RJ 9997 (89085) December 15, 1995
Computer Science

IBM Research Report

MDL Estimation with Small Sample Sizes including an Application to the Problem of Segmenting Binary Strings Using Bernoulli Models

Byron E. Dom

K54/B2
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099
e-mail: dom@almaden.ibm.com

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

MDL Estimation with Small Sample Sizes including an Application to the Problem of Segmenting Binary Strings Using Bernoulli Models

Byron Dom*

April 19, 1996

Abstract

Minimum Description Length (MDL) estimation has proven itself of major importance in a large number of appplications many of which are in the fields of computer vision and pattern recognition. A problem is encountered in applying the associated formulas, however, especially those associated with model cost. This is because most of these are asymptotic forms appropriate only for large sample sizes. Rissanen[27] has recently derived sharper codelength formulas valid for much smaller sample sizes. Because of the importance of these results, it is our intent here to present a tutorial description of them. In keeping with this goal we have chosen a simple application whose relative tractibility allows it to be explored more deeply than most problems: the segmentation of binary strings based on a piecewise Bernoulli assumption. By that we mean that the strings are assumed to be divided into substrings, the bits of which are assumed to have been generated by a single (within a substring) Bernoulli source. After investigating the solution of this problem, we will discuss the extension of these results to more important problems such as piecewise polynomial curve fitting and image segmentation, both of which we plan for future work.

1 Introduction

In this paper we describe some *objective functions* and an algorithm that uses these functions for segmenting 1-D binary strings. The importance of this problem to computer vision and pattern recognition lies in the following. The objective functions we use are based on the minimum description length principle (MDL), which has already proven itself quite useful in these fields. More importantly, two of the objective functions are based on new forms of the stochastic complexity proposed and derived quite recently by Rissanen[27]. These forms correct the most severe limitation of the previously used coldelength formulas: that they were asymptotic forms valid only for large numbers of data, the missing terms being quite large in some cases. Despite this limitation, quite useful results have been obtained using them to tackle important and difficult problems in the fields of computer vision and pattern recognition. For example:

- Image Segmentation: [18, 19, 14, 15, 22, 34, 12, 3, 36]

*Work performed while on sabbatical leave at Catholic University of Belgium (K.U. Leuven) during the calendar year of 1995.

- Multiple Curve Detection: [11, 10]
- Motion Analysis: [1, 3]
- Clustering (Unsupervised Learning): [32, 28]
- Feature Selection and Dimensionality Determination: [30, 31, 4, 33]
- Classifier Construction and Training (Supervised Learning). Examples:
  - Tree Classifiers: [29, 23, 5, 6, 20]
  - Neural Nets: [26, 35, 7, 16, 17]

The advantages of attacking the segmentation problem that we have chosen vis a vis our tutorial intent are the following:

1. pedagogical: the simplicity of the binary-string segmentation problem allows the one to consider the statistical technique while minimizing any confusion induced by the complexities of the associated application.
2. As will be described below, Rissanen's new formulas are based on something called a *maximum-likelihood code*. In most cases this code is not directly useable and one of Rissanen's constributions is in fact the derivation of a *sharp* ($o(1)$) approximation to the ML code. In the case of Bernoulli models, however, we have been able to derive an exact expression, which allows us to compare the behavior of the approximation to that of the exact expression.
3. The one-dimensional nature of the problem combined with the simplicity of the models permit an exhaustive search over all possible segmentations, thus separating the strength of the objective function from effects induced by suboptimal minimization algorithms.

2 A New Definition of Stochastic Complexity

MDL estimation involves computing what Rissanen[25] refers to as the *stochastic complexity* of the data given the "model class" (the set of all models that one wishes to consider). This complexity is simply the codelength required to encode the data using a particlar model-based encoding scheme that chooses its model from among those in the model class. In fact the associated model *is* the MDL estimate. Various forms have been proposed for this encoding scheme. In his recent paper[27] Rissanen proposes a scheme based on the *maximum-likelihood* (ML) probability distribution $\hat{P}(x^n|\xi)$ for the set of n measurements, x^n:

$$\hat{P}(x^n|\xi) \equiv \frac{P(x^n|\hat{\theta}^{(\xi)}(x^n),\xi)}{\sum_{u^n} P(u^n|\hat{\theta}^{(\xi)}(u^n),\xi)}, \tag{1}$$

where $\hat{\theta}^{(\xi)}(x^n)$ is the maximum-likelihood (ML) estimate of the parameter vector $\theta^{(\xi)}$, ξ is the *model structure* and the sum in the denominator is over all possible data vectors. The *model structure* ξ in the case of piecewise polynomial curve fitting (for example) would consist of a specification of the region boundaries and the polynomial orders within each region. Implicit in ξ is thus the number of components, k, in $\theta^{(\xi)}$ (i.e. $k = k(\xi)$). It should be clear that there is one such ML distribution for each model class and structure, ξ. Here we define the *stochastic complexity* to be the minimized code-length:

$$L^*(x^n) \;=\; \min_{\xi} \left[-\log \hat{P}(x^n|\xi) \;+\; L(\xi) \right], \tag{2}$$

In MDL-based estimation the minimizing value of ξ (in equation (2) is sought. [In most of what follows for notational simplicity we will omit the explicit dependence of various quantities on ξ and k except where required for clarity.]

In [27] Rissanen supplies some justification for using this form (equation (2). For example, it can be interpreted as two-part coding with redundancy removal where the following prior distribution is used to encode the parameters, $\hat{\theta}(x^n)$ with $\log \pi_n(\hat{\theta})$ bits:

$$\pi_n(\theta) \;=\; \frac{\sum_{\{x^n|\hat{\theta}(x^n)=\theta\}} P(x^n|\theta)}{\sum_{\{\forall u^n\}} P(u^n|\hat{\theta}(u^n))}. \tag{3}$$

[See Appendix A (section A.4) for a discussion of this prior and why it is a reasonable choice.]

It is a frequent practice to express the probabilities in (1) as:

$$P(x^n|\hat{\theta}(x^n)) \;=\; \int_{u^n \in \delta x^n} f(u^n|\hat{\theta}(u^n))du^n, \tag{4}$$

where δx^n is the quantization cell corresponding to the finite-precision x^n. This representation frequently leads to the approximation:

$$P(x^n|\hat{\theta}(x^n)) \;\approx\; f(x^n|\hat{\theta}(x^n)) \cdot \delta x^n, \tag{5}$$

which is reasonable as long as $f(x^n|\hat{\theta}(x^n))$ varies slowly over δx^n. Using this expression for P, leads to the following approximation for (1), which is referred to as the *maximum-likelihood density*:

$$\hat{f}(x^n) \;=\; \frac{f(x^n|\hat{\theta}(x^n))}{\int f(x^n|\hat{\theta}(x^n))dx^n} \tag{6}$$

For either of these (equations (1) and (6)) to be usable their denominators must (most importantly) remain finite and must also be possible to evaluate or at least approximate. The difficulty of achieving the latter of these requirements motivated Rissanen in [27] to derive the following approximate expression for the logarithm of the maximum-likelihood density (equation (6)):

$$-\log \hat{f}(x^n) \;=\; -\log f(x^n|\hat{\theta}(x^n)) \;+\; \frac{k}{2}\log\frac{n}{2\pi} \;+\; \log \int_{\Gamma} \sqrt{|I(\theta)|}d\theta \;+\; o(1), \tag{7}$$

where

- $I(\theta)$ is the *Fisher Information* matrix and $|I(\theta)|$ is its determinant. The $(i,j)^{th}$ component of $I(\theta)$ is
$$-E_\theta \left\{ \frac{\partial \log f(x|\theta)}{\partial\theta_i \partial\theta_j} \right\}, \tag{8}$$
where $E_\theta\{\ldots\}$ denotes $\int\{\ldots\}f(x|\theta)d\theta$.

- $\Gamma \in \mathbb{R}^k$ is the region of allowable θ values.

- "log" denotes the base-two logarithm. [This convention will be used throughout this paper. Natural logarithms will be represented by "ln".]

This approximation holds for $-\log \hat{P}(x^n)$ also, by simply replacing f with P. Rissanen[27] presents a rigorous proof of this result including the $o(1)$ error bound. Here we present a simpler, less rigorous but more intuitive (and therefore of pedagogical value) derivation of this result.

As mentioned, it is the denominator of (6) that is likely to be problematic and this is where the approximation is made. Thus we are concerned with the integral $\int_\Gamma f(x^n|\hat{\theta}(x^n))dx^n$. First consider a related integral: $\int_\Gamma f(x^n|\theta)dx^n$, where θ is taken to be fixed and not a function of x^n. Note that with a suitable change of variables:

$$\int f(x^n|\theta)dx^n \quad = \quad \int f_n(\hat{\theta}|\theta)d\hat{\theta}, \tag{9}$$

Where $f_n(\hat{\theta}|\theta)$ is the probability density for the ML estimate $\hat{\theta}(x^n)$ given θ.[The subscript n in f_n is there as a reminder that this function depends on n, the number of data points. It is also obviously a function of ξ, as is the dimensionality of θ.] From which we conclude that (this will be justified below):

$$\int f(x^n|\hat{\theta}(x^n))dx^n \quad = \quad \int f_n(\hat{\theta}|\hat{\theta})d\hat{\theta}, \tag{10}$$

where by $f_n(\hat{\theta}|\hat{\theta})$ we mean $f_n(\hat{\theta}|\theta)|_{\theta=\hat{\theta}}$. Next, following Rissanen, we will assume that $\hat{\theta}(x^n)$ obeys the *central-limit theorem*, meaning that:

$$f_n(\hat{\theta}|\theta) \quad \rightarrow \quad |I(\theta)|^{1/2}\left(\frac{n}{2\pi}\right)^{k/2}\exp\left[-\frac{1}{2}(\hat{\theta}-\theta)^T nI(\theta)(\hat{\theta}-\theta)\right], \quad n\rightarrow\infty, \tag{11}$$

and therefore that

$$f_n(\hat{\theta}|\hat{\theta}) = |I(\hat{\theta})|^{1/2}\left(\frac{n}{2\pi}\right)^{k/2}. \tag{12}$$

The only approximation used in this is the assumption of the central-limit form for $f(\hat{\theta}|\theta)$, thus the error in our expression is equal to:

$$\log\left[\int\frac{|I(\hat{\theta})|^{1/2}\left(\frac{n}{2\pi}\right)^{k/2}d\hat{\theta}}{\int f_n(\hat{\theta}|\hat{\theta})d\hat{\theta}}\right]. \tag{13}$$

Taking the logarithm of equation (12) gives the result of equation (7) leaving the error's asymptotic dependence on n (i.e. the $o(1)$ term) to be determined. We are only left to justify the step of (10). To do this we write the integral in question as a Riemann sum:

$$\int f(x^n|\hat{\theta}(x^n))dx^n \;=\; \lim_{\epsilon\rightarrow 0}\sum_i\int_{\delta_i} f(x^n|\theta_i)dx^n \;=\; \lim_{\epsilon\rightarrow 0}\sum_i\int_{\phi_i} f_n(\hat{\theta}|\theta_i)d\hat{\theta} \;=\; \int f_n(\hat{\theta}|\hat{\theta})d\hat{\theta}. \tag{14}$$

In this equation $\{\delta_i\}$ are the regions of integration of the Riemann sum. The corresponding $\hat{\theta}$ regions are $\{\phi_j\}$ and ϵ is their size. In going from the sum on the right-hand side of the first equals sign to the one to the right of the second, we have applied equation (9) to each term in the sum. An expanded derivation of this result is given in Appendix A.

In summary we are left with the possibility of using $L(\xi)$ plus one of (1),(6), or (7), as the quantity to be minimized in MDL-based estimation, depending on the situation.

3 MDL Estimation with Bernoulli Models

We next apply these formulas to the problem of segmenting binary strings into "regions" (substrings) that are assumed to correspond to a single Bernoulli parameter. Thus we consider binary data ($x \in \{0,1\}$ and (using the obvious equivalence between binary strings and integers) $x^n \in \{0,1,\ldots,2^n-1\}$). We will assume a Bernoulli distribution with parameter $\theta \in \mathbb{R}$, where $P(x = 1|\theta) = \theta$ and $P(x = 0|\theta) = 1 - \theta$. Also, for a single Bernoulli string x^n, $P(x^n) = \theta^{n_1}(1-\theta)^{n-n_1}$, where n_1 is the number of 1's in x^n, n_0 the number of 0's and $n_1 + n_0 = n$. The segmentation problem is to divide x^n into substrings $\{x^{(j)}\}$, each of whose bits are described by a Bernoulli parameter $\theta^{(j)}$.

We will begin our development by computing the stochastic complexity of a string assuming a single Bernoulli distribution. Next we will use this result to do hypothesis testing on a set of two binary strings of equal length, with the question being: "Do the strings come from the same or two different distributions?". Finally, we will generalize our hypothesis testing results to the segmentation problem.

3.1 The Stochastic Complexity of a Single Binary String

In this example, we are only interested in computing the complexity of the data strings themselves; there is no issue of estimating the model complexity (We have only a single scalar parameter θ.) and therefore $L(\xi) = 0$.

3.1.1 Two-Part Encoding

Before obtaining expressions for the ML code, we will derive an expression for the codelength based on a two-part code. Given a value for θ, a binary string x^n with n_1 1's can be encoded with:

$$-\log\left[\theta^{n_1}(1-\theta)^{n-n_1}\right] \;=\; -n_1\log\theta \;-\; (n-n_1)\log(1-\theta) \tag{15}$$

bits. This is minimized by choosing $\theta = \hat{\theta} = n_1/n$. For an n-bit string there are thus $n+1$ possible $\hat{\theta}$ values, which are the rational numbers: $\{i/n; i = 0,1,2,\ldots,n\}$. Thus encoding $\hat{\theta}$ corresponds to encoding n_1. Using $\theta = \hat{\theta}$ in equation (15) and adding $\log(n+1)$ bits for encoding $\hat{\theta}$, the two-part codelength is:

$$L(x^n) \;=\; -n_1\log n_1 \;-\; (n-n_1)\log(n-n_1) \;+\; n\log n \;+\; \log(n+1). \tag{16}$$

We should mention that this differs from the classic MDL formula[24] in that in the classic formula our $\log(n+1)$ term would be replaced by $\frac{1}{2}\log n$. The classic formula is an asymptotic form derived for the general case (any form for P or f) and is based on an analysis of the optimal (in the sense of minimum codelength) precision to which $\hat{\theta}$ should be encoded (quantized). In this case there is a natural precision for encoding θ determined by the string length, n.

3.1.2 Using the Maximum-Likelihood Code

Next, we derive codelength expressions (exact and approximate) for the ML code defined above.

3.1.2.1 Exact Expression

Using equation (1) we may write an exact expression for the ML codelength of a binary string assuming Bernoulli models.

$$L(x^n) \;=\; -\log[\hat{\theta}_{n_1}^{n_1}(1-\hat{\theta}_{n_1})^{n-n_1}] \;+\; \log\left[\sum_{\ell=0}^{n}\binom{n}{\ell}\hat{\theta}_\ell^{\ell}(1-\hat{\theta}_\ell)^{(n-\ell)}\right], \tag{17}$$

where the $(L(d)=\log n)$ term has been dropped. Substituting $\hat{\theta}_\ell = (\ell/n)$ gives:

$$L(x^n) \;=\; -\log\left[\left(\frac{n_1}{n}\right)^{n_1}\left(\frac{n-n_1}{n}\right)^{n-n_1}\right] \;+\; \log\left[\sum_{\ell=0}^{n}\binom{n}{\ell}\left(\frac{\ell}{n}\right)^{\ell}\left(\frac{n-\ell}{n}\right)^{n-\ell}\right], \tag{18}$$

Simplifying the first term gives:

$$L(x^n) \;=\; -n_1\log n_1 - (n-n_1)\log(n-n_1) + n\log n \;+\; \log\left[\sum_{\ell=0}^{n}\binom{n}{\ell}\left(\frac{\ell}{n}\right)^{\ell}\left(\frac{n-\ell}{n}\right)^{n-\ell}\right], \tag{19}$$

The probability terms $\left[\left(\frac{n_1}{n}\right)^{n_1}\left(\frac{n-n_1}{n}\right)^{n-n_1}\right]$ for $n_1, \ell = 0, n$ are equal to 1 ($\lim_{x\to 0} x^x \;=\; 1$) and their logarithms are therefore zero. The sum on the RHS of (19) may be simplified to

$$n^{-n}n!\sum_{\ell=0}^{n}\left[\frac{\ell^{\ell}\,(n-\ell)^{n-\ell}}{\ell!(n-\ell)!}\right]. \tag{20}$$

We simplify the notation by defining $q(n)$ giving

$$n^{-n}n!q(n) \tag{21}$$

for this sum. Using this, (19) becomes:

$$L(x^n) \;=\; -n_1\log n_1 - (n-n_1)\log(n-n_1) \;+\; \log n! \;+\; \log q(n) \tag{22}$$

3.1.2.2 Approximate Expression

In this case (binary data and Bernoulli models) we were able to derive an exact expression for the ML code length and have no practical need for the approximate form. Our reason for attacking this problem is, however, pedagogical, after all, and it will be instructive to compare the approximate and exact forms. Here we will compute the stochastic complexity using the approximation derived in [27]:

$$-\log\hat{P}(x^n) \;=\; -\log P(x^n|\hat{\theta}(x^n)) \;+\; \frac{k}{2}\log\frac{n}{2\pi} \;+\; \log\int_\Gamma\sqrt{|I(\theta)|}d\theta \;+\; o(1), \tag{23}$$

In this expression we will substitute

$$-\ln P(x^n|\hat{\theta}(x^n)) \;=\; -n_1\log n_1 \;-\; (n-n_1)\log(n-n_1) + n\log n \tag{24}$$

$$\tag{25}$$

and an expression for $\log\int_\Gamma\sqrt{|I(\theta)|}d\theta$.

$$P(x|\theta),\ x \in \{0,1\} \tag{26}$$

$$P(0|\theta) \ = \ \theta, \ \ P(1|\theta) \ = \ 1-\theta \tag{27}$$

$$\ln P(0|\theta) \ = \ \ln\theta, \ \ \ln P(1|\theta) \ = \ \ln(1-\theta) \tag{28}$$

$$\tfrac{d\ln P(0|\theta)}{d\theta} \ = \ \tfrac{1}{\theta}, \ \ \tfrac{d\ln P(1|\theta)}{d\theta} \ = \ \tfrac{-1}{(1-\theta)} \tag{29}$$

$$\tfrac{d^2\ln P(0|\theta)}{d\theta^2} \ = \ \tfrac{-1}{\theta^2}, \ \ \tfrac{d^2\ln P(1|\theta)}{d\theta^2} \ = \ \tfrac{-1}{(1-\theta)^2} \tag{30}$$

$$I(\theta) \ = \ -E_\theta\left\{\frac{d^2\ln P(x|\theta)}{d\theta^2}\right\} \ = \ -\theta\frac{-1}{\theta^2} \ - \ (1-\theta)\frac{-1}{(1-\theta)^2} \ = \ \frac{1}{\theta(1-\theta)} \tag{31}$$

$$\int_\Gamma \sqrt{|I(\theta)|}d\theta \ = \ \int_0^1 \theta^{-1/2}(1-\theta)^{-1/2}d\theta \ = \ \pi \tag{32}$$

In this case θ is scalar (thus $k=1$) and making the appropriate substitutions into (23) yields:

$$-\log\hat{P}(x^n) \ = \ -n_1\log n_1 \ - \ (n-n_1)\log(n-n_1) + n\log n \ + \ \frac{1}{2}\log n \ + \ \log\sqrt{\pi/2} \ + \ o(1), \tag{33}$$

where $\log\sqrt{\pi/2} \approx 0.33$ and can just be lumped in with "$o(1)$". It is interesting to note that neglecting $\log\sqrt{\pi/2}+o(1)$, in this yields exactly the same expression one obtains using a two-part code with the old asymptotic expression derived in [24] for the parameter cost.

Notice that (neglecting "$o(1)$") both this approximate form (equation (33)) and the exact form (equation (22)), as well as the two-part codelength (equation (15) are of the form:

$$L(n_1,n) \ = \ -n_1\log n_1 \ - \ (n-n_1)\log(n-n_1)+g(n). \tag{34}$$

Thus the difference among these three forms is characterized by only the difference in the forms for $g(n)$. This comparison can be found in the graphs of Figure 1.

3.1.3 Encoding Based on a Bayesian Mixture with a Uniform Prior

In cases where there is a *prior* distribution on θ, $\pi(\theta)$, it may be appropriate to use an encoding scheme based on a *Bayesian mixture*. For the case considered here this gives:

$$L(x^n) = -\log\int_0^1 \theta^{n_1}(1-\theta)^{n-n_1}\pi(\theta)d\theta, \tag{35}$$

which is just the log of the marginal distribution: $P(x^n) = \int P(x^n,\theta)d\theta$.

In the following we will assume that the prior, $\pi(\theta)$ is uniform over the interval $[0,1]$. In this case equation (35) becomes:

$$L(x^n) \ = \ -\log\int_0^1 \theta^{n_1}(1-\theta)^{n-n_1}d\theta \tag{36}$$

Using a result found in most integral tables,

$$\int_0^1 \theta^{n_1}(1-\theta)^{n-n_1}d\theta \ = \ B(n_1+1,n-n_1+1) \ = \ \frac{\Gamma(n_1+1)\Gamma(n-n_1+1)}{\Gamma(n+2)} \ = \ \frac{n_1!(n-n_1)!}{(n+1)!}, \tag{37}$$

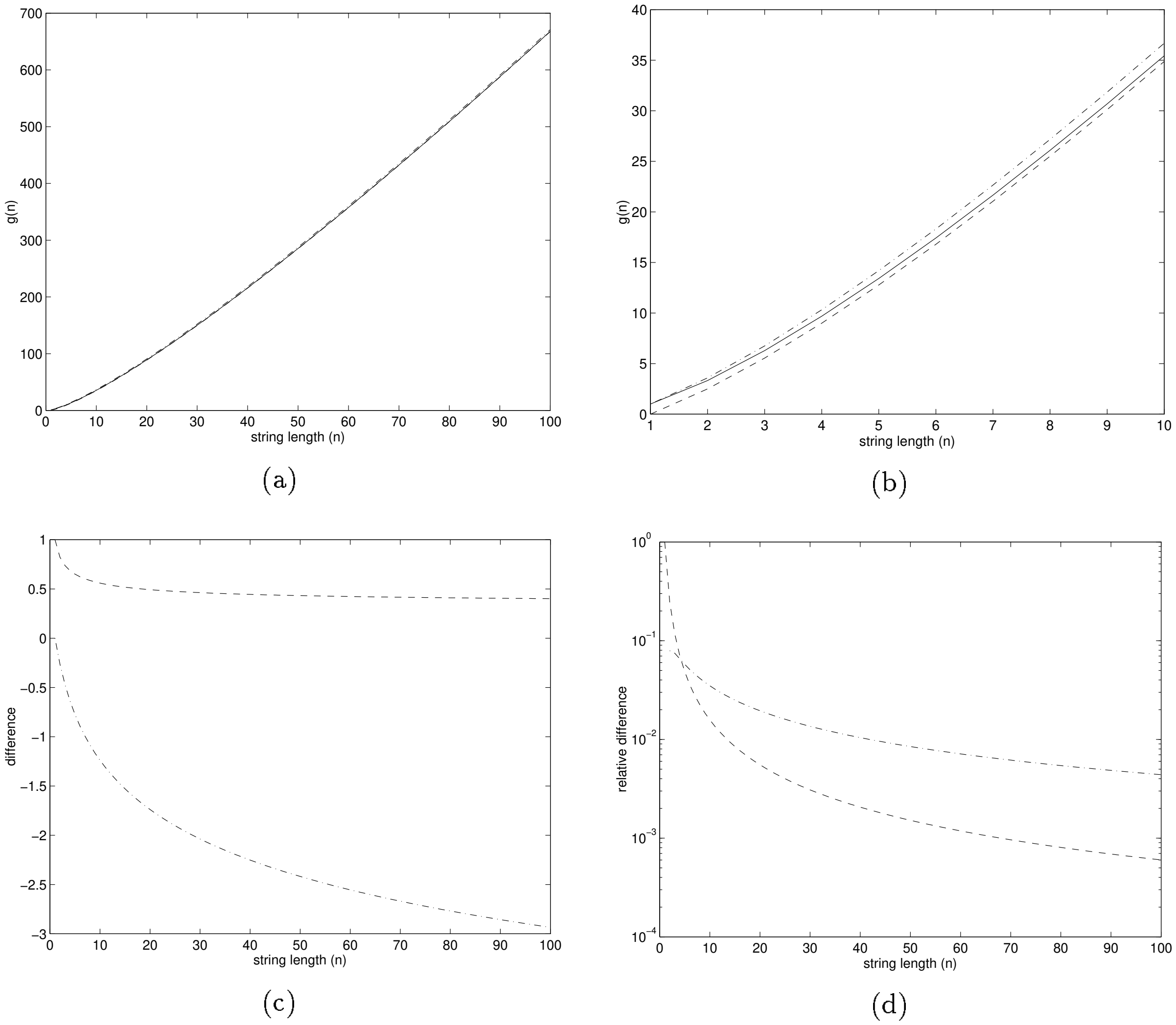


Figure 1: $g(n)$ terms from Bernoulli/ML-code-based stochastic complexity. These are terms that only depend on the string length, n and not on the number of ones. The solid curves are the exact form; the dashed ("$--$") curves are for the approximate form; and the dash-dot curves ("$-\cdot$") represent the two-part codelength of equation (16). (a) shows the three curves over a broad range of n while (b) shows their behavior for small n. The difference (c) and relative difference (d) ($\Delta g/g$) are also shown.

where $B(x,y)$ is the "beta" function and $\Gamma(x)$ is the "gamma" function. Thus we have:

$$L(x^n) \quad = \quad \log \frac{(n+1)!}{n_1!(n-n_1)!}. \quad = \quad \log(n+1) \; + \; \log \binom{n}{n_1} \tag{38}$$

A simple, intuitive way to derive this last result, however, is to notice that a uniform prior over θ is equivalent to assuming that all n_1 are equally likely. This implies a scheme where we first encode n_1 using $\log(n+1)$ bits and then encode the specific string using $\log \binom{n}{n_1}$ bits. Note also, that this is exactly the result obtained from performing a *redundancy removal* on the two-part code. That is, in the two-part code we're using an encoding of the form:

$$L(x^n) \quad = \quad -\log P(x^n|\hat{\theta}(x^n)) \; + \; L(\hat{\theta}(x^n)). \tag{39}$$

This code is *redundant* because it allows for situations that can never occur. Once the decoder has decoded $\hat{\theta}$, it "knows" that x^n must belong to the set of values which would produce the ML parameter estimate of $\hat{\theta}$. This redundancy is removed by simply re-normalizing the probability used in the data encoding term resulting in:

$$L(x^n) \quad = \quad -\log \frac{P(x^n|\hat{\theta}(x^n))}{\sum_{\{u^n|\hat{\theta}(u^n)=\hat{\theta}(x^n)\}} P(u^n|\hat{\theta}(x^n))} \; + \; L(\hat{\theta}(x^n)). \tag{40}$$

In our case the sum in the denominator is equal to:

$$\binom{n}{n_1} \hat{\theta}^{n_1}(1-\hat{\theta})^{n-n_1}, \tag{41}$$

whereas the numerator equals $\hat{\theta}^{n_1}(1-\hat{\theta})^{n-n_1}$, which cancels with the one in the numerator leaving $\log \binom{n}{n_1}$.

3.1.4 Complexity Summary

Summarizing, the forms we have for the complexity of an n-bit binary string with n_1 ones using Bernoulli models are:

1. The two-part code:

$$L_{\mathrm{tp}}(x^n) \;=\; -n_1 \log n_1 \;-\; (n-n_1)\log(n-n_1) \;+\; n\log n \;+\; \log(n+1)$$

2. The maximum-likelihood code (exact expression):

$$L_{\mathrm{mle}}(x^n) \;=\; -n_1 \log n_1 - (n-n_1)\log(n-n_1) \;+\; \log n! \;+\; \log q(n)$$

3. The maximum-likelihood code (approximate expression):

$$L_{\mathrm{mla}}(x^n) \;=\; -n_1 \log n_1 \;-\; (n-n_1)\log(n-n_1) + n\log n \;+\; \frac{1}{2}\log n$$

4. Bayesian mixture with a uniform prior:

$$L_{\mathrm{bm}}(x^n) \;=\; \log(n+1) \;+\; \log\binom{n}{n_1} \;=\; \log\frac{(n+1)!}{n_1!(n-n_1)!}$$

3.1.5 Hypothesis Testing

In this problem we are given two binary strings of length $n/2$, where n is an even integer. Denote these by ζ_1, ζ_2 and let k_1 and k_2 represent the number of ones in these strings. We want to test the hypothesis that these came from two different distributions (with parameters θ_1 and θ_2) against the hypothesis that they are from the same distribution (θ_0). Denote these hypotheses by H_2 and H_1 respectively. Classical and Bayesian hypothesis testing theory are reviewed briefly in Appendix B. In classical hypothesis testing theory one must "design a test", which involves a procedure to which is input two of three parameters: the two error rates and the number of samples. This procedure then outputs a threshold and the third, parameter (the unspecified one). We will not consider this type of procedure here because we are interested in developing formulas and algorithms keeping the number of adjustable parameters to a minimum. In the Bayesian theory, if we wish to minimize the probability of error, we choose the hypothesis ($h \in \{H_1, H_2\}$) that maximizes $P(h|\zeta_1, \zeta_2)$ [this is the MAP estimate.] If we further believe that both hypotheses are equally likely this is equivalent to maximizing $P(\zeta_1, \zeta_2|h)$. To be of further utility the hypotheses must also specify either definite values for the associated θ's or they must specify priors - $\pi(\theta)$. In MDL-based hypothesis testing we choose the model that provides the smallest total description for the data. Said another way, we choose the model that provides the minimum-complexity description. That is:

$$L(\zeta_1\|\zeta_2) \ < \ L(\zeta_1) + L(\zeta_2) \ \Rightarrow \ H_1 \tag{42}$$

$$L(\zeta_1\|\zeta_2) \ \geq \ L(\zeta_1) + L(\zeta_2) \ \Rightarrow \ H_2 \tag{43}$$

$$\tag{44}$$

Where $L(\ldots)$ is one of the codelength expressions derived in the previous section and "$\|$" denotes the concatenation of two strings. In this case $L(\xi)$ will be neglected because it will be taken to be the same (1 bit) for both cases H_1 and H_2.

As discussed in Appendix B, a hypothesis test may be characterized by the *decision regions*, V_1, V_2, (where V_1 is the set of all strings (ζ_1, ζ_2) that will cause us to choose H_1, etc.), the error rates $\epsilon_{1|2}, \epsilon_{2|1}$ (defined in Appendix B) and

$$\epsilon_{\text{total}} \ = \ \epsilon_{1|2}P(H_2) \ + \ \epsilon_{2|1}P(H_1). \tag{45}$$

Figures 2 and 3 present some computed error probabilities for specific one- and two-string cases. Figure 2 shows results for Bernoulli distributions with the associated θ values mentioned in the caption, whereas Figure 3 presents results for Bayesian mixtures (with uniform priors) of Bernoulli distributions. We make the following observations regarding these results.

- The non-monotonic behavior of these plots with increasing n may, at first, seem unexpected, but, as is demonstrated in Appendix C, this is the natural result of the structure of the test. Note that in Figure 3, for example, the total error probabilities, while being a little bumpy for small n, all decay monotonically with n.

- In the same figure, not surprisingly, the Bayesian-mixture encoding scheme produces the best hypothesis testing results, because it agrees exactly with the underlying distributions assumed in computing these.

- Also, note that the two-part code favors the one-distribution hypothesis relative to the others. This is consistent with the fact that the data encoding is really longer than necessary due to the redundancy inherent in this scheme [i.e. The associated "code book" must contain codes that can never occur. See [25] and [27] for discussions of this redundancy issue in MDL estimation.]. The two-part code also yields the worst total error probabilities, which is also consistent with this redundancy.

- The ML-code-based formulas (exact and approximate) do quite well, with the approximate form giving slightly worse overall results, but with the difference going to zero with increasing n. These effects are also consistent with expectations.

It is not our interest or intent to obtain a complete characterization of these various criteria for the problem of hypothesis testing, but rather to gain some qualitative understanding of the behavior of the solutions they produce and to obtain some reassurance that these solutions are consistent with our goal of segmenting binary strings.

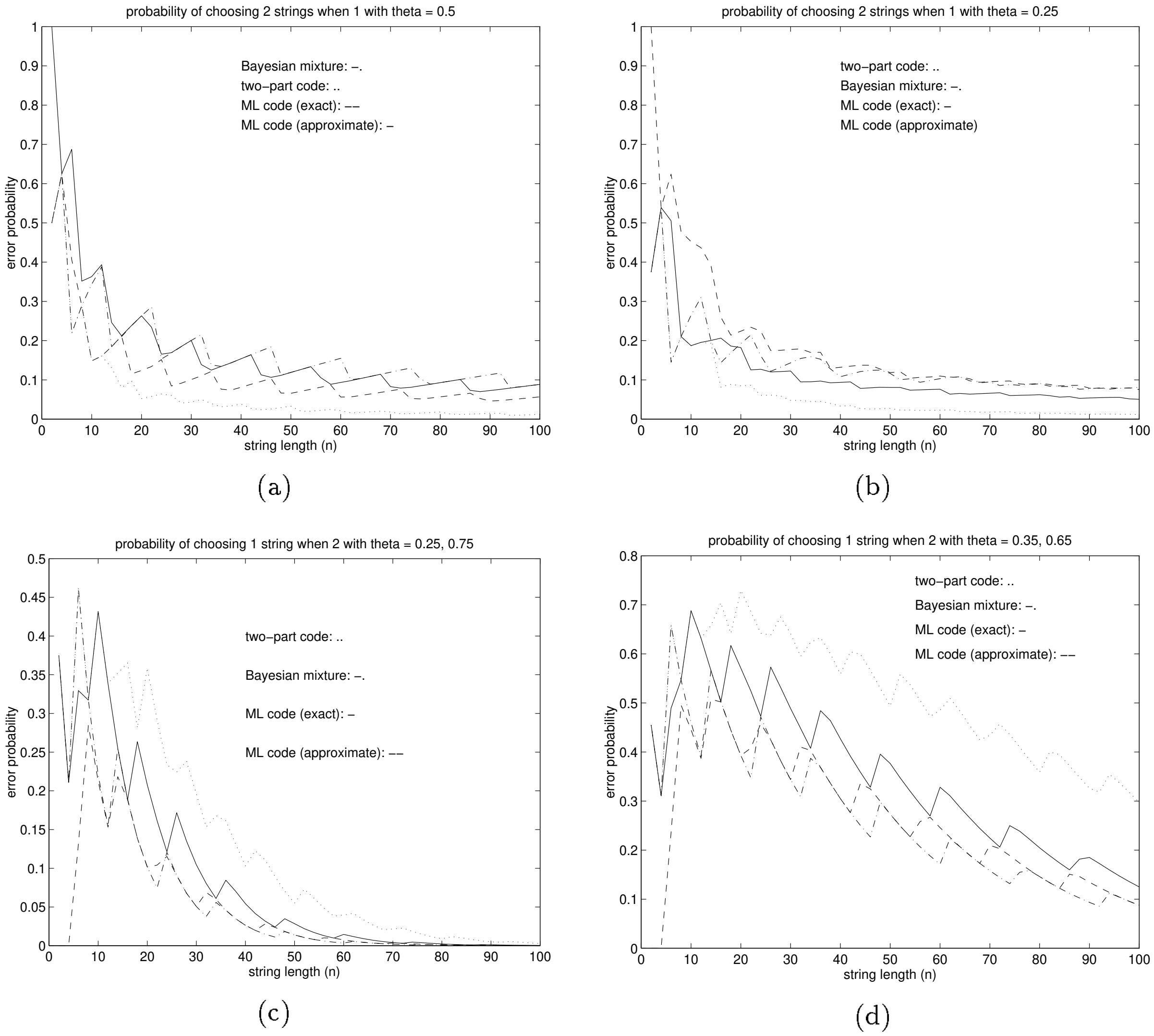


Figure 2: Sample computed error probabilities for Bernoulli hypothesis testing. (a) shows probability of choosing two strings when there is only a single string with $\theta = 0.5$; (b) same thing for $\theta = 0.25$; (c) probability of choosing one string when there are two with $\theta_1 = 0.25$ and $\theta_2 = 0.75$; (b) same thing for $\theta_1 = 0.35$ and $\theta_2 = 0.65$.

3.2 Segmentation of Binary Strings Using Bernoulli Models

Our investigation of hypothesis testing results is useful as a sanity check before moving on to a more complicated generalization of this problem: segmenting binary strings into substrings, each of which is modeled with its own Bernoulli parameter. Our approach will be to first design an encoding scheme based on this *piecewise-Bernoulli* model class; and then try all possible partitionings of the string, evaluating the codelength for the string in each case. We will then choose as best that

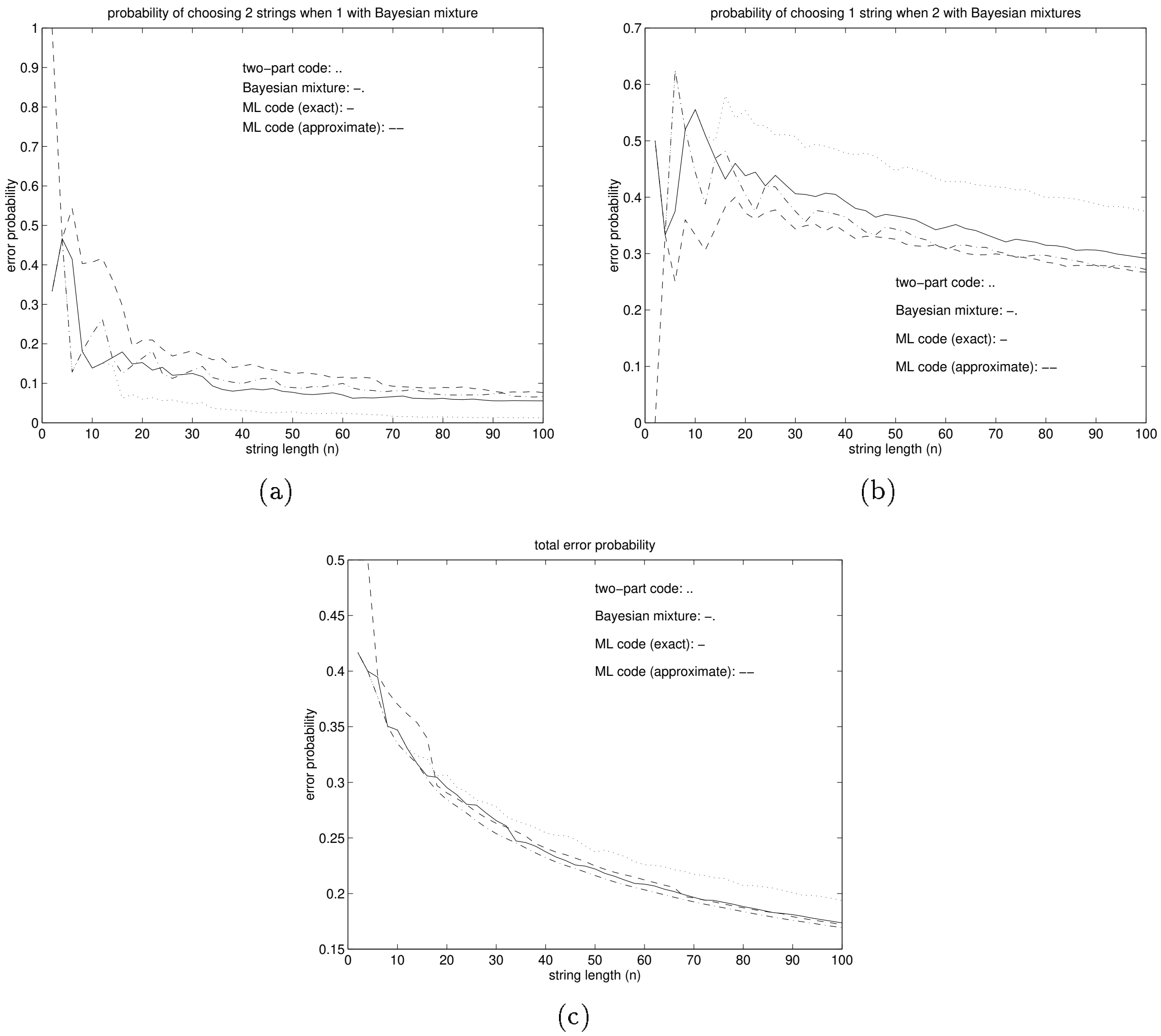


Figure 3: Sample computed error probabilities for Bernoulli hypothesis testing. **(a)** shows probability of choosing two strings when there is only a single string from a distribution which is a Bayesian mixture with a uniform prior. **(b)** probability of choosing 1 when there are 2. **(c)** the total error probability assuming the two cases are equally probable (i.e. 1/2 the sum of (a) and (b).

partitioning yielding the lowest codelength. In this case the partitioning constitutes the model structure ξ and $\theta^{(\xi)}$ is composed of the Bernoulli parameters corresponding to the substrings of this partitioning.

Having chosen a partition of our original length-n string into substrings, each substring will be encoded using one of the encoding schemes derived above.[Encoding schemes will not be mixed within a single string or within a single segmentation.] The total encoding length will then be the sum of the individual substring codelengths plus the length of the description of the partitioning. This warrants a digression returning to our stochastic complexity definition (equations (2) and (1)). If our model (ξ) divides x^n into independent non-overlapping *equivalence classes* (e.g. the individual substrings in our partitioning of a binary string), the ML probability (equation (1)) factors into a product of individual ML probabilities. Let $x^n = \bigcup_j x^{(j)}$, where $\{x^{(j)}\}$ denotes the set of data vectors (*bits* in our binary-string-segmentation example) associated with the j^{th} equivalence class (e.g. substring). Using this we may write:

$$\hat{P}(x^n|\xi) \quad = \quad \prod_j \frac{P(x^{(j)}|\hat{\theta}(x^{(j)})}{\sum_{u^{(j)}} P(u^{(j)}|\hat{\theta}(u^{(j)})}, \tag{46}$$

and, of course:

$$\log \hat{P}(x^n|\xi) \quad = \quad \sum_j \log \frac{P(x^{(j)}|\hat{\theta}(x^{(j)})}{\sum_{u^{(j)}} P(u^{(j)}|\hat{\theta}(u^{(j)})}. \tag{47}$$

There are 2^{n-1} such partitionings. One possible encoding scheme is to use $n-1$ bits, thus making the cost the same for all regardless of the number of substrings. This violates the feeling that partitionings with more substrings are of higher complexity. One way to address this is to first specify the number of substrings and then the actual partitioning conditioned on this number. The number of ways of partitioning an n-bit string into k non-empty substrings is $\binom{n-1}{k-1}$. Using this, one way to encode the partition of the string is to first encode the number of substrings k using $\log n$ bits and then code the particular k-substring partition using $\log \binom{n-1}{k-1}$ bits giving:

$$L(\xi) \quad = \quad \log n \; + \; \log \binom{n-1}{k-1}. \tag{48}$$

This encoding penalizes partitionings with greater numbers of substrings up to $n/2$. Beyond that the codelength actually decreases with number of substrings. We will use this encoding and will be concerned only if our final result has more than $n/2$ substrings, which we don't expect in the cases we will investigate.

Two typical results for 100-bit strings are shown in Figure 4. and another, interesting result is shown in Figure 5, in which the string was generated using a θ that varied linearly along the length of the string.

A large number of such strings were segmented and the results analyzed. Strings were generated with a particular instance of the assumed piecewise-Bernoulli model. Twenty strings were generated for each model instance. A partially reduced summary of such results for a particular case is shown in Figures 6 and 7. The first 50 bits of these were generated with a probability ($\theta = P(1)$) of 0.8 and the second 50 with a probability of 0.2. Figure 6 shows histograms of the number of substrings chosen with the correct answer in this case obviously being 2. Figure 7, on the other hand, shows two sample boundary-distance histograms for the same 20 strings. These are a Hausdorff-like measures. In one the distance from every "ground-truth" boundary point (there is only one in this case) to the closest segmentation boundary point is tabulated. In the other, the same thing is tabulated but for the distance from every segmentation boundary point to the closest ground-truth point. The 2-D versions of such histograms were used to characterize image segmentations in [8], where they were referred to as "distance distribution signatures".

The data from these experiments are further summarized in Table 1, which includes statistics taken from Hausdorff histograms like those shown in Figure 7. For all the numbers shown, the larger the number, the larger the error indication. Based on that, for this experiment, the Bayesian mixture appears to have produced the best results, with the two versions of the ML code being marginally worse and the two-part code worse by a somewhat more significant amount, especially

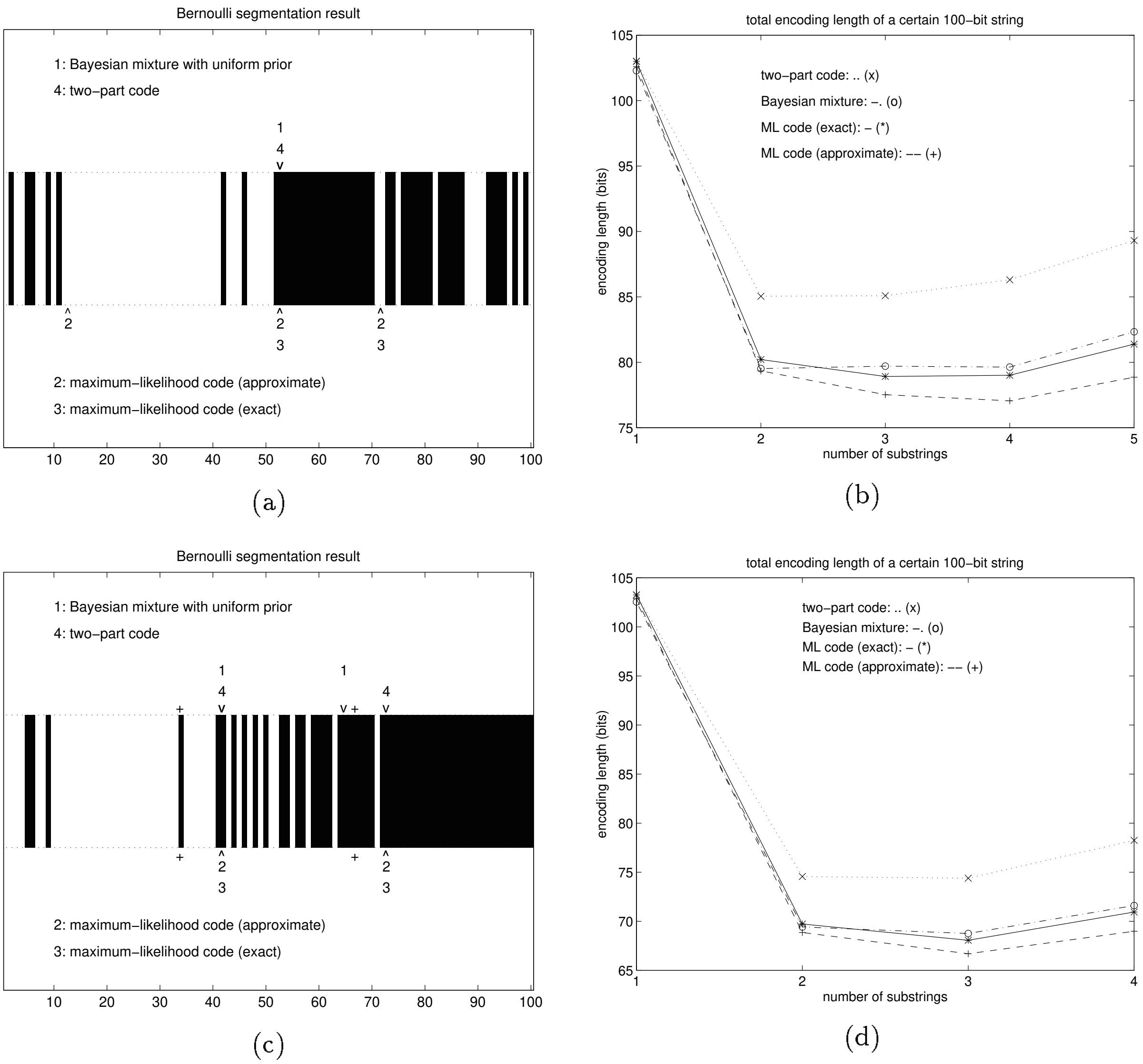


Figure 4: Result of segmenting two binary strings using MDL with the Bernoulli-based encoding schemes described here. The strings are random with (in the first case; (a) and (b)) the first 50 bits generated with a probability of 0.8 and the second 50 with a probability of 0.2, and (in the second case; (c) and (d)) the first 33 bits generated with a probability of 0.92; the second 33 with 0.5 and the last 34 with 0.08. (a) and (c) show the string in bar-code form (black = 0; white = 1). The symbols "$\vee$" and "$\wedge$" indicate the positions of the detected boundaries, with the numerals indicating the associated coding scheme. In (c) the "+" symbols mark the positions of the "real" boundaries. (b) and (d) are graphs of the code lengths obtained with the four schemes as a function of the number of substrings. In each case the value corresponds to the partitioning with the shortest codelength.

as the difference between the two θ parameters (for the first and last 50 bits) decreases. As pointed out previously, two-part encoding favors fewer substrings relative to the others and all the errors it made on the $\theta = 0.3, 0.7$ were choosing a single string rather than two substrings.

A second group of strings generated, segmented and analyzed consisted of 2 sets of 20 strings with each string consisting of 3 substrings. These results are summarized in Tables 2 and 3. The

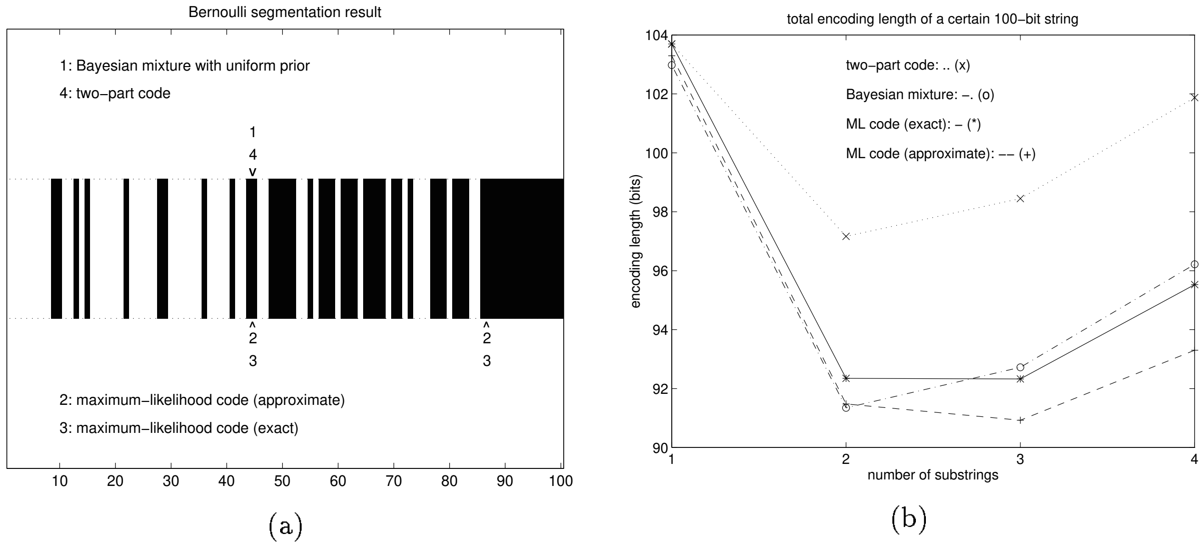


Figure 5: Result of segmenting a binary string using MDL with the Bernoulli-based encoding schemes described here. The string was randomly generated using a "ramp" for θ, varying linearly from 1 to 0 over the length of the string.

Table 1: Summary of Bernoulli Segmentation Error Statistics. Each line of this table corresponds to the analysis of 20 100-bit strings. The strings were Bernoulli randomly generated with the first 50 bits generated with the θ (probability of a bit being 1) values shown and the second 50 being generated with $P(1) = 1 - \theta$. Here n represents the number of strings (out of 20) for which the chosen number of substrings was incorrect. The *Hausdorff* statistics h_1, h_2 and h_T represent the sum of the ground-truth-to-segmentation, segmentation-to-ground-truth and total boundary distance respectively. The superscripts on the various quantities indicate the corresponding encoding scheme: 1: Bayesian mixture; 2: approximate ML; 3: exact ML; and 4: two-part.

θ	n^1	h_1^1	h_2^1	h_T^1	n^2	h_1^2	h_2^2	h_T^2	n^3	h_1^3	h_2^3	h_T^3	n^4	h_1^4	h_2^4	h_T^4
0.3	3	268	118	386	3	268	118	386	4	301	101	402	15	765	15	780
0.2	1	27	42	69	3	25	95	120	2	25	73	98	0	27	27	54
0.1	0	11	11	22	2	11	13	24	2	11	13	24	2	11	13	24
Total	4	306	171	477	8	304	226	530	8	337	187	524	17	803	55	858

Hausdorff statistics presented in Table 3 are the following.

1. n: number of strings for which the wrong number of sub-strings was chosen.

2. thd1: total min. dist. from ground truth

3. thd2: total min. dist. from segmentation

4. ahd1: average min. dist. from ground truth

5. ahd2: average min. dist. from segmentation

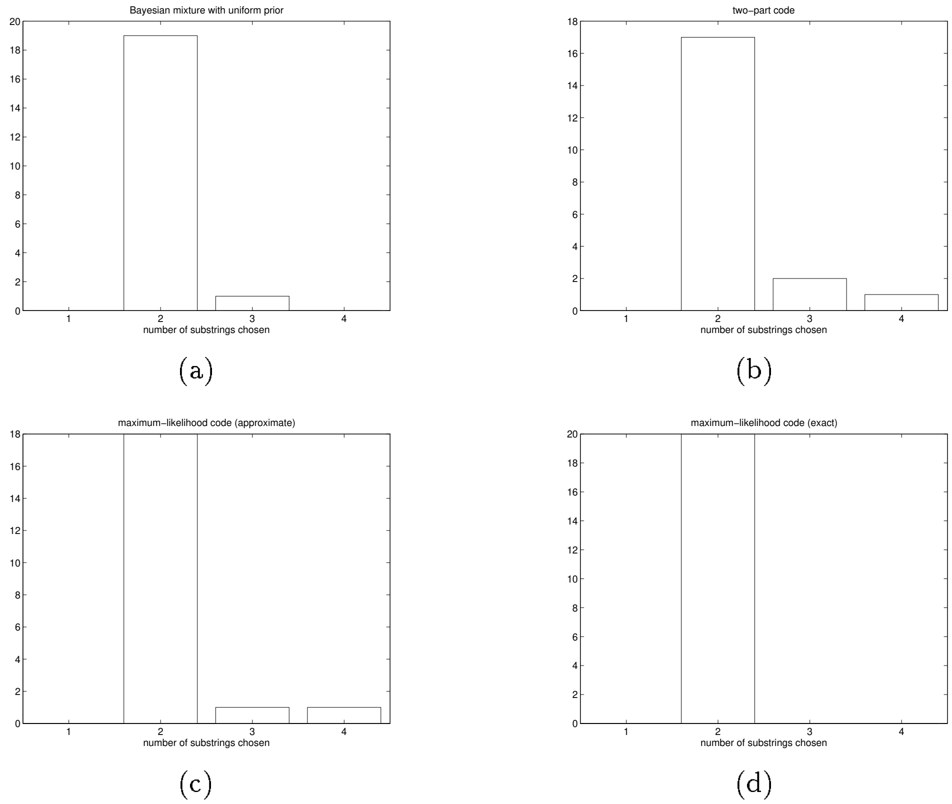


Figure 6: Histograms of the number of substrings chosen in segmenting binary strings using Bernoulli-based encoding models. This one is for 20 100-bit strings. The strings were randomly generated with the first 50 bits generated with $P(1) = 0.2$ and the second 50 being generated with $P(1) = 0.8$. (1) Bayesian mixture; (b) approximate ML code; (c) exact ML code; (d) two-part code.

6. ahd: 0.5*(ahd1 + ahd2)

7. ahdw: average minimum distance of combined histograms.

It is interesting to note that all the coding techniques favor two substrings over three. This is certainly not inconsistent with our Bernoulli hypothesis-testing results, which show a bias toward smaller numbers of substrings at small string lengths.

4 Conclusions, Possible Future Investigations and Extensions

4.1 Conclusions

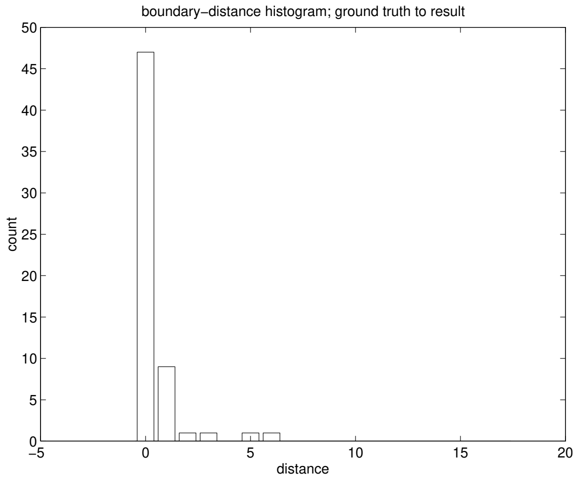
Because the experimental and theoretical investigations presented here are limited, care should be taken in drawing conclusions from them. Observations about the results are given in previous sections and won't be repeated here. It seems reasonable to conclude that the new ML-code-based scheme appears useful for the problems investigated and appears to work better than the two-part scheme as we hoped and expected that it would. It did slightly worse than the Bayesian mixture in the experiments here, but the data is too limited to be certain that the mixture is better. It

Table 2: Summary of Bernoulli Segmentation Error Statistics from the analysis of 2 sets of 20 100-bit strings. The strings were Bernoulli randomly generated with the first 33 bits having $\theta = 0.9$, the second 33 having $\theta = 0.5$ and the last 34 having $\theta = 0.1$. This table shows the number of trials out of 20 for which the indicated number of substrings was chosen, with 3 being the "correct" answer.

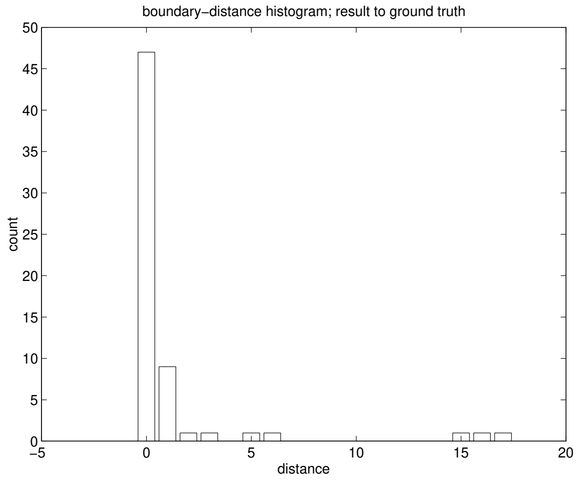
coding scheme	number of substrings			
	1	2	3	4
1	0	11	8	1
2	0	8	11	1
3	0	10	9	1
4	0	17	3	0
1	0	11	9	0
2	0	10	10	0
3	0	11	9	0
4	0	12	8	0

Table 3: Summary of Bernoulli Segmentation Error Statistics from the analysis of 2 sets of 20 100-bit strings. The strings were Bernoulli randomly generated with the first 33 bits having $\theta = 0.9$, the second 33 having $\theta = 0.5$ and the last 34 having $\theta = 0.1$. Here n represents the number of strings (out of 20) for which the chosen number of substrings was incorrect. The other numbers are *Hausdorf* statistics (i.e. "thd1", etc.) and are defined in the text. The "coding scheme: numbers correspond to: 1: Bayesian mixture; 2: approximate ML; 3: exact ML; and 4: two-part.

coding scheme	n	thd1	thd2	ahd1	ahd2	ahd	ahdw
1	11	411	90	5.1375	1.2857	3.2116	3.3400
2	8	332	101	4.1500	1.3836	2.7668	2.8301
3	10	387	98	4.8375	1.3803	3.1089	3.2119
4	17	597	91	7.4625	1.4444	4.4535	4.8112
1	11	378	71	4.7250	1.0290	2.8770	3.0134
2	10	345	71	4.3125	1.0143	2.6634	2.7733
3	11	378	71	4.7250	1.0290	2.8770	3.0134
4	12	411	71	5.1375	1.0441	3.0908	3.2568



(a)



(b)

Figure 7: This figure shows one of the "Hausdorff" histograms analyzed in this work. This one is for 20 100-bit strings. The strings were Bernoulli randomly generated with the first 50 bits generated with $P(1) = 0.2$ and the second 50 being generated with $P(1) = 0.8$. The encoding scheme used to perform the segmentation was the *exact* ML code. (a) is the ground-truth-to-segmentation histogram whereas (b) is for segmentation-to-ground-truth. In computing these the endpoints of the strings were included in both boundary-point sets.

may be that they produce roughly equivalent results so that the choice of which one to use can be made based on tractability considerations.

Some may find the ML-code more satisfying philosophically because it doesn't rely on the *prior*, which some feel to be an unnecessary artificial device. We won't comment on that except to speculate that in most situations the pragmatic considerations of computational cost and accuracy will dominate. In this case at least, we've shown that the Bayesian mixture with a uniform prior is equivalent to two-part coding with redundancy removal, which requires no such device.

To our knowledge piecewise-Bernoulli binary data are of little practical importance. For this reason, our investigations of them will stop at this point, only to continue if it seems, at some future date, that further theoretical understanding might be gained or we become aware of some important application. In closing, in the sequel, we will mention possible extensions of the current work on binary data, as well as some applications of the new MDL formulas to problems with more practical importance.

4.2 More on Binary Data and Bernoulli Models

4.2.1 Optimal-Precision Analysis on Bernoulli Parameters

For the two-part scheme we used here the Bernoulli parameters were encoded to the natural precision of the problem. That is $\hat{\theta}$ was encoded with $\log(n+1)$ bits. Any greater precision would be useless because the ML estimate for θ will always be one of the rational numbers: $\{n_1/n\}$. That is the decoder knows the string length and therefore knows what the possible n_1 (i.e. $\hat{\theta}$) values are. Thus there can clearly be no gain in a precision greater (finer) than one part in $n+1$. The remaining question is: Can any reduction come from using fewer bits to encode $\hat{\theta}$? For example, suppose we group the n_1 values in groups of d so that the decoder will know that the n_1 value is one of d after it decodes $\tilde{\theta}$. In a redundancy corrected code this will mean that:

$$L(x^n) = \log \left[\sum_{i=0}^{d-1} \binom{n}{k(\tilde{\theta}) + i} \right] + \log \left[\frac{n+1}{d} \right], \tag{49}$$

where $k(\tilde{\theta})$ is the first n_1 value in the group indicated by $\tilde{\theta}$. Thus the decrease in the θ code-length has resulted in an increase in the x^n code-length. There is another factor also. Using a variable-precision scheme means that we must encode the precision d, which will take something like $\log(n/2)$ bits. The θ code-length has gone from $\log(n+1)$ to $\log((n+1)/d)$ thus decreasing by $\log d$ bits.

As mentioned in the body of the paper, Rissanen's asymptotic formula from a general optimal-precision analysis[24] gives a code length of:

$$L(x^n) = n_1 \log \hat{\theta} + (n - n_1) \log(1 - \hat{\theta}) + \frac{1}{2} \log n \tag{50}$$

$$= -n_1 \log n_1 - (n - n_1) \ln(n - n_1) + n \ln n + \frac{1}{2} \ln n. \tag{51}$$

4.2.2 Further Theoretical Analysis

Further theoretical might certainly be possible. For example, it might be possible to derive closed-form expressions for the various errors that were computed here theoretically.

4.2.3 More Extensive Experiments on 1-D Strings

Trying to really understand the behavior of the various coding schemes in trying to segment binary strings would obviously benefit from more extensive experimental characterization. Segmentation performance as a function of string length for example. It would be interesting to examine the behavior at string lengths must longer than 100. To do this would require intelligent ways of computing the various code-length tables involved to avoid numerical problems for example. Also, some suboptimal search scheme would be required. As it is, when the number of segments exceeds 5, the current exhaustive-search algorithm requires too much time (i.e. days). Of course this could be alleviated somewhat by implementing the search algorithm in C, but even that will only help a little.

4.2.4 Priors

It might be interesting to look at the segmentation error statistics when the θ's are selected randomly based on a prior distribution. It would be desirable to investigate priors that yield a closed-form expression for the marginal distribution $P(x^n)$, for example priors of the form $\pi(\theta) = (r+1)\theta^r$ or $(r+1)(1-\theta)^r$. To do this we might use the following result found in most integral tables:

$$\int_0^1 \theta^m (1-\theta)^n d\theta = B(m+1, n+1) = \frac{\Gamma(m+1)\Gamma(n+1)}{\Gamma(m+n+2)} = \frac{m!n!}{(m+n)!}, \tag{52}$$

where $B(x, y)$ is the "beta" function and $\Gamma(x)$ is the "gamma" function.

4.3 Alternate scheme for encoding the partitioning of an n-bit binary string

In this scheme the code length increases linearly with the number of partitions. If we attempt to choose $L(\xi) = \alpha k$, where k is one minus the number of substrings (i.e. k is the number of partition boundaries (between substrings)), and attempt to choose α to satisfy the Kraft equality:

$$1 \quad = \quad \sum_{k=0}^{n-1} \binom{n-1}{k} 2^{-\alpha k}, \tag{53}$$

we run into a problem. Letting $\zeta = 2^{-\alpha k}$, this sum is equal to:

$$(1 \; + \; \zeta)^{n-1}, \tag{54}$$

which can't possibly equal 1 (except in the limit $\alpha \to \infty$). Thus we're forced to include a normalizing constant, which we will write as $2^{-\beta}$. That is:

$$2^{\beta} \quad = \quad (1 \; + \; \zeta)^{n-1} \tag{55}$$

$$\beta \quad = \quad (n-1)\log(1+\zeta) \tag{56}$$

The encoding cost of a given partitioning is $\beta \; + \; \alpha k$ bits (where $\alpha > 0$ and β is given by equation (56)), which provides some flexibility in the degree to which we penalize the number of partitions.

4.3.1 Experiments Segmenting Binary Images

While binary images that are well described by spatially piecewise Bernoulli models are rare, it would be straightforward to use the results developed here to segment them. For example, a region merging scheme like that described in [12, 13] could be used and should be very fast since estimation of model parameters and associated code-lengths (the latter by table lookup) can be performed very fast. An MDL/quad-tree-based segmentation algorithm for such images and models was investigated by Nohre[21].

4.4 Gaussian Models and Greyscale/Color Image Segmentation

Of special interest to us is the problem of segmenting greyscale, color and general multi-band images. In previous work[12, 13] we have used models of spatially piecewise-polynomial greyscale surfaces plus Gaussian noise to perform segmentation using a region-merging scheme and an objective function based on MDL. This objective function was based on a two-part encoding scheme that used a chain-code encoding of region boundaries and the asymptotic expression derived by Rissanen[24] for the model-parameter encoding cost. The obvious extension of the current work and one we plan to pursue is the extension of Rissanen's new ML-code-based scheme to Gaussian models and their use in greyscale and vector image segmentation. Such results will also be useful in piecewise-polynomial curve fitting, for example.

Acknowledgment

Thanks to Eric Pauwels for many helpful discussions and for reading and commenting on a draft of this report.

References

[1] Serge Ayer and Harpreet Sawhney. Layered representation of motion video using maximum-likelihood estimation of mixture models and MDL encoding. In *5th ICCV*, pages 777–784, 1995.

[2] James O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, New York; Berlin, 1985. ISBN: 0-387-96098-8.

[3] Trevor Darrell and Alex P. Pentland. Cooperative robust estimation using layers of support. *IEEE Trans PAMI*, 17(5):474–487, 1995.

[4] B. Dom, W. Niblack, and J. Sheinvald. Feature selection with stochastic complexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 1989*, pages 241–248, 1989.

[5] Byron Dom and David Steele. 2^n-tree classifiers and realtime image segmentation. In *MVA '90: IAPR Workshop on Machine Vision Applications*, Tokyo, Japan, 1990. IAPR. (for expanded version see IBM Research Report 7558 (70424) 7/2/90).

[6] Byron Dom and David Steele. MDL-based pruning of 2^n-tree classifiers. Technical Report RJ 8673 (78235), IBM Research Division, IBM Almaden Research Center; San Jose, CA, March 1992.

[7] G. E. Hinton and D. van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the 6-th ACM Annual Conference on Computattional Learning Theory (COLT)*, pages 5–13, 1993.

[8] Qian Huang and Byron Dom. Quantitative methods of evaluating image segmentation. In *IEEE International Conference on Image Processing (ICIP).* IEEE, October 1995.

[9] H. S. Jeffreys. *Theory of Probability.* Oxford Univ. Press, 1961.

[10] J.Sheinvald, B.Dom, W.Niblack, and S.Banerjee. Combining edge pixels into parameterized curve segments using the MDL principle and the Hough transform. In Y.Mahdavieh and R.C. Gonzalez, editors, *Advances in Image Analysis.* SPIE Press Vol. PM08, 1992.

[11] J.Sheinvald, B.Dom, W.Niblack, and S.Banerjee. Detecting parameterized curve segments using the MDL principle and the Hough transform. In *Proceedings of CVPR 92.* IEEE Computer Society, June 1992.

[12] T. Kanungo, B. Dom, W. Niblack, and D. Steele. A fast algorithm for MDL-based multi-band image segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 609–616, Seattle, Washington, June 1994. IEEE Computer Society.

[13] T. Kanungo, B. Dom, W. Niblack, and D. Steele. MDL-based multi-band image segmentation using a fast region merging scheme. Technical Report 9960, IBM Research Division, May 1995.

[14] K. Keeler. Minimal length encoding of planar subdivision topologies with application to image segmentation. In *AAAI 1990 Spring Symposium of the Theory and Application of Minimal Length Encoding*, 1990.

[15] K. Keeler. Map representations and coding-based priors for segmentation. In *IEEE 1991 Conference on Computer Vision and Pattern Recognition*, 1991.

[16] G. D. Kendall and T. J. Hall. Optimal network construction by Minimum Description Length. *Neural Computation*, 5:210–212, 1993.

[17] Leonardo Landi and Byron Dom. A MDL based framework to segment images. In *ICANN '95*, 1995.

[18] Y. G. Leclerc. Constructing simple stable descriptions for image partitioning. *International Journal of Computer Vision*, 3:73–102, 1989.

[19] Y. G. Leclerc. Region grouping using the minimum-description-length principle. In *DARPA Image Understanding Workshop*, 1990.

[20] Manish Mehta, Jorma Rissanen, and Rakesh Agrawal. MDL-based decision tree pruning. In *Int'l Conference on Knowledge Discovery in Databases and Data Mining (KDD-95)*, Montreal, Canada, August 1995.

[21] Ragnar Nohre. *Topics in Descriptive Complexity*. PhD thesis, Technical University of Linkoping, 1993.

[22] Alex Pentland. Part segmentation for objects recognition. *Neural Computation*, 1:82–91, 1989.

[23] J. R. Quinlan and R. L. Rivest. Inferring decision trees using minimum description length principle. *Information and Computation*, 80, 1989.

[24] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 2(11):211–222, 1983.

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

[26] J. Rissanen. Information theory and neural nets. In P. Smolensky, M. Mozer, and D. Rumelhart, editors, *Mathematical Perspectives on Neural Networks*. Laurence Erlbaum Associates, 1994.

[27] J. Rissanen. Fisher information and stochastic complexity. *IEEE Trans Inf Th*, 42(1):40–47, January 1995.

[28] J. Rissanen and E. S. Ristad. Unsupervised classification with stochastic complexity. In *AAAI 1990 Spring Symposium on Theory and Application of Minimal Length Encoding*, 1990.

[29] J. Rissanen and M. Wax. Algorithm for constructing tree structured classifiers. *U.S. Patent No. 4,719,571*, 1988.

[30] J. Segen. Unsupervised feature selection for object recognition. *Pro. SPIE*, 360:132–153, 1982.

[31] J. Segen. Feature selection and constructive interference. In *Proc. 7th Int'l Conf Pat Rec*, pages 1344–1346, July 1984.

[32] J. Segen and A. C. Anderson. A minimal representation criterion for clustering. In *Proc. 12th ann Comp Sci and Stat Symp*, pages 332–334, 1979.

[33] J. Sheinvald, B. Dom, and W. Niblack. A modeling approach to feature selection. In *Proceedings of the 10th International Conference on Pattern Recognition (ICPR)*, pages 535–539, Atlantic City, NJ, June 1990.

[34] J. Sheinvald, B. Dom, W. Niblack, and D. Steele. Unsupervised image segmentation using the minimum description length principle. In *Proceedings of ICPR 92*, August 1992. For an expanded version see: *IBM Research Report RJ 8474 (76695)*, (11/1/91).

[35] R. S. Zemel. *A Minimum Description Length framework for unsupervised learning.* PhD thesis, Department of Computer Science, University of Toronto, 1993.

[36] S. C. Zhu, T. S. Lee, and A. L. Yuille. Region competition: Unifying snakes, region growing, energy/bayes/mdl for multi-band image segmentation. In *Fifth International Conference on Computer Vision*, pages 416–423, MIT, Cambridge, MA, June 1995. IEEE Computer Society - PAMI TC, IEEE Computer Society Press.

Appendices

A A Derivation of the Fisher-Information-Based Approximation to the Stochastic Complexity Based on Maximum-likelihood Encoding

A.1 Preliminaries

We define the *stochastic complexity* to be the minimized code-length:

$$L(x^n) \quad = \quad \min_{\xi} \left[-\log \hat{f}(x^n|\xi) \ + \ L(\xi) \right], \tag{57}$$

where

$$\hat{f}(x^n|\xi) \quad \equiv \quad \frac{f(x^n|\hat{\theta}^{(\xi)}(x^n), \xi)}{\int f(x^n|\hat{\theta}^{(\xi)}(x^n), \xi) dx^n} \tag{58}$$

and ξ is the model *structure parameter* and $k = k(\xi)$ is the number of *degrees of freedom* (number of real-valued parameters (i.e. components in $\theta^{(\xi)}$). The function $\hat{\theta}^{(\xi)}(x^n)$ is the ML estimate of $\theta^{(\xi)}$ for the data x^n. [In the following we will notationally omit the ξ and k dependence of the various terms in this equation.] In this section we will derive Rissanen's approximation[27] to $-\log \hat{f}(x^n)$.

Evaluating the denominator in $\hat{f}(x^n)$ (equation (58)) may be a problem. Because of this, Rissanen (in [27]) derives the following approximation.

$$-\log \hat{f}(x^n) \quad = \quad -\log f(x^n|\hat{\theta}(x^n)) \ + \ \frac{k}{2} \log \frac{n}{2\pi} \ + \ \log \int_{\Gamma} \sqrt{|I(\theta)|} d\theta \ + \ o(1), \tag{59}$$

where

- $I(\theta)$ is the *Fisher Information* matrix and $|I(\theta)|$ is its determinant. The $(i,j)^{th}$ component of $I(\theta)$ is

  $$-E_{\theta} \left\{ \frac{\partial^2 \log f(x|\theta)}{\partial\theta_i \partial\theta_j} \right\}, \tag{60}$$

  where $E_{\theta}\{\ldots\}$ denotes $\int \{\ldots\} f(x|\theta) dx$.

- $\Gamma \subset \mathbb{R}^k$ is the region of allowable θ values.

The primary assumptions he makes in his derivation are that: (1) $I(\theta)$ exists and is continuous; and (2) that as $n \to \infty$ the distribution of maximum-likelihood estimates of θ obeys the *central limit theorem*, which means that:

$$f_n(\hat{\theta}|\theta) \quad \to \quad |I(\theta)|^{1/2} \left(\frac{n}{2\pi}\right)^{k/2} \exp\left[-\frac{1}{2}(\hat{\theta} - \theta)^T nI(\theta)(\hat{\theta} - \theta) \right], \quad n \to \infty. \tag{61}$$

A.2 The Derivation

All but the first term to the right of the equals sign in (59) are an approximation to the log of the denominator of (58). As part of our derivation of the result of equation (59), we will show that this integral is equal to:

$$\int f(x^n|\hat{\theta}(x^n))dx^n \;=\; \int f_n(\hat{\theta}|\hat{\theta})d\hat{\theta}, \tag{62}$$

where, by the function $f_n(\hat{\theta}|\hat{\theta})$ in (62) we mean:

$$f_n(\hat{\theta}|\theta)\big|_{\theta=\hat{\theta}}. \tag{63}$$

That is, to obtain this function, we construct the probability density $f_n(\hat{\theta}|\theta)$ as described above. Then for each $\hat{\theta}$ value we substitute $\hat{\theta}$ for θ. We will derive this result (equation (62)) below, but first we show that it leads to the desired result (equation (59)).

To apply the result of equation (62) we need a form for $f_n(\hat{\theta}|\theta)$, which we will use as in equation (63) to obtain $f_n(\hat{\theta}|\hat{\theta})$. For this form we will use the central limit theorem (equation (61)). When we do this and make the substitution $\theta \rightarrow \hat{\theta}$, the argument of the exponential is zero and the exponential is therefore one, leaving us with:

$$f_n(\hat{\theta}|\hat{\theta}) \;\rightarrow\; |I(\hat{\theta})|^{1/2}\left(\frac{n}{2\pi}\right)^{k/2}, \quad \text{as n} \;\rightarrow\; \infty. \tag{64}$$

Substituting this for our integrand yields:

$$\int f(x^n|\hat{\theta}(x^n))dx^n \;=\; \left(\frac{n}{2\pi}\right)^{k/2}\int |I(\theta)|^{1/2}d\theta \tag{65}$$

whose log is:

$$\frac{k}{2}\log\frac{n}{2\pi} \;+\; \log\int\sqrt{|I(\theta)|}d\theta, \tag{66}$$

the result we sought to prove.

A.3 Showing That $\int f(x^n|\hat{\theta}(x^n))dx^n \;=\; \int f_n(\hat{\theta}|\hat{\theta})d\hat{\theta}$

In deriving the above result we relied on another result, namely:

$$\int f(x^n|\hat{\theta}(x^n))dx^n \;=\; \int f_n(\hat{\theta}|\hat{\theta})d\hat{\theta} \tag{67}$$

To demonstrate this we first write the integral in question as a Riemann sum:

$$\int f(x^n|\hat{\theta}(x^n)dx^n \;=\; \lim_{\epsilon\rightarrow 0}\sum_i\int_{\delta_i} f(x^n|\theta_i)dx^n, \tag{68}$$

where $\{\delta_i\}$ are the regions of integration of the individual integrals in the sum and ϵ is a parameter controlling their size; as $\epsilon \rightarrow 0$ the sizes of all the $\{\delta_i\}$ go to zero. The values $\{\theta_i\}$ are given by $\hat{\theta}(x^n)$ evaluated at the centroids of the $\{\delta_i\}$. Next note that if these regions are defined appropriately we can write:

$$\int_{\delta_i} f(x^n|\theta_i)dx^n \;=\; \int_{\phi_i} f_n(\hat{\theta}|\theta_i)d\hat{\theta}, \tag{69}$$

where the x^n regions are functions of the θ regions: $\{\delta_i(\phi_i) = \hat{\theta}^{-1}(\phi_i)\}$ [that is: δ_i is the set of all points x^n for which $\hat{\theta}(x^n) \in \phi_i$.] which obviously also means that:

$$\lim_{\epsilon \to 0} \sum_i \int_{\delta_i} f(x^n|\theta_i) dx^n = \lim_{\epsilon \to 0} \sum_i \int_{\phi_i} f_n(\hat{\theta}|\theta_i) d\hat{\theta}, \tag{70}$$

Finally, we recognize that the right-hand side of this equation is a Riemann-sum representation of the integral: $\int f_n(\hat{\theta}|\hat{\theta}) d\hat{\theta}$. Linking these results together, we have derived the desired result (equation (62)).

A.4 An Observation About $\pi_n(\theta)$

As we mentioned in the body of the paper, Rissanen, partly justified the use of the ML density in his definition of the stochastic complexity by "deriving" it by starting with a two-part code with redundancy removal and then using the following prior for encoding $\hat{\theta}$. [This is subtly different from what he did, but gets at the same idea.]

$$\pi_n(\theta) = \frac{\sum_{\{x^n|\hat{\theta}(x^n)=\theta\}} P(x^n|\theta)}{\sum_{\{\forall u^n\}} P(u^n|\hat{\theta}(u^n))}. \tag{71}$$

Two-part coding with redundancy removal gives a code-length formula of the form:

$$L(x^n) = -\log \left[\frac{P(x^n|\hat{\theta}(x^n))}{\sum_{\{u^n|\hat{\theta}(u^n)=\hat{\theta}(x^n)\}} P(u^n|\hat{\theta}(u^n))} \right] + \log \pi_n(\hat{\theta}(x^n)), \tag{72}$$

where the sum in the denominator of the first term is over all data vectors u^n whose ML parameter-vector estimates are equal to that of x^n: $\hat{\theta}(x^n)$. Substituting from equation (71) for π gives (after a little algebra) the ML code we've been using.

$$L(x^n) = -\log \hat{P}(x^n) = -\log \left[\frac{P(x^n|\hat{\theta}(x^n))}{\sum_{u^n} P(u^n|\hat{\theta}(u^n))} \right]. \tag{73}$$

This is all from [27]; the interesting observation is the following. If we use a continuous density $f(x^n|\theta)$ to compute $P(x^n|\theta)$, we may write.

$$\pi_n(\theta) = \frac{\int_{\{x^n|\hat{\theta}(x^n)=\theta\}} f(x^n|\hat{\theta}(x^n)) dx^n}{\int f(x^n|\hat{\theta}(x^n)) dx^n}. \tag{74}$$

Proceeding as we did in our derivation of Rissanen's approximation, we transform these integrals over x^n to integrals over θ (or $\hat{\theta}$; the same thing in this case) to obtain:

$$\pi_n(\theta) = \frac{f_n(\theta|\theta)}{\int f_n(\hat{\theta}|\hat{\theta}) d\hat{\theta}} \tag{75}$$

A word about notation: The density $f(\theta|\theta)$ is the same function in both the numerator and denominator of this expression. This is the density of $\hat{\theta}(x^n)$ estimates given a "real" parameter vector θ. For the numerator we might have written $f(\hat{\theta} = \theta|\theta)$ and in the denominator: $f(\hat{\theta}|\theta = \hat{\theta})$. Now, if we use the central-limit-theorem form for $f(\hat{\theta}|\theta)$ as we also did in our derivation of Rissanen's approximation, we have:

$$\lim_{n \to \infty} \pi_n(\theta) \equiv \pi_\infty(\theta) = \frac{|I(\theta)|^{1/2}}{\int |I(\hat{\theta})|^{1/2} d\hat{\theta}}, \tag{76}$$

where $\hat{\theta}$ serves simply as a dummy integration variable in the denominator.

This form (equation (76)) for π is known as *Jeffrey's prior*[9]. It was proposed because of its invariance under scale (dimensional) changes in the representation of θ. Note that the prior with which we began (in its various forms: equations (71),(74), and (75)) has the same invariance property. In fact this invariance holds for any tranformation of θ that maintains the same set of models available by choosing the correct values of the parameter vector. Suppose we transform the parameters θ by any transformation (not necessarily linear) to a corresponding set ψ. Then for any region in θ there is a corresponding region in ψ and the integrals of the two corresponding forms of our prior will be equal. These are versions of what are known in the Bayesian literature as "non-informative" priors. See [2, 3.3 "Noninformative Priors"] for a discussion.

As points of comparison, consider the following Bayesian estimates using the same prior.

1. Bayesian Mixture: If we use the form of the prior in equation (75) to construct a Bayesian mixture we have:
$$f(x^n) \quad = \quad \frac{\int f(x^n|\theta) f(\theta|\theta) d\theta}{\int f(\hat{\theta}|\hat{\theta}) d\hat{\theta}}. \tag{77}$$
We might for example use the log of this as the stochastic complexity. The associated MDL estimation would be equivalent to choosing the model structure k to maximize this $f(x^n)$.

2. MAP Estimate: The *maximum a posteriori* (MAP) estimate of θ is the one that maximizes $f(\theta|x^n)$, which is equivalent to maximizing $f(x^n|\theta)\pi(\theta)$, which is equivalent to an MDL-style two-part coding of the form:
$$\begin{aligned} L(x^n) &= -\log f(x^n|\theta) \; - \; \log \pi(\theta) && (78) \\ &= -\log f(x^n|\theta) \; - \; \log f(\theta|\theta) \; + \; \log \left[\int f(\hat{\theta}|\hat{\theta}) d\hat{\theta} \right]. && (79) \end{aligned}$$
The θ value that minimizes this code-length is the MAP estimate. We should point out that this two-part coding includes neither redundancy removal nor optimal-precision analysis.

B General Hypothesis Testing

B.1 Classical Theory

In two-hypothesis hypothesis testing, we have two possible distributions for describing the data. Denote the associated hypotheses by H_1 and H_2. We also have an observation (scalar or vector) $\zeta \in Z$ based upon which we will choose H_1 or H_2. Based upon some criteria we *construct a test* which has an associated *critical region* $V_1 \subset Z$, where $\zeta \in V \Rightarrow H_1$. Assuming that one of the two hypotheses is true, there are two possible errors that can be made:

1. H_1 is true, but we choose H_2.

2. H_2 is true, but we choose H_1.

Denote the probability of the first type of error by $\epsilon_{2|1}$ and the second by $\epsilon_{1|2}$. Given these definitions:
$$\epsilon_{1|2} \quad = \quad P(V_1|H_2) \quad = \quad \sum_{\zeta \in V_1} P(\zeta|H_2), \tag{80}$$

where there is a similar definition for $\epsilon_{2|1}$. It's common to use the likelihood ratio:

$$L(\zeta) \equiv \frac{P(\zeta|H_1)}{P(\zeta|H_2)}, \tag{81}$$

primarily because of something called the *Neyman-Pearson* lemma [see almost any statistics text for a discussion], which asserts (with an associated proof, of course) that for a given $\epsilon_{2|1}$, $\epsilon_{1|2}$ is minimized by a test of the form:

$$L(\zeta) \geq \kappa \Rightarrow H_1, \tag{82}$$

where the threshold κ is chosen to give the specified $\epsilon_{2|1}$. With what we've described thus far, it's usually possible to specify two of three parameters associated with such tests: the two error probabilities and the number of sample measurements making up ζ.

B.2 Bayesian Hypothesis Testing

What we've described thus far is classical hypothesis testing theory. This can be expanded to what is known as *Bayesian* hypothesis testing if we assume that the *a priori* probabilities of the two hypotheses $(P(H_1), P(H_2))$ are known. This theory is also frequently developed based on a *cost function* $\lambda(\hat{h}, h, \zeta)$, where $\hat{h}$, and h are the chosen and correct hypotheses respectively. Thus the expected cost of a test is given by:

$$C = \sum_{\zeta\in V_1}[\lambda(H_1,H_1,\zeta)P(H_1|\zeta)+\lambda(H_1,H_2,\zeta)P(H_2|\zeta)]+\sum_{\zeta\in V_2}[\lambda(H_2,H_1,\zeta)P(H_1|\zeta)+\lambda(H_2,H_2,\zeta)P(H_2|\zeta)]. \tag{83}$$

where,

$$P(H_1|\zeta) = P(\zeta|H_1)P(H_1)/P(\zeta) \tag{84}$$
$$P(\zeta) = P(\zeta|H_1)P(H_1) + P(\zeta|H_2)P(H_2) \tag{85}$$
$$\zeta \in V_2 \Rightarrow H_2 \tag{86}$$
$$V_1 \cup V_2 = Z \tag{87}$$
$$V_1 \cap V_2 = \emptyset. \tag{88}$$

The test (V_1, V_2) is chosen to minimize C. A special case of interest is the *classification error* cost function:

$$\lambda(H_1,H_2,\zeta) = \lambda(H_2,H_1,\zeta) = 1 \tag{89}$$
$$\lambda(H_1,H_1,\zeta) = \lambda(H_2,H_2,\zeta) = 0. \tag{90}$$

In this case the test cost reduces to:

$$\begin{aligned} C &= P(H_2)\sum_{\zeta\in V_1}P(\zeta|H_2) + P(H_1)\sum_{\zeta\in V_2}P(\zeta|H_1) && (91)\\ &= \sum_{\zeta\in V_1}P(\zeta,H_2) + \sum_{\zeta\in V_2}P(\zeta,H_1) && (92)\\ &= \sum_{\zeta\in V_1}P(H_2|\zeta)P(\zeta) + \sum_{\zeta\in V_2}P(H_1|\zeta)P(\zeta). && (93)\end{aligned}$$

[The three forms come from applying Bayes rule.] It is pretty easy to see that this is minimized by choosing as V_1 the set of all points for which $P(H_1|\zeta) \geq P(H_2|\zeta)$ or:

$$\frac{P(H_1|\zeta)}{P(H_2|\zeta)} \geq 1. \tag{94}$$

Using Bayes rule, equation (94) can be written:

$$\frac{P(\zeta|H_1)P(H_1)}{P(\zeta|H_2)P(H_2)} \geq 1. \tag{95}$$

In the special case where $P(H_1) = P(H_2) = 1/2$ this becomes

$$L(\zeta) \geq 1. \tag{96}$$

This is the case we consider here.

C An Analysis of the Non-monotonic Dependence of Error Probability on String Length in Bernoulli Hypothesis Testing

C.1 The Basic Hypothesis Testing Problem

We will discuss a hypothesis-testing exercise involving binary strings and Bernoulli distributions. We are given two binary strings of length $n/2$, where n is an even integer. Denote these by ζ_1, ζ_2 and let k_1 and k_2 represent the number of ones in these strings. We want to test the hypothesis that these came from two different distributions (with parameters θ_1 and θ_2) against the hypothesis that they are from the same distribution with parameter θ_0. Denote these hypotheses by H_2 and H_1 respectively.

We will perform this test in the usual way using the likelihood ratio:

$$L(\zeta_1, \zeta_2) = \frac{P(\zeta_1, \zeta_2|H_1)}{P(\zeta_1, \zeta_2|H_2)}, \tag{97}$$

which, in this case (Bernoulli models) becomes:

$$L(\zeta_1, \zeta_2) = \frac{P(\zeta_1, \zeta_2|\theta_0)}{P(\zeta_1|\theta_1)P(\zeta_2|\theta_2)} = \frac{\theta_0^{k_1+k_2}(1-\theta_0)^{n-(k_1+k_2)}}{\theta_1^{k_1}(1-\theta_1)^{\frac{n}{2}-k_1}\theta_2^{k_2}(1-\theta_2)^{\frac{n}{2}-k_2}} \geq 1. \tag{98}$$

C.2 The Unexpected Behavior

The unexpected behavior that we wish to understand is shown in Figure 8. There, total error probability $(\epsilon_{1|2} + \epsilon_{2|1})$ decays monotonically with total length n as expected. The two component error probabilities ($\epsilon_{1|2}$ and $\epsilon_{2|1}$) do not, however. We found this behavior surprising and sought to understand it. The associated explanation is presented in the next subsection.

C.3 The Explanation

First we will manipulate equation (98). Defining $t_i = \log\theta_i$, $s_i = \log(1-\theta_i)$, $r_i = t_i - s_i$ and taking the log of (98), yields:

$$(r_0 - r_1)k_1 + (r_0 - r_2)k_2 + \left[s_0 - \frac{1}{2}(s_1 + s_2)\right] n \geq 0. \tag{99}$$

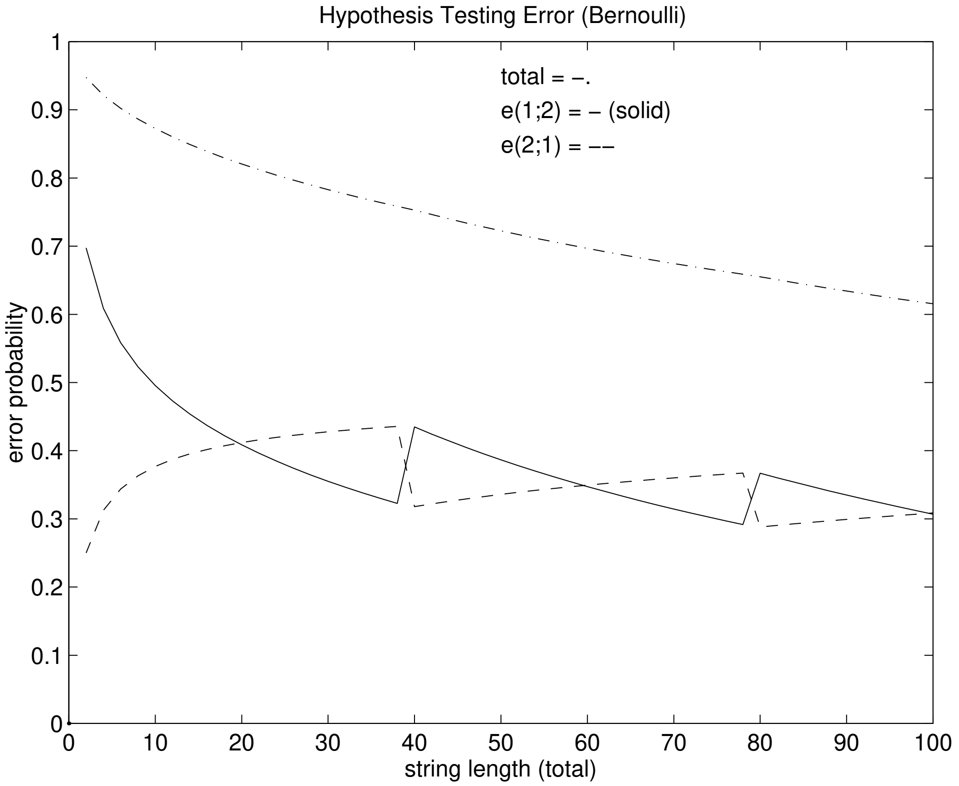


Figure 8: $\theta_0 = 0.5; \theta_1 = 0.45; \theta_2 = 0.55$

We further simplify the notation, writing this as:

$$g_1 k_1 \ + \ g_2 k_2 \ + \ g_0 n \ \ \geq \ \ 0 \tag{100}$$

This equation (100) defines the $V_1 : V_2$ decision boundary. In fact, that boundary is the straight line described by:

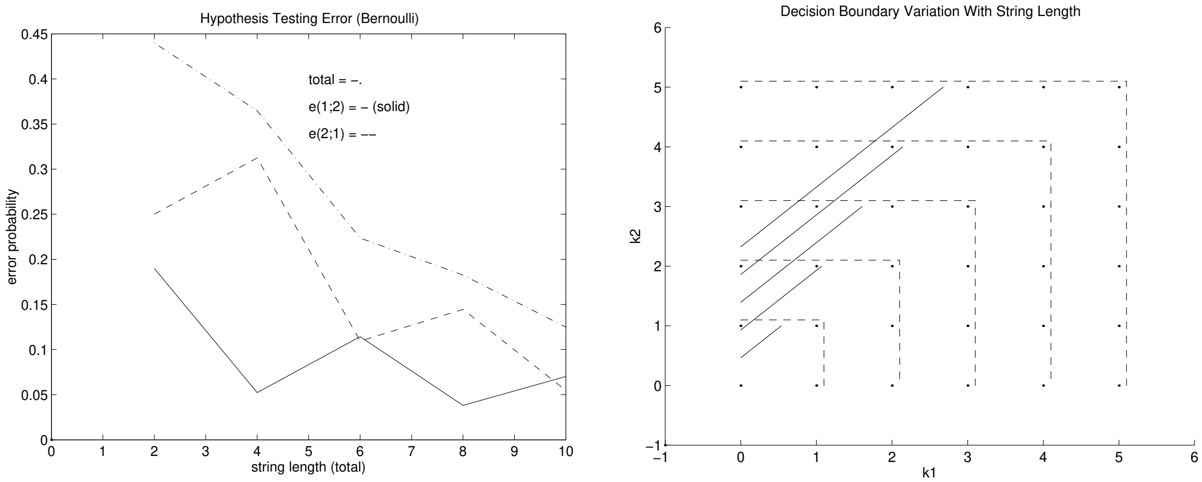
$$g_1 k_1 \ + \ g_2 k_2 \ + \ g_0 n \ \ = \ \ 0 \tag{101}$$

This is a special case of the more general form: $f(k_1, k_2; n) = 0$. Regarding this line, it is perpendicular to the vector: (g_1, g_2), with the slope of the line (dk_2/dk_1) being equal to (g_1/g_2).

If we examine the family of lines obtained by varying n in equation (101), we see that the perpendicular distance between adjacent lines (consecutive n values (changing by 2 each time) is:

$$||\Delta k|| \ \ = \ \ 2\frac{|g_0|}{\sqrt{g_1^2 + g_2^2}}. \tag{102}$$

We next use these results to explain a case that is simpler to analyze (at least graphically) than that of Figure 8. This case is shown in Figure 9, which corresponds to the values $\theta_0 = 0.5; \theta_1 = 0.1; \theta_2 = 0.9$. These values give a decision-boundary slope of 1 and $||\Delta k|| = 0.3288$. The decision boundary moves by steps of this size in the $(-1, +1)$ direction. Well, it so happens that there are diagonal rows of (k_1, k_2) grid points parallel to these decision boundaries and separated by a perpendicular distance of $\sqrt{2}/2 = 0.7071$. Yielding: $\sqrt{2}/(2||\Delta k||) = 2.1507$ for the frequency (in terms of $\Delta n = 2$ increments) at which the decision boundary will cross one of these diagonal rows of grid points. This means that roughly every other decision boundary will jump across one with



(a) (b)

Figure 9: The case $\theta_0 = 0.5; \theta_1 = 0.1; \theta_2 = 0.9$: (a) error probabilities for small string lengths; (b) the movement of decision boundaries with string length.

a jump being made on three consecutive boundaries every once in a while. This effect can be seen in Figure 9.

So how does this explain the non-monotonicity observed in Figures 8 and 9? For those n increments (by 2 each time) when the decision boundary doesn't cross a diagonal of grid points, it's as if it had remained fixed while increasing n. For these increments the observed changes in error probabilities are due to the increased total number of grid points ($= (n/2+1)^2$) and the changes in the forms of $P(\zeta_1, \zeta_2, H_1)$ and $P(\zeta_1, \zeta_2, H_2)$ with n.

As a final test let's see if we can predict the frequency of the jumps observed in Figure 8. Using equation (102), we obtain (for that case) $||\Delta k|| = 0.0354$. For a period of occurrence this gives: $\sqrt{2}/(2||\Delta k||) = 19.97$. So a jump will occur about every 20 decision boundaries, which corresponds to $\Delta n = 40$ in the figure. As can be seen, this prediction is correct.

D MATLAB Scripts Used in the Work Described in this Report

D.1 Script for Computing Code-Length Tables for Bernoulli Strings

```
% file = tab2gen.m
%
% This script generates 4 tables of values used by nrseg.m for doing
% "Bernoulli segmentation". These are codelengths for binary strings
% assuming Bernoulli models with the same parameter for the entire
% string. Table form is: tab(string length, number of ones in string).
%
% tab1: Bayesian-mixture-with-uniform-prior case
% tab2: Approximate ML-code-based
% tab3: Exact ML-code-based
```

```
% tab4: Two-part coding with log(n+1) bits to encode theta-hat.
%
tab1 = zeros(100,100);
tab2 = tab1; tab3 = tab1; tab4 = tab1;
maxlength = 100;
lnf = 0;
for lngth = 1:maxlength
  l2lngth = log2(lngth);             % log(n)
  l2lngthp1 = log2(lngth+1);         % log(n+1)
  lnf = lnf + l2lngth;               % log(n!)
  ll2l = lngth*l2lngth;              % n log(n)
  lngthlglngth = ll2l + 0.5*l2lngth;  % (n+1/2) log(n)
  if (lngth < 120)
    fnstuff = lnf + lq(lngth);
  else
    fnstuff = lngthlglngth;
  end
  for numons = 0:lngth
    if ((numons == 0) | (numons == lngth))
      tab23 = -ll2l;
    else
      tab23 = -((numons*log2(numons)) + ((lngth-numons)*log2(lngth-numons)));
    end
    tab1(lngth,1+numons) = l2lngthp1 + log2(choose(lngth,numons));
    tab2(lngth,1+numons) = tab23 + lngthlglngth;
    tab3(lngth,1+numons) = tab23 + fnstuff;
    tab4(lngth,1+numons) = tab23 + ll2l + l2lngthp1;
  end
end
```

D.2 Script to Plot n-Only-Dependent Terms in Code-Length Formulas

```
%
% This script generates and plots the only-n-dependent terms
% for the exact and approximate (Fisher Information) stochastic
% complexity based on ML coding for binary strings assuming a
% Bernoulli distribution.
%
lnf = 0; lngthlglngth = []; fnstuff = []; gn2p = [];
maxlength = 100;
```

```
for lngth = 1:maxlength
  l2lngth = log2(lngth);
  lnf = lnf + l2lngth;
  lastlll = lngth*l2lngth + 0.5*l2lngth;
  lngthlglngth = [lngthlglngth lastlll];
  gn2p = [gn2p (lngth*l2lngth + log2(1+lngth))];
  if (lngth < 120)
    fnstuff = [fnstuff (lnf + lq(lngth))];
  else
    fnstuff = [fnstuff lastlll];
  end
end
%
%plot(lngthlglngth,'--');
%hold on;
%plot(gn2p,'-.');
%semilogy(2.0*(fnstuff-lngthlglngth)./(fnstuff+lngthlglngth),'-');
%semilogy(abs(fnstuff-gn2p)./fnstuff,'-.');
plot(fnstuff-lngthlglngth,'--');
hold on;
plot(fnstuff-gn2p,'-.');
%plot(fnstuff,'-');
%ylabel('relative difference');
ylabel('difference');
%ylabel('g(n)');
xlabel('string length (n)');
hold off;
```

D.3 Script to Compute $\log q(n)$

```
%
% This function computes the log_2 of the sum q(n) needed in
% computing the exact Bernoulli-ML-code-based stochastic complexity
% of a binary string of length n.
%
function y = lq(n)
y = 1;
denom = prod(1:n);
s = n^n/denom;
for i = 1:n
 denom = i*(denom/(n-i+1));
 s = s + (i^i * (n-i)^(n-i)) / denom;
end
```

```
y = log2(s);
```

D.4 Script for Computing Hypothesis-Testing Error Probabilities

This version computes total error probability for the case of Bayesian mixtures with uniform prior.

```
% file = twocomp.m
%
% This is a special program that I´m continuously modifying. I´m using
% it for computing and plotting things in an effort to gain insight into
% the behavior of the computed probability of choosing 1 string (1 Bernoulli
% parameter) vs 2 in MDL hypothesis testing. This is for two strings
% of length n/2.
%
% TABLES: These contain the codelength for a single string using
%         Bernoulli models and various schemes. They are of the form:
%         tab(string length, number of ones + 1).
%
%   tab1: Bayesian-mixture-with-uniform-prior case
%   tab2: Approximate ML-code-based
%   tab3: Exact ML-code-based
%   tab4: Two-part coding with log(n+1) bits to encode theta-hat.
%
true = 1; false = 0;
ch1sum = []; ch2sum = []; ch3sum = []; ch4sum = [];
nv = [];
testp = []; testnstr = []; testnst = [];
% p = 0.75;
% p1 = 0.35; p2 = 0.65;
maxlngth = 100;
for n = 2:2:maxlngth
  n
  nv = [nv n];
  sum = 0; sump = 0; sumstr = 0;
  sum1 = 0; sum2 = 0; sum3 = 0; sum4 = 0;
  for k1 = 0:(n/2)
    for k2 = 0:(n/2)
      ksum = k1 + k2;
%      bd = (p^(ksum)*(1-p)^(n-ksum));        % single-string probability
%      bd = (p1^k1)*(1-p1)^(n/2-k1)*(p2^k2)*(1-p2)^(n/2-k2);
      bd2 = 2^(-tab1(n/2,k1+1))*2^(-tab1(n/2,k2+1)); % 2-string prob.
      bd1 = 2^(-tab1(n,ksum+1));              % 1-string probability
```

```
      nstr = choose(n/2,k1)*choose(n/2,k2);     % no. of string pairs w/ k1,k2
      sumstr = sumstr + nstr;
      bd1nstr = bd1*nstr;                      % prob. of (k1,k2)
      bd2nstr = bd2*nstr;                      % prob. of (k1,k2)
      sump = sump + bdnstr;
      if ((tab1(n,ksum+1)) < ((tab1(n/2,k1+1))+(tab1(n/2,k2+1))))
        sum1 = sum1 + bd2nstr;
      else
        sum1 = sum1 + bd1nstr;
      end
      if ((tab2(n,ksum+1)) < ((tab2(n/2,k1+1))+(tab2(n/2,k2+1))))
        sum2 = sum2 + bd2nstr;
      else
        sum2 = sum2 + bd1nstr;
      end
      if ((tab3(n,ksum+1)) < ((tab3(n/2,k1+1))+(tab3(n/2,k2+1))))
        sum3 = sum3 + bd2nstr;
      else
        sum3 = sum3 + bd1nstr;
      end
      if ((tab4(n,ksum+1)) < ((tab4(n/2,k1+1))+(tab4(n/2,k2+1))))
        sum4 = sum4 + bd2nstr;
      else
        sum4 = sum4 + bd1nstr;
      end
    end
  end
  testp = [testp sump];
  testnstr = [testnstr sumstr];
  ch1sum = [ch1sum (0.5*sum1)]; % mult. by 0.5 because prior of each hypoth.
  ch2sum = [ch2sum (0.5*sum2)];
  ch3sum = [ch3sum (0.5*sum3)];
  ch4sum = [ch4sum (0.5*sum4)];
end
plot(nv,ch1sum,'c-.');
hold on;
plot(nv,ch2sum,'y--');
plot(nv,ch3sum,'m-');
plot(nv,ch4sum,'w:');
xlabel('string length (n)');
ylabel('error probability');
%title('probability of choosing 1 string when 2 with Bayesian mixtures');
```

```
%title('total probability of error');
text(50,.25,'two-part code: ..');
text(50,.20,'Bayesian mixture: -.');
text(50,.15,'ML code (exact): -');
text(50,.1,'ML code (approximate): --');
hold off;
```

D.5 Script for Analyzing Non-Monotonic Behavior of Error Probabilities

```
%
% file = mech.m (for "mechanism")
%
% This script examines Bernoulli hypothesis testing.
%
% This is simple hypothesis testing. We have two binary strings.
% One hypothesis is that they came from the same Bernoulli process
% with parameter theta0, while the alternate hypothesis is that
% they came from two processes with parameters theta1 and theta2.
% We computer two error probabilities:
%    p12 = probability that we choose the one-process hypoth, when it's 2
%    p21 = probability that we choose the two-process hypoth, when it's 1
%
% maxlength = 44;
theta0 = 0.5; theta1 = 0.45; theta2 = 0.55;
maxlength = 2;
% theta0 = 0.5; theta1 = 0.1; theta2 = 0.9;
t0 = log2(theta0); t1 = log2(theta1); t2 = log2(theta2);
s0 = log2(1-theta0); s1 = log2(1-theta1); s2 = log2(1-theta2);
r0 = t0 - s0; r1 = t1 - s1; r2 = t2 - s2;
g1 = r0 - r1; g2 = r0 -r2;
gn = s0 - 0.5*(s1 + s2);
fn = [];
slope = -g1/g2; deltak = 2*abs(gn)/sqrt(g1^2 + g2^2);
close; hold on;
plot(-1,-1,'w.');
for n = 2:2:maxlength
  fn = gn*n;
  k2int = -fn/g2; k1int = -fn/g1;
  k2n2int = -n*(gn+(g1/2))/g2;
  k1n2int = -n*(gn+(g2/2))/g1;
  p1 = []; p2 = [];
```

```
  if ((k1int >= 0 ) & (k1int <= (n/2)))
    p1 = [p1 k1int]; p2 = [p2 0];
  end
  if ((k2int > 0 ) & (k2int < (n/2)))
    p1 = [p1 0]; p2 = [p2 k2int];
  end
  if ((k1n2int >= 0 ) & (k1n2int <= (n/2)))
    p1 = [p1 k1n2int]; p2 = [p2 (n/2)];
  end
  if ((k2n2int > 0 ) & (k2n2int < (n/2)))
    p1 = [p1 (n/2)]; p2 = [p2 k2n2int];
  end
  if (length(p1) == 2)
      plot(p1,p2,'y-');
  end
xv = [0 (n/2+.1) (n/2+.1)];
yv = [(n/2+.1) (n/2+.1) 0];
plot(xv,yv,'m--');
end
gridpoints = 0:(maxlength/2);
[XP, YP] = meshgrid(gridpoints, gridpoints);
plot(XP,YP,'m.');
hold off;
```

D.6 Script for Segmenting Binary Strings Using Bernoulli Models

```
% file = nrseg.m
%
% function [clp, posp] = seg(level, ppos, pos, posmin, ul, n, ns, cl, clmin, bs)
%
%  A list of components of encoding scheme and their lengths:
%    . length of the string in bits = n (no need to encode)
%    . number of segments: ns ( log(n) bits to encode)
%    . index of partitioning given ns ( log((n-1) \choose (ns-1)) bits
%    . length of k^th segment: length(k) (no need to encode; implicit
%                          in partitioning index.)
%    . Bernoulli parameter (actually number of ones) for each segment:
%                          numones(k) ( log(1+length(k)) bits to encode ).
%    . the actual k^th substring given all the above. This takes
%                     log(length(k) \choose numones(k)) bits.
%
```

```
% bs    = original input bit string
%
% tab1: Bayesian-mixture-with-uniform-prior case
% tab2: Approximate ML-code-based
% tab3: Exact ML-code-based
% tab4: two-part Bernoulli code
%
%
% ---------------------------------------
%
% initialize segment stating positions
%
% ---------------------------------------
%
n = length(bs);
true = 1; false = 0;
clminmin1 = Inf; clminmin2 = Inf; clminmin3 = Inf; clminmin4 = Inf;
clm1 = []; clm2 = []; clm3 = []; clm4 = [];
for ns = 1:4
  ns
  clmin1 = Inf;  clmin2 = Inf;  clmin3 = Inf; clmin4 = Inf;
  ul = [1 ((2:ns) + (n-ns)) n+1];
  index = [1 (2:ns) (n+1)];                % starting position of associated segment
  slength = [ones(1,ns-1) (n-ns+1)];       % slength of associated segment
  numones = [bs(1:(ns-1)) sum(bs(ns:n))];  % number of 1´s in associated segment
  cl1 = [ones(1,ns-1) tab1(slength(ns),numones(ns)+1)];
  cl2 = [ones(1,ns-1) tab2(slength(ns),numones(ns)+1)];
  cl3 = [ones(1,ns-1) tab3(slength(ns),numones(ns)+1)];
  cl4 = [ones(1,ns-1) tab3(slength(ns),numones(ns)+1)];
  done = false;
  while (~done)
    done1 = true; done2 = true; done3 = true; done4 = true;
    codelength1 = sum(cl1);
    if (codelength1 < clmin1)
      clmin1 = codelength1;
      segmin1 = index;
    end
    codelength2 = sum(cl2);
    if (codelength2 < clmin2)
      clmin2 = codelength2;
      segmin2 = index;
    end
```

```
  codelength3 = sum(cl3);
  if (codelength3 < clmin3)
    clmin3 = codelength3;
    segmin3 = index;
  end
  codelength4 = sum(cl4);
  if (codelength4 < clmin4)
    clmin4 = codelength4;
    segmin4 = index;
  end
  done = done1 & done2 & done3 & done4;
  for i = ns:-1:2;
    if (index(i) < ul(i))
      index(i) = index(i) + 1;
      slength(i-1) = slength(i-1) + 1;
      numones(i-1) = numones(i-1) + bs(index(i)-1);
      cl1(i-1) = tab1(slength(i-1),numones(i-1)+1);
      cl2(i-1) = tab2(slength(i-1),numones(i-1)+1);
      cl3(i-1) = tab3(slength(i-1),numones(i-1)+1);
      cl4(i-1) = tab4(slength(i-1),numones(i-1)+1);
      for j = i:ns
        if (j > i)
          index(j) = index(j-1)+1;
        end
        if (j < ns)
          slength(j) = 1;
          numones(j) = bs(index(j));
          cl(j) = 1;
        else
          slength(j) = n - index(j) + 1;
          numones(j) = sum(bs(index(j):n));
          cl1(j) = tab1(slength(j),numones(j)+1);
          cl2(j) = tab2(slength(j),numones(j)+1);
          cl3(j) = tab3(slength(j),numones(j)+1);
          cl4(j) = tab4(slength(j),numones(j)+1);
        end
      end;
      done = false;
      break;
    end
  end
end
```

```
  np = choose(n-1,ns-1);        % number of posssible partitionings with ns
  clmin1 = clmin1 + log2(np);   % add codelength for particular partitioning
  clmin2 = clmin2 + log2(np);
  clmin3 = clmin3 + log2(np);
  clmin4 = clmin4 + log2(np);
  clm1 = [clm1 clmin1];
  clm2 = [clm2 clmin2];
  clm3 = [clm3 clmin3];
  clm4 = [clm4 clmin4];
  if (clmin1 <= clminmin1)
    clminmin1 = clmin1;
    segminmin1 = segmin1;
    nsmin1 = ns;
    done1 = false;
  end
  if (clmin2 <= clminmin2)
    clminmin2 = clmin2;
    segminmin2 = segmin2;
    nsmin2 = ns;
    done2 = false;
  end
  if (clmin3 <= clminmin3)
    clminmin3 = clmin3;
    segminmin3 = segmin3;
    nsmin3 = ns;
    done3 = false;
  end
  if (clmin4 <= clminmin4)
    clminmin4 = clmin4;
    segminmin4 = segmin4;
    nsmin4 = ns;
    done4 = false;
  end
  done = done1 & done2 & done3 & done4;
  if (done) break; end;
end
clminmin1 = clminmin1 + log2(n);   % add codelength for number of segments.
clminmin2 = clminmin2 + log2(n);
clminmin3 = clminmin3 + log2(n);
clminmin4 = clminmin4 + log2(n);
```

D.7 Script for Segmenting Multiple Binary Strings Using Bernoulli Models

```
% file = nrsegloop.m
%
% This script segments binary strings assuming Bernoulli models.
%
% A list of components of encoding scheme and their lengths:
%     . length of the string in bits = n (no need to encode)
%     . number of segments: ns ( log(n) bits to encode)
%     . index of partitioning given ns ( log((n-1) \choose (ns-1)) bits
%     . length of k^th segment: length(k) (no need to encode; implicit
%                         in partitioning index.)
%     . Bernoulli parameter (actually number of ones) for each segment:
%                         numones(k) ( log(1+length(k)) bits to encode ).
%     . the actual k^th substring given all the above. This takes
%                    log(length(k) \choose numones(k)) bits.
%
%  bs    = original input bit string
%
% tab1: Bayesian-mixture-with-uniform-prior case
% tab2: Approximate ML-code-based
% tab3: Exact ML-code-based
% tab4: two-part Bernoulli code
%
%
% p1 = 0.0; p2 = 1.0; deltap = 0.05
% p1 = 0.2; p2 = 0.8   % saved workspace
% p1 = 0.1; ps = 0.9   % nshist1s, etc.
% p1 = 0.1; p2 = 0.9; p3 = 0.5; % nshist1sav3, etc.
% p1 = 0.3; p2 = 0.7;  % nshist1sav2, etc.
% p1 = 0.05; p2 = 0.95;
% p1 = 0.2; p2 = 0.8; p3 = 0.5;
p1 = 0.1; p2 = 0.9; p3 = 0.5;
nshist1 = zeros(1,10); hd1 = zeros(2,52);
nshist2 = zeros(1,10); hd2 = zeros(2,52);
nshist3 = zeros(1,10); hd3 = zeros(2,52);
nshist4 = zeros(1,10); hd4 = zeros(2,52);
lnhd = 52;
cns = 3;                                % number of substrings
% bndry = [1 51 101];                    %"real" boundary locations.
bndry = [1 34 67 101];                     %"real" boundary locations.
```

```
lbt = length(bndry);
for trial = 1:20;
%  bs = [(rand(1,50) > p1) (rand(1,50) > p2)];
  bs = [(rand(1,33) > p1) (rand(1,33) > p3) (rand(1,34) > p2)];
%  ---------------------------------------
%
%  initialize segment stating positions
%
% ---------------------------------------
%
n = length(bs);
true = 1; false = 0;
clminmin1 = Inf; clminmin2 = Inf; clminmin3 = Inf; clminmin4 = Inf;
clm1 = []; clm2 = []; clm3 = []; clm4 = [];
for ns = 1:n
  [trial ns]
  clmin1 = Inf;  clmin2 = Inf;  clmin3 = Inf; clmin4 = Inf;
  ul = [1 ((2:ns) + (n-ns)) n+1];
  index = [1 (2:ns) (n+1)];               % starting position of associated segment
  slength = [ones(1,ns-1) (n-ns+1)];      % slength of associated segment
  numones = [bs(1:(ns-1)) sum(bs(ns:n))];  % number of 1´s in associated segment
  cl1 = [ones(1,ns-1) tab1(slength(ns),numones(ns)+1)];
  cl2 = [ones(1,ns-1) tab2(slength(ns),numones(ns)+1)];
  cl3 = [ones(1,ns-1) tab3(slength(ns),numones(ns)+1)];
  cl4 = [ones(1,ns-1) tab3(slength(ns),numones(ns)+1)];
  done = false;
  while (~done)
    done1 = true; done2 = true; done3 = true; done4 = true;
    codelength1 = sum(cl1);
    if (codelength1 < clmin1)
      clmin1 = codelength1;
      segmin1 = index;
    end
    codelength2 = sum(cl2);
    if (codelength2 < clmin2)
      clmin2 = codelength2;
      segmin2 = index;
    end
    codelength3 = sum(cl3);
    if (codelength3 < clmin3)
      clmin3 = codelength3;
      segmin3 = index;
```

```
  end
  codelength4 = sum(cl4);
  if (codelength4 < clmin4)
    clmin4 = codelength4;
    segmin4 = index;
  end
  done = done1 & done2 & done3 & done4;
  for i = ns:-1:2;
    if (index(i) < ul(i))
      index(i) = index(i) + 1;
      slength(i-1) = slength(i-1) + 1;
      numones(i-1) = numones(i-1) + bs(index(i)-1);
      cl1(i-1) = tab1(slength(i-1),numones(i-1)+1);
      cl2(i-1) = tab2(slength(i-1),numones(i-1)+1);
      cl3(i-1) = tab3(slength(i-1),numones(i-1)+1);
      cl4(i-1) = tab4(slength(i-1),numones(i-1)+1);
      for j = i:ns
        if (j > i)
          index(j) = index(j-1)+1;
        end
        if (j < ns)
          slength(j) = 1;
          numones(j) = bs(index(j));
          cl(j) = 1;
        else
          slength(j) = n - index(j) + 1;
          numones(j) = sum(bs(index(j):n));
          cl1(j) = tab1(slength(j),numones(j)+1);
          cl2(j) = tab2(slength(j),numones(j)+1);
          cl3(j) = tab3(slength(j),numones(j)+1);
          cl4(j) = tab4(slength(j),numones(j)+1);
        end
      end;
      done = false;
      break;
    end
  end
end
np = choose(n-1,ns-1);        % number of posssible partitionings with ns
clmin1 = clmin1 + log2(np);   % add codelength for particular partitioning
clmin2 = clmin2 + log2(np);
clmin3 = clmin3 + log2(np);
```

```
  clmin4 = clmin4 + log2(np);
  clm1 = [clm1 clmin1];
  clm2 = [clm2 clmin2];
  clm3 = [clm3 clmin3];
  clm4 = [clm4 clmin4];
  if (clmin1 <= clminmin1)
    clminmin1 = clmin1;
    segminmin1 = segmin1;
    nsmin1 = ns;
    done1 = false;
  end
  if (clmin2 <= clminmin2)
    clminmin2 = clmin2;
    segminmin2 = segmin2;
    nsmin2 = ns;
    done2 = false;
  end
  if (clmin3 <= clminmin3)
    clminmin3 = clmin3;
    segminmin3 = segmin3;
    nsmin3 = ns;
    done3 = false;
  end
  if (clmin4 <= clminmin4)
    clminmin4 = clmin4;
    segminmin4 = segmin4;
    nsmin4 = ns;
    done4 = false;
  end
  done = done1 & done2 & done3 & done4;
  if (done) break; end;
end
clminmin1 = clminmin1 + log2(n);   % add codelength for number of segments.
clminmin2 = clminmin2 + log2(n);
clminmin3 = clminmin3 + log2(n);
clminmin4 = clminmin4 + log2(n);
ns1 = length(segminmin1) - 1;
nshist1(ns1) = nshist1(ns1) + 1;
ns2 = length(segminmin2) - 1;
nshist2(ns2) = nshist2(ns2) + 1;
ns3 = length(segminmin3) - 1;
nshist3(ns3) = nshist3(ns3) + 1;
```

```
ns4 = length(segminmin4) - 1;
nshist4(ns4) = nshist4(ns4) + 1;
hd1 = hd1 + comphd(bndry,segminmin1,lnhd);
hd2 = hd2 + comphd(bndry,segminmin2,lnhd);
hd3 = hd3 + comphd(bndry,segminmin3,lnhd);
hd4 = hd4 + comphd(bndry,segminmin4,lnhd);
save
end
```

D.8 Script for Plotting Binary Strings with Segmentation Results

```
% MATLAB script for displaying binary strings and associated
% Bernoulli segmentation results.
%
%
%--------------------------------
%  build image matrix and display
%--------------------------------
%
l = length(bs);
im = [];
height = ceil(l/4);
for i = 1:height
  im = [im bs'];
end
im = im';
close;
colormap(hot);
image(255*im);
axis equal;
set(gca,'Ytick',[]);             % no y-axis tick marks
hold on;
plot([0 l],[0.5 0.5],'w:');      % draw horiz. bndy of bit string
plot([0 l],[(height+0.5) (height+0.5)],'w:');
%
%-----------------------------------------------------------
%mark and label segmentation boundaries from various schemes
%-----------------------------------------------------------
%
```

```
for i = 2:(length(segminmin1)-1)   % Bayesian mixture
  text(segminmin1(i),-1,'v');
  text(segminmin1(i),-8,'1');
end
for i = 2:(length(segminmin4)-1)   % two-part
  text(segminmin4(i),-1,'v');
  text(segminmin4(i),-4,'4');
end
for i = 2:(length(segminmin2)-1)   % ML code (approx.)
  text(segminmin2(i),27,'^');
  text(segminmin2(i),29,'2');
end
for i = 2:(length(segminmin3)-1)   % ML code (exact)
  text(segminmin3(i),27,'^');
  text(segminmin3(i),33,'3');
end
text(10,-20,'1: Bayesian mixture with uniform prior');
text(10,-15,'4: two-part code');
text(10,40,'2: maximum-likelihood code (approximate)');
text(10,45,'3: maximum-likelihood code (exact)');
title('Bernoulli segmentation result');
hold off;
```

D.9 Script for Plotting Segmentation Results

```
%
% file = label.m
%
% set(gca,'Xtick',(1:length(clm1)));             % only integers
% xlabel('number of substrings')
% ylabel('encoding length (bits)');
% title('probability of choosing 1 string when 2 with Bayesian mixtures');
plot(dp,n1,'yo');
hold on;
plot(dp,n1,'y-.');
plot(dp,n2,'c+');
plot(dp,n2,'c--');
plot(dp,n3,'m*');
plot(dp,n3,'m-');
plot(dp,n4,'gx');
plot(dp,n4,'g:');
```

```
set(gca,´Xtick´,dp);
xlabel(´Bernoulli parameter (theta) for 1st 50 bits´);
ylabel(´count´);
title(´number of trials (out of 20) wrong no. substrings chosen´);
text(.11,14,´two-part code: .. (x)´);
text(.11,13,´Bayesian mixture: -. (o)´);
text(.11,12,´ML code (exact): - (*)´);
text(.11,11,´ML code (approximate): -- (+)´);
hold off;
```