Tech Report 207
Department of Computer Science, Monash University, Clayton, Vic. 3168, Australia

MDL and MML : Similarities and Differences

(Introduction to Minimum Encoding Inference — Part III)
Amended- March 1995

ROHAN A. BAXTER (rohan@cs.monash.edu.au)
JONATHAN J. OLIVER (jono@cs.monash.edu.au)
Computer Science Department
Monash University
Clayton, Victoria, 3168, AUSTRALIA

Abstract: This paper continues the introduction to minimum encoding inductive inference given by Oliver and Hand. This series of papers was written with the objective of providing an introduction to this area for statisticians. We describe the message length estimates used in Wallace's Minimum Message Length (MML) inference and Rissanen's Minimum Description Length (MDL) inference. The differences in the message length estimates of the two approaches are explained. The implications of these differences for applications are discussed.

Contents

1 Introduction

The Minimum Description Length (MDL) and the Minimum Message Length (MML) principles for model selection and inductive inference are not often distinguished in the literature. A typical example is the following passage from MacKay

> "Although some of the earliest work on complex model comparison involved the MDL framework [PW82], MDL has no apparent advantages, and in my work I approximate the evidence directly" [Mac92, page 17].

Here MacKay has classified the MML results of Patrick and Wallace as MDL. The confounding of the two approaches is understandable because both, at least originally, involved coding the parameters of models and then choosing the model with the shortest message containing a description of the model and the data.

Part of the confusion can be traced to their related origins. MDL's most cited source paper, Rissanen [Ris78] , which derives the $\frac{k}{2}\log_2 n$ complexity term, references MML's most cited source paper, Wallace and Boulton [WB68], and the message length derivation there is preceded by the phrase

> "Following a suggestion by Boulton and Wallace (private correspondence)" [Ris78, page 470].

Wallace, Freeman and Rissanens' Royal Statistical Society meeting review papers on MML and MDL also appeared side by side in 1987 [Ris87, WF87].

In the following, we use the term MDL to cover a number of different message length estimates as espoused by Rissanen [Ris87, Ris89]. MDL message length estimates are widely used in many areas of data modelling and artificial intelligence. MDL codes include two part messages and one part messages. The asymptotic equivalence of the different MDL codes justifies the use of a single name [Ris87].

MML estimates are two part message length estimates developed by Wallace and co-workers (see e.g., [WB68, WB75, WF87]). MML estimates have also had many applications.

In this paper, we use the term *message length* specifically to mean the length of a bit string which describes a theory and data. The term *code length* is used to mean the length of a message which describes a single datum or parameter. A code length may be fractional— see Oliver and Hand [OH94, Section 2.8] for details. In this paper we consider message length estimates for sufficiently regular models[1] and identically and independently distributed data. Message length estimates can, of course, be made for other cases, see Wallace and Freeman [WF92] for an example.

2 Differences

The MML and MDL principles for inductive inference are similar, but distinct. We state these principles, but first define the terms: model and model class. In any inductive inference problem, a set of models is entertained. This *model set* is often considered to be the union of several subsets called *model classes*. Typically a model class is a set of models with the same parametric form, including the same number of parameters[2]. For example, we distinguish the model class of univariate cubics, $MC_3 : y = a_0 + a_1x + a_2x^2 + a_3x^3$ from the model class of univariate quadratics, $MC_2 : y = a_0 + a_1x + a_2x^2$. The model set of univariate polynomials is the union of the model classes, $MC_0 \bigcup MC_1 \bigcup MC_2 \bigcup ... \bigcup MC_k ..., k = 0, 1, 2, ..., \infty$. A *model* is an instance of a model class where all the parameters have been assigned values, e.g., $y = 0.4+3.0x+2.0x^2$. We take inductive inference to involve specifying a single element of the model set.

Now we state the MML and MDL principles:

[1] Sufficiently regular models are those where the log-likelihood function can be approximated well by its quadratic expansion.

[2] In the early MDL literature, a *model class* is almost always associated with an index α of the form (k, θ), where k is the number of components in the parameter vector $(\theta_1, ..., \theta_k)$. The later stochastic complexity literature does not use this indexing. One use of this indexing was to make the connection between MDL and maximum likelihood inductive inference [Ris83, page 418]. Maximum likelihood is used by MDL to choose between models within a model class when the number of parameters, k, is the same.

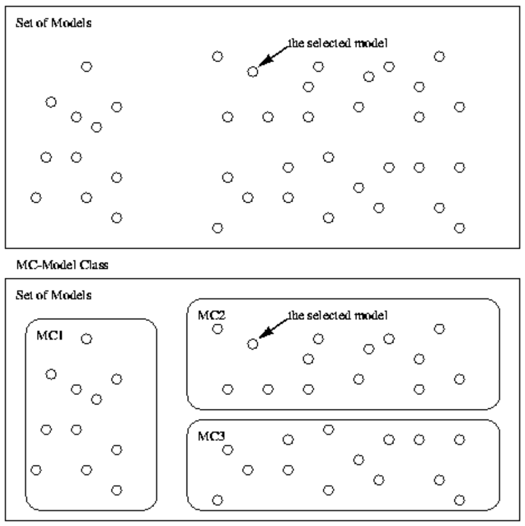

Figure 1: Models and Model Classes in Inductive Inference

MDL Principle [Ris89, page 9]

> "We may take it as an axiom that a model or a model class, which permits the shortest encoding of the data, captures best all the properties in the data we wish to learn". (In practice, we note MDL methods have concentrated on selecting model classes, rather than models).

MML Principle (paraphrasing [WB68, WF87])

> For each model in the set of models considered, we consider the message consisting of a description of a model, followed by the data encoded under the assumption that the model is true. The preferred model amongst a set of competing models will be the one with the shortest two part message length of this structure.

From these statements we see how the principles differ. MML inductive inference selects a model from the set of models, by considering a message which begins by naming that model out of all the models in the set. This is illustrated in the top half of Figure 1. In contrast, MDL inductive inference selects a model class, by considering a message which does not name the model class. The MDL message may, but need not, be two part. If the MDL message is two part, then it also names a model from the model class. In the bottom half of Figure 1, the model is selected from the model class $MC2$. If the MDL message is one part, then no single model is named and no model class is named. Instead all the models in the assumed model class are used to encode the data.

This difference arise from MML and MDL's respective formulations of the objectives for inductive inference. We summarize them as follows:

MDL Objectives: [Ris93a, page 13]

> "The main problem in statistical inference is and remains to find the best model class and the best model in it as judged by shortest code length..."

MML Objectives:

> The end product of inductive inference is the model that gives the shortest two part message length. This is the best explanation of the data. A model may consist of estimates of model parameters, perhaps representing real world quantities. Among other things, the model can be used for prediction.

MML is concerned with models. The end result of MML inductive inference is a model with specified parameter values[3]. MDL has an additional emphasis on 'data modelling', the prediction of new data using model classes. If MDL is used for prediction of new data, no selection of models or parameter estimates are necessarily needed in the message. This is discussed further in Section 2.1. Optimal codes for describing the data using an assumed model class do not have a two part structure:

> "If we seek the shortest message length relative to a model class, there is no compelling reason to encode any parameter values. The same defect can also be seen in the fact that [a two part coding system] is redundant in that each data sequence can be encoded with every parameter value, while in an efficient coding system there ought to be just one codeword for each data sequence"[Ris89, page 58].

The implication of using a two part structure is that the message length of some data using a single model from a model class will be greater than the message length of the data using the entire model class. This is easy to show:

Theorem: Without loss of generality, consider a model class MC with just two models M_1 and M_2. We prove that

$$MessLen(Data\ using\ M_i) > MessLen(Data\ using\ MC), i \in 1, 2$$

Proof: Assume M_1 is the model with the minimal message length. Its message length is calculated by adding the first and second parts of the message together

$$-\log Prob(M_1) - \log Prob(D|M_1)$$

The minimal message length of the data using the MC is a one part code of length:

$$-\log(Prob(M_1)Prob(D|M_1) + Prob(M_2)Prob(D|M_2))$$

We find that

$$\begin{aligned} -\log Prob(M_1) - \log Prob(D|M_1) \quad > \quad & -\log(Prob(M_1)Prob(D|M_1) + \\ & Prob(M_2)Prob(D|M_2)) \end{aligned} \tag{1}$$

Now since $Prob(M_i) > 0$ and $Prob(D|M_i) > 0$, the inequality holds always (by the convexity of the logarithm function). This proof is easily extended to model classes with more than two models. **QED**.

The one part code used to encode the data using MC has an optimal code length relative to a single model class. If we wish to identify a preferred model class from a family of model classes in an MML framework,

[3] This is not the whole story. In general, the end result of MML inductive inference is potentially a full posterior distribution on the discretised models as implied by the message lengths of the models.

we would require a prior distribution over model classes. The MDL framework implicitly assumes a uniform distribution over model classes by not considering this part of the code.

The shortest two part message cannot be expected to be as short as the shortest one part message encoding the data, because it encodes not just the data, but also a choice of model not implied by the data [WF87, page 260]. So while MML and MDL are often grouped under the 'shortest data description length', 'minimum description length' or 'minimum encoding inference' umbrella, we see that MML uses the slightly longer two part message structure. This explains why it is useful to have a different nomenclature for MDL and MML, where MML can be thought of as a contraction of 'Minimum two part Message Length'.

2.1 Stochastic Complexity: One Part Messages

MDL selects model classes by choosing the model class with minimum *stochastic complexity*. A model class is specified by choosing $Prob(D|\theta)$ and $Prob(\theta)$ [Ris89, page 54]. In a parametric probabilistic model class framework, we have:

$$Prob(D) = \int Prob(D|\theta)Prob(\theta)d\theta \tag{2}$$

where D is the data and θ are continuous model class parameters. The *stochastic complexity* (SC) of data D, relative to the model class of Equation (2) is [Ris89, page 59]:

$$SC = -\log Prob(D) \tag{3}$$

$Prob(D)$ is also called the evidence[Goo85] and the marginal likelihood[KR93]. Approximations to Equation (2) are the basis of the Bayesian inference methods of Schwarz[Sch78] and MacKay[Mac92]. These approaches are discussed in Oliver and Baxter[OB94].

If a single model within a model class needs to be chosen, then a non-coding principle, such as the maximum likelihood estimator, is used.

Prior distributions on parameters are used only insofar that they are needed to evaluate the stochastic complexity.

2.2 Interpretations of the Probability Distribution on Model Parameters

A philosophical difference arises between MDL and MML in the interpretation of $Prob(\theta)$ in Equation (2). MML interprets $Prob(\theta)$ as representing prior beliefs, or knowledge, or ignorance about the parameters, θ. $Prob(\theta)$ is interpreted as subjective probabilities. Hence MML selects $Prob(\theta)$ according to prior information, or, equivalently, to maximise its expected coding efficiency.

The MDL literature is characterised by a lack of acceptance of subjective probabilities [Ris83, Ris87, Ris89, LV93, Ris93b]:

> "In our view the parameter θ is generated by our selecting the model class, and it has no other 'inherent' meaning"[Ris89, page 54].

> "In the MDL principle for statistical inference there is no need for the awkward Bayesian interpretations of the meaning of the prior probability on the parameters. Rather, we may interpret distributions, such as $[Prob(D)]$, just as convex linear combinations of the models in the class, whose utility will be assessed on other grounds..."[Ris93b, page 12].

We interpret this as saying that $Prob(\theta)$'s utility is based on the resulting description length of the data. This is very similar to MML's interpretation. However while MML links consideration of prior information with minimizing expected description length, MDL explicitly ignores this link.

Two part messages are used in MDL as approximations to the stochastic complexity expression of Equation (3), where better one part estimates are not available or are computationally difficult to evaluate. The first part of the message encodes the parameter values. The optimal code length of a parameter is $-\log Prob(\theta)$. As we have seen, MDL does not use subjective priors, so what can it do? One approach used is a *universal code*[4]. The rationale for universal codes is to try to avoid biasing any particular parameter

[4]See Section (3.2.1) for a technical definition.

value and so only learn directly from the data. It is recognised that a universal code used by MDL is a prefix code and hence implies a prior probability distribution. The claim is then that this implied prior probability distribution has minimal bias.

The MDL approach to two part codes is examined in detail in Section 3. We will discuss MDL's use of the $\log *$ (log star) universal code for positive integers. Typically it is applied to coding a model indexed by a positive integer.

2.3 A Specific Objection to Priors

We now consider one specific objection to priors as used by MML

> "A similar code-length (similar to MDL) with a prior is discussed in Wallace and Freeman [WF87]. The authors advocate the principle of minimizing the mean code length in the spirit of Shannon's work, which strictly speaking does not allow it to be used to select the model class. Indeed, take a model class which assigns a probability $1-\epsilon$ to the string consisting of 0s only and the rest equally to all the remaining strings. For a small enough ϵ the mean relative to a model can be made as small as we like"[Ris89, page 56].

We interpret this claim as: if the true model is assigned a small enough prior probability relative to a fixed amount of data then MML will never correctly select it. In a formal Bayesian setting, the prior is assigned *before* seeing the data. In this case, even with an extreme prior which assigns probability ϵ to the true model, the MML procedure will recover the true model given sufficient data[BC91]. The MDL and MML treatment of priors is discussed in more detail in Dowe [Dow95]. Leaving aside the issue of the prior coming before the data, one essential difference between MDL and MML is the willingness to risk the scenario of an extreme prior, rather than lack of invariance. In Section 3 we show how the MDL two part message length procedure lacks invariance.

2.4 Prediction with MDL and MML

We take prediction of the value of a new data item, x, to require the specification of a probability density (or distribution) of possible values. Prediction with MDL uses the selected model class, giving the predictive density:

$$Prob(x|D) = \int Prob(x|D,\theta)Prob(\theta)d\theta \qquad (4)$$

Prediction with MML is often done using the best model, $\hat{\theta}$, and the probability $Prob(D|\hat{\theta})$:

$$Prob(x|D) = Prob(x|D,\hat{\theta})Prob(\hat{\theta}) \qquad (5)$$

This prediction is sub-optimal, but convenient. Computationally and cognitively it is often simpler to base prediction on a single model. More accurate prediction can also be done using a weighted combination of predictions from more than one model. The weights assigned to models are based on their message lengths [AWY92, OH96].

In both MDL and MML prediction, if we need to make a single-valued guess at the value of x, we use a loss function, $l(\hat{x}, x)$. The loss function expresses our judgement of how bad it is to guess $\hat{x}$ when the real value is x.

Consider the problem of estimating a density function which consists of a mixture of normal distributions [EH81]. We may consider each set of mixture distributions with the same number of components as a model class. The model class whose predictive code has the minimum length may not necessarily contain the 'best' single model. For example, given some data D, the model class of three component mixtures may be the model class which minimises the SC, but the mixture model of minimal message length (using a two part message) may be from the model class of two component mixtures. MDL suggests we estimate the mixture distribution (using maximum likelihood, for example) from the model class of three component mixtures; MML defines the mixture model of minimal message length (from the model class of two component mixtures) as the 'best' single model of the data.

3 Two Part Message Length Procedures

Two part messages have a two-pronged justification in MDL. Two part messages are used in MDL as approximations to the stochastic complexity expression of Equation (3), where better one part estimates are not available. Recently, algorithmic complexity theorists, Li and Vitanyi[LV93] have justified two part MDL messages as feasible approximations to Solmonoff inductive inference— see [BO95] for further discussion of this point.

MDL does not employ a prior probability distribution (or density) on parameter values. We now examine how MDL uses universal codes to estimate the optimal model code length.

We now describe the Universal Code MDL (UC-MDL) procedure[Ris83, Ris89, LV93]:

UC-MDL inductive inference procedure:

1. Partition the hypothesis space into regions with centres $\theta_i, i \in \{\mathcal{N}\}$ and volumes $AOPV(\theta_i)$, where AOPV stands for Accuracy of Parameter Value[OH94, Section 3]. In general, the size of a region may be a function of its location.
2. Reorder the indexing to meet criteria described in Section 3.1, so $f(i) = j, \{i, j \in \mathcal{N}\}$.
3. Note for each j, we have a probability model $Prob(D|\theta_j)$
4. Choose H_j to minimize
$$UnivCodeLength(j) - \log(Prob(D|H_j))$$
Note that this implicitly defines a prior distribution over hypotheses $Prob(H_i) = 2^{-UnivCodeLength(j)}$.

The UC-MDL procedure just described is incomplete in that we additionally need to specify:

- the discretization volumes $AOPV(H_i), \{i \in \mathcal{N}\}$.
- the ordering criterion from i to j
- $UnivCodeLength(x)$

Choosing the optimal discretization volume is an step in both MDL and MML inductive inference. The choice of the optimal discretization volume involves the balancing of the length of the first part of the message $(-\log Prob(H_i))$ with the length of the second part of the message $(-\log Prob(D|H_i))$. Further discussion of this issue can be found in [WB68, WB75, Ris83, WF87, WD93, OH94].

The point we wish to emphasize is that the UC-MDL inductive inference procedure is not fully specified until we specify f and the universal code to be used.

By way of contrast, we now describe the MML inductive inference procedure which instead adopts a Bayesian philosophy and uses a specific prior:

B-MML inductive inference procedure:

1. Same as for UC-MDL inductive inference problem
2. Note for each j, we have a probability model $Prob(D|\theta_j)$
3. Choose a $Prob(H)$
4. Choose H_i to minimize

$$-\log(\int_{AOPV(H_i)} Prob(H_i)d\theta) - \log Prob(D|H_i)$$

where θ is vector of parameters for model H_i. It is convenient to approximate the integral (for sufficiently small $AOPV(H_i)$) as follows:

$$-\log(Prob(H_i) \times AOPV(H_i)) - \log Prob(D|H_i)$$

The need to specify an optimal discretization volume is common to UC-MDL and B-MML.

However, the specification of a universal code and a mapping function in UC-MDL is replaced in B-MML by the specification of a prior probability distribution or density over the hypothesis space, $Prob(H)$ with $\{H \in \mathcal{H}\}$. Selecting an accurate, or even an approximate, form for $Prob(H)$ is a common problem in Bayesian inductive inference. One advantage over the UC-MDL formulation is that all the assumptions made are encapsulated in $Prob(H)$, rather than in both f and the universal code. We acknowledge that there is often uncertainty in the specification of a prior for B-MML, but believe this uncertainty does not usually result in a message length term of size $O(1)$. This is the case for UC-MDL.

In the next two subsections, we discuss difficulties arising from the selection of a mapping and a universal code.

3.1 Mapping of Limited Precision Real Numbers onto Natural Numbers

Consider a point (x, y) on a plane. Each coordinate could represent an estimate of a parameter of some model. For instance, x could be an estimate of the mean acidity (pH) of soil and y an estimate of the the mean of the logarithm of the ambient temperature measured in Kelvin. We would like to transmit one or more model estimates and estimate the total message length using the universal $\log *$ code for positive integers.

To use a universal code, we need to decide on a mapping from the parameter space to a positive integer[5]. The problem is that there are many mappings we could use. Each mapping implies a different prior probability distribution on the parameters.

3.1.1 Mapping Parameters onto Positive Integers

Let us start by considering how to do the mapping for a single limited precision real number, such as y in our soil example. We must state parameters to a limited precision [OH94, Section 3.0]. So we can convert y to an integer by dividing by the accuracy of parameter value. For example, the limited precision real 2.25 has $AOPV = .01$ and $\frac{2.25}{.01} = 225$.

Next consider how to map the resulting integer onto the positive integers. There are many ways to do this. An obvious way is the following:

```
Mapping 1:
Mapped Integer : 1   2   3   4   5   6   7   8   9   10 ...
Parameter Value: 0   1  -1   2  -2   3  -3   4  -4   5 ...
```

[5]This approach is similar to that advocated by Li and Vitanyi [LV93, page 310] and Rissanen [Ris89].

This mapping onto the $\log *$ code implies a probability distribution where positive integers are equally likely to negative integers *a priori*. If we believe *a priori* this is not the case, we could choose a different mapping to reflect this. For example, the following:

```
Mapping 2:
Mapped Integer : 1   2   3   4   5   6   7   8   9   10 ...
Parameter Value: 0   1   2   -1  3   4   -2  5   6   -3 ...
```

It seems that a criterion is needed for choosing between different mappings. However we are not aware of discussion on this issue in the literature.

Next we consider the problem of mapping parameter vectors onto integers. In our soil example, we have a pair of parameters, (x, y). There are once again many possible mappings. An obvious one is the following:

```
Mapping 3:
Mapped Integer :  1     2     3     4      5      6       7       8      9 ...
Parameter Value: (0,0) (1,0) (1,1) (0,1) (-1,1) (-1,0) (-1,-1) (0,-1) (1,-1) ...
```

Mapping 3 uses ever-increasing concentric squares centred on the origin. The coordinates making up the perimeter of the squares are ordered by starting at one on the x-axis and then counting anti-clockwise.

In our soil example, its seems reasonble to assume that x and y can be efficiently coded in separate parts, giving $\log *x + \log *y$ as the code length estimate (once an appropriate mapping was chosen for each component parameter).

In each mapping, there exists a parameter value assigned to the positive integer 1. In the $\log *$ code, this parameter value is assigned a prior probability of 0.5. The choice of the parameter value to be mapped to the integer 1 is called the *origin* problem. For many model classes, there is an obvious origin. For example, for classification trees[QR89], the null tree is the obvious model to be mapped onto the origin. The choice of origin and a mapping involves the same issues involved in the choice of prior probability distribution on hypotheses. This conclusion differs from that of Li and Vitänyi [LV93, page 311] who state:

> "its genesis [the $\log *$ universal code] shows that it expresses no prior knowledge about the true value of the parameter."

Rissanen has considered the origin problem [Ris83, page 427]

> "Does this invalidate the whole process? We think not. When we selected the enumeration of rectangles A, we assumed that the origin of the space R_k is determined by the entire class of models $Prob(X|\theta)$ and by the functions that map the permitted set of independent parameters into R^k. Hence the origin gets determined without any knowledge of any particular value of θ, and the decoder can calculate it from the description of the background information, whose length just adds a constant to the criterion. If we wish to change the origin to the minimizing parameter θ, so as to make its index 1, we clearly must describe this special choice to the decoder, in which case its cost is the same as that of θ, and nothing is saved. The situation is analogous to the choice of the universal computer in the algorithmic theory of complexity..."

We interpret the argument here as saying that the choice of mapping is determined by background knowledge. For example, we examine background knowledge to choose between mappings one and two described above. If the origin is then changed from the default origin, the code length will increase by a constant in order to describe the change. We find it hard to imagine how background knowledge will reveal a suitable mapping in some cases. The choice of mapping may affect the result of inductive inference made on finite data.

3.1.2 Implicit Mappings

The explicit mappings given above are difficult to work with. A more convenient approach was outlined in [Ris83]. The method given in Section 3.1.1 for converting parameters to integers is convenient only if the parameters have uniform precision throughout the parameter space. Consider the single parameter case. In this case the real number may be broken up into segments with length of value $AOPV(\theta)$.

The more convenient approach is as follows. If we know the AOPV for a particular θ, we can discretise the parameter space with regions of this fixed size. We start at the origin and work outwards in a helical fashion, indexing each region. For example, if we have a single parameter with AOPV 0.05 and origin 0:

```
  8    6    4    2    1    3    5    7    9    11   13   15
|--|----|----|----|----|----|----|----|----|----|----|----|----|---|
-0.20 -0.15 -.10 -0.05 0   .05  .10  .15  .25  .30  .35  .40  .45
```

The region .075, .125 above is indexed as 5. Mapping 3 gives the helical mapping for the two-dimensional case.

We need a way of calculating the index of any parameter value. One way to do this is to measure the distance from the origin using a distance measure[6]. We then estimate how many regions occur in the hyperellipsoid centered on the origin and on whose hypersurface is the parameter region of interest. We then estimate how many regions fall on this hypersurface, say A. We need a code length of $\log A$ to choose the region containing the parameter value. The code length to specify our parameter is then $\log *(\mathit{No.\ of\ regions}) + \log_2 A$ bits.

In the simple example above, the estimate of the number of regions between the origin and θ is $\frac{\theta}{0.05}$. We then need to specify the direction θ lies in, using $\log_2 2$ bits. The code length for θ is then approximately:

$$CodeLength(\theta) \approx \log *(\frac{\theta}{0.05}) + \log_2 2 \tag{6}$$

3.1.3 Parameter Transformations

A separate difficulty is that the above mappings are not invariant under linear and nonlinear parameter transformations. Such invariance is an important property of an inductive inference technique [Jef61, page 391]. If, in the above example, we instead choose transmit $(x, \exp y)$ (i.e., the pH and the temperature) then the message length estimate changes if we use the mappings described above to code x and $\exp(y)$ with $\log *$.

The issue of invariance after parameter transformations only arises for continuous hypothesis spaces. In a discrete hypothesis space, a parameter transformation corresponds to relabelling the hypothesis labels. If previously, $f(H_i) = j$, and the hypothesis index is relabelled so i becomes $k, k \neq i$, then we use a new mapping where $f_1(H_k) = j$. By changing the mapping according to the relabelling, the complexity of the hypotheses remains unchanged.

Now the implicit mapping discussed above is invariant under linear transformations, such as $g(\theta/.05)$. These transformations merely shrink or stretch the parameter segments. Translation just moves the origin along, the mapping is unaffected.

So what about non-linear parameter transformations? Our original distance measure will now give different results and the number of regions estimated to be in the hyperellipsoid will change.

To remain invariant, the change in prior probability must cancel with the change in volume of the region:

$$MessLen(Model\ and\ Data) = -\log Prob(D|\theta) - \log(\frac{prior(\theta)}{AOPV_\theta}) \tag{7}$$

If we transform θ to ϕ with a nonlinear transformation $g(\theta)$, then $prior(\phi) = \frac{prior(g(\theta))}{g'(\theta)}$, where $g'(\theta)$ is the first derivative of the transformation of the parameter. We then require that $AOPV_\theta$ behave the same way, so that $g'(\theta)$ can cancel out of the numerator and denominator. For example, this requirement is satisfied when $AOPV_\theta = \sqrt{|F(\theta)|}$ where $F(\theta)$ is the Fisher information matrix. The Jacobians, $g'(\theta)$ then cancel. This particular $AOPV_\theta$ is used in MML code length estimates for sufficiently regular models.

MML requires the specification of $Prob(\theta)$. This, along with an $AOPV(\theta)$, implies an origin (the most probable θ) and a mapping (an enumeration of models from most probable to least probable).

[6] Rissanen [Ris83, page 424] chooses the quadratic norm $||\theta||_{M(\theta)} = \sqrt{(\theta, M(\theta)\theta)}$. This formula is discussed in Section 4.2.

Integer	Log star codeword	Log star length	Binary Tree codeword	Binary Tree length
1	0	1	0	1
2	10.0	3	100	3
3	11.0	3	11000	5
4	10.100.0	6	10100	5
5	10.101.0	6	1110000	7
6	10.110.0	6	1100100	7
7	10.111.0	6	1010100	7
8	11.1000.0	7	111100000	9
9	11.1001.0	7	111010000	9
10	11.1010.0	7	111000100	9

Table 1: First 10 codewords for log star code

3.2 Choosing a Universal Code

3.2.1 The $\log *$ Code

Once a mapping has been decided upon, the MDL procedure encodes the positive integer using the universal $\log *$ code. In this subsection, we examine this universal code and compare to other possible universal codes.

The $\log *$ function is defined as $\log *(n) = \log_2(n) + \log_2 \log_2(n) + ...$, including only positive terms. The $\log *$ code encodes positive integers n with code length $CodeLength(n) = \log *(n) + \log(c)$, where $c \approx 2.8665064$ [Eli75, Ris83]. Note that this is equivalent to the prior $Prob(n) = \frac{1}{n} * \frac{1}{\log_2 n} * \frac{1}{\log_2 \log_2 n} * ... * \frac{1}{c}$.

How does this function come to be a code length estimate? Observe that any integer $n > 0$ can be written in approximately $\log_2 (n)$ bits[7] using ordinary binary notation. However, this is not a prefix code. To make such a scheme prefix, we can pre-pend a code for the length of n — which requires approximately $\log_2(\log_2 n)$ bits. This in turn requires the length of the length of n which is described in $\log_2(\log_2(\log_2 n))$ bits, etc. Furthermore, using such a recursive scheme requires the sender to identify when the message has finally got to the number n, rather than a description of its length. This can be easily done, by noting that the first digit in a binary description of any positive integer begins with a 1. When the sender finally sends n, she may indicate this by making the next digit a 0. The log* code for integers is then given by the C function in Figure 2. The log* code can be decoded using the C function in Figure 3. Table 1 gives the codewords of the $\log *$ code for the first ten integers. In Table 1, a "." is put between each segment of the string. The "." is included only for the purposes of readability; it is not a part of the codeword.

The codeword for ten, which is "11.1010.0", comes in three segments. The first segment "11." indicates that the second segment is $3 + 1 = 4$ digits long. The second segment is decoded as 10. If the third segment began with the digit 1, then the decoder would expect the third segment to be $10 + 1 = 11$ digits long. However, since the third segment begins with a 0, the decoder knows that the second segment represented n, and hence $n = 10$.

In what sense, if any, is the $\log *$ code an optimal code for positive integers? First we define what is meant by a 'universal code'.

Universal codes have infinite-entropy distributions[8], which tail off[9] after some finite integer. Rissanen has shown that all universal codes are equivalent in the sense that the ratio of their lengths converge as n

[7] It actually requires $floor(log_2 n + 1)$ bits, where $floor(x)$ is the greatest integer less than or equal to x.

[8] An infinite entropy distribution has

$$- \lim_{n \to \infty} \sum_{x<n} Prob(x) \log_2 Prob(x) \rightarrow \infty$$

[9] A distribution is said to 'tail off' if $\exists i$ $s.t.$ $j, k > i$ implies $Prob(j) > Prob(k)$ whenever $j < k$.

```
/* encode receives an integer (decimal notation) and
 * returns its log* codeword (a binary string), except
 * for the final 0, supplied by logStar( int n)
 */
char*
encode(int n)
{
    char *codeword = NULL;
    char *binaryFormOfN = NULL;
    int k;

    if (n > 1)
    {
        binaryFormOfN = convertToBinary( n );
        k = strlen( binaryFormOfN );
        codeword = append( encode( k - 1 ), binaryFormOfN );
    }
    if ( n == 1) codeword = append("","");

    return (codeword);
}

char*
logStar( int n)
{
return ( append( encode( n ),"0") );
}
```

Figure 2: The encode function for the $\log *$ code

```
/*
 * Initial call is decode(codeword,0,1);
 */
int
decode(char *codeword, int nextDigit, int n)
{
    int m;

    if (codeword[nextDigit] == '0') return n;
    else
    {
        /* interpret codeword[nextDigit,...,nextDigit+n] as a
           binary number and return its decimal value
        */
        m = convertToDecimal(codeword, nextDigit, nextDigit+n);
    }
    decode(codeword,nextDigit+n+1,m);
}
```

Figure 3: The decode function for the $\log *$ code

gets large [Ris83]. The specific theorem is that for any universal code[Ris83, Theorem 2]:

$$UnivCodeLength(n) < \log(n) + r(n)$$

where $\frac{r(n)}{\log(n)} \rightarrow 0$ as $n \rightarrow \infty$. The 'tail off' requirement is required to avoid assigning non-zero probabilities to infinite messages. Infinite entropy codes tail off slower than finite entropy priors, so arguably providing a flatter prior over the integers.

The $\log *$ code implies an infinite entropy distribution. The $\log *$ distribution tails off very slowly. This makes the $\log *$ universal code a potentially useful prior on the set of positive integers. However, the $\log *$ code is not the only universal code. There are many alternative universal codes. Although their code lengths converge asymptotically, feasible inductive inference is never done with an asymptotically large amount of data. So the difference in code lengths may affect the inference.

The MML approach uses an explicit prior and devises an optimal message-length estimate according to the prior. However, selecting a prior to represent knowledge about an infinite set can be difficult and a universal code may serve as a useful approximation. Since on a finite set of data, the estimated code lengths using different universal codes may be different, care is needed in choosing an appropriate one to represent our prior belief about a distribution of positive integers.

3.2.2 Useful Property of Universal Codes

Rissanen[Ris83, Theorem 2] showed the asymptotic equivalence of universal codes. However the property that makes universal codes truly useful is only hinted at. Consider the problem of encoding an integer sampled from a finite entropy distribution. It is reasonable to assume that any 'true' distribution of integers will have a finite entropy. The minimal expected code length of an integer is achieved by using the true distribution. The entropy of the true distribution is then the expected code length required to describe an integer. In practical problems, we expect the optimal code length for these integers to be finite. We now prove that if the true distribution of integers has a finite entropy then encoding integers using a universal code will result in a finite expected message length.

LEMMA 1: Let n be an integer sampled from a finite entropy distribution, $Prob(n)$. Then we can expect to encode an integer using a universal code with an infinite entropy distribution, $UnivProb(n)$, with a finite codeword. Formally, we are required to show that

$$-\sum_{n=1}^{\infty} Prob(n) \log_2 UnivProb(n) < \infty \tag{8}$$

PROOF: From Theorem 2 of [Ris83], we have

$$\log_2(n) < -\log_2 UnivProb(n) < \log_2(n) + r(n) \tag{9}$$

where $\frac{r(n)}{\log_2 n} \rightarrow 0$ and $r(n) \rightarrow \infty$ as $n \rightarrow \infty$. To show Inequality (8) holds we can use

$$-\log_2 UnivProb(n) < \log_2(n) + r(n) \tag{10}$$

and show

$$\sum_{n=1}^{\infty} Prob(n)(\log_2(n) + r(n)) < \infty \tag{11}$$

Expanding Inequality (11) shows we require

$$\sum_{n=1}^{\infty}(Prob(n) \log_2(n)) + \sum_{n=1}^{\infty}(Prob(n)r(n)) < \infty \tag{12}$$

From Theorem 2, $\frac{r(n)}{\log_2 n} \rightarrow 0$ and $r(n) \rightarrow \infty$ as $n \rightarrow \infty$, so we have

$$\sum_{n=1}^{\infty}(Prob(n) \log_2 n) > \sum_{n=1}^{\infty}(Prob(n)r(n)) \tag{13}$$

if both are finite. Hence Inequality (11) holds if:

$$2\sum_{n=1}^{\infty} Prob(n)\log_2(n) < \infty \tag{14}$$

We need to show that $\sum_{n=1}^{\infty}(Prob(n)\log_2(n))$ converges. As $Prob(n)$ is a proper distribution (i.e., $\sum_{n=1}^{\infty} Prob(n) = 1$), which tails off, and $\sum_{n=1}^{\infty}\frac{1}{n}$ diverges, we have $\exists k$ s.t. $\forall n > k, \frac{1}{n} > Prob(n)$ and hence $\log_2(n) < -\log_2 Prob(n)$. Therefore

$$\sum_{n=k+1}^{\infty} Prob(n)\log_2(n) + c_1 < -\sum_{n=k+1}^{\infty} Prob(n)\log_2 Prob(n) + c_2 \tag{15}$$

where c_1 and c_2 are the expected message lengths due to the first k integers. As $Prob(n)$ is a finite entropy distribution,

$$-\sum_{n=1}^{\infty} Prob(n)\log_2 Prob(n) < \infty \tag{16}$$

Inequalities (15) and (16) show that Inequality (14) holds and therefore Inequality (8) holds. QED.

3.2.3 Other Universal Codes for Integers

We may use the Quinlan and Rivest code for binary trees to map trees onto positive integers [QR89][10]. The mapping chosen here is to map the tree with one node onto 1, the two trees with two nodes onto 2 and 3, the five trees with three nodes onto $4, 5, 6, 7, 8$ and so on. The codewords for the first ten integers of the binary tree code are shown in Table 1.

Table 2 shows the code lengths of the log star and binary tree universal codes for integers for the first twenty powers of ten. Appendix A contains the code used to calculate the code lengths for the binary tree universal code. For a large integer such as 10^{20} the difference is only 3 bits.

Wallace and Patrick [WP93] described codes for n-arity trees. These are also universal codes. In Table 3, we show the code lengths for the first nine powers of ten. The differences in code lengths are still bounded, but are larger. Higher arity tree codes assign less probability to smaller integers. By increasing the arity we can decrease the probability by an arbitrary amount. Therefore the choice of a universal code can affect the performance in practical inference problems.

An alternative scenario is that we are able to assign prior probabilities to the first k integers. Let us say that the probability mass of these k integers is $\sum_{i=1}^{k} Prob(i) = \pi$. We can then use a renormalized $\log *$ code for $i > k$. The code-lengths for i are then

$$CodeLength(i) = -\log Prob(1-\pi) + \log *(i-k)$$

Despite the attractive properties of $\log *$ for positive integers, we have seen that these properties are not unique. One should still enquire as to the best universal code for an inductive inference problem.

3.3 A lower bound on Universal Codes

There has been some speculation that $\log *$ is a lower bound on all universal codes. For example,

> "In fact, we do not know of any other explicitly given [universal codes] which would have a shorter length than [log*] for large integers."[Ris83, page 424]

and Buntine states

> "[The log* code] is in some sense, and approximately (I've seen no proof here), the slowest, monotonically decreasing normalizable prior on integers."[Bun94]

[10]The Quinlan and Rivest codes a binary tree as follows. Perform an preorder traversal of the binary tree. If a node is a non-leaf, emit a 1. If a node is a leaf, emit a 0.

integer	Tree (code length)	Log Star (code length)	Log Star - Tree (code length difference)
10^1	9	7	-2
10^2	13	13	0
10^3	17	17	0
10^4	21	21	0
10^5	25	28	3
10^6	27	31	4
10^7	31	35	4
10^8	35	38	3
10^9	39	41	2
10^{10}	43	46	3
10^{11}	45	49	4
10^{12}	49	52	3
10^{13}	53	56	3
10^{14}	57	59	2
10^{15}	59	62	3
10^{16}	63	66	3
10^{17}	67	69	2
10^{18}	69	72	3
10^{19}	73	76	3
10^{20}	77	80	3

Table 2: Comparing Universal Codes for Integers

	Arity of Tree Code										
Integer	2	12	22	32	42	52	62	72	82	92	102
10^1	9	10	12	13	14	14	15	15	16	16	16
10^2	13	15	18	19	20	21	22	23	23	24	16
10^3	17	20	24	19	20	21	22	23	23	24	24
10^4	21	25	24	26	27	29	30	30	23	24	24
10^5	25	25	29	32	27	29	30	30	31	32	32
10^6	27	30	29	32	34	36	37	38	31	32	32
10^7	31	35	35	39	34	36	37	38	39	40	41
10^8	35	40	41	39	41	43	44	38	39	40	41
10^9	39	40	41	45	41	43	44	46	47	48	50

Table 3: Comparing n-arity Tree Universal Codes

This speculation can be discounted, especially seeing that the binary tree code tails off slower than $\log *$. Ironically, it turns out that $\log *$ can be used to construct a hierarchy of ever more slowly decreasing priors on integers. We prove this in the lemma below. First we need to describe the structure of the $\log *$ code.

The $\log *$ codeword for the integer ten described in subsection 3.2.1 can be generalised to the following form:

$$1[\mathit{length} \ldots \mathit{length}\ n] \ldots\ 1[\mathit{length}\ \mathit{length}\ n]\ 1\ [\mathit{length}\ n]\ 1\ [n]\ 0$$

We view the bit string as a sequence of *segments*, where each segment either describes the length of the next segment, or the final segment is the number n. We may rewrite such a string as:

$$1\ \ldots\ 110\ [\mathit{length} \ldots \mathit{length}\ n] \ldots [\mathit{length}\ \mathit{length}\ n]\ [\mathit{length}\ n]\ [n]$$

We see that the number of segments is encoded using unary notation. However, for very large integers, we may find it more efficient to encode the number of segments using the log* code.

3.3.1 A Hierarchy of Universal Codes

We may construct a hierarchy of universal codes. Let C^0 be the log* coding scheme. We define coding scheme C^1 to be the same coding scheme as C^0, except we encode the number of segments using C^0. Similarly, we define coding scheme C^k to be the same coding scheme as C^{k-1}, except we encode the number of segments using C^{k-1}. Let $L^k(n)$ be the length of n encoded using C^k.

LEMMA 2. Given a universal coding scheme C^k, there exists an integer m such that all integers larger than m are more efficiently coded using C^{k+1}. More formally, $\exists m \in \mathcal{N}$ such that $\forall\, j > m,\ j \in N,\ L^{k+1}(j) < L^k(j)$.

PROOF. We note that bit strings are for the hierarchy of coding schemes are identical, except in the bits describing the number of segments. Let $S(x)$ denote the smallest number which requires x segments using the log* code.

We prove the lemma by induction on k. For $k = 0$, we see that using unary notation $\mathit{bitlength}(7) = 7$, while $L^0(7) = 6$. Therefore for those integers, $j > S(7),\ L^1(j) < L^0(j)$.

For $k > 0$, we assume that $\exists m$ such that $\forall j > m,\ L^k(j) < L^{k-1}(j)$. Consider encoding the integer $S(m)$. The segment length using C^{k+1} is $L^k(S(m))$ while the segment length using C^k is $L^{k-1}(S(m))$. By assumption $L^k(S(m)) < L^{k-1}(S(m))$, and therefore $\forall j > S(m),\ L^{k+1}(j) < L^k(j)$.
QED.

We note that the integers for which our constructed coding schemes are shorter than log* (i.e., $L^1 < L^0$) are enormous (larger than $2^{2^{2^{16}}}$). Using the adjusted code would therefore affect inductive inference in only the most extreme circumstances.

3.3.2 Steps in the logstar code length

Figure 4 shows a graph of the log star code lengths for the first ten integers. The graph is a step function. We are interested in the size of the step increments. The size is one, except where a new segment is added to the code. Then the size depends on the size of the new segment. For large integers, this size is unbounded. This is an unfortunate characteristic of the log star code: neigbouring integers can have an unbounded code length difference.

The binary tree integer code does not have this characteristic. Its step increments are bounded. For this reason, we recommend the binary tree integer code be used instead of the log star code.

3.4 Summary of Differences

One-part MDL message length estimates are used to select model classes. This is a different problem to the model selection problems considered by two part MML estimates. Since MDL and MML tackle different problems, they are strictly incomparable.

When estimating the stochastic complexity, MDL sometimes uses two part message length estimates. A side-effect of the two part message structure is model selection, in addition to model class selection. In the two part message length case, MML requires the specification of prior probabilities on model parameters, while MDL relies on a mapping from model parameters to positive integers and universal codes.

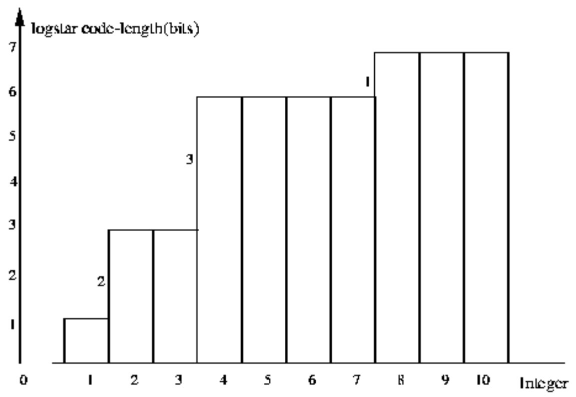

Figure 4: Log star codelengths for the first ten integers

4 MDL and MML Message Length Estimates

In this section, we compare some of the message length estimates proposed in both the MDL and MML literature. These estimates dominate the minimum encoding inference literature, although Sorfkin independently developed a similar two part message length estimate [Sor83]. The first three message length estimates have a two part structure, and the last two message length estimates have a one part structure. The use of a simple example will help ground the discussion. We use the height example from Section 3.2 of Oliver and Hand [OH94].

The following set of heights are assumed to be from a normal distribution. We will calculate various MML and MDL estimates of the minimum message length to transmit the data. The receiver is assumed to know the data is from a normal distribution and that there are 20 data items.

```
181 184 166 178 162
170 173 175 174 169
174 171 174 172 180
179 176 173 172 181
```

These lengths are measured in centimetres. The receiver also believes that μ is uniformly distributed in the range 100...244 cm and that σ is uniformly distributed in the range 0...50 cm.

4.0.1 MML(OH) Code

Oliver and Hand [OH94], following the derivations in Wallace and Boulton [WB68], derive the following expression for this model:

$$MML(OH) = \log_2 \frac{range_\mu}{AOPV_\mu} + \log_2 \frac{range_\sigma}{AOPV_\sigma} + N \log_2 \frac{\bar{s}\sqrt{2\pi}}{\epsilon} + N \frac{s^2 + \frac{\bar{s}^2}{N}}{2\bar{s}^2} \log_2 e \ bits \tag{17}$$

where s is the sample standard deviation, $\bar{s}$ is the unbiased sample standard deviation, ϵ is the accuracy of measurement of the data items, $range_\mu$ is $144cm$, $range_\sigma$ is $50cm$, N is the number of data items, and

$$AOPV_\mu = \bar{s}\sqrt{\frac{12}{N}} \tag{18}$$

and

$$AOPV_\sigma = \bar{s}\sqrt{\frac{6}{N-1}} \tag{19}$$

4.0.2 Calculations for the Example

In the height example, $\epsilon = 1.0$, $N = 20$, $range_\mu = 144cm$, $range_\sigma = 50cm$, $s = 5.23$, $\bar{s} = 5.37$:

$$\begin{aligned} MML(OH) &= \log_2 34.64 + \log_2 16.58 + 20 \log_2 13.45 + 20 * .499 * 1.443 \\ &= 5.11 + 4.05 + 74.99 + 14.40 \\ &= 98.55 \end{aligned} \tag{20}$$

4.1 MDL(1)

The first MDL two part message length estimate we consider is [Ris78]:

$$MDL(1) = -\log_2 Prob(x|\theta) + k/2 \log_2(N) + \log_2 *(k) \tag{21}$$

The first term is the negative log of the likelihood and is a term common to both MDL and MML message length expressions.

k is the number of distinct parameters and N is the number of data items. The second term represents the precision to which each parameter is stated in the message length, $\log \sqrt{N}$ bits per parameter. It was derived by minimizing the message length expression by taking the first derivative.

The third term is relatively small and is often ignored. The $\log *$ code features more prominently in the next MDL code examined.

4.1.1 Calculations for the Example

We applied $MDL(1)$ to the example, where $k = 2$ and $N = 20$, so $k/2 \log(N) = \log 20 = 4.32$

$$MDL(1) = 88.68 + 4.32 + 1.00 = 94.00 \tag{22}$$

Note that asympototically $MDL(1)$ approaches $MML(OH)$. Just make N large in $MML(OH)$, ignore the constants, note that $\bar{s}$ and s asymptote to the true σ and we get

$$MML(OH) \approx MDL(1) \approx \frac{1}{2} \log_2 N + \frac{1}{2} \log_2 N - \log_2 Prob(x|\theta) \tag{23}$$

This explains the difference in message length estimates between $MDL(1)$ and $MML(OH)$ of 4.55 bits for the non-asymptotic case of $N = 20$.

4.2 MDL(2)

A more accurate MDL message length estimate is the following [Ris83, page 426]:

$$MDL(2) = -\log_2 Prob(x|\theta) + k \log_2 \sqrt{|M(\theta)|} + \log_2 C(k) \tag{24}$$

where

$$C(k_{even}) = (2\pi)^{(\frac{1}{2}k)!} 2^{(\frac{1}{2}k)} \tag{25}$$

$$C(k_{odd}) = \pi^{(\frac{1}{2})(k-1)} 2^{k+1} \frac{(\frac{1}{2}(k+1))!}{(k+1)!} \tag{26}$$

and

$$M(\theta) = \frac{-\partial^2 \log_2 Prob(x|\theta)}{\partial\theta^2} \tag{27}$$

This approximation is invariant under linear transformations, but is not invariant under nonlinear transformations of the parameter space [Ris83, page 427]. This code suffers from the origin problem discussed in Section 3.1. The code is actually derived by a counting argument of the type described there. The log terms in $MDL(2)$ are used as a convenient approximation to $\log *$. This approximation is used by Rissanen [Ris87].

A different derivation, but using similar arguments, is given in [Ris93b].

4.2.1 Calculations for the Example

First, consider the $\log_2 C(k)$ term with $k = 2$,

$$\log_2 (2\pi)^{1!} 2 = \log_2 4\pi = 3.65 \tag{28}$$

Second calculate the $\log \sqrt{|M(\theta)|}$ term,

$$\frac{\partial^2 - \log Prob(x|\theta)}{\partial\mu^2} = \frac{N}{\sigma^2} = \frac{20}{5.23^2} \tag{29}$$

$$\frac{\partial^2 - \log Prob(x|\theta)}{\partial\mu\partial\sigma} = (\frac{2}{\sigma^3}) \sum_i (x_i - \mu) = 0.0 \tag{30}$$

since $\sum_i (x_i - \mu) = 0.0$ if we estimate μ as the mean of x_i.

$$\frac{\partial^2 - \log Prob(x|\theta)}{\partial\sigma^2} = \frac{-N}{\sigma^2} + \frac{3}{\sigma^4} \sum_i (x_i - \mu)^2 = \frac{2N}{\sigma^2} \tag{31}$$

$$\frac{\partial^2 - \log Prob(x|\theta)}{\partial\sigma\partial\mu} = \frac{2}{\sigma^3} \sum_i (x_i - \mu) = 0.0 \tag{32}$$

$$M(\mu,\sigma) = \begin{pmatrix} \frac{N}{\sigma^2} & \frac{2}{\sigma^3}\sum_i (x_i - \mu) \\ \frac{2}{\sigma^3}\sum_i (x_i - \mu) & \frac{2N}{\sigma^2} \end{pmatrix} = \begin{pmatrix} .7312 & 0 \\ 0 & 2 \times .7312 \end{pmatrix} \tag{33}$$

The off-diagonal terms in this particular example are 0, but this is not the case in general.

$$|M(\mu,\sigma)| = .7312 \times 2 \times .7312 \tag{34}$$

That makes $0.5 \times \log_2 |M(\mu,\sigma)| = .05$.

$$MDL(2) = 88.68 + 0.05 + 3.65 = 92.38 \tag{35}$$

4.3 MML(WF)

The $MML(WF)$ is a feasible two part message length estimate proposed by [WF87]. The $MML(WF)$ expression is

$$MML(WF) = -\log_2 Prob(x|\theta) - \log_2 Prob(\theta) + \frac{1}{2}\log_2 |F(\theta)| + \frac{k}{2}\log_2 \kappa_k + \frac{k}{2}\log_2 e \tag{36}$$

where $F(\theta) = E(\frac{\partial^2 - \log_e Prob(D|\theta)}{\partial\theta^2})$ is the Fisher information matrix and κ_k is the optimal k dimensional quantising lattice constant (known values and bounds are given in [CS88, pages 59-61]).

The derivation of the $MML(WF)$ AOPV differs from the $MML(OH)$ AOPV in the following respect. $MML(OH)$ minimizes the AOPV for each parameter separately. $MML(WF)$ minimizes the AOPV for all the parameters together, giving an estimate of the optimal volume.

Let us approximate this volume with a k-sphere of radius $AOPV(\theta)$.

$$\frac{\pi^{\frac{k}{2}}}{(\frac{k}{2})!} AOPV(\theta)^k = \frac{(\kappa_k)^{-\frac{k}{2}}}{\sqrt{F(\theta)}} \tag{37}$$

Here $k = 2$, so we get

$$AOPV(\theta) \approx \sqrt{\frac{1}{\kappa_2 \pi \sqrt{F(\theta)}}} \tag{38}$$

Note that in two dimensions, the volume is an area, and that the shape of the optimal quantising area is hexagonal. We are approximating the hexagons by circles.

4.3.1 Calculations for the Example

Now $-\log_2 Prob(\theta)$ is $-\log_2 \frac{1}{144.0 \times 50.0} = 12.81$. The Fisher Information matrix is:

$$F(\theta) = \begin{pmatrix} \frac{N}{\sigma^2} & 0 \\ 0 & \frac{2N}{\sigma^2} \end{pmatrix} \tag{39}$$

The Fisher Information matrix differs from the $M(\mu, \sigma)$ matrix in having zero off-diagonals because in the expectation $\sum(x_i - \mu) = 0$.

The $\frac{1}{2}\log_2 |F(\theta)|$ is then 0.05. The κ_2 term is $\log_2 \frac{5}{36\sqrt{3}}$ The MML estimate is:

$$MML(WF) = 88.65 + 12.81 + 0.05 - 3.64 + 1.44 = 99.30 \tag{40}$$

The $MML(WF)$ estimate is 99.30 bits. This differs from the $MML(OH)$ by 0.75 bits, which is not significant.

Using $\kappa_2 = \frac{36\sqrt{3}}{5}$ and $|F(\theta)| = \frac{2N^2}{\sigma^4}$, the $AOPV_{WF}$ is $\sigma\sqrt{\frac{5}{36\pi\sqrt{6}N}}$. The AOPVs for $MML(OH)$ are $AOPV_\mu = \sigma\sqrt{\frac{12}{N}}$ and $AOPV_\sigma = \sigma\sqrt{\frac{6}{N-1}}$ respectively. Taking their product gives us the area of the rectangle used by $MML(OH)$. Taking the ratio of the area used by the two codes gives:

$$\begin{aligned}
\frac{AOPV_{OH}}{AOPV_{WF}} &= \frac{\sigma^2\sqrt{\frac{72}{N(N-1)}}}{\frac{36\sqrt{3}\sigma^2}{5\sqrt{2}N}} && (41)\\
&= \frac{5\sqrt{144}}{36\sqrt{3}} \times \frac{N}{\sqrt{N(N-1)}} && (42)\\
&\approx \frac{5}{3\sqrt{3}} && (43)\\
&\approx 0.96 && (44)
\end{aligned}$$

We see that the quantisation region for $MML(WF)$ is 4% smaller than those of $MML(OH)$. The effect is that $MML(OH)$ states estimates with slightly more precision than $MML(WF)$. The extra precision is perhaps not warranted by the data.

4.3.2 MML(OH) and MML(WF)

We now examine the similarity of terms in the $MML(OH)$ and $MML(WF)$ expressions for our simple two parameter example: The terms left after taking the difference may be summarised as follows:

$$\begin{aligned}
MML(OH) - MML(WF) &= -\log AOPV_{\mu} - \log AOPV_{\sigma} - \log |F(\theta)| - \frac{k}{2}\log \kappa_2 + \frac{1}{2}\log_2 e \\
&= (-2.06 - 1.59 + .72) - (.05 - 3.64 - 1.44) \qquad (45) \\
&= (-2.93) + 2.21 \\
&= 0.72
\end{aligned}$$

After ignoring the insignificant difference in model costs, we find that the difference is due to $MML(OH)$'s estimate of the error in likelihood due to truncating the parameter estimates. $MML(OH)$ evaluates this error as $\frac{1}{2}\log_2 e$, while $MML(WF)$ determines it to be $\log_2 e$.

The reason for this difference is simply that $MML(OH)$ did not take into account a multi-dimensional (two, in this case) parameter space in its error estimate.

4.4 Predictive MDL

An optimal code for the data can be achieved by by encoding each datum, x_{t+1}, using $-\log_2 Prob(x_{t+1}|x^t)$, where x^t denotes the sequence of data already encoded. The resulting code length expression is just another way of writing the stochastic complexity of Equations (3) and (2):

$$SC = -\sum_{t=1}^{n} \log_2 Prob(x_{t+1}|x^t) \qquad (46)$$

Equation (46) can be evaluated in a Bayesian approach using the following:

$$Prob(x_{t+1}|x^t) = \int Prob(x_{t+1}|\theta)Prob(\theta|x^t)d\theta \qquad (47)$$

The integration required in Equation (47) can be difficult to compute.

Predictive MDL approximates Equations (47) and (46) using the following non-Bayesian approximations. Equation (46) can be approximated by calculating the maximum likelihood estimator $\hat{\theta}_{ML}$ and using this to encode x_{t+1}. This requires the following assumption:

$$Prob(x_{t+1}|x^t) \approx Prob(x_{t+1}|\hat{\theta}_{ML}) \qquad (48)$$

where $\hat{\theta}_{ML}$ was estimated from x^t. An initial maximum likelihood estimate is needed, $\hat{\theta}_0$, to provide an initial $Prob(x_1|\hat{\theta}_0)$ in order to encode the first datum. The use of an initial estimate is equivalent to the use of a conjugate prior on the model parameters in the Bayesian framework.

The PMDL procedure also uses a non-Bayesian method to encode the next datum. $Prob(x_t|\hat{\theta}_0)$ is used to encode data items, x_t, until enough data has been encoded to give a unique maximum likelihood estimate for the all the model parameters, $\hat{\theta}_{ML}$.

A 'fix' needs to be made to this procedure just in case $Prob(x_{t+1}|x^t) = 0$ [Ris89, page 69]. The fixes used are *ad hoc*.

Another difficulty is that the PMDL message length estimate is dependent on the ordering of the data. In some applications the data will have a specific order, such as time series data. In the height example, there is no such natural order. One possibility is to find the very best ordering, requiring a search through $n!$ alternatives. Rissanen proposes another method, which he calls *symmetric Predictive Least Squares*:

> "we may, in fact, modify the entire process so that the result is independent of the ordering of the data. We pick as the first data point $y_{i(1)}$ the one which can be predicted best; i.e., the smallest in absolute value. As $y_{i(2)}$ we pick the nearest data point to $y_{i(1)}$ in the squared distance. Then, estimate the first parameter value and pick as $y_{i(3)}$ that data point which gives the smallest prediction error, and so on."[Ris89, page 132].

```
double
pmdl(double x[], double priorMu, double priorSigma )
{
   double MessLen = 0;
   double mu = priorMu;
   double sigma = priorSigma;

   while (still data left)
   {
      x = nextDatum();
      MessLen = MessLen + log f(x|mu,sigma) * epsilon;
      mu = updateMu( mu, x);
      sigma = updateSigma( sigma, x);
   }
}
```

Figure 5: Predictive MDL Algorithm

Rissanen has commented on the problems of needing an initial estimate and of order dependence:

> "it is only because of a certain singularity in the process, as well as the somewhat restrictive requirement that the data must be ordered, that we do not consider the resulting predictive code length to provide another competing definition for the stochastic complexity, but rather regard it as an approximation..."[Ris89, page 68].

PMDL is a simple method for approximating the stochastic complexity. However PMDL requires large amounts of data to provide accurate estimates of the stochastic complexity. This can be seen by observing that the $Prob(x_t) = Prob(x_t|\hat{\theta_{ML}})$ is inaccurate for small t.

4.4.1 Calculations for the Example

For the height example, and using the order the heights came in, the best initial estimate (μ_0, σ_0^2) was found to be $(162.35, 18.65)$. This was found by doing a grid search. The resulting code length was 92.74 bits. The dependency on the initial estimate chosen is very strong. For instance, using $(100.00, 3.0)$ as the initial estimate gives a code length of 630.00 bits.

The algorithm used is straightforward and is given in Figure 5 using pseudo-C code.

Next, we used the order independent method proposed by Rissanen. For the height example, the best initial estimate (μ_0, σ_0^2) was found to be $(160.0, 2.9)$. This was found by doing a grid search. The resulting code length was 76.88 bits.

The sample size of 20 is not large enough for PMDL to give accurate estimates of the optimal message length. Rissanen intended to apply it to choosing between model classes. Since we only have one model class here, we cannot say anything about its performance in model class selection.

4.5 Stochastic Complexity

The definition of stochastic complexity was given in Equation (3) and Equation (2).

4.5.1 Calculations for the Example

Stochastic Complexity does not apply to our height example. We have only one model class proposed, the set of univariate normal densities, so there is no model class selection to be done. However, it is still interesting to calculate the Stochastic Complexity code length estimate, which is the optimal message length for the model class, and compare it to the other message length estimates.

The Stochastic Complexity can be estimated analytically or numerically. In the simple case considered here we can estimate it analytically, using Laplace's approximation to the integral of the normal density

$$SC = -\log_2(Prob(x|\theta)Prob(\theta)(2\pi)^{\frac{k}{2}}|M|^{-\frac{1}{2}}) \tag{49}$$

where M is the matrix of second derivatives of $-\log_e Prob(x|\theta)$.

Expanding out the terms to allow comparison with earlier message length estimates, we have

$$SC = -\log_2 Prob(x|\theta) - \log_2 Prob(\theta) - \frac{k}{2}\log_2 2\pi + \log_2 \sqrt{|M|} \tag{50}$$

We have evaluated the first two terms in Section 4.3.1 and use those values here:

$$SC = 88.68 + 12.81 - 2.65 + 0.05 \tag{51}$$

The SC total code length is 98.89 bits.

The difference between SC and $MML(WF)$ is .41 bits in this case.

Comparing the terms of both SC and $MML(WF)$, we see that $MML(WF)$ uses lattice constants, while SC uses spheres to estimate the volume. This leads only to a constant difference in message length estimates w.r.t N. SC uses the determinant of the matrix of second differentials, while $MML(WF)$ uses the determinant of the expected matrix of second differentials. For larger amounts of data, the ratio of these terms will converge to 1. For our small example here, it just happens the terms are identical (due to the choice of $\hat{\mu}$ as the sample mean).

This leads some to conclude that there is little difference between the two approaches. Since the derivation of the $MML(WF)$ uses less familiar concepts, the SC approach seems conceptually cleaner. Once again, we reiterate that the real difference is that $MML(WF)$ provides a means of assessing a single model, while SC allows comparisons of model classes.

5 Conclusion

The MML and MDL approaches to inductive inference rely on choosing models and model classes whose message lengths for encoding the data x are minimal.

The main difference between the two part message length estimates of MML and MDL is the interpretation of priors. The main difference between the two part codes and one part codes is that the one part codes do not select a model, but instead provide a predictive model for the data.

The comparisons made in Section 4 are summarised in Table 4. The first column gives the name of the message length estimate used. The second column gives the message length estimate on the height example data. The third column states whether the estimate is invariant under one-to-one nonlinear parameter transformations. The invariance entry for SC was labelled Yes/No, because SC is invariant if evaluated exactly, but is not invariant using the numerical approximations commonly employed. The fourth column indicates whether the estimate is intended to be accurate for small samples rather than an asymptotic approximation. The final column gives the difference between each estimate and the optimal code length given by the stochastic complexity. We believe that the $MML(WF)$ estimate is the best two part message length estimate discussed here. The basis for this belief is its minimal difference from the optimal one part code length, its ability to cope with multi-parameter models, its invariance under one-one parameter transformations and its performance on small data sets.

Note that the optimal accuracy of $MML(WF)$ parameter estimates is about 4% less precise than that of $MML(OH)$ for the two parameter model. Accuracy is measured in terms of the size of the optimal parameter volume chosen. This difference is negligible after considering the numerical approximations made in the derivations.

The final point we wish to highlight concerns the issue of invariance. What happens if we transform the current parameter space nonlinearly? For example, what if we wanted to estimate the square of the mean, rather than the mean? If the estimator chosen is not invariant under non-linear transformations, we will get a different answer. An inductive inference method that gives differenct answers depending on the parameterization of the models seems would seem to be suspect.

Name	Estimate	Invariant?	Small Sample?	Difference
MML(WF)	99.30	Yes	Yes	+0.41
MML(OH)	98.55	No	Yes	-0.34
MDL(1)	94.00	Yes	No	-4.89
MDL(2)	92.38	No	Yes	-6.51
PMDL	92.74	Yes	No	-6.15
SC	98.89	Yes/No	Yes	n/a

Table 4: Summary of Two-Part Message Length Codes

6 Acknowledgements

We would like to thank Chris Wallace for ideas and discussions. We would also like to thank Chris Wallace and David Dowe for critical reading of the manuscript.

References

[AWY92] L. Allison, C.S. Wallace, and C.N. Yee. Finite-state models in the alignment of macromolecules. *Journal of Molecular Evolution*, 35:77–89, 1992.

[BC91] A.R. Barron and T.M. Cover. Minimum complexity density estimation. *IEEE Trans. on Info. Theory*, 37:1034–1054, 1991.

[BO95] R. Baxter and J. Oliver. Universal codes in MDL inductive inference. *Submitted to COLT-95*, 1995.

[Bun94] W. Buntine. Internet:Machine Learning List, 6(27), 1994.

[CS88] J.H. Conway and N.J.A. Sloane. *Sphere Packings, Lattices and Groups.* Springer-Verlag,New York, 1988.

[Dow95] D.L. Dowe. Priors and information-theoretic inductive inference: MML and MDL (in preparation). Technical report, Dept. of Computer Science, Monash University, Clayton, Victoria 3168, AUSTRALIA, 1995.

[EH81] B.S. Everitt and D.J. Hand. *Finite Mixture Distributions.* Chapman and Hall, London, 1981.

[Eli75] P. Elias. Universal codeword sets and representations of the integers. *IEEE Trans. Inform. Theory*, 21:194–203, 1975.

[Goo85] I.J. Good. Weight of evidence: a brief survey. In J.M. Bernado et al., editors, *Bayesian Statistics 2*, pages 249–269. Elsevier, New York, 1985.

[Jef61] H. Jeffreys. *Theory of Probability.* Cambridge, 1961.

[KR93] R.E. Kass and A.E. Raftery. Bayes factors and model uncertainty. Technical Report 571, Dept. of Statistics, Carnegie-Mellon University, Dec. 1993.

[LV93] Ming Li and Paul Vitanyi. *An Introduction to Kolmogorov Complexity and its applications.* Springer Verlag, August 1993.

[Mac92] David J.C. MacKay. *Bayesian Modeling and Neural Networks.* PhD thesis, Dept. of Computation and Neural Systems, CalTech, 1992.

[OB94] J.J. Oliver and R.A. Baxter. MML and Bayesianism: similarities and differences. Technical report TR 206, Dept. of Computer Science, Monash University, Clayton, Victoria 3168, AUSTRALIA, 1994.

[OH94] J.J. Oliver and D.J. Hand. Introduction to minimum encoding inference. Technical report TR 4-94, Dept. of Statistics, Open University, Walton Hall, Milton Keynes, MK7 6AA, UK, 1994. Also available as TR 205 Dept. Computer Science, Monash Uni., Clayton, Vic 3168 AUSTRALIA.

[OH96] J.J. Oliver and D.J. Hand. Averaging over decision trees. *Journal of Classification*, To appear in 1996. An extended version is available as Technical Report TR 5-94, Dept. of Statistics, Open University, Walton Hall, Milton Keynes, MK7 6AA, UK.

[PW82] J.D. Patrick and C.S. Wallace. Stone circle geometries: an information theory approach. In D.C. Heggie, editor, *Archaeoastronomy in the Old World*, pages 231–264. Cambridge University Press, Cambridge, 1982.

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

[Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[Ris83] J. Rissanen. A universal prior for the integers and estimation by MDL. *Ann. of Statistics*, 11(2):416–431, June 1983.

[Ris87] J. Rissanen. Stochastic complexity. *J. R. Statist. Soc. B*, 49(3):223–239, 1987.

[Ris89] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, N.J., 1989.

[Ris93a] J. Rissanen. Fisher information and stochastic complexity. Research report rj 9547, IBM ARC, San Jose, CA 95120-6099, October 1993.

[Ris93b] J. Rissanen. Information theory and neural nets. In P. Smolensky et al., editors, *Mathematical Perspectives on Neural Networks*. Lawrence Erlbaum, 1993.

[Sch78] G. Schwarz. Estimating dimension of a model. *Ann. Stat.*, 6:461–464, 1978.

[Sor83] R. Sorfkin. A quantitative Occam's razor. *International Journal of Theoretical Physics*, 22(12):1091–1103, 1983.

[WB68] C.S. Wallace and D.M. Boulton. An information measure for classification. *Computer Journal*, 11(2):195–209, 1968.

[WB75] C.S. Wallace and D.M. Boulton. An invariant Bayes method for point estimation. *Classification Society Bulletin*, 3(3):11–34, 1975.

[WD93] C.S. Wallace and D.L. Dowe. MML estimation of the von Mises concentration parameter. Technical Report 93/193, Dept. of Computer Science,Monash University,Clayton 3168, Australia, 1993.

[WF87] C.S. Wallace and P.R. Freeman. Estimation and inference by compact coding. *J. R. Statist. Soc B*, 49(3):240–265, 1987.

[WF92] C.S. Wallace and P.R. Freeman. Single factor analysis by MML estimation. *J. R. Statist. Soc. B.*, 54(1):185–209, 1992.

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

7 Appendix: C Code for generating Tree code lengths

```
/* Evaluates number of possible trees with n internal nodes
 * Uses recurrence relation: D_n = sum_(k=0)^(n-1) D_{n-k-1} * D_{k}
 *                           D_0 = 1,D_1 = 1
 */
int
treeNum(int n)
{
int i,N,sum = 0;

if (n == 0) return 1;
else if (n == 1) return 1;
else if (n > 1)
    {
        N = n-1;
        for (i=0; i<=N; i = i+1)
        {
            sum = sum + treeNum(N-i)*treeNum(i);
        }
        return sum;
    }
else printf("caught a negative call \n");
}

/* The Quinlan&Rivest(1989) code for trees has length
 * in bits equal to the number of nodes in the tree,
 * including leaves.
 * If a tree has n internal nodes, it has n+1 leaves,
 * so total number of bits required (2n+1)
 */
int
treeCode(int n)
{
    int i = 0;            /* no. of nodes */
    int sum = 1;          /* total no. nodes so far */

    if (n == 1) return 1; /* special case: null node */

    while (sum < n)
    {
        i = i+1;
        sum = sum + treeNum(i);
    }
    return (2*i+1);
}
```