Tech Report 4-94
Department of Statistics, Open University, Walton Hall, MK7 6AA, UK

Tech Report 205
Department of Computer Science, Monash University, Clayton, Vic. 3168, Australia

July 6th 1994 — Amended November 14th 1994 and December 5th 1996

Introduction to Minimum Encoding Inference

JONATHAN J. OLIVER[1]
DAVID HAND (D.J.Hand@open.ac.uk)
Department of Statistics
Open University
Walton Hall, Milton Keynes, MK7 6AA, UK

Abstract: This paper examines the minimum encoding approaches to inference, Minimum Message Length (MML) and Minimum Description Length (MDL). This paper was written with the objective of providing an introduction to this area for statisticians. We describe coding techniques for data, and examine how these techniques can be applied to perform inference and model selection.

Contents

[1] Jonathan Oliver's current address is: Department of Computer Science, Monash University, Clayton, Vic. 3168, Australia (jono@molly.cs.monash.edu.au).

1 Introduction

The two major proponents of minimum encoding inference are Wallace and Rissanen, who developed similar approaches to inductive inference. Wallace termed the inference method Minimum Message Length (MML) induction and Rissanen termed it Minimum Description Length (MDL) induction. This technical report is the first in a series of three technical reports describing aspects of the minimum encoding approach to inference. This report was written with the objective of providing an introduction to minimum encoding inference for statisticians. We concentrate on the Wallace approach [22, 23, 24, 25] — for details of the Rissanen approach see [18, 19, 20]. The second report (Oliver and Baxter [13]) describes similarities and differences between MML inference and Bayesian inference. The third report (Baxter and Oliver [3]) describes similarities and differences between Wallace's MML approach and Rissanen's MDL approach.

We assume we have some measurements from the real world, and a set of models, $M = \{m_1, m_2, \ldots, \}$, with which we attempt to explain those measurements. MML can be used to assess the quality of each $m_i \in M$.

To assess the quality of a model m_i, we construct a description of the model and of the data in terms of the model. A good model is one leading to a concise total description — the shorter the better. More formally, we encode the model m_i and the measurements (using m_i) as a binary string. Given a particular encoding scheme, we define the *code length* of the data as the length of the encoded binary string. The *complexity* of a string, S, is the length of the shortest string that can be constructed to describe S. Chaitin [6] used Godel's Incompleteness Theorem to demonstrate that it is in fact impossible to determine the complexity of an arbitrary string. MML takes a pragmatic approach to this problem — approximating the complexity of a string, S, as the length of the shortest string we can find that defines S.

To give the reader a feeling for the calculation of code lengths, Section 2 discusses a range of encoding methods, and gives a variety of examples calculating the code lengths of data. Section 3 considers how one may encode a set of real valued measurements modelled by a normal distribution, and derives the MML estimators for the parameters of the normal distribution. Section 4 considers using MML for two non regular model classes, namely the classification tree, and the probabilistic finite state automata. Section 5 briefly examines some properties that coding schemes should exhibit, and discusses the relationship of MML with Bayesianism and MDL. Section 6 compares some of the features of MML inference, with the features of other general criteria used for model selection.

2 Coding Data

In this section, we discuss a range of encoding schemes to demonstrate how one might go about determining the code length of various types of data. We are concerned with calculating the length of description, not actually encoding data. Therefore, we use codes that are efficient in terms of code length, but may not be efficient in the time required to encode/decode data. For the same reason, in some examples we calculate the length of description without actually providing the encoded string.

We begin in Section 2.1 with an example of encoding integers, where we provide the encoding scheme used. We then consider, calculating the code lengths of integers in a more general setting. Section 2.2 considers coding a single integer from a uniform distribution, i.e., those values where we have no belief that one value of the integer is more likely than any other value. Section 2.3 extends this to sequences of such values.

The later half of this section turns its attention to encoding numbers from non-uniform distributions. Section 2.4 considers coding a single integer from a multinomial distribution, and Section 2.6 extends this to sequences of such numbers. Section 2.7 considers coding a real valued number according to probability density functions.

2.1 Example of Coding Schemes for Integers

Consider describing the following sequence of 100 integers each of which has a value from 0 to 7:

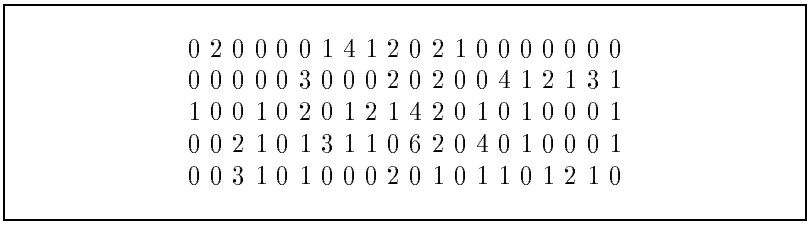

Figure 1: A Sequence of Integers

One approach to encoding such a sequence of numbers in binary is to view each number as a symbol, and use a *code dictionary* to encode each symbol in turn. A code dictionary (such as the one depicted in Figure 2) lists a binary *codeword* for each possible symbol. The code dictionary must be prefix, i.e., no codeword may consist of another codeword followed one or more binary digits. If we allow non-prefix codes, then a message may be ambiguous. For example, if the codeword for 6 was '11', and 7 was '111', then we could not determine whether the message '11111' was 67, or 76.

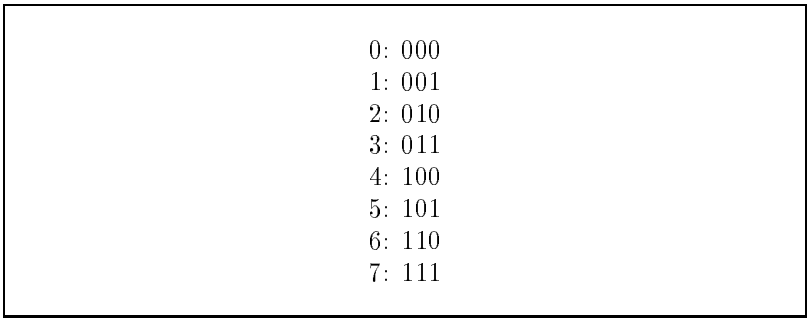

Figure 2: Code Dictionary 1

An obvious way to encode the numbers 0 to 7 is to use the binary representation for each number, as done in Code Dictionary 1 (Figure 2). Encoding the sequence in Figure 1 using Code Dictionary 1 results in the following message (300 bits long):

```
000 010 000 000 000 000 001 100 001 010 000 010 001 000 000 000 000 000 000 000
000 000 000 000 000 011 000 000 000 010 000 010 000 000 100 001 010 001 011 001
001 000 000 001 000 010 000 001 010 001 100 010 000 001 000 001 000 000 000 001
000 000 010 001 000 001 011 001 001 000 110 010 000 100 000 001 000 000 000 001
000 000 011 001 000 001 000 000 000 010 000 001 000 001 001 000 001 010 001 000
```

0 :	1
1 :	01
2 :	001
3 :	0001
4 :	00001
5 :	000001
6 :	0000000
7 :	0000001

Figure 3: Code Dictionary 2

However, if we use Code Dictionary 2 (Figure 3)[2] we achieve a significantly shorter message, which is 186 bits long:

```
1 001 1 1 1 1 01 00001 01 001 1 001 01 1 1 1 1 1 1 1
1 1 1 1 1 0001 1 1 1 001 1 001 1 1 00001 01 001 01 0001 01
01 1 1 01 1 001 1 01 001 01 00001 001 1 01 1 01 1 1 1 01
1 1 001 01 1 01 0001 01 01 1 0000000 001 1 00001 1 01 1 1 1 01
1 1 0001 01 1 01 1 1 1 001 1 01 1 01 01 1 01 001 01 1
```

If someone is to understand such a message, then it is necessary to describe the code dictionary at the beginning of the message. We assume that the receiver knows that the message is describing a sequence of 100 numbers each in the range 0 to 7. An inefficient method for describing a code dictionary is to list the 7 codewords in turn with a terminator between each codeword. A codeword can be described by duplicating each bit in the codeword, and terminating the codeword with "01". Using this convention, Code Dictionary 1 would have the following message (length = 64 bits):

```
000000 01
000011 01
001100 01
001111 01
110000 01
110011 01
111100 01
111111 01
```

Code Dictionary 2 would have the following message (length = 88 bits):

```
11 01
0011 01
000011 01
00000011 01
0000000011 01
000000000011 01
00000000000000 01
00000000000011 01
```

Thus the total message lengths for the two code dictionaries are: 64+300 = 364 bits, and 88+186 = 274 bits respectively.

[2] Code Dictionary 2 corresponds to a code generated from a non-uniform distribution (described in Section 2.4). This particular code is an example of a Rice code [17].

2.2 Coding A Single Integer from a Uniform Distribution

We can encode an integer value x that takes a value in the range $[0 \,..\, (2^N - 1)]$ in a codeword of N bits, by using a binary code. For example, if x can take values from $[0 \,..\, 7]$, then x can be encoded in 3 bits. This encoding approach assumes that each value in this range are equally likely to be encountered. For the remainder of this section, we use the notation

$$x \in [L \,..\, U]$$

to indicate that x is sampled from a uniform distribution on the interval $[L \,..\, U]$.

2.3 Coding A Sequence of Integers from a Uniform Distribution

A naive approach to encoding a sequence of integers, $x_1,\ x_2,\ \ldots\ x_S$ would be to encode each integer in turn and concatenate the codewords. However, more efficient encoding methods are available.

To illustrate one possible coding scheme, we consider an example of describing two integers. Let $x_1 \in [0 \,..\, 4]$, and $x_2 \in [0 \,..\, 5]$. A naive encoding of $(x_1,\ x_2)$ requires 6 bits (3 bits for each). A better approach is to evaluate $C = 6x_1 + x_2$, and encode the pair $(x_1,\ x_2)$ using the binary representation of C. This requires 5 bits, since $C \in [0 \,..\, 29]$. C can be decoded as x_1 and x_2 by letting:

$$x_2 = C \ mod\ 6$$

$$x_1 = \frac{C - x_2}{6}$$

We now extend this scheme to encoding a sequence of integers[3], $x_1,\ x_2,\ \ldots\ x_S$ where each $x_i \in [0 \,..\, N_i - 1]$. Function Eval Code Number evaluates a single integer C, that summarises the sequence. C is in the range $[0 \,..\, R - 1]$, where

$$R = \prod_{i=1}^{i=S} N_i$$

Thus, C can be represented as a bit string with length[4] $ceiling(log_2 R)$. A bit string of this form can be decoded by applying Algorithm Decode. We note that no shortening of the message occurs if each N_i is a power of 2.

Function Eval Code Number

1. $C \leftarrow x_1$
2. For $j \leftarrow 2$ to S
3. $C \leftarrow C \times N_j + x_j$
4. End For
5. Return C

Algorithm Decode

1. For $j \leftarrow S$ downto 2
2. $x_j \leftarrow C \bmod N_j$
3. $C \leftarrow \frac{C - x_j}{N_j}$
4. End For
5. $x_1 \leftarrow C$

[3] In many cases each x_i will come from the same range.

[4] The ceiling of x is the smallest integer y such that $y \geq x$.

2.4 Coding A Single Integer from a Non-Uniform Distribution

In the previous two sections, we considered coding integers sampled from uniform distributions. However, if we are encoding integers sampled from a non-uniform distribution, then we should adjust our coding scheme to reflect this. Section 2.1 demonstrates the reduction in message length that could be achieved by using a code corresponding to a non-uniform distribution. The message describing the numbers from Figure 1 using Code Dictionary 1 (corresponding to a uniform distribution) was of length 364 bits, while the message using Code Dictionary 2 (corresponding to a non-uniform distribution) was of length 274 bits.

We wish to construct a code dictionary for symbols 0, ..., M sampled from a probability distribution where symbol i has probability P_i. Section 2.4.1 discusses how to construct a dictionary for a *restricted probability distribution* (RPD). An RPD is a probability distribution where each P_i can take the form:

$$P_i = \frac{2^{z_i}}{2^y}$$

for integers y, z_0, z_1, ..., z_M. Section 2.4.2 discusses how an arbitrary probability distribution may be approximated by an RPD, which then can be used to construct a code dictionary.

2.4.1 Constructing a Code Dictionary from an RPD

We illustrate this process by constructing Code Dictionary 2 (Figure 3) used in Section 2.1. The codes constructed in this section are equivalent to optimal Huffman codes [11]. In this example, $M = 7$, and each number was sampled from the following RPD:

$$P_0 = \tfrac{1}{2} \quad P_1 = \tfrac{1}{4} \quad P_2 = \tfrac{1}{8} \quad P_3 = \tfrac{1}{16} \quad P_4 = \tfrac{1}{32} \quad P_5 = \tfrac{1}{64} \quad P_6 = \tfrac{1}{128} \quad P_7 = \tfrac{1}{128}$$

The first step is to write each of these probabilities as a fraction of the form $\frac{N_i}{128}$:

$$N_0 = 64 \quad N_1 = 32 \quad N_2 = 16 \quad N_3 = 8 \quad N_4 = 4 \quad N_5 = 2 \quad N_6 = 1 \quad N_7 = 1$$

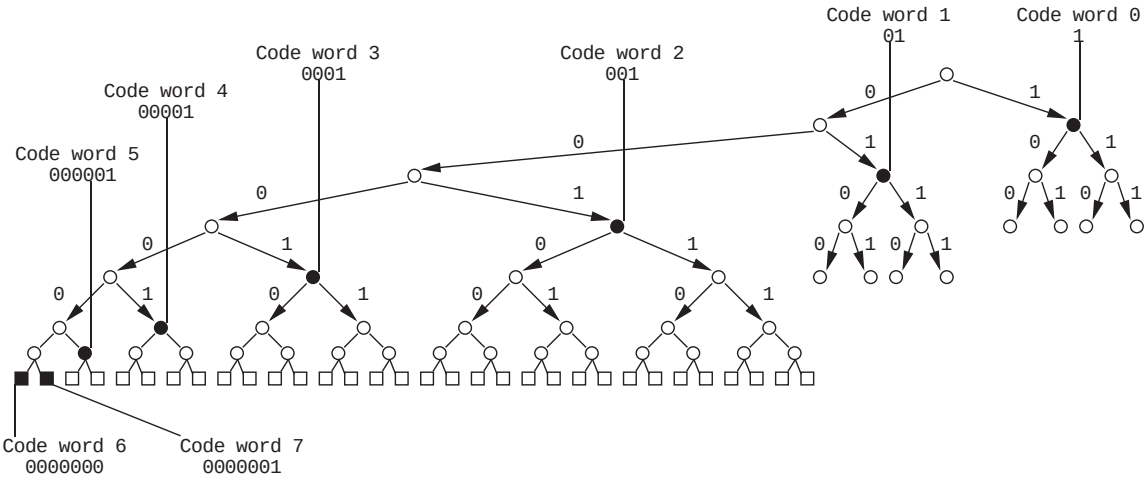

Figure 4: A coding tree

We then construct a binary tree with 128 leaves. An abbreviated tree (with the 32 leftmost leaves) is shown in Figure 4. We mark all left branches with '0' and all right branches with '1'. We allocate N_i consecutive leaves to the i^{th} symbol (define S_i to be the set of leaves allocated to symbol i). For each symbol i, we define H_i, the *highest ancestor* of S_i, to be that node in the tree whose descendant leaves are all elements of S_i (the highest ancestor for each symbol is indicated in Figure 4 by a black node). The codeword for symbol i is found by tracing the path to H_i. For example, we allocate $N_3 = 8$ leaves of the tree in Figure 4 to symbol 3, and define H_3 to be the node found by following the path 0001. Hence 0001 is the codeword for symbol 3.

The code tree is also used to decode a codeword. Given a codeword, we start at the root of the tree, and examine each bit of the codeword in turn, and follow the arc with that value. We stop this process when we arrive at highest ancestor, H_i, and we interpret the codeword as meaning symbol i.

Why we construct codes for RPDs

In the above, we required that each of the P_i have the same denominator, 2^y, so that we could construct a tree with 2^y leaves. We required that each P_i have a numerator of the form $N_i = 2^{z_i}$, so that we could allocate the leaves efficiently. If we order the N_i as $N_0 \leq N_1 \leq \ldots \leq N_M$, then we can guarantee an optimal allocation of leaves. For the example from Figure 4, we re-ordered the N_i before the code tree was constructed.

Calculating the Length of a Codeword

Let y be the height of the code tree, and z_i be the height of the tree rooted at H_i. The length of the codeword for symbol i is

$$y - z_i = log_2(2^y) - log_2(2^{z_i}) = -log_2(\frac{2^{z_i}}{2^y}) = -log_2(P_i)$$

2.4.2 Approximating a Probability Distribution by an RPD

We now consider approximating a discrete probability distribution by an RPD. We assume each P_i of the original distribution is rational, i.e., $\exists$ integers N_i, D_i such that $P_i = \frac{N_i}{D_i}$. We construct the modified RPD, P_i'', in the following way:

1. Let D' be the lowest common multiple of the D_i.

$$D' = lcm(D_0, D_1, \ldots, D_M)$$

2. Let $N_i' = \frac{N_i D'}{D_i}$ and rewrite each of the P_i as:

$$P_i = \frac{N_i'}{D'}$$

3. Let D'' be the smallest power of 2 such that $D'' \geq D'$.

4. Let N_i'' be the greatest power of 2 such that $N_i'' \leq N_i'$.

5. Set $P_i'' = \frac{N_i''}{D''}$

6. Set $P''_{UNUSED} = 1 - \sum_{i=1}^{M} P_i''$. We may have to add this extra (unused) codeword, since each $P_i'' \leq P_i$.

The P_i'' distribution is an RPD, and hence, we can use this distribution to construct a code dictionary, as described in Section 2.4.1. We note that the length of the codeword for symbol i is:

$$-log_2(P_i'') = ceiling(-log_2(P_i))$$

2.5 Example of Constructing a Code Dictionary

Consider describing a sequence of integers in the range [0..5] sampled from

$$P_0 = \frac{1}{3} \quad P_1 = \frac{16}{63} \quad P_2 = \frac{1}{7} \quad P_3 = \frac{8}{63} \quad P_4 = \frac{1}{9} \quad P_5 = \frac{2}{63}$$

We transform this probability distribution into the following RPD (Section 2.4.2).

$$P_0'' = \frac{16}{64} \quad P_1'' = \frac{16}{64} \quad P_2'' = \frac{8}{64} \quad P_3'' = \frac{8}{64} \quad P_4'' = \frac{4}{64} \quad P_5'' = \frac{2}{64} \quad P''_{UNUSED} = \frac{10}{64}$$

We then construct Code Dictionary 3 (Figure 5) from the RPD.

i	P_i	Codeword	$-log_2(P_i)$	$-log_2(P_i'')$
0 :	$\frac{1}{3}$	11	1.585	2
1 :	$\frac{16}{63}$	10	1.977	2
2 :	$\frac{1}{7}$	011	2.807	3
3 :	$\frac{8}{63}$	010	2.977	3
4 :	$\frac{1}{9}$	0011	3.170	4
5 :	$\frac{2}{63}$	00101	4.977	5

Figure 5: Code Dictionary 3

2.6 Coding A Sequence of Integers from a Non-Uniform Distribution

In this section, we consider how we may encode a sequence of integers $x_1,\ x_2,\ \ldots\ x_S$, independently sampled from a probability distribution P_i. The obvious approach is to construct a code dictionary (Section 2.4) and construct a message describing $x_1,\ x_2,\ \ldots\ x_S$ by concatenating the codewords for each symbol. Such a message is easy to construct and decode, but may be longer than necessary.

To demonstrate this inefficiency, consider sending a message describing the pair of numbers (2, 4) according to Code Dictionary 3. If we describe this pair by concatenating the codewords, we get the message '0110011' (length = 7 bits). Another possibility is to consider coding pairs of integers. To do this, we examine the probability of obtaining each possible pair of numbers:

Second Value	First Value					
	0	1	2	3	4	5
0	$\frac{1}{9}$	$\frac{16}{189}$	$\frac{1}{21}$	$\frac{8}{189}$	$\frac{1}{27}$	$\frac{2}{189}$
1	$\frac{16}{189}$	$\frac{256}{3969}$	$\frac{16}{441}$	$\frac{128}{3969}$	$\frac{16}{567}$	$\frac{32}{3969}$
2	$\frac{1}{21}$	$\frac{16}{441}$	$\frac{1}{49}$	$\frac{8}{441}$	$\frac{1}{63}$	$\frac{2}{441}$
3	$\frac{8}{189}$	$\frac{128}{3969}$	$\frac{8}{441}$	$\frac{64}{3969}$	$\frac{8}{567}$	$\frac{16}{3969}$
4	$\frac{1}{27}$	$\frac{16}{567}$	$\frac{1}{63}$	$\frac{8}{567}$	$\frac{1}{81}$	$\frac{2}{567}$
5	$\frac{2}{189}$	$\frac{32}{3969}$	$\frac{2}{441}$	$\frac{16}{3969}$	$\frac{2}{567}$	$\frac{4}{3969}$

We construct a code dictionary for these 36 possible events. The pair (2, 4) has a probability of $\frac{1}{7} \times \frac{1}{9} = \frac{1}{63}$, and hence the codeword for this event is $ceiling(-log_2(\frac{1}{63})) = 6$ bits long.

We now generalise this scheme to an arbitrary sequence of symbols. Let the j^{th} symbol have probability $p_{(k,j)}$ of being symbol k. In principle, we can enumerate all possible sequences. Let sequence i be $X_1,\ X_2,\ \ldots\ X_S$. We define the probability of getting sequence i as

$$P_i = \prod_{j=1}^{S} p_{(X_j,j)}$$

From this probability distribution over sequences, we may construct a code dictionary, which has a codeword for sequence i of length:

$$ceiling(-log_2(P_i)) \;=\; ceiling(-log_2(\prod_{j=1}^{S} p_{(X_j,j)})) \;=\; ceiling(-\sum_{j=1}^{S} log_2(p_{(X_j,j)}))$$

The extension of this scheme to constructing code dictionaries for dependent sequences is straightforward.

2.7 Coding Real Valued Numbers

In this section, we consider how we may go about constructing code dictionaries for real valued quantities, and estimating the length of the codewords. Encoding real valued numbers presents a different set of problems from encoding integer values:

- Firstly, we cannot encode an arbitrary real valued number to infinite precision in a finite message.
- Secondly, the values we measure (such as height or temperature) are only known to within a finite *accuracy of measurement* (AOM)[5]. We are typically given a value as a measurement $\pm \frac{AOM}{2}$. For example, the height of a person may be known with AOM $\epsilon = 1$ cm:

$$height \;=\; 170\,cm \;\pm\; 0.5\,cm$$

A convenient method for encoding real valued measurements, y, (e.g., heights) is to use a probability density function, $f(y)$, over those values, in conjunction with an AOM[6], ϵ. The algorithm Construct Code Dictionary (Figure 6) illustrates how a density function can be used in combination with an AOM to construct a code dictionary.

Construct Code Dictionary$(f,\ \epsilon)$

1. Divide the range of possible y values into cells of width ϵ.
2. Let cell i take y values in the range $[a_i\ ..\ b_i]$.
3. Construct a probability distribution over cells by setting

$$P_i \;=\; \int_{a_i}^{b_i} f(y)\ dy$$

   If this integral is difficult to evaluate we approximate it with:

$$P_i \;\propto\; \epsilon\ f(y_i)$$

   where y_i is the midpoint of cell i.

4. Construct a code dictionary to specify which cell the value falls in. Cell i has a codeword of length:

$$ceiling(-log_2(P_i))$$

Figure 6: An algorithm to construct a code dictionary for real valued measurements

[5] Also known as measurement error.

[6] We assume that the AOM is constant over the range of values we consider.

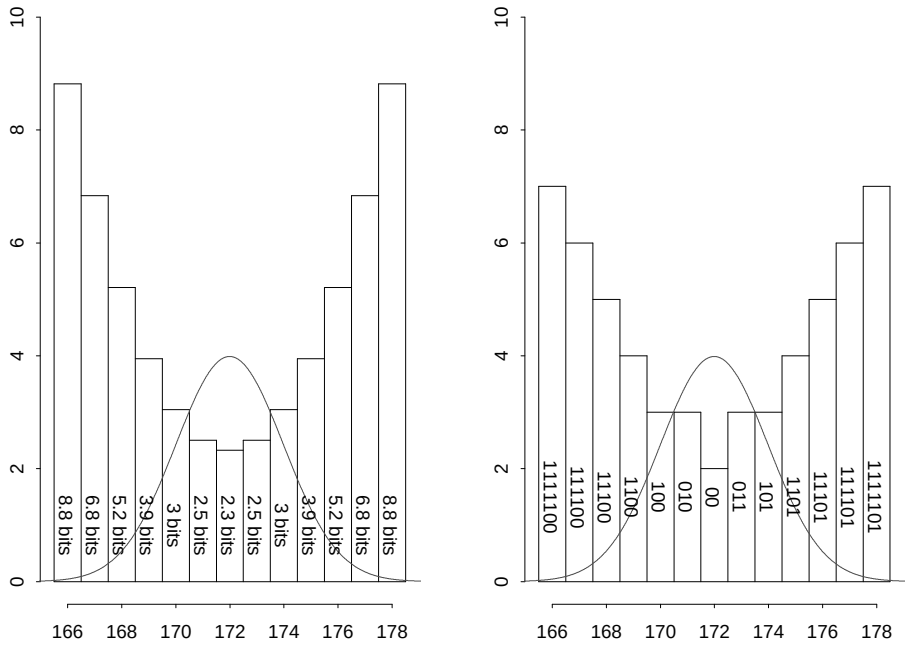

Figure 7: Coding schemes for a set of heights

Figure 7 demonstrates this process for the example where

- y represents heights of people,
- $f(y)$ is a normal density function with $\mu = 172$, and $\sigma = 2$, and
- $\epsilon = 1$ cm.

Figure 7(a) shows the code lengths we wish to assign to each height, and Figure 7(b)[7] gives a code dictionary for this combination of $f(y)$ and ϵ.

2.8 Working with Fractional Numbers of Bits

If we are constructing a bit string for a single height, we may have to resort to using an inefficient code dictionary such as the one depicted in Figure 7(b). However, we are typically considering describing a sequence of measurements. We can then use the method described in Section 2.6 which constructs a code dictionary which effectively allows us to use a fractional number of bits for a single measurement. For example, we could construct a code dictionary for describing a sequence of heights with code lengths very close to those given in Figure 7(a). Thus, a message describing the heights of two individuals with heights 168 cm and 177 cm would require $5.2 + 6.8 = 12$ bits. In effect, we have sent the height of 168 cm in a code length of 5.2 bits.

2.9 Summary of Coding Results

When using MML in practice, we make the following simplifications:

- We allow ourselves to work in fractions of bits.

[7] We have left those codewords beginning with 111111 for heights outside of the range [166 .. 178].

- Encoding an event which we believe is equally likely to have been from N possible events, requires a code length of $log_2 N$ bits.
- Encoding an event which we believe has probability P requires a code length of $-log_2 P$ bits.
- Encoding a real value, y, with an accuracy of measurement, ϵ, which we believe was sampled from a probability density $f(y)$ requires a code length of $-log_2(\epsilon\ f(y))$ bits.

3 Assessing a Model Using MML

In the introduction to this paper, we said that we wished to assess the quality of a model, with respect to some data D, by determining the length of the shortest string that describes D and the model together. One approach is to use two part messages:

- The *model part* of the message describes a model. Typically, this part describes a model which defines a probability distribution over data values.
- The *data part* of the message uses the model to describe the data values. This can be done by constructing a code dictionary from the probability distribution defined in the model part, and using this to encode the data values. In practice, we do not actually construct the code dictionary, we use the formulae from Section 2 to calculate the message length of the data part.

For the remainder of this section, we use two part messages to assess the quality of a unimodal normal model for a set of data consisting of heights of people.

3.1 Encoding Real Valued Parameter Values

Real valued parameters cannot be described to infinite precision in a finite message. A convenient way to encode real valued parameters is to use a similar encoding scheme to that used for real valued measurements (Section 2.7). A code dictionary for parameter values is constructed from a density function[8] which is divided into cells. We term the width of each cell the *accuracy of parameter value* (AOPV). Given a density function and an AOPV, it is a straightforward process to calculate the code length of a given parameter value.

3.1.1 Using a Uniform Density to Encode Parameter Values

Using a uniform density function to construct a code dictionary requires us to define a range over the parameter values. Consider the situation where we have a parameter value in the range $[a\ \ldots\ b]$, and we divide this range into cells AOPV wide. This divides the range into $ceiling(\frac{b-a}{AOPV})$ cells. Specifying to which cell a parameter value belongs requires $log_2\ ceiling(\frac{b-a}{AOPV})$ bits.

3.2 Example of Encoding a set of Heights

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

Figure 8: A set of Heights

Consider the problem of encoding the set of heights in Figure 8 [10] using a two part message. To illustrate, we encode these heights using a code dictionary constructed from a normal density function, such as the density function for heights discussed in Section 2.7. Thus, the message takes the following form:

[8] This density function plays the same role as a prior over parameter values in a Bayesian setting.

1. A statement of which density function we are using to encode the measurements with (in this case a normal density). If we are considering D distinct density functions, and we assume that one density is as likely as another, then the statement of which density function by index i requires $log_2 D$ bits.

2. Given the index i, the specification of the parameter values for this density function. In this case, this is the mean μ and the standard deviation σ.

3. Given the index i and the specification of the parameters, the specification of the data values using the normal density with parameters μ and σ.

3.2.1 Terminology

Throughout this example, and later sections, we have to identify clearly what we mean by a parameter value within the MML framework, since the coding techniques we use specify a cell in which the parameter value lies rather than specifying an exact value. We assume that the receiver of a message interprets the parameter value as being the midpoint of the specified cell. We use the following symbolism:

μ — is the mean as interpreted by the receiver, i.e., the midpoint of a cell.
m — is the sample mean of the data set.
σ — is the standard deviation as interpreted by the receiver.
s — is the sample standard deviation of the data set.

3.2.2 Assumptions

In this example, we assume that the sender and receiver have agreed prior to transmission that:

- Only two density functions will be considered for encoding the data values, the normal density, and the uniform density. If we assume that the data was as likely to have been sampled from a normal density as a uniform density, then the statement of which density to use requires 1 bit. For the remainder of this section, we shall ignore this part of the message.

- Uniform density functions will be used to encode μ and σ, and

$$\mu \text{ comes in the range } [100\ cm\ ..\ 244\ cm] \text{ and } AOPV_\mu = 12\ cm.$$

$$\sigma \text{ comes in the range } [0\ cm\ ..\ 50\ cm] \text{ and } AOPV_\sigma = 2\ cm.$$

- The measurements were made to an accuracy of measurement of $AOM = 1$ cm.

3.2.3 Calculation of Message Length

The statement that we will encode the values with a normal density requires $log_2 2 = 1$ bit. For the set of heights in Figure 8, we find that the sample mean $m = 174.2$ cm and the sample standard deviation $s = 5.23$ cm. We can encode these values using a code length of:

$$log_2(\frac{244\ -\ 100}{12}) = log_2 12 = 3.58 \text{ bits for } \mu, \text{ and}$$

$$log_2(\frac{50\ -\ 0}{2}) = log_2 25 = 4.64 \text{ bits for } \sigma.$$

However, the receiver of this message can only decode an approximation to these values. The receiver of this message will interpret:

$$\mu \text{ as being in the cell } (172,\ 184), \text{ and}$$

$$\sigma \text{ as being in the cell } (4,\ 6).$$

At this point in transmission, both the sender and receiver construct code dictionaries assuming that μ and σ take the midpoints of their cells, i.e., they construct code dictionaries from a $N(\mu = 178,\ \sigma = 5)$ density function (using the approach described in Section 2.7). Figure 9(a) gives the code lengths for each height using this code dictionary. This results in a total message of length:

$$3.58\ +\ 4.64\ +\ 97.08\ =\ 105.30 \text{ bits}$$

Figure 9 (a)
Using $\mu = 178, \sigma = 5$,
$AOPV_\mu = 12, AOPV_\sigma = 2$

3.91, 4.69, 7.80, 3.65, 11.03,
5.49, 4.37, 3.91, 4.11, 5.98,
4.11, 5.06, 4.11, 4.69, 3.76,
3.68, 3.76, 4.37, 4.69, 3.91,
Data Part Code Length: 97.08 bits

Figure 9 (b)
Using $\mu = 174.2, \sigma = 5.23$,
$AOPV_\mu = 0.05, AOPV_\sigma = 0.005$

4.81, 6.05, 5.33, 4.01, 7.38,
4.09, 3.68, 3.66, 3.65, 4.32,
3.65, 3.90, 3.65, 3.77, 4.49,
4.22, 3.73, 3.68, 3.77, 4.81
Data Part Code Length: 86.65 bits

Figure 9: The Code Lengths for the set of Heights

Figure 9(b) gives the code lengths using a code dictionary constructed from $\mu = 174.2$ cm and $\sigma = 5.23$ cm. We note, however, that stating μ and σ to this level of precision requires $AOPV_\mu = 0.05$ cm and $AOPV_\sigma = 0.005$ cm. This results in the code length describing the parameters to be quite long, namely:

$$log_2(\frac{244 - 100}{0.05}) = log_2 2880 = 11.49 \text{ bits for } \mu, \text{ and}$$

$$log_2(\frac{50 - 0}{0.005}) = log_2 10000 = 13.29 \text{ bits for } \sigma.$$

This results in a total message of length:

$$11.49 + 13.29 + 86.65 = 111.43 \text{ bits}$$

Thus, while using the more accurate parameter values led to a smaller data part of the message, the total message length was greater because of the extra model part length required to encode these more accurate values.

3.3 Issues Related to Encoding Parameter Values

The approach of using cells to encode parameters in the example in Section 3.2 raises three important issues:

1. Firstly, we note that the message lengths calculated are dependent upon minor details of the coding process. To avoid this problem, we calculate an expected message length. This is discussed in detail in Section 3.3.1.

2. Secondly, we must determine a method for establishing an appropriate value for the AOPV. If we state parameter values to a high accuracy, then the data part of the message will have nearly the optimal length, but the model part of the message will be lengthy. Stating the parameter values to a low level of accuracy causes the data part of the message to be unnecessarily long.

3. Thirdly, since we are allowing the AOPVs to depend upon the data, we must establish a method for sending the AOPVs to the receiver. This is discussed in Section 3.3.4.

3.3.1 Calculating Expected Message Lengths

The approach of using cells to specify parameter values suffers some undesirable properties. The parameter values (since they are interpreted by the receiver as being the midpoint of a cell) and the length of messages can vary considerably by changing small details of the encoding scheme.

To illustrate these problems, consider the encoding of μ from the example in Section 3.2 using the cell $C_1 = (172, 184)$ (Figure 9(a)), versus the situation of using the cell $C_2 = (168, 180)$. When we encode the data using C_1, the receiver sets $\mu = 178$; when we use C_2 the receiver sets $\mu = 174$. The use of C_1 results in 97.08 bits being required for the data part of the message; the use of C_2 results in 88.77 bits being required for the data part.

Various encoding details (such as the starting point of the range of a parameter, and the value of the AOPV for that parameter) can effect the midpoints of cells and hence parameter values, and message lengths. We therefore treat the parameter value interpreted by the receiver as a random variable. Using the principle of indifference, we assume that a parameter value is uniformly distributed locally around the sample parameter value. In particular, we assume that a parameter value is uniformly distributed in a region which is of size AOPV of that parameter [24, page 244]. For example, we assume that μ is uniformly distributed in the region

$$[m - \frac{AOPV_\mu}{2} \; .. \; m + \frac{AOPV_\mu}{2}]$$

and hence we view it as being equally likely that we code μ as 178, as it is that we code μ as 174. To remove the arbitrariness of which encoding scheme we use, we calculate the expected message length over the region, and state that the MML estimate for the parameter is that value which minimises the expected message length.

3.3.2 Deriving the AOPVs for a Normal Distribution

Symbol	Definition
μ	mean used to code x_i's
σ	s.d. used to code x_i's
m	sample mean
s	sample s.d.
$\bar{s}$	unbiased sample s.d.
Δ_μ	$\mu - m$

In this section, we derive estimates for the optimal AOPVs for a normal distribution. We assume that

- The data (consisting of $x_1 \ \ldots \ x_N$) is fixed, and hence the sample mean and variance are also constants.

$$m = \frac{\sum_{i=1}^{N} x_i}{N} \qquad s^2 = \frac{\sum_{i=1}^{N} (x_i - m)^2}{N}$$

- The message takes the form:
  (i) a statement of μ,
  (ii) a statement of σ, and
  (iii) a description of the x_i's.

We are attempting to find the values of μ, σ, $AOPV_\mu$ and $AOPV_\sigma$ which minimise the length of such a message. To make this derivation more compact, let

$$\bar{s}^2 = \frac{\sum_{i=1}^{N} (x_i - m)^2}{N - 1} \tag{1}$$

$$\Delta_\mu = \mu - m \tag{2}$$

The total message length is:

$$MessLen = code\_length(\mu) + code\_length(\sigma) + code\_length(data) \tag{3}$$

From Section 3.1.1,

$$code\_length(\mu) = log_2 \frac{range_\mu}{AOPV_\mu} \qquad code\_length(\sigma) = log_2 \frac{range_\sigma}{AOPV_\sigma} \tag{4}$$

From Section 2.7,

$$code\_length(data) = \sum_{i=1}^{N} - log_2 \ [\ f(x_i) \ \times \ \epsilon \ ] \tag{5}$$

$$= \sum_{i=1}^{N} - log_2 \left[\ \frac{\epsilon}{\sigma \sqrt{2\pi}} \ e^{- \frac{(x_i - \mu)^2}{2\sigma^2}} \ \right] \tag{6}$$

Rewriting Equation 3 with $(x_i - \mu)^2$ in terms of s^2 and ${\Delta_\mu}^2$

$$MessLen = log_2 \frac{range_\mu}{AOPV_\mu} + log_2 \frac{range_\sigma}{AOPV_\sigma} + N \ log_2 \ \frac{\sigma \sqrt{2\pi}}{\epsilon} + N \ \frac{s^2 + {\Delta_\mu}^2}{2\sigma^2} \ log_2(e) \tag{7}$$

We see immediately that the cell for μ which minimises Equation 7 is the cell containing m (and hence the MML estimate of μ is m). We could attempt to determine the value of $AOPV_\mu$ which minimises Equation 7. However, due to the arguments discussed in Section 3.3.1, it is more sensible find the value of $AOPV_\mu$ which minimises the expected message length. To do this, we assume that μ is distributed uniformly in the region $[m - \frac{AOPV_\mu}{2} \ .. \ m + \frac{AOPV_\mu}{2}]$ (Section 3.3.1). Therefore Δ_μ is distributed uniformly in the range $[-\frac{AOPV_\mu}{2} \ .. \ \frac{AOPV_\mu}{2}]$, and hence[9]

$$E(\Delta_\mu^2) = \frac{{AOPV_\mu}^2}{12} \tag{8}$$

[9] A standard result for the expectation of the square of a uniformly distributed random variable.

We find the expected message length (taking the expectation with respect to μ varying within the cell):

$$E(MessLen) \;=\; log_2 \frac{range_\mu}{AOPV_\mu} \;+\; log_2 \frac{range_\sigma}{AOPV_\sigma} \;+\; N\; log_2\; \frac{\sigma\;\sqrt{2\pi}}{\epsilon} \;+\; N\; \frac{s^2 \;+\; \frac{AOPV_\mu{}^2}{12}}{2\sigma^2}\; log_2(e) \qquad (9)$$

Differentiating $E(MessLen)$ w.r.t. $AOPV_\mu$ gives us:

$$\frac{\partial E(MessLen)}{\partial AOPV_\mu} \;=\; log_2(e) \;\times\; \left(-\frac{1}{AOPV_\mu} \;+\; \frac{N\; AOPV_\mu}{12\;\sigma^2}\right) \qquad (10)$$

Setting $\frac{\partial E(MessLen)}{\partial AOPV_\mu}$ to 0 and solving for $AOPV_\mu$ gives us the optimal value:

$$AOPV_\mu \;=\; \sigma\;\sqrt{\frac{12}{N}} \qquad (11)$$

Using a similar approach, Wallace and Boulton [23] established that the optimal value for σ (and hence the MML estimate for σ) is the unbiased estimate of the standard deviation:

$$\sigma \;=\; \bar{s} \qquad (12)$$

and that the optimal value of $AOPV_\sigma$ is[10]:

$$AOPV_\sigma \;=\; \bar{s}\;\sqrt{\frac{6}{N-1}} \qquad (13)$$

The minimum expected message length is then approximately:

$$E(MessLen) \;=\; log_2 \frac{range_\mu}{AOPV_\mu} \;+\; log_2 \frac{range_\sigma}{AOPV_\sigma} \;+\; N\; log_2\; \frac{\bar{s}\;\sqrt{2\pi}}{\epsilon} \;+\; N\; \frac{s^2 \;+\; \frac{\bar{s}^2}{N}}{2\bar{s}^2}\; log_2(e) \qquad (14)$$

The derivation presented above demonstrates that the MML estimate for the mean of a normal distribution is the sample mean (from Equation 7), and derives the AOPV for the mean (Equation 11). The above sketches out how we may derive the MML estimate and the AOPV for the standard deviation, and the expected message length for a set of measurements.

3.3.3 The Expected Message Length of the Height Example

Here, we calculate the expected message length, using Equation 14. To do this, we determine the optimal values for the AOPVs and calculate the following values:

m	s	$\bar{s}$	N	$AOPV_\mu$	$AOPV_\sigma$
174.2	5.23	5.37	20	4.16	3.02

$$\begin{aligned} E(MessLen) &= log_2 34.64 \;+\; log_2 16.58 \;+\; 20\; log_2\; 13.45 \;+\; 20 \times 0.499 \times 1.443 \\ &= 5.11 \;+\; 4.05 \;+\; 74.99 \;+\; 14.40 \\ &= 98.55 \end{aligned}$$

Here, we attempt to satisfy the reader that the approximations made in the derivation of the expected message length may be reasonable. With the optimal AOPVs from Section 3.3.3, namely $AOPV_\mu = 4.16$ and $AOPV_\sigma = 3.02$, we ran simulations using the region

$$\mu \text{ in the range } (m - \frac{AOPV_\mu}{2},\; m + \frac{AOPV_\mu}{2}) \quad (172.12,\; 176.28)$$

$$\sigma \text{ in the range } (\bar{s} - \frac{AOPV_\sigma}{2},\; \bar{s} + \frac{AOPV_\sigma}{2}) \quad (3.86,\; 6.88)$$

We split this region into 121 (11 by 11) equally spaced points. Each of these points defines a value for μ and σ, and hence we can use Equation 7 to calculate the message length as if those were the values of μ and σ. We evaluated the message length at each point, and the average message length of these 121 points was 99.42 bits, while the (approximate) expected message length given by Equation 14 was 98.55 bits. Furthermore, the message length in this region varied from 97.86 bits (when $\mu = m$, $\sigma = \bar{s}$) to 103.61 bits (when $\mu = 176.09$, $\sigma = 3.99$).

[10] In the derivation of this result, Equation (27) [23, Page 188] should read $\frac{c^2}{12w^2} = \frac{1}{2(n-1)}$

3.3.4 Sending AOPVs

The two part messages used in this section cannot be interpreted unless the receiver knows in advance the AOPVs used by the sender. This is an unreasonable assumption, since we are allowing the AOPVs to depend upon the data (which the message is describing). Hence, we must consider three part codes, the first part describes the AOPVs, in addition to the model part, and the data part. This would appear to lead to an infinite regress, since now we need a coding scheme to describe AOPVs. Wallace and Freeman [24, page 245] argue that in many cases a three part code is not necessary. They offer methods to avoid including a preamble describing the accuracy of the AOPVs.

4 Codes for Non-Regular Structures

The basic principles of MML can be applied to a wide range of model structures. In this section, we consider codes for non-regular structures such as classification trees[11] [5] and finite state automata [8, 9]. MML has been applied to a range of other non-regular problems including Clustering [23, 4], DNA string alignment [1] and Decision Graphs [12].

4.1 Classification Trees

4.1.1 Description of Classification Trees

A classification tree is a non-parametric model which is used to perform classification. A tree partitions an object space into regions, and associates a probability for each class with each region. A new object is classified by examining the probabilities in the region where it falls.

A tree from the Voting domain (described in [21]) is illustrated in Figure 10. In this domain, each object represents a member of the US congress, the class represents whether they are from the Republican or Democratic party, and the attributes take their values from the member's voting pattern on a number of issues. A member may vote 'Yes', 'No', or '?' (which means the member didn't vote or abstained).

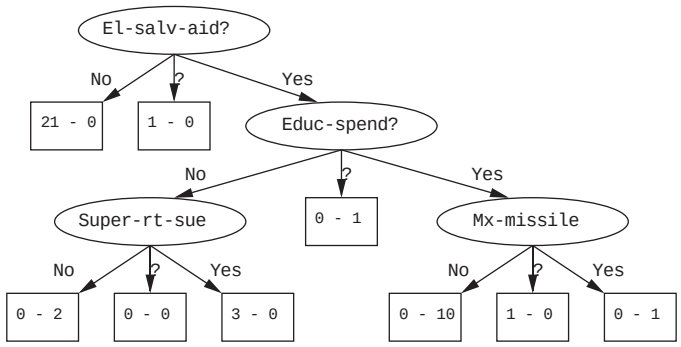

Figure 10: A Classification Tree

A tree consists of *split nodes* (drawn as ovals) and *leaves* (drawn as rectangles). A new object is classified by following an appropriate path down the tree; if that person voted 'Yes' on the "Aid to El-salvador" issue, the right hand path from the root is taken. If they voted "Yes" to "Education Spending" the next right hand path is taken. Ultimately each person will end up at a leaf node. Associated with each leaf node is the probability of being a Democrat; people assigned to this node are classified according to this probability. The class which is associated with a particular leaf will be chosen according to the proportion of training objects from each class which would be assigned to that leaf; for example, if the leaf has more Democrats than Republicans, then this suggests that objects falling in this leaf are more likely to be Democrats.

[11] Termed Decision Trees in the Machine Learning literature [14, 15].

4.1.2 Codes for Classification Trees

We assume the following coding conventions. Both the sender and receiver know the number of attributes, the arity of each attribute, have agreed upon an order for attributes, and have agreed upon values for two parameters, P_L and P_S, which estimate the probability that a node is a leaf, or a split node, respectively.

An unambiguous message which describes a tree can be constructed by performing a prefix traversal of the tree describing each node. Describing a node requires a code length of $-log_2 P_L$ if it a leaf, and $-log_2 P_S$ if it is a split node. If it is a split node then we must specify the attribute that it splits on, which requires $log_2(number\ of\ candidate\ attributes)$ bits. To illustrate this encoding scheme, we shall encode the tree, t_1 from Figure 2. The model part of the message takes the form:

```
Split El-salv-aid
        Leaf
        Leaf
        Split Educ-spend
                Split Super-rt-sue   Leaf   Leaf   Leaf
                Leaf
                Split Mx-missile     Leaf   Leaf   Leaf
```

If we take $P_L = P_S = \frac{1}{2}$, then we can encode a split node with a '1' and a leaf with a '0'. We may specify the attribute used to split the root with the following codewords (assuming that there are only four attributes available):

```
El-salv-aid     00        Educ-spend      01
Super-rt-sue    10        Mx-missile      11
```

The model part of the message is 18.585 bits long:

$$1\ 00\ 0\ 0\ 1\ X^{12}\ 1\ 0^{13}\ \ 0\ 0\ 0\ 0\ 1\ 1^{14}\ 0\ 0\ 0,$$

where X (the specification of the attribute Educ-spend) is a code length of log_2 3 bits. The encoding is most efficient when P_L is the proportion of leaves in the tree, and P_S is the proportion of splits in the tree [25]. Using this encoding method results in a message of length 17.16 bits.

4.2 Probabilistic Finite State Automata

4.2.1 Description of Probabilistic Finite State Automata

A Probabilistic Finite State Automata (PFSA) is a state machine which can be used to explain how a string was generated. A PFSA (such as the one in Figure 11) begins in the initial state, and parses each symbol of the string. A symbol is parsed by following the arc from the current state marked with this symbol (states are shown as circles in Figure 11). Each time a symbol is parsed, the automaton changes state. The new state is found by following the arc labelled by the current input symbol. When the automata parses the delimiter "/", it returns to the initial state.

Georgeff and Wallace [8, 9] presented a method for inferring probabilistic finite state automata (PFSA) from a sequence of symbols. For example, the PFSA shown in Figure 11 was inferred from the string[15]:

$$S: \quad CAAAB/BBAAB/CAAB/BBAB/CAB/BBB/CB/$$

4.2.2 Codes for Probabilistic Finite State Automata

We could encode a string using a PFSA by (1) describing the PFSA, and then (2) describing the string using the PFSA. In fact, we use a message which describes the data and PFSA in a combined form. That is, we describe the components of the PFSA as we use them. This message takes the form:

$$N,\ Delim,\quad <S_0,\ next\_state_1>,\quad <S_1,\ next\_state_2>,\quad \ldots\quad <S_{M-1},\ finish\_state>$$

[12] At this point in the tree, there are three available attributes.

[13] At this point in the tree, there are only two available attributes. Hence, we can use '0' to specify Super-rt-sue and '1' to specify Mx-missile.

[14] To specify Mx-missile.

[15] We note that Georgeff and Wallace [9] left off the final delimeter in string S.

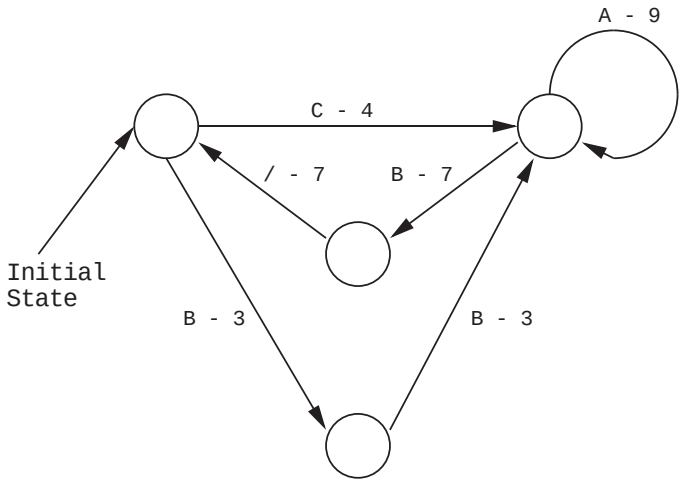

Figure 11: A PFSA with 4 states

where N is the number of states in the PFSA, $Delim$ is the number of delimiters in S, and S_t is the t^{th} symbol in the string. If we assume that N is equally likely to be any value < 100, then we encode N with a code length of $log_2\ 100$ bits. If we assume that $Delim$ is equally likely to be any value < 100, then we encode $Delim$ with a code length of $log_2\ 100$ bits. At each point of transmission, if the current_state is j, the current symbol is X, and the next_state is k, we encode S_t with a codeword of length $-log_2(Prob(S_t = X,\ current\_state = j))$, and $next\_state$ with a codeword of length $-log_2(Prob(next\_state = k,\ current\_state = j))$.

To calculate these probabilities, we define at step t of transmission (i.e., having transmitted $S_1 \ \ldots \ S_{t-1}$):

- $Q_{t,j}(X)$ to be the number of occurrences of the symbol X when the automaton was in *state* j.
- $R_{t,j}(X,k)$ to be the number of times the automaton has progressed from *state* j to *state* k when X was the input symbol being processed.
- $T_{t,j}$ to be the number of times the automaton was in *state* j.

For each state j, we initialise $Q_{0,j}(X)$ and $R_{0,j}(X,k)$ to be 0. We use Laplace's rule of succession to estimate the probability that we parse symbol X if we are in state j at time t

$$Prob(S_t = X,\ current\_state = j) = \frac{Q_{t,j}(X)+1}{T_{t,j}+4} \tag{15}$$

and the probability that we change to state k if we parse symbol X while in state j at time t

$$Prob(next\_state = k,\ current\_state = j) = \frac{R_{t,j}(X,k)+1}{Q_{t,j}(X)+N} \tag{16}$$

In Equation 15, the term $T_{t,j}+4$ appears on the denominator, since there are 4 symbols in the alphabet. In Equation 16, the term $Q_{t,j}(X)+N$ appears on the denominator, since there are N states we may go to.

We also note that the sender may decide to number the states $2 \ \ldots \ N$ in any order she wishes. Hence we may subtract a length of $N-1!$ from the final message length.

The string S can be encoded in a message of length:

$$
\begin{array}{lll}
N: & & log_2(100) \\
Delim: & + & log_2(100) \\
CAAAB/: & - & log_2(\frac{1}{4}) - log_2(\frac{1}{4}) - log_2(\frac{2}{5}) - log_2(\frac{3}{6}) - log_2(\frac{1}{7}) - log_2(\frac{1}{4}) \\
 & - & log_2(\frac{1}{4}) - log_2(\frac{1}{4}) - log_2(\frac{2}{5}) - log_2(\frac{3}{6}) - log_2(\frac{1}{4}) \\
BBAAB/: & - & log_2(\frac{1}{5}) - log_2(\frac{1}{4}) - log_2(\frac{4}{8}) - log_2(\frac{5}{9}) - log_2(\frac{2}{10}) - log_2(\frac{2}{5}) \\
 & - & log_2(\frac{1}{4}) - log_2(\frac{1}{4}) - log_2(\frac{4}{7}) - log_2(\frac{5}{8}) - log_2(\frac{2}{5}) \\
CAAB/: & - & log_2(\frac{2}{6}) - log_2(\frac{6}{11}) - log_2(\frac{7}{12}) - log_2(\frac{3}{13}) - log_2(\frac{3}{6}) \\
 & - & log_2(\frac{2}{5}) - log_2(\frac{6}{9}) - log_2(\frac{7}{10}) - log_2(\frac{3}{6}) \\
BBAB/: & - & log_2(\frac{2}{7}) - log_2(\frac{2}{5}) - log_2(\frac{8}{14}) - log_2(\frac{4}{15}) - log_2(\frac{4}{7}) \\
 & - & log_2(\frac{2}{5}) - log_2(\frac{2}{5}) - log_2(\frac{8}{11}) - log_2(\frac{4}{7}) \\
CAB/: & - & log_2(\frac{3}{8}) - log_2(\frac{9}{16}) - log_2(\frac{5}{17}) - log_2(\frac{5}{8}) \\
 & - & log_2(\frac{3}{6}) - log_2(\frac{9}{12}) - log_2(\frac{5}{8}) \\
BBB/: & - & log_2(\frac{3}{9}) - log_2(\frac{3}{6}) - log_2(\frac{6}{18}) - log_2(\frac{6}{9}) \\
 & - & log_2(\frac{3}{6}) - log_2(\frac{3}{6}) - log_2(\frac{6}{9}) \\
CB/: & - & log_2(\frac{4}{10}) - log_2(\frac{7}{19}) - log_2(\frac{7}{10}) \\
 & - & log_2(\frac{4}{7}) - log_2(\frac{7}{10}) \\
 & - & log_2(3!)
\end{array}
$$

5 Properties of Codes

When designing codes for MML applications, the codes we use should:

- Be a prefix code. A code dictionary is prefix if no codeword consists of another codeword followed one or more binary digits. If a code is not prefix then it may be impossible for a receiver to interpret a given message.

- Be a non-redundant code (discussed in Section 5.1).

- Imply a sensible prior distribution over theories. There is debate over the role of the prior distribution implied by a code (discussed in Section 5.3 and Baxter and Oliver [3]).

5.1 Non-Redundant Codes

A code dictionary for theories is non-redundant if there are not two (or more) codewords for the same theory. Similarly, if a single theory can be used to describe a set of measurements in two (or more) distinct ways, then the coding scheme for the data is redundant.

Quinlan and Rivest [16] suggested a redundant code for describing N data items which were split into 2 classes. Their message took the form:

- A specification of the most common class (1 bit).

- A specification how many items Z were in the most common class. Z is a number between $ceiling(\frac{N}{2})$ and N and, hence, requires $log_2(ceiling(\frac{N}{2}+1))$ bits to specify.

- A specification of which of the N items are in the least common class. There are ${}^{N}C_{Z}$ possible ways in which we may group the items into a group containing Z items, and another group containing $N-Z$ items. Hence this requires $log_2({}^{N}C_{Z})$ bits to specify.

The redundancy in this coding scheme becomes apparent when N is even, and there are $\frac{N}{2}$ items in each class. Consider the situation where $N=8$ and there are 4 items in Class 1 and 4 items in Class 2. We can describe this data with two distinct messages; one which specifies Class 1 as being the most common class, and the other specifies Class 2 as being the most common class.

5.2 The Prior Distribution Implied by a Code

A non-redundant prefix code for theories implies a probability distribution over those theories. Such codes satisfy the Kraft inequality [7]:

$$\sum_{m_i \in M} 2^{-message\_length(m_i)} \leq 1 \tag{17}$$

If the code is also *efficient* then

$$\sum_{m_i \in M} 2^{-message\_length(m_i)} = 1 \tag{18}$$

Therefore, we may construct a normalised probability distribution over theories by assigning:

$$Prob(m_i) = 2^{-message\_length(m_i)} \tag{19}$$

5.3 MML and MDL

There is debate as to how prior knowledge should be used. Wallace and Freeman [24, page 241] take a Bayesian interpretation of Minimum Encoding Inference and conclude:

> "there can be no substitute for careful specification of whatever prior knowledge is available"

Rissanen [20] acknowledges the relation between coding and Bayesian priors. However, he rejects the notion that a probability distribution can

> "capture prior knowledge in an adequate manner".

When considering a prior distribution over a parameter value, he is hesitant about

> "the interpretation difficulty whenever the parameter appears to be a constant — albeit unknown".

Instead the majority of the MDL work uses the concept of a *universal* prior distribution over the set of positive integers to represent complete prior ignorance [18].

Similarities and differences between MML and MDL are discussed in Baxter and Oliver [3].

5.4 Relationship with Bayesianism

MML can be interpreted as a form of Bayesianism. Within a Bayesian framework, we attempt to determine the posterior probability of a model given the data:

$$Posterior(m_i) = Prob(m_i \mid data) = \frac{Prob(m_i) \times Prob(data \mid m_i)}{Prob(data)} \tag{20}$$

where we view $Prob(m_i)$ as the prior probability of m_i.

In cases where we are considering a finite number of models, MML and Bayesianism are very similar approaches. We may maximise the posterior probability of a model by maximising

$$P = Prob(data \mid m_i) \times Prob(m_i) \tag{21}$$

Maximising P is equivalent to minimising the message length:

$$\begin{aligned} -log_2 P &= -log_2 \left(Prob(m_i) \times Prob(data \mid m_i) \right) && (22)\\ &= -log_2 Prob(m_i) + -log_2 Prob(data \mid m_i) && (23)\\ &= code\_length(m_i) + code\_length(data \mid m_i) && (24) \end{aligned}$$

However, when we consider a continuum of models, MML and Bayesianism differ. In a Bayesian framework, we do not construct the posterior probability of a model, but construct a posterior density over models. The MML approach effectively partitions the posterior density to estimate the posterior probability of models. These issues are discussed at length in Oliver and Baxter [13].

6 Criteria Used for Model Selection

Many criteria have been suggested for selecting a model given some data. Examples of such criteria are

- The sums of squares of residuals when fitting a straight line to some data, and
- The likelihood of obtaining the data, when performing mixture modelling.

Typically, each of these criteria is a summary statistic which hopefully reflects how well the model explains the data. A range of optimisation techniques are used to maximise (or minimise) this criterion, and select that model which has the maximum (or minimum) value of the criterion.

For example, in simple least squares linear regression, when fitting a straight line $y = mx + c$ to some (x_i, y_i) points, we minimise the sum of the squared residuals. Here the model consists of a straight line with two parameter m and c, and the criterion minimised is the sum of squares of the residuals.

We distinguish a class of criteria: *model complexity criteria* (MCC) — those criteria which take into account the complexity of the model. Traditional criteria (such as likelihood, sums of squares of residuals) have no terms which reflects the complexity of the model. This is entirely sensible when dealing with models of similar complexity, but this approach breaks down when comparing models of distinct complexity classes. For example, by taking enough terms in a polynomial function, one can exactly fit any arbitrary distribution of a finite number of points [2].

Typically the MCCs have two terms, the first reflects how complex the model is, and the second reflects how well the data fits the model:

- Message Length — The complexity of the model is measured as the length of the model part of the message, and the goodness of fit is measured by the length of the data part of the message.
- Penalised Likelihood — The complexity of the model is measured as a multiple of the number of parameters in the model, and the goodness of fit is measured by the likelihood of the data given the model.
- Posterior Probability — In a Bayesian framework the complexity of the model is measured as the prior probability given to the model, and the goodness of fit is measured by the likelihood of the data given the model.

The Bayesian approach and the MML approach explicitly measure the complexity of the model and the goodness of fit in commensurate terms. Bayesian prior probabilities can be multiplied by probabilities of the data given the model. The code length of a model can be added to the code length of the data. The penalised likelihood method coerces the complexity of the model (measured by the number of parameters) into units that can be compared with a likelihood.

7 Conclusion

In this paper, we introduced the MML approach to inference. We described a range of coding techniques for data. We applied the MML approach to perform inference and model selection.

We examined fitting a normal distribution to some data points, and established that the MML estimates of the mean and standard deviation were the sample mean, and unbiased sample standard deviation. We then examined some non-regular problems (classification trees and probabilistic finite state automata), and presented coding scheme for these problems. Wallace and Freeman [24, page 252] conclude that:

> "It is in completely non-regular problems that the minimum message length approach will produce most strikingly different and, we believe, practically useful results".

8 Acknowledgments

This work was carried out with the support of the Defence Research Agency, Malvern. We would like to thank Chris Wallace and Wray Buntine for valuable discussions, and Andrew Webb, Richard Glendinning, Rohan Baxter, Catherine Scipione, Kevin Korb and Peter Tischer for critical reading of the manuscript. Thanks to Mathew Collins for pointing out a mistake in Section 4.2.

References

[1] 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.

[2] R.A. Baxter and D.L. Dowe. Model selection in linear regression using the MML criterion. In J.A. Storer and M. Cohn, editors, *Proc. 4'th IEEE Data Compression Conference*, page 498, Snowbird, Utah, March 1994. IEEE Computer Society Press, Los Alamitos, CA.

[3] R.A. Baxter and J.J. Oliver. MDL and MML: Similarities and differences. Technical report TR 207, Dept. of Computer Science, Monash University, Clayton, Victoria 3168, Australia, 1994. Available on the WWW from `http://www.cs.monash.edu.au/~jono`.

[4] D.M. Boulton and C.S. Wallace. A program for numerical classification. *Computer Journal*, 13:63–69, 1970.

[5] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[6] G.J. Chaitin. Information-theoretic computational complexity. *IEEE Transactions on Information Theory*, 20:10–15, 1974.

[7] T.M. Cover and A.T. Joy. *Elements of Information Theory.* John Wiley and Sons, Inc., New York, 1991.

[8] M.P. Georgeff and C.S. Wallace. A general criterion for inductive inference. In T. O'Shea, editor, *Advances in artificial intelligence : proceedings of the Sixth European Conference on Artificial Intelligence*, pages 473–482, Amsterdam, 1984. North Holland.

[9] M.P. Georgeff and C.S. Wallace. A general selection criterion for inductive inference. Technical report TR 44, Dept. of Computer Science, Monash University, Clayton, Victoria 3168, Australia, June 1984.

[10] D.J. Hand, F. Daly, A.D. Lunn, K.J. McConway, and E. Ostrowski. Data set 231: Husbands and wives. In *Small Data Sets*, pages 179–181. Chapman and Hall, London, 1994.

[11] D.A. Lelewer and D.S. Hirschberg. Data compression. *ACM Computing Surveys*, 19:261, 1987.

[12] J.J. Oliver. Decision graphs - an extension of decision trees. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 343–350, 1993. Extended version available as TR 173, Dept. of Computer Science, Monash University, Clayton, Victoria 3168, Australia. Available on the WWW from `http://www.cs.monash.edu.au/~jono`.

[13] 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. Available on the WWW from `http://www.cs.monash.edu.au/~jono`.

[14] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[15] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

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

[17] Robert F. Rice. Some practical noiseless coding techniques, Part II, Module PSI14,K+. JPL Publication 91-3, Jet Propulsion Laboratories, November 1991.

[18] J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11:416–431, 1983.

[19] J. Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society (Series B)*, 49:223–239, 1987.

[20] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, Singapore, 1989.

[21] J. Schlimmer and R. Granger. Incremental learning from noisy data. *Machine Learning*, 1:317–354, 1986.

[22] C.S. Wallace. Classification by minimum-message-length inference. In G. Goos and J. Hartmanis, editors, *Advances in Computing and Information – ICCI '90*, pages 72–81. Springer-Verlag, Berlin, 1990.

[23] C.S. Wallace and D.M. Boulton. An information measure for classification. *Computer Journal*, 11:185–194, 1968.

[24] C.S. Wallace and P.R. Freeman. Estimation and inference by compact coding. *Journal of the Royal Statistical Society (Series B)*, 49:240–252, 1987.

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