RJ 10116 (91932) April 13, 1998
Computer Science

Research Report

THE MINIMUM DESCRIPTION LENGTH PRINCIPLE AND ILL-POSED PROBLEMS IN COMPUTER VISION

Byron Dom

IBM Research Division
Almaden Research Center
650 Harry Rd.
San Jose, CA 95120-6099

IBM Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

THE MINIMUM DESCRIPTION LENGTH PRINCIPLE AND ILL-POSED PROBLEMS IN COMPUTER VISION

Byron Dom

IBM Research Division
Almaden Research Center
650 Harry Rd.
San Jose, CA 95120-6099
email: dom@almaden.ibm.com

April 13, 1998

ABSTRACT: Ill-posed problems are ubiquitous in computer vision. They are inverse problems where the associated inverse operation is not defined. This paper starts by defining and discussing ill-posed problems with an emphasis on computer vision problems. It then describes an approach to statistical estimation known as the Minimum Description Length principle (MDL). Following this, the application of MDL to these ill-posed problems is described and example applications are discussed.

Keywords: computer vision, MDL, minimum description length, regularization

1. Overview

Our goal in writing this paper is to discuss the issue of ill-posed problems in computer vision and a statistical estimation paradigm, the *Minimum Description Length Principle* (MDL) for solving them. We start by defining and discussing ill-posed problems with an emphasis on computer vision problems. We then describe the Minimum Description Length principle. Following this description, we discuss the application of MDL to machine-learning and computer-vision applications, taking as an example the problem of unsupervised image segmentation.

1.1. Ill-Posed Problems in Statistical Inference

There frequently arise in every-day life problems where there is insufficient information to obtain an exact or even a formally approximate solution even in principle. Such problems, when posed without careful thought, frequently have a large (possibly infinite) number of potential solutions or they do have a single solution, but that solution is absurd, clearly not desirable. Despite this, we (the human mind/brain/visual system) solve such problems all the time, often with little or no conscious thought. Such problems are often referred to as "ill-posed" or "under-constrained". An example is the curve-fitting exercise depicted in Figure 1. These points were generated by adding white Gaussian noise to a 3rd-degree polynomial. Looking at this data, we can't say for certain what function generated it, even if we know it was a polynomial, but, if asked to pick the "most likely" polynomial degree we will usually choose correctly.

As stated thus far, this curve-fitting problem is "ill-posed", if for no other reason, because it is vague. If we use more precise language, we might state it as something like "choose the polynomial that gives the best fit to the data", where "best" would be defined in terms of some error criterion like mean-square deviation. This problem definition suffices if the polynomial degree is fixed, leaving us only the problem of choosing the coefficients. If we are free to choose the degree, however, the "best" solution by our criterion will be the Lagrange interpolating polynomial, which goes through every data point. This same situation is obviously also true for other types of curve fitting as well: fitting Fourier series, splines, and so on. The crucial issue is that of model complexity. Without further constraining the solution, we will always obtain a better fit (as measured by criteria such as least squares) by increasing the model complexity until we reach the point where we have either chosen the most complex model available or we have succeeded in fitting the data exactly, with no residual error between the data and model values. ["Complexity" is usually associated fairly directly with the number of adjustable parameters the model has.]

1.2. The (Ill-Posed) Computer Vision Problem

In computer vision we usually seek an abstract (and therefore simplified) description of a real scene, which may be either static or time varying. Letting D denote the desired

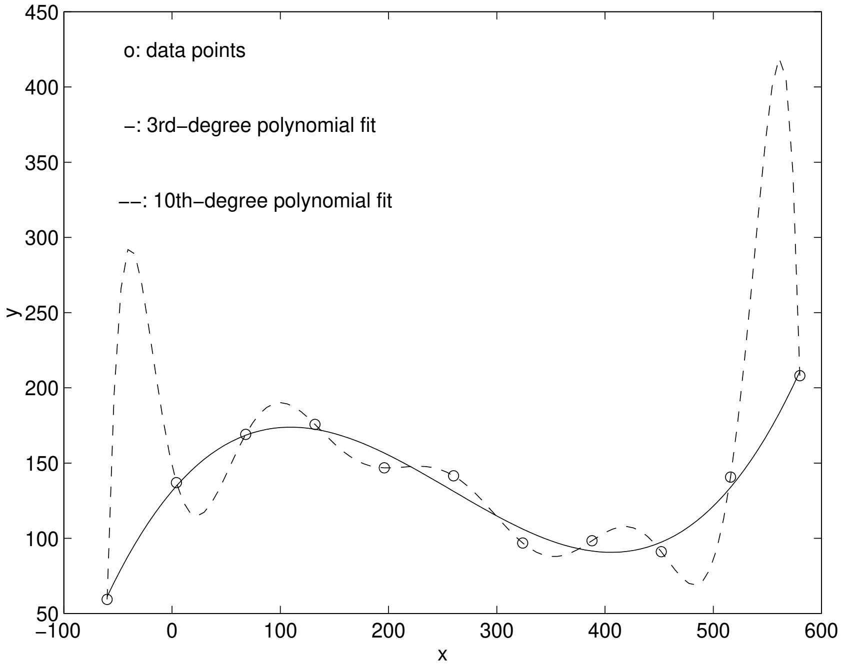

Figure 1: A curve fitting problem demonstrating the problem of over fitting

description and S the scene, we will write this as

$$D = \Omega(S), \quad (1)$$

which assumes that, in principle, given an absurdly detailed description of S, we could compute D using some procedure, Ω.

We are given one or more digital images from which to extract this description. These images might be still images of a single static scene, where each image corresponds to a different spectral band of light (e.g. a color image) or different sensing modalities (e.g. an optical image and an acoustic range image). They might also be multiple images taken from different positions (e.g. video frames taken of a static scene from a moving camera) and so on. We indicate the process that produces the image(s) I from the scene S as:

$$I = \Phi(S), \quad (2)$$

where the operator Φ consists, in general, of both a deterministic and a stochastic part. In fact, if the image acquisition process is complicated, stochastic degradations (e.g. "noise") may enter at multiple points in the process. It is also common to treat part of the natural scene itself as a kind of stochastic degradation of an ideal scene. An example of this is the treatment of image texture as a kind of noise in many texture-analysis/description approaches.

The description we seek may be at any level of abstraction and any level of precision. It might be a high-level description like a list of objects (e.g. car, cup, cat) and their positions and orientations or it might be an intermediate-level description such as might result from an image segmentation - a list of image regions. Such an intermediate description may be desired as an end in itself or as an intermediate step on the way to a higher level description. What we seek in this context is an estimation procedure Ψ that will operate on I to give us an estimate $\hat{D}$ of D:

$$\hat{D} = \Psi(I). \quad (3)$$

Ideally $\Psi = \Omega(\Phi^{-1})$ but the inverse transformation Φ^{-1} usually doesn't exist, resulting in an *ill-posed problem*.

1.3. Solving Ill-Posed Problems

Minimum Description Length (MDL) methods and most other approaches to solving this problem usually assume a mathematical form for the degradation process, which is combined with a description of S (usually containing D explicitly), producing a *model* M for the observed data (I in this case). The model-complexity issue almost always shows up in the estimation of D and may show up in other parts of the model as well. Because of the practical limits on our knowledge about S and computational-tractability considerations, the model form will usually be simple relative to the inherent complexity of S and the image acquisition process.

The most common program for solving this and other ill-posed, inverse problems is the following.

1. Construct an *objective function*:

$$\Phi(I, M) \text{ where } D \subset M,$$

where I is the data, M is the model and D is the sought description. The objective function $\Phi(I, M)$ is constructed so that the desired solution corresponds to its global minimum. An important point is that Φ must be based in part on some "regularizing" principle, which penalizes model complexity in some way that is based on sound principles. The notation "$D \subset M$" is simply meant to imply that the description is, in general, part of the model specification.

2. Devise an *optimization procedure* to find the optimal solution:

$$\hat{M} = \arg\{\min_M[\Phi(I, M)]\}$$

In most practical problems, it is not feasible to find the global minimum of Φ with certainty in a reasonable (i.e. polynomial) amount of time. Frequently (if not usually) exhaustive search is required to guarantee this. For this reason, sub-optimal procedures that find a local minimum (not guaranteed to be the the global minimum) are used. Many problems are such that, even when the solution found isn't actually the global minimum, it is adequate for the intended purpose.

As will be discussed in the subsequent section, MDL provides a basic principle and associated techniques for constructing $\Phi(I, M)$, whereas the algorithm devised to find $\hat{M}$ will depend on the form of Φ, the particular problem and its associated computational constraints.

To make all of this more concrete, consider the problem of grey-scale image partitioning or segmentation. In this process the image is divided into mutually exclusive regions whose union is usually the entire image. [Occasionally people speak of segmenting single regions. Such cases can be viewed as a segmentation of the image into two regions - the region of interest and everything else.] Let us assume that the D we seek is a description of the partitioning of the image into regions. To adequately model most images we will need to have more than just a description of the region boundaries. We will need some description of the spatial distribution of grey-levels within each region. One such model is to assume that the grey-levels are normally and independently distributed with a mean that is a polynomial function of the spatial image coordinates (see [30] and [23] for example). Let x represent the spatial image coordinates (e.g. the pixel coordinates (i, j)) and let y represent that associated grey-value.

We may write this model as a probability distribution:

$$p(\mathbf{y}|\theta) = \prod_j \prod_{\{i|x_i \in \rho_j\}} p(y_i \mid \theta_j), \tag{4}$$

where

$$p(y_i|\theta_j) = \frac{1}{\sqrt{(2\pi)}\sigma_j} \exp\left\{-\frac{[y_i - \mu_j(x_i)]^2}{2\sigma_j^2}\right\}, \tag{5}$$

where

- θ is the vector of all real-valued adjustable model parameters,
- $\{\rho_j\}$ are the regions composing our segmentation,
- θ_j is the vector of parameters associated only with region ρ_j.
- $\mathbf{y} = \{y_i\}$ a vector composed of all of image grey-levels and
- $\mu_j(x)$, the spatially dependent (polynomial) mean of this distribution for the region ρ_j.
- $x = \{x_i\}$ the pixel spatial coordinate values

Note that for this model θ consists of the polynomial coefficients of the grey-scale surfaces and the standard deviations $\{\sigma_j\}$.

In this case $I = (\mathbf{x}, \mathbf{y})$ and a complete specification of the model M consists of the following:

1. a specification of the partitioning of the image into the set of regions $\{\rho_j\}$. This would include the number of regions and a list of the pixels contained in each region or (more likely) a description of the region boundaries.
2. the model parameters for each region: the polynomial degree, the polynomial coefficients and the standard deviation. The latter two make up what we have referred to as θ_j.

We usually use the term "model" to refer to a specific model instance, which corresponds to a specific set of values of these parameters (both discrete and real-valued). Our model class $\mathcal{M}$ is the set of all such models (model instances) we consider. In this case (as in most) there is no precise, explicit specification of the transformation Φ, but its manifestations are implicit in our choice of $\mathcal{M}$. In some applications the description D we seek may consist of all this information, but in many applications, only the region description is of interest.

If we formulate this problem by saying that we seek the best least-squares fit to the image data, then we can simply make every pixel a separate region and associate a constant (0-degree) polynomial with each such region, yielding a perfect fit to the data but a useless description. Just as in the previously presented curve-fitting problem, we need some further constraint: something that penalizes the model complexity. This additional "something" will be provided by MDL, which we will use to formulate a $\Phi(I, M)$ for this problem in a subsequent section.

2. The Minimum Description Length Principle

In simple terms the Minimum Description Length Principle might be stated as follows.

> The best model (from a selected class) for describing a set of data is that giving the shortest code length when used to encode (compress) the data.

2.1. The History of MDL

The roots of MDL have been traced at least as far back as William of Ockham (circa 1320) to whom is attributed what is now referred to as "Ockham's Razor", which has been frequently invoked by physicists. Paraphrasing, this principle might be stated as follows.

> If two models describe a collection of data equally well and we have no other prior knowledge on which to base our choice, we should choose the simpler of the two.

In more recent terms, it can be traced to two fields. The first of these is *Information Theory*, whose foundation is the landmark work of Shannon[54] and the second is *Algorithmic Complexity*, which is usually attributed to Kolmogorov[29] [It is even referred to by some as "Kolmogorov Complexity".] and its roots appear to lie in his work and that of Solomonov[58] and Chaitin[4]. While many workers have contributed to it, MDL itself is usually associated with the work of Rissanen, its most prolific proponent whose earliest paper on the subject is [39]. We will not discuss this further except to refer the reader to Leclerc[31] and Rissanen [41] for more detailed historical accounts.

2.2. MDL Fundamentals

The *Minimum Description Length Principle* (MDL) is a paradigm for doing statistical estimation, specifically model selection. MDL's strength is in determining model *structure* (a.k.a. *order* or *complexity*).

MDL has been discussed in detail by Rissanen[41]. We summarize it briefly here, referring the reader to [41] and other cited works for more details.

The estimation problem it addresses is the following. We are given:

1. a vector $\mathbf{y}$ of n data values, $\{y_i | i = 1, 2, \ldots, n\}$. In most cases $\mathbf{y}$ will be considered to be values of a dependent variable and we will be given an associated vector $\mathbf{x}$ of n values of an independent variable, x: $\{x_i | i = 1, 2, \ldots, n\}$. [This is consistent with our use of these symbols (x and y) in our previous segmentation example, but constitutes a generalization of that usage.] Exceptions to this dependent/independent split are problems like density estimation and pure clustering, where there are just values of variables. We will denote the set of possible $\mathbf{y}$ values by Λ which may be either finite or infinite in extent. In general the individual components (points) of which $\mathbf{y}$ and $\mathbf{x}$ are composed will be vector-valued. In our previous segmentation example the individual values of x were two-dimensional vectors (pixel coordinates), whereas the y values were scalars (grey-levels). If we treat color-image segmentation then the y values will also be vector-valued.

   While we will sometimes treat $\mathbf{y}$ and $\mathbf{x}$ as real-valued for theoretical purposes, we assume that (as is always true in the real world) they are given to finite precision (quantized). The *quantization cells* (sets of real-valued points that all quantize to the same $\mathbf{y}$ value) correspond to these vectors are represented by $\delta\mathbf{y}$ and $\delta\mathbf{x}$, which are thus dimension-n subsets (regions) of $\mathbb{R}^n$.

2. a class Ξ of candidate models. Each model instance (element of Ξ) will be specified by the pair $(\theta^{(\xi)}, \xi)$ with which is associated a probability distribution $P(\mathbf{y}|\theta^{(\xi)}, \xi)$, where the *model structure* ξ is inherently discrete, but $\theta^{(\xi)}$ is usually assumed theoretically to be real-valued with the set of its possible values $\Gamma(\xi)$ being a dimension-$k(\xi)$ subset of $\mathbb{R}^{k(\xi)}$, i.e. $\theta^{(\xi)} \in \Gamma(\xi) \subset \mathbb{R}^{k(\xi)}$, where $k(\xi)$ is the number of components in $\theta^{(\xi)}$. In our segmentation problem ξ corresponds to the partitioning of the image into regions and the associated enumeration of the corresponding polynomial degrees, whereas $\theta^{(\xi)}$ is composed of the associated polynomial coefficients and standard deviations $\{\sigma_j\}$. [A point on notation: throughout this discussion we will use the symbols - $\{M \equiv (\theta^{(\xi)}, \xi)\}$ - interchangeably, depending on what we want to emphasize.]

Using these definitions we might reformulate our previous simple statement of the Minimum Description Length Principle as follows:

> Given the data $(\mathbf{x}, \mathbf{y})$ and class Ξ of models, we will choose the model $(\theta^{(\xi)}, \xi) \in \Xi$ that allows the data to be encoded with the shortest code length. The coding scheme will be one that uses the model $P(\mathbf{y}|\theta^{(\xi)}, \xi)$ in the sense of Shannon[54] to construct a code with lengths that are approximately $-\log P(\mathbf{y}|\theta^{(\xi)}, \xi)$ bits (where "log" represents the base-two logarithm) plus the number of bits required to encode the model itself.

In traditional Shannon coding we assume that $P(\mathbf{y}|\theta^{(\xi)}, \xi)$ is known *a priori* (i.e. not estimated), which gives us our code-length function. Algorithms such as Huffman's (see for example [59, Section 2.5; p. 39ff]) can be used to actually construct such codes, but we won't be concerned with this. Our estimation will be based on the associated length formulas. As mentioned, in MDL-based estimation we use models of the form: $P(\mathbf{y}|\theta^{(\xi)}, \xi)$. These models can be quite complicated in some cases. Frequently, we will use models involving continuous probability density functions $f(\mathbf{y})$ and using this, $P(\mathbf{y}) = \int_{\{\mathbf{w}\in\delta\mathbf{y}\}} f(\mathbf{w})d\mathbf{w}$, which is usually approximated by: $P(\mathbf{y}) = f(\mathbf{y}) \cdot ||\delta\mathbf{y}||$, where $||\delta\mathbf{y}||$ represents the *content* or "volume" of the data quantization cell $\delta\mathbf{y}$. As is the usual practice we will omit $\delta\mathbf{y}$ from most formulas here because it makes no difference in the model estimation process. In some cases, however, this may produce formulas where it appears that which model is estimated might depend on the units in which the data $\mathbf{y}$ is represented. [Care should be taken in using this approximation however, because it doesn't always hold and its use may cause problems in such cases.] There is more than one way to use $P(\mathbf{y}|\theta^{(\xi)}, \xi)$ to encode $\mathbf{y}$. We survey these in the following subsection.

As a final point here, we should mention that, in cases where there are both dependent and independent variables $(\mathbf{x}, \mathbf{y})$, there is a question of whether or not $\mathbf{x}$ must be described (encoded) by our model. The answer is that our model should include an explicit description of $\mathbf{x}$ if there are some important model parameters associated with their distribution that we need to estimate. For example, in our image-segmentation example, if the grey-levels correspond to the pixels of a digital image, which are on a regular rectangular or hexagonal grid, it is reasonable to omit $\mathbf{x}$ from our *description*. They may, however, appear in our description-length formulas, as they do in our segmentation example.

In the sequel we refer to the processes of *encoding* and *decoding* data throughout the discussion. We will also use associated language, speaking of things like "the decoder", "the encoder" and "the code book". This is clearly unnecessary for the formal development of MDL, but we have found it to be of considerable pedagogical value.

2.3. Background: Ways of Using Models to Encode Data

We know from Shannon[54] that for *symbols* $\{s\}$ "emitted" independently from a *source* S according to an emission probability $P_S(s)$ that the shortest encoding of these symbols that can be achieved is one for which the code lengths are given by $-\log P_S(s)$ and the average code length for such an encoding scheme is given by the *entropy* of the source:

$$E_S\{L(s)\} \;=\; H(S) \;\equiv\; -\sum_s P_S(s)\log P_S(s). \qquad (6)$$

This theorem forms the basis for most model-based data encoding/compression schemes. For the purposes of this discussion we will divide these schemes into three types: (1) one-part, (2) two-part and (3) predictive. In most cases in what follows we consider the total collection of data (I, $\mathbf{y}$ or $(\mathbf{x},\mathbf{y})$ depending on the problem) to be a single symbol in this regard, though we may treat the individual y_i's as symbols. [Note that for a simple one-code-word-per-symbol scheme each symbol's code word must be of integer length, but that $\log P_S(s)$ is not guaranteed to be an integer. In performing estimation of the type we do here, this difference is not important and we ignore it.]

In discussing MDL, we assume that the data $\mathbf{y}$ are discrete, quantized and that they are encoded as *binary prefix codes*, which can always be decoded with a binary decoder tree like that shown in Figure 2. [Prefix codes are those where the code word for one symbol never appears as a prefix in another symbol's code word. Thus the code words can be strung together one after the other with no special separator symbols (e.g. commas ",") and can still be decoded.] While MDL can, in principle, be formulated without these assumptions, they are the easiest to work with and are usually true in practice.

In the diagram of a hypothetical binary decoder tree in Figure 2 the internal nodes are single-bit tests and the bit they are testing can be determined by their level in the tree. The root node (top) tests the first bit in the string; its children test the 2nd bit and so on. The binary strings displayed at the leaves are codewords, which correspond to decoded symbols. The *prefix* property of the associated code is obvious.

The code length of any symbol is equal to the length of the path from the root node to its corresponding leaf. It is easy to see from this tree that something called the *Kraft inequality* (see [7, Section 5.5]) must hold. This says that for a prefix encoding scheme (code

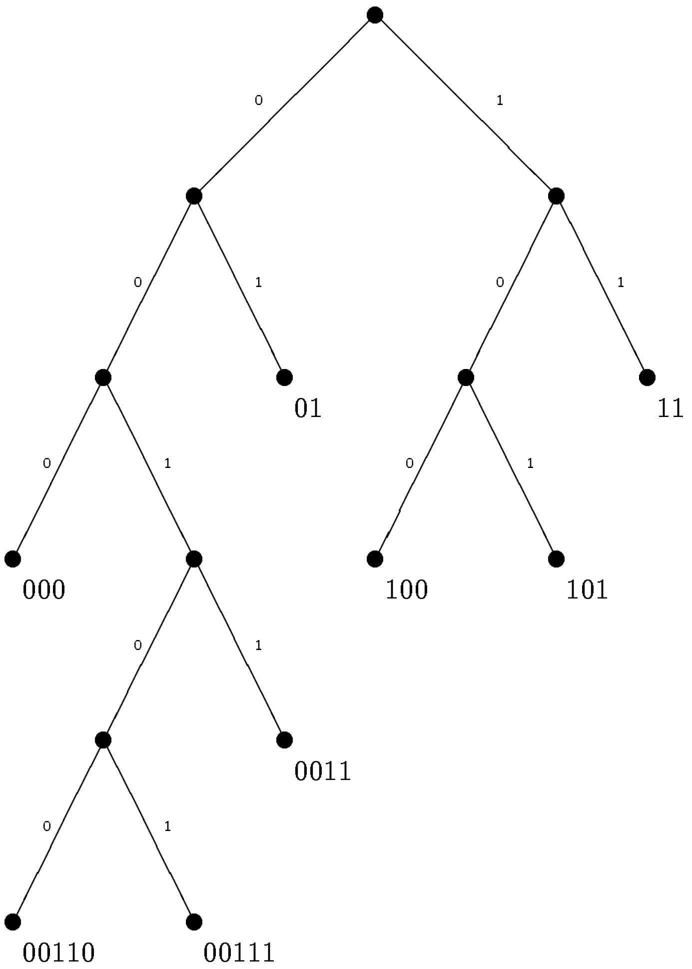

Figure 2: A sample decoder tree for a limited binary prefix code.

book) for a set of symbols $\{s_i\}$ with associated code lengths $\{L_i\}$, the following inequality holds.

$$\sum_{\text{all symbols } i} 2^{-L_i} \leq 1, \qquad (7)$$

Codes for which this holds with equality are called *complete*, whereas those where it holds with strictly "$<$" are referred to as *incomplete*. This will be discussed is more detail in the sequel. Complete codes correspond to *complete* decoder trees in which every node is either a leaf or an internal node with two children. We digress briefly here to discuss these cases.

2.3.1. Kraft's Inequality and Complete Codes It is easy to see that equation (7) holds with equality for complete codes by the following reasoning. Imagine that we start at the root node of our decoder tree with a unit mass. We send half of it down each branch (left, right) and at each node with two children, we repeat this process until our unit mass is distributed over the leaves of the tree. For a complete code every node is either a leaf or has two children and the mass at the leaf corresponding to symbol i is equal to 2^{-L_i}. Q.E.D.

For the case of incomplete codes the decoder tree will be incomplete which means that it will have internal nodes with only one child. In the preceeding reasoning about the distribution of a unit mass over the leaves of the tree, if a node is not a leaf, but has only one child, all of its mass goes down that path. The result is that every such node in the path to a leaf adds one to the code length without splitting the mass in half. Thus after distribution of the mass, the number 2^{-L_i} will be less than the mass at the corresponding leaf and equation (7) holds with "$<$". Q.E.D. again.

Due to Shannon's theorem and this result, we relate code lengths and probabilities by:

$$P(s_i) = 2^{-L_i}.$$

It is also clear from this that if $P(s_i)$ has parameters, their *maximum-likelihood* (ML) estimates will also be minimum-length estimates.

In the following subsections we first briefly review the various encoding approaches. Following that, we examine in detail certain issues related to MDL-based estimation.

2.3.2. One-Part Coding The simplest, most straightforward and most fundamental way of using a model to encode data is the case treated by Shannon - that where the model is known. In this case there is no estimation involved and we assume that the model is known by both the encoder and decoder, which use it to construct codebooks using Huffman coding or some other mutually agreed on algorithm. Of course, in practical applications of data compression this model doesn't have to be the "true" model that describes the distribution of the data. We will return to this issue after we have discussed other encoding methods.

2.3.3. Two-Part Coding In two-part coding there is no agreed upon model, but rather an agreed upon model class $\mathcal{M}$ from which the model to be used for encoding will be chosen. In this case the decoder must be informed by the encoder which model is being used so that the decoder can construct the correct code book. Thus the total prefix code for the data will consist of a preamble that encodes the model followed by the data encoded using the model. Thus the associated code length can be expressed as:

$$L_{2P}(\mathbf{y}) = L(\mathbf{y}|M) + L(M) \tag{8}$$
$$L_{2P}(\mathbf{y}) = -\log P(\mathbf{y}|M) + L(M), \tag{9}$$
$$L_{2P}(\mathbf{y}) = -\log P(\mathbf{y}|\theta^{(\xi)}, \xi) + L(\theta^{(\xi)}|\xi) + L(\xi) \tag{10}$$

The parameter code length $L(M)$ is frequently associated with the Bayesian notion of a *prior*[2] $\pi(M)$; the two being related by:

$$\pi(M) = 2^{-L(M)}, \tag{11}$$

which derives from the *Kraft inequality.* If the model class $\mathcal{M}$ is countable, this identification is reasonable. If it is not (because the models include real-valued parameters - the more usual case), on the other hand, the correspondence is less clear.

In the usual case, for a given ξ equation (10) is minimized by letting $\theta^{(\xi)} = \hat{\theta}^{(\xi)}$ - its *maximum-likelihood* (ML) estimate. This may constitute a slight approximation, which is justified by the fact that $\log P(\mathbf{y}|\theta^{(\xi)}, \xi)$ is a sharply peaked function of $\theta^{(\xi)}$ whereas $L(\theta^{(\xi)}|\xi)$ will usually be slowly varying. The question of which scheme to use for encoding $\hat{\theta}^{(\xi)}$ is important and involves subtle issues. These will be in subsequent subsections. We rewrite equation (10) here in its minimized (over ξ and $\theta^{(\xi)}$) form:

$$\hat{L}_{2P}(\mathbf{y}) = -\log P(\mathbf{y}|\hat{\theta}^{(\hat{\xi})}, \hat{\xi}) + L(\hat{\theta}^{(\hat{\xi})}|\hat{\xi}) + L(\hat{\xi}). \tag{12}$$

2.3.4. Predictive Coding While it can be used for other things, predictive coding is most applicable to cases like time series where the data have a natural ordering. Given a sequence $\mathbf{y} = \{y_t | t = 1, 2, 3 \ldots, n\}$, we define the *partial sequence* $y^t = \{y_1, y_2, \ldots, y_t\}$. Thinking of the individual elements y_t as symbols, the idea behind predictive coding is that a new code book is computed after each symbol is transmitted. This code book is based only on the statistics of the data y^t transmitted up to that point so that the encoder and the decoder are ensured of computing the same code book. Here, for a fixed model structure ξ, the code length for y_t is given approximately by:

$$L(y_t) = -\log P(y_t|\hat{\theta}^{(\xi)}(y^{t-1}), \xi). \tag{13}$$

Thus the total code length is given by:

$$L_p(\mathbf{y}) = L(\mathbf{y}|\xi) + L(\xi), \tag{14}$$

where:

$$L(\mathbf{y}|\xi) \;=\; \sum_{t=1}^{n} -\log P(y_t|\hat{\theta}^{(\xi)}(y^{t-1}), \xi) \tag{15}$$

Note that because the parameter values $\hat{\theta}$ don't have to be encoded explicitly, no associated code length term appears in these formulas.

Two practical problems that may arise when using predictive coding in MDL-based estimation are the following. First, some procedure is required for computing the estimates $\{\hat{\theta}^{(\xi)}(y^{t-1})\}$ for small t. If one has a "zero-order" encoding scheme (equivalently a prior $\pi(\theta^{(\xi)}|\xi)$) the *maximum a posteriori* (MAP) estimate can (should) be used:

$$\hat{\theta}^{(\xi)}(y^{t-1}) \;=\; \arg\max_{\theta^{(\xi)}}[P(\theta^{(\xi)}|y^t, \xi)] \;=\; \arg\max_{\theta^{(\xi)}}[P(y^t|\theta^{(\xi)}, \xi)\pi(\theta^{(\xi)}|\xi)]. \tag{16}$$

The second problem is that the results obtained will depend, in general, on the order in which the $\{y_t\}$ are analyzed. If n is sufficiently large, however, the results will be independent of this order.

2.4. Two-Part-Coding Refinements

Of the three coding approaches described in the previous section the most commonly used in MDL-based estimation is two-part. In this subsection we present certain refinements to the two-part scheme that improve its usability and improve the accuracy of the results obtained.

2.4.1. Two-Part Coding and Optimal Precision Rissanen[40] has derived what is now considered the "classical" form for the real-valued-parameter ($\theta^{(\xi)}$) code length $L(\theta^{(\xi)}|\xi)$ based on an asymptotic optimal-precision analysis of the parameter encoding. That result is:

$$L(\hat{\theta}^{(\xi)}|\xi) \;\sim\; \frac{k(\xi)}{2}\log n, \tag{17}$$

where $k(\xi)$ is the number of freely adjustable parameters in $\theta^{(\xi)}$. This length function holds only for the *maximum-likelihood* estimate $\hat{\theta}^{(\xi)}$ of $\theta^{(\xi)}$. In cases where there really is a prior distribution $\pi(\theta^{(\xi)}|\xi)$ it should be incorporated in $L(\theta^{(\xi)}|\xi)$ also. For this case Rissanen's analysis yields simply

$$L(\hat{\theta}^{(\xi)}|\xi) \;\sim\; -\log\pi(\hat{\theta}^{(\xi)}|\xi) \;+\; \frac{k(\xi)}{2}\log n. \tag{18}$$

Equations (17 and 18) only hold for large n, but have frequently been used even for small n due to the lack of a computationally tractable alternative. For some problems such an alternative exists. These will be discussed below.

Here we review briefly the derivation of equation (17). In two-part coding (equation (10)) the real-valued $\hat{\theta}^{(\xi)}$ must be encoded to some finite precision. Denote this precision by d and the finite-precision (quantized) parameter values by $\theta_d^{(\xi)}$. Then we might write:

$$L_{2P}(\mathbf{y}; d) \;\; = \;\; -\log P(\mathbf{y}|\theta_d^{(\xi)}) \; + \; L(\theta_d^{(\xi)}|\xi) \; + \; L(\xi) \; + \; L(d). \tag{19}$$

The precision d may be included among the variables over which the code length (equation (19) is minimized for a given $\mathbf{y}$. Rissanen does this in [40] and obtains the following asymptotic expression for the minimized code length:

$$\hat{L}(\mathbf{y}) \sim -\log P(\mathbf{y}|\hat{\theta}^{(\xi)}) \; + \; \frac{k(\xi)}{2}\log n \; + \; L(\xi), \tag{20}$$

which has been used extensively. In certain problems, however, one may be forced to deal with very small values of n and this can definitely cause problems.

An example of a case where this issue (asymptotic code-length formulas) becomes important is the region-merging-based segmentation algorithm of Kanungo et.al.[23, 24]. This algorithm performs a type of piecewise surface fitting where an image is divided into regions and a two-dimensional polynomial is fit to each region, just as in the example we gave earlier. The regions in this image-segmentation example are an example of *equivalence classes.* In such problems there will be a term like equation (20) for each equivalence class, plus a term for the encoding length for the description of the partitioning of the data I into these classes (e.g. the image partitioning in segmentation). The individual regions may be quite small (even one pixel) and accurate code length formulas become important not just for their utility in estimating the correct polynomial order, but also in making decisions like whether or not to merge adjacent regions. In that algorithm an *ad hoc* scheme was used to deal with these small regions, while $(k/2)\log n$ was used for the parameter encoding cost of larger regions.

2.4.2. Obtaining Non-Asymptotic Code-Length Formulas in Two-Part Coding

In Section 2.3.1 we explained the issue of *completeness* in coding. Here we examine this issue for two-part coding as we described it in Section 2.3.3. As we will explain, these two-part codes are incomplete and modifying them so that they are complete results in formulas that are *sharper* (less asymptotic) than equation (20) and in some cases completely non-asymptotic.

That two-part codes as we have thus-far described them are incomplete can be seen by the following reasoning. In a real-world coding scheme, if the decoder knows the procedure that the encoder will use to compute the estimates $(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$ from the data, it can use this information to eliminate values of $\mathbf{y}$. This is because it knows that a limited set of $\mathbf{y}$ values could have led to the estimates $(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$. This set of values is the "inverse image" in $\mathbb{R}^n$ of $(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$, which we represent by $\Omega(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$. In principle, the decoder can determine what these possible values are. For this reason the two-part scheme represented by (10) allows for possibilities that can never exist. That is, in the usual two-part coding, even though the encoder has encoded $(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$ and transmitted it to the decoder, it uses codes for $\mathbf{y}$ that assume that any value of $\mathbf{y}$ can occur in combination with $(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$, which obviously isn't the case. As a result, the probability $P(\mathbf{y}|\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$ is normalized to sum to 1 when summed over all $\mathbf{y}$ values and the codebook associated with $(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$ would contain codes for all $\mathbf{y}$ values.

If we were to construct a decoder tree for this entire encoding scheme, a certain subtree (corresponding to the first part of the code words) would carry out the task of determining the model. Thinking of that subtree as a decoder tree, its leaves correspond to models. For each model, there is a separate decoder tree that we attach to the corresponding leaf to get the total decoder tree. Each of these attached decoder trees has leaves corresponding to every possible value of $\mathbf{y}$. Now if we prune from this tree all branches that can never be taken, we clearly have an incomplete tree (internal nodes with only one child), which means that the associated code is also incomplete by definition.

2.4.3. Making Two-Part Codes Complete A complete code corresponding to the incomplete one described in Section 2.3.3 can be constructed by renormalizing $P(\mathbf{y}|\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$ to sum to 1 when summed over only those $\mathbf{y}$ values in $\Omega(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$. Thus $-\log P(\mathbf{y}|\hat{\theta}^{(\hat{\xi})}, \hat{\xi})$ in equation (12) can be replaced by:

$$L_c(\mathbf{y}|\hat{\theta}^{(\hat{\xi})}, \hat{\xi}) \;=\; -\log\left[\frac{P(\mathbf{y}|\hat{\theta}^{(\hat{\xi})}, \hat{\xi})}{\sum_{\{\mathbf{w}\in\Omega(\hat{\theta}^{(\hat{\xi})}, \hat{\xi})\}} P(\mathbf{w}|\hat{\theta}^{(\hat{\xi})}, \hat{\xi})}\right], \tag{21}$$

where the subscript "c" on L_c in equation (21) stands for "complete".

If we have an encoding scheme for the parameters $\theta^{(\xi)}$ with an associated length function $L(\theta^{(\xi)}, \xi)$, we can write our total code length as:

$$L_c(\mathbf{y}) \;=\; L_c(\mathbf{y}|\hat{\theta}^{(\hat{\xi})}, \hat{\xi}) \;+\; L(\hat{\theta}^{(\hat{\xi})}, \hat{\xi}). \tag{22}$$

It is interesting to note that:

$$\sum_{\mathbf{y}\in\Lambda} 2^{-L_c(\mathbf{y})} \;=\; 1, \tag{23}$$

[where Λ is the set of all possible $\mathbf{y}$ values] in this case and as a result, we can think of two-part coding with *incompleteness* removal as simply one-part (in the spirit of Shannon) coding with a somewhat complicated probability distribution. In this view the parameter estimation (i.e. in $\hat{\theta}(\mathbf{y})$) can be considered to be "under the covers". [This is not strictly true the way we treat things here because the incompleteness associated with ξ has not been removed. In principle this could be handled in the same way as $\hat{\theta}^{(\hat{\xi})}$, but in practice that would be very difficult.]

2.4.4. Using a Maximum-Likelihood Code to Make Two-Part Codes Complete

In [43] Rissanen proposes addressing the *completeness* issue using a scheme based on the *maximum-likelihood* (ML) probability distribution $\hat{P}(\mathbf{y}|\xi)$:

$$\hat{P}(\mathbf{y}|\xi) \equiv \frac{P(\mathbf{y}|\hat{\theta}^{(\xi)}(\mathbf{y}),\xi)}{\sum_{\{\mathbf{w}\in\Lambda\}} P(\mathbf{w}|\hat{\theta}^{(\xi)}(\mathbf{w}),\xi)}, \tag{24}$$

where, the sum in the denominator is over all possible data vectors. It should be clear that there is one such ML distribution for each model class and structure, ξ. [In the rest of this section, for notational simplicity, we will omit the explicit dependence of various quantities on ξ and $k(\xi)$.]

In [43] Rissanen points out that using this form (equation (24)) is equivalent to *incompleteness* removal of the type of equation (22) with $L(\hat{\theta}^{(\hat{\xi})},\hat{\xi}) = \log \pi_n(\hat{\theta})$ where:

$$\pi_n(\theta) = \frac{\sum_{\{\mathbf{y}|\hat{\theta}(\mathbf{y})=\theta\}} P(\mathbf{y}|\theta)}{\sum_{\{\mathbf{w}\in\Lambda\}} P(\mathbf{w}|\hat{\theta}(\mathbf{w}))}. \tag{25}$$

Using this expression for π, leads to the following approximation for (24) in terms of $\hat{f}$ (referred to as the *maximum-likelihood density*):

$$\hat{P}(\mathbf{y}) \approx \hat{f}(\mathbf{y})\cdot\|\delta\mathbf{y}\| = \left[\frac{f(\mathbf{y}|\hat{\theta}(\mathbf{y}))}{\int_{\{\mathbf{w}\in\Lambda\}} f(\mathbf{w}|\hat{\theta}(\mathbf{w}))d\mathbf{w}}\right]\cdot\|\delta\mathbf{y}\|. \tag{26}$$

[A technical point: We have used Λ to represent both the region of integration (set of possible $\mathbf{w}$ values) for this integral and to represent the set of values over which the sum of equation (24) is taken. What we mean by Λ in the case of the integral is the obvious continuous region in $\mathbb{R}^n$ (implied by this discrete-to-continuous approximation) containing the discrete set Λ.]

For either of these (equations (24) and (26)) 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 [43] to derive the following approximate expression for the logarithm of the maximum-likelihood density (equation (26)):

$$-\log\hat{f}(\mathbf{y}) = -\log f(\mathbf{y}|\hat{\theta}(\mathbf{y})) + \frac{k}{2}\log\frac{n}{2\pi} + \log\int_\Gamma \sqrt{|I_f(\theta)|}d\theta + o(1), \tag{27}$$

where

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

$$-E_\theta\left\{\frac{\partial \log f(y|\theta)}{\partial\theta_i\partial\theta_j}\right\}, \tag{28}$$

  where $E_\theta\{\ldots\}$ denotes $\int\{\ldots\}f(y|\theta)d\theta$.

- $\Gamma \equiv \Gamma(\xi) \subset \mathbb{R}^{k(\xi)}$ is the region of allowable θ values. Selection of Γ may be straightforward, being determined by the problem. It may (for example) be all of $\mathbb{R}^k(\xi)$. In some cases, however, the integral $\int_\Gamma \sqrt{|I_f(\theta)|}d\theta$ may diverge when using the desired Γ. In such cases selection of an alternate Γ may be a delicate issue (refer to [43]).

This result, while still asymptotic, is much sharper than the previous ($\frac{k}{2}\log n$) form.

Summarizing Rissanen's ML code-length functions, we are left with the possibility of using $L(\xi)$ plus one of: the log of (24), the log of (26), or (27), as the quantity to be minimized in MDL-based estimation, depending on the situation. Alternately, if we have another encoding scheme for the parameters, we can use it as indicated in equation (22).

2.4.5. Recent Results in Using Complete Two-Part Codes for MDL-Based Estimation We have recently derived non-asymptotic code-length formulas for MDL-based estimation in linear regression, a special case of which is the polynomial surface fitting required to effect the image segmentation algorithm mentioned as an example in Section 1.3 and used in [23, 24]. We derived formulas based on an ML code and based on a uniform parameter encoding and compared them experimentally with the classic $(k/2)\log n$ form and with each other. This work is documented in [11]. The corresponding formulas are:

- Formula derived using an ML code:

$$L_{\mathrm{ML}}(\mathbf{y}) \;=\; -\log \hat{f}(\mathbf{y}) \;=\; n\log\hat{\sigma}(\mathbf{y}) \;+\; \log\left\{\frac{4\left(\frac{n}{2\pi e}\right)^{\frac{n}{2}}\left[\left(\frac{\mathsf{d}}{2\hat{\sigma}_0}\right)^{m} - 1\right]}{m^2\Gamma(\frac{m}{2})\Gamma((n-m)/2)}\right\} \tag{29}$$

- Formula derived using a uniform parameter encoding (UPE):

$$L_{\mathrm{UPE}}(\mathbf{y}) \;=\; (n-m-1)\log\hat{\sigma}(\mathbf{y}) \;+\; \log\left\{\frac{4\pi^{n/2}n^{n/2}(\mathsf{d}/2)^m[(\mathsf{d}/2)-\hat{\sigma}_0]}{m\Gamma((n-m)/2)\Gamma(\frac{m}{2})}\right\} \tag{30}$$

In these formulas $\hat{\sigma}(\mathbf{y})$ is the measured standard deviation of the fit, e is the base of the natural logarithms, m is the number of regression functions (model complexity in this case), d is the range of possible data values (e.g. $\mathsf{d} = 256$ for 8-bit grey-scale images), $\Gamma(\ldots)$ is the "Gamma" function, and $\hat{\sigma}_0$ is a parameter whose value can, in principle, be derived. A rationalization (see [11]) can be suggested for setting $\hat{\sigma}_0$ equal to the data quantization (e.g. 1 for integer grey-levels) and this was done in the simulation experiments in [11].

In those experiments both of these formulas gave results significantly superior to those produced using $(k/2)\log n$ and the UPE formula performed better overall than the ML formula, though the ML formula was superior in some cases.

In other recent related work[12] we applied these methods to the problem of segmenting binary strings using Bernoulli models. In that study the same ordering was observed among the results: both methods (ML and UPE) of incompleteness were used and both performed significantly better than $(k/2)\log n$. Also, the UPE form performed slightly better than the ML form.

2.5. MDL in Practice: How Strictly Should One Adhere to the Coding Paradigm?

How important is it that one adhere strictly to the coding paradigm on which MDL is based? In the extreme case, one would make sure that all the coding formulas used agree strictly with a realizable encoding of the image. The major problems with this are: (1) the computational burden it implies and (2) many practical, realizable encoding schemes have artifacts that arise due to practical considerations. For example, it is possible to encode images using a scheme based on quad trees, which have certain properties that make them desirable. The associated code lengths will depend heavily, however, on how well region boundaries happen to follow the quad tree boundaries. Only in very pathological cases will such boundaries have any relationship to the underlying physical scene. We are, after all, really concerned with estimation and only with coding at a certain abstract theoretical level rather than in practical terms.

On the other hand, awareness of these facts may create certain pitfalls, in that it can cause one to develop a somewhat cavalier attitude toward the coding paradigm. At this other extreme, it is possible to get so careless with the formalities of coding that important regularizing aspects of it are lost (a case of "throwing out the baby [at least in part] with the bath water").

3. Other Approaches to Estimation Problems Addressed by MDL

In this section we briefly review alternate (to MDL) approaches to solving ill-posed problems. There is some overlap among these.

3.1. Relation to Bayesian Estimation

Bayesian estimation[2] is based largely on the central concept of a *prior distribution* or just "prior". We alluded to this briefly in previous sections. The concept here (and this is where it differs from classical statistics) is to treat the model parameters as random variables with their own distribution (the prior), which is usually represented by "π".
The Maximum *A Posteriori* (MAP) Estimate: The most common Bayesian estimate is the maximum *a posteriori* (MAP) estimate, defined by:

$$\hat{M} = \arg\max_{M \in \mathcal{M}} [P(M|\mathbf{y})] \quad (31)$$

$$= \arg\max_{M \in \mathcal{M}} [P(\mathbf{y}|M)\pi(M)], \quad (32)$$

where the second equation is obtained from the first by applying Bayes rule and dropping $P(\mathbf{y})$ because it is the same for all M.

If the model class is discrete (countable) this may be seen as equivalent to the MDL estimate based on two-part coding by simply identifying the model code length with the prior:

$$L(M) = -\log \pi(M). \quad (33)$$

Other MDL estimation formulas do not have Bayesian counterparts however, especially when real-valued parameters are required to specify a model instance. For example, those obtained by predictive coding, optimal precision analysis and two-part codes rendered complete by the methods described above.

Bayesian Mixtures: In Bayesian statistics one can obtain an estimate of the actual probability $P(\mathbf{y})$ by taking a kind of weighted average over all models. This average, which is sometimes referred to as a "Bayesian mixture", is really just the marginal distribution $P(\mathbf{y})$ obtained by summing the Bayesian joint probability $P(\mathbf{y}, M)$ over all the models in $\mathcal{M}$:

$$P(\mathbf{y}) = \sum_{M \in \mathcal{M}} [P(\mathbf{y}|M)\pi(M)]. \tag{34}$$

This distribution can be shown to have certain optimality properties.

If we divide the model specification M into $\theta^{(\xi)}$ and ξ, we can apply this Bayesian-mixture procedure partially, to just $\theta^{(\xi)}$. We can rewrite our prior $\pi(M) = \pi(\theta^{(\xi)}|\xi)\pi(\xi)$ and then integrate over $\theta^{(\xi)}$ as follows:

$$P(\mathbf{y}|\xi) = \int_{\Gamma(\xi)} P(\mathbf{y}|\theta^{(\xi)}, \xi)\pi(\theta^{(\xi)}|\xi)d\theta^{(\xi)}. \tag{35}$$

In fact we can, in principle, integrate over only some of the components of $\theta^{(\xi)}$ and some of the components of ξ. For example, in our segmentation example, we could in theory integrate over $\theta^{(\xi)}$ and sum over all possible polynomial degrees, leaving only the description of the region boundaries to estimate, if this was our only interest. The important point is that summing or integrating over all values of a certain parameter removes the ability to estimate a specific value of that parameter.

Such mixtures/marginals can be used in a kind of supervised learning sometimes referred to as "Bayesian learning". As an example, let $\mathbf{y}$ represent a set of training vectors from which we hope to estimate the probability density $P(y|\xi)$, where, here, y represents a new (not from the training data) vector. Using these, the density we wish to construct is $P(y|\mathbf{y}, \xi)$. Note that this can be written as:

$$P(y|\mathbf{y}, \xi) = \frac{P(y, \mathbf{y}|\xi)}{P(\mathbf{y}|\xi)}, \tag{36}$$

where we can substitute integrals of the form of equation (35) for $P(\mathbf{y}|\xi)$ and $P(y, \mathbf{y}|\xi)$. With these substitutions, equation (36) can also be manipulated into the following form:

$$P(y|\mathbf{y}, \xi) = \int_{\Gamma(\xi)} P(y|\theta^{(\xi)}, \xi)P(\theta^{(\xi)}|\mathbf{y}, \xi)d\theta^{(\xi)} \tag{37}$$

where

$$P(\theta^{(\xi)}|\mathbf{y}, \xi) = \frac{P(\theta^{(\xi)}, \mathbf{y}|\xi)}{P(\mathbf{y}|\xi)} = \frac{P(\mathbf{y}|\theta^{(\xi)}, \xi)\,\pi(\theta^{(\xi)}|\xi)}{P(\mathbf{y}|\xi)}.$$

3.2. Algorithmic Complexity

We mentioned algorithmic complexity in our brief history of MDL. The key notion is that the complexity of a "string" (could be a set of data values given to a certain precision) is equal to the length of the shortest computer program that generates the string. For long sequences the bulk of this program will consist of the very code whose length is the minimum description length. Thus the minimum description length and the algorithmic (Kolmogorov) complexity are asymptotically equal. In our opinion what Rissanen and others have done in creating the MDL principle is to take (and develop) from algorithmic complexity that part that is truly relevant to doing model selection/estimation, eliminating in the process the cumbersome details of specific computer languages, etc. We are over simplifying somewhat and algorithmic complexity (algorithmic information theory) is concerned with more than just model selection. We refer the reader to the interesting discussion in [7, Chapt. 7, "Kolmogorov Complexity"] and to two books by Chaitin[5, 6].

3.3. Regularization Theory

In its broadest use the term "regularization" refers to the process of making ill-posed problems well-posed, by applying some further constraint(s) not present in the original problem statement. By this definition, all the techniques mentioned here including MDL are regularization methods. There is, however, a formal mathematical theory that studies various regularization methods. The genesis of this theory is usually attributed to Tikhonov[61] and is reviewed in another chapter[18] of this volume.

The following is a simple example that captures the essence of how "regularization" has been applied in computer vision problems. Suppose we want to do curve fitting with order determination as was investigated in [11]. Here we express the least-squares fitting in the continuous domain. Suppose we have an observed function $y(x), x \in [a, b]$ to which we wish to fit a parametric function $g(x; \theta^{(\xi)}, \xi)$. In the case of polynomial curve fitting ξ would correspond to the polynomial degree and $\theta^{(\xi)}$ to the coefficients. Our (unregularized) least-squares objective function is:

$$\Phi(\theta^{(\xi)}, \xi; \{y(x)\}) \quad = \quad \int_a^b [g(x; \theta^{(\xi)}, \xi) - y(x)]^2 dx. \tag{38}$$

This is clearly best satisfied by making $g(x; \theta^{(\xi)}, \xi)$ as close to $y(x)$ as is possible. In the usual polynomial curve-fitting problem $y(x)$ is really a set of discrete points and we are free to let $g(x; \theta^{(\xi)}, \xi)$ be the Lagrange interpolating polynomial, as mentioned previously. The "regularization" would be applied in this case would probably be to add a smoothness constraint or smoothness penalty term. For example:

$$\Phi(\theta^{(\xi)}, \xi; \{y(x)\}) \quad = \quad \int_a^b [g(x; \theta^{(\xi)}, \xi) - y(x)]^2 dx \; + \; \alpha \int_a^b \left[\frac{dg(x; \theta^{(\xi)}, \xi)}{dx} \right]^2 dx, \tag{39}$$

where α is a parameter that is adjusted to give the most pleasing result.

Such approaches may work well for certain problems but have two disadvantages relative to MDL: While the choice of the regularizing term/constraint/functional will presumably be made on some meaningful basis, it is essentially an *ad hoc* process. Secondly, one is left with the question of what value to choose for α, while MDL is free of such parameters.

3.4. Resampling Schemes for Prediction Error Estimation

Two related sampling schemes have to do with doing statistical estimation directly from training data with as few theoretical assumptions as possible. These are "cross validation" and "the bootstrap". They are similar in spirit and can be used to make various statistical estimates. The only such estimate that will concern us here is that of prediction error. We have estimated (fit) some model to our data $(\mathbf{x}, \mathbf{y})$ and we wish to estimate the prediction error that will be observed for new data drawn from the same population but not used in the estimation process. It is clear that, if we had a reliable estimate of this prediction error, it would be an ideal thing to use in determining the correct model. These error estimates would be used in exactly the same way we used the MDL code-length-based objective function.

We don't have such "new" data, however. This is especially true in the small-sample-size case in which we need to use all of our data for estimating model parameters. These schemes use certain strategies for resampling the original data. They are described and compared in [15] and [16] and more recently they were compared estimating the classification accuracy in an experimental study[28]. We review them only very briefly here.

3.4.1. Cross Validation If one has a truly large amount of training data, it may be feasible to divide the data into two sets; call them "training" and "test". A model is selected/estimated using the training set and then the prediction error is measured for the test set. If there is a large number of data points relative to the number of model parameters, it may be possible to do good estimation of both the model and the prediction error this way. In the more general case (and especially with small sample sizes), however, it is desirable to use as much of the data as possible to do model estimation. In leave-one-out cross validation one removes one training sample from the training set, estimates the model parameters based on the remaining $(n-1)$ training vectors, and then uses the so-obtained model to predict the withheld value. This is repeated, withholding all n training-data points, one at a time, and the prediction error estimate is the average of the n associated errors. In general cross validation the points are withheld k at a time with an associated trade off between computation time and estimation accuracy.

3.4.2. The Bootstrap Conceptually, the Bootstrap is based on the *empirical distribution function* $\tilde{P}$ defined as follows. Given a set A (e.g. some region of the data space - the space corresponding to the individual data values (x, y)), the probability associated with A

is equal to the number of data points lying in it. We may associate a density $\tilde{f}$ for data vectors y with this as follows:

$$\tilde{f}(y) \;=\; \sum_{i=1}^{n} \delta(y - y_i), \tag{40}$$

where $\delta(y)$ is an impulse ("Dirac delta") function. Using this, we have:

$$\tilde{P}(A) \;=\; \int_{\{y \in A\}} \tilde{f}(y) dy. \tag{41}$$

Bootstrap samples are formed by sampling the original data randomly n times with replacement so that some of the original data points may appear more than once and others may not appear at all. These are the samples one would obtain using the empirical density $\tilde{f}$ as the basis for a random number generator. Each of these Bootstrap samples is then used to estimate the model parameters. The so-obtained models are then used to predict values for the original data set $\mathbf{y}$. This is repeated some number of times and the average is taken as the prediction-error estimate. There are also a number of refinements (described in [15]) to this simple approach. Not surprisingly, because many of the same data points are used both in training and error assessment in each Bootstrap sample, the technique gives overly optimistic estimates of prediction error. This is a disadvantage it has relative to cross validation. On the other hand, it has lower variability than cross validation. See [15, Chapt.17] for a discussion of these points. It is worth mentioning that the fact that the Bootstrap error estimates have this downward bias doesn't, by itself, make it bad for our estimation purposes. The important point it whether or not the minimum occurs at the "true" model structure.

3.4.3. Comparison with MDL There is a certain philosophical appeal to these techniques because they appear to get directly at what is clearly a very important estimation criterion; in fact, the most important in many applications. These schemes have two disadvantages compared to MDL and Bayesian methods however:

- They are computationally quite expensive, requiring multiple passes through the data.
- They can be shown to be *inconsistent* in the sense that they will tend to choose a model complexity that is too high as $n \to \infty$ (See [41]). This was observed experimentally in [10, 55].

The question of these how these techniques compare with MDL in terms of the accuracy of the model-selection results has not, to our knowledge, been answered either theoretically or through a comprehensive experimental study. Cross validation was compared experimentally with an MDL-based feature-selection criterion and found to give inferior model selection results in [10, 55].

3.5. Other Selection Criteria

A number of model-selection criteria have been proposed over the years, most of which are *ad hoc*. At least two have a probabilistic foundation and have attained some stature. These are Akaike's Information Criterion (AIC)[1] and Schwartz's "Bayesian Information Criterion" (BIC)[49]. These are summarized in [41] and [15] and we refer the reader to these summaries and the original papers for further information.

4. Statistical Classification and Machine Learning

Some of the most straightforward applications of MDL are in areas closely related to statistical pattern recognition and machine learning. Here we will summarize three:

- supervised learning (e.g. classifier construction)
- unsupervised learning (e.g. clustering)
- feature selection and dimensionality determination

In the discussion we treat these as three separate problems, but any two or all three may appear together in the same problem.

4.1. Supervised Learning

In supervised learning one is usually given data (referred to as "training" data) consisting of values of an independent variable or variables x and similar values of a dependent variable y. It is this provision of the dependent-variable values that constitutes "supervision" and (as will be explained below) differentiates this type of learning from the *unsupervised* variety. There is assumed to be some partially deterministic functional relationship between the two and we wish to construct an algorithm that will predict the value of y for new (not from the training data) values of x. The curve-fitting problem we gave as an example early in this paper can be considered a simple example of a supervised learning problem. It might be made more complicated by increasing the dimensionality of the variables x and y. Further complication would be added by making the model *piecewise* polynomial as in the image segmentation example. The relationship we wish to learn may be either completely deterministic or partially stochastic, as in the curve fitting problem. The latter is the more usual case in practical problems and the place where MDL is the most useful.

Perhaps the most common example of supervised learning is statistical classification. A common example of which is optical character recognition (OCR). In this case the dependent variable can usually take one of a finite number of values corresponding to the "classes": for example the symbols (e.g. letters, numbers and punctuation) in OCR. This is in contrast to the problem of *regression* (e.g. curve fitting) in which the dependent variable is assumed to be real or at least many valued. It is this supervised-classification problem with a finite number of classes that we will discuss here. The generalization to other problems (e.g. regression) should usually be straightforward.

In most pattern recognition problems the independent variable x is vector valued. In statistical pattern recognition parlance we say we wish to "construct a classifier" to "classify" (predict the value of y) the associated *object* (e.g. a two-dimensional pattern in OCR) given the "feature vector" x. The set of possible values of x is referred to as "feature space". [We will use this language in this discussion.] There is a close correspondence between *classifiers* and *models*. For example in the classic *Bayes* classifier the ideal classification accuracy is achieved by computing the conditional probability $P(c|x)$ for all the classes and assigning the object to the class for which $P(c|x)$ is largest. In the unlikely case that we actually know the "true" functional form of $P(c|x)$, we would simply use it. In the more usual situation we have some set of "models" from which we may choose one to estimate the "true" P. The most common example of this is the case where $P(x|c)$ is assumed to be Gaussian and the model is estimated by estimating the parameters of the Gaussian distribution - the *mean vector* and the *covariance matrix*.

A classifier may be implemented as a simple closed-form function or a complicated algorithm. Various types of classifiers are surveyed in books on *statistical pattern recognition*[16, 64] and *machine learning*[63, 35], two fields with a significant amount of overlap. We won't discuss all types here. In supervised learning MDL is useful primarily in cases where the classifier is of variable complexity and we wish to determine the optimum complexity. Such classifiers are usually referred to as "non-parametric"[16]. Two families of classifiers of this type that have shown a lot of promise and in which there has been a lot of interest in recent years are *tree classifiers* and *artificial neural networks*.

4.1.1. Tree Classifiers There have been many tree classifiers proposed. The best known are probably CART[3], ID3[38] or its descendant C4.5[37]. Tree classifiers constitute a kind of hierarchical decomposition of feature space, dividing it into a set of non-overlapping regions. The leaves of the tree are associated with classes. Once the tree has been constructed, an object is classified by applying a succession of tests to its *feature vector*. These tests correspond to the root and internal nodes of the tree and the branch taken after applying each test is determined by the test outcome. The process is repeated until a leaf is reached and the corresponding class is assigned to the object.

In most tree-classifier construction (learning) algorithms the tree is first "grown" to some size and then "pruned" back to some smaller size by removing branches. Both the growing and pruning process have an associated objective function and both are usually performed using a simple greedy scheme. It is in the selection/construction of an objective function for the pruning process that MDL is usually applied. As an example we will consider the binary trees grown by CART. In the most common version of the CART algorithm, during the classification process, the test at a given node is performed on a single *feature* (component of x). There is an associated threshold and if the feature value is less than or equal to the threshold, the left branch is taken; otherwise the right branch is taken. This process is repeated until a leaf is reached. Associated with each leaf is a class, which is assigned to the object being classified.

The process of growing the CART trees proceeds as follows. Starting at the root, the best feature and the best threshold for that feature are selected and the resulting test splits the training data into two sets. For each of these sets and their associated children (child nodes) the process is then repeated and so on recursively for all of its descendents until leaves are reached. In this case "best" refers to minimizing an "impurity" measure. In [3] two different impurity measures are used. One of these is a heuristic measure referred to as the "Gini criterion", while the other turns out to be the *mutual information*[7] between the class variable and the test outcome. A leaf is reached (in the learning process) when no further improvement in the impurity measure is possible (e.g. when the mutual information of the class variable and the test outcome is simply equal to entropy of the class variable).

MDL-based pruning attempts to minimize a two-part code length-based objective function of the form:

$$L(\mathbf{y}) \;=\; L(\mathbf{y}|T) \;+\; L(T), \tag{42}$$

where T represents the classification tree, $L(T)$ its encoding cost and $L(\mathbf{y}|T)$ the code length of the data using T in an encoding scheme. There is more than one way to construct encoding schemes and thus, more than one form for $L(T)$ and $L(\mathbf{y}|T)$. Every feature vector x will be assigned to a leaf of the tree and with that leaf is associated a class histogram $h(c)$ describing the training vectors that were assigned to the leaf. Let ℓ be an index identifying a leaf of the tree and $h_\ell(c)$ the associated class histogram. The maximum-likelihood (and therefore minimum code length) estimate of $P_\ell(c)$ is $h_\ell(c)/n_\ell$ where n_ℓ is the number of training vectors assigned to leaf ℓ. One way to encode the class values of the training data is to construct codes with lengths equal to $-\log \tilde{P}_\ell(c)$. We might also write $\tilde{P}_\ell$ as $\tilde{P}(c|x)$ because it is the feature vector x that tells the decoder (via the tree T) which node ℓ.

An alternate scheme, used in both [36] and [14] is to transmit an encoded version of a (hopefully) sparce binary vector indicating which training vectors were misclassified by the tree. That vector would then be followed by an encoded version of a list of the correct class codes for those misclassified vectors. The class labels for the correctly classified vectors don't need to be transmitted because the encoder has the classification tree and can use it to determine the correct labels. We will refer to this as the "exception" encoding scheme.

In addition to the encoding scheme for the data conditioned on T, to form the complete objective function, the issue of encoding T must be addressed. To our knowledge, the following encoding scheme for encoding CART trees has never been used or appeared in the literature. It seems, however, to be a straightforward modification of the encoding scheme used in [14] for encoding 2^n trees. We use it largely because it is (in our opinion) the easiest scheme to understand. For CART trees we would first encode the number of nodes ν in the tree, following that we encode the tree structure using a simple index. The latter requires $\log \beta(\nu)$ bits where $\beta(\nu)$ is the number of *ordered* trees with ν nodes. This is given by:

$$\beta(\nu) \;=\; \frac{1}{\nu+1}\binom{2\nu}{\nu},$$

[See [14] for a derivation of this. These numbers are generally known as *Catalan* numbers. See [19] for a discussion.] In addition to the tree structure, the class labels of the leaves and the feature/threshold combinations associated with the non-leaf nodes must be encoded. The class labels of the leaves may be simply encoded as a list of length $\nu_\ell \log K$, where ν_ℓ is the number of leaves. On the other hand, in the usual case where $\nu_\ell \gg K$, an alternate scheme will take fewer bits. Let q_i represent the number of leaves with class label i (thus $\sum_{i=1}^{K} q_i = \nu_\ell$), then we may first encode the set $\{q_i\}$, which can be done using $\log \binom{\nu_\ell + K - 1}{K-1}$, which is the logarithm of the number of sets of K non-negative integers that sum to ν_ℓ. Having encoded $\{q_i\}$, the specific leaf labeling consistent with it can be encoded using

$$\log \binom{\nu_\ell}{q_1, q_2, \ldots, q_K}$$

bits.

Finally, having encoded the tree structure and class labels of the leaves, we must encode the features and thresholds associated with all of the non-leaf (i.e. decision) nodes. The features can be encoded using $\log m$ bits each, where m is the number of features. The thresholds, on the other hand are real-valued parameters and can be encoded using (for example) $(1/2)\log n$ bits per threshold, where n is the number of training vectors. Note that choosing to encode the class labels of the leaves implies that the class values will be encoded using what we described as the "exception scheme". If we use the scheme that uses class codes with lengths equal to $\log \tilde{P}_\ell(c)$, then we must encode the class histogram $h_\ell(c)$ for each leaf.

The preceeding discussion of the process of constructing an MDL-based objective function for pruning CART trees has been somewhat superficial due to space limitations. For a more detailed treatment including derivations of the various combinatorial formulas involved consult [14].

Some other examples of using MDL in the construction of classification trees are the following. To our knowledge the first use was in the algorithm described in [45] and later in [62]. In this case an MDL-based objective function was used in growing the tree and no further pruning process was used. An improved version was described in [41]. An algorithm for pruning ID3[38] trees was described in [36] and later in [37]. This algorithm used the *exception* scheme we presented above for encoding the class labels, but a different scheme than the one we presented for encoding the classification trees. The 2^n classification trees described in [13] were created using an MDL-based pruning algorithm, later described in detail in [14].

The algorithm described in [34] also constructs CART-style classification trees using a slightly different objective function. This approach was designed to be scalable for use with very large data sets encountered in data mining applications.

4.1.2. Neural Networks Another family of classifiers often goes by the name "neural networks" or "connectionist classifiers". These terms appear to us to be used somewhat loosely, so here we will use the somewhat more precise (and narrow) term "multi-layer perceptrons" (MLPs) These classifiers follow (in a sense) the standard statistical pattern recognition paradigm of having one *discriminant function* for each class. This function is usually associated (sometimes loosely) with the conditional probability $P(c|x)$. For a given x, $P(c|x)$ is evaluated and the class assigned is given by:

$$\arg\max_c P(c|x).$$

In using MLPs for classification the values of the output nodes are usually treated as discriminant-function values. The original "Perceptron"[46, 47] was simply a linear classifier. That is, for feature vector $\mathbf{x}$ with components $\{x_i\}$, the class-c discriminant function (value of the output node) is given by:

$$\begin{aligned} d(c|x) &= \textstyle\sum_i w_{c,i} x_i \\ &= w_c \cdot x. \end{aligned}$$

What was different about the Perceptron was the sequential learning algorithm rather than the usual one-shot standard-linear-regression algorithm.

The Perceptron is usually given as the genesis of the Neural Network field. The so-called "multi-layer Perceptron" (MLP) consists of Perceptron-like single-layer networks placed in series, the most commonly use form being the two-layer network, which is said to have, in addition to inputs and output nodes, a single layer of "hidden" nodes. The outputs of the m^{th} layer $\{z_k^{(m)}\}$ are computed from those of the previous $(m-1)^{th}$ layer as follows:

$$z_k^{(m)} = f_k^{(m)} \left(\sum_i w_{c,i} z_i^{(m-1)} \right).$$

This single-layer function differs from the original Perceptron because it includes the "activation" functions $\{f_k^{(m)}\}$. These can be simply *identity* functions, *threshold* functions and other things. The most common form, however, is the *sigmoid* (a.k.a "soft threshold" function):

$$f(s) = \frac{1}{1 + \exp[-(s+\phi)/\theta]},$$

where $s = \sum_i w_i z_i$ (a weighted sum of the outputs from the previous layer). The parameters ϕ and θ can vary from layer to layer or even node to node. In the usual case, they are constant over the entire network.

Where does MDL come in? In using MLPs the question of how many layers and how many nodes per layer to use comes up. Also, even when the nodes-and-layers decisions have been made, one may choose to omit some of the inter-layer connections. These issues are all dimensions of the model-complexity question for MLPs. As an example consider the fully connected two-layer MLP mentioned above. Also, assume that the parameters ϕ and θ are fixed and the same for all nodes. Assume that the numbers of inputs and outputs are fixed, being determined by the sets of features and classes respectively. In this case the network-structure question reduces to the number of internal nodes. When the usual, sequential *back-propagation* algorithm is used for learning (estimating the weights), the natural form of MDL to use is the *predictive* version described in Section 2.3.4. The algorithm would consist of starting with some reasonable minimum number of internal nodes and perform back-propagation training, applying the network to each training example (to estimate the class probabilities $\{P(c|x)\}$) before the example is used to update the weights. The sum of the so-obtained $\log P(c|x)$ values is the total code length for that structure. This process is repeated, adding internal nodes one at a time, until the total code length begins to increase. This approach is discussed in detail in [42].

For general discussions of regularization techniques for selecting neural network structure see [32] and [17].

4.2. Unsupervised Learning; Clustering

In unsupervised learning there is no training data *per se*; there is just data. The learning algorithm attempts to discover structure in the data for some purpose. Most unsupervised learning problems may be thought of and frequently approached as probability *density estimation*. Given a collection $\mathbf{x}$ of data values x, it may be desired to obtain an estimate of the probability density $P(x)$ that is valid over some range of the possible x values. This is classic *density estimation*[50]. The term *clustering*[22] (a.k.a. "unsupervised classification") refers to the process of grouping the x values composing $\mathbf{x}$ into "clusters" based on some grouping criteria, usually a similarity measure. Perhaps the most common is simply Euclidean distance and the most commonly used clustering algorithm is probably "k-means"[see [22]], which groups the points into k clusters based on Euclidean distance. The following example clustering problem combines clustering, density estimation and the use of MDL.

Assume that $x \in \mathbb{R}^m$ and that there are n such values in $\mathbf{x}$. Let K denote the number of clusters and let k represent the number of freely adjustable parameters. We will assume that $P(x)$ is a *Gaussian mixture*:

$$P(x) \quad = \quad \sum_{j=1}^{K} \frac{\alpha_j}{(2\pi)^{m/2}|\mathbf{\Sigma}_j|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_j)^T \mathbf{\Sigma}_j^{-1}(x-\mu_j)\right], \tag{43}$$

The coefficients $\{\alpha_j\}$ are known as "mixing coefficients". For fixed K, maximum-likelihood estimates for the model parameters $\{\alpha_j\}$, $\{\mu_j\}$ and $\{\mathbf{\Sigma}_j\}$ are sought. The most common approach to performing this estimation for mixture models (especially Gaussian mixtures) is to use the EM algorithm[9] [See also [33].].

Having obtained this density estimate, it can be used to cluster $\mathbf{x}$ as follows. Think of each mixture component as corresponding to a class in a supervised classification problem. The individual Gaussians are associated with $P(x|c)$ (here substitute the mixture component j for c) and the mixing coefficients $\{\alpha\}$ are associated with the *a priori* class probabilities $P(c)$, $(c = j)$. Having made this association, the points are assigned to clusters j for which $P(j|x)$ is maximum. This was referred to as the *Bayes classifier* under "Supervised Learning" (Section 4.1) above.

The application for MDL in this problem is in determining the number of clusters K. We may associate K with the symbol ξ used in preceeding sections. The real-valued parameters $\theta^{(K)}$ consist of the sets:

$$\{\alpha_j, \mu_j, \mathbf{\Sigma}_j, j = 1, 2, \ldots, K\}$$

and their numbers are K for the α_j's, mK for the mean vectors $\{\mu_j\}$ and $Km(m+1)/2$ for the unique elements of the covariance matrices $\{\mathbf{\Sigma}_j\}$, for a total of:

$$k(K) \quad = \quad K\left(\frac{m^2+3m+2}{2}\right). \tag{44}$$

The code length of the data plus model in a two-part scheme can be used as an objective function whose minimum over K gives the optimum value of K - the number of clusters. Using the optimal-precision-based parameter code length described in section 2.4.1, we have:

$$L(M_K) \quad = \quad \frac{k(K)}{2}\log n,$$

where we have assumed that $L(K)$ is constant for all values of K. [In principle, as discussed in Section 2.4.4, a sharper formula can be obtained. But this would require solving the problem of obtaining, for example, a closed-form expression for correction term in equation (27):

$$\log \int_\Gamma \sqrt{|I_f(\theta)|}d\theta,$$

or an expression for the exact form in equation (26).]

The data code length is given by:

$$L(\mathbf{x}|M_K) \quad = \quad -\sum_{i=1}^{n} \log P(x_i|M_K), \tag{45}$$

where $P(x_i|M_K)$ is given by equation (43).

The clustering algorithm consists of first performing density estimation by fitting the mixture model of equation (43) to $\mathbf{x}$ (e.g. by using the EM algorithm) for a range of K values deemed reasonable and choosing the model (K value) that minimizes $L(\mathbf{x}|M_K) + L(M_K)$. Having obtained this mixture model, it is then used to classify the points composing $\mathbf{x}$ in the manner described above.

For other approaches to using MDL in clustering see (for example) [53] and [44].

4.3. Feature Selection and Dimensionality Determination

4.3.1. Feature Selection In supervised learning/classification there are two closely related problems often carried out together. These are the problems of *feature selection* and *dimensionality determination*. The term "feature selection" is often used in a way that includes dimensionality determination, but we will differentiate these as follows. In feature selection (a.k.a. "variable selection") one has available some number N of features from which a size-m ($m < N$) subset is to be selected for some purpose such as classification or regression. We assume classification. The goal is to select the best subset based on some *cost function*. For this we will assume that the goal is to minimize classification error. For the purposes of discussion let us assume that the model class to be considered is *multi-variate Gaussian*. There are $\binom{N}{m}$ possible subsets, which we will index by s. As in the usual supervised classification problem, we assume a class/feature distribution $P(x, c)$: [Here we use x to represent the full N-dimensional feature vectors and $x^{(s)}$ to represent the corresponding s-subset vectors.]

$$P(x^{(s)}, c) \;=\; P(x^{(s)}|c)\, P(c),$$

where $P(c)$ is either known *a priori* or estimated from the training data and:

$$P(x^{(s)}|c) \;=\; \frac{1}{(2\pi)^{m/2}|\mathbf{\Sigma}_c^{(s)}|^{1/2}} \exp\left[-\frac{1}{2}(x^{(s)} - \mu_c^{(s)})^T (\mathbf{\Sigma}_c^{(s)})^{-1}(x^{(s)} - \mu_c^{(s)})\right], \tag{46}$$

The error-minimizing subset is the one that maximizes $P(\mathbf{c}|\mathbf{x}^{(s)})$:

$$P(\mathbf{c}|\mathbf{x}^{(s)}) \;=\; \prod_{j=1}^{n} P(c_j|x_j^{(s)}),$$

assuming as usual that the individual $(c_j, x_j^{(s)})$ pairs are independent of each other. Anticipating the use of MDL, we point out that this choice also minimizes the class-label code length: $-\log P(\mathbf{c}|\mathbf{x}^{(s)})$.

4.3.2. Dimensionality Determination Using this approach to subset selection, how does one determine the optimal number of features? Simply maximizing $P(\mathbf{c}|\mathbf{x}^{(s)})$ doesn't work in this case because it will lead to over fitting. In the MDL approach the decrease in $-\log P(\mathbf{c}|\mathbf{x}^{(s)})$ induced by increasing m is offset by an increase in the model description length and the optimum number of features is the one for which the sum is minimized.

A model specification consists of the following in this case:

- the number of features m: a small constant number of bits
- the feature subset: $\log \binom{N}{m}$ bits

- the real-valued model parameters: $\frac{k}{2}\log n$ bits, where $k = K\left(\frac{m^2+3m+2}{2}\right)$ as in equation (44).

This approach to feature selection and dimensionality determination was taken in the work described in [10, 55]. Other work using MDL in this problem is described in [51, 52].

5. Image Segmentation

In computer vision MDL has probably been applied more to the problem of image segmentation than to any other. This must largely be due to the importance of segmentation to computer vision and to the fact that it is still, in a sense, an open problem. To review briefly, the problem of segmentation is that of dividing an image into a set of (usually) non-overlapping contiguous regions whose union is the entire image. This grouping is usually based on some similarity or homogeneity measure. Image segmentation approaches have been surveyed (see e.g. [20, 21]), but new algorithms appear continually in the literature. For a given image the issue of which segmentation is optimal is application-dependent in general. Nonetheless, most images can be "naturally" segmented manually by observers with little or no knowledge of the target application and, for many applications this "natural" segmentation coincides with the optimal application-dependent segmentation.

5.1. Supervised segmentation

There can be a blurring of the distinction between *supervised* and *unsupervised* segmentation because there are varying degrees to which prior knowledge can be incorporated into a segmentation scheme. At the supervised end of this continuum is supervised *pixel classification*, the simplest form of which is thresholding of a grey-scale (single-band) image into two or more classes. To complete the segmentation process the resulting class-labeled image is usually grouped into *connected components*. Pixel classification is also frequently applied to multi-band images such as color (three bands - e.g. red, green and blue) and the multi-spectral images used in remote sensing[60]. In some completely supervised problems, there are no model-complexity issues - everything is specified. In others, however, there may be model-structure issues, such as the number of features.

5.2. Unsupervised Segmentation

In completely unsupervised segmentation there are no training data and no *a priori* classes. As in supervised segmentation, the simplest form of unsupervised segmentation involves the thresholding of grey-scale images. In the unsupervised case, however, the threshold values and frequently the number of thresholds are adjustable parameters. Also, as in the supervised case described above, such techniques may be extended to higher dimensions (e.g. more than one band). The generalization of thresholding to higher dimensions is a non-trivial matter. In fact, this is essentially the problem of *clustering* discussed in the previous section and all of those techniques can be applied. In the unsupervised analog of pixel classification, in the simplest case, the "bands" correspond to intrinsic properties of the pixels themselves (e.g. measured reflected light in various spectral (wavelength) bands). Beyond that, segmentation algorithms often incorporate some form of spatial information. One way is to include spatial information among the "bands" (features). These might include for example, certain statistical measures of the pixel values in a neighborhood including the pixel to which they are to be associated. Another way is simply to include the pixel spatial coordinates as features. These approaches still use standard clustering algorithms. Other, more sophisticated approaches incorporate spatial information in other ways. We discuss these approaches in the sequel.

Most segmentation algorithms consist (either explicitly or implicitly) of two major parts: an *objective function* and an *optimization algorithm.* The objective function is a function of the image segmentation and its optimum (minimum or maximum) corresponds to the best segmentation. This is the most common place to include application-specific knowledge. For most objective functions the only known way to be certain of finding the global optimum is by exhaustively trying all possible segmentations. This obviously involves a prohibitive computational expense and, for this reason, a sub-optimal search is usually employed. The role of MDL in this process is in the construction of the objective function. In fact the objective function is the code length for the image data (pixel values) when encoded using a model of which a segmentation of the image is part.

5.2.1. An MDL-Based objective Function for Segmentation There are three segmentation algorithms[30, 23, 65] all based on very similar MDL-based objective functions, but all using different optimization algorithms. To our knowledge, the first of these[30] was the first to use MDL for segmentation and is still the most impressive in terms of the results obtained. The objective functions used in [30] and [23] are essentially identical except that an approximation is made in [30] before the optimization procedure. Here we discuss their development. In each region the spatial variation of grey levels (or values of the multiple bands) is described by a polynomial in the two spatial coordinates (x, y) plus Gaussian "noise". Said another way, they are distributed according to a Gaussian distribution with a fixed covariance matrix and a spatially varying mean. The objective function is based on a two-part code: data code length plus model code length:

$$L(\mathbf{y}|M) \quad + \quad L(M), \tag{47}$$

where $\mathbf{y}$ represents the collection of pixel values for the entire image and M represents the model.

The Data Code Length

Using this model,the data values are encoded using codes based on the assumed Gaussian model. The data code length for the entire collection of pixel values is equal to:

$$L(\mathbf{y}|M) \;=\; -\log P(\mathbf{y}|M) \;=\; -\sum_j \log p(\mathbf{y}_j|\theta_j), \tag{48}$$

where:

$$p(\mathbf{y}_j|\theta_j) \;=\; \prod_{i\in\omega_j} p(y_i \mid \theta_j), \tag{49}$$

in which ω_j represents the j^{th} region, θ_j its model parameters, $\mathbf{y}_j$ its data (y_i) values,

$$p(y_i|\theta_j) \;=\; \frac{1}{(2\pi)^{m/2}|\mathbf{\Sigma}_j|^{1/2}} \exp\left\{-\frac{1}{2}[y_i-\mu_j(x_i)]^T\mathbf{\Sigma}_j^{-1}[y_i-\mu_j(x_i)]\right\}, \tag{50}$$

and m is the number of bands (e.g. $m = 1$ for simple grey-scale images). The independence assumption implicit in equation (49) implies simply that the variations about the spatially varying mean are assumed independent. The actual (mean plus variation) values will be dependent because of the smoothly varying mean. Also, the values of two different bands for the same pixel may be dependent. This dependence will be characterized by the appropriate element of the covariance matrix $\mathbf{\Sigma}_j$.

The code-length-minimizing parameter values corresponding to equation (49) with equation (50) substituted for $p(y_i \mid \theta_j)$ are easily shown to be the maximum-likelihood values. This means that, for a given polynomial degree, the coefficients of the polynomial $\mu_j(x)$ are the least-squares values and $\mathbf{\Sigma}_j$ is the sample covariance matrix $\mathbf{S}_j$ for the pixels in region ω_j. Making these substitutions into equation (49) yields the following expression for the code length of the collection $\mathbf{y}_j$ of all pixel values in the j^{th} region[23]:

$$L(\mathbf{y}_j|M) \;=\; \frac{n_j}{2}\left[m(1+\log 2\pi) \;+\; \log|\mathbf{S}_j|\right] \tag{51}$$

It it noteworthy that this data cost holds regardless of the functional form used for the spatially dependent means, polynomial or otherwise.

The Model Code Length

For this model class the description of the model consists of a specification of the following.

1. the partitioning Ω of the image into regions $\Omega = \{\omega_j\}$.
2. for each region:
   (a) the unique elements of the covariance matrix
   (b) the polynomial degree
   (c) the polynomial coefficients

Thus we may write our model description length as The first of these model components, the partitioning into regions, is encoded using a chain code. This gives a code length roughly proportional to the total boundary length of the partitioning. The polynomial degrees of the regions are encoded using a small, fixed number of bits per region. The covariance matrix elements and the polynomial coefficients are real-valued and are encoded using a scheme that yields a code length of the form of equation (17), where the number of pixels in the associated region will be substituted for n.

For the partitioning the code length obtained is[23]:

$$L(\text{partitioning}) \;\approx\; \sum_i (l_i \log 3 + L_0(l_i)),$$

where the sum $\sum_i$ is over all boundary segments, l_i is the length in chain-code steps of the i^{th} boundary segment and $L_0(\ldots)$ is a function defined/derived in [23]. Using this and the other results presented thus far we can write our objective function as follows.

$$\begin{aligned} L(\ldots) = \quad & \textstyle\sum_i (l_i \log 3 + L_0(l_i)) + \sum_j \frac{K_j}{2} \log n_j + \\ & \textstyle\sum_j \frac{n_j}{2} \left[m \log(2\pi) + \log |\mathbf{\Sigma}_j| + \mathrm{trace}\left(\mathbf{\Sigma}_j^{-1} S_j \right) \right], \end{aligned} \tag{52}$$

where $\sum_i$ is a sum over all boundary segments and $\sum_j$ is a sum over all regions (not to be confused with the covariance matrix $\mathbf{\Sigma}_j$). Noting that the minimum-code-length (also maximum-likelihood) value of $\mathbf{\Sigma}_j$ is S_j this can be further simplified to:

$$\mathcal{L}(\mathbf{c}|\mathbf{y}) = \frac{n}{2} m(1 + \log 2\pi) \; + \sum_i [l_i \log 3 + L_0(l_i)] + \frac{1}{2} \sum_j [n_j \log |S_j| + K_j \log n_j] \tag{53}$$

5.2.2. Optimization Procedures   Greedy Region Merging

In the interest of speed a greedy region-merging was used in [23]. This has certain pitfalls but is much faster than the alternatives. Starting with an initial segmentation of the image (which may be just the image itself, with each pixel considered to be a separate region, or it may consist of larger regions produced by some heuristic device) the algorithm successively merges pairs of neighboring regions provided that the mergers decrease the total code-length. At each step the pair of regions producing the greatest code-length decrease are merged. Despite the simplicity of this device, quite good segmentations are produced. One such result is shown in Figure 3.

Continuation Methods

Equation (51) is appropriate once a region has been clearly specified. This may not be possible in some optimization procedures. A case in point is the continuation-based scheme used by Leclerc in [30]. In this algorithm, at any intermediate stage, there are no well defined regions. Let $\mathcal{L}_L(\mathbf{c})$ represent the objective function used, which is an approximation to the of single-band single-$\mathbf{\Sigma}$ (*sigma* for a single band) form of equation (52). For piecewise-constant models its form is:

$$\mathcal{L}_L(\mathbf{c}) \;\equiv\; \mathcal{L}_L(\mathbf{u}) \;=\; a \sum_{i \in I} \left(\frac{z_i - u_i}{\sigma} \right)^2 \;+\; \frac{b}{2} \sum_{i \in I} \sum_{j \in N_i} [1 - \delta(u_i - u_j)], \tag{54}$$

(a) (b)

Figure 3: (a) A single band image of a house; (b) segmentation result using up to 2nd-degree polynomials in the region-merging scheme of [23].

where

- I represents the image, the pixels of which are indexed by a single index (e.g. i or j).
- $z = \{z_i | i \in I\}$: z_i is the "measurement" (e.g. grey level) for the i^{th} pixel.
- $u = \{u_i | i \in I\}$: u_i is the mean (e.g. grey level) for the i^{th} pixel. This is the spatially constant (for each region) mean of the model. In this case u is, in a sense, equivalent to c.
- σ: is the standard deviation (another model parameter). This is assumed constant (the same for all regions) in this model, but may be allowed to vary (i.e. $\sigma \to \sigma_i$).
- $\delta(x) = 1$ when $x = 0$ but $= 0$ otherwise ("Kronecker delta").
- $a = 1/(2 \log 2)$.
- $b =$ "the encoding cost per unit boundary length" (i.e. per step in the chain code.)

The *continuation* method used in [30] embeds $\mathcal{L}_L(\mathbf{c})$ in a parametric (with parameter s) family of objective functions $\{\mathcal{L}_L(\mathbf{c}; s)\}$, where at some large s, $\mathcal{L}_L(\mathbf{c}; s)$ has a single minimum and for which the number and position of the local minima converge to those of

$\mathcal{L}_L(\mathbf{c})$ as s approaches zero. The only part of equation (54) that must be addressed by this embedding is the *Kronecker delta*, which he replaces with:

$$e_{i.j}(u,s) \;\equiv\; \exp\left[-\frac{(u_i-u_j)^2}{(s\sigma)^2}\right], \tag{55}$$

where, clearly

$$\lim_{s\to 0} e_{i.j}(u,s) \;=\; \delta(u_i-u_j). \tag{56}$$

Using this, the condition for minimizing $L(u,s)$ with respect to u becomes.

$$\frac{\partial L(u,s)}{\partial u} \;=\; b \;+\; A(u,s)u \;=\; 0, \tag{57}$$

where b and A are determined by the algebraic manipulation of the components of this vector equation.

Equation (57) is solved using a Gauss-Seidel iteration. Once this iteration has converged for a given s, he reduces s using:

$$s^{t+1} \;=\; rs^t \text{ where } (r\in(0,1)). \tag{58}$$

There is a trade-off between accuracy and computation time associated with r. The value he uses is $r=0.95$. This outer iteration (on s) is repeated until "s^{t+1} is sufficiently close to zero".

In [30] this approach is extended to cases where each region can have a different variance σ_i^2 and where the mean can vary polynomially within a region.

Variational Methods - "Active Contours"

In [65], Zhu and Yuille describe an algorithm that attempts to minimize an MDL-based objective function similar to Leclerc's[30]. In fact they describe it as the "continuum limit" of his. They write their objective ("energy") function as (using essentially their notation):

$$E[\Gamma,\{\alpha_i\}] \;=\; \sum_{i=1}^{M}\left\{\frac{\mu}{2}\oint_{\partial R_i} ds \;-\; \int\!\!\int_{R_i}\log P(I(x,y)|\alpha_i)dxdy \;+\; \lambda\right\}, \tag{59}$$

where:

- $\{R_i\}$ is the set of regions into which the image has been partitioned.
- M is the number of regions.
- Γ_i is the boundary of R_i. They also use ∂R_i for this.
- Γ is the partitioning of the image: $\Gamma=\bigcup_{i=1}^{M}\Gamma_i$. This is equivalent to the $\mathbf{c}$ of the previous section.

- $I(x, y)$ is the image intensity (grey-level) function.
- α_i is the parameter vector for the probability density for the grey levels in R_i.
- μ is the boundary encoding cost per unit boundary length.
- λ is the cost of encoding α_i. This is assumed to be constant for all regions.
- The first term is divided by 2 because each boundary segment gets counted twice.

Their problem is to find the partitioning, Γ, and the associated parameter vectors, $\{\alpha_i\}$, that minimize equation (59). They use a two-step procedure to do this. After selecting an initial Γ (procedure for that to be described below), they iterate the following two steps.

1. Holding Γ fixed, they calculate the $\{\alpha_i\}$ that minimize (59). This is a straightforward maximum-likelihood (ML) estimation problem where α_i is chosen to maximize:

$$\int\int_{R_I} \log P(I(x, y)|\alpha_i) dx dy.$$

2. Holding the $\{\alpha_i\}$ fixed, they adjust the boundaries, Γ to minimize (59). For solving this problem they use a variational approach similar to that used for *snakes*[25].

The first step is especially simple in the case they consider - Gaussian distributions with spatially piecewise-constant means and variances. The ML estimates for the two components of α_i in this case are the sample mean and the sample variance. The second step is not quite so simple, however.

To minimize (59) with respect to Γ they want to solve the following equation:

$$\frac{\delta E}{\delta \Gamma} = 0, \tag{60}$$

where E is the energy of equation (59) and the $\frac{\delta}{\delta}$ notation denotes the *functional derivative* in the sense of the *calculus of variations.* When written in a considerably different form, equation (60) is known as the *Euler equation.* Rather than solve (60) directly they perform a steepest-descent minimization of E. That is, for boundary points $\vec{v} = (x, y)$:

$$\frac{d\vec{v}}{dt} = -\frac{\delta E}{\delta \vec{v}}. \tag{61}$$

The interpretation of equation (61) is the following. Consider t (the "time") to be the variable with respect to which $\vec{v}$ and E change during the minimization. Thus, at each step in the iteration, $\vec{v}$ changes by an amount proportional to $\frac{\delta E}{\delta \vec{v}}$. This can be used to write the "motion equation" for points $\{\vec{v}\}$ on the common boundary between regions i and j:

$$\frac{d\vec{v}}{dt} = \vec{n}_{\vec{v}} \left\{ \log \left[\frac{P(I(\vec{v})|\alpha_i)}{P(I(\vec{v})|\alpha_j)} \right] - \mu \kappa_{\vec{v}} \right\}, \tag{62}$$

where $\kappa_{\vec{v}}$ is the boundary curvature at $\vec{v}$.

In order to make the algorithm more stable (reliable) Zhu and Yuille modify their *energy* function for use in this boundary motion equation. Rather than basing the motion on the code length of just the boundary point $\vec{v}$, they base it on the average code length of the pixels in a circular window centered on $\vec{v}$.

The complete algorithm of [65] is actually more involved than that described thus far, consisting of the variational method just described embedded in the following overall procedure:

1. The algorithm is initialized by placing N small *seed* regions uniformly randomly throughout the image. The pixels in these regions are assumed described by a Gaussian model as described. The large background region, on the other hand is assumed described by a uniform distribution.

2. The iterative variational boundary adjustment described above is applied until it converges to a set of regions.

3. A greedy region merging procedure like that used in [23] is used to merge all adjacent pairs of regions until the next merge would produce an increase in code length.

In [65] this algorithm is applied to both color and texture segmentation with impressive results.

5.2.3. Other MDL-based Segmentation Approaches Some other works on applying MDL to the problem of image segmentation are:

- Keeler '90[26], '91[27], who uses piecewise-linear boundary coding combined with a topological description (planar maps).

- Darrel and Pentland '95[8] who use the notion of *layers of support*, which may overlap and which may be consist of multiple connected components. They use robust estimation estimation techniques in their optimization. In addition to segmenting static images they apply their technique to motion segmentation (sequences of images e.g. video).

- Sawhney and Ayer[48] also perform motion segmentation using the the layers-of-support concept and a combination of robust estimation and mixture-density fitting using the EM algorithm.

6. Summary and Conclusion

Though most are not cited here, there are now many papers describing MDL-based approaches to computer vision problems. For example, we (with others) have applied it with the Hough transform to the problem of multiple curve detection[57, 56] with excellent results. It has proven to be a natural and effective criterion to apply in regularizing the ill-posed problems so ubiquitous in computer vision. The challenges to the practitioner who wishes to use it are the construction of a tractable objective function whose optimum corresponds to the ideal solution and the development of an efficient algorithm for finding its optimum point. It is hoped that the discussions presented here will be helpful in this regard.

Acknowledgment: Many helpful and enjoyable discussions with Eric Pauwels. Jianchang Mao for a discussion on the use of MDL and other regularization techniques in the estimation of neural-net structure.

References

[1] H. Akaike. A new look at the statistical model identification. *IEEE Transactions*, AC-19:716–723, 1974.

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

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth & Brooks/Cole, 1984. ISBN: 0-534-98054-6.

[4] G. J. Chaitin. On the lengths of programs for computing finite binary sequences. *JACM*, 13:547–569, 1966.

[5] G. J. Chaitin. *Algorithmic Information Theory.* Cambridge University Press, Cambridge, England, UK, 1987. ISBN 0-521-34306-2.

[6] G. J. Chaitin. *Information-Theoretic Incompleteness.* World Scientific, New York, 1992. ISBN 981-02-1208-9.

[7] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley, New York, 1991.

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

[9] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society*, B39:1–38, 1977.

[10] 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.

[11] Byron Dom. MDL estimation for small sample sizes and its application to linear regression. Technical Report RJ 10030, IBM Research Division, IBM Almaden Research Center; San Jose, CA, June 1996. Postscript version available at IBM *CyberDigest:* http://www.watson.ibm.com:8080/; submitted for publication.

[12] Byron Dom. MDL estimation with small sample sizes including an application to the problem of segmenting binary strings using Bernoulli models. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, June 1997. longer version: IBM Research Report RJ 9997 (89085). Postscript version available at IBM *CyberDigest:* http://www.watson.ibm.com:8080/.

[13] 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).

[14] 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.

[15] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman and Hall, New York, 1993. ISBN 0-412-04231-2.

[16] K. Fukunaga. *An Introduction to Statistical Pattern Recognition, 2nd ed.* Academic Press, New York, 1990.

[17] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural network architectures. *Neural Computation*, 7:219–269, 1995.

[18] A. Goncharsky. Ill-posed problems in image processing. In *Proceedings of The International Workshop on Computer Vision and Applied Geometry*, August 1995.

[19] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, New York, 1989. ISBN 0-201-14236-8.

[20] R. M. Haralick and L. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29:100–132, 1985.

[21] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*, volume 1. Addison-Wesley, 1992.

[22] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988. ISBN 0-13-022278-X.

[23] 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.

[24] 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.

[25] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.

[26] 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.

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

[28] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI '95*, pages 1137–1143, 1995.

[29] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:3–18, 1965.

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

[31] Y. G. Leclerc. *The Local Structure of Image Intensity Discontinuities.* PhD thesis, McGill University, Montréal, Québec, Canada, May 1989.

[32] Jianchang Mao. Regularization techniques in artificial neural networks. In *World Congress on Neural Networks*, volume 4, pages 75–79, 1993.

[33] Geoffrey J. McLachlan and Kaye E. Basford. *Mixture Models: Inference and Applications to Clustering.* Marcel Decker, Inc., 1988. ISBN 0-8247-7691-7.

[34] 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.

[35] Balas K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan-Kaufmann, 1991.

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

[37] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993. ISBN: 1-55860-238-0.

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

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

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

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

[42] 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.

[43] J. Rissanen. Fisher information and stochastic complexity. *IEEE Trans Inf Th*, 42(1):40–47, January 1996.

[44] 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.

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

[46] F. Rosenblatt. Two theorems of statistical separability in the perceptron. In *Proceedings of Symposium No. 10 held at the National Physical Laboratory, Nov. 1958*, volume 1, pages 421–456. H.M Stationary Office, London, 1959.

[47] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, NewYork, 1962.

[48] Harpreet Sawhney and Serge Ayer. Compact representation of videos though dominant and multiple motion estimation. *IEEE Trans PAMI*, 18(8), 1996.

[49] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

[50] David W. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization.* Wiley-Interscience, 1992. ISBN 0-471-54770-0.

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

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

[53] 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.

[54] C. E. Shannon. A mathematical theory of communication. *Bell Syst Tech J.*, (3):379–423, 1948.

[55] 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.

[56] 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.

[57] 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.

[58] R. J. Solomonoff. A formal theory of inductive inference, parts i and ii. *Information and Control*, 7:1–22,224–254, 1964.

[59] James A. Storer. *Data Compression: Methods and Theory.* Computer Science Press, Rockville, MD, 1988. ISBN: 0-7167-8156-5.

[60] P. H. Swain and S. M. Davis. Remote sensing: The quantitative approach. *McGraw-Hill*, 1978.

[61] A. N. Tikhonov. Regularization of incorrect problems. *Soviet. Math. Dokl.*, 4:1624–1627, 1963.

[62] M. Wax. Tree structured classification by the MDL principle. In *Proceedings of ICASSP 90*, pages 2157–2160, Albuqurque, New Mexico, April 1990.

[63] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn.* Morgan-Kaufmann, 1990.

[64] Tzay Y. Young and Editors King-Sun Fu, editors. *Handbook of Pattern Recognition and Image Processing.* Academic Press, 1986.

[65] S. C. Zhu and A. L. Yuille. Region competition: Unifying snakes, region growing, and Bayes/MDL for multi-band image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(9):884–900, September 1996.