Transfer Learning Using the Minimum Description Length Principle with a Decision Tree Application

MSc Thesis (*Afstudeerscriptie*)

written by

Höskuldur Hlynsson
(born May 27th, 1977 in Reykjavik, Iceland)

under the supervision of **Dr Maarten van Someren**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
April 4, 2007	Prof Dr Pieter Adriaans
	Prof Dr Peter van Emde Boas
	Dr Maarten van Someren

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Contents

“ The Tarot has an unprecise relationship with the dimension of time. Contrary to popular opinion the cards cannot be used to predict the future. If the future were a definite and unalterable fact then the cards could be used to predict it. But the future is not a predetermined and planned event; if it were it would require a planner ([...] If the future were predetermined, then every detail of every event would have to be considered in advance, in effect creating the future in all its minutiae before it actually happens). What will happen tomorrow varies, sometimes subtly, sometimes dramatically. It is constantly adapting and changing, responding to the events and decisions made today. Viewed in this way, the future is not a definite and unavoidable fact; what actually occurs is one of a number of possibilities. The concept may be represented visually in the shape of a cone[...]. ”

— The Ancient Egyptian Tarot, by Clive Barret.

“ Prediction is very difficult, especially if it is about the future. ”

— Attributed to Niels Bohr, Physicist.

“Predicting the future is as difficult as reconstructing the past. ”

— Result from Information Theory homework in Peter van Emde Boas' course, UvA.

Abstract

Transfer learning is about how learning from one domain or a collection of domains can be applied to another. It is learning from similarities and parallels, from experience. This paper is about a distribution free, data driven, extendable framework for transfer learning, based on the minimum description length principle. We define transfer learning in terms of a specific framework, where we have a collection of hypothesis from other application domains available, but not the data, a learning algorithm consistent over domains and a new, previously unseen learning task.

Chapter 1

Introduction

“ ’Begin at the beginning’, the King said, gravely, ’and go till you come to the end; then stop.”’
— Lewis Carol, Alice in Wonderland

In this chapter we introduce the topic and the purpose of this paper and define the basic concepts and notation.

1.1 Orientation and Outline

This paper is about an approach to a form of meta-learning, or *transfer learning*, as applied to supervised learning. Meta-learning studies how learning algorithms can increase their classification performance, as measured by e.g. error rate, through knowledge about previous applications. The difference between meta-learning and *base-learning* is in the way the algorithms obtain a bias on the learning procedure: base-learners have a fixed bias while the meta-learner tries to induce a proper bias for the specific learning task at hand, based on knowledge about previously analysed tasks. An example of a fixed bias is “Small decision trees are more probable than big decision trees”. An example of variable bias would be “decision trees from this task-determined collection of trees are more probable then others”. Base-learners seek to improve the quality of their learning as the number of training examples in a task increases; meta-learners seek to improve the quality of the learning-strategy as the number of tasks increases also. We can view meta-learning as ’learning from experience’ where experience is understood as knowledge obtained from analysis of several tasks. A survey of meta-learning is given in [VD02].

The specific instance of meta-learning we will address in this paper is how we *transfer* learning done over some collection of related tasks to a new task where information about the datasets from the *old* tasks is withheld and only the learned structures are available. To motivate our approach, we allude shortly to our framework. First, we’ll give a couple of examples of what sort of problems we have in mind.

Example *Medical Diagnosis* Hospitals have a diagnosis procedure for a condition affecting an organ. Each hospital only has information about the measurements it makes on its own patients. Other hospitals have made the same sort of tests but because of privacy legislation cannot share their data. There are no limitations on sharing knowledge not traceable to individuals, which they do by making summarized information available in the form of learned structures. Now, a clinic starts testing for this condition. How can it make its procedures more effective by using experience gained on its own patients and the knowledge *about* others experience?

Example *Language acquisition* A person knows Portuguese and Italian and is learning Spanish. How can knowledge of the other two languages be of use? That person can both benefit from associating grammatical rules (syntactics) and word structures (semantics) from the languages already studied to the new one, without referring to specific examples (words/sentences).

We formulate our approach to transfer learning as a communication problem and use the Minimum Description Length (MDL) principle as the induction principle. When we speak of *knowledge* we mean a hypothesis which will almost always be uncertain, i.e. only true to a degree in relation to a dataset.

Consider the following scenario: I have some knowledge constructed from a particular collection of related application domains[1], which we term an *environment*, and you have a dataset, perhaps a small one. You decide that your learning task is in some way similar to the domains that I have knowledge of. You therefore acquire the knowledge that I have but not the data. Based on your data you decide what knowledge is applicable to your learning task, if any. I can not show you my data, from which I learned. Whether we share which learning algorithms were used is of no significance.

Furthermore, my knowledge must in some way simplify, or accelerate, your learning task in the sense that the probability of error decreases. Otherwise it is of no practical use to you. We like things to be as simple as possible, so you will not adopt overly complex knowledge to describe your data. There are two reasons to prefer simple explanations, all other things being equal: they are easier to remember and have less risk of explaining noise.[2] However, you might like the knowledge you adopt to be slightly more complex then what you would induce from your dataset alone. Otherwise you would just describe the knowledge you can draw from your dataset in form of a hypothesis without my assistance.

There must be some sort of a balance between the knowledge you adopt from my collection and how well this knowledge describes your data. When you adopt knowledge from my environment based on your limited data, you have *transferred* knowledge to your learning task. In order to give the knowledge which can be derived from your own dataset alone some part in our inference procedure, you add that knowledge to the environment[3]. After all, you are

[1]In what follows, the term *domain* refers to something abstract and conceptual which is assigned meaning by people. Domains do not have instantiations; example spaces, which are defined later, do have instantiations in the forms of datasets or samples.

[2]Some disciplines are more prone to explaining noise then others: the financial press is perhaps the most eager to offer explanations for everything. In general, every change (or lack there of) has an explanation. Elaborate explanations of minute price movements (less then 1/1000 of a percent) abound.

[3]This may appear strange, but there are strong theoretical reasons for doing this, as will become clear later. The exact way of doing this will be addressed in detail.

hypothesizing that your dataset belongs to the environment. A formalization and an exploration of a framework for inducing transfer along these lines is the subject of this paper.

Why is transfer learning useful? Transfer learning is of use since we almost never have enough data. Enough data would be so much data, that more data would not give us any information that would be of practical use. This is a common problem of inference. For instance, in [BC91] they talk about $n = 1000000$ to get a estimate of variance correct to 10 *bits*[4] from a normal distribution. By including information about the environment from which the data originates, we would hope to make up for the lack of data and obtain better hypothesis, as measured by error rate, while guarding against inapplicable information. An instance of this would be to obtain the same results with little data as with much data.

The learning task may thus be phrased as:

> Given a learner, a collection of hypotheses derived from related data-generating domains and a dataset for a new task (hypothesized being) related to the given collection, find an optimal hypothesis describing the domain of the given dataset, such that it maximizes performance on the new task.

We will elaborate on the terms used in the statement of the task to give it more precise meaning as this paper develops. The important part is that the data we are given is supposed to be of a similar kind as the domains giving rise to the knowledge. We use parallels and similarities when solving tasks in our daily lives. Doors to cars open in a very similar way to those of houses, although car doors never open into the car, this is very common for doors to houses to do. The omission of the data giving rise to the knowledge may seem restrictive, but in a way emulates how humans operate. We've opened *a lot* of different doors in our time and learned that there are basically only a few things to keep in mind: first turn, twist or yank, then push or pull. When faced with a new door we will not recall all those doors we've opened, just the main principles. If the door is to a house and has a handle, try turning the handle, then push. When successful, we have learned how to open yet another door. We frequently make do with abstractions.

The development of the framework encapsulating this sort of inference process described above will be based on minimizing the communication cost of the knowledge used to describe the new data and the new data with respect to that knowledge. The rationale for this sort of inference will be given in Section 1.3 on MDL.

In order to give expressions about the scenario described above precise meaning we will need definitions from measure and learning theory which are introduced in the next section. Then, after discussing the MDL principle for prediction and learning, we can give a definition of *transfer* and give a meta-algorithm for transfer learning. We are interested in asymptotic properties of the meta-algorithm and we give a proof of convergence in the chapter thereafter. To show how this framework might be put to work, we give an instantiation of the meta-algorithm using decision trees and give examples of its use in the final chapter.

[4]This is a far larger n then traditional statistical education would have us believe.

1.2 Preliminaries and Notation

We are liberal in notation and will use xa to mean concatenation of the symbols x and a. A little more formally, $xa \in X \times A$, where typically X is some cartesian product of the set A; A is frequently referred to as an alphabet. For instance, if $x = 10011 \in B^5$ and $a = 0 \in B$, then $xa = 100110 \in B^6$. where $B = \{0, 1\}$.

We define a measure:

Definition 1.2.1 *Let Ω be a set and $\mathcal{F} \subset \mathcal{P}(\Omega)$, where $\mathcal{P}(\Omega)$ is the set of all subsets of Ω. A* measure *is a function $\mu : \mathcal{F} \to [0, \infty[$ such that for any enumerated sequence $\{E_i\}$ of disjoint sets from $\mathcal{F}$ we have* countable additivity*:*

$$\mu(\cup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} \mu(E_i)$$

A probability measure *is a measure taking values in $[0, 1]$, such that $\mu(\emptyset) = 0$ and $\mu(\Omega) = 1$.*

We have the usual definition of conditioning on measures:

$$\mu(y|x) = \frac{\mu(xy)}{\mu(x)}.$$

Two functions, or *random variables*, $x, y : X \to \mathbf{R}$, X an object space, are *independent* with respect to a measure μ iff $\mu(xy) = \mu(x)\mu(y)$.

We will refer to a *anti-semi measure* property of some estimators and predictors:

Definition 1.2.2 *A* semi measure *is a function $\mu : \Omega^* \to [0, 1]$, such that $\mu(\emptyset) \leq 1$ and $\mu(x) \geq \sum_{a \in \Omega} \mu(xa)$.*

Semi measures thus allow for 'probability leaks'.[5] The *anti-semi measure* property corresponds to the last inequality in the definition reversed. When dealing with binary classification problems the set Ω is $\{0, 1\}$.

When we talk about probability measures we always have a *σ-algebra* in mind:

Definition 1.2.3 *A* σ-algebra *$\mathcal{F}$ on Ω is a collection of subsets of Ω which satisfies $\emptyset \in \mathcal{F}$,*

Closure under countable unions: $\forall i \in I \subseteq \mathbf{N}\ (s_i \in \mathcal{F}) \longrightarrow \bigcup_{i \in I} s_i \in \mathcal{F}$ *and*

Closure under complementation: $s \in \mathcal{F} \longrightarrow \bar{s} \in \mathcal{F}$,

where $\bar{s}$ denotes the complement of the set s and $\mathbf{N}$ is the set of the natural numbers.

The space $(\Omega, \mathcal{F})$ is called measurable *and $(\Omega, \mathcal{F}, \mu)$ is called a* measure space*, where μ is a measure on $\mathcal{F}$.*

[5] Semi measures arise naturally in Solomonoff's universal induction, see [Sol64], and account for undecidable problems, which again are reflected in the uncomputability of Kolmogorov complexity of a given input because of the halting problem. In this paper we are mainly concerned with computable approximations so we will abstain from further computability theory. As it happens, roulette wheels in casinos have probability leaks too, where the 'missing' probability of a player winning is what makes running a casino good business.

We need to restrict our attention to some proper subset of $\mathcal{P}(\Omega)$ when talking about measures over subsets of the set Ω, since generally if we accept the axiom of choice and countable additivity there are non-measurable sets in $\mathcal{P}(\Omega)$. A σ-algebra is such a restriction which has the desirable properties for dealing with outcomes of trials. [6]

We don't need the full measure theoretic heavy machinery to obtain our results. We can make do with the following definition of an *expectation*:

Definition 1.2.4 *The* expectation *of a function $\varphi : \Omega \to \mathbf{R}$ with respect to a measure μ is, for all $\omega \in \Omega$,*

$$\mathbf{E}\ \varphi = \mathbf{E}_\mu\ \varphi(\omega) = \sum_{\omega \in \Omega} \varphi(\omega)\mu(\omega).$$

The subscript μ is left unspecified when there is no risk of confusion.

When we are learning from data, we need some input to learn from, and when we are learning under supervision we also have some class labels of the input data. Our data comes from an *example space*, which *reality* generates.

Definition 1.2.5 *Reality*[7] *generates the pairs $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots$ of* examples. *We call x_i an* object *and y_i a* (class) label. *The objects and labels are from measurable spaces X and Y named* object space *and* label space, *respectively*[8]. *We require $X \neq \emptyset$ and an σ-algebra on Y different from $\{\emptyset, Y\}$. An* example space[9] *is then $Z =_{\text{def}} X \times Y$. A* sample *or* dataset *S^n is a particular set of observed values of Z^n.*

This definition implies that there are no missing observations in our samples. The requirement on the σ-algebra over Y is because, were it not satisfied, there is no learning to be done, since there is essentially just one label to choose from. An object consists of *features* or *attributes*. If $|X| = n$ then we have n features, referred to by $\xi_{i,1}, \xi_{i,2}, \ldots, \xi_{i,n}$ for the ith object.

A function $\mu : X \to Y$, which may have generated the data, is called a *concept* and is necessarily measurable[10] If such a function doesn't exist, there is no learning to be done. We will only be interested in stochastic concepts, for which either the concept is noisy itself or the observation process is not exact.

In what follows all functions will be assumed to be measurable, unless otherwise stated.

Definition 1.2.6 *A* learning algorithm, *or a* learner, *$\mathcal{L}$ is a function $\mathcal{L} : Z^n \to \mathcal{H}_\mathcal{L}$, where $\mathcal{H}_\mathcal{L}$ is a set of available concepts for the algorithm to learn. Each $h \in \mathcal{H}_\mathcal{L}$ is a function from X to Y, from the objects to the labels. We call $\mathcal{H}_\mathcal{L}$ a* hypothesis space. *A hypothesis space is* complete *if it contains the* true *concept that generated the data.*

[6] We note that a σ-algebra corresponds to a set of computable inputs to a fixed universal Turing machine.

[7] We won't attempt a definition of this thing and trust the readers intuition.

[8] In what follows, we use X instead of Ω above and view random variables and their domains as essentially the same thing.

[9] Sometimes referred to as *input-output* space.

[10] If it were not measurable, it wouldn't be computable and thus the concept not learnable. We leave the issue of whether trying to learn an uncomputable concept makes any sense untouched.

An instance of a learner is a classifier*, i.e. given a sample S^n from Z^n we have $\mathcal{L}(S^n) = h$, where $h \in \mathcal{H}_{\mathcal{L}}$, so a classifier is a function $\mathcal{L}(S^n) = h$ and is applied to objects to be classified by $\mathcal{L}(S^n)(x) = h(x) \in Y$.*

The subscript for the hypothesis space will be omitted when there is no ambiguity. If a space contains all functions from the input space to the output space, it is obviously complete.

When have a dataset from an example space and want to construct a classifier to classify previously unseen objects, we face a *learning problem*, or a *task*, in which we try to recover (learn) a *concept* which presumably generated the data. Learning can be done in either *realizable* or *non-realizable* setting, depending on whether the *true* hypothesis is available to the learner or not, respectively. The learner solves this learning problem and produces a classifier. The learner does this by searching through a space of functions available to the algorithm for a function that best fits the evidence of the dataset in some sense. This space is frequently unstructured and the search procedure is subject to converging to local optima. Thus we would like to influence the search for a good hypothesis by directing the search procedure in a certain way by inducing a *search bias* on the learner.

Definition 1.2.7 *A* search bias *on a learner is a partial ordering of $\mathcal{H}$, i.e. $\forall a, b, c \in \mathcal{H}$ the following hold: $a \preceq a$, $a \preceq b \wedge b \preceq a \rightarrow a = b$ and $a \preceq b \wedge b \preceq c \rightarrow a \preceq c$.*

A bias of a learner A is stronger *then that of B if $|\mathcal{H}_{\mathcal{L}_A}| \leq |\mathcal{H}_{\mathcal{L}_B}|$.*

The search bias is important, because we are often happy with methods that stop at local optima and which local optima we get stuck in during the search depends on the order. Later in this paper, we will use a description length function to impose the order on $\mathcal{H}$. In another light, we would like to impose an ordering on $\mathcal{H}$, s.t. we get a good $h \in \mathcal{H}$ early on in the search. An example of such a bias are those of decision tree construction algorithms which do not consider more complicated trees if they can make do with the simpler ones.

There are other types of bias concepts in use for learning theory: generally it refers to "any basis for choosing one generalization over another, other then strict consistency with the observed training instances" [Mit80]. The search bias is a *relative* bias, as it does not exclude hypotheses but orders them. Another type of bias is the *strict*, which excludes hypotheses from the space under consideration. These are said to be strong or weak, depending on how aggressively they exclude hypotheses, as indicated in the last definition.

Another characterization of bias is the *inductive bias*, which is the minimal set of assertions B such that for any concept μ to be learned and the sample S for the learning such that the classification of any object $x \in X$ follows deductively. That is

$$\forall x \in X (B \wedge S \wedge x) \vdash \mathcal{L}(S)(x)$$

This differs from the earlier concepts of bias in that it doesn't operate directly on the hypotheses space.

In general, each learning algorithm has its own implicit bias in the way it learns. Without a bias, an inductive learning algorithm cannot generalize beyond the training data. For more on this, see [Mit80].

There are various methodologies used to infer a function from some collection based on data. In the next section we describe the minimum description length principle (MDL), which is such a method. Another method related to the MDL is described in 4.1.2 on decision tree learning. The decision tree method also gives a nice way of representing the learned functions.

1.3 Minimum Description Length Inference

“ The fundamental idea behind the MDL Principle is that any regularity in a given set of data can be used to compress the data, i.e. to describe it using fewer symbols than needed to describe the data literally.”
— [Grü98]

“ We never want to make the false assumption that the observed data actually were generated by a distribution of some kind, say Gaussian, and then go on to analyze the consequences and make further deductions. Our deductions may be entertaining but quite irrelevant to the task at hand, namely, to learn useful properties from the data.”
— [Ris89]

When we want infer a mathematical description of some data generating process which captures its underlying structure, we are inferring a function. How we do that is based on the inference principle[11] we adopt. Inference is conceptually rather tricky. Consider the following example:

Example Given ’010101’ from $\{0, 1\}^*$, a set of arbitrary length strings of ’0’ and ’1’, which pattern should we infer when given a choice between

1. Repeat ’01’
2. Repeat ’010101’
3. Repeat ’010’ ?

Which explains the data best? Which is the simplest? What if we are also given the strings ’01101’ and ’01010101’ as examples?

The function we are inferring, the “explanatory theory”, we call a *hypothesis* and group together in *hypothesis spaces*[12], which are collections of hypotheses. When talking about *models* we usually have a specific restriction of the hypothesis space in mind, as may be expressed by a parametrization of a specific functional form such as the normal or beta-distribution.[13]We will use the term

[11]An inference principle is essentially the same as a bias.

[12]These are always required to be sets.

[13]Thus, *model selection* is the selection of a restriction of the space of all possible functions; these restrictions may be disjoint or have a non-empty intersection. An example would be to select the number of parameters to use for a model (i.e. degree of a polynomial, number of parameters in regression). Hypothesis selection would then be the selection of a specific set of values (points) from the model.

classification model in the same way as hypothesis though and refer to parameterized models when we are talking about such things. The term *knowledge* in this paper is also the same as hypothesis.

When inferring a hypothesis based on some data we need to state explicitly the principle we are using on each occasion. There are essentially two ways of dealing with this sort of scenario, each with its ancient advocate. The first is to 'not multiply explanations beyond necessity' and is referred to as Occam's razor and finds its modern embodiment in the MDL principle, which we adopt in this paper. The second is the Epicurean principle of multiple explanations: if more than one hypothesis is consistent with the observations, keep all of those. Epicurus' principle relates to Bayesian mixture probabilities, which will be defined later, and prediction with expert advice, which will not be discussed at all in this paper, but see e.g. [Vov98] for a discussion of this very interesting approach.

The most intuitively appealing way of inferring a function which is to describe some datasource from sampled data is the Bayesian approach. Although *ideal*, there are however considerable difficulties in specifying the prior probabilities necessary for applications. This problem can be circumvented to an extent by making the inference procedure as purely data-dependent as possible. The Minimum Description Length (MDL) style of inference take that approach and the strength of the inference procedures is almost as good as that of the Bayesian. Instead of probability we use description length, which is equivalent to probability as will be explained. For practical applications, MDL promises to be a good approximation. [14]

Why should we base an inference procedure on a description length concept? This is motivated in a sender-receiver setting, where the sender has a dataset and a set of hypotheses to describe the data with. The sender wants to communicate the data as efficiently as possible to the receiver, and can use hypothesis to describe his data more compactly. Thus, he uses the hypothesis to compress the data and transmits the data by coding a model and the data w.r.t. the model using a suitable encoding, or description method. We define the MDL principle as follows:

Definition 1.3.1 *Given a sample of data D and a countable collection of hypotheses $\mathcal{H}$ the MDL principle is to select the $H \in \mathcal{H}$ which minimizes the sum of*

- *the bit length of the description of the hypothesis H and*
- *the bit length of the description of the data based on the hypothesis H*[15].

The first term is abbreviated as $L_{C_1}(H)$ the second $L_{C_2}(D|H)$ so the principle is to select the hypothesis H_{MDL} such that

$$H_{MDL} = \underset{H \in \mathcal{H}}{\arg\min} \{L_{C_1}(H) + L_{C_2}(D|H)\},$$

where C_1 is the description method for the hypothesis and C_2 is the description method describing the data w.r.t. the hypothesis.

[14]There are close similarities between the Bayesian method and MDL, but MDL deals with codelengths while Bayes with probabilities and there exist some differences between the inferences made. MDL has a existence separate from the Bayesian methodology. For more, see [Grü04].

[15]This is called the *stochastic complexity*.

We will not distinguish between the two description methods involved in the definition and assume that the difference is understood.

An interesting thing is that every induction problem can be phrased as a binary sequence prediction problem and classification is a special case of sequence prediction. This is the power of the universal Turing machine.

The MDL principle tries to strike a balance between how complex the hypothesis is and how much it "explains" the data. The main motivation is to avoid overfitting the hypothesis to the data, i.e. modelling the noise. We want to factor out the regularities in the data by using our collection of hypotheses but leave the (*suitably* random) noise. This could be phrased as selecting an hypothesis that gives an optimal balance between how complex the hypothesis is and how well that hypothesis explains the data and thus an instance of Occam's razor.

The MDL principle might seem a bit arbitrary. A justification of the principle based on Bayesian learning theory follows.

When learning from data we have some collection of hypotheses $\mathcal{H}$ about the data to choose from. Before observing any data we might claim some distribution over $\mathcal{H}$, given by $P(H)$. After observing some data we hopefully gain some useful information about which hypotheses are more likely then others, obtaining $P(H|D)$. An appealing way to select a hypothesis from $\mathcal{H}$ is to select the most probable one after observing the data. This gives the *maximum a posteriori* (MAP) hypothesis:

$$\begin{aligned} H_{MAP} &= \arg\max_{H\in\mathcal{H}}\{P(H|D)\} \\ &= \arg\max_{H\in\mathcal{H}}\left\{\frac{P(D|H)P(H)}{P(D)}\right\} \\ &= \arg\max_{H\in\mathcal{H}}\{P(D|H)P(H)\} \\ &= \arg\min_{H\in\mathcal{H}}\{-\log_2 P(D|H) - \log_2 P(H)\} \end{aligned}$$

where we have dropped $P(D)$ since it doesn't depend on H. Observe that if each H has the same probability we get the maximum likelihood estimator

$$H_{ML} = \arg\max_{H\in\mathcal{H}} P(D|H).$$

Now, to go from H_{MAP} to the hypothesis selected by the MDL principle we need to observe that codelengths are equivalent to probabilities. Thus, in a sense, we obtain the Bayesian priors via codelengths. This relation is given by a fundamental inequality from information theory, namely the Kraft inequality. First, we define what is meant by a *description method*, and related concepts.

Definition 1.3.2 *Let* $\mathbf{A}$ *be an alphabet. A* coding system C *is a relation between* $\mathbf{A}$ *and* $\cup_{k\geq 1}\{0,1\}^k$*. If* $(a,b)\in C$ *then we say that* b *is a codeword for* a*. A* description method *is a coding system* C *such that for each* $b\in\cup_{k\geq 1}\{0,1\}^k$ *there is at most one* $a\in\mathbf{A}$ *with* $(a,b)\in C$ *The* description length *of* $a\in A$ *under a description method* C *is the cardinality of* b*, such that* $(a,b)\in C$*.*

We are only concerned with *prefix* description methods, i.e. description methods such that they don't have codewords as their prefix: if x and y are any codewords, then there doesn't exist a codeword z such that $zx = y$. In what follows, whenever we talk of description methods, we are talking about prefix description methods. For details about MLD, see [Grü04] , for the Kraft inequality giving the correspondence between prefix description lengths and probability distributions:

Theorem 1.3.3 *For any prefix description method C for a finite alphabet $A = \{1, \ldots, m\}$, the codeword lengths $L_C(1), \ldots, L_C(m)$ must satisfy*

$$\sum_{a \in A} 2^{-L_C(a)} \leq 1.$$

Conversely, given a set of codeword lengths satisfying the inequality, there exists a prefix description method with these lengths.

A proof may be found in [CT91], Section 5.2.

This tells us that there exists a (defective) probability distribution [16]

$$P(x) = 2^{-L_C(x)}, \forall x \in \Omega.$$

[17] Of course, we choose the correspondence in such a way that short codelengths correspond to high probability, and vice versa. With the correspondence between codelengths and probabilities we get the MDL principle as it is stated above.

When using MDL for prediction there are three ways to proceed, all based on the idea of using the best model from the class. We give descriptions of these methods when we need to use them in Section 3.2

Much can be said about MDL and its relation to Kolmogorov complexity (KC) but this is sufficient for our purposes. See e.g. [LV97] and [Grü04] for more on KC and MDL.

Although a very powerful inference principle with solid theoretical and philosophical foundations, the MDL principle does have some shortcomings. Criticisms and a list of problems is for instance given in [HOGV04].

To conclude this chapter, we pick up the examples from Section 1.1 and see what the MDL principle has to say about them intuitively. In the next chapter we'll give a definition of transferability based on MDL.

Example *Language acquisition* Consider a collection of grammatical rules from Portuguese and Italian and a dataset consisting of sentences from Spanish, along with their part-of-speech tagging. Inferring which rules apply sufficiently well to justify using them to assist in acquiring Spanish can be done using MDL. We evaluate the usefulness of the given grammatical rules by applying them to the collection of sentences and seeing whether the sentences are described in a shorter way by the help of the rules then without them. That is, if we can *compress* the given corpus by using the rules. Or, in more common language, if it is easier to remember the sentences by the help of a rules, that rule is useful. As it happens, our brains seem to be wired especially well to deal with grammatical rules.

[16] A *defective* probability distribution fails to satisfy $\sum_{x \in \Omega} P(x) = 1$, Ω an outcome space, but satisfies $\sum_{x \in \Omega} P(x) \leq 1$.

[17] The Shannon-Fano codes have these codelengths. Moreover, in Solomonoff's universal induction we have the *universal prior* by $2^{-\text{shortest program length}}$ and we have in MDL the priors by $2^{-\text{shortest codelength}}$.

Next along the lines of the last example, we address the issue of transfer in a MDL setting.

Chapter 2

Transfer learning

Transfer learning deals with using knowledge obtained from one learning task in another learning task. In general, transferring knowledge is difficult for us humans. Teachers report that children don't apply knowledge they obtained in one subject to another until they have developed considerable cognitive abilities, see e.g. [Has00] for a cognitive perspective.

In recent years researchers have started to focus on algorithms to transfer learning and the momentum seems to be increasing. Prominent institutes such as The Defense Advanced Research Projects Agency (DARPA), an agency of the United States Department of Defense responsible for the development of new technology for use by the military, have started to funnel funds to research in that area. To quote the DARPA project description BAA05-29:

> 1.1 PROGRAM OBJECTIVES
> The goal of the Transfer Learning Program solicited by this BAA is to develop, implement, demonstrate and evaluate theories, architectures, algorithms, methods, and techniques that enable computers to apply knowledge learned for a particular, original set of tasks to achieve superior performance on new, previously unseen tasks. This goal reflects the observation that key cognitive abilities of humans include the abilities to generalize, abstract, reuse, reorganize and apply knowledge learned in previous life experiences to novel situations.

This agrees with our view that successful development of transfer learning theories are a fundamental stepping stone towards a true *machine intelligence*. Machines capable of transfer learning would display cognitive abilities far beyond those seen so far. One might even venture so far as to say that when we have good transfer learning paradigms the development of machine learning will first take flight, perhaps on its own accord.

In this chapter we define our understanding of *transfer*, give a framework for transfer learning and a meta algorithm for inducing transfer. We also address a few issues related to transfer learning. Proof of a critical asymptotic property, namely the convergence to a true distribution as more data in the dataset under consideration becomes available and that distribution is in the hypothesis space, is given in the next chapter and the steps necessary to instantiate the meta algorithm described in this chapter in the chapter thereafter.

2.1 Transfer learning

When we acquire a dataset from some example space, we think of the data as arising from some *domain* that encapsulates some structure governing its behaviour. This term “domain” is necessarily somewhat vague and subject to interpretation. Two or more example spaces can be thought of as being related if their domains are related. This *relatedness* concept is quantified with respect to some task by the *transferable* notion defined in def. 2.1.2. [1]

For instance, two medical diagnosis tests, i.e. classifiers, of diseases affecting the same organ may be thought of as related domains, since the organ has some structure governing the behaviour of the diagnosis. If we have one or more histories of diagnosis test for some organ, and are devising a new test then we would like to use the information already obtained about the structure of diseases affecting that organ to speed up the learning of the structure for the new test.[2] In essence we would be transferring knowledge about the organ itself since the diseases exploit weaknesses in organs. Thus, we would like to group the other tests into an *environment* and transfer the learning from the environment to the new application domain. We define an environment as follows:

Definition 2.1.1 *An* environment $\mathcal{E}$ *is defined by* $\mathcal{E} = (\emptyset, Z_1, Z_2, \ldots, Z_n)$, $n \in \mathbf{N}, n > 0$, *where* $Z_i, i \in \{1, 2, \ldots, n\}$, *are example spaces.*

We include the empty set for technical reasons. Including it allows for a default behaviour of doing nothing.

[1]Another related point of view is provided by the hierarchical Bayes, which can be sketched as follows in its basic form. Given several sequences of random variables $x_1, x_2, \ldots, x_m$, each with corresponding length of $n_i, i = 1, \ldots, m, n_i \in \mathbf{N}$, which are unrestrictedly infinitely exchangeable have a joint density

$$p(x_1(n_1), \ldots, x_m(n_m)) = \int_{\Theta^m} \prod_{i=1}^{m} \prod_{j=1}^{n_i} p(x_{ij}|\theta_i)\, dQ(\theta_1, \ldots, \theta_m),$$

where $Q(\theta_1, \ldots, \theta_m)$ is the prior over Θ^m. In general, nothing can be said about Q. However, if the x_is are such that Q has interesting structure then various forms of hierarchical models arise. Thus, we quantify the *relatedness* of the x_is through the structure of Q. This leaves it up to the modeler to decide which forms of Q are interesting. This might be quantified by the Kullback-Leibler divergence between Q and a uniform distribution or a non-informative prior of the same dimension, for instance. One interpretation of the KL-divergence between two distributions P and Q is, that it is the expected number of extra bits needed to encode an event from P if the code used corresponds to the distribution Q.

A generic density version of such a model might take the form:

$$p(x_1, \ldots, x_k|\theta_1, \ldots, \theta_k) = \prod_{i=1}^{k} p(x_i|\theta_i) \tag{2.1}$$

$$p(\theta_1, \ldots, \theta_k|\phi) = \prod_{i=1}^{k} p(\theta_i|\phi) \tag{2.2}$$

$$p(\phi) \tag{2.3}$$

The difference between this model and ordinary learning is that the bias of the learner, eq. (2.2) is now conditioned on a *hyper parameter* ϕ and this has a *hyper bias* as represented by the distribution in eq. (2.3).

This seems like a fruitful way to look at transfer; for the interpretation and further discussion, see [BS94]. To make the connection with our framework, we would need to limit the attention to sufficient statistics arising from the given sequences. This will not be further pursued in this paper.

[2]This is related to comparing or combining clinical trials.

Note that there is no mention of the example spaces being in any way related. We would however hope that the underlying domains would be related in some way, although it is difficult to quantify this[3]. Transfer should occur when there is a computable relation between the domains. If there is no such relation, no transfer should occur. The understanding of an environment given above also permits a single non-empty domain to be used for transfer.

The primary goal of grouping example spaces into an environment is to draw useful knowledge from it for transfer to a new domain and its example space. This makes us wonder if we can require some of the features to be the same, in some sense. However, that sort of requirement would always be superficial at best. The features *temperature* and *pressure* occur in may domains[4]. We have no reason to suspect that measurements of temperature and pressure in a two different situations reflect knowledge from related domains. For instance, one set of measurements might come from a situation where we are deciding to play tennis. Another from a geological investigation of a geothermal drilling hole drilled for deciding if to harvest the energy. Simple syntax is of little use for grouping domains into environments; we need to consider *why* we are solving particular classification problem. There is art in the construction of useful environments.

We can however talk about knowledge contained in an environment without complications:

Definition 2.1.2 *Given an environment* $\mathcal{E} = (\emptyset, Z_1, Z_2, \ldots, Z_n)$*,* $n \in \mathbf{N}$*, the* environment knowledge[5] *is a collection of classifiers*

$$\mathcal{M}_{\mathcal{E}} = (h_1, h_2, \ldots\ h_m),$$

such that

$$\forall i\, \exists j\, (h_j = \mathcal{L}(Z_i)), j \in \{1, 2, \ldots, m\},$$

i.e. there is at least one instance of a learner for each classification source constituting the environment.

Say we have a new domain and an associated example space Z_{k+1}. We want to build a classifier for the new domain and have a dataset S^n_{k+1}, $n \in \mathbf{N}$. We suspect that Z_{k+1} belongs to an environment $\mathcal{E} = (\emptyset, Z_1, Z_2, \ldots, Z_k)$. In order to make use of the available information we need some measurement of "transferability". Also, as the dataset S^n_{k+1} grows[6], we would presumably like to adopt more detailed or complex knowledge from the environment since our confidence in the environment knowledge grows. Thus, we would like knowledge of different complexity to choose from. We can induce knowledge of different complexity from the environment knowledge by restricting attention to a subset of the features available from each domain or a subset of the values that each

[3]This issue refers to the *meta-meta-nature* of the problem, similar to the way that transfer learning refers to the meta-nature of the original learners.

[4]These have a well defined meaning in physics and obey the equation of state for an ideal gas: $PV = NkT$, where P is pressure, V is volume, N is the number of molecules, k is Boltzmann's constant and T is temperature. In a classification setting objects get another less quantifiable dimension: the *purpose* or goal of the task.

[5]The word "knowledge" is used to refer to uncertain, i.e. statistical, knowledge too.

[6]In the real world, we are frequently faced with a situation where more data becomes available as time passes.

feature can take. A formal definition of this sort of restrictions is technically not straightforward and will not be attempted here. Selecting the features for inclusion in a restriction of specific size is the feature subset selection problem, which has been studied extensively.

A sensible way to partition the knowledge for use from each domain is to take the most informative set of features, w.r.t. the dataset from that domain, from each domain in increasing order of conditional complexity of the resulting classifier, given the features already selected. So, we first select the single most informative feature, then the two most informative features, the second conditional on the first, and so forth. Thus we get the best quality of hypothesis for each complexity, as measured by the hypothesis codelength, and a linear ordering of these optimal hypotheses with respect to the total codelength. These we then evaluate with respect to data to choose a suitably sized hypothesis.

As an example, for decision trees constructed by an information theoretical measure such as *information gain*, , defined in 4.1.2, this procedure takes place quite naturally. There is no need for the learner to evaluate the informativeness of restrictions of different sizes w.r.t. a dataset, as that information is encoded into the tree when it is constructed.

An information measure quantifies the amount of regularity that a function distills from a dataset, loosely speaking. Examples of such an information measure is information gain and the MDL codelength function. Our primary tool for evaluating the usefulness of an environment is that of *transferability* which we define in terms of a modified minimum description length principle. We introduce weights over the environment knowledge $\mathcal{M}_\mathcal{E}$ to reflect that we have more information then just the data alone. This information may be subjective. The weights are functions from the environment knowledge to the positive reals. These are discussed further in Sections 2.3 and 2.4.

Definition 2.1.3 *Given environment knowledge $\mathcal{M}_\mathcal{E}$ and weights $w(h)$ we say that the knowledge contained in $h \in \mathcal{M}_\mathcal{E}$ is* transferable with respect to the dataset S^n_{k+1}, $n \in \mathbf{N}$, *if*

$$L^{wh}_C(S^n_{k+1}) =_{\text{def}} L_C(S^n_{k+1}|h) + w(h)L_C(h) \leq L_C(S^n_{k+1}).$$

The collection of all transferable classifiers we call the transferable set

Note that $L_C(\emptyset) = 0$ and we always have the empty set in $\mathcal{M}_\mathcal{E}$ so the transferable set contains $\emptyset$ at least. When selecting between two hypothesis of the same complexity, we would of course consider the one which is more informative. This is relevant for both construction of new hypotheses in $\mathcal{M}_\mathcal{E}$ and selection of hypothesis for transfer.

When coding the data with respect to the empty hypothesis for a binary classification problem, we will use one bit to designate the default class and then code the exceptions from that class. This corresponds to a very naïve Bayesian classifier which just uses the most frequent class as its prediction. The encoder the codes the exceptions from this class. We give more detail on this sort of scheme in Section 4.2.2.

Note also that $L^{wh}_C(S^t_{k+1})$ defines a linear order on any collection of hypotheses $\mathcal{H}$. Unless we have provably optimal codes for the objects under consideration speaking of this hypothesis being x-times more transferable then another doesn't make any sense in our setting; we are only concerned with the order, not size. In most applications we have approximate codelengths but not optimal ones.

Now we can answer the question: How do we select knowledge to be transferred?

$$\text{Transfer } h^*, \text{ s.t. } h^* = \underset{h \in \mathcal{M}_{\mathcal{E}}}{\arg\min}\, L_C^{wh}(S_{k+1}^n).$$

Before giving a statement of the proposed algorithm, we motivate it further in a communication framework.

2.2 A Communication Framework for Transfer Learning

In this section we will show how to extend the basic sender-receiver setting underlying the minimum description length principle to cover the present setting in line with our definition of transferability. We also illustrate the relation between the algorithms involved in our framework.

We consider it our goal to perform well on the new task, for which we have data, using the knowledge available from the environment as well as we can. This may involve assigning different weights to the available hypotheses, depending on how credible we think they are. For instance, hypothesis occurring in many domains might be considered as more probable then others. We are minimizing the communication cost of the new dataset to a receiver, which we use the prior information to help us to do. We are not too concerned with time or memory complexity; the only thing we require at this stage is computability.

Now, recall the discussion from Section 1.1. We are now ready to make things a bit more precise, by using definitions given so far. When thinking about transfer learning in terms of a communication framework we need to state who communicates what to whom. Our problem is then to utilize the communication *channels* as efficiently as possible. We do that by encoding our data as efficiently as possible. To begin with, we'll assume that all learners involved are using the same learning algorithm. We'll discuss how this assumption can be relaxed at the end of this section.

Picking up from where we left of in Section 1.1, the situation is as follows: I have a collection of hypotheses and you have a dataset and are looking for a good hypothesis about your data. In line with the MDL principle, the combined length of the description of the hypothesis you adopt and the description of the data, w.r.t. your new knowledge should be minimized. The rationale behind this is that this approach allows you to express the data itself as compactly as possible, by using a hypothesis and the exceptions from that hypothesis. If this minimization yields a hypothesis that is simple, and has no exceptions, you are in luck and get a very concise way of expressing your data. If, however, there is no good hypothesis to choose from in the collection of hypotheses I propose with which to express your data, you end up using a hypothesis based on your data alone, or, if everything else fails, no hypothesis at all. In any case, if you can find a more compact way of expressing the data w.r.t. to some hypothesis then without it, resources are saved. In all of this, the hope is that by adopting knowledge from my environment you will be able to have superior learning performance.

If you like a hypothesis from the collection of hypotheses that I give you, you may feel like combining more then one hypothesis into a more elaborate one.

The foregoing discussion raises a few questions for you:

- What are those theories or hypotheses?
- How do I decide that a hypothesis describes my data well?
- How do I describe a hypothesis?
- How do I describe my data w.r.t. a hypothesis?
- How can I gain confidence using that the knowledge I am adopting will actually help me to learn the structure of the new learning task?
- Will I be adopting different theories from your environment, coming from different domains, as my dataset grows?
- How can I gain more certainty in that any of your theories apply to my domain?

The first two of these questions are the main concerns of this paper. The theories to be considered are those that come from a learner which has been applied to an environment. The theories that describe the data well are the transferable hypotheses. The answers to the remaining questions are given by our MDL inference principle.

For ease of discussion, lets say that I, call me A, have the knowledge from an environment expressed as hypotheses from a learner. You, called B, have a small dataset from a new domain, and you are communicating your data to a third party, called C. The party C only plays a conceptual role in our scenario. You and C share the feature data (the input objects) of the dataset, but C doesn't know the classes that each object belongs to. Thus, you only have to communicate the class-labels to C. If the structure of B's learning problem is less rich then A has to communicate, B just ignores what is irrelevant. This scenario is illustrated in Figure 2.1

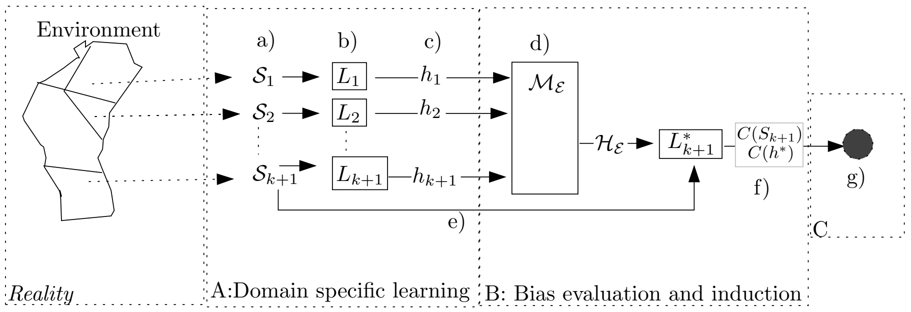

Figure 2.1: The communication scenario for transfer learning The agents A, B and C communicate the knowledge and data as shown

A few notes on Figure 2.1:

a) Data Our data comes in the form of a sample from some datasource, which presumably has *some* structure, S_{k+1} is included, as per the assumption.

b) Learner The learners do their thing with the data, including $\mathcal{L}_{k+1}$. In general, we need only restrict the learners such that they have the same representation of the learned hypothesis.

c) Each learner sends the learned hypothesis to $\mathcal{M}_{\mathcal{E}}$.

d) $\mathcal{M}_{\mathcal{E}}$: B is allowed to add hypotheses to $\mathcal{M}_{\mathcal{E}}$, by deriving new hypothesis from the given ones. In an adjusted measure of the applicability of the hypothesis, each hypothesis is given weight according to their origin and frequency, i.e. whether they come from the environment or induced from the S_{k+1} dataset. $\mathcal{H}_{\mathcal{E}}$ is thus the set of hypothesis proposed by the *environment*, considering S_{k+1} as a part of it.
$\mathcal{H}_{\mathcal{E}}$ represents the knowledge available to the agent B as offered by the environment knowledge, the dataset under consideration and a cognitive process in $\mathcal{M}_{\mathcal{E}}$.

e) S_{k+1}: the new data is used to evaluate the hypotheses in $\mathcal{M}_{\mathcal{E}}$ and select a single most relevant (partial) hypothesis.

f) L^*_{k+1}: agent B selects an optimal hypothesis from $\mathcal{M}_{\mathcal{E}}$ evaluated by S_{k+1} and uses it as the hypothesis about the workings of the system under observation. The agent B communicates the chosen MDL hypothesis, i.e. the transferred knowledge, and the data with respect to that hypothesis to the agent C. This information is sent as a code, from a given encoding scheme. This where the *minimum* in MDL comes into play.

g) The minimum description length representation of the class-data w.r.t. $h^* \in \mathcal{M}_{\mathcal{E}}$ is passed on to C, who only serves a conceptual purpose as a receiver. h^* is the transferred knowledge.

It might be asked what the position of the algorithm for inducing the transfer is, in relation to item d) above and traditional algorithms. In Figure 2.2 the flow of a implementation of this framework is shown.

As shown there, we can consider the base-learning algorithm (L), the algorithm cooking up the hypotheses (M) and the transfer learning (TL) algorithm separately. In a sense this is a wrapper framework, as it is not dependent on the specific implementations. This we state as Algorithm 3, Transfer Inducer (TI), which encapsulates what is shown on Fig. 2.2. We'll refer again to this figure when discussing the experiments.

2.3 A Meta-Algorithm

With the communication scenario from the previous section in mind, we now state a meta version of the algorithm, as given in Alg. 1, 2 and 3.

In Algorithm 1 we like to keep the h_{k+1} separate for convergence reasons discussed later.

In Algorithm 2 we need prior weights w. These prior weights reflect the degree of belief of how probable each hypothesis is in the current task. How do we motivate the assignment of the weights $w(H)$? The weights may be

Algorithm 1: (M) A hypothesis grinder/combiner; partitions the knowledge contained in a hypothesis into simpler hypothesis allowing for combinations and assigns prior weights to each hypothesis. This function should resemble a cognitive process

```
input  : A set of hypotheses $\mathcal{H} = \{h_1, \dots, h_k\}$, and a special
         hypothesis $h_{k+1}$
output: $\mathcal{M}_{\mathcal{E}} = \{(h_i, w_i)\}_{i \in I}$

% find more general hypothesis
$\mathcal{M}_1^* \leftarrow$ chop($\mathcal{H}$)
$\mathcal{M}_2^* \leftarrow$ chop($h_{k+1}$)

% each $h_i \in \mathcal{M}_{\mathcal{E}}$ gets weight $w_i = n_1 \cdot w_A + n_2 \cdot w_B$, $n_i$ the number of
occurrences of $h$ in $\mathcal{M}_i^*$, i.e. how many instances of $h$ there are in $\mathcal{M}_i^*$
$\mathcal{M}_{\mathcal{E}} \leftarrow$ combine(($\mathcal{M}_1^*, w_A$), ($\mathcal{M}_2^*, w_B$))

% Notes:
chop: this function “chops” the knowledge contained in a hypothesis
into components of less complexity, which constitute the output,
making it more general in the process since it covers less cases then
the original hypothesis. This function should not go beyond it’s input;
i.e. the union of all its outputs should be a subset of the input. The
term “chop” comes from how this function might behave on decision
trees, where we would chop the tree into subtrees.
```

Algorithm 2: (TL) Scheme for inducing transfer from a hypothesis-collection and a dataset

```
input  : A set of hypothesis $\mathcal{M}_{\mathcal{E}} = \{h_1, \dots, h_n\}$ with associated
         weights $w_i$, learning algorithm $\mathcal{L}$ with the same hypothesis
         representation as $\mathcal{M}_{\mathcal{E}}$ , dataset $S$
output: $h^*$: MDL hypothesis over $\mathcal{M}_{\mathcal{E}}$ w.r.t. $S$

%codelength of each hypothesis
for $i \leftarrow 1$ to $n$ do
    lh[i]$\leftarrow L_C(h_i)$
end
lh[n+1] $\leftarrow L_C(h_i)$

%codelength of data, given each hypothesis
for $i \leftarrow 1$ to $n$ do
    lDh[i]$\leftarrow L_C(S|H)$
end

%(weighted) total codelength
for $i \leftarrow 1$ to $n$ do
    tLC = lDh[i] + w[i]*ld[i]
end
%find the minimum description length
indexOfMinLC $\leftarrow$ indexOf( min[tLC] )

return $\mathcal{M}_{\mathcal{E}}$( indexOfMinLC )
```

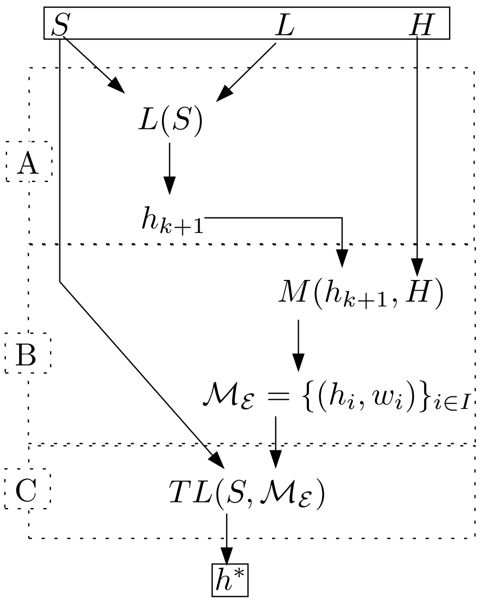
S
L
H
L(S)
A
h_{k+1}
$M(h_{k+1}, H)$
B
$\mathcal{M}_{\mathcal{E}} = \{(h_i, w_i)\}_{i \in I}$
C
$TL(S, \mathcal{M}_{\mathcal{E}})$
h^*

Figure 2.2: Relation between types of algorithms that appear in this paper, which can be considered as a Transfer Inducer (TI, Alg. 3). H is a collection of hypotheses h, L is a learning algorithm, S is a dataset, M is a hypothesis induction algorithm, that might for instance make partial hypotheses from those contained in H, and TL evaluates hypothesis for transfer producing a single most probable hypothesis h^*. The A part of the figure corresponds to ordinary learning, B to hypothesis space construction and C to the transfer learning/evaluation.

thought of as mappings from the prior $P(H)$ to a new prior $P(H)^{1/(f+1)}$, where f is the frequency of the hypothesis in M, assigning $f = 0$ to the hypotheses constructed from S_{k+1}. Thus, $w(L_{k+1}(S_{k+1})) = 1$, and another hypothesis h occurring once would get $w(h) = 1/2$, and so forth, by the correspondence between codelengths and probabilities. The essence of transfer to the new task is captured by these weights. Contrary to the approach in Statistical Learning Theory, where the model is penalized by some *regularization factor* to prevent overfitting, we discount the models that the environment indicates are more probable.

In [Zha04] and [BC91] a modified information complexity minimization crite-

Algorithm 3: (TI) A transfer inducer; wrapper for the framework in fig. 2.2.

```
input  : S, L, M, TL as defined above
output: $h^*$, an optimal hypothesis

% Join the elements of the algorithm:
return $h^* = TL(S, M(L(S), H))$
```

rion[7] is studied where the model complexity $L(h)$ is assigned some *regularization parameter* λ. The contribution of the model complexity to the total then becomes $\lambda L(H)$. This λ is used in a similar way as our w and has been shown to make the estimation method more robust for $\lambda > 1$. Unfortunately, neither paper offer any intuitive interpretation of this parameter and it only serves a technical purpose to improve the bounds derivable. In this case, the ends justify the means. It is not clear from either paper why, besides the technical justification, we should allow for this parameter and sparse advise is offered on the value to assign to λ.

The relevance of these results for our work is that it might be advisable to use $w(H) > 1$ for all hypothesis. However, that undermines the purpose of transfer, namely to adopt more complex hypotheses then the data alone give rise to, as indicated by the environment.

Zhang's results imply for us, that we put not too small of a prior weight on a hypothesis that is close to the "truth" (even if such a hypothesis is not in the space under consideration), so we should take care not to assign too low weight to the empirical distribution, the distribution constructed form the new task, since in general we expect the underlying learner to be convergent.

Since we have not included any information on the size of the dataset used for constructing the hypotheses we would like substructures from each learner (domain) to be available, since we may only have very limited data in S_{k+1}. What sort of structures might these be?

One particularly appealing thing to do in Alg. 1 with the elements of H, the set of given hypotheses, is to construct less specific hypotheses from, i.e. hypotheses based on fewer features and with fewer partitions of the input-space. These hypotheses would represent a subset of the knowledge contained in each hypothesis, allowing the algorithm to refrain from over generalizing based on the limited data in S_{k+1} and adopt a subset of the knowledge contained in each hypothesis. Another appealing thing to do with the hypotheses is to join disjoint hypotheses, allowing for overlap between two or more domains; specifically this might be useful based on restricted hypothesis. We illustrate how this might proceed in Section 4.4.

To use this algorithm we need to instantiate all the steps in the procedure, which we give an example of in Chapter 4 using decision trees. The issue of coding the relevant structure for the inference is usually quite tricky, but for a decision tree implementation it turns out to be rather straight forward, see 4.2.

2.4 Discussion

Here we discuss the minimization of the code length for the environment as a whole. In all cases, we have a new, previously unseen task to solve, given a new dataset S_{k+1}, and the environment knowledge $\mathcal{M}_{\mathcal{E}}$. The question addressed in this section is what to optimize.

If we have available the datasets $S_i, i \in \{1, 2, \dots, k\}, k \in \mathbf{N}$, from the domains used to construct the hypotheses in $\mathcal{M}_{\mathcal{E}}$, a sensible thing to do would be to consider the new task as a part of the environment and solve:

$$\arg\min_{H}\{L(H) + L(H| \cup_{i=1}^{k+1} S_i)\} = H^*.$$

[7]The MDL principle is such a criterion.

The hypothesis H^* minimizes the total codelength for the whole environment.

If the datasets are not available, as we assume in this paper, we can consider two distinct minimization problems to solve when finding an optimal hypothesis for the new task. The first is the one considered in this paper:

$$\arg\min_{H \in \mathcal{M}_{\mathcal{E}}} \{w(H)L(H) + L(S_{k+1}|H)\},$$

where the weights $w(H)$ are given positive reals and adjusted to allow for more complex models then the data alone would.

The other might be formalized as follows. Still, $\mathcal{M}_{\mathcal{E}}$ is fixed, but there is no need for specifying the weights $w(H)$. From the environment knowledge, we find the optimal *hyper-hypothesis* or *-prior*:

$$\arg\min_{h \in \Theta}\{L(h) + \sum_j L(H_j|h) + L(S_j|H_j)\} = \arg\min_{h \in \Theta}\{L(h) + \sum_j L(H_j|h)\} = h_a^*,$$

since the terms $L(S_j|H_j)$ does not depend on h; Θ is the space for the hyper-prior. Then, when we get new data S_{k+1}, which we consider to be a part of the environment abstractly described by h_a^*, we solve

$$\arg\min_{H}\{L(h_a^*) + L(H|h_a^*) + L(S_{k+1}|H)\} = \arg\min_{H}\{L(H|h_a^*) + L(S_{k+1}|H)\},$$

to get the optimal description of the new data, when assuming that it originates from the same environment.[8] To apply this method, we need to specify the hyper-prior, or a code length function for the hypotheses from Θ and $L(H|h_a^*)$, which does not appear to be obvious. This approach could be termed three part MDL, and has close relations with the hierarchical Bayes method.

The approach taken in this paper appears to be more robust to misspecification of the environment then when specifying a hyper-prior. Misspecification of the hyper-prior would affect the codelength of the hypothesis given the hyper-prior in a adverse way. The alternative optimization outlined in this section does not concentrate on the new task, but on the environment as a whole when viewing the new task as a part of it. It might be said that the method with the weights concentrates on performing well on the new, *current* task, while the other approach, outlined in this section, concentrates on finding good abstractions based on everything in the environment. One finds good predictions for what happens next, the other constructs good models of the environment. If there is even one domain that doesn't group well with the other tasks[9] then the abstraction might be "confused", while the method we devote this paper to does not get so easily confused. It would seem that our method, with the weights, concentrates on performing well on the new task, while the other on any unseen task. The latter case is addressed in [Bax00]. Are these problems distinct? Is the three part description length minimization better then weighing the prior knowledge? A proof that our approach is better or not then the three part MDL approach would be nice, but that is the subject of another paper.

In short, which optimization problem we solve, and what we assume, depends on the purpose of our investigations: are we learning about the environment or learning to solve the new task? In this paper, we concentrate on performing well on the new task.

[8] This is slightly reminiscent of the expectation-maximization (EM) method [DLR77].

[9] In the hierarchical Bayes sense.

Which learning algorithms? So far we have assumed that all learners are using the same learning algorithm. Now we ask: what restrictions should be imposed on the learners producing the hypotheses in $\mathcal{M}_{\mathcal{E}}$? Basically the only restriction is that we have a coding scheme to produce the codelengths of the hypothesis so we can evaluate if the hypothesis is transferable or not. We are not concerned with how the hypothesis are made, only whether they are useful or not for the task at hand.

Taking a closer look at the framework, another thing reveals itself. The usefulness of the hypotheses in $\mathcal{M}_{\mathcal{E}}$, as measured by the transferability, can be influenced by the construction of the hypothesis. But, if the hypotheses are bad that will be reflected in poor transferability.

Further, we can repeatedly apply the TI algorithm to unexplained data, until the transferable set is exhausted.

No Free Lunch and Transfer learning Researchers are sometimes concerned with *No Free Lunch* (NFL) style results, see [WM95]. These results rely on the premise that all inputs are equally likely a priory, observations are i.i.d. and the problem spaces are finite, which is rarely the case. As concerns our algorithm, the bias of the underlying algorithm is enriched with information from an environment, which further directs the search for an optimal hypothesis away from any sort of uniform prior.

To conclude this chapter, we recap that the whole of the bias is implemented in $\mathcal{M}_{\mathcal{E}}$, along with the weights, where the environment is given relative preference over the structures rising from the current dataset, and only hypotheses in $\mathcal{M}_{\mathcal{E}}$ are considered for the new task. Note that this is not really restrictive, since we eventually have almost sure converge to the hypothesis constructed by the new dataset when we have enough data in the sample and this turns out to be the true hypothesis in the realizable case. We now turn our attention to this issue.

Chapter 3

Convergence

"The truth is not out there."
— Not Agent Mulder from the television show The X-files.

In this chapter we address the asymptotic properties of our method. Why do we need asymptotic properties? Without those we can't state that our method is actually of any use. If more data isn't of use for our method, then it wouldn't really be adjusting to the properties of the data, or to the structure indicated by the data. We are looking for algorithms that learn from data, so this is essential. Asymptotic properties, however, do not necessarily tell us anything about small sample properties: in practice these are the properties of real importance.

To provide formal grounds for the utility of our method we need theorem 3.2.4 which gives bounds for the divergence of a probability measure from a *true* measure. Loosely speaking, this theorem tells us that if the true measure, or classifier, is in the collection of hypotheses $\mathcal{M}$ then our method will recover it eventually. This result hinges on the learning problem being realizable, using MDL as the basis of inducing transfer and the underlying method being convergent. These asymptotic results give us a guarantee of the behaviour of the proposed framework but do not state anything about restricted size sample results It may very well be that a suspicious looking hypothesis will be selected for transfer based on a sample, but eventually (*in mean sum*, as defined later) when there is enough data, a wrong bias on the hypothesis space will be overcome. In the end, there is only the data to base inference on.

The convergence results are due to Hutter and Poland [HP05] but adapted to cover only our present setup, with some simplifications and more detail then in the original paper. Their work draws on Solomonoff's seminal paper [Sol78], containing results from 1968(!), where he proves for the Bayes mixture distribution ξ over any collection of measures C containing the *true* distribution μ, $\xi(x) = \sum_{\nu \in C} w_\nu \nu(x)$, that

$$\sum_{t=1}^{\infty} \mathbf{E} \sum_{a \in X} (\mu(a|z) - \xi(a|z)) \leq \ln w_\mu^{-1}.$$

where z is sequence from X^*[1], and an earlier paper by Hutter [Hut01] with

[1]An interesting account by Solomonoff of discoveries can be found at `http://world.std.`

results from 1999:

$$\sum_{t=1}^{\infty} \mathbf{E} \sum_{a \in X} (\mu(a|z) - \rho_{\text{norm}}(a|z)) \leq w_\mu^{-1} + \ln w_\mu^{-1},$$

where

$$\varphi_{\text{norm}}(a|z) =_{\text{def}} \frac{\varphi(a|z)}{\sum_b \varphi(b|z)} = \frac{\varphi(za)}{\sum_b \varphi(b|z)}$$

for any function $\varphi : Z^n \to [0,1]$, $n \in \mathbf{N}$, where za is the sequence z with a added to the end. The result we need is the convergence between the true measure and a MDL predictor, which will be defined shortly.

This chapter is on the technical side but is nevertheless important since without asymptotic statements about our algorithm we don't know if they are of any use.

3.1 Convergence Concepts

The proofs of convergence given in Section 3.2 are based on the concept of *convergence in mean sum.* This we relate to the more familiar *convergence in probability*, which is the probability theory equivalence of the measure theoretic *almost everywhere*.

The following convergence concepts are of interest:

Definition 3.1.1 *A sequence of random variables $x_1(\omega), x_2(\omega), \ldots$ from Ω converges to $x(\omega)$, $x_i : \Omega \to \mathbf{R}$*

in probability *if* $\lim_{t\to\infty} P(\{\omega \in \Omega : |x_t(\omega) - x(\omega)| > \varepsilon\}) = 0$

with probability one *if* $P(\{\omega \in \Omega : \lim_{t\to\infty} x_t(\omega) = x(\omega)\}) = 1$

in mean sum (i.m.s.) $\sum_{t=1}^{\infty} \mathbf{E}[(x_t(\omega) - x(\omega))^2] < \infty$

where P is a probability measure. Convergence with probability one is also referred to as almost sure *and* almost everywhere convergence.

Almost sure convergence is frequently used in probability theory and is the concept used in the strong law of large numbers, which states that a sample average tends to the mean of the distribution generating the observations almost surely, loosely speaking.

These concepts are related by the following lemma, which shows that the *i.m.s.* concept is the strongest of the three:

Lemma 3.1.2

1. *Almost sure convergence implies convergence in probability.*
2. *Convergence in mean sum implies almost sure convergence .*

Proof See [Ser80].

`com/~rjs/dart.pdf`

Thus, our proofs of convergence *i.m.s.* imply convergence with probability one. For a further discussion see [Ser80]. There are various other convergence concepts in use, but these are sufficient for our purposes.

We are mainly interested in when a sequence of distributions converges to a specific distribution: either the *true* distribution or some distribution which is *sufficiently* close to such a distribution. For a discussion on what *sufficiently* means in this context, see [RH06].

3.2 Proof of Convergence

We now show that under some conditions the MDL estimation method recovers the *true* classifier μ, if there is such a thing. We follow the nomenclature of Hutter and Poland but use slightly different notation[2].

Consistent with the notation in previous Sections, we use $y \in Y$ to denote class labels and $x \in X$ to denote the objects for classification. We always deal with object-label pairs $z = (x, a) \in (X, Y)$. We use x_a or z_a to denote an object corresponding to the label a. We use $z^n_{\underline{a}} \in X^n \times Y^{n-1}$ to denote a history of n object-label pairs, but the label for a is missing; the missing observation is the one corresponding to the t-*th* element of x^t. We use $z^n_{\underline{y^k}} \in X^n \times Y^{n-k}$ to denote a history of n object-label pairs, but the labels for a subset of k items missing. When we use $\varphi(x^t|y^t)$ the objects and labels are paired. As in the MDL literature, the term xy, where x and y are from an alphabet X, refers to the concatenated string.

This is slightly different from the normal situation when $z^t \in Z^t = (X \times Y)^t$, which we'll use at times, but there should be no risk of confusion. The foregoing is the normal situation when using a classifier: first we build the classifier using some training set with all the labels supplied, then we use the classifier on a set with the labels missing. The application of the classifier to the situation when the labels are missing is where the classifier actually becomes useful. We assume that all examples are independently identically distributed.

We define an MDL estimator as

Definition 3.2.1 *Let $\mathcal{C}$ be a countable class of measures and w_ν be (prior) weights assigned to each $\nu \in \mathcal{C}$, s.t. $\sum_{\nu \in \mathcal{C}} w_\nu \leq 1$, $w_\nu > 0, \forall \nu \in \mathcal{C}$. The* MDL estimator *of a class label y, given its object is*

$$\rho(y) = \max_{\nu \in \mathcal{C}} \{w_\nu \nu(y|x_y)\}$$

The correspondence of ρ to the two part description length has already been described in the Section 1.3 (hint: think logarithms).

We need to see that the maximum in the definition is attainable. This we show next

Theorem 3.2.2 *The MDL estimator exists.*

Proof See [HP05].

To prove the results we need MDL predictors. These are defined as:

[2]The point of this is to hopefully make the classification references more clear.

Definition 3.2.3 *The* static MDL predictor *is*

$$\rho^{\text{static}}(a|z_a^t) = \rho^{y^{t-1}}(a|z_a^t) = \frac{\rho^{y^{t-1}}(y^{t-1}a|x_a)}{\rho^{y^{t-1}}(y^{t-1}|x_a)}$$

The dynamic MDL predictor *is*

$$\rho^{\text{dyn}}(a|z_a^t) = \rho^{y^{t-1}a}(a|z_a^t) = \frac{\rho^{y^{t-1}a}(y^{t-1}a|x_a)}{\rho^{y^{t-1}}(y^{t-1}|x_a^t)}$$

To ensure that the MDL predictors are indeed measures we normalize them by

$$\varphi_{\text{norm}}(a|z_a) =_{\text{def}} \frac{\varphi(a|z_a)}{\sum_b \varphi(b|z_b)} = \frac{\varphi(z_a a)}{\sum_b \varphi(b|z_b)}$$

for any function $\varphi : Z^n \to [0,1]$, $n \in \mathbf{N}$.

The difference between ρ^{static} and ρ^{dyn} is illustrated by

$$\rho^{\text{dyn}} = \frac{\max_{\nu\in\mathcal{C}}(w_\nu \nu(y^{t-1}a|x_a^t)}{\max_{\nu\in\mathcal{C}}(w_\nu \nu(y^{t-1}|x_a^t)} \quad \textit{versus} \quad \frac{\max_{\nu\in\mathcal{C}}(w_\nu \nu(y^{t-1}|x_a^t)}{\max_{\nu\in\mathcal{C}}(w_\nu \nu(y^{t-1}|x_a^t)} = \rho^{\text{static}}$$

so the dynamic predictor is optimized for all possible outcomes $a \in Y$, while the static one uses the estimate based only on the history y^{t-1}.

The following theorem establishes bounds for the measures induced by our classifiers. The importance of this theorem is, as theorem 3.2.11 states, that it provides both necessary and sufficient conditions for the true distribution to be recovered by the MDL-method in a rather strong sense. We'll apply this to decision trees in chapter 4.

Theorem 3.2.4 *Let $\mathcal{H}$ be a countable collection of classification models and $\mathcal{C}$ the set of probability measures induced by $\mathcal{H}$ containing the true measure μ. Then for any given set of inputs $z^t \in Z^t$*

$$\sum_{t=1}^{\infty} \mathbf{E} \sum_{a\in\{0,1\}} (\mu(a|z_a^t) - \rho(a|z_a^t))^2 \le 21 w_\mu^{-1}$$

where ρ is the MDL dynamic predictor.

Before proving this theorem we first establish a few results. The *Bayes mixture distribution* of a countable collection of distributions $\mathcal{C}$ over an example space Ω, such that each distribution in $\mathcal{C}$ is associated with a weight w_ν is

$$\xi(x) = \sum_{\nu\in\mathcal{C}} w_\nu \nu(x),\ \forall x \in \Omega. \tag{3.1}$$

The weights can be interpreted as a prior on the distributions in $\mathcal{C}$.

The first property of use follows trivially from the definitions of the quantities involved:

Lemma 3.2.5 *For the MDL estimator ρ and the Bayes mixture ξ, for any $y^k \in Y^k$ and any input $z_{y^k}^t \in X^t \times Y^{t-k}$ we have*

$$0 \le \rho(y^k|z_{y^k}^t) \le \xi(y^k|z_{y^k}^t) \le 1$$

A few simple properties of the MDL predictor are the following:

Lemma 3.2.6 *For any countable collection of measures $\mathcal{C}$, labels $y, a \in Y$ and objects z^t_{ya} from $X^t \times Y^{t-2}$*

1. $\rho^{ya}(ya|z^t_{ya}) =_{def} \rho(ya|z^t_{ya}) \geq \rho^z(ya|z^t_{ya})$
2. *ρ is an anti-semi measure:* $\sum_{a\in X} \rho^{ya}(ya|z^t_{ya}) \geq \sum_{a\in X} \rho^y(ya|z^t_{ya})$
3. $|\rho - \rho^x| \leq 1$ *and*
4. *For any $y^k \in Y^k$,*
$$\frac{\mu(y^k|z^t_{y^k})}{\rho(y^k|z^t_{y^k})} \leq w_\mu^{-1}.$$

Proof All these item are simple to establish.

1. From the definition of the estimator, we have
$$\rho(ya|z^t_{ya}) = \max_{\nu\in\mathcal{C}}\{w_\nu \nu(ya|z^t_{ya})\} \text{ and}$$
$$\rho^z(ya|z^t_{ya}) = \max_{\nu\in\mathcal{C}}\{w_\nu \nu(z|z^t_{ya})\}$$
so we see that ρ^{z^t} is a most equal to ρ, $\forall z^t \in Z^t$.
2. Follows directly from property 1. above.
3. Trivial, since both are in the interval [0,1].
4. We have $\mu(y|z_y) \in \mathcal{C}$ and since μ is the true distribution, we have that
$$w_\mu \mu(y|x_y) \leq w_{\nu^y} \nu(y|z_y) = \rho(y|z_y) \Leftrightarrow$$
$$\frac{\mu(y|z_y)}{\rho(y|z_y)} \leq \frac{1}{w_\mu} = w_\mu^{-1}.$$
and we are done. □

The next lemma is a central tool.

Lemma 3.2.7 *For any countable collection of measures $\mathcal{C}$ with $\mu \in \mathcal{C}$ and inputs $z^t_y \in X^t \times Y^{t-1}$ and the Bayesian mixture predictor ξ defined as in 3.1.*

$$\rho^y(y|z^t_y) - \sum_{a\in X} \rho(ya|z^t_y) \;\leq\; \xi(y|z^t_y) - \sum_{a\in X} \xi(ya|z^t_y)$$

for all $y \in Y^k$.

Proof In the proof we will make use of the fact from lemma 3.2.6 that $\rho^{ya}(ya|z_{ya}) \geq \rho^y(ya|z_{ya}), \forall y, a \in Y, z_{ya} \in Z^t - ya$, to get the inequality. The interpretation of $\nu(ya|z_{ya})$ is, that it is the probability measure of ya, given the object/input z_{ya}, i.e. a sample without the labels y and a.

We have, that

$$\begin{aligned}
\sum_{a\in X}\xi(ya|z_{ya}) - \sum_{a\in X}\rho^{ya}(ya|z_{ya}) &= \sum_{a\in X}\left(\xi(ya|z_{ya} - \rho^{ya}(ya|z_{ya})\right)\\
&\leq \sum_{a\in X}\left(\xi(ya|z_{ya}) - \rho^{y}(ya|z_{ya})\right)\\
&= \sum_{a\in X}\left(\sum_{\nu\in\mathcal{C}} w_\nu(ya)\nu(ya|z_{ya}) - w_{\nu^y}\nu^y(ya|z_{ya})\right)\\
&= \sum_{a\in X}\left(\sum_{\nu\in\mathcal{C}\setminus\{\nu^y\}} w_\nu\nu(ya|z_{ya})\right)\\
&= \sum_{\nu\in\mathcal{C}\setminus\{\nu^y\}} w_\nu\nu(y|z_{ya})\ [\mathcal{C}\text{ consists of measures}]\\
&= \sum_{\nu\in\mathcal{C}} w_\nu\nu(y|z_{ya}) - w_{\nu^y}\nu^y(y|z_{ya})\\
&= \xi(y|z_{ya}) - \rho^y(y|z_{ya}).
\end{aligned}$$

So, we get

$$\begin{aligned}
&\sum_{a\in X}\xi(ya|z_{ya} - \sum_{a\in X}\rho^{ya}(ya|z_{ya}) \leq \xi(y|z_{ya}) - \rho^y(y|z_{ya})\\
\Leftrightarrow\ &\rho^y(y|z_{ya}) - \sum_{a\in X}\rho^{ya}(ya|z_{ya}) \leq \xi(y|z_{ya}) - \sum_{a\in X}\xi(ya|z_{ya}).
\end{aligned}$$

□

We note in passing that $\xi - \rho$ is a semi measure.

The next couple of properties are slightly more difficult to establish. The first gives bounds between the true measure and a normalized MDL predictor, the second between two types of MDL predictors. The first is used in the proof for 3.2.10, the second is used in the proof of the main theorem of this chapter. Both he proofs are along the same lines, so only the second will be given. The proof for the first assertion is given in detail in [HP05].

Lemma 3.2.8 *For any countable collection of measures $\mathcal{C}$, such that the true measure μ is in $\mathcal{C}$ and any input $z \in Z^t$ we have*

1.
$$\sum_{t=1}^{\infty}\mathbf{E}\sum_{a\in X}\left(\mu(a|z_a) - \rho_{\text{norm}}(a|z_a)\right) \leq w_\mu^{-1} + \ln w_\mu^{-1}.$$

2.
$$\begin{aligned}
&\sum_{t=1}^{\infty}\mathbf{E}\sum_{a\in X}\Big|\rho_{\text{norm}}(a|z_a) - \rho(a|z_a)\Big|\\
&= \sum_{t=1}^{\infty}\mathbf{E}\left|1 - \sum_{a\in X}\rho(a|z_a)\right| \leq 2\ w_\mu^{-1}.
\end{aligned}$$

Proof Recall that $(\cdot)^+ = \max\{0, \cdot\}$. To prove the second item, we begin by noting

$$\begin{aligned}
\sum_a |\rho_{\text{norm}}(a|z_a^t) - \rho(a|z_a^t)| &= \sum_a \frac{\rho(a|z_a^t)}{\sum_b \rho(b|z_b^t)} \left|1 - \sum_b \rho(b|z_b^t)\right| \\
&= \left|1 - \sum_b \rho(b|z_b^t)\right| \\
&= \frac{(\sum_a \rho(a|z_a^t) - \rho(z^t))^+}{\rho(z_n^t)} + \frac{(\rho(z^t) - \sum_a \rho(a|z_a^t))^+}{\rho(z_n^t)},
\end{aligned}$$

and, by taking expectation and summing over all t

$$\begin{aligned}
& \sum_{t=1}^{\infty} \mathbf{E} \frac{(\sum_a \rho(a|z_a^t) - \rho(z^t))^+}{\rho(z_n^t)} + \sum_{t=1}^{\infty} \mathbf{E} \frac{(\rho(z^t) - \sum_a \rho(a|z_a^t))^+}{\rho(z_n^t)} \\
= \;& \sum_{t=1}^{\infty} \sum_{z_t \in Z^t} \frac{\mu(z^t)(\sum_a \rho(a|z_a^t) - \rho(z^t))^+}{\rho(z_n^t)} + \sum_{t=1}^{\infty} \sum_{z_t \in Z^t} \frac{\mu(z^t)(\rho(z^t) - \sum_a \rho(a|z_a^t))^+}{\rho(z_n^t)} \\
\leq \;& w_\mu^{-1} \sum_{t=1}^{\infty} \sum_{z_t \in Z^t} \frac{\mu(z^t)(\sum_a \rho(a|z_a^t) - \rho(z^t))^+}{\rho(z_n^t)} + w_\mu^{-1} \sum_{t=1}^{\infty} \sum_{z_t \in Z^t} \frac{\mu(z^t)(\rho(z^t) - \sum_a \rho(a|z_a^t))^+}{\rho(z_n^t)} \quad \text{[by 3.2.6.4]} \\
= \;& w_\mu^{-1} \sum_{t=1}^{\infty} \sum_{z_t \in Z^t} \left|\rho(z^t) - \sum_{a \in X} \rho(y^t a|x^{t+1})\right| \\
= \;& w_\mu^{-1} \sum_{t=1}^{\infty} \sum_{z_t \in Z^t} \left(\sum_{a \in X} \rho(y^t a|x^{t+1}) - \rho(y^t|x^t) + 2(\rho(y^t|x^t) - \sum_{a \in X} \rho(y^t a|x^{t+1}))^+\right) \\
= \;& w_\mu^{-1} \lim_{n \to \infty} \sum_{t=1}^{n} \sum_{z_t \in Z^t} \left(\sum_{a \in X} \rho(y^t a|x^{t+1}) - \rho(y^t|x^t) + 2(\rho(y^t|x^t) - \sum_{a \in X} \rho(y^t a|x^{t+1}))^+\right) \\
\leq \;& w_\mu^{-1}(2\xi(\emptyset) \underbrace{-\rho(\emptyset)}_{\leq 0} + \lim_{n \to \infty} \sum_{t=1}^{n} \sum_{z_t \in Z^t} (\underbrace{\rho(z^t) - 2\xi(z^t)}_{\leq 0})) \quad \text{[by 3.2.7 and telescoping]} \\
\leq \;& w_\mu^{-1}(2\xi(\emptyset)) \\
\leq \;& 2w_\mu^{-1} \text{ [since } \xi \in [0,1] \text{]},
\end{aligned}$$

where we have used that $|u| = u^+ + (-u)^+ = -u + 2u^+$ and $1 \geq \xi \geq \rho \geq 0$. $\square$

Yet another result necessary to obtain the proof of the main theorem is

Lemma 3.2.9 *For any collection of measures $\mathcal{C}$, such that the true measure μ is in $\mathcal{C}$ and any input $z \in Z^t$ we have*

$$\sum_{t=1}^{\infty} \mathbf{E} \sum_{a \in X} \left| \rho_{\text{norm}}^{z_a}(a|z_a) - \rho^{z_a}(a|z_a)\right| = \sum_{t=1}^{\infty} \mathbf{E} \left|1 - \sum_{a \in X} \rho^{z_a}(a|z_a)\right| \leq w_\mu^{-1}.$$

Proof The first equality holds because the normalized MDL predictor summed over all possible outcomes sums to 1. Let $l(x)$ denote the size of the example x.

To get the inequality, we have by 3.2.7 and 3.2.6.4 for all $n \in \mathbf{N}$

$$\begin{aligned}
\sum_{t=1}^{n} \mathbf{E} \left| 1 - \sum_{a \in X} \rho_{\text{dyn}}^{z_a}(a|z_a) \right| &= \sum_{t=1}^{n} \mathbf{E} \frac{\rho(z_a) - \sum_{a \in X} \rho(z_a a)}{\rho(z_a)} \\
&= \sum_{t=1}^{n} \sum_{l(z_a = t-1)} \mu(z_a) \frac{\rho(z_a) - \sum_{a \in X} \rho^{z_a}(z_a a)}{\rho(z_a)} \\
&\le w_\mu^{-1} \sum_{t=1}^{n} \left(\sum_{l(z_a=t-1)} \rho(z_a) - \sum_{a \in X} \rho^{z_a}(z_a a) \right) \\
&\le w_\mu^{-1} \sum_{t=1}^{n} \left(\sum_{l(z=t-1)} \xi(z) - \sum_{a \in X} \xi(za) \right) \\
&\le w_\mu^{-1} \left(\xi(\emptyset) - \sum_{l(z_a=n)} \xi(z_a) \right) \\
&\le w_\mu^{-1},
\end{aligned}$$

since the series *telescopes*, i.e. successive positive and negative terms cancel each other out in the length of the examples. Since this holds for all $n \in \mathbf{N}$ it also holds in the limit. □

Now, for measurable functions $f, g : X^n \to [0,1]$ and inputs and history $z \in Z^t$, define

$$\Delta_{z_a}(f, g) =_{\text{def}} \left(\sum_{t=1}^{\infty} \mathbf{E} \left\{ \sum_{a \in X} (f(a|z_a) - g(a|z_a))^2 \right\} \right)^{\frac{1}{2}}.$$

Since $\Delta_u(\cdot,\cdot)$ is proportional to the 2-norm, the triangle inequality $\Delta_{z_a}(f,h) \le \Delta_{z_a}(f,g) + \Delta_{z_a}(g,h)$ holds.

Lemma 3.2.10 *For any countable collection of measures $\mathcal{C}$, such that the true measure μ is in $\mathcal{C}$ and any input and history $z \in Z^t$ we have*

$$\sum_{t=1}^{\infty} \mathbf{E} \sum_{a \in X} \Big(\mu(a|z_a) - \rho_{\text{dyn}}^{y_{<t}}(a|z_a) \Big)^2 \le 8 w_\mu^{-1}$$

i.e.

$$\Delta^2(\mu(a|z_a), \rho_{\text{dyn}}^{y_{<t}}(a|z_a) \le 8 w_\mu^{-1}.$$

Proof Since

$$\ln w_\mu^{-1} \le w_\mu^{-1} - 1 \le w_\mu^{-1}$$

we have by 3.2.8

$$\Delta(\mu, \rho_{\text{norm}}) \le \sqrt{2 w_\mu^{-1}}.$$

Since by lemma 3.2.6.3 $|\rho_{\text{dyn}} - \rho_{\text{norm}}| \le 1$, we get by multiplying both sides of 3.2.8 with that quantity, that

$$\Delta(\rho_{\text{norm}}, \rho_{\text{dyn}}) \le \sqrt{2 w_\mu^{-1}}.$$

By the triangle inequality we get

$$\Delta(\mu, \rho_{\text{dyn}}) \leq \Delta(\mu, \rho_{\text{norm}}) + \Delta(\rho_{\text{norm}}, \rho_{\text{dyn}}) \leq \sqrt{2w_\mu^{-1}} + \sqrt{2w_\mu^{-1}} = 2\sqrt{2w_\mu^{-1}}$$

so

$$\Delta(\mu, \rho_{\text{dyn}}) \leq \sqrt{8w_\mu^{-1}}.$$

□

We are thus ready to prove the result in theorem 3.2.4, the main result.

Proof of Theorem 3.2.4 In order to get

$$\sum_{t=1}^{\infty} \mathbf{E} \sum_{a \in \{0,1\}} (\mu(a|z_a) - \rho(a|z_a))^2 \leq 21w_\mu^{-1}$$

that is

$$\Delta^2(\mu(y|z_y), \rho(y|z_y)) \leq 21w_\mu^{-1}$$

we have from lemma 3.2.6 and by the triangle inequality that

$$\begin{aligned}\sum_{a \in X} |\rho_{\text{dyn}}(a|z_a) - \rho_{\text{static}}^{z_a}(a|z_a)| &= \left|\sum_{a \in X} \rho(a|z_a) - \rho_{\text{static}}^{z_a}(a|z_a)\right| \\ &\leq \left|\sum_{a \in X} \rho_{\text{dyn}}(a|z_a) - 1\right| + \left|1 - \sum_{a \in X} \rho_{\text{static}}^{z_a}(a|z_a)\right| \\ &\leq 2w_\mu^{-1} + w_\mu^{-1},\end{aligned}$$

with the last inequality obtained from lemma 3.2.8 and lemma 3.2.9. Therefore, by 3.2.6.3

$$\Delta^2(\rho_{\text{dyn}}, \rho_{\text{static}}), \leq \sum_{t=1}^{\infty} \mathbf{E} \sum_{a \in X} |\rho_{\text{dyn}}(a|z_a) - \rho_{\text{static}}^{z_a}(a|z_a)| \leq 3w_\mu^{-1}.$$

We also have, by lemma 3.2.10

$$\Delta(\mu(y|z_y), \rho_{\text{dyn}}(y|z_y)) \quad \leq \quad 2\sqrt{2w_\mu^{-1}}$$

Since the triangle inequality holds for $\Delta(\cdot, \cdot)$ we get, by squaring both sides of the inequality:

$$\begin{aligned}\Delta^2(\mu(a|z_a), \rho_{\text{static}}^{z_a}(a|z_a)) &\leq \Delta^2(\mu(a|z_a), \rho_{\text{dyn}}(a|z_a)) + \Delta^2(\rho_{\text{dyn}}(a|z_y), \rho_{\text{static}}^{z_a}(a|z_a)) \\ &\quad +2 \cdot \Delta(\mu(a|z_a), \rho_{\text{dyn}}(a|z_a)) \cdot \Delta(\rho_{\text{dyn}}(a|z_y), \rho_{\text{static}}^{z_a}(a|z_a)) \\ &= \left(2\sqrt{2w_\mu^{-1}}\right)^2 + 3w_\mu^{-1} + 2\left(2\sqrt{2w_\mu^{-1}}\right)\sqrt{3w_\mu^{-1}} \\ &= 8w_\mu^{-1} + 3w_\mu^{-1} + 4\sqrt{2}\sqrt{3}w_\mu^{-1} \\ &= 11w_\mu^{-1} + 4\sqrt{6}w_\mu^{-1} \\ &\leq 21w_\mu^{-1},\end{aligned}$$

since $2.5 \geq \sqrt{6}$, and we are done. □

The following theorem establishes the relation between the bound in last theorem and convergence in mean sum and thus completes our development of convergence results.

Theorem 3.2.11 *For any two measures φ and μ*

$$\varphi \overset{\text{i.m.s.}}{\longrightarrow} \mu \Leftrightarrow \sum_{t=1}^{\infty} \mathbf{E} \sum_{a \in \{0,1\}} (\mu - \rho)^2 < \infty.$$

Proof See [Ser80].

Convergence in mean sum is a very strong convergence concept, see lemma 3.1.2.

There is more information about our method to be distilled from theorem 3.2.4. In their paper, Hutter and Poland indicate how the rate of convergence can be quantified, see [HP05], page 5.

Convergence and the Meta-Algorithm Our algorithm adds the hypothesis obtained from the *new* dataset into the collection of hypotheses, which gives rise to the collection of measures. This assures convergence of our method, contingent on the convergence of the underlying classifier. As this is a meta-algorithm, this seems all we could hope for. For completeness, we state this as a theorem:

Theorem 3.2.12 *Given a realizable learning problem, the algorithms given in Algorithm 1 , 2 and 3 converge i.m.s. to the true hypothesis.*

Proof *Trivial, by theorem 3.2.4 and 3.2.11.*

As an example, the hypothesis space for decision tree learners is complete, so our method is complete also, i.e. converges to the true measure almost surely, when implemented for decision trees.

In the next chapter we'll illustrate the usage of the meta-algorithm.

Chapter 4

Instantiation Of Meta-Algorithm

In this chapter we describe how the meta-algorithm from Section 2.3 can be instantiated with a decision tree algorithm, giving the L and TL algorithms of Figure 2.2. We postpone M to the next chapter.

First we describe decision tree learning and an algorithm to construct decision trees. Then we describe how to obtain the codelengths from these constructs and finally how to code exceptions in data from a tree. When we have discussed these things we can try out the framework on data, which we'll do in the next chapter.

4.1 Decision Tree Learning

In this section we explain decision tree learning and its most popular embodiment. After giving a brief overview of the main ideas, we'll explain some subtleties of how decision trees work in relation to a freely available implementation, J48 in Weka [WF05], which we use in the next chapter. For a rigorous probabilistic treatment of tree classifiers quite different from the one presented here, see [DGL97].

4.1.1 General Ideas

Consider an example space consisting of a discrete input space X and a discrete output space Y, linked by some function μ. Decision tree learning is a method for learning such functions μ when presented with a set of training examples, which can easily be adopted to a continuous input space. The learned structure is represented as a tree.

Inputs are classified from the root down to a leaf node, which gives the class of the object being classified. In each internal node, including the root, a test is performed on the value of a feature of the example being classified and the example travels down the tree until it reaches a leaf, which gives the class for that example. These internal nodes do not necessarily need to test all features to get a classification and can base them on a subset of available features.

Decision trees themselves are just a structure for representing concepts, i.e. functions, and as such have no preference for one structure or the other. A bias is provided by the algorithms that are used to construct the trees. Usually they take the form "use as short a tree as suitable".

Most decision tree learning algorithms build their trees by recursively splitting the input data based on some feature depth-first, until either the examples or the features are exhausted. These splits partition the input space into complex decision boundaries, as is e.g. shown in Figure 4.1.

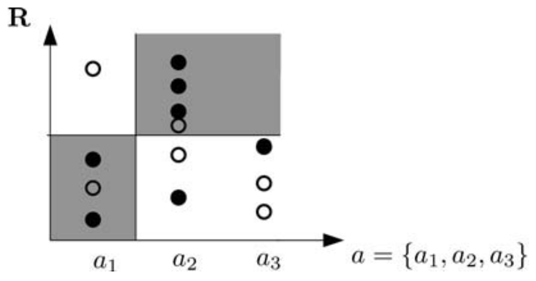

Figure 4.1: A partition of the input space for 2 features, one continuous, the other discrete. Examples are represented as circles for a class of '0' and a disk for '1'. The class for the shaded region is '1', the other '0'.

Decision trees are complete in the sense that for discrete input and output spaces they can represent all functions between those. Decision trees are a disjunction of conjunctions. Missing data is not dealt with in normal decision tree procedures and needs to be handled specially.

4.1.2 C45 and Weka's J48

Quinlan's decisions tree algorithms have long been as close to "off the shelf" machine learning as you can get, starting with ID3 and continuing to the present C5.0. The reason for this is, in addition the properties discussed above, that these algorithms are very versatile and robust. They handle both discrete and continuous data, are not overly sensitive to the information criterion used to construct the trees and have a very intuitive representation. A short description of J48, an implementation of C4.5, version 8, follows.

Building

The procedure for building a tree proceeds by recursively splitting the dataset based on some feature, which is selected according to an information measure, until there is no feature left to split on or no more data with different labels in the subset. The resulting leaf is assigned a class-label by majority vote. This procedure is a locally optimal (i.e. *greedy*) so after the building is done we might want to reconsider some structures, which is done by pruning or subtree raising.

There are basically two possibilities for an information criterion to split the dataset: *information gain* and the *gain ratio*. Both are based on the *entropy* of a dataset S.

Definition 4.1.1 *The* entropy *of a dataset S with outcomes in $\mathbf{A} \leftrightarrow \{1, \ldots, n\}, n \in \mathbf{N}$ is*

$$H(S) = \sum_{a \in \mathbf{A}} -p_i \log_2(p_i)$$

where p_i is the probability of the $i-th$ symbol. When the probabilities aren't known we make use of the relative frequency of each symbol in S.

Entropy measures how much we can compress a given dataset. More specifically, it gives the upper bound for compression of the dataset. In some sense, it also gives a measure of how much information is contained in a dataset. Entropy is higher as the distribution on the symbols is more uniform.

Definition 4.1.2 *The* information gain *of a feature A given a dataset S is*

$$Gain(S, A) = H(S) - \sum_{v \in Value(A)} \frac{|S_v|}{|S|} H(S_v),$$

where $Value(A)$ represents the set of possible values of A and S_v the subset of S corresponding to the value v of A, i.e. $S_v = \{s \in S | A(s) = v\}$.

The information gain measures the expected reduction of entropy by knowing the value of the feature A. Thus it measures how much information we would be extracting from the dataset by splitting on A.

This measure does not take into account how many different values the feature A takes. If A takes on a large number of values, knowing the value of A will obviously greatly reduce the entropy of the dataset and also make the resulting tree more complex. We wish to penalize this increase in complexity. This is to some degree done by the *gain ratio*. First we define the *split information:*

Definition 4.1.3 *The* split information *of a feature A given a dataset S is*

$$SplitInfo(S, A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \left(\frac{|S_i|}{|S|} \right),$$

where $S_1, S_2, \ldots, S_c$ are the c subsets of examples resulting from partitioning the dataset S by the c-valued attribute A.

This measures how broad and uniform the split on A is. To paraphrase [Mit97]: $SplitInfo$ is the entropy of S with respect to the values of a feature A, which is in contrast to the use in $Gain$, where we considered only the entropy of S with respect to the target feature whose value is to be predicted by the learned tree.

We use the split information to define the information gain ratio:

Definition 4.1.4 *The* information gain ratio *of a feature A is*

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInfo(S, A)}$$

When using this criterion to split the data we are looking for a suitable balance between the $Gain$ and the $SplitInfo$, such that when A takes on a large number of values the gain is penalized. This sacrifices some of the elegance of simply using the information gain but is more in the MDL style, which we prefer.

To handle continuously valued features the J48 algorithm applies *linear discriminant analysis* on one feature, selecting the single optimal splitting in one dimension by simply selecting the value the gives the best split. For other ways of doing this, see [DGL97].

The trees produced by this procedure tend to be overly specific since it continues the procedure until we have either finished the labels or exhausted the data. Thus we overfit the tree to the data and have poor out of sample performance. Instead of fixing this drawback to the procedure during the building phase, we take another look at the tree after building it and *prune* the branches as described next.

Generalizing after building

To prune a tree J48 offers *rule post pruning.* There is a bijection between a decision tree and a set of rules. The pruning procedure maps a tree to a collection of rules by making a rule from each leaf node to the root. Then generalize each rule by removing those preconditions that give improved estimated accuracy of the rule, as estimated by the data. This estimate can be obtained by any non-parametric or out-of-training-sample method. Then, the rules are sorted based on the estimated accuracy and applied in this order. The benefit of using rules is that the (apparent) dependency on the order in which the features are considered is removed, allowing for discrimination between the context in which the rules are used in a more global way, resulting in better generalization performance.

Subtree raising is another way to prune a constructed tree offered by J48. This method involves deleting nodes, redistributing the training set over the new tree and evaluating the usefulness of the construction (or, more properly, the destruction). This method is computationally expensive.

Another way to prune a tree is to use the MDL principle. This is described in [MRA95]. The results of using MDL to prune do not indicate that the addition in complexity of the procedure justifies its use. The *GainRatio* can be considered as an approximation to MDL.

4.1.3 Obtaining Measures from Trees

To apply the theorems from Chapter 3 we need to convert our decision trees to probability measures. Consider an example space (X, Y) and a sample S^n drawn from this space. Each decision tree $h : X \rightarrow Y$ implicitly defines a probability measure over the inputs $x \in X$, given a set of examples. For a given input x the measure of a $y \in Y$ is simply the relative frequency of y in the leaf of the tree. For a binary classification problem this becomes

$$\begin{aligned} P_h(y = 0|S^n) &= \frac{|\{x \in S^n : h(x) = 0\}|}{n}, \\ P_h(y = 1|S^n) &= 1 - P_h(y = 0|S^n). \end{aligned}$$

This also gives us a predictor. The prediction of a label $y \in Y$ given its object $x \in X$ is $\max_{y \in Y} P(y|x)$.

Example Say we have the tree depicted in Figure 4.2. X has the alphabet $\{a, b\}$ and Y the alphabet $\{s, t\}$. We have the input $\{((a, s), 0)\ , ((a, s), 0),$

$((a,s),1)$,$((a,t),0)$,$((a,t),1)$, $((a,t),1)$, $((b,s),0)$, $((b,s),1)$, $((b,s),1)$, $((b,s),1)$, $((b,t),0)$, $((b,t),0)$, $((b,t),0)$, $((b,t),1)\}$. Then the induced probability distribution is $P(0|(a,s)) = 2/3$,$P(0|(a,t)) = 1/3$, $P(0|(b,s)) = 1/4$, $P(0|(b,t)) = 3/4$ with the remaining probabilities given by the complementary property of probabilities.

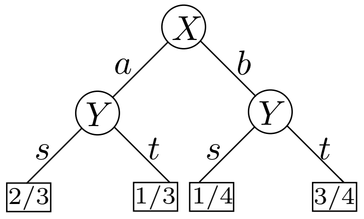

Figure 4.2: Tree for example 4.1.3. Probabilities of class '0' in boxes.

Remark A proof of the convergence of a tree classifier can be found in [DGL97].

4.2 Coding Trees and Data

To apply our algorithm with decision trees we need to obtain codelengths for trees and exceptions from the trees. This we describe now.

4.2.1 Coding Trees

To code the trees, we will use the following scheme, basically based on a paper by Quinlan and Rivest, [QR89]. We are not concerned with tree construction, so we can allow ourself a slight modification in the coding of exceptions and default class labels by doing them in bulk, i.e. all at once. This gives us an approximation to the optimal code, which has a relative bias towards smaller trees.

As in Figure 2.1 we have two agents, B and C, such that B has the complete data and C has the data, without the class-labels, B needs to compactly describe a hypothesis, i.e. tree, and the exceptions in the data from a hypothesis, such that the combined description length is minimized. Moreover, and perhaps most importantly, B and C agree beforehand on the exact ordering of the values each feature can take, so that when the we name an feature there is no need to name the order in which it's values appear, so that when we list whether the child of a node in a tree are (proper) subtrees or leaves, there is no ambiguity about which feature label each subtree belongs to. For inference purposes, the description method itself is of no interest, only the length.

For simplicity, we will only consider binary classification problems, but will account for non-binary trees. The extension to multiclass problems obscures the basic ideas.

There are three things to code, two for the tree itself, the third for the data

1. (a) structure, the features selected at each node and

   (b) cut values for continuously valued features

2. default classes for leaves
3. exceptions in the data from the classes given by the tree

The method is recursive, depth first and is as follows:

1. Tree structure description
   (a) *Structure description*
   The length of the code is obtained by:
   - the codelength for specifying the feature at each node is $\log(k)$, where k is the number of features left to consider in the tree and log is the logarithm to base 2. For continuously valued features, we do not remove them from the set of available features when coding. To see that this is indeed the length, note that after a feature has been used in the tree, it cannot occur in the subtree emanating from that node, except for numeric features which may have other cut values. Call this length $L_A(k)$. We use Q&R's simple code, see [QR89], which uses one bit to designate whether a node is a leaf or not.
   - the codelength for a leaf is 1 (in addition the codelength for specifying the class, which is coded globally)
   - the codelength for a subtree (not including leaves) is $1 + L_A(k) +$ the codelength of the subtree thereunder.

   (b) *Cut value description*
   When using continuously valued features in a tree we cut the dataset into two parts. Each example is then classified as being either larger or smaller then this value. Each time we use a such a feature we get a new cut-value. These values we need to communicate; in our framework B is the sender and C is the receiver.

   We can at least take two approaches. First, we can imagine that B has an ordered list of values, in the order used in the tree (top-bottom, left-right). This list defines the value of the cut for each application of a feature, since C can pop the values of the list for each application of the feature. This list needs to be sent to C, which can be done by arithmetic or range encoding for instance.

   Another way would be to view each cut value as a feature by itself. This would entail a dynamic set of features for use in classification, since the feature-set would be constructed after the samples are obtained and thus B and C would not be able to share the features as is conventionally done in such circumstances. This approach will not be further addressed and we will stick to the first approach.

   Use of arithmetic coding for transmitting the list of cut values adds only little to the codelengths involved and complicates the implementation further. We will imagine that the list of cut values is shared by B and C, since we are mainly concerned with inferring a good prior from $\mathcal{M}$, see Figure 2.1 for instance.

2. Codelength for classes

   Now we have a tree, with the position and number (k) of the leaves given. We can then describe the default class for each leaf as follows:

- take the whole sequence of k leaves and give the position of l exceptions from the default class[1]. This can be done with $L(\cdot,\cdot)$ bits[2]. The function L is given by

$$L(k,l) = \log\left(\frac{k+1}{2}+1\right) + \log\binom{k}{l}$$

  and is elaborated on in Section 4.2.2 on coding strings of '0's and '1's

If one is concerned with the construction of trees, a more segmented approach may be beneficial, for instance to simplify the search problem. We are only concerned with the descriptions of given trees in this paper so we will employ this technique.

3. Coding exceptions

   Recall that B and C share all inputs and B is to transmit only the class of each example to C. To code exceptions from the data, we take as input the whole data sequence (n things), the order of which is shared by B and C. The specific order of the dataset does not matter, and it doesn't seem to be a limiting assumptions that the order is shared. After taking our hypothesis into consideration, i.e. classifying with a decision tree, we have some exceptions from the hypothesis, and those exceptions we encode as a single sequence, which can be done with $L(n,w)$ bits, where w is a wrongly classified example.

Example To see that the codelength is as described and the code is indeed prefix-free we'll give a short example with the tree in Figure 4.3. By following

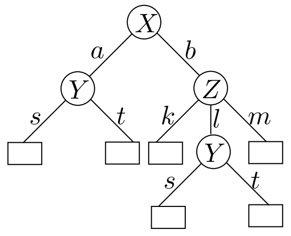

Figure 4.3: A tree coded in example

the procedure described above, traversing the tree depth first from the root, left to right, we get the code

```
1 X 1 Y 0 0 1 Z 0 1 Y 0 0 0
```

What if we are given the code above and want to reconstruct the tree, that is to decode a message of that form? In words, we start by reading the first '1' and have the root X. Then, we have the left branch [3] consisting of the non-leaf node (second '1') Y and then it's two children are both leaves (the first two '0's). Then, we regress back up the tree to the last non-leaf node, since there are no

[1]Recall, we only address the two class classification problem here.
[2]This may be inefficient for small tree sizes
[3]Left and right only need to be consistently used in the coding and decoding.

n	k	$L(n,k)$
10	0	2.7004
10	1	6.0224
10	2	8.1923
10	3	9.6073
10	4	10.4147
10	5	10.6777
100	0	5.6865
100	10	49.6632
100	50	102.0352
1000	0	8.9701
1000	10	86.7718
10000	0	12.2881
10000	10	123.3677

Table 4.1: Some values of the codelength function $L(n,k)$

more non-leaf nodes in this subtree, which is X. We continue to construct the rest of the tree in the same way.

There are other views on how to code trees, see [WP93] for a discussion of Quinlan and Rivest's paper and further analysis.

4.2.2 Coding strings of '0' and '1's

Here we describe the codelength function $L(n,k)$ which can be used to code exceptions from any hypothesis in a binary classification problem. In binary classification problems, each bit tells us if a class is correctly described by a hypothesis or not. Since we may assume that no more then half of the classes are incorrectly described by a hypothesis, at least half of the bits are 1, denoting that the class was correctly given by a hypothesis.

Now, I want to communicate a bit string of length n with k 1s, given an upper bound b on k. First I transmit the value of k using $\log(b+1)$ bits. This is because we know that the value of k is no larger then b, so we need only say which element of the set $\{0,1,2,\ldots,b\}$ we are referring to, which can be done by bisecting the set until there is only one element remaining. This code is sufficient for our purposes but we point out that there are quite a few codes for coding the integers, see [Grü04].

Then, when I've told you k there are $\binom{n}{k}$ possible strings of length n with exactly k 1s. We both know the value of n beforehand, so the codelength function L is given by

$$L(n,k) = \log\left(\frac{n+1}{2}+1\right) + \log\binom{n}{k}$$

Some values for this codelength function are given in Table 4.1.

4.3 Instantiation of Meta-Algorithm

In this chapter we have thus far described decision tree learning, its most popular implementation and how to code decision trees. In other words, we have described the L and TL algorithms of Figure 2.2. The methods given for the coding of hypothesis are only dependent on the representation of the hypotheses and may therefore be applied to any decision tree implementation.

We are now only left with specifying the M algorithm of Figure 2.2 to have a full instantiation of our transfer learning framework. A very simple implementation of the M algorithm would be to simply limit the size of the decision trees. The rationale for doing this is that these partial trees allow us to transfer the most significant knowledge from the domains in increasing order of complexity in a way that is relatively simple to implement. This is the version of M we use for our experiments in the next chapter. Before turning to data, we discuss what we are, or should be, optimizing.

4.4 How to Combine Environment Knowledge and Task Specifics

To conclude this chapter, we give an illustration of how an application of our framework could proceed. An appealing way of looking at transfer learning with decision trees might be the following. We have some data and some environment knowledge. We consider the environment to be relevant to our current task and thus wish to *initialize* a decision tree with knowledge from the environment and subsequently build on that basic knowledge with the data for the new task. This might also be viewed as elaborating with task specifics on knowledge from the environment. We might want to do this because there are some specifics in the current task that are not in the previously analysed tasks or that the current task has some features not seen previously. This sort of initialization is in effect guiding the search for a good hypothesis towards the hypothesis suggested by the environment, i.e. giving the learning a relative bias.

In general, injecting a bias by initializing the hypothesis in such a way would require an invasive surgery of the learning algorithm with quite a few pitfalls to look out for. In the case of decision tree learning, subsequent pruning of the tree after the building would be quite elaborate, since the pruning procedure would need to consider other things then just the relevance of the structure under consideration w.r.t. the data. Our framework wraps around the underlying learning algorithm obliterating the need to change the algorithm itself and exhibits the behaviour described above very naturally.

To illustrate this behaviour, say we have some data S and simple environment knowledge consisting of a single hypothesis h, which partitions the data as shown in Figure 4.4 (1)a. Now, in addition to this hypothesis h, we have constructed a hypothesis from the data, but after considering the environment we reach the conclusion, by calculating the minimum description length, that h describes our task better. Thus we apply h to S and subsequently get the sets S1, S21, S22 and S23 of examples. These examples we re-consider. For each of these sets we repeat the process, but the only hypothesis from the environment has already been applied to this data and doesn't need to be reconsidered. Thus, we only need to consider the structures learned from each of these smaller datasets and the empty hypothesis. If there is no learning to be done from the data, as viewed by the learning algorithm, that branch is done. If some non-empty hypothesis is relevant, as per the minimum description length, we apply it to our data and repeat the process. In this way we might find a structure applicable to a subset, such as depicted in Figure 4.4 (1)b.

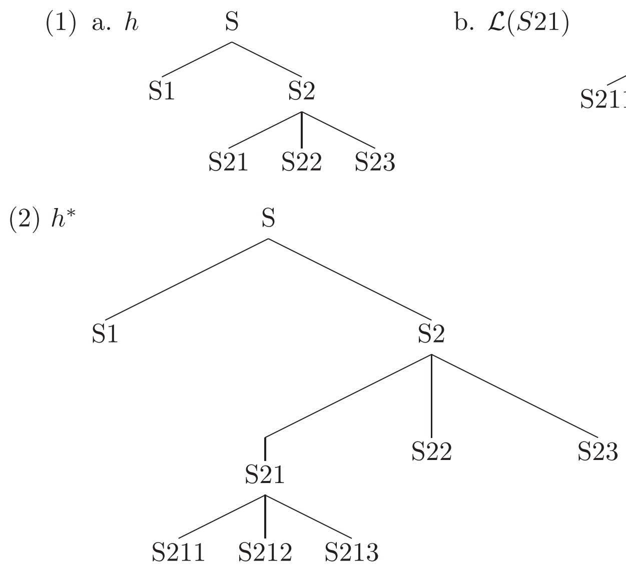

Figure 4.4: Relationship between datasets involved in inducing a decision tree from a dataset and an environment in our framework. In (1)a. we use the hypothesis h on the data S, which splits the data into the sets S1, S21, S22 and S23, which might contain misclassified examples that we could use to improve performance. In (1)b. we apply the learner $\mathcal{L}$ to the set S21, obtaining a structure such as illustrated. In (2) we have combined all the learning done on S in our framework into one structure.

When there are no more hypotheses to be evaluated, we are done. The hypothesis we deem most suitable for the data, given the environment and our subjective beliefs about it, for this hypothetical example is given in Figure 4.4 (2). This tree has as its most informative sections, as per the current task, the knowledge transferred from the environment. For each of the nodes from the transferred knowledge we considered local information to extend the tree, and found one structure interesting enough to add to the tree, as measured by the minimum description length. In this way we can use global properties of the environment as the core of the knowledge about the current task and adopt to specific local knowledge as warranted by the data. Intuitively, this feels like a sensible thing to do and our framework is capable of capturing this behaviour in a elegant way, striking a balance between global properties of the environment and adapting to local ones.

Under which conditions would we observe this behaviour described above? The hypothesis space would need to contain a decent hypothesis to start with and the prior weights would need to be sensible. We are dependent on good subjective judgement concerning the usefulness of the environment. However, if the environment poorly fits our new task, it should not have a very adverse effect on the performance of the underlying learning algorithm, although some slow down is to be expected.

Taking a step back from the situation described above, it might be suggested to first consider *only* hypothesis from the environment, excluding hypotheses induced from the new task. Then, after deciding which hypothesis from the environment best fits the new task, take a look at what might be induced from the new data to extend that hypothesis. The bias put on the method like this is quite strong and would be of use, if shown to be sensible. However, this is not a very intelligent course of action for several reasons. If the environment is poorly constructed, we might be forced to use hypotheses that are very ill-suited for the new task. Further, this makes convergence to a true model contingent on the truth being available from the start, which seems to be a very strict assumption. By using the methodology advocated in this paper, each and every hypothesis must earn its place and prior beliefs are tested against what can be induced from the data at each and every step. Reference to Galileo, who struggled against accepted "wisdom", is in order: *In questions of science the authority of a thousand is not worth the humble reasoning of a single individual.* Our method is conservative in nature, not dogmatic.

Is the final structure given by our method optimal, in the sense that it minimizes the probability of misclassification of examples from the current task? If the subjective prior weights assigned to hypotheses from the environment are in accordance with the current task, the sum total codelength for all the hypotheses produced in each recurrence is less then that of h^*, the optimal hypothesis according to our method, and our assumptions on the validity of the MDL inference procedure agree with reality, h^* does indeed minimize this error, as shown above in Theorem 3.2.12.

Would we get the same structure if only considering the current task? Almost certainly not, unless we have large amounts of data and the knowledge from the environment is superfluous to the task. If this were not the case, our method and transfer learning would be futile.

Next, we turn our attention to data.

Chapter 5

Experiments

“ Torture the data long enough, and it will admit to anything you want. ”
— Unknown

Now it is time to try the ideas described so far on some data: perform some experiments. The purpose of the experiments is twofold. On one hand, we want to go through the hoops of implementation of the ideas described; to get a feel for the workings of the algorithm; to see that it is actually usable from a implementation standpoint. On the other hand, we are looking to see whether our concept of transferability does anything at all from a learning standpoint. To see behaviour different from the situation where there is no option of transfer. To see if transfer can be induced by the MDL principle across domains in an environment in this framework.

The point of the experiments is thus that knowledge from the environment that fits the data may initially, with few examples, be better than knowledge from the data only. This is because we are basing our inference on more information then the data alone; we usually know more about the situation that the data comes from then just the data itself[1]. We can almost always do with more data, or knowledge , either to improve our hypotheses or to verify results. Data is a bit like money: I don't know anyone how has enough of either. The experiments show that indeed the method produces hypothesis that are different from the ones obtained from the data alone, have shorter codelength according to our assumptions and recover the truth asymptotically.

In the next section we illustrate some properties of our method in as simple a setting as possible. In the section thereafter we address a situation closer to a real world setting.

5.1 Simple Simulated Example

Here we show how our method might behave on a artificial example. Without tweaking, it finds the true distribution from the start with high probability. This example also illustrates the asymptotics of our method: by adding the

[1]This is pretty much the subjective Bayesian viewpoint.

empirical hypothesis, the one induced from the data alone, to the environment with reasonable weights, the method has good asymptotic bounds.

Say we have a domain that generates symbols from the alphabet $\{a, b, c, d\}$ according to the distribution P, with the probabilities given in Table 5.1. We want to learn the distribution for this source of data. For this little thought experiment we will assume that we have several distributions that can be seen as hypotheses from domains in an environment.

For simplicity, we use powers of one-half for the probabilities. The codelengths correspond to the powers of this number. The codelengths are Huffman codes, or prefix free codes. The code-tree is shown in Figure 5.1.

Symbol (X)	$P(X)$	$C_P(X)$	$Q(X)$	$C_Q(X)$	$R(X)$	$C_R(X)$
a	1/2	0	1/8	111	1/2	0
b	1/4	10	1/8	110	1/8	111
c	1/8	110	1/4	10	1/4	10
d	1/8	111	1/2	0	1/8	110

Table 5.1: Three distributions over $\{a, b, c, d\}$ and corresponding prefix codes.

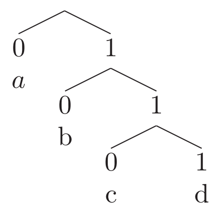

Figure 5.1: Huffman code tree for the distribution in Table 5.1

We see in that table that the distributions P and Q are further from each other then P and R. How much further? We answer that by calculating the Kullback-Leibler divergence between these two distributions. The *KL-divergence* is defined by

$$D(f||g) = \sum_{x \in X} f(x) \cdot \log_2 \left(\frac{f(x)}{g(x)} \right)$$

for any two probability measures $f, g : X \to [0, 1]$ and an countable X[2].

What makes KL-divergence interesting is that we can interpret it as the additional cost, in bits when using the base 2 logarithm, of coding a sequence using g which was really generated by f[3]. The KL-divergence from P to Q and

[2]For uncountable X the sum is becomes an integral.

[3]Keeping the correspondence between codelengths and distributions discussed in Section 1.3 in mind.

R is as follows:

$$\begin{aligned}
D(P||Q) &= \sum_{x\in\{a,b,c,d\}} P(x)\cdot \log_2\left(\frac{P(x)}{Q(x)}\right) \\
&= \frac{1}{2}\cdot\log_2\left(\frac{1/2}{1/8}\right)+\frac{1}{4}\cdot\log_2\left(\frac{1/4}{1/8}\right)+\frac{1}{8}\cdot\log_2\left(\frac{1/8}{1/4}\right)+\frac{1}{8}\cdot\log_2\left(\frac{1/8}{1/2}\right) \\
&= \frac{1}{2}\cdot\log_2(4)+\frac{1}{4}\cdot\log_2(2)+\frac{1}{8}\cdot\log_2(1/2)+\frac{1}{8}\cdot\log_2(1/4) \\
&= 7/8,
\end{aligned}$$

and in a similar way

$$D(P||R) = 1/8.$$

Thus we incur an expected extra cost of 7/8 bits when coding from P under the code for Q, and 1/8 for R.

To use TI, Algorithm 3, we need to encode the probabilistic model we are working with. This we do by giving an index into a list of all possible models. The number of different models that can arise for encoding a 4 symbol source with a Huffman code is $2\cdot 4\cdot 3\cdot 2 = 48$, where the first '2' comes from the fact that there are essentially only two possible binary tree constructs possible[4], the '4' from choosing the first letter to put into the tree, '3' from choosing the second and the last '2' for choosing the second to last letter. Thus we can give an index into this list of equiprobable models by using $\lceil \log_2(48)\rceil = 6$ bits.[5] The learner L is a simple empirical estimate of the probability and M has nothing to do.

Now we can run simulations. We use the Mergence Twister algorithm of [MN98] to generate uniformly distributed numbers in the interval $[0, 1]$ and assign each symbol to a length of that interval corresponding to its probability. An example of a single run is given in Figure 5.2. Several questions come to mind concerning our method.

First, using our weighted two part measure of transferability, do we recover the true distribution P when it is available in the hypothesis space? Yes, we have a strong preference for the *truth*, although we adjust to variations in the data and will occasionally prefer the empirical distribution. This is not the large sample situation, as discussed below.

Second, do we ever like bad hypothesis? When we have a hypothesis that is almost true, such as the distribution R, we will almost never select a very bad hypothesis such as the distribution Q. If the bad hypothesis is bad enough, the empirical one will out perform it from the start, as seen in Fig. 5.2.

Third, say that the truth isn't available but the distribution R is. Will our method select the distribution R as the best one? When, if ever, will the method prefer the empirical distribution? As we see Figure 5.2, we will start by using R as our working hypothesis about the domain under investigation. When we get enough data, we will switch to the empirical distribution, which converges to the true distribution, almost surely. In the limit, when we have infinite data, we will be using the correct distribution as the hypothesis about the data. We

[4]Assuming non-zero probabilities for all symbols.

[5]Noting that the tree having all codelengths equal to two contributes nothing to actual compression, we can also look at this such that we only use the other possible tree and use one bit to designate whether we are coding with Huffman or not. The end result is the same though, 6 bits.

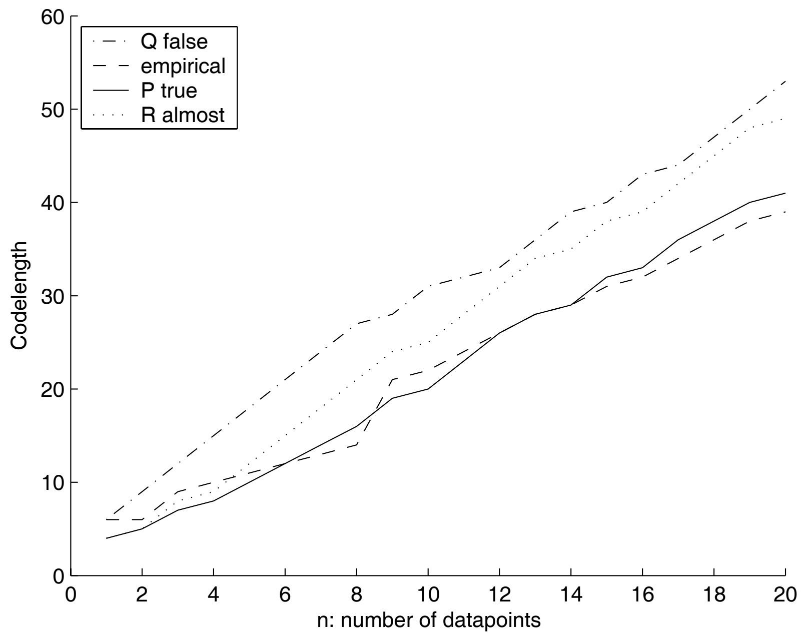

Figure 5.2: Codelength for a source P according to different models/hypotheses; single simulated run. Lower is better.

note that in the case shown in Fig. 5.2, it holds for $n \leq 4$ that the empirical distribution is not worse then "R almost". For $n > 4$ the empirical is better.

The KL-divergence is a measure of approximation error. How long does it take for an estimate of a distribution to become good? We ran 100 simulations on this, as shown in Figure 5.3. There we show the KL-divergence between a simple frequency estimate of the probabilities and the true probabilities which we happen to know for this example. We notice that the estimate becomes good relatively quickly with not too wide error bounds[6]. On average, we only need 0.1 extra bits after about 20 observations and after about 100 less then 0.025 bits. In this situation the effective extra cost of using an empirical distribution rather then the true one converges fast to zero, since the encoding method is tolerant of minor variations in the underlying probability distribution. The one standard deviation error bounds have the interpretation that we expect about two-thirds of the examples to fall within this range; this is because of the central limit theorem.

From this example we conclude that our method finds the correct distribution fast when it is available, recovers via the empirical distribution (the hypothesis learned from the data) rather quickly from being handed a good bias, such as R, and almost immediately when handed a bad bias, as Q. It prefers good biases (R) over bad ones (Q) and converges almost surely to the

[6]In the beginning of each simulated run, not all symbols in the alphabet have appeared in the data and the empirical distribution in this example only estimates the probabilities of the symbols as they appear. In a proper implemetation allowance should be made for unseen symbols.

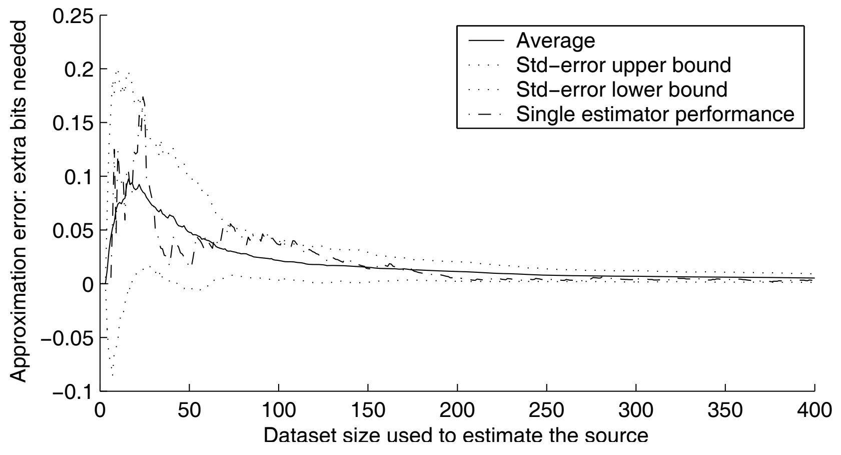

Figure 5.3: KL-divergence between a distribution and its estimated distribution, with one standard deviation error bounds, based on 100 simulated runs.

truth. Our method shows promise.

5.2 Decision Trees and Real World Data

Now we try instantiating the framework in a considerably more complex situation then considered in the last section. We build on the material from the previous chapter, using one type of learner and a simple M function (see Figure 2.2), as described later in this chapter. First we describe shortly the implementation and how we prepared the data. Then we report on the data and results of the experiments performed and conclude with some remarks on if we have accomplished anything by these experiments.

5.2.1 Implementation

A stitch work of tools was used to run experiments. Decision trees were built using Weka [WF05][7]. Weka is open source so we were able to make the necessary changes for our framework, namely to limit the depth of the generated trees and add functions to obtain codelengths. To enrich the environment, which we use to induce transfer, a restriction was applied to the building of trees such that the depth of the trees was limited; this corresponds to the *chop* function in Algorithm 1. In practice this entailed building a version of Weka for each depth-restriction and running experiments with this build of the software. These restricted-depth builds were then run on different sized datasets w.r.t. the domains constituting an environment. This resulted in quite a few experimental results as detailed in the tables in Appendix A.1. These results were written into text-files and pre-processed for further analysis using shell-scripts. The analysis was then done in Matlab[8], with some large-number features added by

[7]`http://www.cs.waikato.ac.nz/ml/weka/`
[8]`http://www.mathworks.com/`

stitching some Java-functions capable of large-number processing to it. Finally, the graphs were generated using Matlab also.

For a full scale implementation a consistent environment would need to be prepared, for instance by adding features to Weka, and a robust database back-end would be necessary. The number of dimensions in the problem is such that it is difficult to imagine larger scale experiments without those.

The effort invested in doing a careful large scale implementation would not go unrewarded. There would be no need to adjust the learning procedure or structures already in place in such a system when faced with a new problem, hypothesis set or learning algorithm: the system always returns a minimum description length hypothesis. When extending such a system with a e.g. a new learning algorithm (L in Figure 2.2) then the framework only needs to know how to encode the knowledge representation of the algorithm to apply it. Thus, problem solving skills can be grown incrementally step in this framework.

5.2.2 Data and Pre-Processing

Unfortunately, appropriate datasets are difficult to come by, especially since the application domains envisioned are usually very concerned with privacy.

For our experiments we used the Adult dataset from the UCI Machine Learning Repository, see [DNM98]. The task for this dataset it to predict whether a person makes over USD 50K per year. This prediction can be based on 14 features, detailed in Table 5.2. In this table we see that the 'native-country' feature has quite a few possible values, so the information gain ratio is more suitable for this dataset then the information gain measure as the criterion for splitting, which is the one used by J48. Also, there are six continuous features.

To make the dataset amenable to our experiments we split the data into different "domains" based on the *occupation* feature. This gives a collection of 14 datasets, which can then be considered as domains of an environment. One of the datasets arising from this feature-based split is designated as the one being used for learning (the *new* data) and the other 13 constitute an environment from which we may transfer hypotheses. We note that the Priv-house-serv domain only contains 3 instances of income greater then USD 50K out of a total 232, so the J48 tree is trivial and we omit this domain from further calculations in this paper. The Adult dataset was found to be suitable for conducting experiments, since it is readily available on-line, has features that could be interpreted as arising from different domains and has sufficient data to allow such splitting. We feel that this gives an adequate dataset for trying out our framework, although it leaves something to be desired in its realism.

Statistics on the split-dataset are given in Table 5.3. For the whole dataset the frequencies are 24.78% with income higher then 50k,75.22% with less.

age	continuous
workclass	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked
fnlwgt	continuous
education	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool
education-num	continuous
marital-status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse
occupation	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces
relationship	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried
race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black
sex	Female, Male
capital-gain	continuous
capital-loss	continuous
hours-per-week	continuous
native-country	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands

Table 5.2: Features and their values for the Adult dataset

Domain nr.	Occupation (domain)	Nr. of datapoints
1	Adm-clerical	5540
2	Armed-Forces	14
3	Craft-repair	6020
4	Exec-managerial	5984
5	Farming-fishing	1480
6	Handlers-cleaners	2046
7	Machine-op-inspct	2970
8	Other-service	4808
x	Priv-house-serv	232
9	Prof-specialty	6008
10	Protective-serv	976
11	Sales	5408
12	Tech-support	1420
13	Transport-moving	2316

Table 5.3: The domains and number of datapoints used for experiments

5.2.3 Experimental Results

Having prepared our data, we can apply the implementation. We are looking to see transfer taking place, which would be reflected in an adopting a hypothesis from another domain then the dataset under evaluation. If we see transfer take place at all, we have "empirical evidence" by the construction of our domains that the method does detect similarities between domains and induce transfer. That is, if transfer takes place, our concept is of some use, or, it works because it works[9].

Before taking a look at the numerical results, we give a few examples of the elements of $\mathcal{M}_{\mathcal{E}}$. Recall that transfer in our setting involves selecting a single decision tree from a collection of decision trees $\mathcal{M}_{\mathcal{E}}$. In Figures 5.4, 5.5 and 5.6 all the trees of depth 1 obtained during the experiments are shown. As can be seen they tend to be rather similar, frequently choosing capital gain as the most informative feature. The split of the data given in the tree in Figure 5.6 might appear to be irrelevant, but further construction with this tree reveals that this split is useful (too big to be printed). So, how do we go about getting results?

As described above, we generate trees of different depths and evaluate the transferability of the domains to the 'Exec-Man' domain by calculating the two-part codelength using different sizes of this set. In this way we obtain the codelengths of the decision trees, for each domain and depth of tree, as detailed in Table 5.6. In order to see how transferability behaves with different sizes of the dataset we also obtained a series of tables giving the codelength of the data w.r.t. each of these trees, as detailed in the tables in Appendix A.1. The expected behavior is confirmed: the tendency is to select more detailed knowledge for transfer from the available domains and the probability of selecting the empirical distribution, the distribution constructed form the new task, from the task under consideration increases as the dataset grows, which was one of the things the algorithm was designed to capture. As mentioned, reporting and visualization is a bit tricky, so only a snapshot of the relevant numbers is presented.

Now for some more detailed results. We'll discuss the unweighted codelength measure first and address the adjusted weighted codelength measure, where hypotheses get weight in the codelength according to their domain-origin and frequency, a little later in this section. First, there is the codelength of the trees of varying depth from different domains. These numbers are given in Table 5.6 and plotted in Figure 5.7. In Figure 5.7 we see that the codelength grows more or less linearly in the depth of the tree[10]. Constant codelength between depths in this table means the available data in that domain doesn't warrant more complex hypotheses. The variability of the codelength of the hypotheses of the same depth is because more labels for a feature mean larger codelength, since we need to code each and every leaf with a bit indicating that it is a leaf. The codelength of the hypotheses gives an ordering of the complexity of the hypotheses under consideration. Figure 5.7 gives an indication of nature of the complexities of the hypotheses of the models from the domains under consideration.

[9]The proof of the pudding is in the eating.

[10]Incidentally, naïve application of linear regression over the dataset, without repeated datapoints, gives a relationship of the kind: codelength = -117 + 61* treeDepth, see Figure A.1 in appendix. This tendency is explained by the role played by the logarithm in the codelength functions.

We note that the data is exhausted for all domains, since we see that relaxing the restriction of depth to 14 does not increase codelength, i.e. that the largest tree the algorithm is willing to construct has already been constructed[11]. This means that the underlying decision tree construction algorithm is of the opinion that there is no more knowledge to be extracted from that data.

To continue collecting results for analysis, after having obtained the codelength for the trees themselves, we add the codelength for the data given the trees for each tree. Then, we select the minimum over all domains and depths for and transfer that tree to our current task as the most useful hypothesis available given the data. If our method is to be considered to be of any use, we should select hypotheses different from the ones selected by the base learner, preferably from a different domain on some occasions[12]. This is indeed the case, as shown in Table 5.5.

Examples of codelengths are given in Table 5.7 and plotted in Figure 5.8 for the situation where there are 2000 examples in the dataset. The transferred hypothesis in this case comes from domain nr. 11 (Sales) and has depth 5. Transfer from another domain then the one under consideration is taking place in this case. The codelength for this domain is plotted in Figure 5.9 to show how the total codelength behaves. This sort of a *convex-ish* graph is what we like to see when applying MDL. It shows that the optimal tradeoff between the hypothesis complexity and the complexity of the data w.r.t. the hypothesis is obtained by balancing those two, not taking either to the extreme. As concerns transfer, this figure shows that the hypothesis adopted for transfer is different from the one constructed by looking solely at the available Exec-Man data. Transfer does take place in this setting so we have an indication that the method works.

Now for something slightly different. Lets imagine that we have a datasource giving us more and more data and we want to observe how our framework transfers knowledge from the other domains. To start with, we have a dataset, so we have little “confidence” in adopting knowledge to the new task, so we would expect smaller trees to be selected. Also, we would expect progressively larger trees to be selected as the dataset grows. This is indeed the case, as seen in Table 5.4. Underlying tables are given in detail in Appendix A.1.

Table 5.4 summarizes which hypotheses are selected for each size of the dataset used to induce transfer, both for the normal codelength and the adjusted codelength. The empirical distribution is never selected for the data considered here. The adjusted codelength measure discounts the more frequent hypothesis in $\mathcal{M}_{\mathcal{E}}$ and accounts for the tree constructed by using the dataset alone, adjusted to favor the hypotheses arising from the given environment[13] over the ones arising from the given dataset. This table also includes a column of results if a “self referring” hypothesis is included, i.e. the complete dataset of Exec-man is treated as the other domains. These results are interesting because these hypotheses reflect our best guess at the structure of the dataset under investigation. The last column in Table 5.4 gives the total codelength when applying J48 to the dataset by itself without using a hypothesis for putting the

[11] Although this occurrence of the same codelength does not occur more then once for one of the domains, the data is exhausted for that one too.

[12] This is not a criteria for the usefulness of the method, it would just be nice to see that sort of a behaviour, since it is to be expected in practice.

[13] Recall that this is required for convergence reasons, see chapter 3.

other numbers into some context.

We notice in Table 5.4 that when the self-referring hypothesis is included (hypothesis built by using the whole data of Exec-Man), the unweighted codelength settles on that domain when the dataset grows. That is, the method selects hypotheses of different depth from the domain that gives the best estimate of the underlying distribution. This is good news. If the framework would not settle on the domain that data is actually from, we would be faced with rather pathological behaviour in a simple setting, although the method may refrain from adopting as comprehensive a hypothesis as if only having the data.

However, when the self-referring hypothesis is not included, as in the second set of results in Table 5.4, the hypothesis selected for transfer is from other domains, and the size of the hypotheses selected then is quite a bit smaller then when the self referring one is included, indicating a worse fit then with the self referring one. The method still selects the same domain for transfer over dataset-size ranges, so it shows some stability in its preference of models.

When we come to the weighted measure, as in the third set of results in Table 5.4, we notice that the tendency is to select larger hypotheses and not to concentrate on one particular hypothesis as is done in the unweighted case. Which do we prefer? One feels that the adjusted measure behaves most like one would prefer. The reason is that it makes the most allowances for the environment; giving knowledge from the environment more chance of being adopted for transfer, based on its relevance to the current task as estimated by the data, which is the whole point of the exercise.

Next, we have Table 5.5, comparing the hypotheses selected for different dataset sizes against the hypotheses constructed by using the dataset alone. Here we note that the complexity of the models selected by our method is increasingly larger compared to that constructed by just using the data. We also note that the ratio between the total codelength of the transferred hypothesis to the empirical one is getting closer to 1, at which point the method will start selecting the empirical hypothesis. Apparently not very much more data is needed to start selecting the empirical model.

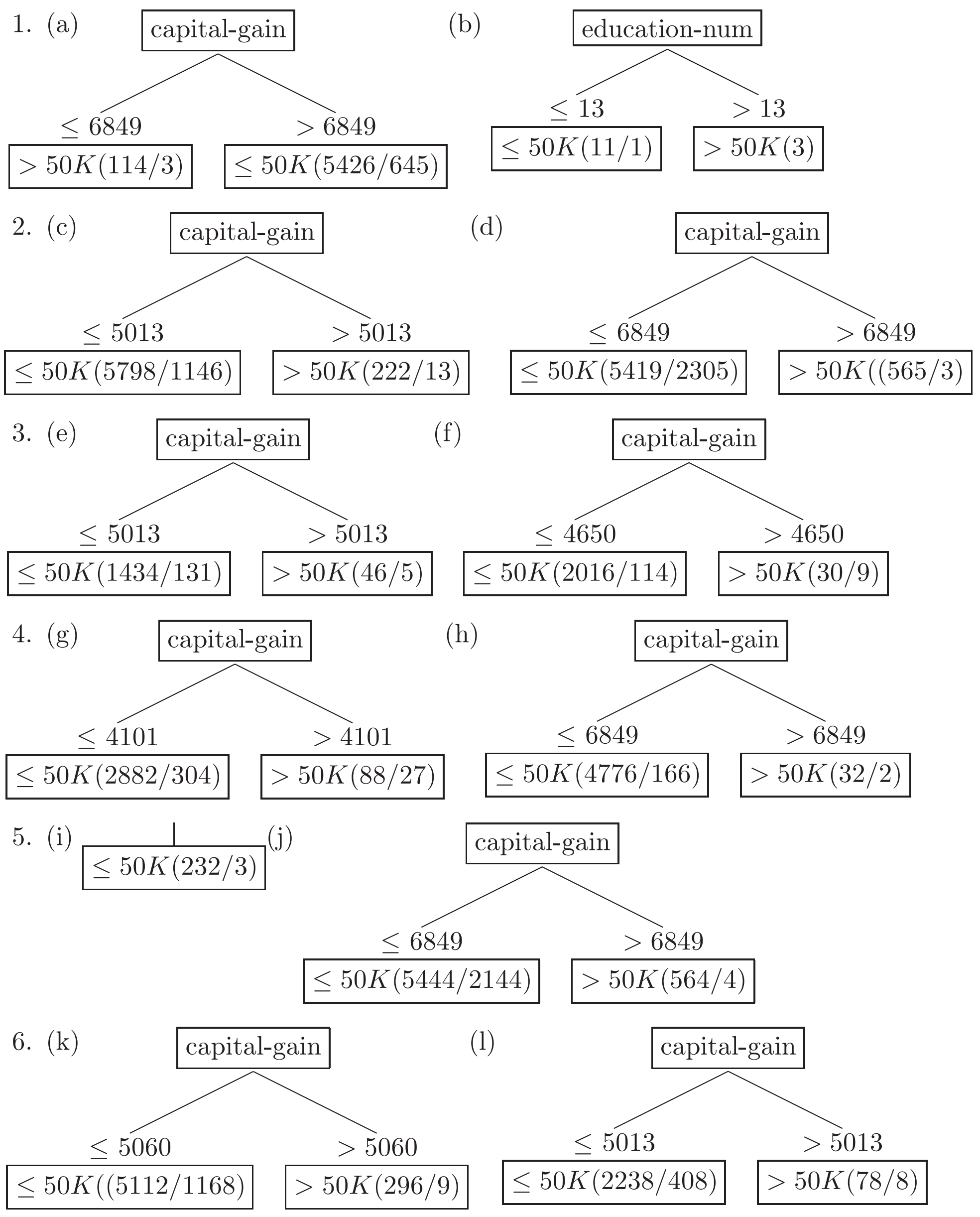

Figure 5.4: Trees produced of depth 1: (a) Adm-clerical (b) Armed-Forces (c) Craft-repair (d) Exec-managerial (e) Farming-fishing (f) Handlers-cleaners (g) Machine-op-inspct (h)Other-service (i)Priv-house-serv (j)Prof-specialty (k) Sales (l) Transport-moving

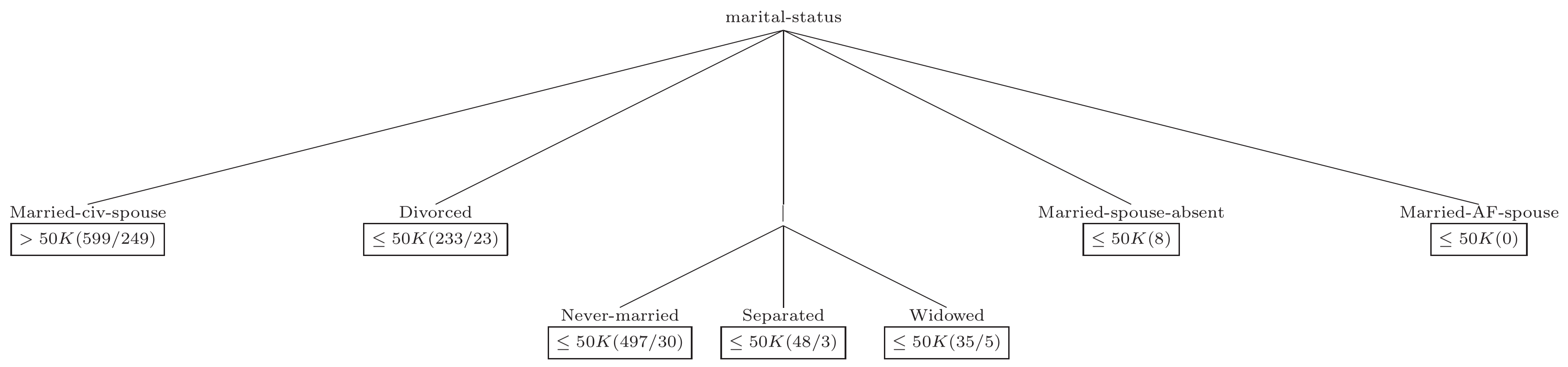

Figure 5.5: Tree of depth 1 for Protective-serv

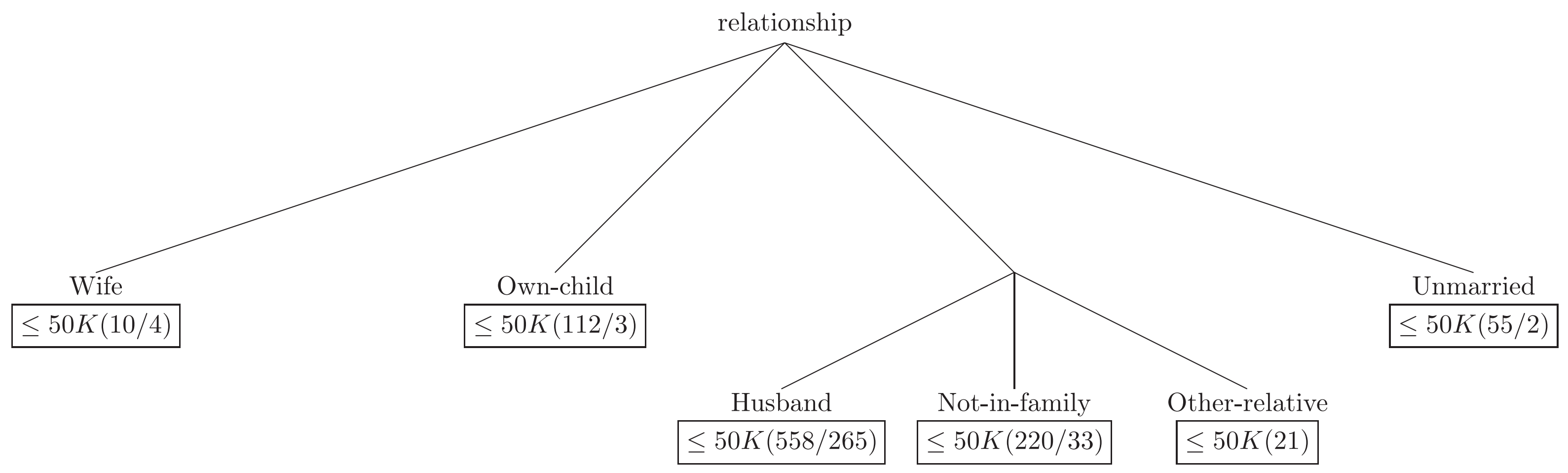

Figure 5.6: Tree of depth 1 for Tech-support

	Selected Hypotheses									
	Self-ref inc.			**W/o Self-ref**			**Weighted**			
$\lvert S_{k+1}\rvert$	D-Nr.	Depth	L_C^h	D-Nr.	Depth	L_C^h	D-Nr.	Depth	L_C^h	$L_C(S_{k+1})$
20	1	1	32	1	1	32	1	1	26.5	32
50	13	1	61	12	12	61	12	1	51.5	116
100	13	1	102	12	1	102	12	1	92.5	199
200	13	1	183	12	1	183	12	1	173.5	278
500	13	1	441	12	1	441	9	2	426	740
1000	4	3	827	9	2	835	9	2	819	987
2000	4	3	1614	9	2	1648	11	5	1628.5	1915
5984	4	6	4639	9	2	4832	9	8	4761	5358

Table 5.4: Domain selected for different dataset sizes from Exec-man including the self-referring one (hypothesis built by using the whole data of Exec-Man, nr. 4). The first column ($|S_{k+1}|$) is the size of the subset of the Exec-Man dataset used for the evaluation. Each of the next three sets of results has a column for the domain number (D-Nr.) the maximum depth of the selected tree (Depth) and the codelength of the selected model (L_C). The first set of model selection results ("Self-ref inc.") shows the results of the model selection when we have the self-referring model (hypothesis built by using the whole data of Exec-Man, model number 4) included. The second set ("W/o Self-ref") does not have model nr. 4 available. The third set of results ("Weighted") has the self-ref model available for selection, but here we have the weighted codelength, where we apply different weights to the models, depending on origin and frequency in the environment. The final column has the codelength of the dataset by itself without any model, for comparison purposes.

Remarks For every algorithm there is a dataset that the method performs well on. Performance results are always to be taken lightly. Some authors go even further, such as [DGL97]: "*Simulations on particular examples should never be used to compare classifiers.*" In line with those standpoints our experiments are more to experiment with the implementation itself then to get results.

Never the less, our method seems to be able to do something useful. It induces transfer as shown in Table 5.3 and is successful in that sense. This is however just a small sample of the experiments to be done to fully evaluate this way of doing transfer learning; to see if it is capable if doing something useful. The sampling methodology is naïve at best and out of training sample performance is not quantified. Again, there are considerable implementational difficulties, which seem for the experiments to require a proper database and considerable computational power for a full evaluation. Further, it is uncertain how to properly test such an framework on-line.

In the beginning of this chapter we stated that in order for our method so show any promise, we would at least need to see behaviour different from the situation when there is no possibility of transferring any knowledge. Do our experiments show this behaviour? The adjusted codelength measure exhibits exactly this sort of behaviour and certainly feels appropriate.

The method defaults to the model generated by the dataset available in the absence of further evidence to support transfer so we avoid pathological

	Transferred				Empirical		
n	D-nr.	Depth	$L(h)$	$L_C^{wh}(S^n)$	$L(h)$	${}_E L_C^{wh}(S^n)$	$L_C^{wh}(S^n)/{}_E L_C^{wh}(S^n)$
20	1	1	21	27	11	32	0.8281
50	12	1	52	52	71	116	0.4440
100	12	1	100	93	110	199	0.4648
200	12	1	199	174	109	278	0.6241
500	9	2	410	426	326	740	0.5757
1000	9	2	803	819	226	987	0.8298
2000	11	5	1557	1629	408	1915	0.8504
5984	9	8	4482	4761	905	5358	0.8886

Table 5.5: Comparison of hypotheses selected with environment knowledge available and without. The first column contains the number of examples in the dataset used (n). Then we have the transferred results, D-nr is the domain number, L(h) is the unweighted hypothesis codelength and $L_C^{wh}(S^n)$ is the weigthed two part codelength. Then we have the corresponding numbers based on only the data from the dataset with n observations, and unweighted two part code length ($w = 1$). Finally we have the ratio between the two total codelengths.

behaviour[14]. Does it perform better then other learners? There is nothing in our experiments that allows us to state that. However, algorithms and data to compare our framework are not widely available and the experiments do show that the framework is flexible enough to learn across domains. MDL can be used to induce transfer in our framework.

[14]As with all statistical methods, it is possible to get "unlucky" and have a sample that gives rise to the wrong behaviour, but that's the nature of the game.

		Depth													
	Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Adm-clerical	11	30	42	62	85	96	130	190	247	294	322	322	322	322
2	Armed-Forces	11	11	11	11	11	11	11	11	11	11	11	11	11	11
3	Craft-repair	11	51	86	107	128	150	161	172	213	279	347	433	537	577
4	Exec-managerial	11	22	43	96	175	300	408	541	669	789	850	897	905	905
5	Farming-fishing	11	22	33	44	55	55	55	55	55	55	55	55	55	55
6	Handlers-cleaners	11	22	22	22	22	22	22	22	22	22	22	22	22	22
7	Machine-op-inspct	11	33	64	74	86	86	86	86	86	86	86	86	86	86
8	Other-service	11	22	22	22	22	22	22	22	22	22	22	22	22	22
9	Prof-specialty	11	32	76	132	193	324	429	558	711	847	905	996	1080	1080
10	Protective-serv	15	44	69	115	135	156	191	221	241	241	241	241	241	241
11	Sales	11	33	64	93	115	148	200	271	395	526	576	608	630	630
12	Tech-support	19	76	132	164	176	198	209	219	219	219	219	219	219	219
13	Transport-moving	11	42	63	85	96	96	96	96	96	96	96	96	96	96

Table 5.6: Codelength for hypotheses of different depth; no data.

	Depth													
Domain nr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1940	1949	1876	1898	1902	1680	1696	1752	1759	1771	1790	1790	1790	1790
2	1962	1962	1962	1962	1962	1962	1962	1962	1962	1962	1962	1962	1962	1962
3	1937	1959	1895	1918	1928	1931	1937	1683	1718	1843	1862	1906	1960	1950
4	1940	1945	1593	**1619**	1595	1599	1627	1668	1715	1745	1761	1783	1787	1787
5	1937	1942	1869	1891	1880	1880	1880	1880	1880	1880	1880	1880	1880	1880
6	1942	1942	1942	1942	1942	1942	1942	1942	1942	1942	1942	1942	1942	1942
7	1943	1951	1900	1903	1904	1904	1904	1904	1904	1904	1904	1904	1904	1904
8	1940	1945	1945	1945	1945	1945	1945	1945	1945	1945	1945	1945	1945	1945
9	1940	1632	1659	1646	1654	1742	1748	1782	1859	1925	1947	1995	2063	2063
10	2012	1969	1940	1958	1969	1983	2000	2016	2026	2026	2026	2026	2026	2026
11	1937	1948	1883	1898	**1629**	1645	1724	1741	1825	1889	1903	1912	1922	1922
12	1691	1690	1706	1703	1709	1649	1655	1659	1659	1659	1659	1659	1659	1659
13	1937	1875	1903	1896	1901	1901	1901	1901	1901	1901	1901	1901	1901	1901

Table 5.7: The “Domain nr.” is related to the domain in Table 5.3. Total codelength for hypotheses and data w.r.t. hypotheses of different depth, for the Exec-Man dataset with 2000 instances. The “self-ref” is hypothesis included (hypothesis built by using the whole data of Exec-Man, nr. 4). The lowest codelengths are in bold-face, one with the self-ref hypothesis included (model nr. 4) and the other without the self-ref hypothesis. There is only a small difference between these two, so the method does not strongly prefer the self-ref one to the other. Notice that model (11,5) it not much larger then the other two.

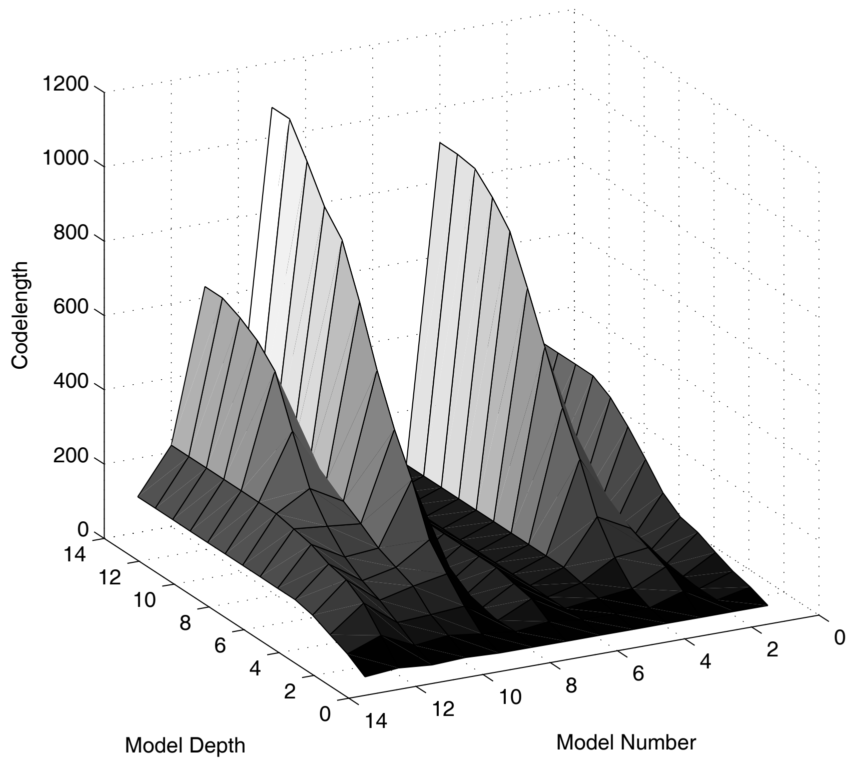

Figure 5.7: Codelength for decision trees of different depths; no data.

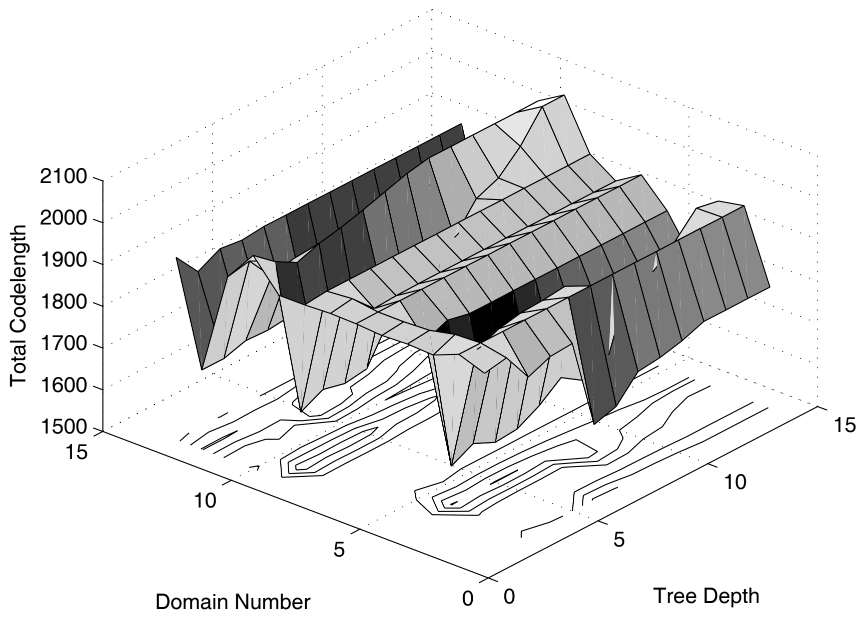

Figure 5.8: Total weighted code length for different depths and domains for dataset with 2000 instances. The domain selected is plotted in Fig. 5.9

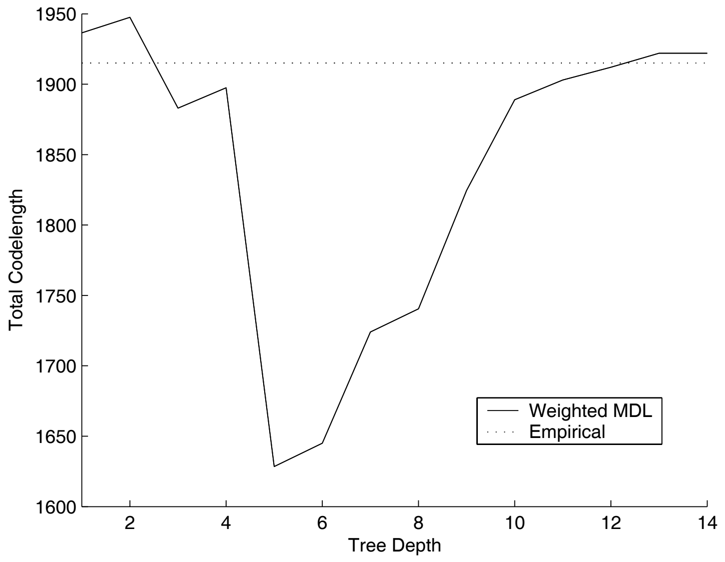

Figure 5.9: Total weighted code length for different depths for the selected domain (11: for a dataset with 2000 instances. The empirical distribution is constant for this dataset and represented by a dotted line. Note a local minima at depth 3.

Chapter 6

Conclusions and Discussion

Transfer learning is about how learning from one domain or a collection of domains can be applied to another. It is learning from similarities and parallels. It is learning from experience. This paper is about a distribution free, data driven, extendable framework for transfer learning, based on the minimum description length principle. We define transfer learning in terms of a specific framework, where we have a collection of hypothesis from other application domains available, a learning algorithm consistent over domains and a new, previously unseen learning task.

The given collection of domains, from which the hypotheses originate, we term an environment for the algorithm. The proposed method makes allowances for the specifics of the new task by considering the new task as a part of the environment. The applicability of the available hypotheses is evaluated w.r.t. the new task and the method proposed is biased towards adopting hypotheses from the environment in which it is embedded. How strongly we prefer hypotheses from the environment to the ones induced from the current task is subjective and left to the modeler as the framework stands. In this sense the framework induces an bias on the learner, based on the environment.

Our framework consists of 3 algorithms: a base learner, a hypothesis space digester/cognisizer that produces less specific hypothesis from the ones given by the environment and may combine them into more complex ones, and a MDL based evaluator of the available hypotheses w.r.t. the current task, and a learner.

We showed that, under the assumption that the learning problem is realizable and the underlying learner converges to the true distribution between the input-output spaces, our method converges to the true distribution as more data becomes available for the new task. This is the guaranteed worst case performance of our framework. By the general motivation for transfer learning the hope is that the environment is rich enough to contain solutions to our tasks that give better small sample performance.

We tried our framework in two situations on data. The first was a very simple simulation based on a distribution on four symbols. This illustrated how our framework prefers “truth” to empirically constructed hypothesis and hypothesis that did not generate the data. When the “truth” was not available in the environment, our framework prefers near truth, as quantified by KL-divergence, to more far fetched explanations for the data and eventually accepting the empirically constructed hypothesis as the true one. The empirical hypothesis converged to the true distribution almost surely in this case.

Then we gave an example of how our proposed framework might be implemented using a decision tree algorithm, a simplistic hypothesis space digester, which produced trees of varying restricted depth from a given tree, and a encoding scheme for decision trees and exceptions from the tree, giving the two part MDL codelength. This implementation we tried on artificially segmented data from the real world, giving encouraging results, albeit not spectacular. The overall impression from experiments is that this sort of a modified MDL inference principle can certainly be used to induce transfer between domains.

Several issues have not been addressed. We have not quantified the rate of convergence and under what conditions assurances can be given on the rate of convergence. Baxter presents a framework in [Bax00] where bounds are given for performance on *any unseen* task from a environment, and data is available from all domains. This is different from our framework, where we are concerned with *previously unseen* tasks, i.e. we know what task we are dealing with at the moment and only have hypothesis available but not the data. Baxter works in the PAC framework, which is very different from the MDL approach to learning.

We have not given a general quantification of the hyper-prior, or the distribution of the domains which we use to construct an environment. This is for a good reason. In our setting that would entail constructing a coding scheme of a model for the hypotheses, which is far from obvious how to do and can only be addressed case by case. Moreover, our method is essential distribution free, so specifying the distribution would undermine that strength of the method. However, specifying a distribution allows for quantification of how the domains in an environment are *related*, and an approach based on hierarchical Bayes seems to be the most fruitful approach.

We have not addressed the issue of confidence of the predictions, i.e. how much probability we believe there to be of making a false prediction of a class of an object. This should be addressed for real world applications. A promising way of assessing the confidence of the predictions is given in [GV02], which assigns two measures to each individual prediction called *confidence* and *credibility*. It has the additional benefit of wrapping around the learning algorithm.

A strength of the MDL approach to inference is that it induces a single hypothesis from the hypothesis space. In our framework this means in principle that we do not need to worry about the hypothesis space being full of junk, or irrelevant information for the task at hand: the data will simply show the irrelevant information to be of no use and either discard all knowledge in the hypothesis space or adopt the most suitable one.

A difficult part of the proposed framework is to make a good hypothesis space digester or cognisizer, which produces more useful or usable hypotheses from the environment. Specifically, as the framework is presented in the paper, the digester would need to combine compatible hypothesis into larger ones to get a more comprehensive or complex hypotheses. There is a way around this complicated, ill describable combining step. We can consider the available hypotheses as features and emulate the way decision tree construction algorithms work, without being restricted to decision tree learners. This would be done in the following way: First, construct as many partial hypothesis as the environment warrants, limited by the data itself. Then, for the given dataset and learner, apply the framework, selecting the best hypothesis according to the two part codelength. While there is enough misclassified data according to the selected hypotheses, reapply the framework. In this way, we would recursively filter out all the regularity in the data as warranted by the hypothesis space and data. This looks like a interesting approach to try to implement and closely resembles decision tree construction. Also, this recursive application of the framework is slightly reminiscent of ensemble methods, in that it concentrates on the difficult areas of classification. We illustrated how this might be carried out with decision trees, but did not apply this method to data.

In conclusion, we feel that our framework is usable for collecting and using experience in an efficient way.

Appendix A

In these appendices we produce various material not suited for inclusion in the main text.

A.1 Experimental Result Tables

Here we reproduce some of the tables generated by our algorithm for the Adult dataset in more detail then in the text, complementing the results from 5.3. These tables are the unweighted total codelengths obtained by running our algorithm. In all cases the self-referring hypothesis, the hypothesis built when using the whole of the Exec-Man dataset, is included and the lowest codelength is highlighted for the cases when the self-referring hypothesis is included and not.

	Depth													
Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	**32**	51	63	83	106	117	151	211	268	315	343	343	343	343
2	33	33	33	33	33	33	33	33	33	33	33	33	33	33
3	32	72	107	128	149	171	182	193	234	299	367	453	555	595
4	32	43	61	114	195	318	426	561	689	810	871	918	926	926
5	32	43	54	65	76	76	76	76	76	76	76	76	76	76
6	32	43	43	43	43	43	43	43	43	43	43	43	43	43
7	32	54	85	95	107	107	107	107	107	107	107	107	107	107
8	32	43	43	43	43	43	43	43	43	43	43	43	43	43
9	32	52	96	150	210	342	447	578	729	867	923	1014	1098	1098
10	37	65	90	136	156	177	212	242	262	262	262	262	262	262
11	32	54	85	114	136	169	221	291	416	547	596	628	650	650
12	39	96	152	182	194	218	229	239	239	239	239	239	239	239
13	32	63	84	106	117	117	117	117	117	117	117	117	117	117

Table A.1: Total codelength for hypothesis and data w.r.t. hypotheses of different depth, for dataset with 20 instances; self-ref hypothesis included.

	Depth													
Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	63	82	94	114	137	146	179	240	296	343	371	371	371	371
2	63	63	63	63	63	63	63	63	63	63	63	63	63	63
3	63	103	137	158	179	201	212	221	262	329	397	483	586	626
4	63	74	84	137	217	337	445	578	706	831	892	939	947	947
5	63	74	84	95	106	106	106	106	106	106	106	106	106	106
6	63	74	74	74	74	74	74	74	74	74	74	74	74	74
7	63	85	115	125	137	137	137	137	137	137	137	137	137	137
8	63	74	74	74	74	74	74	74	74	74	74	74	74	74
9	63	73	117	171	234	363	468	595	746	884	942	1033	1119	1119
10	67	96	119	165	185	207	242	272	292	292	292	292	292	292
11	63	85	115	144	163	195	248	316	442	573	623	655	677	677
12	**61**	117	173	206	218	240	251	261	261	261	261	261	261	261
13	63	93	114	136	147	147	147	147	147	147	147	147	147	147

Table A.2: Total codelength for hypothesis and data w.r.t. hypotheses of different depth, for dataset with 50 instances; self-ref hypothesis included.

	Depth													
Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	112	131	141	161	184	187	219	281	337	384	412	412	412	412
2	114	114	114	114	114	114	114	114	114	114	114	114	114	114
3	111	151	184	205	226	248	259	259	302	369	437	523	626	666
4	112	123	122	175	246	369	477	610	738	862	923	972	980	980
5	111	122	131	142	153	153	153	153	153	153	153	153	153	153
6	112	122	122	122	122	122	122	122	122	122	122	122	122	122
7	112	134	163	172	184	184	184	184	184	184	184	184	184	184
8	112	123	123	123	123	123	123	123	123	123	123	123	123	123
9	112	111	157	211	276	403	504	629	780	916	974	1065	1155	1155
10	118	145	165	211	231	254	289	319	339	339	339	339	339	339
11	111	133	162	191	201	232	286	354	481	612	662	694	716	716
12	**102**	159	211	247	259	275	286	296	296	296	296	296	296	296
13	111	140	161	183	194	194	194	194	194	194	194	194	194	194

Table A.3: Total codelength for hypothesis and data w.r.t. hypotheses of different depth, for dataset with 100 instances; self-ref hypothesis included.

	Depth													
Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	210	229	236	256	279	275	302	371	424	471	499	499	499	499
2	214	214	214	214	214	214	214	214	214	214	214	214	214	214
3	210	250	280	301	322	344	355	345	388	461	528	614	716	752
4	210	221	202	255	331	444	546	681	807	924	981	1037	1045	1045
5	210	221	226	238	248	248	248	248	248	248	248	248	248	248
6	210	221	221	221	221	221	221	221	221	221	221	221	221	221
7	211	232	258	267	279	279	279	279	279	279	279	279	279	279
8	210	221	221	221	221	221	221	221	221	221	221	221	221	221
9	210	193	239	291	356	485	585	712	865	999	1057	1146	1237	1237
10	218	243	258	304	324	347	382	412	432	432	432	432	432	432
11	210	232	256	285	284	315	367	428	561	692	742	774	796	7ř96
12	**183**	242	293	328	340	344	355	365	365	365	365	365	365	365
13	210	234	256	277	288	288	288	288	288	288	288	288	288	288

Table A.4: Total codelength for hypothesis and data w.r.t. hypotheses of different depth, for dataset with 200 instances; self-ref hypothesis included.

	Depth													
Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	501	520	511	533	555	515	546	617	670	717	745	745	745	745
2	509	509	509	509	509	509	509	509	509	509	509	509	509	509
3	500	540	553	576	597	618	629	582	624	717	783	869	970	1002
4	501	512	**446**	499	564	671	767	900	1026	1140	1192	1243	1251	1251
5	500	511	501	515	522	522	522	522	522	522	522	522	522	522
6	501	511	511	511	511	511	511	511	511	511	511	511	511	511
7	502	523	535	543	554	554	554	554	554	554	554	554	554	554
8	501	512	512	512	512	512	512	512	512	512	512	512	512	512
9	501	442	489	527	603	725	825	938	1090	1233	1289	1378	1475	1475
10	519	537	550	594	614	638	673	703	723	723	723	723	723	723
11	500	522	529	558	521	549	608	669	801	936	984	1016	1038	1038
12	**441**	493	545	587	599	598	609	619	619	619	619	619	619	619
13	500	508	533	551	562	562	562	562	562	562	562	562	562	562

Table A.5: Total codelength for hypothesis and data w.r.t. hypotheses of different depth, for dataset with 500 instances; self-ref hypothesis included.

	Depth													
Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	983	1002	973	999	1018	923	938	1029	1072	1116	1145	1145	1145	1145
2	988	988	988	988	988	988	988	988	988	988	988	988	988	988
3	982	1023	1016	1044	1065	1083	1094	980	1032	1147	1210	1296	1395	1418
4	983	994	**827**	880	949	1026	1118	1230	1349	1451	1503	1552	1560	1560
5	982	993	964	984	986	986	986	986	986	986	986	986	986	986
6	984	993	993	993	993	993	993	993	993	993	993	993	993	993
7	985	1005	1003	1011	1021	1021	1021	1021	1021	1021	1021	1021	1021	1021
8	983	994	994	994	994	994	994	994	994	994	994	994	994	994
9	983	**835**	880	916	972	1110	1188	1308	1453	1593	1643	1732	1834	1834
10	1018	1018	1013	1053	1074	1100	1134	1165	1185	1185	1185	1185	1185	1185
11	982	1004	991	1020	909	937	1014	1062	1203	1340	1385	1411	1433	1433
12	852	895	944	978	990	961	974	984	984	984	984	984	984	984
13	982	973	1003	1016	1027	1027	1027	1027	1027	1027	1027	1027	1027	1027

Table A.6: Total codelength for hypothesis and data w.r.t. hypotheses of different depth, for dataset with 1000 instances; self-ref hypothesis included.

	Depth													
Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1945	1964	1897	1929	1944	1728	1761	1847	1882	1918	1951	1951	1951	1951
2	1967	1967	1967	1967	1967	1967	1967	1967	1967	1967	1967	1967	1967	1967
3	1942	1984	1938	1971	1992	2006	2017	1769	1824	1982	2035	2122	2228	2238
4	1945	1956	**1614**	1667	1682	1749	1831	1938	2049	2139	2186	2231	2239	2239
5	1942	1953	1885	1913	1907	1907	1907	1907	1907	1907	1907	1907	1907	1907
6	1947	1953	1953	1953	1953	1953	1953	1953	1953	1953	1953	1953	1953	1953
7	1948	1967	1932	1940	1947	1947	1947	1947	1947	1947	1947	1947	1947	1947
8	1945	1956	1956	1956	1956	1956	1956	1956	1956	1956	1956	1956	1956	1956
9	1945	**1648**	1697	1712	1750	1904	1962	2061	2214	2348	2399	2493	2603	2603
10	2019	1991	1974	2015	2036	2061	2095	2126	2146	2146	2146	2146	2146	2146
11	1942	1964	1915	1944	1686	1719	1824	1876	2022	2152	2191	2216	2237	2237
12	1700	1728	1772	1785	1797	1748	1759	1768	1768	1768	1768	1768	1768	1768
13	1942	1896	1934	1938	1949	1949	1949	1949	1949	1949	1949	1949	1949	1949

Table A.7: Total codelength for hypothesis and data w.r.t. hypotheses of different depth, for dataset with 2000 instances; self-ref hypothesis included.

	Depth													
Domain	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	5773	5792	5598	5642	5641	5021	5002	5150	5135	5172	5203	5203	5203	5203
2	5842	5842	5842	5842	5842	5842	5842	5842	5842	5842	5842	5842	5842	5842
3	5760	5808	5627	5673	5694	5701	5712	4978	5067	5377	5380	5473	5570	5537
4	5773	5784	4720	4773	4789	**4639**	4702	4724	4814	4838	4866	4884	4887	4887
5	5760	5771	5569	5620	5582	5582	5582	5582	5582	5582	5582	5582	5582	5582
6	5760	5771	5771	5771	5771	5771	5771	5771	5771	5771	5771	5771	5771	5771
7	5764	5795	5630	5639	5640	5640	5640	5640	5640	5640	5640	5640	5640	5640
8	5773	5784	5784	5784	5784	5784	5784	5784	5784	5784	5784	5784	5784	5784
9	5773	4832	4905	**4843**	4863	5018	4981	5040	5187	5333	5377	5482	5627	5627
10	5997	5845	5724	5764	5782	5818	5854	5886	5905	5905	5905	5905	5905	5905
11	5760	5782	5593	5622	4870	4886	5102	5124	5306	5404	5400	5419	5435	5435
12	5007	4959	4998	5004	5016	4866	4882	4882	4882	4882	4882	4882	4882	4882
13	5760	5587	5648	5623	5634	5634	5634	5634	5634	5634	5634	5634	5634	5634

Table A.8: Total codelength for hypothesis and data w.r.t. hypotheses of different depth, for dataset with 5984 instances; self-ref hypothesis included.

A.2 Relation between tree depth and codelength

Figure A.1 shows the relation between the codelength and the depth of the generated trees along with an regression line obtained by linear regression on the dataset from Table 5.7 with out repeated points. From the figure, there appears to be a linear relationship, or at least not a strong exponential explosion in the complexity of the hypotheses as the depth of the hypotheses is increased.

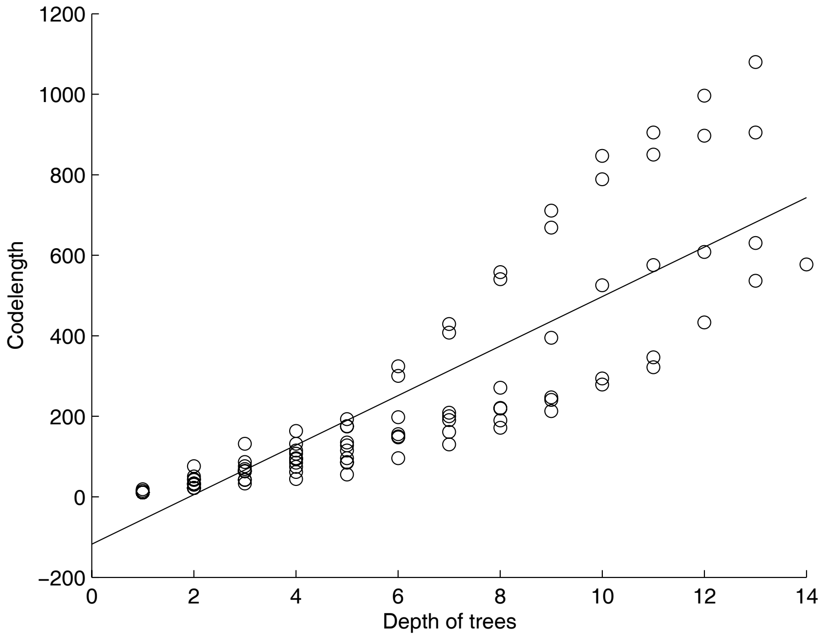

Figure A.1: Codelength versus tree depth for the Adult dataset.

Bibliography

[Bax00] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[BC91] Andrew R. Barron and Thomas M. Cover. Minimum complexity density estimation. *IEEE Transactions on Information Theory*, 37(4):1034–1054, 1991.

[BS94] J. Bernardo and A. Smith. *Bayesian Theory.* John Wiley and Sons, New York, 1994.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & sons, 1991.

[DGL97] L. Devroye, L. Gyorfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition.* Springer, 1997.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the *EM* algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[DNM98] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

[Grü98] Peter Grünwald. *The Minimum Description Length Principle and Reasoning under Uncertainty.* PhD thesis, Universiteit van Amsterdam, CWI, ILLC Dissertation Series 1998-03., 1998.

[Grü04] Peter Grünwald. A tutorial introduction to the minimum description length principle, 2004.

[GV02] Alex Gammerman and Volodya Vovk. Prediction algorithms and confidence measures based on algorithmic randomness theory. *Theor. Comput. Sci.*, 287(1):209–217, 2002.

[Has00] Robert Haskell. *Transfer of Learning.* Elsevier, 2000.

[HOGV04] José Hernández-Orallo and Ismael García-Varea. Explanatory and creative alternatives to the MDL priciple. *Foundations of Science, Springer Netherlands*, pages 185–207, 2004.

[HP05] Marcus Hutter and Jan Poland. Asymptotics of discrete MDL for online prediction, 2005.

[Hut01] Marcus Hutter. New error bounds for Solomonoff prediction. *Journal of Computer and System Science*, 62(4):653–667, June 2001. Submitted December 1999, revised December 2000.

[LV97] Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications, 2ed.* Springer, February 1997.

[Mit80] Tom M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers Computer Science Department, New Brunswick, New Jersey, 1980.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[MRA95] Manish Mehta, Jorma Rissanen, and Rakesh Agrawal. MDL-based decision tree pruning. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 216–221, 1995.

[QR89] J.R. Quinlan and R. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[RH06] Daniil Ryabko and Marcus Hutter. On sequence prediction for arbitrary measures. Technical Report 13-06, IDSIA, 2006.

[Ris89] Jorma Rissanen. Stochastic complexity in statistical inquiry. *World Scientific Series in Computer Science*, 15, 1989.

[Ser80] Robert J. Serfling. *Approximation Theorems of Mathematical Statistics*. John Wiley & Sons Inc., New York, 1980. Wiley Series in Probability and Mathematical Statistics.

[Sol64] Ray J. Solomonoff. A formal theory of inductive inference. part I and II. *Information and Control*, 7(2):1–22, 224–254, 1964.

[Sol78] Ray J. Solomonoff. Complexity-based induction systems: comparisons and convergence theorems. *IEEE Transactions on Information Theory*, 24:422–432, 1978.

[VD02] R. Vilalta and Y. Drissi. A perspective view and survey of metalearning, 2002.

[Vov98] V. Vovk. A game of prediction with expert advice. *Journal of Computer and System Sciences*, 56(2):153–173, 1998.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, June 2005.

[WM95] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM, 1995.

[WP93] C.S. Wallace and J.D. Patrick. Coding decision trees. *Machine Learning*, 11:7–22, 1993.

[Zha04] Tong Zhang. On the convergence of mdl density estimation. In John Shawe-Taylor and Yoram Singer, editors, *COLT*, volume 3120 of *Lecture Notes in Computer Science*, pages 315–330. Springer, 2004.