only constant amount of additional code,  $l(q) \stackrel{\pm}{=} l(p)$ .

Let  $q'$  be a program as above. Then we can construct the requisite program  $p'$  as follows. Program  $p'$  given any  $X := \langle \tau_{1,N} : \tau_i \in \mathcal{J}_{\mu_i} \rangle$  extracts  $\mu_i$  from  $\tau_i = \langle \gamma, \pi \rangle$  using  $\pi$  and  $\gamma$ . It then runs  $q(\langle \mu_{1,N} \rangle)$  to get  $\langle \rho_{1,M} \rangle$ .  $p'$  then outputs  $Y$  where  $Y := \langle \kappa_{1,M} \rangle$  where  $\kappa_i := \langle \rho_i, \pi \rangle$  and  $\pi$  is simply the statement  $\S \forall x : \rho_i(x) \uparrow \rho_i(x) \S$ .  $\square$

**PROOF.** [Proof of Lemma 5.6] We prove each part in turn.

**Part 1.** With  $\mu_i$  set to  $\varepsilon$  in Lemma 5.7 part 1 we get  $K(\mu) \stackrel{+}{\leq} K_{\mathcal{P}}(\mu)$ ; by part 2 of Lemma 5.7 we get  $K_{\mathcal{P}}(\mu) \stackrel{+}{\leq} K(\mu)$ . Hence this proves part 1 of this lemma.

**Part 2.** This follows from Lemma 5.7 with  $\mu_i := \epsilon$  via similar reasoning as part 1.

**Part 3.** This follows from Lemma 5.7 with  $\mu_i := \mu'$  via similar reasoning as part 1.

**Part 4.** This follows from part 3 and definitions of  $E_1$  and  $E_{\mathcal{P}1}$ .

**Part 5.** This can be easily proved using the method in Lemma 5.7.

**Part 6.** This follows from parts 4, 5 and Theorem 3.10.

**Part 7.** This follows from Lemma 5.7 with  $\mu_i := \arg K_{\mathcal{P}}(\rho)$  via similar reasoning to parts 1 and 3.

**Part 8.** This now follows by Lemma 3.4 part 4, and parts 1, 2 and 3.  $\square$

So, in our definition of  $K''$ , if we include the additional information required to make the witness constructive by outputting the proof, we immediately get the equivalence between Kolmogorov complexity of functions and bit strings. We should also note that the above applies for  $\mu \notin \mathcal{V}$  and notions of computability other than lower semicomputability.

## 6 Practical Approximations

In this section we develop practical approximations to the universally optimal sequential transfer prior  $\xi_{\text{TL}}$  of Sect. 4. The goal here is to investigate if the theory can be of help in constructing practically effective transfer algorithms. We consider transfer learning using Bayesian binary decision trees (Breiman et al., 1993; Chipman et al., 1998) and in this setting develop approximations to our distance measure and universal transfer learning prior. We then apply our approximations in 9 transfer experiments to transfer across 7 arbitrarily chosen data sets from the UCI machine learning repository (Newman et al., 1998). The arbitrary choice of data sets make our experiments the most general transfer experiments to date. This batch of experiments is in fact part of a larger set of 144 transfer experiments with similar results (Mahmud, 2008).

### 6.1 Bayesian Classification in Practice

We assume that we have a set of learning/prediction problems (see below), such that each problem  $i$  is defined by input space  $\mathcal{I}_i$  (possibly infinite), finite output space  $\mathcal{O}_i$  and finite hypothesis space  $\mathcal{H}_i$ . When we consider transfer learning later, each task will belong to one of these learning problems. Each  $\mathbf{h} \in \mathcal{H}_i$  is a computable conditional probability measure on  $\mathcal{O}_i$ , conditioned on elements of  $\mathcal{I}_i$ . So for  $y \in \mathcal{O}_i$  and  $x \in \mathcal{I}_i$ ,  $\mathbf{h}(y|x)$  gives the probability of output being  $y$  given input  $x$ . Given  $D_n = ((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)) \in (\mathcal{I}_i \times \mathcal{O}_i)^n$ , the probability of  $D_n$  according to  $\mathbf{h} \in \mathcal{H}_i$  (i.e. the *likelihood* of  $\mathbf{h}$ ) is given by:

$$\mathbf{h}(D_n) := \mathbf{h}(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_n) := \prod_{k=1}^n \mathbf{h}(y_k | x_k) .$$

Given  $D_n$ , the conditional probability of a new sample  $(x_\iota, y_\iota) \in \mathcal{I}_i \times \mathcal{O}_i$  for any conditional probability measure  $\mu$  is given, as usual, by:

$$\mu(y_\iota | x_\iota, D_n) := \frac{\mu((x_\iota, y_\iota) \oplus D_n)}{\mu(D_n)} := \frac{\mu(y_\iota, y_1, y_2, \dots, y_n | x_\iota, x_1, x_2, \dots, x_n)}{\mu(y_1, y_2, \dots, y_n | x_1, x_2, \dots, x_n)} . \quad (6.1)$$

where  $a \oplus b$  denotes inserting the element  $a$  in front of list  $b$ . So the learning problem is: given a training sample  $D_n$ , where for each  $(x_k, y_k) \in D_n$  the output  $y_k$  is assumed to have been chosen according a  $\mathbf{h} \in \mathcal{H}_i$ , learn  $\mathbf{h}$ . The prediction problem is to predict the label of new sample  $x_\iota$  using (6.1). In this setting the sequence of predictions are made over the sequence of outputs to be observed. As before, this is done using a Bayes mixture  $\mathbf{M}_W$  over  $\mathcal{H}_i$ :

$$\mathbf{M}_W(D_n) := \sum_{\mathbf{h} \in \mathcal{H}_i} \mathbf{h}(D_n)W(\mathbf{h}) \text{ with } \sum_{\mathbf{h} \in \mathcal{H}_i} W(\mathbf{h}) \leq 1 \quad (6.2)$$

Now for any  $X \in \mathcal{I}_i^\infty$  let us write  $D_n(x) \in X$  to mean that the inputs of  $D_n$ , in order, are the first  $n$  elements of  $X$ . Then the convergence bound of Theorem 4.1 is expressed as: for any  $\mathbf{h}_j \in \mathcal{H}_i$ , and any fixed  $x \in \mathcal{I}_i$  and any fixed  $X \in \mathcal{I}_i^\infty$ ,

$$\sum_{n=1}^{\infty} \sum_{D_n(x) \in X} \mathbf{h}_j(D_n) \sum_{y \in \mathcal{O}_i} [\mathbf{M}_W(y|x, D_n) - \mathbf{h}_j(y|x, D_n)]^2 \leq -\ln W(\mathbf{h}_j) \ . \quad (6.3)$$

Unfortunately, in practice it is often impossible to predict directly using the mixture because of difficult sums or integrals present in it. But note that we can rewrite the mixture in terms of the posterior  $Pr(\mathbf{h}|D_n)$  as follows:

$$\mathbf{M}_W(y_\iota|x_\iota, D_n) = \sum_{\mathbf{h} \in \mathcal{H}_i} \mathbf{h}(y_\iota|x_\iota, D_n)Pr(\mathbf{h}|D_n) = \sum_{\mathbf{h} \in \mathcal{H}_i} \mathbf{h}(y_\iota|x_\iota)Pr(\mathbf{h}|D_n) \ .$$

where

$$Pr(\mathbf{h}|D_n) := \frac{\mathbf{h}(D_n)W(\mathbf{h})}{Pr(D_n)} = \frac{\mathbf{h}(D_n)W(\mathbf{h})}{\sum_{\mathbf{h} \in \mathcal{H}_i} \mathbf{h}(D_n)W(\mathbf{h})} \ .$$

Now by sampling  $N$  points  $\rho_j$  from  $\mathcal{H}_i$  according to  $Pr(\mathbf{h}|D_n)$ , we approximate  $\mathbf{M}_W(y_\iota|x_\iota, D_n)$  by  $\widehat{\mathbf{M}}_W$ :

$$\widehat{\mathbf{M}}_W(y_\iota|x_\iota) := \frac{1}{N} \sum_{j=1}^N \rho_j(y_\iota|x_\iota) \ .$$

One the most popular class of methods for sampling from the posterior are the MCMC algorithms. In these methods we simulate a Markov chain with the posterior as the stationary/limiting distribution. When the chain converges to the posterior, we can simulate the former to sample from the latter. The Metropolis-Hastings algorithm, a popular MCMC method that we use in our experiments, is described in Sect. 6.3 and algorithm 1.

For an introduction to the Bayesian approach to machine learning, see Andrieu et al. (2003); Neal (2004); Mackay (2003). For full details on the Bayesian approach to statistics see Bernardo and Smith (1994), and for more details on Markov chain Monte Carlo see Gilks et al. (1996); Robert and Casella

(2005). For an introduction to Markov chains and fast mixing/convergence to the stationary distribution (and additional references), see Behrends (2000); Häggström (2002); Meyn and Tweedie (1993).

We seek to perform sequential transfer using decision trees. Since the Kolmogorov complexity  $K$  is computable only in the limit, to apply the methods and results in Sect. 4 to transfer using Bayesian decision trees, we need to approximate  $K$  and hence  $\xi_{\text{TL}}$ . Furthermore we also need to specify the spaces  $\mathcal{H}_i, \mathcal{O}_i, \mathcal{I}_i$  and how to compute the approximation of  $\xi_{\text{TL}}$ . We address each issue in turn.

## 6.2 Bayesian Decision Tree Model for Classification

We will consider transfer learning with Bayesian binary decision trees as our hypothesis space  $\mathcal{H}_i$ s and in this section we describe their model (Chipman et al, 1998). We assume that  $\mathcal{I}_i := [0, 1]^{|\mathbf{f}_i|}$ , where  $\mathbf{f}_i$  is a finite set of features, and finite  $\mathcal{O}_i := \mathbb{N}_{o_i}$ ,  $o_i \in \mathbb{N}$ . Decision trees partition the input space  $\mathcal{I}_i$  into a finite set of hypercubes defined by axis parallel hyperplanes. We assume that within each hypercube  $h_k$  the distribution over  $o_i$  classes is given by a multinomial distribution with parameter  $\vec{\theta}_k$ , a vector of  $o_i$  elements such that  $\sum_{j=1}^{o_i} \vec{\theta}_k(j) = 1$ . So for any sample  $D_n$ , the likelihood of  $h_k$  given a particular value of the parameter  $\vec{\theta}_k$  is given by:

$$h_k(D_n | \vec{\theta}_k) = \prod_{j=1}^{o_i} \vec{\theta}_k(j)^{m_{kj}} .$$

where  $m_{kj}$  is the number of pairs  $(x, y) \in D_n$  with  $x \in h_k$  and  $y = j$ . We do not include the  $\frac{n!}{\prod_j m_{kj}!}$  term above because we consider  $D_n$  to be a sequence of pairs, rather than a representative of any sample with  $m_{kj}$  elements of class  $j$ . We assume a Dirichlet prior over the parameters  $\vec{\theta}_k(j)$  (Friedman and Singer, 1998), for which the density function is given via hyperparameters  $\alpha_{kj}$ :

$$P(\vec{\theta}_k) := \frac{\Gamma(\sum_j \alpha_{kj})}{\prod_j \Gamma(\alpha_{kj})} \prod_j \vec{\theta}_k(j)^{\alpha_{kj}-1} .$$

where  $\Gamma$  is the gamma function,  $\Gamma(x) := \int_0^\infty t^{x-1} e^{-t} dt$ . We set  $\forall k, j \alpha_{kj} = 1$  which corresponds to the uniform prior over the  $o_i$  dimensional simplex. Now, we have

$$h_k(D_n) := \int h_k(D_n | \vec{\theta}_k) P(\vec{\theta}_k) d\vec{\theta}_k = \Gamma(o_i) \frac{\prod_j \Gamma(m_{kj} + 1)}{\Gamma(\sum_j m_{kj} + 1)} . \quad (6.4)$$

Therefore, the probability that  $y_\iota = a$  for input  $x_\iota \in h_k$  is just:

$$h_k(y_\iota = a | x_\iota, D_n) := \frac{h_k((y_\iota = a, x_\iota) \oplus D_n)}{h_k(D_n)} = \frac{m_{ka} + 1}{\sum_j m_{kj} + 1} . \quad (6.5)$$

Hence, given the partitions  $h_k$ , the predictive distribution is determined solely by the sample  $D_n$ , and so in Bayesian decision tree learning we learn what these partitions should be, i.e. we learn the *structure* of the tree. To that end,  $\mathbf{h}$  will now denote the tree structure only, consisting of  $M_{\mathbf{h}}$  partitions  $h_k$ .

The likelihood and posterior, respectively, of a tree  $\mathbf{h}$  are defined as:

$$\mathbf{h}(D_n) := \prod_{k=1}^{M_{\mathbf{h}}} h_k(D_n) \quad , \quad Pr(\mathbf{h} | D_n) = \frac{W(\mathbf{h}) \prod_{k=1}^{M_{\mathbf{h}}} h_k(D_n)}{Pr(D_n)} .$$

where  $W(\mathbf{h})$  is the prior over the tree structure. We now need to define  $W(\mathbf{h})$  to complete the definition of any Bayesian decision tree learning algorithm, transfer or otherwise. Towards that end, we now give a precise, recursive definition of the structure of a  $\mathbf{h} \in \mathcal{H}_i$ :

$$\mathbf{h} := \mathbf{n}_{root}, \quad \mathbf{n}_j := r_j \mathbf{C}_j \emptyset \emptyset \mid r_j \mathbf{C}_j \mathbf{n}_L^j \emptyset \mid r_j \mathbf{C}_j \emptyset \mathbf{n}_R^j \mid r_j \mathbf{C}_j \mathbf{n}_L^j \mathbf{n}_R^j .$$

So each decision tree is defined by its root node  $\mathbf{n}_{root}$ . Each node is either terminal or non-terminal, and consists of a rule  $r$  and a vector  $\mathbf{C}$  and two child nodes (sub-decision-trees). Each rule  $r$  is of the form  $f < v$ , where  $f \in \mathbf{f}_i$  and  $v$  is a value for  $f$ . Categorical features are converted to integer valued features for this purpose.  $\mathbf{C}$  is a vector of size  $o_i$ , with component  $j$  corresponding to the  $j^{th}$  class. The  $\mathbf{n}.\mathbf{C}(j)$  contains the value of  $m_{kj}$  for all the inputs in  $D_n$  that belong to the partition defined by its ancestors. We restrict the possible values of  $v$  for each feature to the values observed in the sample  $D_n$ , and so this makes the space of possible trees finite and brings the Bayesian decision tree framework discussed so far, in the framework of finite hypothesis space discussed in Sect. 6.1. Note that although all nodes do not need to store the rule, pointers and  $\mathbf{C}$ , we include them for each node because they were used in our implemented decision trees for book-keeping purpose.

Classification of an input  $x$  using a tree  $\mathbf{h}$  is performed by traversing it starting with the root node as the current node. If the current node is terminal, we use its  $\mathbf{C}$  to output the distribution over  $\mathcal{O}_i$  using (6.5). Otherwise, if  $x$  satisfies the rule at the current node, we traverse to its left child and we traverse to its right child if it does not satisfy the rule.

We are now in a position to define our prior over the tree structure in the next subsection.

### 6.3 Transfer Learning in Bayesian Decision Trees

To apply approximation of our transfer method to Bayesian decision trees, we need to define the approximation to Kolmogorov complexity  $K(\mathbf{h})$  of tree  $\mathbf{h}$ . Now, the size of each tree is  $S c_0$  where  $S$  is the number of nodes, and  $c_0$  is a constant, denoting the size of each rule entry, the outgoing pointers, and  $\mathbf{C}$ . Since  $c_0$  and the length of the program code  $p_0$  for computing the tree output are constants independent of the tree, we define the length/complexity of a tree as  $Kxt(\mathbf{h}) := S$ . So our approximation to  $K(\mathbf{h})$  is defined to be  $Kxt(\mathbf{h})$ . Hence, in the single task case, the prior we use is the approximation to the Solomonoff-Levin prior  $2^{-K(\mathbf{h})}$  and is given by:

$$W_{Kxt}(\mathbf{h}) := \frac{2^{-Kxt(\mathbf{h})}}{Z_{Kxt}} .$$

where the  $Z_{Kxt}$  is a normalization term. The  $Z_{Kxt}$  exists, here because  $\mathcal{H}$ s are finite, and in general because  $k_i = S c_0 + l(p_0) + O(1)$  gives lengths of programs, which are known to satisfy the Kraft inequality  $\sum_i 2^{-k_i} \leq 1$ .

For the transfer learning case, we need to approximate  $K(\cdot|\cdot)$ . We are going to consider transferring from  $m - 1$  previously learned trees, and so without loss of generality, assume that  $\mathbf{h} \in \mathcal{H}_{i_m}$  and  $\mathbf{h}' \in \mathcal{H}_{i_j}$ ,  $j < m$ . We now approximate  $K(\cdot|\cdot)$  using a function that is defined for a single previously learned tree as follows:

$$Kxt_2(\mathbf{h}|\mathbf{h}') := Kxt(\mathbf{h}) - d(\mathbf{h}, \mathbf{h}') .$$

where  $d(\mathbf{h}, \mathbf{h}')$  is the maximum number of overlapping nodes between  $\mathbf{h}$  and  $\mathbf{h}'$  starting from the their respective root nodes:

$$\begin{aligned} d(\mathbf{h}, \mathbf{h}') &:= d(\mathbf{h}.\mathbf{n}_{root}, \mathbf{h}'.\mathbf{n}_{root}) & d(\mathbf{n}, \emptyset) &:= 0 \\ d(\mathbf{n}, \mathbf{n}') &:= 1 + d(\mathbf{n}_L, \mathbf{n}'_L) + d(\mathbf{n}_R, \mathbf{n}'_R) & d(\emptyset, \mathbf{n}') &:= 0 . \end{aligned}$$

and so in the transfer learning case, the prior, when there is only one previously learned tree, is:

$$W_{Kxt_2}(\mathbf{h}|\mathbf{h}') := \frac{2^{-Kxt_2(\mathbf{h}|\mathbf{h}')}}{Z_{Kxt_2}} .$$

We can directly sample from both the priors defined so far, a fact that will become useful below when we sample from the posterior using a MCMC algorithm. We can do so by growing the decision tree dynamically. Call a  $\emptyset$  in  $\mathbf{h}$  a hole. Then for  $W_{Kxt}(\mathbf{h})$ , during the generation process, we first generate an integer  $k$  according to  $2^{-t}$  distribution (easy to do using a pseudo random number generator). Then at each step we select a hole uniformly at random

and then create a node there with two more holes and the rule generated randomly. Note that  $W_{K\hat{x}t}(\mathbf{h})$  gives equal probability to all trees of the same complexity  $k$ , while giving trees of complexity  $k$  half as probability as the trees of complexity  $k - 1$ . So the above procedure samples from the prior as it samples  $k$  according to  $2^{-t}$  and gives equal probability to every tree of size  $K\hat{x}t(\mathbf{h}) = k$  by growing the tree uniformly at random.

In the transfer learning case, for the prior  $W_{K\hat{x}t_2}(\mathbf{h}|\mathbf{h}')$  we first generate an integer  $k$  according to  $2^{-t}$  distribution. Then we generate a tree using the above procedure until we get a tree  $\mathbf{h}$  with  $K\hat{x}t_2(\mathbf{h}|\mathbf{h}') = k$ .  $W_{K\hat{x}t_2}(\mathbf{h}|\mathbf{h}')$  gives equal probability to all trees of the same conditional complexity  $K\hat{x}t_2(\mathbf{h}|\mathbf{h}')$  equal to  $k$ , while giving trees of complexity  $k$  half as probability as the trees of complexity  $k - 1$ . So the above procedure samples from the transfer prior as it samples  $k$  according to  $2^{-t}$  and gives equal probability to every tree of size  $K\hat{x}t_2(\mathbf{h}|\mathbf{h}') = k$  by growing the tree uniformly at random.

For  $m - 1$  previously learned trees  $\mathbf{h}_{1,m-1}$ , with  $\mathbf{h}_j \in \mathcal{H}_{i_j}$ , we define  $K\hat{x}t_m$  as an ‘averaging’ of the contributions of each of the  $m - 1$  previously learned trees:

$$K\hat{x}t_m(\mathbf{h}|\mathbf{h}_{1,m-1}) = -\log\left(\frac{1}{m-1}\sum_{i=1}^{m-1}2^{-K\hat{x}t_2(\mathbf{h}|\mathbf{h}_i)}\right).$$

In the transfer learning case for a fixed set of  $m - 1$  previously learned trees, the prior, and hence our approximation to  $\xi_{\text{TL}}(\mathbf{h}|\langle\mathbf{h}_{1,m-1}\rangle)$  is

$$W_{K\hat{x}t_m}(\mathbf{h}|\langle\mathbf{h}_{1,m-1}\rangle) := \frac{2^{-K\hat{x}t_m(\mathbf{h}|\mathbf{h}_{1,m-1})}}{Z_{K\hat{x}t_m}}.$$

which reduces to:

$$\frac{1}{[(m-1)Z_{K\hat{x}t_m}]} \sum_{i=1}^{m-1} 2^{-K\hat{x}t_2(\mathbf{h}|\mathbf{h}_i)}. \quad (6.6)$$

To sample from this, we can simply select one of the  $m - 1$  trees at random and then use the procedure for sampling from  $2^{-K\hat{x}t_2}$  to get the new tree. So, finally, the approximation of the transfer learning mixture  $\mathbf{M}_{\xi_{\text{TL}}}$  is now:

$$\mathbf{M}_{W_{K\hat{x}t_m}}(D_n) := \sum_{\mathbf{h} \in \mathcal{H}_{i_m}} \mathbf{h}(D_n) W_{K\hat{x}t_m}(\mathbf{h}|\langle\mathbf{h}_{1,m-1}\rangle).$$

So by (6.3), the error bound from Theorem 4.1 for  $\mathbf{M}_{W_{K\hat{x}t_m}}$  is given by  $K\hat{x}t_m(\mathbf{h}|\mathbf{h}_{1,m-1}) \ln 2 + \ln Z_{K\hat{x}t_m}$  (the  $\ln Z_{K\hat{x}t_m}$  term is a constant that is same for all  $\mathbf{h} \in \mathcal{H}_{i_m}$ ). In our experiments we actually used the exponent  $1.005^{-K\hat{x}t_m}$  instead of  $2^{-K\hat{x}t_m}$  above to speed up convergence of our MCMC method (see Sect. 6.5).

---

**Algorithm 1** Metropolis-Hastings Algorithm
 

---

- 1: Let  $D_n$  be the training sample;
- 2: Select the current tree/state  $\mathbf{h}_{cur}$  using the proposal distribution  $q(\mathbf{h}_{cur})$ .
- 3: **for**  $i = 1$  to  $J$  **do**
- 4:   Choose a candidate next state  $\mathbf{h}_{prop}$  according to the  $q(\mathbf{h}_{prop})$ .
- 5:   Draw  $u$  uniformly at random from  $[0, 1]$
- 6:   Set  $\mathbf{h}_{cur} := \mathbf{h}_{prop}$  if  $A(\mathbf{h}_{prop}, \mathbf{h}_{cur}) > u$ , where  $A$  is defined by

$$A(\mathbf{h}, \mathbf{h}') := \min \left\{ 1, \frac{\mathbf{h}(D_n) 2^{-K\hat{x}t_m(\mathbf{h}|\mathbf{h}_{1,m-1})} q(\mathbf{h}')}{\mathbf{h}'(D_n) 2^{-K\hat{x}t_m(\mathbf{h}'|\mathbf{h}_{1,m-1})} q(\mathbf{h})} \right\}$$

7: **end for**

---

As in standard Bayesian MCMC methods, the idea will be to draw  $N$  samples  $\mathbf{h}_{m_i}$  according to the posterior:

$$Pr(\mathbf{h}|D_n) := \frac{\mathbf{h}(D_n) W_{K\hat{x}t_m}(\mathbf{h}|\langle \mathbf{h}_{1,m-1} \rangle)}{Pr(D_n)} .$$

Then we will approximate  $\mathbf{M}_{W_{K\hat{x}t_m}}(y_\iota|x_\iota, D_n)$  by

$$\widehat{\mathbf{M}}_{W_{K\hat{x}t_m}}(y_\iota|x_\iota) := \frac{1}{N} \sum_{i=1}^N \mathbf{h}_{m_i}(y_\iota|x_\iota) .$$

We will use the standard Metropolis-Hastings algorithm to sample from  $\mathbf{M}_{W_{K\hat{x}t_m}}$  (see Sect. 6.2 for references). The algorithm is given in Table 1. The algorithm is first run for some  $J = T$ , to get the chain to converge, and then starting from the last  $\mathbf{h}_{cur}$  in the run, the algorithm is run again for  $J = N$  times to get  $N$  samples for  $\widehat{\mathbf{M}}_{W_{K\hat{x}t_m}}$ . In our experiments we set  $T = 1000$  and  $N = 50$ . We set  $q$  to our prior  $W_{K\hat{x}t_m}$ , and hence the acceptance probability  $A$  is reduced to  $\min\{1, \mathbf{h}(D_n)/\mathbf{h}'(D_n)\}$ . Note that every time after we generate a tree according to  $q$ , we set the  $\mathbf{C}$  entries to  $m_{kj}$  values obtained from the training sample  $D_n$ .

The main question that one will ask about the approximations presented here is just how good they are. One very meaningful and appropriate way to answer this question is by looking at how well these methods perform in practice. This is done in the next section where we show that our approximations perform quite well and enable us to perform very general and successful transfer experiments.

## 6.4 Experiments

### 6.4.1 Setup of the Experiments

In our experiments we used 7 data sets from the UCI machine learning repository (Newman et al., 1998). The data sets and a summary of their properties are given in Table 1. To show transfer of information, we chose 3 data sets to transfer to, and for each such data set we chose 3 other data sets at random to transfer from. So there were 9 pairs of data sets that we performed transfer experiments for. For each data set we split the samples into a training set and a testing set, consisting of 80% and 20% of the samples respectively. Then for each of the 9 pairs, we first learned the transfer-from data set using the prior  $W_{K_{\hat{t}}}$ , and then learned the transfer-to data set using the prior  $W_{K_{\hat{t}_m}}(\cdot | \langle h_{1,50} \rangle)$  where  $h_{1,50}$  are the trees sampled at the end of learning the transfer-from data set using the MH algorithm (as described at the end of the preceding section).

The results of these experiments are presented below. All error rates etc. reported were obtained by averaging over 10 different runs. Before each run the samples for each data set were shuffled randomly before splitting it up into the training and testing set. The training for each data set in every case was done using the training set, while the results reported are all for performance on the testing set. We also performed 135 other transfer experiments using other combinations of the percent of data used as training and testing sets (Mahmud, 2008). The results of these experiments also, by and large, exhibit the same properties as the results in this paper, which we now discuss and interpret.

### 6.4.2 Description and Interpretation of the Results

The last column of Table 1 gives the results of learning each of the data sets individually using the prior  $W_{K_{\hat{t}}}$ , our approximation to the Solomonoff-Levin prior. This set of experiments were performed to ensure that any observed improvement in performance is due to transfer and not because our single-task learner was faulty. From a survey of literature it seems the error rate for our classifier is always at least a couple of percentage points better than C4.5. As an example, for *ecoli* our classifier outperforms Adaboost and Random Forests from Breiman (2001), but is a bit worse than these for *german*.

The results for our transfer learning experiments are given in Table 2. As can be seen, for most of the transfer-to data sets, there is noticeable percentage improvement in performance (we use this metric following Mihalkova et al. (2007), who perform experiments most closely related to ours). The improvement for both *ecoli* and *bc-wisc* vary with respect to the transfer-from data set, however, for *aus* there is significant improvement for all the transfer-from

Database	# of S	# F	# Class.	Rf.	Err.,Std.
E-coli	336	7	8	ecoli	10.89%, 5.8
Yeast	1484	8	10	yeast	14.89%, 2.73
Australian Credit	690	14	2	aus	18.9%, 2.1
German Credit	1000	20	2	german	31.1%, 4.47
Hepatitis	155	19	2	hep	19.8%, 1.38
Breast Cancer, Wisc.	699	9	2	bc-wisc	8.99%, 2.3
Heart Disease, Cleve.	303	14	5	heart	23.3%, 1.8

Table 1

Summary of the data sets used in the transfer experiments. **S** means samples, **F** means features, **Class.** means classes and **Rf.** means the reference name used in the text for the corresponding data set. The **Err.,Std.** column gives the error and standard deviation for the data set using the single task  $W_{Kxt}$  prior.

data sets. While in the experiments above, we do not see any reduction in performance, in the 135 other experiments we have done (Mahmud, 2008), we see about 21 cases where performance is negative – but in almost all the cases it is  $> -2\%$  and never  $< -10\%$  (and mainly for the bc-wisc data set). Aside from this, the results for the other experiments are similar to the ones presented above. This seems to give evidence that the approximations to our transfer method in Sect. 4 are effective as they bear out the theoretical expectation that transfer should never hurt too much.

The results above can be explained as follows. MCMC methods are essentially stochastic exploration methods with nice convergence guarantees. When we perform transfer learning, we change the prior so that the MCMC algorithm explores certain areas of the hypothesis space with higher probability. Now the single-task learner with prior  $W_{Kxt}$  is simply a Bayesian learner with a Occam prior, assigning higher probability to smaller trees. Use of the  $W_{Kxt_m}$  priors during transfer forces the MH algorithm to focus its attention on trees with size possibly larger than recommended by the Occam prior. For this reason, transfer learning improves performance. The reason it does not degrade performance significantly is because, being a stochastic exploration method, MH automatically rejects larger trees that cause lower performance in the transfer learning case. Another reason is because,  $Kxt_m(\mathbf{h}|\mathbf{h}_{1,m-1}) \ln 1.005 + \ln Z_{Kxt_m} \stackrel{+}{\leq} Kxt(\mathbf{h}|\mathbf{h}_{1,m-1}) \ln 1.005 + \ln Z_{Kxt}$  (which are of course error bounds from Def. 4.2 for our transfer and non-transfer priors respectively) where the constant of inequality is because  $Z_{Kxt_m} \stackrel{\times}{\leq} Z_{Kxt}$ , and the constant is independent of  $\mathbf{h}$  and  $\mathbf{h}_{1,m-1}$ .

<b>To</b>	<b>Base Err.</b>	<b>From</b>	<b>Trans. Err.</b>	<b>% Improvement</b>
ecoli	10.89%, 5.8	yeast	8.96%, 4.95	17.72%
ecoli	10.89%, 5.8	german	9.55%, 5.14	12.3%
ecoli	10.89%, 5.8	bc-wisc	10.15%, 2.89	6.8%
bc-wisc	8.99%, 2.3	heart	8.85%, 2.47	1.56%
bc-wisc	8.99%, 2.3	aus	7.77%, 2.47	13.57%
bc-wisc	8.99%, 2.3	ecoli	7.27%, 2.58	19.13%
aus	21.93%, 4.03	german	14.31%, 3.4	24.64%
aus	21.93%, 4.03	ecoli	14.31%, 2.4	24.64%
aus	21.93%, 4.03	hep	15.04%, 2.1	20.8%

Table 2

Results of 9 transfer experiments, with each row giving the results for a particular transfer experiment. The **To** column gives the names of the transfer-to data sets. The **Base Err.** column gives the error and standard deviation for the transfer-to data set in the single task setting. The **From** column gives the names of the transfer-from data sets. The **Trans. Err.** column gives the error and standard deviation for the transfer-to data sets when transferring from the transfer-from data sets. The **% Improvement** column gives the difference between the Base Err. and Trans. Err. column as a percentage of the former.

### 6.5 Problems and Future Extensions

In this section we described a crude but computationally feasible approximation to our optimal but impractical sequential transfer prior  $\xi_{\text{TL}}$ . We will now look at the ways this approximation is crude and how these may be remedied.

Our approach to measuring similarity between trees is very basic. We only look at commonality that exists between the decision trees in terms of their structure and even this is very superficial. We ignore any deeper similarities that may exist between the trees, for instance in terms of their node values. Given that we would expect a good approximation to  $K$  to uncover and exploit such similarities, we need to develop more powerful and sophisticated measures of distance between trees. One possible way might be to follow Cilibrasi and Vitanyi (2005), who use standard compressors, say gzip, to approximate  $K$ . In their approach, the function  $C_{\text{gzip}}(x)$  gives the length of string  $x$  after being compressed by gzip, and this is used as the approximation  $K(x)$ . And then  $C_{\text{gzip}}(x|y) := C_{\text{gzip}}(xy) - C_{\text{gzip}}(y)$  measures how much addition of  $y$  to  $x$  helps in compressing  $x$ , and hence is an approximation to  $K(x|y)$ . Applying this to

our scenario,  $C_{gzip}(\mathbf{h}|\mathbf{h}')$  will give the relatedness between decision trees. While this is a very general approach to approximating  $K$ , it seems to us that it might be more useful to restrict ourselves to a group of specific machine learning domains and then derive compression based distance functions suitable for measuring relatedness between hypothesis that are suitable for the group.

Our measure of complexity also has theoretical problems. When coding a tree, we ignore the multiplicative constant  $c_0$  which denotes the size of each node, consisting of a rule entry, two outgoing pointers, and the vector  $\mathbf{C}$  for the node. From a theoretical perspective, we may ignore additive constants when measuring complexity but not multiplicative ones. This is because in the limit of increasing complexity, the additive constant becomes less and less significant, but the multiplicative constant does not. This was an oversight on our part, and it may be fixed by adopting a more sophisticated approximation to  $K$  as discussed in the above paragraph.

We used the base 1.005 instead of 2 in the priors in our experiments, which implies that when our MCMC method converges we will not be sampling from the approximation to the distribution recommended by the theory. This was done mainly because of practical reasons – the main problem with using base 2 was that it made longer trees exceedingly unlikely to be tested during MCMC. For instance it will take about 22700 MCMC steps before the probability that we do not test a tree with 15 nodes goes down below 0.5. This is very costly in terms of processing time, and especially so given the number of experiments we wished to perform. There does not seem to be a solution to this problem, as MCMC sampling based Bayesian methods, while very powerful, are fundamentally computationally intensive.

Another problem with our instantiation of the Metropolis-Hastings algorithm is that we do not do any formal convergence analysis when deciding when to stop. The  $J = 1000$  steps figure was chosen empirically – it seemed that in almost all the experiments, the quality of the current tree in the MH algorithm did not change much after about 1000 steps. It is possible that if we stop by using some standard convergence result from MCMC theory or by doing a convergence analysis, then we might get better performance. This needs to be checked in future versions of our practical work.

One final issue, which is actually common to all MCMC sampling based Bayesian learning algorithm, is that we are using  $N = 50$  points to approximate the mixture  $\mathbf{M}_{W_{K^t m}}$  via  $\widehat{\mathbf{M}}_{W_{K^t m}}$ . By the Central Limit Theorem, the distribution of  $\sqrt{N}(\mathbf{M}_{W_{K^t m}} - \widehat{\mathbf{M}}_{W_{K^t m}})$  converges weakly to  $Normal(0, \sigma^2)$  as  $N \rightarrow \infty$  where  $\sigma^2$  is the variance of  $\mathbf{h}(y_i|x_i)$  with respect to the posterior. So even when we are eventually sampling from the posterior, since we do not know  $\sigma^2$ , or how fast the rate of convergence is, it is not really clear how good an approximation  $\widehat{\mathbf{M}}_{W_{K^t m}}$  is for  $N = 50$ .

Indeed, because of the last three issues mentioned above, it seems that the behavior of our MCMC algorithm is probably closer to that of a stochastic search algorithm rather than an ideal sampling algorithm. Addressing these issues should be a focus of any future research on this topic. It is possible that the solution may be something as radical as leaving the Bayesian paradigm when implementing the theory of Sect. 4, and translating the theory to a paradigm of stochastic search over program space, such as Genetic Programming (Koza et al., 1999).

However, even with a crude and problematic approximation to the theory, we were able to perform 144 individual transfer learning experiments to transfer across seemingly unrelated tasks. Furthermore, we observed notable improvement in performance in large majority of the cases, and only a couple of notable reductions in performance. In the above sense, our transfer experiments were the most general to date. And so we may conclude that the experiments show that our Solomonoff-Induction based theory can provide very useful guidance in constructing powerful, practical transfer algorithms.

## 7 Discussion

In this paper we formally solved some of the key problems of transfer learning in the same sense that Solomonoff Induction solves the problem of inductive inference. We defined universal transfer learning distances and showed how these may be used to automatically transfer the right amount of information in our universally optimal Bayesian sequential transfer method. We also discussed various extensions to this theory. We note that the results and discussion in Sects. 3 and 4 also apply when instead of previous tasks we use arbitrary prior knowledge/bit strings. So our methods are also universally optimal Bayesian methods for using prior knowledge. We also briefly looked at Kolmogorov complexity of probability measures and under what conditions it is appropriate to define it as the complexity of a program computing it. Finally, we developed a simple approximation to the theory to perform the most general transfer experiments to date, showing that in addition to providing a theoretical foundation for transfer learning, the theory also provides fine guidance on how to construct effective practical transfer algorithms.

## Acknowledgements

We would like to thank Samarth Swarup, Sylvian Ray and Kiran Lakkaraju for their comments. The comments from three anonymous referees and Marcus Hutter were also invaluable in improving the paper.

## References

- Andrieu, C., de Freitas, N., Doucet, A., Jordan, M. I., 2003. An introduction to MCMC for machine learning. *Machine Learning* 50(1-2), 5–43.
- Baxter, J., March 2000. A model of inductive bias learning. *Journal of Artificial Intelligence Research* 12, 149–198.
- Behrends, E., 2000. *Introduction to Markov Chains: With Special Emphasis on Rapid Mixing*. Vieweg Verlag, Berlin.
- Ben-David, S., Schuller, R., 2003. Exploiting task relatedness for learning multiple tasks. In: *Proceedings of the 16<sup>th</sup> Annual Conference on Learning Theory*.
- Bennett, C., Gacs, P., Li, M., Vitanyi, P., Zurek, W., July 1998. Information distance. *IEEE Transactions on Information Theory* 44(4), 1407–1423.
- Bernardo, J. M., Smith, A. F. M., 1994. *Bayesian Theory*. Wiley, New York.
- Breiman, L., 2001. Random forests. *Machine Learning* 45, 5–32.
- Breiman, L., Friedman, J., Olshen, R., Stone, C., 1993. *Classification and Regression Trees*. Chapman and Hall, New York.
- Caruana, R., 1993. Multitask learning: A knowledge-based of source of inductive bias. In: *Proceedings of the 10<sup>th</sup> International Conference on Machine Learning*.
- Caruana, R., 1997. Multitask learning. *Machine Learning* 28, 41–75.
- Chaitin, G. J., 1975. A theory of program size formally identical to information theory. *Journal if the ACM* 22(3), 329–340.
- Chipman, H. A., George, E. I., McCulloch, R. E., 1998. Bayesian CART model search. *Journal of the American Statistical Association* 93, 935–948.
- Cilibrasi, R., Vitanyi, P., 2005. Clustering by compression. *IEEE Transactions on Information theory* 51(4), 1523–1545.
- Dearden, R., Friedman, N., Russell, S., 1998. Bayesian q-learning. In: *Proceedings of 15<sup>th</sup> National Conference on Artificial Intelligence (AAAI)*. AAAI Press, Menlo Park, CA.
- Friedman, N., Singer, Y., 1998. Efficient bayesian paramter estimation in large discrete domains. In: *Proceedings of 13<sup>th</sup> Neural Information Processing Systems Conference*.
- Gacs, P., 1974. On the symmetry of algorithmic information. *Soviet Mathematics Doklady* 15, 1477–1480.
- Gilks, W. R., Richardson, S., Spiegelhalter, D., 1996. *Markov Chain Monte Carlo in Practice*. Chapman and Hall, London.
- Grunwald, P., Vitanyi, P., 2004. Shannon information and Kolmogorov complexity. Submitted to *IEEE Transactions on Information Theory*.
- Hägglström, O., 2002. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press.
- Hutter, M., 2002. The fastest and shortest algorithm for all well defined problems. *International Journal of Foundations of Computer Science* 13(3), 431–443.
- Hutter, M., 2003. Optimality of Bayesian universal prediction for general loss

- and alphabet. *Journal of Machine Learning Research* 4, 971–1000.
- Hutter, M., 2004. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer-Verlag, Berlin.
- Juba, B., 2006. Estimating relatedness via data compression. In: *Proceedings of the 23<sup>rd</sup> International Conference on Machine Learning*.
- Kolmogorov, A., 1965. Three approaches to the quantitative definition of information. *Problems of Information and Transmission* 1(1), 1–7.
- Koza, J. R., Andre, D., III, F. H. B., Keane, M., 1999. *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman.
- Li, M., Chen, X., Ma, B., Vitanyi, P., 2004. The similarity metric. *IEEE Transactions on Information Theory* 50(12), 3250–3264.
- Li, M., Vitanyi, P., 1997. *An Introduction to Kolmogorov Complexity and its Applications*, 2nd Edition. Springer-Verlag, New York.
- Mackay, D., 2003. *Information Theory, Inference, and Learning Algorithms*, 1st Edition. Cambridge University Press.
- Mahmud, M. M. H., 2008. *Universal transfer learning*. Ph.D. thesis, University of Illinois at Urbana-Champaign.
- Mahmud, M. M. H., Ray, S., 2007. Transfer learning using Kolmogorov complexity: basic theory and empirical evaluations. In: *Proceedings of the 21<sup>st</sup> Neural Information Processing Systems Conference*.
- Meyn, S. P., Tweedie, R. L., 1993. *Markov chains and stochastic stability*. Springer-Verlag, New York.
- Mihalkova, L., Huynh, T., Mooney, R., 2007. Mapping and revising markov logic networks for transfer learning. In: *Proceedings of the 22<sup>nd</sup> National Conference on Artificial Intelligence (AAAI)*.
- Neal, R. M., 2004. Bayesian methods for machine learning, NIPS tutorial.
- Newman, D., Hettich, S., Blake, C., Merz, C., 1998. UCI repository of machine learning databases.  
URL <http://www.ics.uci.edu/~mllearn/MLRepository.html>
- Pratt, L., 1992. Discriminability-based transfer between neural networks. In: *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, pp. 204–211.
- Robert, C. P., Casella, G., 2005. *Monte Carlo Statistical Methods*. Springer, Berlin.
- Schmidhuber, J., November 1994. On learning how to learn learning strategies. Tech. Rep. FKI-198-94, Fakultat Fur Informatik, Technische Universitat Munchen.
- Shoenfield, J., 1967. *Mathematical Logic*, 1st Edition. Addison-Wesley, Menlo Park.
- Singh, S. P., May 1992. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning* 8 (3-4), 323–339.
- Solomonoff, R. J., 1978. Complexity-based induction systems: comparisons and convergence theorems. *IEEE Transactions on Information Theory* 24(4), 422–432.
- Swarup, S., Ray, S. R., 2006. Cross domain knowledge transfer using structured

- representations. In: Proceedings of the 21<sup>st</sup> National Conference on Artificial Intelligence (AAAI).
- Taylor, M., Stone, P., 2007. Cross-domain transfer for reinforcement learning. In: Proceedings of the 24<sup>th</sup> International Conference on Machine Learning.
- Thrun, S., Mitchell, T., 1995. Lifelong robot learning. *Robotics and Autonomous Systems* 15, 25–46.
- Thrun, S., Pratt, L. Y. (Eds.), 1998. *Learning To Learn*. Kluwer Academic Publishers, Boston, MA.
- Vilalta, R., Drissi, Y., October 2002. A perspective view and survey of meta-learning. *Artificial Intelligence Review* 18 (2), 77–95.
- Zvonkin, A. K., Levin, L. A., 1970. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Math. Surveys* 25(6), 83–124.