Minimum Description Length Model Selection

de Rooij, S.

Publication date
2008

Link to publication

Citation for published version (APA):
de Rooij, S. (2008). *Minimum Description Length Model Selection*. [Thesis, fully internal, Universiteit van Amsterdam].

Chapter 1

The MDL Principle for Model Selection

Suppose we consider a number of different hypotheses for some phenomenon. We have gathered some data that we want to use 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 [67, 71]. 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 minimum total description length as a function of the data is denoted L_{mdl}:

$$H_{\text{mdl}} := \arg\min_{H\in\mathcal{H}} \Big(L(H) + L_H(D)\Big), \tag{1.1}$$

$$L_{\text{mdl}}(D) := \min_{H\in\mathcal{H}} \Big(L(H) + L_H(D)\Big). \tag{1.2}$$

(In case that there are several H that minimise (1.1), we take any one among those with smallest $L(H)$.) 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 prefer to argue in the opposite direction: only if the truth is simple, or at least has a simple approximation, we stand a chance of learning its mechanics based on a limited data set [39]. In general, we try to avoid assumptions about the truth as much as possible, and focus on effective *strategies* for learning instead. 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 1.1.3), introduced the problem of language inference based on only positive examples as an example application of his new theory [84]; it also serves as a good example for MDL inference.

Suppose we want to automatically infer a language based on a list of valid example sentences D. We assume that the vocabulary Σ, which contains all words that occur in D, is background knowledge: we already know the words of the language, but we want to learn from valid sentences how the words may be ordered. For the set of our hypotheses $\mathcal{H}$ we take all context-free grammars (introduced as "phrase structure grammars" by Chomsky in [19]) that use only words from Σ and and additional set N of nonterminal symbols. N contains a special symbol called the *starting symbol.* A grammar is a set of production rules, each of which maps a nonterminal symbol to a (possibly empty) sequence consisting of both other nonterminals and words from Σ. A sentence is grammatical if it can be produced from the starting symbol by iteratively applying a production rule to one of the matching nonterminal symbols.

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 $\lceil \log |A| \rceil$. (Throughout this thesis, $\lceil \cdot \rceil$ denotes rounding up to the nearest integer, and log denotes binary logarithm. Such notation is listed on page 201.) 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. This way we need $|D| \lceil \log(|\Sigma| + 1) \rceil$ bits to encode the data. We are looking for grammars H which allow us to compress the data beyond this baseline value.

We first 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 $\lceil \log(|N \cup \{\diamond\} \cup \Sigma|) \rceil$ bits for each of the other symbols. The $\diamond$ symbol signals the end of each rule; two consecutive $\diamond$s signal the end of the entire grammar. If the grammar H has r rules, and the summed length of the replacement sequences is s, then we can calculate that the number of bits we need to encode the entire grammar is at most

$$(s + r) \lceil \log(|N| + |\Sigma| + 1) \rceil + (r + 1) \lceil \log(|N| + 1) \rceil. \tag{1.3}$$

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 each sentence and to mark the end of the data.

Now we consider two very simple correct grammars, both of which only need one nonterminal symbol S. The "promiscuous" grammar (terminology due to Solomonoff [84]) has rules S $\to$ S S and 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 $s = 2 + |\Sigma|$ 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 $\lceil \log(|\Sigma| + 1) \rceil$ 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 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, for both grammars $H \in \{H_1, H_2\}$, we can say that we do not compress the data at all, since in both cases 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 (1.1), 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 algorithmic aspects of finding the best hypothesis in an enormous hypothesis space is mostly outside the scope of this thesis: only in Chapter 6 we address this search problem explicitly.

The Road Ahead

In the remainder of this chapter we describe the basics of MDL inference, with special attention to *model selection*. Model selection is an instance of MDL inference where the hypotheses are sets of probability distributions. For example, suppose that Alice claims that the number of hairs on people's heads is Poisson distributed, while Bob claims that it is in fact geometrically distributed. After gathering data (which would involve counting lots of hair on many heads in this case), we may use MDL model selection to determine whose model is a better description of the truth.

In the course of the following introduction we will come across a number of puzzling issues in the details of MDL model selection that warrant further investigation; these topics, which are summarised in Section 1.4, constitute the subjects of the later chapters of this thesis.

1.1 Encoding Hypotheses

We return to the three questions of the previous section. The intuition behind the MDL principle was that "useful" hypotheses should help compress the data. For now, we will consider hypotheses that are "useful to compress the data" to be "useful" in general. We postpone further discussion of the the third question: what this means exactly, to Section 1.3. 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 needs only be *one* that, by sheer force of luck, allows for compression of the data. 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.

The next section is concerned with the definition of $L_H(D)$, but for the time being 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 thesis we use only length functions that correspond to prefix codes; we will explain what this means in Section 1.2.1.

Consider the case where the best candidate hypothesis, i.e. the hypothesis $\hat{H} = \arg\min_{H\in\mathcal{H}} L_H(D)$, 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 $L_{\mathrm{mdl}}(D)$ 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

$$\mathcal{R}(L, \mathcal{M}, D) \;:=\; L(D) - \inf_{L'\in\mathcal{M}} L'(D). \tag{1.4}$$

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 $\mathcal{R}(L_{\text{mdl}}, \{L_H : H \in \mathcal{H}\}, D)$ is small, *whatever data we observe.* This ensures that whenever $\mathcal{H}$ contains a useful hypothesis that allows for significant compression of the data, we are able to 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 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 1.2.1.)

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 $\hat{H}$ to exhibit good performance on future data unless we were able to compress the data using L_{mdl}.

1.1.1 Luckiness

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 hypotheses are especially small, at the small cost of increasing the code lengths for the other hypotheses by a negligible amount. This can be desirable, because *if* a hypothesis in $\mathcal{H}'$ turns out to be useful, then we are lucky, and we can 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 [39]; [6] 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. To keep things simple 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, $|\Sigma| = 500$ terminals, $r = 100$ rules and replacement sequences summing to a total length of $s = 2,000$. Using (1.3) we can calculate the luckiness code length for such a grammar as at most $\lceil 2100 \log(521) + 101 \log(21) \rceil = 19397$ bits.

Now we will investigate what happens if we use a *uniform* code on all possible grammars $\mathcal{H}'$ of up to that size instead. One may or may not want to verify that

$$|\mathcal{H}'| = \sum_{r=1}^{100} |N|^r \sum_{s=0}^{2000} (|N| + |\Sigma|)^s \binom{s+r-1}{s}.$$

Calculation on the computer reveals that $\lceil \log(|\mathcal{H}'|) \rceil = 19048$ bits. Thus, we compress the data 331 bits better than the code that we used before. While this shows that there is room for improvement of the luckiness code, the difference is actually not very large compared to the total code length. On the other hand, with the uniform code we *always* need 19048 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 $\lceil 110 \log(521) + 11 \log(21) \rceil = 1042$ bits to describe that grammar and therefore outcompresses the uniform code by 18006 bits: in that case we learn a lot more with the luckiness code.

Now that we have computed the minimal worst-case regret we have a target for improvement of 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 to use the luckiness code or the uniform code on the data.

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!

1.1.2 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 typically 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 monotonically 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 nonincreasing code that has the shortest code word for hypothesis n is uniform on the set $\{H_1, \ldots, H_n\}$ (and 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 [68] (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.

1.1.3 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, which can be interpreted as a language in which computer programs are expressed. 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 [60].

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 [8, 1]. To summarise, our version of ideal MDL tells us to pick

$$\min_{H \in \mathcal{H}} K(H) + L_H(D), \tag{1.5}$$

which is (1.1), except that now $\mathcal{H}$ is the set of *all* hypotheses represented by computable length functions, and $L(H) = K(H)$. (Note that $L_H(D) \approx K(D|H)$ iff D is $P(\cdot|H)$-random.)

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

$$\exists c > 0 : \forall H \in \mathcal{H} : K(H) \leq L(H) + c. \tag{1.6}$$

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. 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.

On the other hand, performance guarantees such as (1.6) 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 [91], 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 appreciated as a theoretical ideal that can serve as an inspiration for the development of methods that can be applied in practice.

1.2 Using Models to Encode 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. The problem of model selection is central to this thesis, 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.

1.2.1 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 [25, 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 1.2.1 (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 [25, page 82]. □

If the inequality is strict, 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

$$\forall x \in \mathcal{X} : \qquad P(x) := 2^{-L(x)}. \tag{1.7}$$

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

$$\forall x \in \mathcal{X} : \qquad L(x) := \lceil -\log P(x) \rceil. \tag{1.8}$$

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

$$H(P) := \sum_{x \in \mathcal{X}} -P(x) \log P(x). \tag{1.9}$$

(This H should not be confused with the H used for hypotheses.) According to Shannon's noiseless coding theorem [79], 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 (1.8) stays within one bit of entropy. This bit is a consequence of the requirement that code lengths have to be integers.

Note that apart from rounding, (1.7) and (1.8) 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 [76, 12]: for example, the frequentist school of thought holds that probability only carries any meaning 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. According to the Bayesian school on the other hand, a probability can also express a degree of belief in a certain proposition, even outside the context of a repeatable experiment. In both cases, specifying the probability of an event usually means making a statement about (ones beliefs about) the true state of nature. This is problematic, because people of necessity often work with very crude probabilistic models, which everybody agrees have no real truth to them. In such cases, probabilities are used to represent beliefs/knowledge which one a priori knows to be false! Using code lengths allows us to avoid this philosophical can of worms, because code lengths do not carry any such associations. Rather, codes are usually judged on the basis of their performance in practice: a good code achieves a short code length on data that we observe in practice, whether it is based on valid assumptions about the truth or not. We embrace this "engineering criterion" because we feel that inference procedures should be motivated solely on the basis of guarantees that we can give with respect to their performance in practice.

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, 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 [REF] which can usually be applied to achieve the ideal code length to within a few bits; this is outside the scope of this thesis because for MDL inference we only have to compute code lengths, and not actual codes.

Example 4. In Section 1.1.2 we remarked that a good code for the natural numbers always achieves code length close to $\log n$. Consider the distribution $W(n) = f(n) - f(n+1)$. If $f : \mathbb{N} \to \mathbb{R}$ is a decreasing function with $f(1) = 1$ and $1/f \to 0$, then W is an easy to use probability mass function that can be used as a prior distribution on the natural numbers. 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 $f(n) = 1/\log(n+1)$.

1.2.2 Sequential Observations

In practice, model selection is often used in a *sequential* setting, where rather than observing one outcome from some space $\mathcal{X}$, we keep making more and more observations $x_1, x_2, \ldots$. Here we discuss how probability theory can be applied to such a scenario. There are in fact three different ways to do it; while the first is perhaps easiest to understand, the other two will eventually be used as well so we will outline all of them here. This section can be skipped by readers who find themselves unconfused by the technicalities of sequential probability. We consider only countable outcome spaces for simplicity; the required definitions for uncountable outcome spaces are given later as they are needed.

A first approach is to define a *sequence* of distributions $P^1, P^2, \ldots$ Each P^n is then a distribution on the n-fold product space $\mathcal{X}^n$ which is used in reasoning about outcome sequences of length n. Such a sequence of distributions defines a random process if it is *consistent*, which means that for all $n \in \mathbb{N}$, all sequences $x^n \in \mathcal{X}^n$, the mass function satisfies $P^n(x^n) = \sum_{x \in \mathcal{X}} P^{n+1}(x^n, x)$. With some abuse of notation, we often use the same symbol for the mass function and the distribution; we will sometimes explicitly mention which we mean and sometimes consider it clear from context.

A mathematically more elegant solution is to define a distribution on infinite sequence of outcomes $\mathcal{X}^\infty$, called a *probability measure.* A sequence of outcomes $x^n \in \mathcal{X}^n$ can then be interpreted as an event, namely the set of all infinite sequences of which x^n is a prefix; the marginal distributions on prefixes of length $0, 1, \ldots$ are automatically consistent. With proper definitions, a probability measure on $\mathcal{X}^\infty$ uniquely defines a random process and vice versa. Throughout this thesis, wherever we talk about "distributions" on sequences without further qualification we actually mean random processes, or equivalently, suitably defined probability measures.

A random process or probability measure defines a *predictive distribution* $P(X_{n+1}|x^n) = P(x^n; X_{n+1})/P(x^n)$ on the next outcome given a sequence of previous observations. A third solution, introduced by Dawid [26, 29, 28] is to turn this around and start by specifying the predictive distribution, which in turn defines a random process. Dawid defines a *prequential forecasting system* (PFS) as an algorithm which, when input any initial sequence of outcomes x^n, issues a probability distribution on the next outcome. The total "prequential" probability of a sequence is then given by the *chain rule*:

$$\begin{aligned} P(x^n) &= P(x^1) \cdot \frac{P(x^2)}{P(x^1)} \cdots \frac{P(x^n)}{P(x^{n-1})} \\ &= P(x_1) \cdot P(x_2|x^1) \cdots P(x_n|x^{n-1}). \end{aligned} \tag{1.10}$$

This framework is simple to understand and actually somewhat more powerful than that of probability measures. Namely, if a random process assigns probability zero to a sequence of observations x^n, then the predictive distribution $P(X_{n+1}|x^n) = P(x^n; X_{n+1})/P(x^n) = 0/0$ is undefined. This is not the case for a PFS which *starts* by defining the predictive distributions. This property of FPSs becomes important in Chapters 4 and 5, because there we consider predictions made by *experts*, who may well assign probability zero to an event that turns out to occur!

1.2.3 Model Selection and Universal Coding

Now that we have identified the link between codes and probability distributions it is time to address the question which codes we should choose to represent hypotheses. In case that the hypothesis is given in terms of a probability distribution or code from the outset, no more work needs to be done and we can proceed straight to doing MDL model selection as described at the start of this chapter. Here we consider instead the case where the hypothesis is given in the form of a *model* $\mathcal{M} = \{P_\theta : \theta \in \Theta\}$, which is a set of random processes or probability measures 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. Now it is no longer immediately clear which single code should represent the hypothesis.

To motivate the code $L_{\mathcal{M}}$ that represents such a model, we apply the same reasoning as we used in Section 1.1. 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 first introduce a function $\hat{\theta} : \mathcal{X}^* \to \Theta$ called the *maximum likelihood estimator*, which is defined by:

$$\hat{\theta}(x^n) := \arg\max_{\theta \in \Theta} P_\theta(x^n). \tag{1.11}$$

(The following can be extended to the case where arg max is undefined, but we omit the details here.) Obviously, the maximum likelihood element of the model also minimises the code length. We prefer to abbreviate $\hat{\theta} = \hat{\theta}(x^n)$, if the sequence of outcomes x^n is clear from context.

Definition 1.2.2. Let $\mathcal{M} := \{L_\theta : \theta \in \Theta\}$ 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

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

This is quite different from the standard definition of universality [25], because it is formulated in terms of individual sequences rather than expectation. Also it is a very general formulation, 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 introduce a distribution 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 $f(\hat{\theta}, n) = \frac{k}{2} \log \frac{n}{2\pi} + g(\hat{\theta})$, 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.

We have now seen that in MDL model selection, universal codes are used on two levels: on one level, each model is represented by a universal code. Then another universal code (a two-part code, see below) is used to combine them into a single code for the data.

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

Two-Part Codes

In Section 1.1 we defined a code L_{mdl} that we now understand to be universal for the model $\{L_H : H \in \mathcal{H}\}$. 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 source symbol. In that case one must ensure that the encoding function is well-defined by specifying exactly which representation is associated with each source word D. Since we are only concerned with code *lengths* however, it suffices to adopt the convention that we always use one of the shortest representations.

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 in the set $\{0/n, 1/n, \dots, n/n\}$. 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. For the data we can use the code corresponding to θ. The total code length is minimised by ML distribution, so that we know that the regret is always exactly $\log(n+1)$; by using slightly cleverer discretisation we can bring this regret down even more such that it grows as $\frac{1}{2}\log n$, which, as we said, is usually achievable for uncountable single parameter models.

The Bayesian Universal Distribution

Let $\mathcal{M} = \{P_\theta : \theta \in \Theta\}$ 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 the field of Bayesian statistics, inference is always based on this joint distribution. We may, for example, calculate $P(\mathbf{x^n})$, where $\mathbf{x^n} = \{(x^n, \theta) : \theta \in \Theta\}$ denotes the event in the joint space that a particular sequence of outcomes x^n is observed. Second, we can calculate how we should update our beliefs about the state of the world after observing outcome x^n. Let $\boldsymbol{\theta} = \{(x^n, \theta) : x^n \in \mathcal{X}^n\}$ denote the event in the joint space that θ is true. Then we have:

$$P(\boldsymbol{\theta}|\mathbf{x^n}) = \frac{P(\mathbf{x^n} \cap \boldsymbol{\theta})}{P(\mathbf{x^n})} = \frac{P(\mathbf{x^n}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathbf{x^n})}. \tag{1.12}$$

This result is called *Bayes' rule*; its importance to inference stems from the idea that it can be used to update beliefs about the world W on the basis of new observations x^n. The conditional distribution on the hypotheses is called the *posterior distribution*; with considerable abuse of notation it is often denoted $W(\theta \mid \mathbf{x^n})$.

Note that the marginal distribution satisfies:

$$-\log P(\mathbf{x^n}) = -\log \sum_{\theta \in \Theta} P_\theta(\mathbf{x^n}) w(\theta) \leq -\log P_{\hat\theta}(\mathbf{x^n}) - \log W(\hat\theta),$$

where $\hat\theta$ is the maximum likelihood estimator, the element of $\mathcal{M}$ that minimises the code length for the data. Thus we find that if we use a Bayesian universal code, we obtain 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. 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 h and $t = n - h$ denote the number of heads and tails in x^n, respectively. Now we can calculate the Bayes marginal likelihood of the data:

$$P_{\text{bayes}}(x^n) = \int_0^1 P_\theta(x^n) \cdot 1 \;\mathrm{d}\theta = \int_0^1 \theta^h (1-\theta)^t \,\mathrm{d}\theta = \frac{h!\,t!}{(n+1)!}.$$

Using Stirling's approximation of the factorial function, we find that the corresponding code length $-\log P_{\text{bayes}}(x^n)$ equals $-\log P_{\hat\theta}(x^n) + \frac{1}{2}\log n + O(1)$. Thus we find roughly the same regret as for a well-designed two-part code.

Prequential Universal Distributions

An algorithm that, given a sequence of previous observations x^n, issues a probability distribution on the next outcome $P(X_{n+1}|x^n)$, is called a *prequential forecasting system* (PFS) [26, 29, 28]. A PFS defines a random process by the chain rule (1.10). Vice versa, for any random process P on $\mathcal{X}^\infty$, we can calculate the conditional distribution on the next outcome $P(X_{n+1} \mid X^n = x^n) = P(x^n; X_{n+1})/P(x^n)$, provided that $P(x^n)$ is positive. This so-called *predictive distribution* of a random process defines a PFS. We give two important prequential universal distributions here.

First, we may take a Bayesian approach and define a joint probability measure on $\mathcal{X}^\infty \times \Theta$ based on some prior distribution W. As before, this induces a marginal probability measure on $\mathcal{X}^\infty$ which in turn defines a PFS. In this way, the Bayesian universal distribution can be reinterpreted as a prequential forecasting system.

Second, since the Bayesian predictive distribution can be hard to compute it may be useful in practice to define a forecasting system that uses a simpler algorithm. Perhaps the simplest option is to predict an outcome X_{n+1} using the maximum likelihood estimator for the previous outcomes $\hat{\theta}(x^n) = \arg\max_\theta P_\theta(x^n)$. We will use this approach in our running example.

Example 5 (continued)**.** The ML estimator for the Bernoulli model parameterised by the probability of observing heads, equals the frequency of heads in the sample: $\hat{\theta} = h/n$, where h denotes the number of heads in x^n as before. We define a PFS through $P(X_{n+1}|x^n) := P_{\hat{\theta}}$. This PFS 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 $\tilde{\theta} = (h+1)/(n+2)$.

Perhaps surprisingly, in this case the resulting PFS 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. In general, the distribution indexed by such a "tweaked" ML estimator may be quite different from the Bayesian predictive distribution.

Although the prequential ML code has been used successfully in practical inference problems, the model selection experiments in Chapter 2 show that usage of PFSs based on estimators such as the "tweaked" ML estimator from the example, leads to significantly worse results than those obtained for the other three universal codes. This is the subject of Chapter 3, where we show that in fact the prequential ML code does achieve the same asymptotic regret as the other universal codes, *provided* that the distribution that generates the data is an element of the model. Under misspecification the prequential ML code has different behaviour that had not been observed before.

Normalised Maximum Likelihood

The last universal code we discuss is the one preferred in the MDL literature, because if we fix some sample size n in advance, it provably minimises the worst-case regret. It turns out 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. This is precisely what the Normalised Maximum Likelihood (NML) distribution achieves:

$$P_{\text{nml}}(x^n) := \frac{P_{\hat{\theta}(x^n)}(x^n)}{\sum_{y^n \in \mathcal{X}^n} P_{\hat{\theta}(y^n)}(y^n)}. \tag{1.13}$$

For all sequences 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:

$$\inf_L \sup_{x^n} \mathcal{R}(L, \mathcal{M}, x^n) = \log \sum_{y^n \in \mathcal{X}^n} P_{\hat{\theta}(y^n)}(y^n) \tag{1.14}$$

Under some regularity conditions on the model it can be shown [72, 39] that there is a particular continuous function g such that the parametric complexity is less than $\frac{k}{2} \log \frac{n}{2\pi} + g(\hat{\theta})$, as we required in Section 1.2.3. We return to our Bernoulli example.

Example 5 (continued)**.** The parametric complexity (1.14) 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:

$$\log \sum_{y^n \in \mathcal{X}^n} P_{\hat{\theta}(y^n)}(y^n) = \log \sum_{h=0}^{n} \binom{n}{h} \left(\frac{h}{n}\right)^h \left(\frac{n-h}{n}\right)^{n-h}. \tag{1.15}$$

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.

The NML distribution has a number of significant practical problems. First, it is often undefined, because for many models the numerator in (1.13) is infinite, even for such simple models as the model of all Poisson or geometric distributions. Second, $|\mathcal{X}^n|$ may well be extremely large, in which case it may be impossible to actually calculate the regret. In a sequential setting, the number of terms grows exponentially with the sample size so calculating the NML probability is hopeless except in special cases such as the Bernoulli model above, where mathematical trickery can be applied to get exact results (for the multinomial model, see [50].) Chapter 2 addresses the question of what can be done when NML is undefined or too difficult to compute.

1.3 Applications 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 machine 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, which is designed to achieving compression of the data, is also suitable to some extent in settings with a different objective, so that, at least to some extent, MDL provides a reliable one-size-fits-all solution.

1.3.1 Truth Finding

We have described models as formal representations of hypotheses; truth finding is the process of determining which of the hypotheses is "true", in the sense that the corresponding model contains the data generating process. The goal is then to use the available data to identify this true model on the basis of as little data as possible.

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.

Theorem 1.3.1 (Model Selection Consistency). *Let $\mathcal{H} = \{\mathcal{M}_1, \mathcal{M}_2, \ldots\}$ be a countably infinite set of parametric models. For all $n \in \mathbb{Z}^+$, let $\mathcal{M}_n$ be 1-to-1 parameterised by $\Theta_n \subseteq \mathbb{R}^k$ for some $k \in \mathbb{N}$ and define a prior density w_n on Θ_n. Let $W(i)$ define a prior distribution on the model indices. We require for all integers $j > i > 0$ that with w_j-probability 1, a distribution drawn from Θ_j is mutually singular with all distributions in Θ_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

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

Proof. Proofs of various versions of this theorem can be found in [5, 28, 4, 39] and others. □

The theorem uses Bayesian codes, but as conjectured in [39] and partially in Chapter 5, it can probably be extended to other universal codes such as NML. Essentially it expresses that *if* we are in the ideal situation where one of the models contains the true distribution, we are guaranteed that this model will be selected once we have accumulated sufficient data. This property of model selection criteria is called *consistency.* Unfortunately, this 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, it is not known under what circumstances the model selection criterion δ selects the model that contains the distribution $P_{\tilde{\theta}}$ minimising the Kullback-Leibler divergence $D(P^* \| P_{\tilde{\theta}})$. 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 1.3.1. Therefore consistency of MDL and Bayesian model selection is reassuring.

1.3.2 Prediction

The second application of model selection is prediction. Here the models are used to construct a *prediction strategy*, which is essentially the same as a prequential forecasting system: an algorithm that issues a probability distribution on the next outcome given the previous outcomes. There are several ways to define a prediction strategy based on a set of models; we mostly consider what is perhaps the most straightforward method, namely to apply a model selection criterion such as MDL to the available data, and then predict according to some estimator that issues a distribution on the next outcome based on the selected model and the sequence of previous outcomes.[1]

The performance of a prediction strategy based on a set of models is usually analysed in terms of the *rate of convergence*, which expresses how quickly the distribution issued by the prediction strategy starts to behave like the data generating process P^*. From the information inequality ($D(P^* \| Q) \geq 0$ with equality for $Q = P^*$) we know that P^* itself is optimal for prediction. The expected discrepancy between the number of bits needed by the prediction strategy to encode the next outcome and the number of bits needed by the true distribution P^* is called the *risk*; a prediction strategy has a good rate of convergence if the risk goes down quickly as a function of the sample size. A number of results that give explicit guarantees about the risk convergence rate for prediction strategies based on the MDL model selection criterion are given in [39]. Interestingly, the

[1]This strategy does not take into account the predictions of any of the other models, while at the same time we can never be sure that the selected model is actually the best one. It is better to take a weighted average of the predictions of the models in question; the most common practice is to weight the models by their Bayesian posterior probability, a procedure called *Bayesian model averaging*; see Chapter 5.

risk may converge to zero even if P^* is not an element of any of the models under consideration. This is useful in practical applications: for example, if the models correspond to histogram densities with increasing numbers of bins, then the risk converges to zero if P^* is described by any bounded and continuous density; see Chapter 5 for more details.

Roughly speaking, two groups of model selection criteria can be distinguished in the literature. Model selection criteria of the first group have often been developed for applications of prediction; criteria such as Akaike's Information Criterion (AIC) and Leave One Out Cross-Validation (LOO), exhibit a very good risk convergence rate, but they can be inconsistent, i.e. they keep selecting the wrong model regardless of the available amount of data. The second group contains criteria such as the Bayesian Information Criterion (BIC), as well as regular Bayesian/MDL model selection, which can be proven consistent but which are often found to yield a somewhat worse rate of convergence. It is a long standing question whether these two approaches can be reconciled [103].

An assumption that is implicitly made when MDL or Bayesian model selection are used for prediction, is that there is one model that exhibits the best predictive performance from the start, and our only task is to identify that model. However, in Chapter 5 we argue that this approach is in fact often too simplistic: in reality, which model can provide the best predictive performance may well *depend on the sample size*. In the histogram density estimation example from above, we typically do not expect any particular model to contain the true distribution; instead we expect to use higher and higher order models as we gain enough data to estimate the model parameters with sufficient accuracy to make good predictions.

In Chapter 5 we attempt to combine the strong points of the two different approaches to model selection by dropping the assumption that one model has best performance at all sample sizes. This results in a modification of MDL and Bayesian model selection/averaging, called the *switch-distribution*, that achieves better predictive performance, *without sacrificing consistency.* Note that the switch-distribution can be used to improve MDL model selection as well as Bayesian model selection, although the idea may not agree very well with a Bayesian mindset, as we will discuss.

The switch-distribution ties in with existing universal prediction literature on *tracking the best expert*, where the assumption that one predictor is best at all sample sizes has already been lifted [94, 46, 95]. In the past, that research has not been connected with model selection though. In Chapter 6 we describe how the switch-code lengths can be computed efficiently, and introduce a formalism that helps to define other practical methods of combining predictors. These ideas help understand the connection between many seemingly very different prediction schemes that are described in the literature.

1.3.3 Separating Structure from Noise

Algorithmic Rate-Distortion Theory is a generalisation of ideal MDL. Recall from 1.1.3 that in ideal MDL, Kolmogorov complexity is used as the code length function for the hypothesis $L(H) = K(H)$; model selection is then with respect to the set $\mathcal{H}$ of all computable hypotheses. This approach is generalised by introducing a parameter α, called the *rate*, that is used to impose a maximum on the complexity of the hypothesis $K(H)$. The selected hypothesis H, and the code length it achieves on the data $L_H(D)$, now become functions of α. The latter function is called the *structure function*:[2]

$$h_D(\alpha) = \min_{H \in \mathcal{H}, K(H) \leq \alpha} L_H(D) \tag{1.16}$$

We write $H(\alpha)$ for the hypothesis that achieves the minimum at rate α. We use dotted symbols $\doteq$, $\dot{<}$ and $\dot{\leq}$ for (in)equality up to an independent additive constant; the dot may be pronounced "roughly".

A hypothesis H is called a *sufficient statistic* if $K(H) + L_H(D) \doteq K(D)$. Intuitively, such hypotheses capture all simple structural properties of the data. Namely, if there are any easily describable properties of D that are not captured by H, then we would have $K(D|H) \dot{<} L_H(D)$. But in that case $K(D) \dot{\leq} K(H) + K(D|H) \dot{<} K(H) + L_H(D)$ which is impossible for sufficient statistics H.

If the rate is high enough, certainly for $\alpha = K(D)$, the hypothesis $H(\alpha)$ is a sufficient statistic. The most thorough separation of structure and noise is obtained at the lowest rate at which $H(\alpha)$ is a sufficient statistic. At even lower rates, some structural properties of D can no longer be represented; the structure function can then be used as an indicator of how much structure has been discarded.

Algorithmic rate-distortion theory generalises even further by introducing a distortion function, which allows expression of the kind of properties of the original object we consider important. It is an analogue of Shannon's classical rate-distortion theory [79, 37]. The novelty of algorithmic rate-distortion theory compared to classical rate-distortion theory is that it allows analysis of *individual objects* rather than expected properties of objects drawn from a source distribution. This is useful because it is often impossible to define a source distribution that is acceptable as a reasonable model of the process of interest. For example, from what source distribution was Tolstoy's War and Piece drawn?

In Chapter 6 we introduce this new theory of algorithmic rate-distortion in more detail, and then proceed to put it to the test. To obtain a practical method, we approximate Kolmogorov complexity by the compressed size under a general purpose data compression algorithm. We then compute the rate-distortion characteristics of four objects from very different domains and apply the results to

[2]In Kolmogorov's original formulation, the codes L_H for $H \in \mathcal{H}$ were uniform on finite sets of possible outcomes.

denoising and lossy compression. The chapter is written from the point of view of algorithmic rate-distortion theory, but includes a discussion of how the approach relates to MDL as we describe it in this introduction.

1.4 Organisation of this Thesis

In this introduction we have outlined MDL model selection. An important ingredient of MDL model selection is universal coding (Section 1.2.3). We have described several approaches; the universal code preferred in the MDL literature is the code that corresponds to the Normalised Maximum Likelihood (NML) distribution (1.13), which achieves minimal regret in the worst case. However it turns out that the NML distribution is often undefined. In those cases, the way to proceed is not agreed upon. In Chapter 2, we evaluate various alternatives experimentally by applying MDL to select between the Poisson and geometric models. The results provide insight into the strengths and weaknesses of these various methods.

One important result of the experiments described in Chapter 2 is that the prequential forecasting system, one of the universal models introduced in Section 1.2.3), defined to issues predictions on the basis of the maximum likelihood distribution, achieves a regret very different from that achieved by other models. This leads to inferior model selection performance. This was unexpected, because in the literature, e.g. [71], the prequential ML universal model is described as a practical approximation of other universal codes, certainly suitable for model selection. In Chapter 3 we analyse the regret of the prequential ML universal distribution under misspecification, that is, the data are drawn from a distribution outside the model. The behaviour under misspecification is important to model selection, since there the data generating distribution is often in only one of the considered models (if that many). Together, chapters 2 and 3 form the first leg of this thesis.

The second leg involves the relationship between model selection and prediction (see Section 1.3). It is based on the observation that the model that allows for the best predictions may *vary with the sample size*. For example, simple models with only few parameters tend to predict reasonably well at small sample sizes while for large models often a lot of data is required before the parameters can be estimated accurately enough to result in good predictions. While this may seem obvious, we show that the codes that are typically used in Bayesian and MDL model selection do not take this effect into account and thus the achieved code lengths are longer than they need to be. In Chapter 4 we describe a number of alternative ways to combine the predictions of multiple codes. It is also shown how the resulting code lengths can still be computed efficiently. The discussion expands on results in the source coding and universal prediction. One of the described codes, called the *switch-code*, is specifically designed to improve model selection results. In Chapter 5 we show how model selection based on the switch-code, while remaining consistent, achieves the best possible rate of convergence. This resolves a perceived distinction between consistent model selection criteria on the one hand, and criteria that are suitable for prediction, which achieve a fast rate of convergence, on the other hand.

In Chapter 6, the third leg of this thesis, we put the new algorithmic rate-distortion theory to the test. Algorithmic rate-distortion theory is introduced in Section 1.3.3, more thoroughly in Chapter 6 itself, and even more thoroughly in [92]. It is a generalisation of ideal MDL (Section 1.1.3) and a variation on classical rate-distortion theory [79]; it allows for analysis of the rate-distortion properties of individual objects rather than of objects drawn from some source distribution. Algorithmic rate-distortion theory uses Kolmogorov complexity which is uncomputable. To obtain a practical method, we approximate Kolmogorov complexity by the compressed size under a general purpose data compression algorithm. We then compute the rate-distortion characteristics of four objects from very different domains and apply the results to denoising and lossy compression.