Luckiness and Regret in Minimum Description Length Inference

Steven de Rooij
University of Cambridge
steven@statslab.cam.ac.uk

Peter D. Grünwald
Centrum Wiskunde & Informatica
University of Leiden
peter.grunwald@cwi.nl

June 2, 2009

Abstract

Minimum Description Length (MDL) inference is based on the intuition that *understanding* the available data can be defined in terms of the ability to *compress* the data, i.e. to describe it in full using a shorter representation. This brief introduction discusses the design of the various codes used to implement MDL, focusing on the philosophically intriguing concepts of *luckiness* and *regret*: a good MDL code exhibits good performance in the worst case over all possible data sets, but achieves even better performance when the data turn out to be simple (although we suggest making no a priori assumptions to that effect). We then discuss how data compression relates to performance in various learning tasks, including parameter estimation, parametric and nonparametric model selection and sequential prediction of outcomes from an unknown source. Last, we briefly outline the history of MDL and its technical and philosophical relationship to other approaches to learning such as Bayesian, frequentist and prequential statistics.

1 Introduction

Suppose we have been observing some phenomenon for a while, and we now consider a number of alternative hypotheses to explain how the data came about. We want to use the data somehow to evaluate these hypotheses and decide which is the "best" one. In case that one of the hypotheses exactly describes the true mechanism that underlies the phenomenon, then that is the one we hope to find. While this may already be a hard problem, available hypotheses are often merely approximations in practice. In that case the goal is to select a hypothesis that is useful, in the sense that it provides insight in previous observations, and matches new observations well. Of course, we can immediately reject hypotheses that are inconsistent with new experimental data, but hypotheses often allow for some margin of error; as such they are never truly inconsistent but they can vary in the degree of success with which they predict new observations. A quantitative criterion is required to decide between competing hypotheses. The Minimum Description Length (MDL) principle is such a criterion [26, 33]. It is based on the intuition that, on the basis of a useful theory, it should be possible to *compress* the observations, i.e. to describe the data in full using fewer symbols than we would need using a literal description. According to the MDL principle, the more we can compress a given set of data, the more we have learned about it. The MDL approach to inference requires that all hypotheses are formally specified in the form of *codes*. A code is a function that maps possible outcomes to binary sequences; thus the length of the encoded representation of the data can be expressed in bits. We can encode the data by first specifying

the hypothesis to be used, and then specifying the data with the help of that hypothesis. Suppose that $L(H)$ is a function that specifies how many bits we need to identify a hypothesis H in a countable set of considered hypotheses $\mathcal{H}$. Furthermore let $L_H(D)$ denote how many bits we need to specify the data D using the code associated with hypothesis H. The MDL principle now tells us to select that hypothesis H for which the total description length of the data, i.e. the length of the description of the hypothesis $L(H)$, plus the length of the description of data using that hypothesis $L_H(D)$, is shortest. The hypothesis yielding the minimum total description length as a function of the data is denoted H_{mdl}, and the resulting code length is L_{mdl}:

$$L_{\text{mdl}}(D) \quad = \quad L(H_{\text{mdl}}) + L_{H_{\text{mdl}}}(D) \quad = \quad \min_{H \in \mathcal{H}} \Big(L(H) + L_H(D) \Big). \tag{1}$$

Intuitively, the term $L(H)$ represents the complexity of the hypothesis while $L_H(D)$ represents how well the hypothesis is able to describe the data, often referred to as the goodness of fit. By minimising the sum of these two components, MDL implements a tradeoff between complexity and goodness of fit. Also note that the selected hypothesis only depends on the *lengths* of the used code words, and not on the binary sequences that make up the code words themselves. This will make things a lot easier later on.

By its preference for short descriptions, MDL implements a heuristic that is widely used in science as well as in learning in general. This is the *principle of parsimony*, also often referred to as Occam's razor. A parsimonious learning strategy is sometimes adopted on the basis of a belief that the true state of nature is more likely to be simple than to be complex. We point out right at the outset that MDL is *not* based on such a belief. Instead, we try to avoid assumptions about the truth as much as possible, and focus on effective *strategies* for learning instead. As explained in Section 5.2, even if the truth is "complex", it is often a very effective strategy to prefer "simple" hypotheses for small sample sizes. That said, a number of issues still need to be addressed if we want to arrive at a practical theory for learning:

1. What code should be used to identify the hypotheses?
2. What codes for the data are suitable as formal representations of the hypotheses?
3. After answering the previous two questions, to what extent can the MDL criterion actually be shown to identify a "useful" hypothesis, and what do we actually mean by "useful"?

These questions, which lie at the heart of MDL research, are illustrated in the following example.

Example 1 (Language Learning)**.** Ray Solomonoff, one of the founding fathers of the concept of Kolmogorov complexity (to be discussed in Section 2.4), introduced the problem of language inference based on only positive examples as an example application of his new theory [39]; it also serves as a good example for MDL inference.

We will model language using context-free grammars (introduced as "phrase structure grammars" by Chomsky in [11]). A grammar is a set of *production rules*, each of which maps a symbol from a set of *nonterminal symbols* N to a (possibly empty) *replacement sequence* consisting of both other nonterminals and *terminal symbols*, which are words from some dictionary Σ. A sentence is grammatical if it can be produced from the *starting symbol*, which is a particular nonterminal, by iteratively applying a production rule to one of the matching nonterminal symbols. The following

grammar is an example. It uses the standard abbreviation that multiple rules for the same nonterminal may be combined using a pipe symbol, i.e. two rules $n \to r_1$ and $n \to r_2$ are written $n \to r_1 \mid r_2$. The empty sequence is denoted ϵ.

$$
\begin{array}{lcl}
\text{Sentence} & \to & \text{Nounphrase Verbphrase} \\
\text{Nounphrase} & \to & \text{Determiner Adjectives Noun} \mid \text{Adjectives Noun} \\
\text{Determiner} & \to & \texttt{the} \mid \texttt{a} \\
\text{Adjectives} & \to & \text{Adjective Adjectives} \mid \epsilon \\
\text{Adjective} & \to & \texttt{big} \mid \texttt{complex} \mid \texttt{careful} \mid \texttt{curvy} \\
\text{Noun} & \to & \texttt{men} \mid \texttt{statistician} \mid \texttt{model} \\
\text{Verbphrase} & \to & \text{Verb Nounphrase} \\
\text{Verb} & \to & \texttt{prefers} \mid \texttt{avoids}
\end{array}
\tag{2}
$$

This grammar accepts such sentences as `The careful statistician avoids the complex model`, or `The curvy model prefers big men`. We proceed to infer such a grammar from a list of valid example sentences D. Assume that we already know the words of the language, i.e. Σ contains all words that occur in D and is background knowledge. The task is to use D to learn how the words may be ordered. For the set of our hypotheses $\mathcal{H}$ we take all context-free grammars that use only words from Σ. Note that $\mathcal{H}$ is a countably infinite set.

We need to define the relevant code length functions: $L(H)$ for the specification of the grammar $H \in \mathcal{H}$, and $L_H(D)$ for the specification of the example sentences with the help of that grammar. In this example we use very simple code length functions; later in this introduction, after describing in more detail what properties good codes should have, we return to language inference with a more in-depth discussion.

We use *uniform codes* in the definitions of $L(H)$ and $L_H(D)$. A uniform code on a finite set A assigns binary code words of equal length to all elements of the set. Since there are 2^l binary sequences of length l, a uniform code on A must have code words of length at least $\lceil \log |A| \rceil$. (Here and in the following, $\lceil \cdot \rceil$ denotes rounding up to the nearest integer, and log denotes binary logarithm.) To establish a baseline, we first use uniform codes to calculate how many bits we need to encode the data literally, without the help of any grammar. Namely, we can specify every word in D with a uniform code on $\Sigma \cup \diamond$, where $\diamond$ is a special symbol used to mark the end of a sentence. Two consecutive diamonds signal the end of the data. Let w denote the total number of words in D, and let s denote the number of sentences. With the uniform code we need $(w+s+1)\lceil \log |\Sigma \cup \{\diamond\}| \rceil$ bits to encode the data. We are looking for grammars H which allow us to compress the data beyond this baseline value. We specify $L(H)$ as follows. For each production rule of H, we use $\lceil \log |N \cup \{\diamond\}| \rceil$ bits to uniformly encode the initial nonterminal symbol, and $\lceil \log |N \cup \Sigma \cup \{\diamond\}| \rceil$ bits for each symbol in the replacement sequence. The $\diamond$ symbol signals the end of each rule; an additional diamond marks the end of the entire grammar. If the grammar H has r rules, and the summed length of the replacement sequences is l, then we can calculate that the number of bits we need to encode the entire grammar is at most

$$(r+1)\lceil \log |N \cup \{\diamond\}| \rceil + (l+r)\lceil \log |N \cup \Sigma \cup \{\diamond\}| \rceil. \tag{3}$$

The first term is the length needed to describe the initial nonterminal symbols of each rule; the $+1$ is because, to signal the end of the grammar, we need to put an extra diamond. The second term is the length needed to describe the replacement sequences; the $+r$ is because, at the end of each rule, we put an extra diamond.

If a context-free grammar H is correct, i.e. all sentences in D are grammatical according to H, then its corresponding code L_H can help to compress the data, because it does not need to reserve code words for any sentences that are ungrammatical according to H. In this example we simply encode all words in D in sequence, each time using a uniform code on the set of words that *could* occur next according to the grammar. Again, the set of possible words is augmented with a $\diamond$ symbol to mark the end of the data. For example, based on the grammar (2) above, the 1-sentence data sequence $D =$`the model avoids men` is encoded using $\lceil \log 10 \rceil + \lceil \log 7 \rceil + \lceil \log 2 \rceil + \lceil \log 9 \rceil + \lceil \log 10 \rceil$, because, by following the production rules we see that the first word can be a determiner, an adjective, or a noun; since we also allow for a diamond, this gives 10 choices in total. Given the first word `the`, the second word can be an adjective or a noun, so that there are 7 choices (now a diamond is not allowed); and so on. The final $\log 10$ is the number of bits needed to encode the final diamond.

Now we consider two very simple correct grammars, both of which only need one nonterminal symbol S (which is also the starting symbol). The "promiscuous" grammar (terminology due to Solomonoff [39]) has rules $\mathrm{S} \to \mathrm{S}\ \mathrm{S}$ and $\mathrm{S} \to \sigma$ for each word $\sigma \in \Sigma$. This grammar generates *any* sequence of words as a valid sentence. It is very short: we have $r = 1 + |\Sigma|$ and $l = 2 + |\Sigma|$, so the number of bits $L(H_1)$ required to encode the grammar essentially depends only on the size of the dictionary and not on the amount of available data D. On the other hand, according to H_1 all words in the dictionary are allowed in all positions, so $L_{H_1}(D)$ requires as much as $\lceil \log(|\Sigma \cup \{\diamond\}|) \rceil$ bits for every word in D, which is equal to the baseline. Thus this grammar does not enable us to compress the data.

Second, we consider the "ad-hoc" grammar H_2. This grammar consists of a production rule $\mathrm{S} \to\ d$ for each sentence $d \in D$. Thus according to H_2, a sentence is only grammatical if it matches one of the examples in D exactly. Since this severely restricts the number of possible words that can appear at any given position in a sentence given the previous words, this grammar allows for very efficient representation of the data: $L_{H_2}(D)$ is small. However, in this case $L(H_2)$ is at least as large as the baseline, since in this case the data D appear literally in H_2!

Both grammars are clearly useless: the first does not describe any structure in the data at all and is said to *underfit* the data. In the second grammar random features of the data (in this case, the selection of valid sentences that happen to be in D) are treated as structural information; this grammar is said to *overfit* the data. Consequently, both grammars $H \in \{H_1, H_2\}$ do not enable us to compress the data at all, since the *total* code length $L(H) + L_H(D)$ exceeds the baseline. In contrast, by selecting a grammar H_{mdl} that allows for the greatest total compression as per (1), we avoid either extreme, thus implementing a natural tradeoff between underfitting and overfitting. Note that *finding* this grammar H_{mdl} may be a quite difficult search problem, but the computational aspects of finding the best hypothesis in large hypothesis spaces are not considered in this text. $\Diamond$

1.1 Overview

The three fundamental questions of page 2 will be treated in more detail in the following text. First we discuss codes for the set of hypotheses in terms of the concepts of *luckiness* and *regret* in Section 2. Although most previous treatments put less emphasis on luckiness (or fail to mention it altogether), it is our position that both concepts are fundamental to the MDL philosophy. Then in Section 3 we discuss how the data should be encoded based on a hypothesis. This requires that we outline the close relationship between codes and probability distributions, linking information theory to statistics. We then focus on the special case that the hypothesis is formulated in terms

of a *model*, which is a set of codes or probability distributions. We describe different methods of constructing *universal codes* to represent such models, which are designed to achieve low regret in the worst case. After describing the process of designing codes, in Section 4 we will try to convince you that it may be useful to try and compress your data, even if you are really interested in something different, such as truth finding or prediction of future events, which, arguably, are the primary goals of statistics. Finally in Section 5 we place Minimum Description Length in the context of other interpretations of learning.

2 Encoding the Hypothesis: A Matter of Luckiness and Regret

The intuition behind the MDL principle is that "useful" hypotheses should help compress the data. For now, we will simply define "useful" to be the same as "useful to compress the data". We postpone further discussion of what this means to Section 4. What remains is the task to find out which hypotheses are useful, by using them to compress the data.

Given many hypotheses, we could just test them one at a time on the available data until we find one that happens to allow for substantial compression. However, if we were to adopt such a methodology in practice, results would vary from reasonable to extremely bad. The reason is that among so many hypotheses, there could well be one that, by sheer force of luck, allows for significant compression of the data, even though it would probably do a very bad job of predicting future outcomes. This is the phenomenon of overfitting again, which we mentioned in Example 1. To avoid such pitfalls, we required that a *single* code L_{mdl} is proposed on the basis of all available hypotheses. The code has two parts: the first part, with length function $L(H)$, identifies a hypothesis to be used to encode the data, while the second part, with length function $L_H(D)$, describes the data using the code associated with that hypothesis.

Intuitively, the smaller $L_H(D)$, the better H fits data D. For example, if $\mathcal{H}$ is a set of probability distributions, then the higher the likelihood that H achieves on D, the smaller L_H. Section 3 is concerned with the precise definition of $L_H(D)$. Until then, we will assume that we have already represented our hypotheses in the form of codes, and we will discuss some of the properties a good choice for $L(H)$ should have. Technically, throughout this text we use only length functions that correspond to prefix codes; we will explain what this means in Section 3.1.

2.1 Avoid Regret

Consider the case where the best candidate hypothesis, i.e. the hypothesis $\hat{H} \in \mathcal{H}$ that minimises $L_H(D)$, achieves substantial compression. It would be a pity if we did not discover the usefulness of $\hat{H}$ because we chose a code word with unnecessarily long length $L(\hat{H})$. The *regret* we incur on the data quantifies how bad this "detection overhead" can be, by comparing the total code length $L_{\text{mdl}}(D)$ to the code length achieved by the best hypothesis $L_{\hat{H}}(D)$. A general definition, which also applies if $\hat{H}$ is undefined, is the following: the *regret* of a code L on data D with respect to a set of alternative codes $\mathcal{M}$ is[1]

$$\mathcal{R}(L, \mathcal{M}, D) \;:=\; L(D) - \inf_{L' \in \mathcal{M}} L'(D). \tag{4}$$

[1]In this text inf and sup are used for infimum and supremum, generalisations of minimum and maximum respectively.

The reasoning is now that, since we do not want to make a priori assumptions as to the process that generates the data, the code for the hypotheses $L(H)$ must be chosen such that the regret $\mathcal{R}(L_{\text{mdl}}, \{L_H \mid H \in \mathcal{H}\}, D)$ is small, *whatever data we observe*. This ensures that whenever $\mathcal{H}$ contains a useful hypothesis that allows for substantial compression of the data (more than this small regret), we detect this because L_{mdl} compresses the data as well.

Example 2. Suppose that $\mathcal{H}$ is finite. Let L be a uniform code that maps every hypothesis to a binary sequence of length $l = \lceil \log_2 |\mathcal{H}| \rceil$. The regret incurred by this uniform code is always exactly l, whatever data we observe. This is the best possible worst-case guarantee: all other length functions L' on $\mathcal{H}$ incur a strictly larger regret for at least one possible outcome (unless $\mathcal{H}$ contains useless hypotheses H which have $L(H) + L_H(D) > L_{\text{mdl}}(D)$ for all possible D). In other words, the uniform code minimises the *worst-case regret* $\max_D \mathcal{R}(L, \mathcal{M}, D)$ among all code length functions L. (We discuss the exact conditions we impose on code length functions in Section 3.1.)

Thus, if we consider a finite number of hypotheses we can use MDL with a uniform code $L(H)$. Since in this case the $L(H)$ term is the same for all hypotheses, it cancels and we find $H_{\text{mdl}} = \hat{H}$ in this case. What we have gained from this possibly anticlimactic analysis is the following *sanity check*: since we equated learning with compression, we should not trust $\hat{H}$, the hypothesis that best fits the data, to exhibit good performance on future data unless we were able to compress the data using H_{mdl}. $\Diamond$

2.2 Try to Get Lucky

There are many codes L on $\mathcal{H}$ that guarantee small regret, so the next task is to decide which we should pick. As it turns out, given any particular code L, it is possible to select a special subset of hypotheses $\mathcal{H}' \subset \mathcal{H}$ and modify the code such that the code lengths for these special hypotheses are especially small, at the cost of increasing the code lengths for the other hypotheses only slightly. This can be desirable, because *if* a hypothesis in $\mathcal{H}'$ turns out to be useful, then we are lucky, and we can identify that hypothesis more readily to achieve superior compression. On the other hand, if all hypotheses in the special subset are poor, then we have not lost much. Thus, while a suitable code must always have small regret, there is quite a lot of freedom to favour such small subsets of special hypotheses.

Examples of this so-called *luckiness principle* are found throughout the MDL literature, although they usually remain implicit, possibly because it makes MDL inference appear subjective. Only recently has the luckiness principle been identified as an important part of code design. The concept of luckiness is introduced to the MDL literature in [19]; [5] already uses a similar concept but not under the same name. We take the stance that the luckiness principle introduces only a mild form of subjectivity, because it cannot substantially harm inference performance. Paradoxically, it can only really be harmful *not* to apply the luckiness principle, because that could cause us to miss out on some good opportunities for learning!

Example 3. To illustrate the luckiness principle, we return to the grammar learning of Example 1. We will not modify the code for the data $L_H(D)$ defined there; we will only reconsider $L(H)$ that intuitively seemed a reasonable code for the specification of grammars. Note that $L(H)$ is in fact a luckiness code: it assigns significantly shorter code lengths to shorter grammars. What would happen if we tried to avoid this "subjective" property by optimising the worst-case regret without considering luckiness?

To keep things simple, we reduce the hypothesis space to finite size by considering only context-free grammars with at most $|N| = 20$ nonterminals, $|\Sigma| = 490$ terminals, $R = 100$ rules and replacement sequences summing to a total length of $L = 2000$. Using (3) we can calculate the luckiness code length for such a grammar as at most $101\lceil\log(21)\rceil + 2100\lceil\log(511)\rceil = 19405$ bits.

However, Example 2 shows that for finite $\mathcal{H}$, the worst-case regret is minimised for the uniform code. To calculate the uniform code length, we first count the number of possible grammars $H' \subset H$ within the given size constraints. One may or may not want to verify that

$$|\mathcal{H}'| \;=\; \sum_{r=1}^{R} |N|^r \sum_{l=0}^{L} |N \cup \Sigma|^l \binom{l+r-1}{l}.$$

Calculation on the computer reveals that $\lceil \log |\mathcal{H}'| \rceil = 18992$ bits. Thus, we could compress the data 413 bits better than the code that we used before. This shows that there is room for improvement of the luckiness code, which may incur a larger regret than necessary. On the other hand, with the uniform code we *always* need 18992 bits to encode the grammar, even when the grammar is very short! Suppose that the best grammar uses only $r = 10$ and $s = 100$, then the luckiness code requires only $11\lceil\log(21)\rceil + 110\lceil\log(511)\rceil = 1045$ bits to describe that grammar and therefore out-compresses the uniform code by 17947 bits: this grammar is identified much more easily using the luckiness code.

A simple way to combine the advantages of both codes is to define a third code that uses one additional bit to specify whether the luckiness code or the uniform code is used. The regret of this third code is at most one bit more than minimal in the worst case, while many simple grammars can be encoded extra efficiently. This one bit is the "slight increase" that we referred to at the beginning of this section. ◊

Note that we do not mean to imply that the hypotheses which get special luckiness treatment are *necessarily* more likely to be true than any other hypothesis. Rather, luckiness codes can be interpreted as saying that this or that special subset *might* be important, in which case we should like to know about it!

2.3 Infinitely Many Hypotheses

When $\mathcal{H}$ is countably infinite, there can be no upper bound on the lengths of the code words used to identify the hypotheses. Since any hypothesis *might* turn out to be the best one in the end, the worst-case regret is often infinite in this case. In order to retain MDL as a useful inference procedure, we are forced to embrace the luckiness principle. A good way to do this is to order the hypotheses such that H_1 is luckier than H_2, H_2 is luckier than H_3, and so on. We then need to consider only codes L for which the code lengths increase with the index of the hypothesis. This immediately gives a lower bound on the code lengths $L(H_n)$ for $n = 1, \ldots$, because the nondecreasing code that has the shortest code word for hypothesis n is uniform on the set $\{H_1, \ldots, H_n\}$ (and is unable to express hypotheses with higher indices). Thus $L(H_n) \geq \log |\{H_1, \ldots, H_n\}| = \log n$. It is possible to define codes L with $L(H_n) = \log n + O(\log \log n)$, i.e. not much larger than this ideal. Rissanen describes one such code, called the "universal code for the integers", in [28] (where the restriction to monotonically increasing code word lengths is not interpreted as an application of the luckiness principle as we do here); in Example 4 we describe some codes for the natural numbers that are convenient in practical applications.

2.4 Ideal MDL

In MDL hypothesis selection as described above, it is perfectly well conceivable that the data generating process has a very simple structure, which nevertheless remains undetected because it is not represented by any of the considered hypotheses. For example, we may use hypothesis selection to determine the best Markov chain order for data which reads "110010010000111111...", never suspecting that this is really just the beginning of the binary expansion of the number π. In "ideal MDL" such blind spots are avoided, by interpreting *any* code length function L_H that can be implemented in the form of a computer program as the formal representation of a hypothesis H. Fix a universal prefix Turing machine U. With some slight caveats, computer languages such as JAVA, C or LISP can be thought of as universal prefix mechanisms [20]. The result of running program T on U with input D is denoted $U(T, D)$. The *Kolmogorov complexity* of a hypothesis H, denoted $K(H)$, is the length of the shortest program T_H that implements L_H, i.e. $U(T_H, D) = L_H(D)$ for all binary sequences D. Now, the hypothesis H can be encoded by literally listing the program T_H, so that the code length of the hypotheses becomes the Kolmogorov complexity. For a thorough introduction to Kolmogorov complexity, see [24].

In the literature the term "ideal MDL" is used for a number of approaches to model selection based on Kolmogorov complexity; for more information on the version described here, refer to [6, 20]. The version of ideal MDL that we adopt here tells us to pick

$$\min_{H \in \mathcal{H}} K(H) + L_H(D), \tag{5}$$

which is (1), except that now $\mathcal{H}$ is the set of *all* hypotheses represented by computable length functions, and $L(H) = K(H)$.

In order for this code to be in agreement with MDL philosophy as described above, we have to check whether or not it has small regret. It is also natural to wonder whether or not it somehow applies the luckiness principle. The following property of Kolmogorov complexity is relevant for the answer to both questions. Let $\mathcal{H}$ be a countable set of hypotheses with computable corresponding length functions. Then for all computable length functions L on $\mathcal{H}$, we have

$$\exists c > 0 : \forall H \in \mathcal{H} : K(H) \leq L(H) + c. \tag{6}$$

Roughly speaking, this means that ideal MDL is ideal in two respects: first, the set of considered hypotheses is expanded to include all computable hypotheses, so that any computable concept is learned given enough data; since with any inference procedure that can be implemented as a computer algorithm — as they all can — we can only infer computable concepts, this makes ideal MDL "universal" in a very strong sense. Second, it matches all other length functions up to a constant, including all length functions with small regret as well as length functions with *any* clever application of the luckiness principle. Thus, we may think of the Kolmogorov Complexity code length $K(H)$ as implementing a sort of "universal" luckiness principle.

On the other hand, performance guarantees such as (6) are not very specific, as the constant overhead may be so large that it completely dwarfs the length of the data. To avoid this, we would need to specify a particular universal Turing machine U, and give specific upper bounds on the values that c can take for important choices of $\mathcal{H}$ and L. While there is some work on such a concrete definition of Kolmogorov complexity for individual objects [40], there are as yet no concrete performance guarantees for ideal MDL or other forms of algorithmic inference.

The more fundamental reason why ideal MDL is not practical, is that Kolmogorov complexity is uncomputable. Thus it should be understood as a theoretical ideal that can serve as an inspiration for the development of methods that can be applied in practice.

3 Encoding the Data

On page 2 we asked how the codes that formally represent the hypotheses should be constructed. Often many different interpretations are possible and it is a matter of judgement how exactly a hypothesis should be made precise. There is one important special case however, where the hypothesis is formulated in the form of a set of probability distributions. Statisticians call such a set a *model*. Possible models include the set of all normal distributions with any mean and variance, or the set of all third order Markov chains, and so on. Model selection is a prime application of MDL, but before we can discuss how the codes to represent models are chosen, we have to discuss the close relationship between coding and probability theory.

3.1 Codes and Probability Distributions

We have introduced the MDL principle in terms of coding; here we will make precise what we actually mean by a code and what properties we require our codes to have. We also make the connection to statistics by describing the correspondence between code length functions and probability distributions.

A *code* $C : \mathcal{X} \to \{0,1\}^*$ is an injective mapping from a countable source alphabet $\mathcal{X}$ to finite binary sequences called *code words*. We consider only *prefix codes*, that is, codes with the property that no code word is the prefix of another code word. This restriction ensures that the code is *uniquely decodable*, i.e. any concatenation of code words can be decoded into only one concatenation of source symbols. Furthermore, a prefix code has the practical advantage that no look-ahead is required for decoding, that is, given any concatenation S of code words, the code word boundaries in any prefix of S are determined by that prefix and do not depend on the remainder of S. Prefix codes are as efficient as other uniquely decodable codes; that is, for any uniquely decodable code with length function L_C there is a prefix code C' with $L_{C'}(x) \leq L_C(x)$ for all $x \in \mathcal{X}$, see [12, Chapter 5]. Since we never consider non-prefix codes, from now on, whenever we say "code", this should be taken to mean "prefix code".

Associated with a code C is a *length function* $L : \mathcal{X} \to \mathbb{N}$, which maps each source symbol $x \in \mathcal{X}$ to the length of its code word $C(x)$.

Of course we want to use efficient codes, but there is a limit to how short code words can be made. For example, there is only one binary sequence of length zero, two binary sequences of length one, four of length three, and so on. The precise limit is expressed by the Kraft inequality:

Lemma 3.1 (Kraft inequality)**.** *Let $\mathcal{X}$ be a countable source alphabet. A function $L : \mathcal{X} \to \mathbb{N}$ is the length function of a prefix code on $\mathcal{X}$ if and only if:*

$$\sum_{x \in \mathcal{X}} 2^{-L(x)} \quad \leq \quad 1.$$

Proof. See for instance [12, page 82]. □

If, for a given code C with length function$L$, the inequality holds strictly, then the code is called *defective*, otherwise it is called *complete*. (The term "defective" is usually reserved for probability distributions, but we apply it to code length functions as well.)

Let C be any prefix code on $\mathcal{X}$ with length function L, and define

$$\forall x \in \mathcal{X} : \qquad P(x) := 2^{-L(x)}. \tag{7}$$

Since $P(x)$ is always positive and sums to at most one, it can be interpreted as a probability mass function that defines a distribution corresponding to C. This mass function and distribution are called *complete* or *defective* if and only if C is.

Vice versa, given a distribution P, according to the Kraft inequality there must be a prefix code L satisfying

$$\forall x \in \mathcal{X} : \qquad L(x) := \lceil -\log P(x) \rceil. \tag{8}$$

To further clarify the relationship between P and its corresponding L, define the *entropy* of distribution P on a countable outcome space $\mathcal{X}$ by

$$H(P) := \sum_{x \in \mathcal{X}} -P(x) \log P(x). \tag{9}$$

(This H should not be confused with the H used for hypotheses.) According to Shannon's noiseless coding theorem [37], the mean number of bits used to encode outcomes from P using the most efficient code is at least equal to the entropy, i.e. for all length functions L' of prefix codes, we have $E_P[L'(X)] \geq H(P)$. The expected code length using the L from (8) stays within one bit of entropy. This one bit is a consequence of the requirement that code lengths have to be integers. Note that apart from rounding, (7) and (8) describe a one-to-one correspondence between probability distributions and code length functions that are most efficient for those distributions.

Technically, probability distributions are usually more convenient to work with than code length functions, because their usage does not involve rounding. But conceptually, code length functions are often more intuitive objects than probability distributions. A practical reason for this is that the probability of the observations typically decreases exponentially as the number of observations increases, and such small numbers are hard to handle psychologically, or even to plot in a graph. Code lengths typically grow linearly with the number of observations, and have an analogy in the real world, namely the effort required to remember all the obtained information, or the money spent on storage equipment. A second disadvantage of probability theory is the philosophical controversy regarding the interpretation of probability, which is discussed in Section 5.3.

In order to get the best of both worlds: the technical elegance of probability theory combined with the conceptual clarity of coding theory, we generalise the concept of coding such that code lengths are no longer necessarily integers. While the length functions associated with such "ideal codes" are really just alternative representations of probability mass functions (or even densities), and probability theory is used under the hood, we will nonetheless call negative logarithms of probabilities "code lengths" to aid our intuition and avoid confusion. Since this difference between an ideal code length and the length using a real code is at most one bit, this generalisation should not require too large a stretch of the imagination. In applications where it is important to actually encode the data, rather than just compute its code length, there is a practical technique called arithmetic coding [25] which can usually be applied to achieve the ideal code length to within a few bits; for the purposes of MDL inference it is sufficient to compute code lengths, and we do not have to construct actual codes.

Example 4. In Section 2.3 we remarked that a good code for the natural numbers always achieves code length close to $\log n$. Consider a probability distribution W defined as $W(n) = f(n) - f(n+1)$ for some function $f : \mathbb{N} \to \mathbb{R}$. If f is decreasing with $f(1) = 1$ and $f \to 0$, then W is an easy to use probability mass function that can be used as a basis for coding the natural numbers. To see this, note that $\sum_{i=1}^{m} W(i) = 1 - f(n+1) < 1$ and that therefore $\sum_{i=1}^{\infty} W(i) = 1$. For $f(n) = 1/n$ we get code lengths $-\log W(n) = \log(n(n+1)) \approx 2 \log n$, which, depending on the application, may be small enough. Even more efficient for high n are $f(n) = n^{-\alpha}$ for some $0 < \alpha < 1$ or $f(n) = 1/\log(n+1)$. $\Diamond$

3.2 Universal Coding and Model Selection

We have discussed earlier what code L should be used to identify the hypotheses. Here we address the question what code L_H should represent a hypothesis $H \in \mathcal{H}$. So far we have treated the data as a monolithic block of information, but it is often convenient to think of the data as a sequence of observations $D = x_1, x_2, \ldots, x_n$ instead. For instance, this makes it easier to discuss sequential prediction later on. For simplicity we will assume each observation x_i is from some countable outcome space $\mathcal{X}$; we also abbreviate $x^n = x_1, \ldots, x_n$.

In case that the hypothesis H is given in terms of a probability distribution P from the outset, $H = P$, then we identify it with the code satisfying (8) but without the integer requirement, so that $L_H(x^n) = -\log P(x^n)$. There are various reasons why this is the only reasonable mapping [19]. Now no more work needs to be done and we can proceed straight to doing MDL hypothesis selection as described above.

In the remainder of this section, we consider instead the case where the hypothesis is given in the form of a *model* $\mathcal{M} = \{P_\theta \mid \theta \in \Theta\}$, which is a set of probability distributions parameterised by a vector θ from some set Θ of allowed parameter vectors, called the *parameter space*. For example, a model could be the set of all fourth order Markov chains, or the set of all normal distributions. In this case it is not immediately clear which single code should represent the hypothesis.

To motivate the code that represents such a model, we apply the same reasoning as we used in Section 2. Namely, the code should guarantee a small regret, but is allowed to favour some small subsets of the model on the basis of the luckiness principle. To make this idea more precise, we define a *maximum likelihood estimator* as a function $\hat{\theta} : \mathcal{X}^* \to \Theta$ satisfying

$$P_{\hat{\theta}(x^n)}(x^n) = \max\{P_\theta(x^n) \mid \theta \in \Theta\}. \tag{10}$$

Obviously, $\hat{\theta}(x^n)$ also minimises the code length. We abbreviate $\hat{\theta} = \hat{\theta}(x^n)$ if the sequence of outcomes x^n is clear from context. We start by associating with each individual $P_\theta \in \mathcal{M}$ the corresponding code L_θ with $L_\theta(x^n) = -\log P(x^n)$, and henceforth write and think of $\mathcal{M}$ as a set of *codes* $\{L_\theta \mid \theta \in \Theta\}$.

Definition 3.2. Let $\mathcal{M} := \{L_\theta \mid \theta \in \Theta\}$ be a (countable or uncountably infinite) model with parameter space Θ. Let $f : \Theta \times \mathbb{N} \to [0, \infty)$ be some function. A code L is called *f-universal* for a model $\mathcal{M}$ if, for all $n \in \mathbb{N}$, all $x^n \in \mathcal{X}^n$, we have

$$\mathcal{R}(L, \mathcal{M}, x^n) \leq f(\hat{\theta}, n).$$

Our notion of universality is called *individual-sequence* universality in the information-theoretic literature [19]. When information theorists talk of universal codes, they usually refer to another

definition in terms of expectation rather than individual sequences [12]. Our formulation is very general, with a function f that needs further specification. Generality is needed because, as it turns out, different degrees of universality are possible in different circumstances. This definition allows us to express easily what we expect a code to live up to in all these cases.

For finite $\mathcal{M}$, a uniform code similar to the one described in Example 2 achieves f-universality for $f(\hat{\theta}, n) = \log |\mathcal{M}|$. Of course we incur a small overhead on top of this if we decide to use luckiness codes.

For countably infinite $\mathcal{M}$, the regret cannot be bounded by a single constant, but we can avoid dependence on the sample size. Namely, if we define a mass function W on the parameter set, we can achieve f-universality for $f(\hat{\theta}, n) = -\log W(\hat{\theta})$ by using a Bayesian or two-part code (these are explained in subsequent sections).

Finally for uncountably infinite $\mathcal{M}$ it is often impossible to obtain a regret bound that does not depend on n. For parametric models however it is often possible to achieve f-universality for

$$f(\hat{\theta}, n) = \frac{k}{2} \log \frac{n}{2\pi} + g(\hat{\theta}), \tag{11}$$

where k is the number of parameters of the model and $g : \Theta \to [0, \infty)$ is some continuous function of the maximum likelihood parameter. Examples include the Poisson model, which can be parameterised by the mean of the distribution so $k = 1$, and the normal model, which can be parameterised by mean and variance so $k = 2$. Thus, for parametric uncountable models a logarithmic dependence on the sample size is the norm.

In MDL model selection, universal codes are used on two levels. On the inner level, each model H is represented by a universal code for that model. Then on the outer level, a code with length function $L(H) + L_H(D)$ is always used, see (1). This code is also universal, this time with respect to $\{L_H \mid H \in \mathcal{H}\}$, which is typically countable. This outer-level code is called a *two-part code*, explained in more detail below. Thus, MDL model selection proceeds by selecting the H (a model, i.e. a family of distributions) that minimises the two-part code $L(H) + L_H(D)$, where $L_H(D)$ itself is a universal code relative to model H.

We now describe the four most common ways to construct universal codes, and we illustrate each code by applying it to the model of Bernoulli distributions $\mathcal{M} = \{P_\theta \mid \theta \in [0, 1]\}$. This is the "biased coin" model, which contains all possible distributions on 0 and 1. The distributions are parameterised by the probability of observing one: $P_\theta(X = 1) = \theta$. Thus, $\theta = \frac{1}{2}$ represents the distribution of an unbiased coin. The distributions in the model are extended to n outcomes by taking the n-fold product distribution: $P_\theta(x^n) = P_\theta(x_1) \cdots P_\theta(x_n)$.

3.2.1 Two-Part Codes

In Section 2 we defined a code L_{mdl} that we now understand to be universal for the model $\{L_H \mid H \in \mathcal{H}\}$. This kind of universal code is called a two-part code, because it consists first of a specification of an element of the model, and second of a specification of the data using that element. Two-part codes may be defective: this occurs if multiple code words represent the same data sequence x^T. In that case one must ensure that the encoding function is well-defined by specifying exactly which representation is associated with each data sequence. Since we are only concerned with code *lengths* however, it suffices to adopt the convention that we always use one of the shortest representations.

We mentioned that, in model selection problems, universal coding is applied on two levels. At the outer level, two-part coding is always used because it allows us to associate a single hypothesis

with the achieved code length $L_{\mathrm{mdl}}(x^n)$. In model selection problems, two-part coding *can* also be used at the inner level. An example of a two-part code that can be used as a universal code on the inner level follows below. We will see later that on this inner level, other universal codes are sometimes preferable because they are more efficient and do not require discretisation.

Example 5. We define a two-part code for the Bernoulli model. In the first part of the code we specify a parameter value, which requires some discretisation since the parameter space is uncountable. However, as the maximum likelihood parameter for the Bernoulli model is just the observed frequency of heads, at a sample size of n we know that the ML parameter is one of $0/n$, $1/n$, $\ldots$, n/n. We discretise by restricting the parameter space to this set. A uniform code uses $L(\theta) = \log(n+1)$ bits to identify an element of this set. Therefore the resulting regret is always exactly $\log(n+1)$. By using a slightly more clever discretisation we can bring this regret down to about $\frac{1}{2}\log n + O(1)$, which we mentioned is usually achievable for uncountable single parameter models. $\Diamond$

3.2.2 The Bayesian Universal Distribution

Let $\mathcal{M} = \{P_\theta \mid \theta \in \Theta\}$ be a countable model; it is convenient to use mass functions rather than codes as elements of the model here. Now define a distribution with mass function W on the parameter space Θ. This distribution is called the *prior distribution* in the literature as it is often interpreted as a representation of a priori beliefs as to which of the hypotheses in $\mathcal{M}$ represents the "true state of the world". More in line with the philosophy outlined above would be the interpretation that W is a *code* which should be chosen for practical reasons to optimise inference performance. At any rate, the next step is to define a *joint distribution* P on $\mathcal{X}^n \times \Theta$ by $P(x^n, \theta) = P_\theta(x^n)W(\theta)$. In this joint space, each outcome comprises a particular state of the world and an observed outcome.

In Bayesian statistics, inference is always based on this joint distribution. For our purpose, the one relevant quantity is the *marginal likelihood* of the data:

$$P(x^n) = \sum_{\theta \in \Theta} P_\theta(x^n)W(\theta). \tag{12}$$

Note that the marginal likelihood is a weighted average, satisfying

$$P_{\hat{\theta}}(x^n)W(\hat{\theta}) \;\leq\; P(x^n) \;\leq\; P_{\hat{\theta}}(x^n), \tag{13}$$

where the first inequality is obtained by underestimating (12) by a single term of the sum, and the second by overestimating by the largest item the weighted average is taken over. Together these bounds express that the Bayesian marginal likelihood is at most a factor $W(\hat{\theta})$ worse than the best element of the model, which is not very much considering that $P_{\hat{\theta}}(x^n)$ is often exponentially small in n.

Note that this Bayesian universal code yields a code length less than or equal to the code length we would have obtained with the two-part code with $L(\theta) = -\log W(\theta)$. Since we already found that two-part codes are universal, we can now conclude that Bayesian codes are at least as universal, with regret *at most* $-\log W(\hat{\theta})$. On the flip side, the sum involved in calculating the Bayesian marginal distribution can be hard to evaluate in practice.

Example 5 (continued)**.** Our definitions readily generalise to uncountable models with $\Theta \subseteq \mathbb{R}^k$, with the prior distribution given by a density w on Θ. Rather than giving explicit definitions we revisit our running example.

We construct a Bayesian universal code for the Bernoulli model. For simplicity we use a uniform prior density, $w(\theta) = 1$. Let n_0 and $n_1 = n - n_0$ denote the number of zeroes and ones in x^n, respectively. Now we can calculate the Bayes marginal likelihood of the data:

$$P(x^n) = \int_0^1 P_\theta(x^n) \cdot 1 \ \mathrm{d}\theta = \int_0^1 \theta^{n_1}(1-\theta)^{n_0} \, \mathrm{d}\theta = \frac{n_0!\, n_1!}{(n+1)!}.$$

Using Stirling's approximation of the factorial, we find that the corresponding code length $-\log P(x^n)$ equals $-\log P_{\hat{\theta}}(x^n) + \frac{1}{2}\log n + O(1)$. Thus we find a regret similar to that achieved by a well-designed two-part code, but the constant will be slightly better. $\Diamond$

3.2.3 Normalised Maximum Likelihood

The Normalised Maximum Likelihood universal code is preferred in the MDL literature, because it provably minimises the worst-case regret. It is not hard to show that the code minimising the worst-case regret must achieve equal regret for all possible outcomes x^n. In other words, the total code length must always be some constant longer than the code length achieved on the basis of the maximum likelihood estimator [19, Chapter 6]. The Normalised Maximum Likelihood (NML) distribution is defined to achieve exactly this:

$$P_{\mathrm{nml}}(x^n) := \frac{P_{\hat{\theta}(x^n)}(x^n)}{\sum_{y^n \in \mathcal{X}^n} P_{\hat{\theta}(y^n)}(y^n)}. \tag{14}$$

For all outcomes x^n, the regret on the basis of this distribution is exactly equal to the logarithm of the denominator, called the *parametric complexity* of the model:

$$\inf_L \sup_{x^n} \mathcal{R}(L, \mathcal{M}, x^n) = \log \sum_{y^n \in \mathcal{X}^n} P_{\hat{\theta}(y^n)}(y^n). \tag{15}$$

Under some regularity conditions on the model it can be shown [34, 19] that, if we restrict the parameters to an *ineccsi* set $\Theta_0 \subset \Theta$, i.e. a subset of Θ that is closed, bounded, has nonempty interior and excludes the boundary of Θ, then the parametric complexity is finite and given by

$$\frac{k}{2}\log\frac{n}{2\pi} + \log \int_{\Theta_0} \sqrt{\det I(\theta)} \, \mathrm{d}\theta + o(1), \tag{16}$$

in accordance with (11) for constant function $g(\hat{\theta})$. Here $\det I(\theta)$ is the determinant of the Fisher information matrix, a fundamental quantity in statistics (see [19, Chapter 6] for a precise statement). This is an important result: for example, it implies that we can never hope to construct universal codes with regret smaller than (16) on any substantial subset Θ_0 of Θ. Thus, if the parametric complexity is large, then with any coding method whatsoever we need a lot of bits to code the data, on top of the best-fitting element of the model. The model is thus "complex", independently of the particular description method we use. We may also look at this in a different way: we see from the sum in (15) that a model is complex iff for many data y^n sequences, it contains a distribution that fits these sequences well in the sense that $P_{\hat{\theta}(y^n)}(y^n)$ is large. Thus, the parametric complexity of a model is related not so much to how many distributions it contains, but to how many patterns can be fit well by a distribution in it. It can be shown [19, Chapter 6] that this is related to the number of "essentially different" distributions the model contains.

We note that neither the parametric complexity (15) nor its approximation (16) depend on the particular chosen parameterisation, i.e. the function which maps θ to P_θ. While the parameterisation is arbitrary, the parametric complexity and its approximation are inherent properties of the model $\mathcal{M}$ under consideration.

Example 5 (continued)**.** The parametric complexity (15) has exponentially many terms, but for the Bernoulli model the expression can be significantly simplified. Namely, we can group together all terms which have the same maximum likelihood estimator. Thus the minimal worst-case regret can be rewritten as follows:

$$\log \sum_{y^n \in \mathcal{X}^n} P_{\hat{\theta}(y^n)}(y^n) = \log \sum_{n_1=0}^{n} \binom{n}{n_1} \left(\frac{n_1}{n}\right)^{n_1} \left(\frac{n-n_1}{n}\right)^{n-n_1}. \tag{17}$$

This term has only linearly many terms and can usually be evaluated in practice. Approximation by Stirling's formula confirms that the asymptotic regret is $\frac{1}{2}\log n + O(1)$, the same as for the other universal distributions. $\Diamond$

The NML distribution has a number of significant practical problems. First, it is often undefined, because for many models the denominator in (14) is infinite, even for such simple models as the Poisson or geometric distributions. Second, $\mathcal{X}^n$ is exponentially large in n, so calculating the NML probability exactly is only possible in special cases such as the Bernoulli model above, where the number of terms is reduced using some trick. Something similar is possible for the more general multinomial model, see [22], but in most cases (15) has to be approximated, which introduces errors that are hard to quantify.

3.2.4 Prequential Universal Distributions and Predictive MDL

A distribution P (or its corresponding code), defined for a sequence of outcomes $x^n := x_1, x_2, \ldots, x_n$ can be reinterpreted as a *sequential (probabilistic) prediction strategy* using the chain rule:

$$\begin{aligned} P(x^n) &= P(x^1) \cdot \frac{P(x^2)}{P(x^1)} \cdot \ldots \cdot \frac{P(x^n)}{P(x^{n-1})} \\ &= P(x_1) \cdot P(x_2|x^1) \cdot \ldots \cdot P(x_n|x^{n-1}). \end{aligned} \tag{18}$$

The individual factors are the actual probabilities $P(x_1), P(x_2 \mid x^1), \ldots$ assigned to $x_1, x_2, \ldots$ by the conditional "predictive" probability *distributions* $P(X_1 = \cdot \mid x_1), P(X_2 = \cdot \mid x^2), \ldots$ Thus, to any distribution P for the data, there corresponds a sequential prediction strategy, where we predict x_1 by marginalising P before having seen any data, then we look at x_1, condition P accordingly and marginalise again to obtain a prediction of x_2, and so on. The chain rule shows that the probability of x^n must be equal to the product of the individual predictive probabilities. Vice versa, any prediction strategy that will issue a distribution on the next outcome x_{i+1} given any sequence of previous outcomes x^i, defines a distribution (or code) on the whole data via the chain rule. Together with the correspondence between code length functions and probability distributions described above, this links the domains of coding and sequential prediction.

An algorithm that, given a sequence of previous observations x^n, issues a probability distribution on the next outcome $P(X_{n+1}|x^n)$, may be viewed as a sequential (probabilistic) prediction strategy. In Dawid's theory of prequential analysis such algorithms are called *prequential forecasting systems*

[13, 15, 14]. A prediction strategy defines a distribution on x^n by the chain rule (18). Vice versa, for any distribution P on $\mathcal{X}^n$, application of the chain rule yields a prediction strategy. We give two important prediction strategies here.

First, we may take a Bayesian approach and define a joint distribution on $\mathcal{X}^n \times \Theta$ based on some prior distribution W. As before, this induces a marginal distribution on $\mathcal{X}^n$ which in turn defines a prediction strategy. Thus, the Bayesian universal distribution can be reinterpreted as a prediction strategy: it is in fact an important special case of a prequential universal distribution.

Second, since the Bayesian predictive distribution can be hard to compute, it may be useful in practice to define prediction strategies that use simpler algorithms. Perhaps the simplest option is to predict an outcome X_{n+1} using a maximum likelihood estimator (10) evaluated on the previous outcomes. This "prequential maximum likelihood plug-in " approach was suggested independently by Dawid and by Rissanen who calls it *predictive MDL* [29, 30], and we will use it in our running example.

Example 5 (continued)**.** The ML estimator for the Bernoulli model parameterised by $P_\theta(X = 1) = \theta$, equals the frequency of ones in the sample: $\hat{\theta}(x^n) = \frac{1}{n}\sum_{n=1}^{n} x_n$. We define a prediction strategy through $P(X_{n+1} = 1|x^n) := P_{\hat{\theta}(x^n)}(X_{n+1} = 1) = \hat{\theta}(x^n)$.

This strategy is ill-defined for the first outcome. Another impractical feature is that it assigns probability 0 to the event that the second outcome is different from the first. To address these problems, we slightly tweak the estimator: rather than $\hat{\theta}$ we use $\tilde{\theta} = (1+\sum_{n=1}^{n} x_n)/(n+2)$. Perhaps surprisingly, in this case the resulting prediction strategy is equivalent to the Bayesian universal distribution approach we defined in the previous section: $P_{\tilde{\theta}}$ turns out to be the Bayesian predictive distribution for the Bernoulli model if a uniform prior density $w(\theta) = 1$ is used. This is somewhat coincidental: for non-Bernoulli models, the distribution indexed by such a "tweaked" ML estimator is usually quite different from the Bayesian predictive distribution [19, Chapter 9]. $\lozenge$

The ML plug-in code is useful because it is often easier to implement than the other universal codes: it does not require discretisation, like the 2-part code, nor integration over the parameter space, like the Bayesian code, nor summation over all possible data sequences, like NML. Instead it only requires calculation of the ML estimator.

While the prequential ML code has been used successfully in practical inference problems, it is shown in [16] that (in expectation) it does not necessarily achieve the desired regret (11) of $(k/2)\log n + O(1)$ unless the data are actually sampled from a distribution in the model. Application of the ML plug-in code thus requires exactly the kind of assumption about the "truth" that we have been trying to avoid. Therefore the ML plug-in code should be used with care, especially in applications of model selection.

Another feature of prequential universal codes is that in general they are sensitive to the ordering of the data, which may be undesirable in case the data have no natural ordering. Interestingly, this does not apply to the Bayesian sequential prediction strategy, which is order-independent in i.i.d. settings. That is: if we change the order of the data x^n, then the individual predictions of x_i given x^{i-1} change, but the product (18) of these probabilistic predictions remains the same.

Unordered Data When prequential universal codes are defined for models that assume the data to be unordered (the most important case being i.i.d. data), the result often assigns different code lengths to different permutations of the data sequence, even if all codes that make up the models do not. Fortunately, the Bayesian prequential code does not suffer this effect. Neither does it

occur for the 2-part and NML universal codes, which are not prequential. However the ML plug-in code length generally does depend on the order of the data. Also, several methods to turn NML codes into prediction strategies are studied in the literature [35]; the resulting prequential codes do typically depend on the order of the data. Similarly, recent attention to *model switching* (Section 5.5) has resulted in codes that introduce order dependence.

If the data are unordered, should we not encode the data as a *set* rather than as a *sequence*, in other words should we not try to avoid encoding the (arbitrary) order altogether? We do not favour such an approach, because once the order of the data is disregarded, we can no longer compare the code lengths we obtain to those we would get using other hypotheses that do assume order dependence! It would prevent us from testing our *assumption* that the data are unordered. A second, more practical argument against disregarding the order is that it is often easier to encode sequences than it is to encode sets. As long as all permutations of the sequence are assigned equal code lengths, no bias is introduced when comparing code lengths for sequences rather than sets.

However, as we mentioned many prequential codes do depend on the order. This means that we have to make sure that these codes still achieve low regret in the worst case, i.e. that their performance does not deteriorate too much when the data appear in a particularly unlucky order. Unfortunately, for many prequential codes such worst-case results have not been proven, and we have to make do with performance guarantees either in expectation or with high probability. While this issue does not appear cause too much trouble in practice, it is slightly at odds with our interpretation of MDL philosophy.

4 The Purpose of MDL

While we have equated "learning" with "compressing the data" so far, in reality we often have a more specific goal in mind when we apply learning algorithms. This touches on the third question we asked on page 2: what do we mean by a "useful" hypothesis? In this section we will argue that MDL inference, while designed to achieve optimal compression of the data, is also suitable in some settings with a different objective, so that, at least to some extent, MDL provides a reliable one-size-fits-all solution.

Example 1 (continued)**.** We return to grammar learning. The code we used for the data given the grammar is not very sophisticated. A better approach is the following: to encode each sentence, we start with the starting symbol, and then we repeatedly specify which production rule is to be applied to the first remaining nonterminal, until none are left. We still have to choose a code/distribution on the rules that match every nonterminal, to be able to specify which matching rule is applied. Of course, in practice, some production rules will be used much more often than others, and by specifying the "right" probabilities (approximately matching empirical frequencies) for each rule, we may achieve substantial compression of the data. Such a grammar, where each nonterminal has a distribution on the matching production rules is called a *stochastic context-free grammar*.

We generally do not know how we should set the probabilities to obtain optimal code lengths. However, we now have the tools to minimise the worst-case regret. Namely, each grammar defines a *model* on sentences, a set of distributions that is parameterised by the probabilities of applying production rules to matching nonterminals. For each model, we can subsequently define one of the universal codes of Section 3.2. This results in a code for each grammar that performs *almost as*

well as the stochastic context-free grammar with optimally tuned probabilities of the production rules.

Each of the applications of MDL we describe below can be usefully applied to stochastic context free grammars. (1) We may use the two-part code we defined for grammar and data to *compress* the data (Section 4.1). (2) We may use the grammar to *predict* the continuation of the data (Section 4.3). Language models are used for prediction in applications such as speech recognition. (3) We may infer a grammar that matches the data well, without attempting to learn the model parameters as well. We then obtain a *model selection* problem (Section 4.2.2). Or alternatively (4) we may attempt to learn not only the grammar, but the parameters as well. This is the *estimation problem* of Section 4.2.1. Note that in these last two cases, theoretical consistency results only carry over to real data in case those data are really sampled from a stochastic context free grammar, which may be unrealistic. ◊

4.1 Data Compression

Until now we have considered MDL for compression, which is an important practical goal in itself. A substantial amount of progress in the data compression literature builds on MDL research. For example, the elegant Context Tree Weighting algorithm [48] expands on earlier MDL-related work by Rissanen [27]. The following subsections discuss in detail how compression is related to learning from data, first in the form of "truth finding", which includes estimation and model selection, and then in the form of prediction. The MDL Principle is more general though: at least in principle, it can be applied to all problems involving inductive inference, not just compression, prediction and truth finding. For example, it is eminently suited for denoising and separating structure from noise, and it has also been used for similarity analysis, clustering, lossy data compression and DNA sequence alignment. We will not go into these applications here; for a non-exhaustive list with references, see [19, Chapter 1].

4.2 Truth Finding

Let $\mathcal{H}$ be a set of probability distributions, and let x^n be the observed data. The MDL principle provides a clear interpretation of learning without the assumption that the data are sampled from one of the distributions in $\mathcal{H}$, or indeed without any probabilistic assumptions at all. It can be applied in situations in which some $P \in \mathcal{H}$ is "adequate" (in the sense that it leads to valid insights) or "useful" (in the sense that it leads to reasonable predictions of future data; such P may be quite far off from the underlying "truth"). *Still*, as a sanity check, we want to make sure that MDL methods provably perform well in the special case in which some $P^* \in \mathcal{H}$ is true, in the sense that the data are sampled from P^*. Below we verify that MDL methods indeed behave reasonably in such ideal circumstances. First, in Section 4.2.1, we consider the case where the goal is simply to infer a good approximation of P^*. This includes the traditional statistical problems of parametric and nonparametric density estimation. In that situation, MDL performs well in the sense that, with high probability, the MDL methods output a good approximation to the true P^* even at modest sample sizes. In Section 4.2.2 we consider model selection, where $\mathcal{H}$ is carved up in finite or countably many parametric models $\mathcal{M}_1, \mathcal{M}_2, \ldots$, and the goal is to identify the (smallest) $\mathcal{M}_{\delta^*}$ containing the true P^*. In that situation, MDL performs well in the sense that, with high probability, when input a large enough sample, the MDL methods outputs $\mathcal{M}_{\delta^*}$. Finally, in Section 4.3, we very briefly consider a version of truth finding more closely related to prediction.

4.2.1 Estimation

Here we assume that the true distribution P^* lies in the hypothesis space $\mathcal{H}$, and investigate whether we can approximate P^* using MDL. The hypothesis space $\mathcal{H}$ may be a parametric model (e.g., the set of Bernoulli distributions, or the set of second-order Markov chains); in that case our goal reduces to a most familiar topic of statistics: parameter estimation. Alternatively, $\mathcal{H}$ may be so large that it cannot be indexed by a finite set of real-valued parameters. An example is the set of all distributions for $0 < X < 1$ that have differentiable densities. In that case, our goal is what statisticians call "nonparametric density estimation". Yet another possibility is that $\mathcal{H} = \bigcup_{k=1,2,\ldots} \mathcal{M}_k$, where each $\mathcal{M}_k$ is a k-dimensional parametric model. In this last scenario we assume the truth to be "parametric", but the dimension is unknown. The difference with model selection proper, which is discussed further below, is that in estimation we are interested in inferring a single element P^* residing in some of the $\mathcal{M}_k$, whereas in model selection we only try to find out which $\mathcal{M}_k$ contains P^*.

For all such estimation problems we use a two-part code; we first encode an element P of $\mathcal{H}$ using a code length function denoted L, and then we encode the data using the code that corresponds to P. In total we need $L(P) - \log P(x^n)$ bits to encode x^n via P. We then select the $\ddot{P} \in \mathcal{H}$ that minimises this total. This $\ddot{P}$ may be viewed as an estimator in the statistical sense: it maps data sequences of arbitrary length to distributions in $\mathcal{H}$. In general, the performance of statistical estimators is often expressed in terms of their *statistical risk*, the expected distance between $\ddot{P}$ and the true P^*. For technical reasons it is convenient to measure the risk in terms of the *squared Hellinger distance* D_{He}, a distance measure between probability distributions (see [19] for a definition). With respect to this distance, the risk of the 2-part MDL estimator relative to true distribution P^* at sample size n is defined as

$$R_n(P^*, \ddot{P}) = \underset{X^n \sim P^*}{E} \left[D_{\mathrm{He}}(P^*, \ddot{P}) \right]. \tag{19}$$

Note that $\ddot{P}$ is really a function of the X^n over which we take expectation. As more and more data are gathered, the 2-part MDL estimator tends to get better and better in the sense that the risk tends to get smaller and smaller. The *rate of convergence* expresses how quickly the risk converges to 0. We have the following:

Theorem 4.1. *Suppose that, for all $P^* \in \mathcal{H}$, data $X_1, X_2, \ldots$ are i.i.d. under P^*. Then, under a mild regularity condition on the code length function $L(H)$, we have, for all $P^* \in \mathcal{H}$,*

$$R_n(P^*, \ddot{P}) \leq \frac{E_{X^n \sim P^*} \left[L(\ddot{P}) - \log \ddot{P}(X^n) - [-\log P^*(X^n)] \right]}{n}. \tag{20}$$

In words: *the statistical risk of two-part code MDL relative to P^* is bounded by the expected overhead of the two-part code relative to P^*, divided by n.* This provides a deep link between data compression and truth finding: *if* our code happens to achieve a small code length using some approximation of the true distribution P^* (i.e. the right-hand side of (20) is small), then the expected performance of the two-part estimator is very good (i.e., the left-hand side of (20) is small). For example, if P^* has a finite code length $L(P^*)$ under the code L, then the right-hand side of (20) is bounded by

$$n^{-1}[L(\ddot{P}) - \log \ddot{P}(X^n) - [-\log P^*(X^n)]] \leq n^{-1}[L(P^*) - \log P^*(X^n) + \log P^*(X^n)] = \frac{L(P^*)}{n}.$$

Thus the risk converges to 0 at least at rate $1/n$, which, as compared to other statistical estimators, is quite fast. Note the role of luckiness: the smaller the code length we assigned to (good approximations of) P^*, the better the performance bound on two-part MDL. In many cases, the structure of $\mathcal{H}$ is such that we can design codes with good worst-case behaviour, even if $\mathcal{H}$ is uncountable. For example, if $\mathcal{H}$ is a parametric model and if a two-part code with a clever discretisation scheme is used, then it can be shown using (20) that $\max_{P^* \in \mathcal{H}} R_n(P^*, \ddot{P}) = O(1/n)$. This is the *minimax optimal rate*, i.e. in the worst case over all $P^* \in \mathcal{H}$, no statistical estimation method can have rate smaller than c/n for some $c > 0$. In the same way one can use (20) to show, for suitably chosen L, that two-part MDL achieves the minimax convergence rates for a range of nonparametric problems. Compare this to the classical maximum likelihood estimation method, which also achieves the minimax rate on parametric problems, but which may fail utterly on nonparametric problems.

Theorem 4.1 is due (in slightly different versions) to A. Barron, J. Li and T. Zhang, and appeared first in Li's Ph.D. thesis [23]. Its precise statement and implications are discussed at length in Chapter 15 of [19]. It is of fundamental importance, since it directly links compression and learning: the better we can compress data from P^*, the faster we can identify a good approximation of P^*.

4.2.2 Model Selection

We have described models as formal representations of hypotheses; we will now try to determine which of the hypotheses is "true", in the sense that the corresponding model contains the data generating distribution P^*. The goal is then to identify this true model on the basis of as little data as possible. In the MDL approach to this problem, we use a two-part code on the outer level (since, based on data x^n, we want to output a particular model $\mathcal{M}$, which therefore has to be encoded explicitly), and we can use any type of universal code at the inner level.

Since the universal code for the true model achieves a code length not much larger than the code length of the best code in the model (it was designed to achieve small regret), and the best code in the model achieves code length at most as large as the data generating distribution, it seems reasonable to assume that, as more data are being gathered, a true model will eventually be selected by MDL. This intuition is confirmed in the form of the following consistency result, which is one of the pillars of MDL model selection. The result applies to Bayesian model selection as well.

We give a precise statement of the theorem, but also provide a more informal explanation for those unfamiliar with all the concepts involved.

Theorem 4.2 (Model Selection Consistency)**.** *Let $\mathcal{H} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$ be a countably infinite set of parametric models. For all $i \in \mathbb{Z}^+$, let $\mathcal{M}_i$ be 1-to-1 parameterised by $\Theta_i \subseteq \mathbb{R}^k$ for some $k \in \mathbb{N}$ and define a prior density w_i on Θ_i. Define a prior distribution W on the model indices. We require for all integers $j > i > 0$ that with w_j-probability 1, a distribution drawn from $\mathcal{M}_j$ is mutually singular with all distributions in $\mathcal{M}_i$. Define the MDL model selection criterion based on Bayesian universal codes to represent the models:*

$$\delta(x^n) = \arg\min_i \left(-\log W(i) - \log \int_{\theta \in \Theta_i} P_\theta(x^n) w_i(\theta) \, \mathrm{d}\theta \right).$$

Then for all $\delta^* \in \mathbb{Z}^+$, for all $\theta^* \in \Theta_{\delta^*}$, except for a subset of Θ_{δ^*} of Lebesgue measure 0, it holds for $X_1, X_2, \ldots \sim P_{\theta^*}$ that

$$P_{\theta^*}(\exists n_0 \, \forall n \geq n_0 : \delta(X^n) = \delta^*) \;=\; 1.$$

Proof. Proofs of various versions of this theorem can be found in [4, 14, 3, 19] and others. □

Thus, the theorem can be applied if to a sequence of models, as long as models do not contain a significant number of distributions that also occur in earlier models. The δ-function in the theorem indicates which model is selected by MDL given a specific sequence of observations. The theorem then states, roughly, that from some sample size n_0 onwards the true model will be always be selected by MDL.

The theorem uses a Bayesian code on the inner level (the distributions within each model). However as conjectured in [19], it can probably be extended to other universal codes on the inner level, such as NML (for some special cases, it was proven in [41] that this is indeed the case). Note that (as we keep emphasising) we do not make any assumptions about the true distribution P^*. However, *if* we are in the ideal situation where one of the models contains P^*, then the theorem guarantees that this model will be selected once sufficient data have been accumulated. This property of model selection criteria is called *consistency*. Unfortunately, the theorem says nothing about the more realistic scenario where the models are merely approximations of the true data generating process. If the true data generating process P^* is not an element of any of the models, we might still be interested in finding the best approximating model, for instance the model that contains the distribution $P_{\tilde{\theta}}$ minimising the Kullback-Leibler divergence $D(P^* \| P_{\tilde{\theta}})$. It is not known under what circumstances δ actually achieves this. On the other hand, some model selection criteria that are often used in practice (such as maximum likelihood model selection, or AIC [2]) are known *not* to be consistent, even in the restricted sense of Theorem 4.2. Therefore consistency of MDL and Bayesian model selection is reassuring.

At this point, we should add that MDL model selection, while being consistent, does not always achieve the minimax optimal convergence rate, in the following sense: suppose we first use MDL model selection to select a model $\mathcal{M}_k$, and then, within the chosen model, we use a second estimator (which may again be a two-part code MDL estimator), to select a distribution $\dot{P}_\theta \in \mathcal{M}_k$. Then, for some standard situations such as linear regression, using MDL model selection in the first stage, the Hellinger risk $R_n(P^*, \ddot{P}_\theta)$ as defined in Section 4.2.1 will not converge to 0 at the fastest possible rate; in general, it is slower by a $\log n$ factor. The same holds for BIC and Bayes factor model selection: all these model selection methods are typically consistent, but do not achieve the optimal rate. In contrast, methods such as AIC and leave-one-out cross-validation (see Section 4.3 below) can be inconsistent, but do achieve the minimax rate. Quite recently, we have shown that with a different, more sophisticated code at the outer level of MDL selection — the so-called *switch code* — this problem can be overcome, and both consistency and optimal convergence rates are achieved; see Section 5.5.

4.3 Prediction

To use MDL for sequential prediction, we have to use codes that can be rendered as prequential forecasting systems. This includes any prequential universal code, in particular the Bayesian universal code and "predictive MDL" [29, 30]. See Section 3.2.4 for more information about prequential codes.

There are several ways to evaluate the performance of a probabilistic prediction strategy P. In the theoretical machine learning literature [8], it is customary to look at the *accumulated loss*

defined as

$$\sum_{i=1}^{n} \mathrm{LOSS}(x_i, P(X_i = \cdot \mid x^{i-1})). \tag{21}$$

Here LOSS can be any function mapping pairs of outcomes $x \in \mathcal{X}$ and probability distributions on $\mathcal{X}$ to the real numbers, possibly including infinity. A loss function with particularly useful theoretical properties is the logarithmic loss, $\mathrm{LOSS}(x, P) = -\log P(x)$. With this loss function, the accumulated loss reduces to

$$\sum_{i=1}^{n} -\log P(X_i = x_i \mid x^{i-1}) = -\log \prod_{i=1}^{n} P(x_i \mid x^{i-1}) = -\log P(x^n), \tag{22}$$

which is just the code length of the sample! This binds compression and predictive performance together directly: the more we compress, the better the predictive performance in terms of one step ahead prediction loss. In the context of MDL, let P represent a universal code relative to some set of hypotheses $\mathcal{H}$. Then (22) implies that, if one of the distributions in $\mathcal{H}$, used as a prediction strategy, predicts x^n well, and P is guaranteed to have small regret relative to $\mathcal{H}$, then P is also guaranteed to predict x^n well. An important question is of course whether this result transfers from the logarithmic loss to the particular loss function that is of interest in the problem at hand. It turns out that, if, as in Section 4.2.1, we are prepared to assume that one of the distributions in $\mathcal{H}$ is true, then (with some caveats) this is indeed the case; a more precise statement can be found in [19, Chapter 17].

But even if we stick to logarithmic loss, the simple identity (22) has several important consequences. First of all, it is the basis of a fundamental theorem, due to A. Barron [5] and closely related to Theorem 4.1, that gives explicit guarantees about the convergence rate for prediction strategies based on MDL prequential universal codes. Analogously to Theorem 4.1, the better a code compresses data sampled from P^*, the faster it learns to predict outcomes from P^* well. This is discussed in Chapter 15 of [19]. The theorem also has consequences for estimation, because we may also think of the predictions $P(X_n = \cdot \mid X^{n-1})$ as an estimate of P^* rather than as a predictor of X_n. In this way, each prequential universal code in fact defines an estimator relative to $P^* \in \mathcal{H}$; this provides an alternative, "prequential" rather than "2-part", MDL estimation method. Under this interpretation, Barron's theorem now expresses something even more similar to Theorem 4.1: the better we can compress data from P^* using a prequential code, the faster the estimator based on this code converges to P^*.

Yet another consequence of (22) is that in some common situations, MDL model selection is a special case of Dawid's theory of prequential model validation [13, 15, 14]. His *weak prequential principle* states that we should evaluate a prequential forecasting system based solely on the predictions it made *for the data that were actually observed*, not the predictions that it might have made for other sequences of observations. This can be achieved by measuring the performance of a forecasting system in terms of its one step ahead prediction loss (21). Thus, suppose we want to choose between a finite number of models $\mathcal{M}_1, \ldots, \mathcal{M}_k$. According to the prequential idea, we should associate each of these models with a prediction strategy, , and then pick the model whose associated prediction strategy achieves the smallest accumulated loss (21) on $x_1, \ldots, x_n$. The prediction strategies could, for example, be maximum likelihood plug-in codes as in the example of Section 3.2.4. Such an approach is similar to leave-one-out cross validation in the sense that the models are evaluated using observed data only. The only difference is that in cross validation the prediction of each x_i is based on both the previous outcomes $x_1, \ldots, x_{i-1}$ and the future outcomes

$x_{i+1}, \ldots, x_n$. In prequential validation, only the previous outcomes are used. As we can see from (22), if prequential validation is applied with the logarithmic loss function and prediction strategies $P_1, \ldots, P_k$ associated with models $\mathcal{M}_1, \ldots, \mathcal{M}_k$ respectively, then it is simply equivalent to MDL model selection between these models with a uniform code (prior) on the outer level, and the codes based on $P_1, \ldots, P_k$ on the inner level.

5 MDL In Perspective

In this section we link MDL to other approaches to learning, discussing both philosophical and technical differences and similarities. We also trace the development of the MDL principle back to its origin in algorithmic information theory and Kolmogorov complexity, and point out areas of recent development.

5.1 Basic Principles

The following advice summarises the basic principles of MDL as discussed above.

- Use codes to represent hypotheses about the data; measure the usefulness of the hypothesis by achieved compression.
- Use a code that achieves low regret whatever data are observed, and carefully consider its luckiness properties. This applies to both the inner and the outer level for model selection.
- Design codes with performance guarantees that depend only on the model, but *not on the data generating process given the model*.[2]

5.2 Ockham's Razor

When two models fit the data equally well, MDL will choose the one that is the "simplest" in the sense that it allows for a shorter description of the data. As such, it implements a precise form of Occam's razor — even though as more and more data become available, the model selected by MDL may become more and more "complex"! Occam's razor is sometimes criticised for being either (1) arbitrary or (2) false [47, 18]. Do these criticisms apply to MDL as well?

1. "Occam's Razor (and MDL) Is Arbitrary" Because "description length" is a syntactic notion it may seem that MDL selects an arbitrary model: different codes would have led to different description lengths, and therefore, to different models. By changing the encoding method, we can make 'complex' things 'simple' and vice versa. This criticism overlooks the fact that we are not allowed to use just any code we like! MDL prescribes, for each model $\mathcal{M}_k$ under consideration, the use of a universal code L_k that achieves small worst-case regret. This prescription puts strong constraints on what models may be viewed as "simple" and what models may be viewed as "complex". First, we note that, if the models $\mathcal{M}_k = \{P_\theta \mid \theta \in \Theta_k\}$ are parametric and the parameter sets are suitably bounded, then we can associate with each model an inherent complexity, namely its worst-case regret (15). We already explained why it makes sense to call this quantity "parametric

[2] This may seem to be at odds with our discussion of MDL consistency in Section 4.2: do we not assume that the true distribution P^* is in one of the models there? However, the two statements "We do not know anything about P^*" and "If P^* is in one of the models, our method behaves especially nicely" are not contradictory.

complexity". It does not depend on the chosen parameterisation, and is thus certainly not arbitrary. But in practice, we do not want to restrict the parameter set, and then we have to use codes L_k with different regret on different sequences. However, it can be shown that, whatever code we L_k we use, for all subsets Θ_0 of Θ, for "most" sequences x^n such that $\hat{\theta}(x^n) \in \Theta_0$, the regret of L_k is larger than (16), which we repeat here for convenience:

$$\frac{k}{2} \log \frac{n}{2\pi} + \log \int_{\Theta_0} \sqrt{\det I(\theta)} \, \mathrm{d}\theta + o(1). \tag{23}$$

For a precise statement, see [19, Chapter 14]. Thus, the fraction of sequences for which L_k is "lucky" and achieves regret substantially smaller than (23) is vanishingly small. Thefore, it still makes sense to call the regret of L_k the "complexity of model $\mathcal{M}_k$ relative to sequence x^n". For most sequences, this complexity will be equal to $(k/2) \log n$ plus some small constant.

Example 6. Let us consider a prototypical model selection example, regression with polynomial models. The data are given by $(x_1, y_1), \ldots, (x_n, y_n)$ with $(x_i, y_i) \in \mathbb{R}^2$. The models are of form $\mathcal{M}_k = \{P_\theta \mid \theta \in \mathbb{R}^{k+1}\}$ where $P_\theta \in \mathcal{M}_k$, $\theta = (\theta_0, \ldots, \theta_k)$, prescribes that $Y = f_\theta(X) + Z$, where $f_\theta(x) = \sum_{j=0}^{k} \theta_j x^j$ is a k-degree polynomial and Z is a normally distributed error (noise) term with mean 0. P_θ is extended to n outcomes by independence. For example, $\mathcal{M}_1$ represents the set of 1-dimensional polynomials, i.e. straight lines $Y = aX + b$, turned into probability distributions by adding a stochastic noise term. For now we restrict ourselves to model selection between $\mathcal{M}_1$ and $\mathcal{M}_2$. If we perform MDL model selection with these two models, we must associate each $\mathcal{M}_k$ with a universal code defined by its length function L_k, and we must also provide a universal code on the outer level with lengths $L(k)$ for $k \in \{1, 2\}$. We may interpret the regret achieved by L_k relative to $\mathcal{M}_k$ for a given sequence as the "complexity" of $\mathcal{M}_k$ relative to that sequence. Now the sometimes-heard argument "MDL is arbitrary since one could have used codes L_1 and L_2 such that $\mathcal{M}_2$ is always preferred over $\mathcal{M}_2$", is simply wrong. Since the code on the outer level should be universal and achieve small worst-case regret, we prefer a uniform code here. The code L_2 we assign to $\mathcal{M}_2$ should be such that its regret is approximately $(3/2) \log n$ plus a small constant, since $\mathcal{M}_2$ has 3 free parameters. The regret of L_1 relative to $\mathcal{M}_1$ should be approximately $(2/2) \log n = \log n$ plus a small constant. As indicated by (23), codes such that for most sequences, the regret relative to $\mathcal{M}_2$ is substantially smaller than $(3/2) \log n$ simply do not exist. Codes such that for most sequences, the regret relative to $\mathcal{M}_1$ is substantially larger than $(2/2) \log n$ do exist, but they are highly wasteful, and the MDL principle does not allow us to use them. Hence, if proper MDL codes are used, then for most sequences , the "complexity" associated with $\mathcal{M}_2$ relative to that sequence, which we identified with the regret of L_2 relative to that sequence, is substantially larger than the complexity associated with $\mathcal{M}_1$ relative to that sequence, the difference being about $(1/2) \log n$. It can be shown that this implies that, if proper MDL codes are used, then there *must* be sequences on which MDL chooses $\mathcal{M}_1$ rather than $\mathcal{M}_2$ and vice versa. There are no "MDL codes" for which $\mathcal{M}_1$ is always preferred over $\mathcal{M}_2$ or vice versa. $\Diamond$

It is true that MDL leaves room for subjective input in the form of the luckiness principle. The statistician has the freedom to choose some special set A of hypotheses, or parameters within a model, and make them especially easy to learn. Of course, it may occur that these lucky few turn out *not* to improve compression performance, while giving another set B special treatment would have improved performance by a lot. This may seem arbitrary, but consider that the alternative is to make *everything* hard to learn. Also consider that subjective is not the same as arbitrary,

there may be good reason to choose A rather than B as the lucky set, and if there is not, then we can even consider the union $A \cup B$ as a potential lucky set, at the modest price of increasing the codelength for sequences with maximum likelihood parameters in A slightly. It is completely impossible however to make *everything* lucky, so there are hard constraints on the freedom of the statistician. To illustrate, we turn back to the regression example for polynomials of degree one or two. Given enough data, we should be able to estimate the parameters of any second degree polynomial up to any desired precision, but this will occur a lot faster if the parameters happen to be in the special subset $\mathcal{M}_1$, which can thus be viewed as a lucky set. We could also have considered an alternative lucky set $\mathcal{M}_1'$, containing all polynomials with $Y_i = aX_i + b + 13X^2$. The choice between $\mathcal{M}_1$ and $\mathcal{M}_1'$ is subjective (not arbitrary), but whichever we pick, we can only make a code such that $\mathcal{M}_1$ is really lucky if $\mathcal{M}_1$ is a very small fraction of $\mathcal{M}_2$, for which the worst-case regret is set in stone. Here "really lucky" means that the regret of sequences with maximum likelihood distributions close to or in $\mathcal{M}_1$ is significantly smaller than the regret of the remaining sequences.

2. "Occam's Razor Is False" It is often claimed that Occam's razor is false — we often try to model real-world situations that are arbitrarily complex, so why should we favour simple models? In the words of Webb [47], "What good are simple models of a complex world?"[3]

The short answer is: even if the true data-generating machinery is very complex, it may be a good strategy to prefer simple models for small sample sizes. Thus, MDL (and the corresponding form of Occam's razor) is a *strategy* for inferring models from data ("choose simple models at small sample sizes"), not a statement about how the world works ("simple models are more likely to be true") — indeed, a strategy cannot be true or false; it is "clever" or "stupid." And the strategy of preferring simpler models is clever even if the data-generating process is highly complex, as illustrated by the following example.

Example 7. (Simple Models of Complex Sources) We return to the polynomial regression setting of the previous example, but now we suppose that the data are sampled from a very complex source, say, a polynomial of degree 1,000. Then, if we use two-part code MDL to learn a polynomial for data $((x_1, y_1), \ldots, (x_n, y_n))$, the model $\mathcal{M}_k$ selected by MDL at sample size n will have $k < 1,000$ for small n, but k will typically increase with n, and almost surely it will settle on $\mathcal{M}_{1000}$ for all n larger than some n_0 (by consistency Theorem 4.2). Now, one could claim that MDL infers the "wrong" degree polynomial for $n < n_0$. Following this line of reasoning, suppose that we somehow knew the "right" degree, and redefined our method to report: the degree is 1,000. If we then try to fit this model for small n, say $n < 1000$, we would have far to few data to accurately estimate all 1001 parameters, which could well lead to disastrous predictions of future outcomes from the same source! On the other extreme, a low degree polynomial, say a parabola, may not be able to fit the data so well, but it has so few parameters that 1,000 observations suffice to estimate the optimal parameter values for this simple approximation quite accurately. Therefore, the mean squared error achieved on the available data should be a good indication of the squared error that should be expected on future data. So even though the model is "wrong" in the sense that it is much simpler than the true source, it yields much more reliable predictions. MDL implements a tradeoff between these two extremes: given enough data, the complex truth will be discovered, but if there is insufficient data to do so, MDL will report a simple, but useful, approximation.

The source we considered so far was complex, but at least it was in one of the considered models. It is far from uncommon however that the source is not in *any* of the models. The same reasoning

[3]Quoted with permission from *KDD Nuggets 96,28*, 1996.

applies in this case: a polynomial of sufficiently low degree will give a correct impression of how well it will predict future data, even if the data are not polynomial and none of the considered models is "true"! ◊

The ideas in this example appear not only in the MDL literature, but is also the basis of Vapnik's [42] structural risk minimisation approach and many standard statistical methods for nonparametric inference. In such approaches one acknowledges that the data-generating machinery can be very complex, and might not even be consistent with the considered models. Nevertheless, it is still a good strategy to approximate it by simple hypotheses (low-degree polynomials) as long as the sample size is small. Summarising:

The Inherent Difference between Under- and Overfitting (A) If we choose an overly simple (small) model for our data, then the best-fitting point hypothesis within the model is likely to be almost the best predictor, within the simple model, of future data coming from the same source. On the other hand, (B), if we overfit (choose a very complex, i.e., large, model) and there is noise in our data, then, *even if the complex model contains the "true" point hypothesis*, the best-fitting point hypothesis within the model is likely to lead to very bad predictions of future data coming from the same source. This statement is imprecise and is meant to convey the general idea, but it becomes provably true if we use MDL's measure of model complexity; we measure prediction quality by logarithmic loss; and we assume that one of the distributions in $\mathcal{H}$ actually generates the data. In fact, statement (A) is an implication of Barron's theorem about MDL prediction that we briefly referred to in Section 4.3.

5.3 MDL, Bayesian Inference and Frequentist Statistics

The MDL, Bayesian and Frequentist schools of thought differ in their interpretation of how the concept of probability relates to the real world.

The frequentist school of thought holds that probability can only express something about the real world in the context of a repeatable experiment. The frequency of a particular observation converges as more observations are gathered; this limiting value is then called the probability. This interpretation is too restrictive for many applications; for example the probability that a suspect is guilty (as often required in legal reasoning), or the probability that it will rain tomorrow, are not frequentist, because no repeatable experiments are involved.

According to the subjective Bayesian school on the other hand, a probability expresses a degree of belief in a certain proposition, which can make sense even outside the context of a repeatable experiment. However, this interpretation is also problematic, because in practice people of necessity often work with very crude models, which everybody agrees have no real truth to them. In such cases, probabilities are often used to represent "beliefs" which one knows a priori to be false! (In order to avoid offending people, we have to point out here that there are actually many different Bayesian schools of thought, and many practitioners would probably hesitate to subscribe completely to the subjectivist interpretation.) For further discussion, see e.g. [36, 7].

In stark contrast, from an MDL point of view a probability distribution is really just another word for a code, which allows us to avoid this philosophical can of worms altogether. We do not think of codes in terms of "truth" or "belief"; instead, a code is judged by its *efficiency* when it is applied in practice: a good code achieves a short code length on the data that we observe, whether it is based on valid assumptions about the truth or not.

In spite of these quite profound differences in interpretation, *technically* MDL and Bayesian inference are very closely related. In fact, because Bayesian codes are usually mathematically elegant and, with a suitable choice of prior, can be made to achieve low regret in the worst case, they more often than not turn out to be the code of preference in MDL settings. (There is then a close correspondence between "luckiness functions" and prior distributions, but as explained in [19, Chapter 17], some differences remain.) However, MDL research sometimes employs non-Bayesian codes, while Bayesians sometime use priors which do not guarantee low regret. In fact, there are some known inconsistency problems with Bayesian inference in nonparametric settings [17]. These problems are invariably due to the use of priors which achieve larger regret than necessary. By adopting nonparametric priors with small regret, as prescribed by the MDL principle, the problems disappear. This is a direct consequence of results such as Theorem 4.1, which show that small coding regret implies fast learning.

5.4 A Brief History of MDL

The practical MDL principle that we discuss in this book has mainly been developed by Rissanen in a series of papers starting in 1978 with [26]. It has its roots in the theory of Kolmogorov complexity [24], developed in the 1960s by Solomonoff [39], Kolmogorov [21] and Chaitin [9, 10]. Among these authors, Solomonoff (a former student of the famous philosopher of science, Rudolf Carnap) was explicitly interested in inductive inference. His 1964 paper contains explicit suggestions on how the underlying ideas could be made practical, thereby foreshadowing some of the later work on two-part MDL. While Rissanen was not aware of Solomonoff's work at the time, Kolmogorov's 1965 paper [21] did serve as an inspiration for Rissanen's [26] first development of MDL.

Another important inspiration for Rissanen was Akaike's (1973) AIC method for model selection (Section 4.2), essentially the first model selection method based on information-theoretic ideas [1, 2]. Even though Rissanen was inspired by AIC, both the actual method and the underlying philosophy are substantially different from MDL.

Minimum *Message* Length MDL is much closer related to the *Minimum Message Length (MML) Principle* [43], developed by Wallace and his coworkers in a series of papers starting in 1968 with the groundbreaking [44]; other milestones are [45] (1975) and [46] (1987). Remarkably, Wallace developed his ideas without being aware of the notion of Kolmogorov complexity. Although Rissanen became aware of Wallace's work before the publication of the first MDL paper [26], he developed his ideas mostly independently, being influenced more by Akaike and Kolmogorov. Indeed, despite the close resemblance of both methods in practice, the underlying philosophy is very different. A 10-page discussion of the precise relationship between MML and MDL, both technically and philosophically, can be found in Chapter 17 of [19].

The first publications on MDL only mention two-part codes. Important progress was made in 1984 when Rissanen used prequential codes for the first time [29], and again in 1987 [32], when Rissanen introduced Bayesian mixture codes in MDL. This led to the development of the notion of stochastic complexity as the shortest code length of the data given a model [31, 32]. However, the full development of the notion of "parametric complexity" (15) had to wait until 1996, when (again) Rissanen made the connection to Shtarkov's *normalised maximum likelihood code* [34]. In the mean time, A. Barron [4] showed in his impressive Ph.D. thesis how a specific version of the two-part code criterion has excellent frequentist statistical consistency properties. This was extended in 1991 by Barron and Cover [6] who achieved a breakthrough for two-part codes: they gave clear prescriptions

on how to design codes for hypotheses, relating codes with good minimax code length properties to rates of convergence in statistical consistency theorems, both for parametric and nonparametric problems. Some of the ideas of Rissanen's 1987 and Barron and Cover's 1991 paper were, as it were, unified when in 1996 Rissanen [34] introduced the normalised maximum likelihood code. The resulting theory was summarised for the first time by Barron, Rissanen and Yu [3], and is the main subject of the first (2007) comprehensive overview of the field, [19]. In this book, the modern versions of MDL based on explicit rather than ad hoc goals when designing a code is called "refined MDL".

5.5 Recent Developments

While this text is intended to be introductory rather than exhaustive, there have been a couple of recent developments in MDL research that shed new light on its interpretation, and which we consider relevant for anyone who currently uses or plans to use MDL or Bayesian inference in practice.

Luckiness Functions Although we emphasised the importance of luckiness in the design of the code for hypotheses in Section 2, it did not really play a part in the definition of the universal codes later in Section 3.2. This is for historical reasons: it was not known until recently (and nor was it considered important) how universal codes could be constructed with *both* low-worst case regret *and* a freely chosen luckiness structure. In [19] it is shown how this can be achieved using *luckiness functions*. A luckiness function $a : \Theta \to \mathbb{R}$ roughly expresses which parameter values in θ are especially easy to learn, should they provide good fit. While luckiness-related concepts were always implicit in MDL-like inference, the idea of using a luckiness function seems due to Barron [5], although he did not call it this way. The luckiness terminology was first adopted by Grünwald [19], who imported it from the computational learning theory literature [38] where it plays a somewhat similar role.

Model Switching The outer-level codes that are used in model selection problems are designed to achieve low worst-case regret, where regret is normally measured against the "best" model, i.e. the one that allows for the most compression. However, in predictive settings, practitioners often find that *there is no single best model*. Instead, simpler models tend to predict well as long as the sample size is small, while more complex models predict better as more data are gathered. This so-called *catch-up phenomenon* leads to the idea of measuring regret not against the model with the best *overall* predictive performance, but against a sequence of models, where each model in the sequence is used in turn to predict a number of outcomes. There are efficient and simple algorithms to achieve this, which often leads to significant improvements in model selection performance, both in theory and in practice. For example, in many situations, one can achieve both consistency and optimal convergence rates; see Section 4.2.2. For more details, please refer to [41].

References

[1] H. Akaike. Information theory and an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki, editors, *Second International Symposium on Information Theory*, pages 267–281, Budapest, 1973. Akademiai Kiado.

[2] H. Akaike. A new look at statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[3] A. Barron, J. Rissanen, and B. Yu. The Minimum Description Length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998.

[4] A.R. Barron. *Logically Smooth Density Estimation*. PhD thesis, Dept. of Electrical Engineering, Stanford University, Stanford, CA, 1985.

[5] A.R. Barron. Information-theoretic characterization of Bayes performance and the choice of priors in parametric and nonparametric problems. In *Bayesian Statistics 6*, pages 27–52. Oxford University Press, 1998.

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

[7] J.O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer Series in Statistics. Springer-Verlag, New York, revised and expanded second edition, 1985.

[8] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning and Games*. Cambridge University Press, Cambridge, UK, 2006.

[9] G.J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13:547–569, 1966.

[10] G.J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM*, 16:145–159, 1969.

[11] N. Chomsky. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3), September 1956.

[12] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Series in telecommunications. John Wiley, 1991.

[13] A.P. Dawid. Present position and potential developments: Some personal views, statistical theory, the prequential approach. *Journal of the Royal Statistical Society, Series A*, 147(2):278–292, 1984.

[14] A.P. Dawid. Prequential analysis, stochastic complexity and Bayesian inference. In J.M. Bernardo, J.O. Berger, A.P. Dawid, and A.F.M. Smith, editors, *Bayesian Statistics 4*, pages 109–125. Oxford University Press, 1992.

[15] A.P. Dawid. Prequential data analysis. In M. Ghosh and P.K. Pathak, editors, *Current Issues in Statistical Inference: Essays in Honor of D. Basu*, Lecture Notes-Monograph Series, pages 113–126. Institute of Mathematical Statistics, 1992.

[16] Steven de Rooij and Peter Grünwald. An empirical study of minimum description length model selection with infinite parametric complexity. *Journal of Mathematical Psychology*, 50:180–190, 2006.

[17] P. Diaconis and D. Freedman. On the consistency of Bayes estimates. *The Annals of Statistics*, 14(1):1–26, 1986.

[18] P. Domingos. The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.

[19] P.D. Grünwald. *The Minimum Description Length Principle*. MIT Press, June 2007. Chapter 17, containing further discussion of philosophical and conceptual issues, can be freely downloaded from www.grunwald.nl.

[20] P.D. Grünwald and P.M.B. Vitányi. Algorithmic information theory. In P. Adriaans and J. van Benthem, editors, *Handbook of the Philosophy of Science*, volume 8: Philosophy of Information, pages 289–325. Elsevier Science, 2008.

[21] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):1–7, 1965.

[22] P. Kontkanen and P. Myllymäki. A linear-time algorithm for computing the multinomial stochastic comp lexity. *Information Processing Letters*, 103:227–233, September 2007.

[23] J.Q. Li. *Estimation of Mixture Models*. PhD thesis, Yale University, New Haven, CT, 1999.

[24] M.Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, New York, 3rd edition, 2008.

[25] J. Rissanen. Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20(3), 1976.

[26] J. Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.

[27] J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, IT-29(5):656–664, 1983.

[28] J. Rissanen. A universal prior for integers and estimation by Minimum Description Length. *Annals of Statistics*, 11:416–431, 1983.

[29] J. Rissanen. Universal coding, information, prediction and estimation. *IEEE Transactions on Information Theory*, 30:629–636, 1984.

[30] J. Rissanen. A predictive least squares principle. *IMA Journal of Mathemathical Control and Information*, 3:211–222, 1986.

[31] J. Rissanen. Stochastic complexity and modeling. *Annals of Statistics*, 14:1080–1100, 1986.

[32] J. Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society, Series B*, 49:223–239, 1987. Discussion: 252–265.

[33] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*, volume 15 of *Series in Computer Science*. World Scientific, 1989.

[34] J. Rissanen. Fisher information and stochastic complexity. *IEEE Transactions on Information Theory*, 42(1):40–47, 1996.

[35] J. Rissanen and T. Roos. Conditional NML universal models. In *2007 Information Theory and Applications Workshop (ITA-07)*, pages 337–314, 2007.

[36] L.J. Savage. *The Foundations of Statistics*. Dover Publications, 1954.

[37] C.E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 623–656, 1948.

[38] J. Shawe-Taylor, P. Bartlett, R.C. Williamson, and M. Anthony. Structural risk minimisation over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

[39] R.J. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Information and Control*, 7:1–22, 224–254, 1964.

[40] J. Tromp. Binary lambda calculus and combinatory logic. Available at http://www.cwi.nl/ tromp/cl/cl.html, 2007.

[41] T. van Erven, P.D. Grünwald, and S. de Rooij. Catching up faster by switching sooner: a prequential solution to the AIC-BIC dilemma. Available at http://arxiv.org/abs/0807.1005, 2008. Submitted to Journal of the Royal Statistical Society, Series B. A much shorter version appears in the proceedings of NIPS 2008.

[42] V.N. Vapnik. *Statistical Learning Theory.* Wiley, New York, 1998.

[43] C.S. Wallace. *Statistical and Inductive Inference by Minimum Message Length*. Springer-Verlag, New York, 2005.

[44] C.S. Wallace and D.M. Boulton. An information measure for classification. *Computing Journal*, 11:185–195, 1968.

[45] C.S. Wallace and D.M. Boulton. An invariant Bayes method for point estimation. *Classification Society Bulletin*, 3(3):11–34, 1975.

[46] C.S. Wallace and P.R. Freeman. Estimation and inference by compact coding. *Journal of the Royal Statistical Society B*, 49:240–251, 1987. Discussion: pages 252–265.

[47] G.I. Webb. Further experimental evidence against the utility of Occam's razor. *Journal of Artificial Intelligence Research*, 4:397–417, 1996.

[48] F.M.J. Willems, Y.M. Shtarkov, and Tj.J. Tjalkens. Reflections on 'the Context-Tree Weighting method: Basic properties'. *Newsletter of the IEEE Information Theory Society*, 1997.