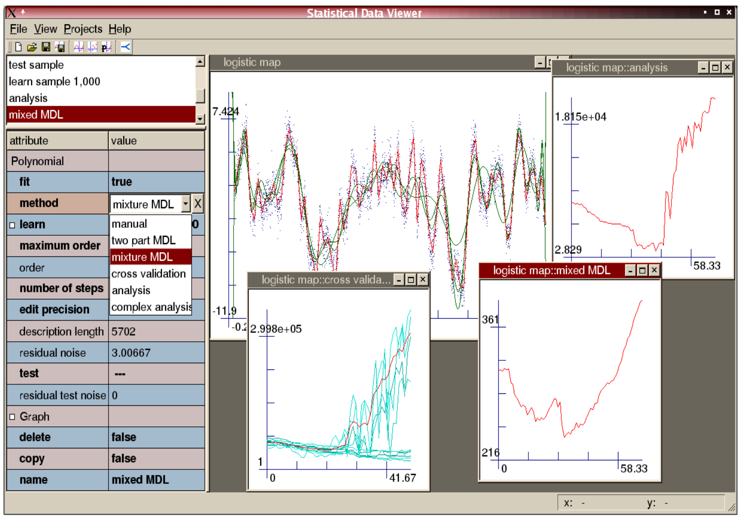
The Paradox of Overfitting

Volker Nannen

Master's thesis, April 2003



attribute	value
Polynomial	
fit	true
method	mixture MDL
learn	
maximum order	
order	
number of steps	
edit precision	
description length	5702
residual noise	3.00667
test	---
residual test noise	0
Graph	
delete	false
copy	false
name	mixed MDL

Supervisors:

Peter Grünwald	National Research Institute for Mathematics and Computer Science, Amsterdam
Rineke Verbrugge	Rijksuniversiteit Groningen

Artificial Intelligence
Rijksuniversiteit Groningen

Preface

The Faculty of Artificial Intelligence in Groningen does research on the technical aspects of cognition: reasoning, navigation, communication and learning. A technical approach requires its representations and algorithms to be robust and easy to use. It adds a crucial constraint to its domain of research: that of complexity. But complexity is more than a prominent constraint on real world applications. In recent years complexity theory has developed into an exciting area of mathematical research. It is a powerful mathematical tool that can lead to surprising solutions where conventional methods come to a dead end. Any theory of cognition that dismisses complexity constraints for the sake of theoretical freedom misses a powerful mathematical ally. This is why I have chosen complexity theory as the theme of my master's thesis, and specifically the experimental evaluation of an application of complexity theory on learning algorithms.

Applied sciences like medicine, physics or chemistry spend immense sums of money on technologies that visualize the objects of their research. Progress is published in all the media formats available. First of all this is done to get a better grip on the complicated problems of the science. But with such a wealth of information even a lay person finds it relatively easy to understand the synthesis of a virus, a special type of brain damage or the problem of bundling plasma streams in nuclear fusion. In statistics and complexity theory tools for visualization are rare and multi media publications are unheard of. This is not without consequences as the general public has almost no idea of statistics and complexity theory. The Nobel prize economy 2002 was shared by Daniel Kahneman for his findings on the sort of statistical awareness that actually governs our stock markets [KST82]. Though the rules of thumb that real-world decision-makers use are strong, they do not reflect statistical insight. To help the individual, whether scientist or not, to understand the theory in question I have tried to fit it into a modern graphical user interface.

My internal supervisor at the Faculty of Artificial Intelligence at the university of Groningen was Rineke Verbrugge. My external supervisor was Peter Grünwald from the Quantum Computing and Advanced Systems Research Group at the Centrum voor Wiskunde en Informatica, the National Research Institute for Mathematics and Computer Science in the Netherlands, situated in Amsterdam.

This document is written in pdf-LaTeX. It contains colored images and hyperlinks that can best be accessed with a pdf-viewer like ACROREAD. Together with the application and other information it is online available at

http://volker.nannen.com/work/mdl

Contents

List of Figures

Notations

The following conventions are used throughout this thesis:

σ^2	the mean squared distance or the variance of a distribution
c	a constant
$C(\cdot)$	plain Kolmogorov complexity
$D(\cdot\|\|\cdot)$	relative entropy or Kullback Leibler distance
f, g	functions
$H(\cdot)$	entropy
i, j, m, n	natural number
$J(\cdot)$	Fisher information
$K(\cdot)$	prefix Kolmogorov complexity
$K(\cdot\|\cdot)$	conditional prefix Kolmogorov complexity
k	degree, number of parameters
$\mathcal{L}$	learning algorithm
m	a model
m_p	a model that defines a probability distribution
$m_{k\in\mathbb{N}}$	a model that defines a probability distribution with k parameters
$\mathcal{N}(\cdot,\cdot)$	normal or Gaussian distribution
p, q	probabilistic distributions
s	a binary string or code
$\|s\|$	the length of a binary string in bits
s^*	a shortest program that computes s
$\mathcal{S}$	a set of strings
$\|\mathcal{S}\|$	the cardinality of a set
x, y	real numbers

1 Introduction

The aim of this master's thesis is the experimental verification of Minimum Description Length (MDL) as a method to effectively minimize the generalization error in data prediction. An application was developed to map the strength of the theory in a large number of controlled experiments: the *Statistical Data Viewer*. This application is primarily intended for scientists. Nevertheless, great care was taken to keep the interface simple enough to allow uninitiated students and interested outsiders to playfully explore and discover the complex laws and mathematics of the theory.

This first chapter will introduce you to model selection and the theory of Minimum Description Length. Chapter Two deals with the problems of experimental verification. To make you familiar with the *Statistical Data Viewer* and to give you an idea of the practical problems involved in model selection, it will also lead you through all the steps of a basic experiments. Chapter Three describes some selected experiments on two dimensional regression problems. Chapter Four discusses the results and Chapter Five gives a short summary of all the thesis. The appendix elaborates some of the algorithms that were used for the experiments.

It is not necessary to actually use the application in order to understand the experiments that are described in this thesis. They were selected for diversity and are intended to give you an optimal view on the problem. But they cannot cover all aspects of model selection. If you like to understand the problems of model selection even better you might want to conduct some experiments of your own.

1.1 The paradox of overfitting

Machine learning is the branch of Artificial Intelligence that deals with learning algorithms. Learning is a figurative description of what in ordinary science is also known as model selection and generalization. In computer science a model is a set of binary encoded values or strings, often the parameters of a function or statistical distribution. Models that parameterize the same function or distribution are called a family. Models of the same family are usually indexed by the number of parameters involved. This number of parameters is also called the degree or the dimension of the model.

model

To learn some real world phenomenon means to take some examples of the phenomenon and to select a model that describes them well. When such a model can also be used to describe instances of the same phenomenon that it was not trained on we say that it generalizes well or that it has a small generalization error. The task of a learning algorithm is to minimize this generalization error.

Classical learning algorithms did not allow for logical dependencies [MP69] and were not very interesting to Artificial Intelligence. The advance of techniques like neural networks with back-propagation in the 1980's and Bayesian networks in the 1990's has changed this profoundly. With such techniques it is possible to learn very complex relations. Learning algorithms are now extensively used in applications like expert systems, computer vision and language recognition. Machine learning has earned itself a central position in Artificial Intelligence.

A serious problem of most of the common learning algorithms is overfitting. Overfitting occurs when the models describe the examples better and better but get worse and worse on other instances of the same phenomenon. This can make the whole learning process worthless. A good way to observe overfitting is to split a number of examples in two, a training set, and a test set and to train the models on the training set. Clearly, the higher the degree of the model, the more information the model will contain about the training set. But when we look at the generalization error of the models on the test set, we will usually see that after an initial phase of improvement the generalization error suddenly becomes catastrophically bad. To the uninitiated student this takes some effort to accept since it apparently contradicts the basic empirical truth that more information will not lead to worse predictions. We may well call this *the paradox of overfitting* and hence the title of this thesis.

It might seem at first that overfitting is a problem specific to machine learning with its use of very complex models. And as some model families suffer less from overfitting than others the ultimate answer might be a model family that is entirely free from overfitting. But overfitting is a very general problem that has been known to statistics for a long time. And as overfitting is not the only constraint on models it will not be solved by searching for model families that are entirely free of it. Many families of models are essential to their field because of speed, accuracy, easy to teach mathematically, and other properties that are unlikely to be matched by an equivalent family that is free from overfitting. As an example, polynomials are used widely throughout all of science because of their many algorithmic advantages. They suffer very badly from overfitting. ARMA models are essential to signal processing and are often used to model time series. They also suffer badly from overfitting. If we want to use the model with the best algorithmic properties for our application we need a theory that can select the best model from any arbitrary family.

1.2 An example of overfitting

Figure 1 on page 3 gives a good example of overfitting. The upper graph shows two curves in the two-dimensional plane. One of the curves is a segment of the Lorenz attractor, the other a 43-degree polynomial. A Lorenz attractor is a complicated self similar object[1]. Here it is only important because it is definitely not a polynomial and because its curve is relatively smooth. Such a curve can be approximated well by a polynomial.

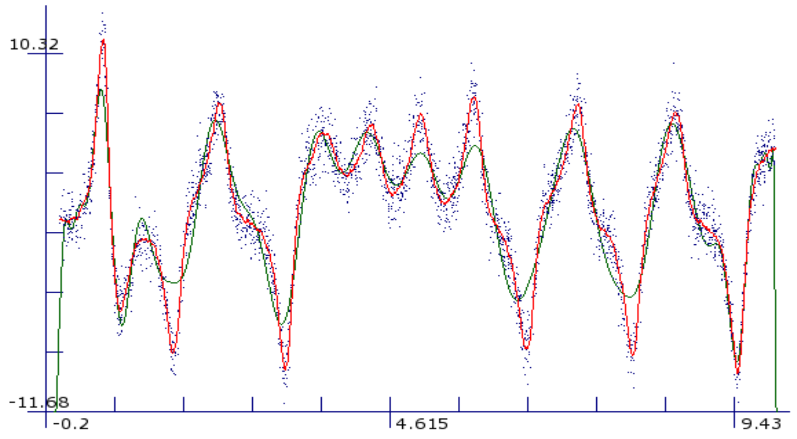
An n-degree polynomial is a function of the form

$$f(x) \; = \; a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \; , \quad x \in \mathbb{R} \tag{1}$$

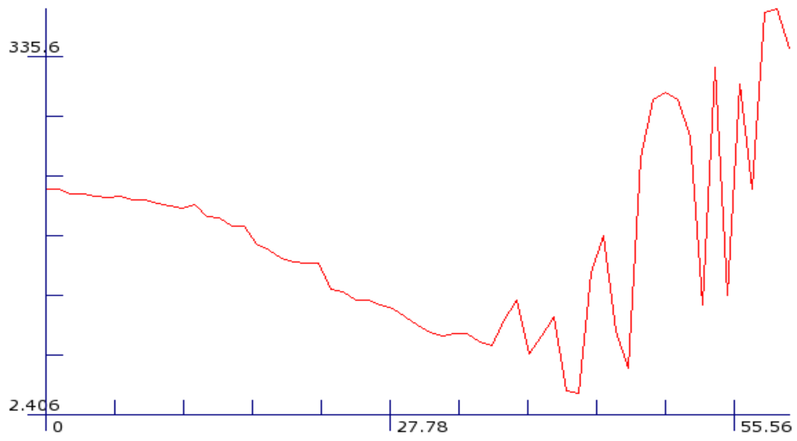
with an $n+1$-dimensional parameter space $(a_0 \ldots a_n) \in \mathbb{R}^{n+1}$.

[1] Appendix 5 on page 61 has more information on the Lorenz attractor.

Figure 1: An example of overfitting



Lorenz attractor and the optimum 43-degree polynomial (the curve with smaller oscillations). The points are the 300 point training sample and the 3,000 point test sample. Both samples are independently identically distributed. The distribution over the x-axis is uniform over the support interval $[0, 10]$. Along the y-axis, the deviation from the Lorenz attractor is Gaussian with variance $\sigma^2 = 1$.



Generalization (mean squared error on the test set) analysis for polynomials of degree 0–60. The x-axis shows the degree of the polynomial. The y-axis shows the generalization error on the test sample. It has logarithmic scale.

The first value on the left is the 0-degree polynomial. It has a mean squared error of $\sigma^2 = 18$ on the test sample. To the right of it the generalization error slowly decreases until it reaches a global minimum of $\sigma^2 = 2.7$ at 43 degrees. After this the error shows a number of steep inclines and declines with local maxima that soon are much worse than the initial $\sigma^2 = 18$.

A polynomial is very easy to work with and polynomials are used throughout science to model (or approximate) other functions. If the other function has to be inferred from a sample of points that witness that function, the problem is called a regression problem.

Based on a small training sample that witnesses our Lorenz attractor we search for a polynomial that optimally predicts future points that follow the same distribution as the training sample—they witness the same Lorenz attractor, the same noise along the y-axis and the same distribution over the x-axis. Such a sample is called i.i.d., independently identically distributed. In this thesis the i.i.d. assumption will be the only assumption about training samples and samples that have to be predicted.

The Lorenz attractor in the graph is witnessed by 3,300 points. To simulate the noise that is almost always polluting our measurements, the points deviate from the curve of the attractor by a small distance along the y-axis. They are uniformly distributed over the interval [0, 10] of the x-axis and are randomly divided into a 300 point training set and a 3,000 point test set. The interval [0, 10] of the x-axis is called the support.

The generalization analysis in the lower graph of Figure 1 shows what happens if we approximate the 300 point training set by polynomials of rising degree and measure the generalization error of these polynomials on the 3,000 point test set. Of course, the more parameters we choose, the better the polynomial will approximate the training set until it eventually goes through every single point of the training set. This is not shown in the graph. What is shown is the generalization error on the 3,000 points of the i.i.d. test set. The x-axis shows the degrees of the polynomial and the y-axis shows the generalization error.

Starting on the left with a 0-degree polynomial (which is nothing but the mean of the training set) we see that a polynomial that approximates the training set well will also approximate the test set. Slowly but surely, the more parameters the polynomial uses the smaller the generalization error becomes. In the center of the graph, at 43 degrees, the generalization error becomes almost zero. But then something unexpected happens, at least in the eyes of the uninitiated student. For polynomials of 44 degrees and higher the error on the test set rises very fast and soon becomes much bigger than the generalization error of even the 0-degree polynomial. Though these high degree polynomials continue to improve on the training set, they definitely do not approximate our Lorenz attractor any more. They *overfit*.

1.3 The definition of a good model

Before we can proceed with a more detailed analysis of model selection we need to answer one important question: what exactly is a good model. And one popular belief which is persistent even among professional statisticians has to be dismissed right from the beginning: the model that will achieve the lowest generalization error does *not* have to have the same degree or even be of the same family as the model that originally produced the data.

To drive this idea home we use a simple 4-degree polynomial as a source function. This polynomial is witnessed by a 100 point training sample and a 3,000 point test sample. To simulate noise, the points are polluted by a Gaussian distribution of variance $\sigma^2 = 1$ along the y-axis. Along the x-axis they are uniformly distributed over the support interval [0, 10]. The graph of this example and the analysis of the generalization error are shown in Figure 2. The generalization error shows that a 4-degree polynomial has a comparatively high generalization error. When trained on a sample of this size and noise there is only a very low probability that a 4-degree polynomial will ever show a satisfactory generalization error. Depending on the actual training sample the lowest generalization error is achieved for polynomials from 6 to 8 degrees.

This discrepancy is not biased by inaccurate algorithms. Neither can it be dismissed as the result of an unfortunate selection of sample size, noise and model family. The same phenomenon can be witnessed for ARMA models and many others under many different circumstances but especially for small sample sizes. In [Rue89] a number of striking examples are given of rather innocent functions the output of which cannot reasonably be approximated by any function of the same family. Usually this happens when the output is very sensitive to minimal changes in the parameters. Still, the attractor of such a function can often be parameterized surprisingly well by a very different family of functions[2].

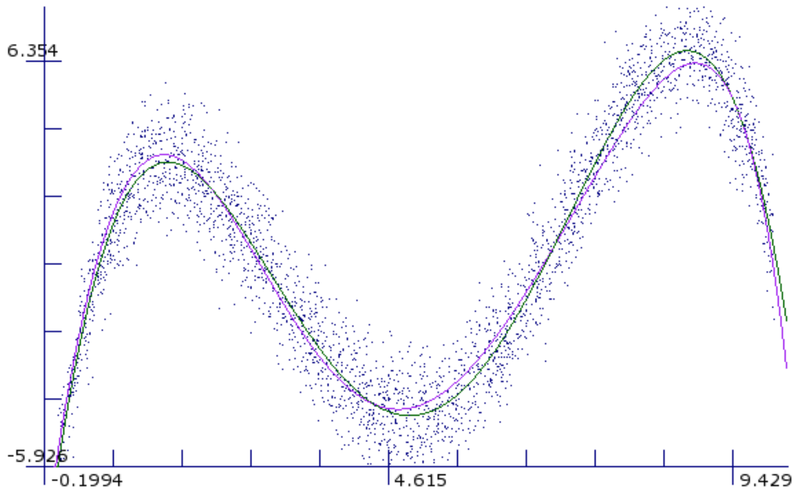
For most practical purposes a good model is a model that minimizes the generalization error on future output of the process in question. But in the absence of further output even this is a weak definition. We might want to filter useful information from noise or to compress an overly redundant file into a more convenient format, as is often the case in video and audio applications. In this case we need to select a model for which the data is most typical in the sense that the data is a truly random member of the model and virtually indistinguishable from all its other members, except for the noise. It implies that all information that has been lost during filtering or lossy compression was noise of a truly random nature. This definition of a good model is entirely independent from a source and is known as *minimum randomness deficiency.* It will be discussed in more detail on page 12.

We now have three definitions of a good model :

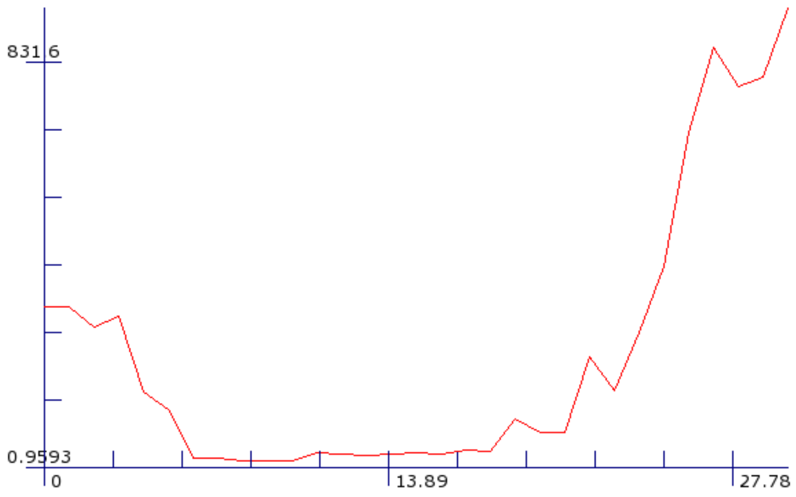
1. identifying family and degree of the original model for reconstruction purposes
2. minimum generalization error for data prediction
3. randomness deficiency for filters and lossy compression

[2] To add to the confusion, a function that accurately describes an attractor is often advocated as the original function. This can be compared to confusing a fingerprint with a DNA string. Both are unique identifiers of their bearer but only one contains his blueprint.

Figure 2: Defining a good model



Original 4-degree polynomial (green), 100 point training sample, 3,000 point test sample and 8-degree polynomial trained on the training sample (blue). In case you are reading a black and white print: the 8-degree polynomial lies above the 4-degree polynomial at the left peak and the middle valley and below the 4-degree polynomial at the right peak.



The analysis of the generalization error σ^2. A 0-degree polynomial achieves $\sigma^2 = 14$ and a 4-degree polynomial $\sigma^2 = 3.4$ on the test sample. All polynomials in the range 6–18 degrees achieve $\sigma^2 < 1.3$ with a global minimum of $\sigma^2 = 1.04$ at 8 degrees. From 18 degrees onwards we witness overfitting. Different training samples of the same size might witness global minima for polynomials ranging from 6 to 8 degrees and overfitting may start from 10 degrees onwards. 4 degrees are always far worse than 6 degrees. The y-axis has logarithmic scale.

We have already seen that a model of the same family and degree as the original model does not necessarily minimize the generalization error.

The important question is: can the randomness deficiency and the generalization error be minimized by the same model selection method?

Such a general purpose method would simplify teaching and would enable many more people to deal with the problems of model selection. A general purpose method would also be very attractive to the embedded systems industry. Embedded systems often hardwire algorithms and cannot adapt them to specific needs. They have to be very economic with time, space and energy consumption. An algorithm that can effectively filter, compress and predict future data all at the same time would indeed be very useful. But before this question can be answered we have to introduce some mathematical theory.

1.4 Information & complexity theory

This section gives only a raw overview of the concepts that are essential to this thesis. The interested reader is referred to the literature, especially the textbooks

Elements of Information Theory
by Thomas M. Cover and Joy A. Thomas, [CT91]

Introduction to Kolmogorov Complexity and Its Applications
by Ming Li and Paul Vitányi, [LV97]

which cover the fields of information theory and Kolmogorov complexity in depth and with all the necessary rigor. They are well to read and require only a minimum of prior knowledge.

Kolmogorov complexity. The concept of Kolmogorov complexity was developed independently and with different motivation by Andrei N. Kolmogorov [Kol65], Ray Solomonoff [Sol64] and Gregory Chaitin [Cha66], [Cha69].[3]

The Kolmogorov complexity $C(s)$ of any binary string $s \in \{0,1\}^n$ is the length of the shortest computer program s^* that can produce this string on the Universal Turing Machine UTM and then halt. In other words, on the UTM $C(s)$ bits of information are needed to encode s. The UTM is not a real computer but an imaginary reference machine. We don't need the specific details of the UTM. As every Turing machine can be implemented on every other one, the minimum length of a program on one machine will only add a constant to the minimum length of the program on every other machine. This constant is the length of the

$C(\cdot)$

UTM

[3] *Kolmogorov complexity* is sometimes also called *algorithmic complexity* and *Turing complexity*. Though Kolmogorov was not the first one to formulate the idea, he played the dominant role in the consolidation of the theory.

implementation of the first machine on the other machine and is independent of the string in question. This was first observed in 1964 by Ray Solomonoff.

Experience has shown that every attempt to construct a theoretical model of computation that is more powerful than the Turing machine has come up with something that is at the most just as strong as the Turing machine. This has been codified in 1936 by Alonzo Church as Church's Thesis: the class of algorithmically computable numerical functions coincides with the class of partial recursive functions. Everything we can compute we can compute by a Turing machine and what we cannot compute by a Turing machine we cannot compute at all. This said, we can use Kolmogorov complexity as a universal measure that will assign the same value to any sequence of bits regardless of the model of computation, within the bounds of an additive constant.

Incomputability of Kolmogorov complexity. Kolmogorov complexity is not computable. It is nevertheless essential for proving existence and bounds for weaker notions of complexity. The fact that Kolmogorov complexity cannot be computed stems from the fact that we cannot compute the output of every program. More fundamentally, no algorithm is possible that can predict of every program if it will ever halt, as has been shown by Alan Turing in his famous work on the halting problem [Tur36]. No computer program is possible that, when given any other computer program as input, will always output `true` if that program will eventually halt and `false` if it will not. Even if we have a short program that outputs our string and that seems to be a good candidate for being the shortest such program, there is always a number of shorter programs of which we do not know if they will ever halt and with what output.

Plain versus prefix complexity. Turing's original model of computation included special delimiters that marked the end of an input string. This has resulted in two brands of Kolmogorov complexity:

$C(\cdot)$ **plain Kolmogorov complexity:** the length $C(s)$ of the shortest binary string that is delimited by special marks and that can compute x on the UTM and then halt.

$K(\cdot)$ **prefix Kolmogorov complexity:** the length $K(s)$ of the shortest binary string that is *self-delimiting* [LV97] and that can compute x on the UTM and then halt.

The difference between the two is logarithmic in $C(s)$: the number of extra bits that are needed to delimit the input string. While plain Kolmogorov complexity integrates neatly with the Turing model of computation, prefix Kolmogorov complexity has a number of desirable mathematical characteristics that make it a more coherent theory. The individual advantages and disadvantages are described in [LV97]. Which one is actually used is a matter of convenience. We will mostly use the prefix complexity $K(s)$.

Individual randomness. A. N. Kolmogorov was interested in Kolmogorov complexity to define the individual randomness of an object. When s has no computable regularity it cannot be encoded by a program shorter than s. Such a string is truly random and its Kolmogorov complexity is the length of the string itself plus the commando `print`[4]. And indeed, strings with a Kolmogorov complexity close to their actual length satisfy all known tests of randomness. A regular string, on the other hand, can be computed by a program much shorter than the string itself. But the overwhelming majority of all strings of any length are random and for a string picked at random chances are exponentially small that its Kolmogorov complexity will be significantly smaller than its actual length.

This can easily be shown. For any given integer n there are exactly 2^n binary strings of that length and $2^n - 1$ strings that are shorter than n: one empty string, 2^1 strings of length one, 2^2 of length two and so forth. Even if all strings shorter than n would produce a string of length n on the UTM we would still be one string short of assigning a $C(s) < n$ to every single one of our 2^n strings. And if we want to assign a $C(s) < n - 1$ we can maximally do so for $2^{n-1} - 1$ strings. And for $C(s) < n - 10$ we can only do so for $2^{n-10} - 1$ strings which is less than 0.1% of all our strings. Even under optimal circumstances we will never find a $C(s) < n - c$ for more than $\frac{1}{2^c}$ of our strings.

Conditional Kolmogorov complexity. The conditional Kolmogorov complexity $K(s|a)$ is defined as the shortest program that can output s on the UTM if the input string a is given on an auxiliary tape. $K(s)$ is the special case $K(s|\epsilon)$ where the auxiliary tape is empty. $K(\cdot|\cdot)$

The universal distribution. When Ray Solomonoff first developed Kolmogorov complexity in 1964 he intended it to define a universal distribution over all possible objects. His original approach dealt with a specific problem of Bayes' rule, the unknown prior distribution. Bayes' rule can be used to calculate $P(m|s)$, the probability for a probabilistic model to have generated the sample s, given s. It is very simple. $P(s|m)$, the probability that the sample will occur given the model, is multiplied by the unconditional probability that the model will apply at all, $P(m)$. This is divided by the unconditional probability of the sample $P(s)$. The unconditional probability of the model is called the prior distribution and the probability that the model will have generated the data is called the posterior distribution.

$$P(m|s) = \frac{P(s|m)\ P(m)}{P(s)} \tag{2}$$

Bayes' rule can easily be derived from the definition of conditional probability:

$$P(m|s) = \frac{P(m,s)}{P(s)} \tag{3}$$

and

$$P(s|m) = \frac{P(m,s)}{P(m)} \tag{4}$$

The big and obvious problem with Bayes' rule is that we usually have no idea what the prior distribution $P(m)$ should be. Solomonoff suggested that if the

[4] Plus a logarithmic term if we use prefix complexity

true prior distribution is unknown the best assumption would be the universal distribution $2^{-K(m)}$ where $K(m)$ is the prefix Kolmogorov complexity of the model[5]. This is nothing but a modern codification of the age old principle that is wildly known under the name of Occam's razor: the simplest explanation is the most likely one to be true.

Entropy. Claude Shannon [Sha48] developed information theory in the late 1940's. He was concerned with the optimum code length that could be given to different binary words w of a source string s. Obviously, assigning a short code length to low frequency words or a long code length to high frequency words is a waste of resources. Suppose we draw a word w from our source string s uniformly at random. Then the probability $p(w)$ is equal to the frequency of w in s. Shannon found that the optimum overall code length for s was achieved when assigning to each word w a code of length $-\log p(w)$. Shannon attributed the original idea to R.M. Fano and hence this code is called the Shannon-Fano code. When using such an optimal code, the average code length of the words of s can be reduced to

$$H(s) \;=\; -\sum_{w \in s} p(w) \log p(w) \tag{5}$$

$H(\cdot)$

where $H(s)$ is called the entropy of the set s. When s is finite and we assign a code of length $-\log p(w)$ to each of the n words of s, the total code length is

$$-\sum_{w \in s} \log p(w) \;=\; n\,H(s) \tag{6}$$

Let s be the outcome of some random process W that produces the words $w \in s$ sequentially and independently, each with some known probability $p(W = w) > 0$. $K(s|W)$ is the Kolmogorov complexity of s given W. Because the Shannon-Fano code is optimal, the probability that $K(s|W)$ is significantly less than $nH(W)$ is exponentially small. This makes the negative log likelihood of s given W a good estimator of $K(s|W)$:

$$\begin{aligned} K(s|W) \;&\approx\; n\,H(W) \\ &\approx\; \sum_{w \in s} \log p(w|W) \\ &=\; -\log p(s|W) \end{aligned} \tag{7}$$

$D(\cdot||\cdot)$

Relative entropy. The relative entropy $D(p||q)$ tells us what happens when we use the wrong probability to encode our source string s. If $p(w)$ is the true distribution over the words of s but we use $q(w)$ to encode them, we end up with an average of $H(p) + D(p||q)$ bits per word. $D(p||q)$ is also called the

[5] Originally Solomonoff used the plain Kolmogorov complexity $C(\cdot)$. This resulted in an improper distribution $2^{-C(m)}$ that tends to infinity. Only in 1974 L.A. Levin introduced prefix complexity to solve this particular problem, and thereby many other problems as well [Lev74].

Kullback Leibler distance between the two probability mass functions p and q. It is defined as

$$D(p||q) \;=\; \sum_{w \in s} p(w) \log \frac{p(w)}{q(w)} \tag{8}$$

Fisher information. Fisher information was introduced into statistics some 20 years before C. Shannon introduced information theory [Fis25]. But it was not well understood without it. Fisher information is the variance of the score V of the continuous parameter space of our models m_k. This needs some explanation. At the beginning of this thesis we defined models as binary strings that discretize the parameter space of some function or probability distribution. For the purpose of Fisher information we have to temporarily treat a model m_k as a vector in $\mathbb{R}^k$. And we only consider models where for all samples s the mapping $f_s(m_k)$ defined by $f_s(m_k) = p(s|m_k)$ is differentiable. Then the score V can be defined as

$$\begin{aligned} V \;&=\; \frac{\partial}{\partial\, m_k} \;\ln p(s|m_k) \\ &=\; \frac{\frac{\partial}{\partial\, m_k}\; p(s|m_k)}{p(s|m_k)} \end{aligned} \tag{9}$$

The score V is the partial derivative of ln $p(s|m_k)$, a term we are already familiar with. The Fisher information $J(m_k)$ is $J(\cdot)$

$$J(m_k) \;=\; E_{m_k}\left[\frac{\partial}{\partial\, m_k}\;\ln p(s|m_k)\right]^2 \tag{10}$$

Intuitively, a high Fisher information means that slight changes to the parameters will have a great effect on $p(s|m_k)$. If $J(m_k)$ is high we must calculate $p(s|m_k)$ to a high precision. Conversely, if $J(m_k)$ is low, we may round $p(s|m_k)$ to a low precision.

Kolmogorov complexity of sets. The Kolmogorov complexity of a set of strings $\mathcal{S}$ is the length of the shortest program that can output the members of $\mathcal{S}$ on the UTM and then halt. If one is to approximate some string s with $\alpha < K(s)$ bits then the best one can do is to compute the smallest set $\mathcal{S}$ with $K(\mathcal{S}) \leq \alpha$ that includes s. Once we have some $\mathcal{S} \ni s$ we need at most $\log|\mathcal{S}|$ additional bits to compute s. This set $\mathcal{S}$ is defined by the Kolmogorov structure function $h_s(\cdot)$

$$h_s(\alpha) \;=\; \min_{\mathcal{S}} \left[\, \log|\mathcal{S}| : \mathcal{S} \ni s,\; K(\mathcal{S}) \leq \alpha \right] \tag{11}$$

which has many interesting features. The function $h_s(\alpha) + \alpha$ is non increasing and never falls below the line $K(s)+O(1)$ but can assume any form within these constraints. It should be evident that

$$h_s(\alpha) \;\geq\; K(s) - K(\mathcal{S}) \tag{12}$$

Kolmogorov complexity of distributions. The Kolmogorov structure function is not confined to finite sets. If we generalize $h_s(\alpha)$ to probabilistic models m_p that define distributions over $\mathbb{R}$ and if we let s describe a real number, we obtain

$$h_s(\alpha) \;=\; \min_{m_p} \big[-\log p(s|m_p) : p(s|m_p) > 0,\; K(m_p) \leq \alpha \big] \tag{13}$$

where $-\log p(s|m_p)$ is the number of bits we need to encode s with a code that is optimal for the distribution defined by m_p. Henceforth we will write m_p when the model defines a probability distribution and m_k with $k \in \mathbb{N}$ when the model defines a probability distribution that has k parameters. A set $\mathcal{S}$ can be viewed as a special case of m_p, a uniform distribution with

$$p(s|m_p) \;=\; \begin{cases} \frac{1}{|\mathcal{S}|} & \text{if } \; s \in \mathcal{S} \\ 0 & \text{if } \; s \notin \mathcal{S} \end{cases} \tag{14}$$

$\delta(\cdot|m_p)$

Minimum randomness deficiency. The randomness deficiency of a string s with regard to a model m_p is defined as

$$\delta(s|m_p) \;=\; -\log p(s|m_p) - K(\, s|m_p,\, K(m_p)\,) \tag{15}$$

for $p(s) > 0$, and ∞ otherwise. This is a generalization of the definition given in [VV02] where models are finite sets. If $\delta(s|m_p)$ is small, then s may be considered a *typical* or *low profile* instance of the distribution. s satisfies *all* properties of low Kolmogorov complexity that hold with high probability for the support set of m_p. This would not be the case if s would be exactly identical to the mean, first momentum or any other special characteristic of m_p.

Randomness deficiency is a key concept to any application of Kolmogorov complexity. As we saw earlier, Kolmogorov complexity and conditional Kolmogorov complexity are not computable. We can never claim that a particular string s does have a conditional Kolmogorov complexity

$$K(s|m_p) \;\approx\; -\log p(s|m_p) \tag{16}$$

typicality

The technical term that defines all those strings that do satisfy this approximation is *typicality*, defined as a small randomness deficiency $\delta(s|m_p)$.

Minimum randomness deficiency turns out to be important for lossy data compression. A compressed string of minimum randomness deficiency is the most difficult one to distinguish from the original string. The best lossy compression that uses a maximum of α bits is defined by the minimum randomness deficiency function

$\beta_s(\cdot)$

$$\beta_s(\alpha) \;=\; \min_{m_p} \big[\delta(s|m_p) : p(s|m_p) > 0,\; K(m_p) \leq \alpha\big] \tag{17}$$

Minimum Description Length. The Minimum Description Length or short MDL of a string s is the length of the shortest two-part code for s that uses less than α bits. It consists of the number of bits needed to encode the model m_p that defines a distribution and the negative log likelihood of s under this distribution.

MDL

$\lambda_s(\cdot)$

$$\lambda_s(\alpha) \;=\; \min_{m_p} \big[-\log p(s|m_p) + K(m_p) : p(s|m_p) > 0,\; K(m_p) \le \alpha \big] \qquad (18)$$

It has recently been shown by Nikolai Vereshchagin and Paul Vitányi in [VV02] that a model that minimizes the description length also minimizes the randomness deficiency, though the reverse may not be true. The most fundamental result of that paper is the equality

$$\beta_s(\alpha) \;=\; h_s(\alpha) + \alpha - K(s) \;=\; \lambda_s(\alpha) - K(s) \qquad (19)$$

where the mutual relations between the Kolmogorov structure function, the minimum randomness deficiency and the minimum description length are pinned down, up to logarithmic additive terms in argument and value.

MDL minimizes randomness deficiency. With this important result established, we are very keen to learn whether MDL can minimize the generalization error as well.

1.5 Practical MDL

From 1978 on Jorma Rissanen developed the idea to minimize the generalization error of a model by penalizing it according to its description length [Ris78]. At that time the only other method that successfully prevented overfitting by penalization was the Akaike Information Criterion (AIC). The AIC selects the model m_k according to

$\mathcal{L}_{AIC}(\cdot)$

$$\mathcal{L}_{AIC}(s) \;=\; \min_k \big[n\;\log \sigma_k^2 + 2k \big] \qquad (20)$$

where σ_k^2 is the mean squared error of the model m_k on the training sample s, n the size of s and k the number of parameters used. H. Akaike introduced the term $2k$ in his 1973 paper [Aka73] as a penalty on the complexity of the model.

Compare this to Rissanen's original MDL criterion:

$\mathcal{L}_{Ris}(\cdot)$

$$\mathcal{L}_{Ris}(s) \;=\; \min_k \big[-\log p(s|m_k) + k \log \sqrt{n} \;\; \big] \qquad (21)$$

Rissanen replaced Akaike's modified error $n\log(\sigma_k^2)$ by the information theoretically more correct term $-\log p(s|m_k)$. This is the length of the Shannon-Fano code for s which is a good approximation of $K(s|m_k)$, the complexity of the data given the k-parameter distribution model m_k, typicality assumed[6]. Further, he penalized the model complexity not only according to the number of parameters but according to both parameters and precision. Since statisticians at that time treated parameters usually as of infinite precision he had to come up with a reasonable figure for the precision any given model needed and postulated it to be $\log \sqrt{n}$ per parameter. This was quite a bold assumption but it showed reasonable results. He now weighted the complexity of the encoded data against the complexity of the model. The result he rightly called Minimum Description Length because the winning model was the one with the lowest combined complexity or description length.

Rissanen's use of model complexity to minimize the generalization error comes very close to what Ray Solomonoff originally had in mind when he first developed Kolmogorov complexity. The maximum a posteriori model according to Bayes' rule, supplied with Solomonoff's universal distribution, will favor the Minimum Description Length model, since

$$\begin{aligned} \max_m \, [P(m|s)] &= \max_m \left[\frac{P(s|m)\; P(m)}{P(s)}\right] \\ &= \max_m \left[P(s|m)\; 2^{-K(m)}\right] \\ &= \min_m \, \big[-\log P(s|m) + K(m)\big] \end{aligned} \tag{22}$$

Though Rissanen's simple approximation of $K(m) \approx k \log \sqrt{n}$ could compete with the AIC in minimizing the generalization error, the results on small samples were rather poor. But especially the small samples are the ones which are most in need of a reliable method to minimize the generalization error. Most methods converge with the optimum results as the sample size grows, mainly due to the law of large numbers which forces the statistics of a sample to converge with the statistics of the source. But small samples can have very different statistics and the big problem of model selection is to estimate how far they can be trusted.

In general, two-part MDL makes a strict distinction between the theoretical complexity of a model and the length of the implementation actually used. All versions of two-part MDL follow a three stage approach:

1. the complexity $-\log p(s|m_k)$ of the sample according to each model m_k is calculated at a high precision of m_k.
2. the minimum complexity $K(m_k)$ which would theoretically be needed to achieve this likelihood is estimated.
3. this theoretical estimate $E\big[K(m_k)\big]$ minus the previous $\log p(s|m_k)$ approximates the overall complexity of data and model.

[6] For this approximation to hold, s has to be typical for the model m_k. See Section 1.4 on page 12 for a discussion of typicality and minimum randomness deficiency.

Mixture MDL. More recent versions of MDL look deeper into the complexity of the model involved. Solomonoff and Rissanen in their original approaches minimized a two-part code, one code for the model and one code for the sample given the model. Mixture MDL leaves this approach. We do no longer search for a particular model but for the number of parameters k that minimizes the total code length $-\log p(s|k) + \log(k)$. To do this, we average $-\log\, p(s|m_k)$ over all possible models m_k for every number of parameters k, as will be defined further below.

$\mathcal{L}_{mix}(\cdot)$

$$\mathcal{L}_{mix}(s) \;=\; \min_k \Big[-\log p(s|k) + \log k \Big] \tag{23}$$

Since the model complexity is reduced to $\log k$ which is almost constant and has little influence on the results, it is not appropriate anymore to speak of a mixture code as a two-part code.

Let M_k be the k-dimensional parameter space of a given family of models and let $p(M_k = m_k)$ be a prior distribution over the models in M_k[7]. Provided this prior distribution is defined in a proper way we can calculate the probability that the data was generated by a k-parameter model as

$$p(s|k) \;=\; \int_{m_k \in M_k} p(m_k)\; p(s|m_k)\; dm_k \tag{24}$$

Once the best number of parameters k is found we calculate our model m_k in the conventional way. This approach is not without problems and the various versions of mixture MDL differ in how they address them:

- The binary models m_k form only a discrete subset of the continuous parameter space M_k. How are they distributed over this parameter space and how does this effect the results?
- what is a reasonable prior distribution over M_k?
- for most priors the integral goes to zero or infinity. How do we normalize it?
- the calculations become too complex to be carried out in practice.

Minimax MDL. Another important extension of MDL is the minimax strategy. Let m_k be the k-parameter model that can best predict n future values from some i.i.d. training values. Because m_k is unknown, every model $\hat{m}_k$ that achieves a least square error on the training values will inflict an extra cost when predicting the n future values. This extra cost is the Kullback Leibler distance

$$D(m_k||\hat{m}_k) \;=\; \sum_{x^n \in X^n} p(x^n|m_k) \log \frac{p(x^n|m_k)}{p(x^n|\hat{m}_k)}. \tag{25}$$

The minimax strategy favors the model m_k that minimizes the maximum of this extra cost.

$\mathcal{L}_{mm}(\cdot)$

$$\mathcal{L}_{\text{mm}} \;=\; \min_{k} \;\max_{m_k \in M_k}\; D(m_k || \hat{m}_k) \tag{26}$$

1.6 Error minimization

Any discussion of information theory and complexity would be incomplete without mentioning the work of Carl Friedrich Gauss (1777–1855). Working on astronomy and geodesy, Gauss spend a great amount of research on how to extract accurate information from physical measurements. Our modern ideas of error minimization are largely due to his work.

Euclidean distance and mean squared error. To indicate how well a particular function $f(x)$ can approximate another function $g(x)$ we use the Euclidean distance or the mean squared error. Minimizing one of them will minimize the other so which one is used is a matter of convenience. We use the mean squared error. For the interval $x \in [a, b]$ it is defined as

$$\sigma_f^2 \;=\; \frac{1}{b-a} \int_a^b \big(f(x) - g(x)\big)^2 \, dx \tag{27}$$

This formula can be extended to multi-dimensional space.

Often the function that we want to approximate is unknown to us and is only witnessed by a sample that is virtually always polluted by some noise. This noise includes measurement noise, rounding errors and disturbances during the execution of the original function. When noise is involved it is more difficult to approximate the original function. The model has to take account of the distribution of the noise as well. To our great convenience a mean squared error σ^2 can also be interpreted as the variance of a Gaussian or normal distribution. The Gaussian distribution is a very common distribution in nature. It is also akin to the concept of Euclidean distance, bridging the gap between statistics and geometry. For sufficiently many points drawn from the distribution $\mathcal{N}\big(f(x), \sigma^2\big)$ the mean squared error between these points and $f(x)$ will approach σ^2 and approximating a function that is witnessed by a sample polluted by Gaussian noise becomes the same as approximating the function itself.

$\mathcal{N}(\cdot,\cdot)$

Let a and b be two points and let l be the Euclidean distance between them. A Gaussian distribution $p(l)$ around a will assign the maximum probability to b if the distribution has a variance that is equal to l^2. To prove this, we take the first derivative of $p(l)$ and equal it to zero:

[7] For the moment, treat models as vectors in $\mathbb{R}^k$ so that integration is possible. See the discussion on Fisher information in Section 1.4 on page 11 for a similar problem.

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\,\sigma}\,p(l\,) &= \frac{\mathrm{d}}{\mathrm{d}\,\sigma}\,\frac{1}{\sigma\sqrt{2\pi}}\,e^{-l^2/2\sigma^2} \\
&= \frac{-1}{\sigma^2\sqrt{2\pi}}\,e^{-l^2/2\sigma^2} + \frac{1}{\sigma\sqrt{2\pi}}\,e^{-l^2/2\sigma^2}\left(\frac{2\,l^2}{2\,\sigma^3}\right) \\
&= \frac{1}{\sigma^2\sqrt{2\pi}}\,e^{-l^2/2\sigma^2}\left(\frac{l^2}{\sigma^2}-1\right) \\
&= 0
\end{aligned}
\tag{28}
$$

which leaves us with

$$
\sigma^2 \;=\; l^2. \tag{29}
$$

Selecting the function f that minimizes the Euclidean distance between $f(x)$ and $g(x)$ over the interval $[a,b]$ is the same as selecting the maximum likelihood distribution, the distribution $\mathcal{N}\big(f(x),\,\sigma^2\big)$ that gives the highest probability to the values of $g(x)$.

Maximum entropy distribution. Of particular interest to us is the entropy of the Gaussian distribution. The optimal code for a value that was drawn according to a Gaussian distribution $p(x)$ with variance σ^2 has a mean code length or entropy of

$$
\begin{aligned}
H(P) &= -\int_{-\infty}^{\infty} p(x)\log p(x)\;dx \\
&= \frac{1}{2}\log\left(2\pi e\sigma^2\right)
\end{aligned}
\tag{30}
$$

To always assume a Gaussian distribution for the noise may draw some criticism as the noise may actually have a very different distribution. Here another advantage of the Gaussian distribution comes in handy: for a given variance the Gaussian distribution is the maximum entropy distribution. It gives the lowest log likelihood to all its members of high probability. That means that the Gaussian distribution is the safest assumption if the true distribution is unknown. Even if it is plain wrong, it promises the lowest cost of a wrong prediction regardless of the true distribution [Grü00].

2 Experimental verification

As an expert on statistics and machine learning you are asked to supply a method for model selection to some new problems:

> A number of deep sea mining robots have been lost due to system failure. Given the immense cost of the robots, the mining company wants you to predict the risk of a loss as accurate as possible. Up to now about a hundred of the machines have been lost, under very different conditions. Decennia of experience with deep sea mining have taught the mining company to use some sophisticated risk evaluation models that you have to apply. Can you recommend MDL?
>
> A deep sea mining robot has got stuck between some rocks. The standard behaviors that were supposed to free the robot have failed. Some of the movements it made have won it partial freedom, others have made the situation only worse. So far, about a hundred movements have been tried and the outcomes have been carefully recorded. To choose the next action sequence, can you recommend MDL?

Before we are going to use MDL on problems that are new to us, we want to be sure that it is indeed a strong theory, valid even without the need of any preprocessing of the data or other optimizations that are common if a method is repeatedly applied to the same domain. In the case of MDL there are countless ways of expanding and compressing data and sooner or later we will find one that matches good models to short descriptions in a specific domain. But this does not convince us that it can be universally applied.

To experimentally verify that MDL would be a good choice to solve the problems above, we want to

1. test it on a broad variety of problems
2. prove that we use the shortest description possible

2.1 The Statistical Data Viewer

There are two factors that limit the type of data that is usually used for statistical experiments: availability and programming constraints.

Data availability. A major problem in machine learning and in statistics in general is the availability of appropriate data. One of the basic principles of statistics says that a sample that has been used to verify one hypothesis cannot be used to verify another one. And one and the same sample can never be used to both select and to prove a hypothesis. If we would do so, we would simply optimize on the sample in question.

This applies to model selection as well. Not only are the individual models hypotheses in the statistical sense, a general method for model selection can itself be viewed as a hypothesis. If we want to experimentally verify methods for model selection we need a large supply of unspoiled problems and data.

Mathematical programming. A major obstacle to an objective diversity of the data is the way the common mathematical packages work. They are based on scripting languages that make heavy use of predefined function calls. This integrates nicely with most mathematical concepts, which are usually defined as functions. But it is not the preferred way to handle complex data structures or to conduct sophisticated experiments. It also severely reduces the quality of the outcome of an experiment. It is quite difficult and tiring to manipulate the graphical representations of data by using function calls.

Another problem of a function oriented approach is that it is the responsibility of the user to keep the definitions and the results of an experiment together. Users are often reluctant to play with the settings of complex experiments as that requires an extensive version control of scripts and respective results. And instead of spending a lot of time on writing a lot of different scripts for a lot of different experiments, researchers tend to do only minor changes to an experiment once it works correctly, and repeat it on large amounts of similar data. It is easier to try a method ten thousand times on the same problem space than to try it a few times on two different problem spaces.

A visual approach. To overcome the limitations of mathematic scripting and to guarantee diversity of the learning problems in an objective way we developed the *Statistical Data Viewer*: an arbitrary precision math application with a modular graphical user interface that can visualize all the abstract difficulties of model selection. It is well documented and can be downloaded for free at

http://volker.nannen.com/work/mdl

The *Statistical Data Viewer* makes all the definitions and results of an experiment available through a sophisticated editor. Experiments can be set up in a fast and efficient way. Problems can visually be selected for diversity. The performance of selection methods can be analyzed in a number of interactive plots. All graphical representations are fully integrated into the control structure of the application, allowing the user to change views and to select and manipulate anything they show.

To develop a working application within reasonable limits of time, the first version of the application is limited to the model family of polynomials and to two-dimensional regression problems, which are easier to visualize. Polynomials are used widely throughout science and their mathematics are well understood. They suffer badly from overfitting.

Great care was taken to give uninitiated students and interested lay persons access to the theory as well. No scripting language is needed to actually set up an experiment. The predefined mathematical objects—problems, samples, models and selection methods—are all available as visible objects. They can be specified and combined through the interactive editor which is extremely easy to use. The essential versions of MDL are implemented. They can be analyzed and compared with another independent and successful method for model selection, cross-validation. In addition, the user can try his or her own penalization term for two-part MDL.

The progress of an experiment is observable and the execution can be disrupted at any moment. If possible, the graphs in the plots show the state of an experiment during its execution. It is possible to reproduce everything with little effort. All the data are saved in a standardized well documented format that allows for verification and further processing by other programs. The graphs are of scientific quality and available for print.

Available learning problems. To provide the user with a broad range of interesting learning problems, a number of random processes are available. They include

- smooth functions which can be approximated well by polynomials, e.g. the sinus function
- functions that are particularly hard to approximate like the step function
- fractals
- polynomials

When taking a sample from a process it can be distorted by noise from different distributions: Gaussian, uniform, exponential and Cauchy, which generates extreme outliers. Section 3.4 will explain them in more detail and will use them for a number of experiments. The distribution over the support of a process can also be manipulated. It can be the uniform distribution or a number of overlapping Gaussian distributions, to simulate local concentrations and gaps. This option is explained and used in Sections 3.2 and 3.3.

Samples can also be loaded from a file or drawn by hand on a canvas, to pinpoint particular problems.

Objective generalization analysis. To map the performance in minimizing the generalization error in an objective way it is possible to check every model against a test sample drawn from the same source as the training sample. This has drawn some criticism from experts as the conventional way to measure the generalization error seems to be to check a model against the original source if it is known. I opted against this because:

- When speaking of generalization error the check against a test sample gives a more intuitive picture of the real world performance.
- For data drawn by hand or loaded from a file the original distribution might be unknown. In this case all we can do is to set aside part of the data as a test sample for evaluation. Depending on whether the source is known or unknown we now would have two different standards, the original function for known sources and a test set for unknown sources.

- If the source is known the test sample can be made big enough to faithfully portray the original distribution (by the law of large numbers) and give as fair a picture of the generalization error as a check against the original process.
- When using the original source the correlation between model and source has to be weighted against the distribution over the support set of the source. Especially in the case of multiple Gaussian distributions over the support this can be extremely difficult to compute.
- Although we concentrate on i.i.d. samples, this method allows us also to train a model on a sample with Gaussian noise and to measure the generalization error on a test sample that was polluted by Cauchy noise (extreme outliers) and vice versa or to use different distributions over the support set.

An independent method to compare. Comparing the predictions of MDL with the generalization analysis on an i.i.d. test set does not show whether MDL is a better method for model selection than any other method. For this reason the application includes an implementation of cross-validation. Cross-validation is a successful method for model selection that is not based on complexity theory. It can be seen as a randomized algorithm where we select the model that is most likely to have a low generalization error. The method divides the training sample a couple of times into a training set and a test set. For each partition it fits the model on the training set and evaluates the result against the respective test set in the same way as the generalization analysis uses an independent test sample. The results of all partitions are combined to select the best model.

2.2 Technical details of the experiments

We are primarily concerned with two versions of MDL: Rissanen's original version of two-part MDL and the modern mixture MDL. We compare the performance of these methods with an objective analysis of the generalization error and with another successful method for data prediction that is independent of the MDL theory: cross-validation. To keep the computations feasible we limit the polynomials in the experiments to 150 degrees or less.

Rissanen's original two-part MDL. The implementation of Rissanen's original version is strait forward. Rissanen calculated the combined complexity of model and data to model as

$$-\log p(s|m_k) + k \log \sqrt{n} \tag{31}$$

Let $s = \{(x_1, y_1) \ldots (x_n, y_n)\}$ be a two-dimensional training sample and let our models be polynomials with a Gaussian deviation. For each number of parameters k we first calculate the polynomial $f_k(x)$ that has the least squared error from s. The precise algorithm is described in Appendix B on page 64. The model that defines our probability distribution is

$$m_k \;=\; \mathcal{N}\big(f_k(x),\, \sigma_k^2\big) \tag{32}$$

The calculation of the code length $-\log p(s|m_k)$ is quite simple[8]. The optimal code length for s is the sum of the optimal code length of the n values. Since the expected code length of each value is $H(m_k)$, the expected code length of $-\log p(s|m_k)$ is

$$\begin{aligned} -\log p(s|m_k) \;&=\; nH(m_k) \\ &=\; \frac{n}{2}\log\left(2\pi e \sigma_k^2\right) \\ &=\; \frac{n}{2}\log\sigma_k^2 + \frac{n}{2}\left(2\pi e\right) \end{aligned} \tag{34}$$

which lets us implement Rissanen's original version of MDL as

$$\begin{aligned} \mathcal{L}_{\text{Ris}}(s) \;&=\; \min_k \left[\frac{n}{2}\log\sigma_k^2 + \frac{n}{2}\log\left(2\pi e\right) + k\log\sqrt{n}\right] \\ &=\; \min_k \Big[n\log\sigma_k^2 + k\log n\Big] \end{aligned} \tag{35}$$

This, in fact, is very similar to the AIC.

Mixture MDL. In an as yet unpublished paper [BL02], Andrew Barron and Feng Liang base a novel implementation of mixture MDL on the minimax strategy. We will not reproduce their paper here. Instead, we will outline the practical consequences of their approach. They assume the prior distribution over the k-dimensional parameter space M_k to be uniform. Unfortunately, a uniform distribution over an infinite parameter space is not a proper distribution—it does not have measure one. For any uniform distribution with $p(m_k) > 0$ the integral will tend to infinity. To normalize, we would have to divide by infinity, which makes no sense.

To make the distribution proper, Barron and Liang condition it on a small initial subset $s' \subset s$ of the training set s. First, they calculate the probability $p(s'|k)$ with an improper prior distribution over m_k. Then they divide by the probability $p(s|k)$, according to the same improper prior distribution. Whatever impossible term would have normalized the two probabilities, the division cancels it out. The result of the division is sufficient to find the minimax model.

[8] To calculate the probability of the actual Euclidean distance $\sqrt{n\sigma^2}$ between $f_k(x)$ and any s of size n would not get us anywhere. The distribution over the normalized squared distance $\frac{n\sigma'^2}{\sigma^2}$ is chi-square with n degrees of freedom. For $\sigma'^2 = \sigma^2$ we get a constant value that depends on n but not on σ^2:

$$\chi_n^2(n) \;=\; \frac{1}{n!!\sqrt{2\pi}}\left(\frac{n}{e}\right)^{n/2} \tag{33}$$

(36) gives the complete minimax equation for mixture MDL according to Barron and Liang. The individual terms will be explained below. Because we calculate the code length, we take the negative logarithm of the probabilities and the division becomes an inverted subtraction: $\mathcal{L}_{BF}(\cdot)$

$$\begin{aligned}\mathcal{L}_{\mathrm{BF}}(s) \;=\; \min_k \Bigg[& \;\frac{n-k}{2}\log\left(n\sigma_n^2\right) \;+\; \frac{1}{2}\log|S_n| \;-\; \log\Gamma\left(\frac{n-k}{2}\right) \\ & -\left[\frac{m-d}{2}\log\left(n\sigma_m^2\right) \;+\; \frac{1}{2}\log|S_m| \;-\; \log\Gamma\left(\frac{m-k}{2}\right)\right]\Bigg]\end{aligned} \tag{36}$$

n is the size of the sample s and m is the size of the small initial subset $s' \subset s$, much smaller than n but bigger than k. $n\sigma_n^2$ is the squared Euclidean distance between the training sample $\{(x_1, y_1)\ldots(x_n, y_n)\}$ and the least square polynomial:

$$n\sigma_i^2 \;=\; \min_{f_k}\sum_{i=1}^{n}\left(y_i - f_k(x)\right)^2 \tag{37}$$

$|S_n|$ is the determinant of the Vandermonde matrix

$$\begin{bmatrix} n & \sum x & \sum x^2 & \ldots & \sum x^k \\ \sum x & \sum x^2 & \sum x^3 & \ldots & \sum x^{k+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum x^k & \sum x^{k+1} & \sum x^{k+2} & \ldots & \sum x^{2k} \end{bmatrix} \tag{38}$$

where $\sum x$ stands for $\sum_{i=1}^{n} x_i$. The calculation of a Vandermonde matrix and its determinant are discussed in Appendix 5 on page 64. The determinant of the Vandermonde matrix is the Fisher information of Section 1.4 on page 11 . It estimates the variance of the rate of change of $\ln p(m_k)$.

To prevent the Vandermonde matrix from becoming singular, the algorithm used in the experiments uses a subset $s' \subset s$ of size $m = 1.5k$. To keep m still smaller than n, the maximum number of parameters that is accepted is $k = \frac{1}{2}n$.

$\Gamma(x)$ is the gamma-function, an extension of the factorial to complex and real number arguments. It is a rest term of the calculations in [BL02] and will not be elaborated on here. For half integer arguments the gamma-function takes the special form

$$\Gamma\left(\frac{n}{2}\right) \;=\; \frac{(n-2)!!\sqrt{\pi}}{2^{(n-1)/2}} \tag{39}$$

where $n!!$ is the double factorial

$$n!! \;=\; \begin{cases} n\cdot(n-2)!! & n>1 \\ 1 & n\in\{0,1\} \end{cases} \tag{40}$$

Generalization analysis. The analysis of the generalization error is straightforward. For reasons explained in Section 2.1 on page 20 it is based on a test sample $t = \{(x_1, y_1) \dots (x_m, y_m)\}$ that follows the same distribution as the training sample. For a reliable analysis the size m of the test set has to be big, usually several thousand points. For each parameter size k the least square polynomial $f_k(x)$ is calculated from the training set. The generalization error e_k is the mean squared error $f_k(x)$t on the m points of the test set

$$e_k = \frac{1}{m} \sum_{i=1}^{m} \big(y_i - f(x_i)\big)^2 \tag{41}$$

Taking the mean squared error of the polynomial on the test set is *not* the only relevant aspect of generalization behaviour. The cost of a wrong prediction may instead be linear in the error, as is often the case in finances where we have to pay interest on lended money. It can be exponential as is often the case when we estimate computer running time. And it can have an absolute threshold as in the case of electric circuits that will burn when overloaded. The probabilistic model m_k provides us with a normal distribution over the error according to $\mathcal{N}\big(f(x), \sigma_k^2\big)$. As we saw earlier, the normal distribution lines up best with a quadratic cost function. But ultimately we want to be able to find the best model for any arbitrary cost function, and this is material for future research.

Cross-validation. To compare MDL with another successful method for data prediction we use cross-validation. Cross-validation is not based on complexity theory. It is basically a randomized algorithm that accepts a model if it has performed well on a sufficient number of independent trials. The version we use in the experiments splits the training sample randomly into 10 small partitions of equal size. If the training set contains 300 points then each partition will contain 30 points. For each of these 30 points we fit the polynomials on the remaining 270 points and then take the mean squared error of such a polynomial on our 30 points. This is similar to what we do in the generalization analysis, only that in this case the test sample is much smaller and the result is much more unreliable. But when we take the average generalization error on all of the 10 small partitions we get a good approximation of the real generalization analysis.

We could approximate the generalization analysis even better if we would use more than 10 partitions. But the algorithm will become much slower and the slight increase in performance does not justify this. 10 partitions give a result that is generally very good. The only problem with 10 partitions is that they show a tendency to randomly accept single models while rejecting all its neighbors. These single models depend on the particular distribution of the points over the partitions and do not reflect the facts. It can be cured by taking the average over adjacent models in the graph. This is called smoothing and is a very common practice in signal processing. The smoothing algorithm used in the experiments takes the average over only three adjacent models which seems to be quite enough. The complete algorithm is $\mathcal{L}_{cv}(\cdot)$

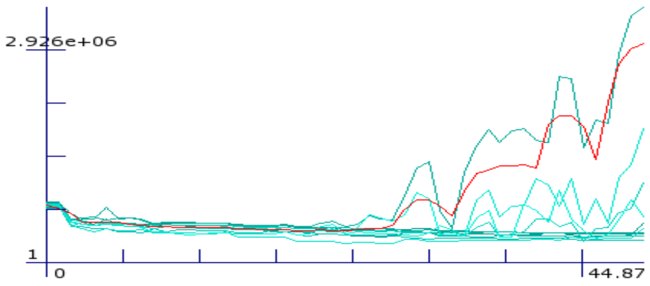
$$\mathcal{L}_{\text{CV}}(s) = \min_k \left[\frac{1}{3} \sum_{i=k-1}^{k+1} v(i) \right] \quad (42)$$

which calculates the moving average of $v(k)$, the average of the generalization error e_i of the individual partitions i. To allow smoothing over all values of k the algorithm is modified for $k = 0$ and $k = max$ to take the moving average over only two values. $v(k)$ is the average over the ten partitions

$$v(k) = \frac{1}{10} \sum_{i=1}^{10} e_i \quad (43)$$

e_i is defined in the same way as for the generalization analysis. Figures 3 gives an example of cross-validation.

Figure 3: Cross-validation



The x-axis shows the number of parameters. The y-axis shows the predicted generalization error. The predictions of each of the ten partitions are visible in shades of green. The smoothed average is visible in red. If you are reading a black and white print the red line should be darker than the green lines. It is the second line from above.

2.3 A simple experiment

To make you familiar with the *Statistical Data Viewer* and to introduce you to some of the basic problems of model selection, this section will take you through all the steps of a simple experiment.

The sinus wave is very common in nature and comparatively easy to model by a polynomial. To find out whether MDL is a useful method for data prediction we conduct our first experiment on a sinus wave of a single frequency.

Experiment 1: validity of MDL

Hypothesis: MDL can minimize the generalization error.

Source: a sinus wave with frequency $f = 0.5$ and amplitude 1.

Noise: normal (Gaussian) with variance $\sigma^2 = 1$.

Support: uniform over the interval [0, 10].

Sample: three samples of 50, 150 and 300 points.

Test set: i.i.d., 3,000 points.

Figure 4 shows how the source signal is created in the application and Figure 5 shows how the samples are drawn from the source and how an experiment for model selection is started.

50 points. A sinus wave with $f = 0.5$ shows 10 alternating peaks and valleys over the interval [0, 10]. An n-degree polynomial can have at the most $n - 1$ alternating peaks and valleys. We therefore expect polynomials with a low generalization error to be of at least 11 degrees.

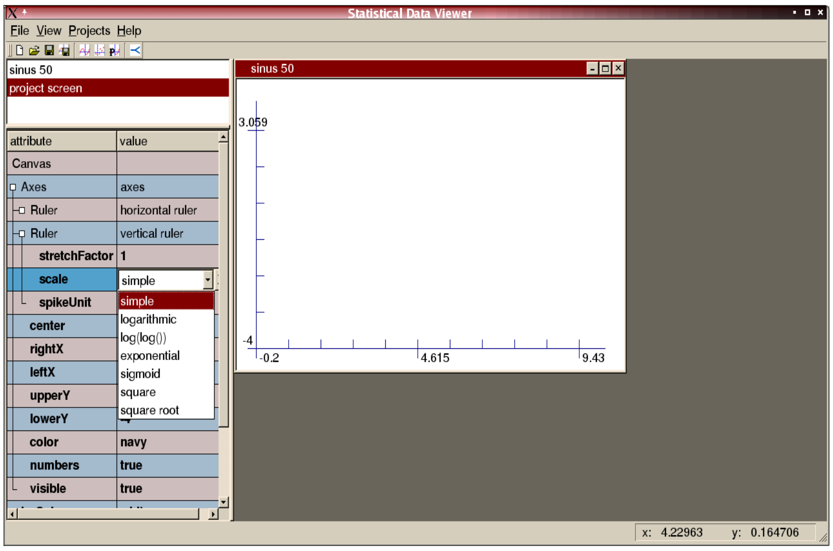
Figure 6 shows that a training set of 50 points is too small to capture this threshold. 11 degrees do not show a local decrease in generalization error. On the contrary, from this point on the generalization error starts to get really bad. The best generalization error is achieved for the 0-degree polynomial, which is nothing but the mean of the training sample. For degree 0–10 the generalization error is almost as good and never worse than two times the optimum. But for 11 degrees the expected error rises to ten times the optimum and then increases so fast that the LOG-LOG scale is needed to make the result readable.

All three methods agree that the 0-degree polynomial is optimal. Cross-validation even predicts correctly that exactly from 11 degrees onwards the generalization error will get very bad.

150 points. With 150 points the generalization analysis shows that the optimum lies at 17 degrees. The error between 14 and 18 degrees is less than half way between the optimum and the error of the 0-degree polynomial. From 21 degrees onwards the generalization error is never again lower than that of a 0-degree polynomial. After that it increases so rapidly that the LOG-LOG scale has to be used again.

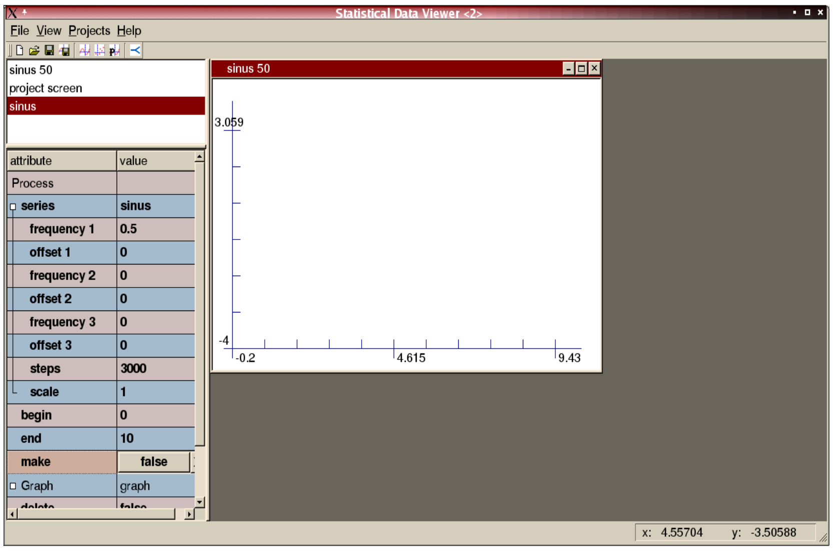
Mixture MDL and Rissanen's original version give a good picture of the situation. Both choose 15 degrees as the optimum and predict a low generalization error for models between 14 and 18 degrees. Cross-validation is a disappointment. It shows essentially the same picture as with 50 points: almost constant generalization error for 0–10 degrees, optimum for 0 degrees and a very high generalization error beyond 10 degrees.

Figure 4: Creating the source signal



attribute	value
Canvas	
Axes	axes
Ruler	horizontal ruler
Ruler	vertical ruler
stretchFactor	1
scale	simple
spikeUnit	
center	
rightX	
leftX	
upperY	
lowerY	
color	navy
numbers	true
visible	true

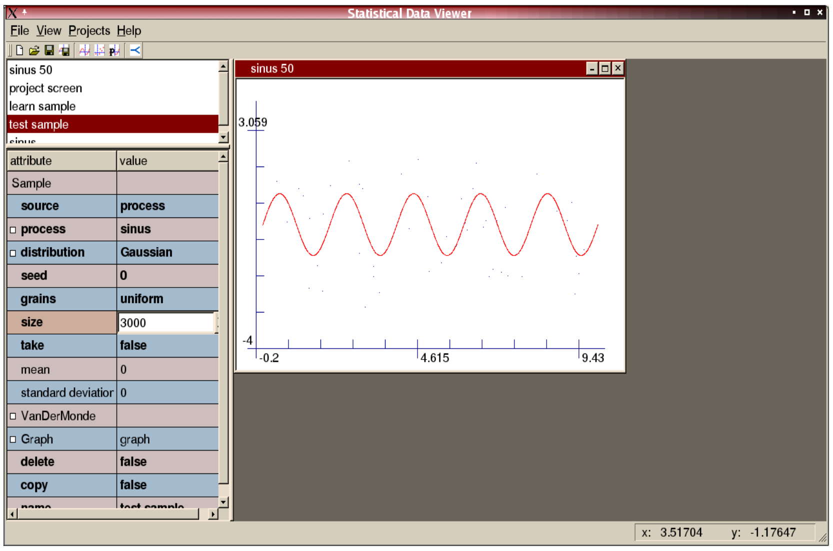
Clicking on the NEW PROJECT icon has created a new project. The project name can be defined and the attributes of the main plot can be set, e.g. range, size, scale, color and the origin of the rulers. The list view makes it possible to organize the attributes into a comprehensive hierarchy and to access them easily.



attribute	value
Process	
series	sinus
frequency 1	0.5
offset 1	0
frequency 2	0
offset 2	0
frequency 3	0
offset 3	0
steps	3000
scale	1
begin	0
end	10
make	false
Graph	graph

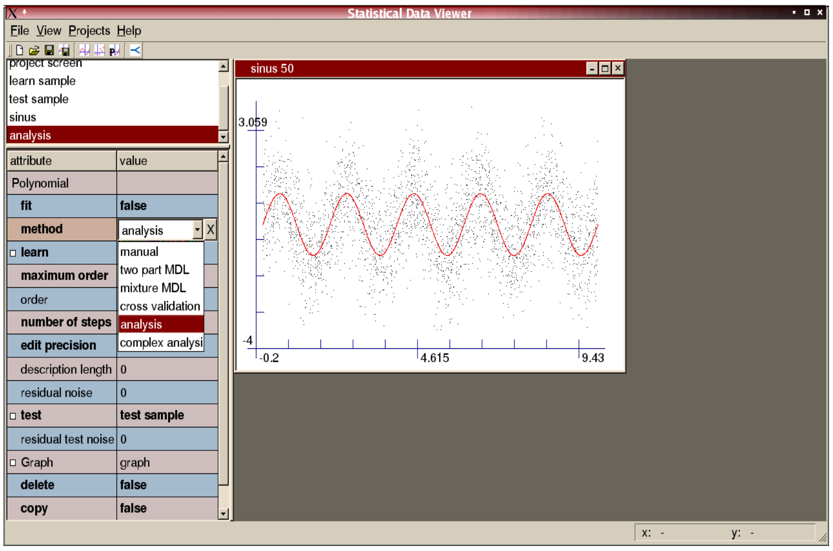
Clicking on the PROCESS icon has loaded a new process into the editor. From the available processes the sinus wave has been selected and the attributes were set: one frequency of $f = 0.5$ and support interval $[0, 10]$. A click on the MAKE button in the editor has created the signal.

Figure 5: Creating the samples and starting an experiment



attribute	value
Sample	
source	process
process	sinus
distribution	Gaussian
seed	0
grains	uniform
size	3000
take	false
mean	0
standard deviatior	0
VanDerMonde	
Graph	graph
delete	false
copy	false
name	test sample

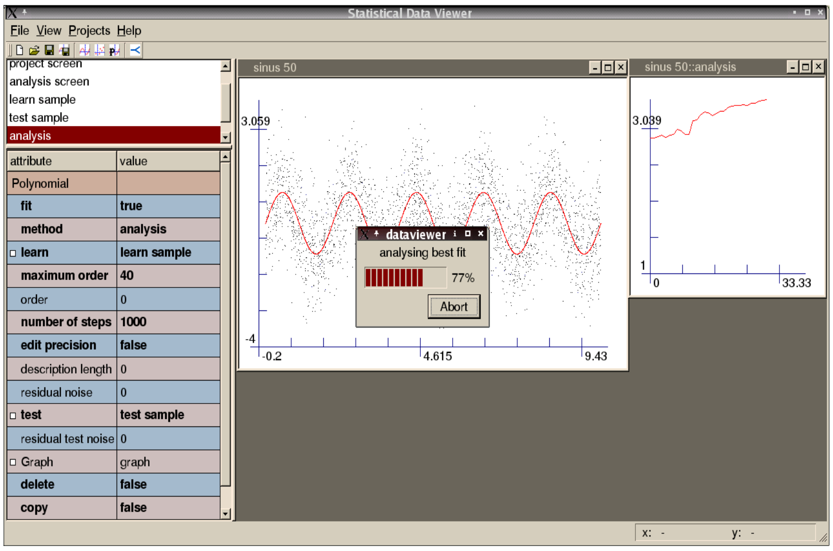
Clicking on the SAMPLE icon loads a new sample into the editor. A sample of 50 points has already been created and is visible on the main plot. Now the test sample is being defined: source and noise are specified and the size is entered. Clicking on the TAKE button in the editor will draw the sample.



attribute	value
Polynomial	
fit	false
method	analysis
learn	
maximum order	
order	
number of steps	
edit precision	
description length	0
residual noise	0
test	test sample
residual test noise	0
Graph	graph
delete	false
copy	false

Clicking on the POLYNOMIAL icon has loaded a new polynomial into the editor. The training and the test sample are already specified and now the method for model selection is chosen from the pull-down menu. After the maximum order for the selection process is entered, a click on the FIT button in the editor will start the experiment.

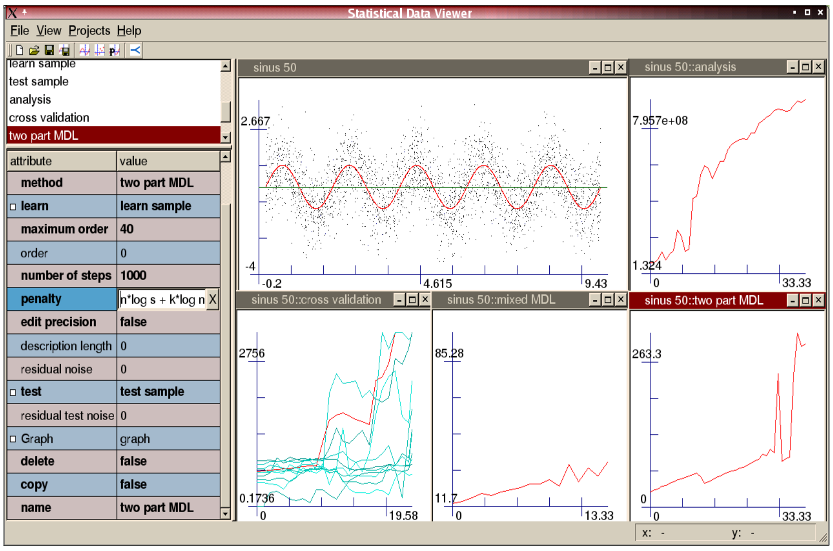
Figure 6: Sinus wave, witnessed by 50 points



analysing best fit 77%

attribute	value
Polynomial	
fit	true
method	analysis
learn	learn sample
maximum order	40
order	0
number of steps	1000
edit precision	false
description length	0
residual noise	0
test	test sample
residual test noise	0
Graph	graph
delete	false
copy	false

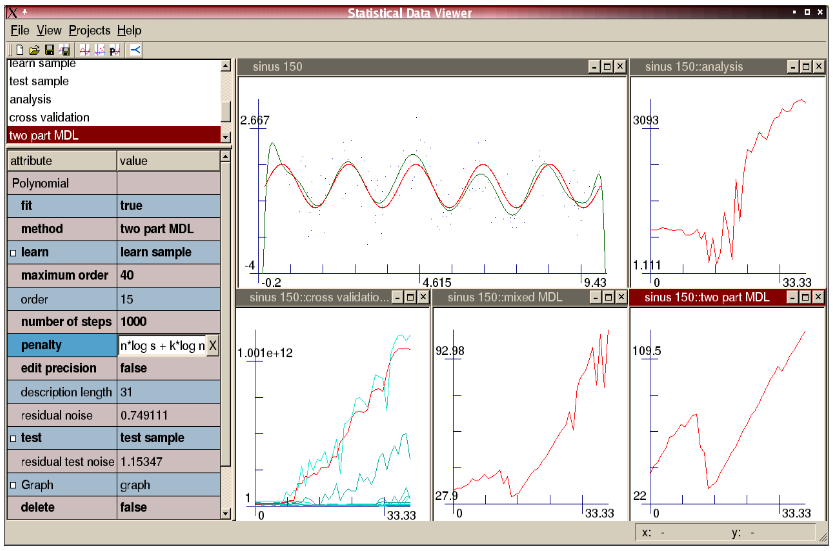
The generalization analysis is started: all polynomials in the range 0–40 degrees are fitted to the 50 training points and then checked against the 3,000 test points. The results are immediately plotted. The progress dialog allows to cancel the execution if it takes too long or if the early results call for a different design of the experiment.



attribute	value
method	two part MDL
learn	learn sample
maximum order	40
order	0
number of steps	1000
penalty	n*log s + k*log n
edit precision	false
description length	0
residual noise	0
test	test sample
residual test noise	0
Graph	graph
delete	false
copy	false
name	two part MDL

4 experiments have been conducted on the 50 point sample and the results were plotted into separate plots. The generalization analysis in the upper right shows that the 0-degree polynomial is the best choice. In the lower part from left to right are the three results from cross-validation, mixture MDL and Rissanen's original two-part MDL. All agree with the analysis that zero degrees is the best choice.

Figure 7: Sinus wave, witnessed by 150 and 300 points

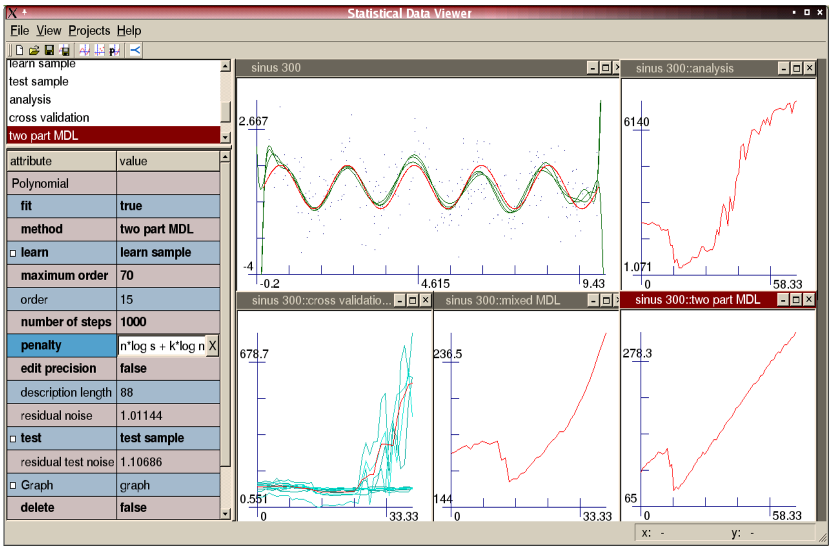


learn sample
test sample
analysis
cross validation
two part MDL

attribute	value
Polynomial	
fit	true
method	two part MDL
learn	learn sample
maximum order	40
order	15
number of steps	1000
penalty	n*log s + k*log n
edit precision	false
description length	31
residual noise	0.749111
test	test sample
residual test noise	1.15347
Graph	graph
delete	false

150 point training sample. The generalization analysis shows 17 degrees as the optimum. Mixture MDL and Rissanen's original version slightly underfit and choose 15 as the optimum.

Cross-validation shows essentially the same picture as with 50 points. Anything from 0 to 10 degrees is good but 0 degrees is best.



learn sample
test sample
analysis
cross validation
two part MDL

attribute	value
Polynomial	
fit	true
method	two part MDL
learn	learn sample
maximum order	70
order	15
number of steps	1000
penalty	n*log s + k*log n
edit precision	false
description length	88
residual noise	1.01144
test	test sample
residual test noise	1.10686
Graph	graph
delete	false

300 point training sample. The generalization analysis now shows 18 degrees as optimal. 15-25 degrees are less than 5 percent worse than the optimum error. From 40 degrees on the generalization error increases rapidly.

All methods now generalize well. Rissanen's original version and mixture MDL choose 15 degrees, cross-validation 20 degrees.

300 points. With 300 points we see all methods generalizing well. The generalization analysis now shows 18 degrees as optimal. 13–33 degrees are less than half way between a 0-degree polynomial and the optimum. From 40 degrees on the generalization error increases rapidly. Rissanen's version chooses 17 degrees, mixture MDL 15 degrees and cross-validation 20 degrees, all of which are less than 10 percent worse than the optimum.

The three methods not only make a good choice, they also capture the general outline of the generalization analysis. Polynomials of degree 15–25 are shown as good candidates by all the three methods. They also show that anything from 35 onward is worse than taking the 0-degree polynomial.

Conclusion. Our simple experiment has shown that both our versions of MDL are good methods for data prediction and error minimization. When compared to cross-validation they can be called at least of equal strength.

2.4 Critical points

Concluding that a method does or does not select a model close to the optimum is not enough for evaluating that method. It may be that the selected model is many degrees away from the real optimum but still has a low generalization error. Or it can be very close to the optimum but only one degree away from overfitting, making it a risky choice. A good method should have a low generalization error *and* be a save distance away from the models that overfit.

How a method evaluates models other than the optimum model is also important. To safeguard against any chance of overfitting we may want to be on the safe side and choose the lowest degree model that has an acceptable generalization error. This requires an accurate estimate of the per model performance. We may also combine several methods for model selection and select the model that is best on average. This too requires reliable per model estimates. And not only the generalization error can play a role in model selection. Speed of computation and memory consumption may also constrain the model complexity. To calculate the best trade off between algorithmic advantages and generalization error we also need accurate per model performance.

When looking at the example we observe a number of critical points that can help us to evaluate a method:

the origin: the generalization error when choosing the simplest model possible. When speaking of polynomials this is the expected mean of y ignoring x.

the initial region: it may contain a local maximum slightly worse than the origin or a plateau where the generalization error is almost constant.

the region of good generalization: the region that surrounds the optimum and where models perform better than half way between origin and optimum.

Often the region of good generalization is visible in the generalization analysis as a basin with sharp increase and decrease at its borders and a flat plateau in the center where a number of competing minima are located.

the optimum model: the minimum within the region of good generalization.

false minima: models that show a single low generalization error but lie outside or at the very edge of the region of good generalization.

overfitting: from a certain number of degrees on all models have a generalization error worse than the origin. This is where overfitting really starts.

Let us give some more details about three important features:

Region of good generalization. The definition of the region of good generalization as better than half way between origin and optimum needs some explanation. Taking an absolute measure is useless as the error can be of any magnitude. A relative measure is also useless because while in one case origin and optimum differ only by 5 percent with many models in between, in another case even the immediate neighbors might show an error two times worse than the optimum but still much better than the origin.

Better than half way between origin and optimum may seem a rather weak constraint. With big enough samples all methods might eventually agree on this region and it may become obsolete. But we are primarily concerned with small samples. And as a rule of thumb, a method that cannot fulfill a weak constraint will be bad on stronger constraints as well.

False minima. Another point that deserves attention are the false minima. Both analysis plots in Figure 7 feature them and they are very prominent in Figure 1 on page 3. While different samples of the same size will generally agree on the generalization error at the origin, the initial region, the region of good generalization and the region of real overfitting, the false minima will change place, disappear and pop up again at varying amplitudes. They can even outperform the real optimum. The reason for this can lie in rounding errors during calculations and in the random selection of points for the training set. And even though the training sample might miss important features of the source due to its restricted size, the model might hit on them by sheer luck, thus producing an exceptional minimum.

Cross-validation particularly suffers from false minima and has to be smoothed before being useful. Taking the mean over three adjacent values has shown to be a sufficient cure. Both versions of MDL seem to be rather free of it.

Point of real overfitting. The point where overfitting starts also needs some explanation. It is tempting to define every model larger than the optimum model as overfitted and indeed, this is often done. But such a definition creates a number of problems. First, the global optimum is often contained in a large basin with several other local optima of almost equal generalization error. Although we assume that the training sample carries reliable information on the general outline of generalization error, we have no evidence that it carries information on the exact quality of each individual model. It would be wrong to judge a method as overfitting because it selected a model 20 degrees too high but of a low generalization error if we have no indication that the training sample actually contained that information. On the other hand, at the point on the x-axis where the generalization becomes forever worse than at the origin the generalization error usually shows a sharp increase. From this point on differences are measured in orders of magnitude and not in percent which makes it a much clearer boundary. Also, if smoothing is applied, different forms of smoothing will favor different models within the region of good generalization while they have little effect on the location of the point where the generalization error gets forever worse. And finally, even if a method systematically misses the real optimum, as long as it consistently selects a model well within the region of good generalization of the generalization analysis it will lead to good results. But selecting or even encouraging a model beyond the point where the error gets forever worse than at the origin is definitely unacceptable.

3 Selected experiments

This section takes you through some selected experiments on two dimensional regression problems. They were selected for broadness, not for completeness. It is impossible to cover the whole range of possible experiments on a few pages. If you are interested, you can download the application and do some experiments of your own.

3.1 A hard problem: the step function

While the sinus wave is easy to model by a polynomial, the step function is particularly hard. Due to its nature, the step function is hard to model by any smooth function as everyone can tell who has ever built a filter for radio or sound signals. No polynomial can exactly match it and convergence for high degree models is very slow. With slow convergence we mean that over all the models in the range $k \in [0, max]$ the mean squared error σ^2 on the training set improves only slowly. But even though slow convergence requires a high complexity of the model to get a somewhat acceptable generalization error, it does not justify a method to overfit. See Appendix 5 on page 62 for details on the step function in use.

Experiment 2: the step function

Hypothesis: MDL can minimize the generalization error even when the source is hard to model.

Source: a step function with 10 random alternations within the interval [0, 10] and amplitude 10.

Noise: normal with variance $\sigma^2 = 1$.

Support: uniform over the interval [0, 10].

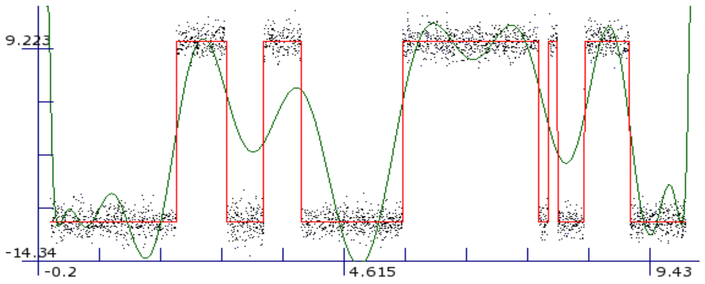
Sample: two samples of 100 and 400 points.

Test set: i.i.d., 3,000 points.

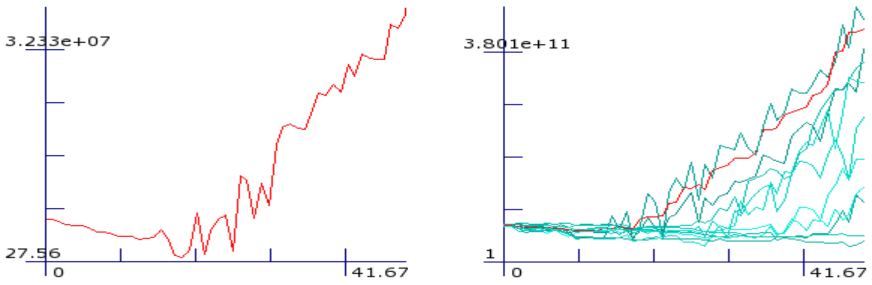
100 points. In Figure 8 our three methods are tried on a training sample of 100 points. Given the 10 alternations in the step function this is extremely small and we do not expect any method to come close to the optimum. Nevertheless, selecting a model from the region of overfitting is unacceptable and must be called a failure.

The generalization analysis shows that the 19-degree polynomial achieves the best generalization error on the 3,000 point test sample. Overfitting starts at 26 degrees and good generalization (better than half way between the origin and the optimum) is achieved for degree 7–20. Cross-validation predicts an 11-degree polynomial which is well within this region. Its region of good generalization is

Figure 8: Step function, witnessed by 100 points

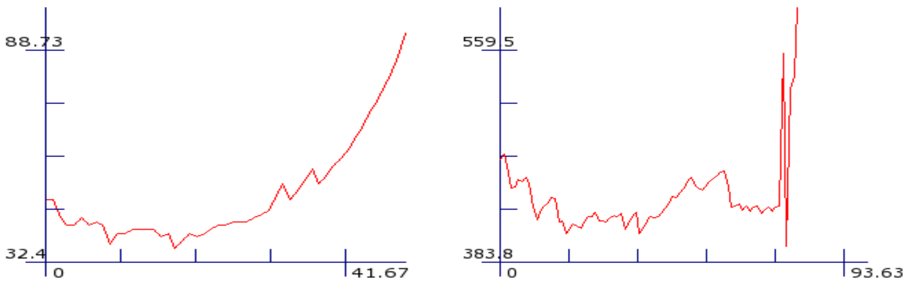


Step function with 10 alternations, 100 point training sample, 3,000 point test sample and the correct 19-degree polynomial.



Left: analysis for degree 0–70, trained on the 100 point training sample and checked against the 3,000 point i.i.d. test sample. The y-axis has LOG-LOG scale. The optimum is found at 19 degrees, good generalization in the range 7–20 and overfitting from 26 degrees onwards.

Right: cross-validation predicts the optimum at 11, good generalization in the range 4–15 and overfitting from 18 degrees onwards.



Left: mixture MDL. Predicts optimum at 18 degrees, good generalization in the range 8–25 and overfitting from 34 degrees onward.

Right: Rissanen's original version predicts the optimum at 18 degrees although another minimum at 38 degrees is only marginally weaker. The global minimum at 78 degrees is a false minimum. Predicts good generalization for the range 9–48 and overfitting from 79 degrees onwards.

4–15 degrees which is a good approximation of the generalization analysis. It does not include points that overfit.

Both versions of MDL predict the optimum at 18 degrees, though Rissanen's original version was very close to predicting a 38 degree model. Rissanen's region of good performance is 9–48 and includes many models that actually overfit. The point were it itself predicts overfitting is ridiculously high: 79 degrees. The reason for this might be the slow convergence rate. For many degrees the polynomials make slow but continuous progress in minimizing the empirical error, almost in direct proportion with the number of parameters. Rissanen's original penalty term is linear in the number of parameters and therefore cannot counter such a steady improvement.

Mixture MDL predicts good generalization for degree 8–25 and overfitting from 24 degrees onwards. This comes remarkably close to the results of the generalization analysis. Repeated execution of the same experiment shows that mixture MDL generally gives a good copy of the generalization analysis but occasionally falls prey to false minima. On one occasion it selected a model of degree 3, another time one of degree 28. The graph in Figure 8 shows a local minimum at degree 9, well within the region of good generalization. In the range between origin and optimum it misses the optimum only by 8 percent and could easily have outperformed it.

400 points. Figure 9 shows our three methods on a training sample of 400 points. It basically confirms the previous findings.

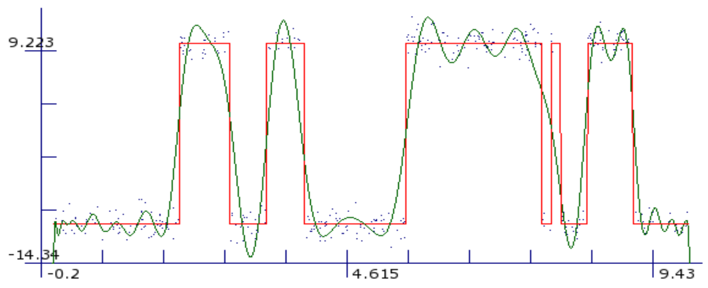
The generalization analysis promotes 44 degrees as the correct model, a high degree given the sample size, which reflects the slow convergence of the models with the step function. Cross-validation shows that the sample contains most of the essential information. It predicts an optimum that is only marginally worse than the correct model and which lies well within the region of good generalization. Its region of good generalization is bigger than that of the generalization analysis but does not include points that overfit.

Rissanen's original version gives a very bad prediction for the optimum: 89 degrees. And anything from 17 to 150 degrees is a good model. For high degrees it does not even show a tendency to the worse. If MDL is a valid theory for data prediction than Rissanen's original assumptions must be wrong. Clearly, the number of parameters and the log sample size alone are not enough to calculate model complexity.

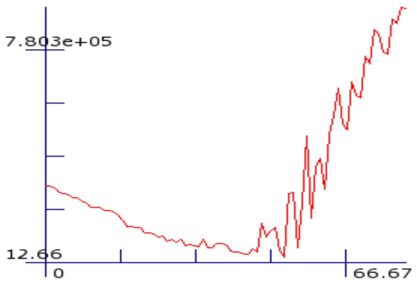
Mixture MDL shows that MDL is indeed capable of predicting a good model even under adverse conditions. Over a number of experiments on samples of the same size it consistently predicts models that perform even better than those predicted by cross-validation. It also consistently includes points that overfit in its region of good generalization. Nevertheless, the improvement against Rissanen's original version is remarkable and vindicates MDL as a valid method for data prediction.

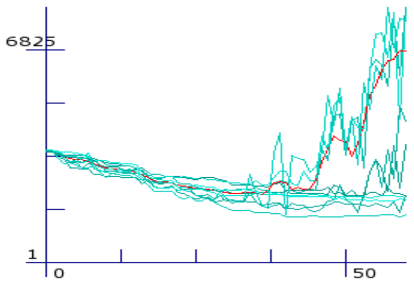
Conclusion. Rissanen's original version evidently is not capable to cope with slow convergence. Of course, letting the penalty term rise faster than linearly in

Figure 9: Step function, witnessed by 400 points



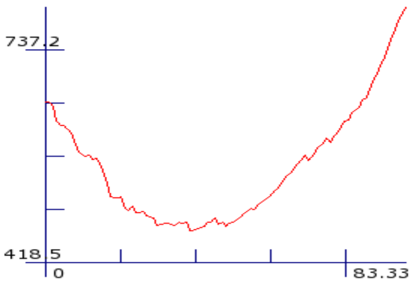
Step function with 10 alternations, 400 point training sample and the correct 44-degree polynomial. The error on the test sample now is less than half that of the 19-degree model, which was the correct model for the 100 point training sample. As the models converge very slowly, even with such a high degree polynomial there is evidently still much room for improvement.

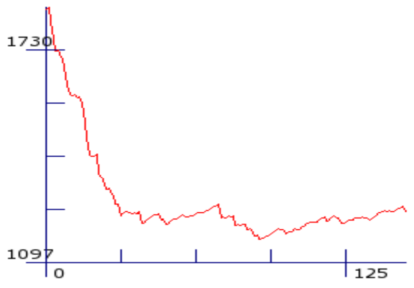




Left: generalization analysis for degree 0–80. Predicts the optimum at 44 degrees, good generalization in the range 9–37 and overfitting from 54 degrees onwards. A false minimum at 53 degrees slightly outperforms the optimal model.

Right: cross-validation for the same problem. Predicts the optimum at 36, good generalization in the range 9–45 and overfitting from 52 degrees onwards.

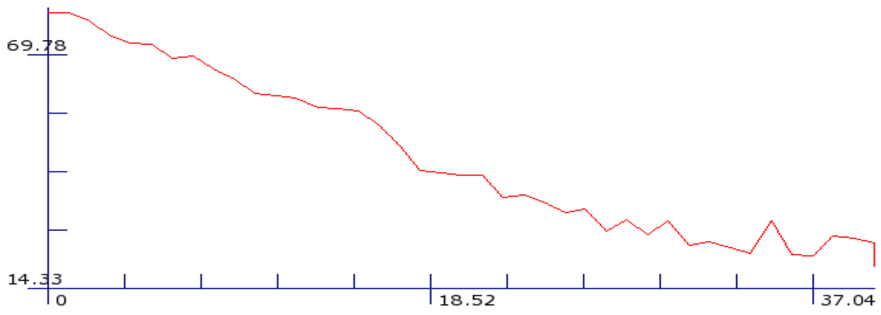




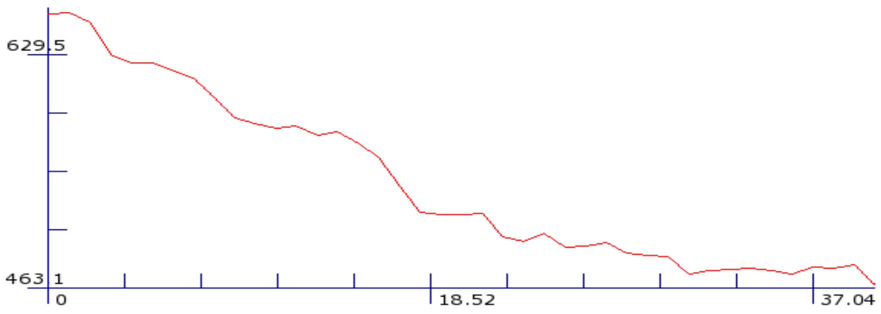
Left: mixture MDL predicts the optimum at 40 degrees, good generalization in the range 16–69 and overfitting from 87 degrees onwards.

Right: Rissanen's original version for the range 0–150 predicts good generalization anywhere from 17 degrees onwards and 89 degrees as the optimum. No prediction of overfitting.

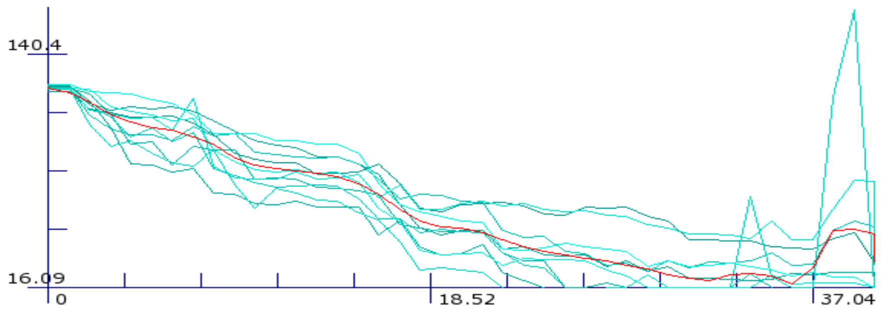
Figure 10: Fine details of Figure 8



generalization analysis for 0–40 degrees



mixture MDL for 0–40 degrees



cross-validation for 0–40 degrees

All plots are magnifications from Figure 8 on page 35 for the range 0–40 degrees. Especially mixture MDL gives a detailed copy of the generalization analysis.

Experiments over a wide range of samples and sources show that if the sample contains detailed information on the source, it is more likely that mixture MDL will reproduce it than cross-validation. Also, the quality of the mixture MDL reproductions is usually better.

the number of parameters will improve it and we are free to fine tune the penalty term on the problem. But the optimal penalty on one family of problems might prove to be a catastrophe on the next one. The application allows the user to enter his own penalty term and we strongly encourage you to try to find a penalty term that performs well on *all* sorts of problems. You will not succeed.

Mixture MDL shows that a theoretically sound estimate of the needed model complexity is a better strategy than finding the proper penalty term by trial and error. Its performance on this difficult problem almost equals that of cross-validation, especially when the training sample is very small, and we cannot reasonably expect more.

Additional findings. A close look at the different graphs shows that for models of complexity lower than the optimum both mixture MDL and cross-validation can give a very detailed copy of the structure of the generalization analysis. Figure 10 magnifies the graphs of the generalization analysis, mixture MDL and cross-validation each in the range 0–40 degrees. Although at this time we are primarily concerned with the performance of MDL at the critical values, it is very interesting to observe how much information the sample actually contains about the source and how much of this information can be revealed by MDL. As a general finding over many different trials, mixture MDL gives the most detailed copy of the generalization analysis.

3.2 Sparse support

To further explore the performance of MDL we consider a problem that is particularly disastrous to polynomials: sparse representation of the support set. Experience teaches that the sample mean is the best estimate for regions of the support set which are sparsely represented in the training sample, regardless of the problem at hand. Polynomials tend to do the opposite, at least at the far ends of the support set: they go to extreme values as soon as the function has passed the last point in the training sample. And the higher the degree of the polynomial the faster it goes to extreme values. This behavior leads to a very large generalization error on any test point that might show up beyond the extreme points of the training sample.

To provide non-uniform distributions over the support set the application allows to concentrate the data around a number of points on the x-axis. In the application they are called grains. The distribution around these points is normal. When there is one grain, most points will be in the center of the support set and the far ends will be represented very sparsely. If more than one grain is used they are positioned equally far from each other and from the edges of the support set such that there are sparse regions both at the edges and between the grains. The test sample will be subjected to the same distribution over the support as the training sample.

Experiment 3: bad support

Hypothesis: MDL can deal with non-uniform distributions over the support interval.

Source: Lorenz attractor

Noise: normal (Gaussian) with variance $\sigma^2 = 1$.

Support: Within the total support interval [0, 10] the distribution is normal around 3 centers with $\sigma^2 = 0.83$. They are situated at $x = \{2.5, 5, 7.5\}$.

Sample: one sample of 300 points.

Test set: i.i.d., 3,000 points.

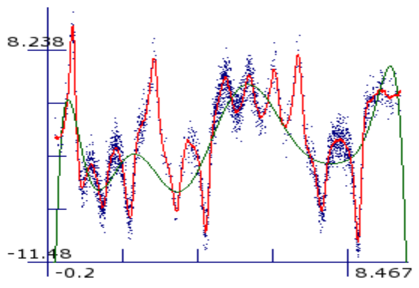
The source will be a time series that is generated by a Lorenz attractor. The Lorenz attractor as a function has nothing in common with polynomials. Nevertheless, unlike the step function, the resulting graph is remarkably smooth and can be approximated reasonably well by a polynomial of high degree. Experiments with a uniform distribution over the support show that for a 300 point training sample the correct model for the attractor used in the experiment is about 35 degrees. This does not increase much with larger training samples.

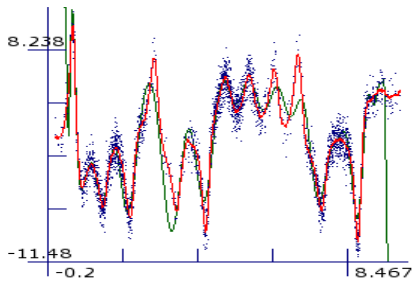
The analysis in Figure 11 shows that with a non-uniform support the correct model is only a fraction of that of a uniform support: 10 degrees instead of 35 degrees. Cross-validation continues to give excellent predictions and proves that the sample contains all the relevant information. Both versions of MDL fail completely. While the analysis shows that overfitting starts at 19 degrees, mixture MDL predicts 24–54 degrees as good, 35 degrees as the optimum and lets overfitting start at 72 degrees. Rissanen's version is even worse, predicting 39 as the optimum, 20–138 as good and does not show any sign of overfitting lower than 150 degrees, which was the maximum degree looked at.

To be fair, in a similar experiment conducted on the easier sinus wave but with the same non-uniform distribution over the support, mixture MDL gave correct predictions. But the Lorenz attractor cannot be considered an exceptionally hard problem and cross-validation proves that all relevant information is present. MDL never gave correct predictions on this problem, even with training samples as large as 1,000 points.

Conclusion. Bad support is a particular hard problem for polynomial models. Nevertheless, cross-validation shows us that the training sample contains all necessary information to correctly minimize the generalization error. MDL fails completely. When presented with the same Lorenz attractor but a uniform distribution over the support, mixture MDL predicts essentially the same as when the support is non-uniform. For uniform distributions the predictions turn out to be correct. This shows that mixture MDL has a structural deficit in recognizing differences in the distribution over the support set.

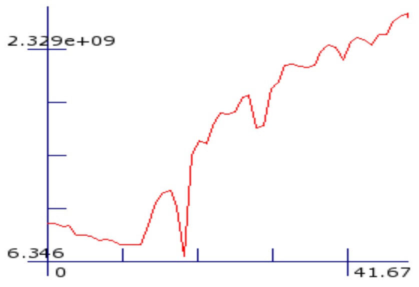
Figure 11: Lorenz attractor with sparse support

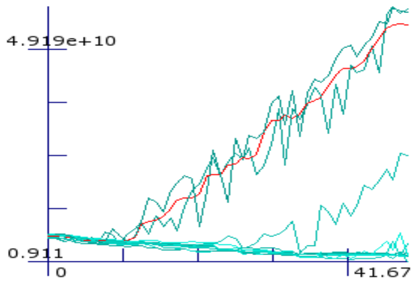




Left: Lorenz attractor, 300 point training sample, 3,000 point test sample and the correct 10-degree polynomial. The 3 concentrations of points are clearly visible.

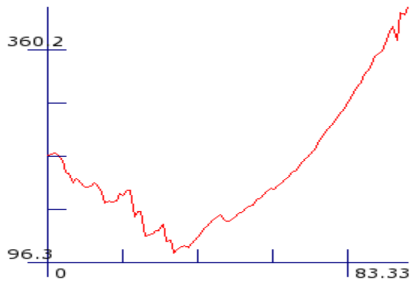
Right: Lorenz attractor and the wrong 35 degree model predicted by mixture MDL. The model completely misses the test points at the two edges.

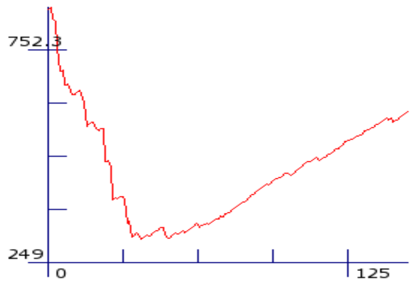




Left: analysis for degree 0–50, optimum at 10 degrees, good generalization in the range 4–13 and overfitting from 20 degrees onwards. There is a single false minimum at 19 degrees which outperforms the optimum by 30 percent.

Right: cross-validation for the same problem. Predicts the optimum at 11, good generalization in the range 4–11 and overfitting from 12 degrees onwards.





Left: mixture MDL predicts the optimum at 35 degrees, good generalization in the range 24–54 and overfitting from 72 degrees onwards.

Right: Rissanen's original version for degree 0–150, predicts the optimum at 39, good generalization in the range 20–138 and no overfitting.

3.3 Normalization

It is sometimes suggested to normalize the sample before applying the learning algorithm. The polynomial should not immediately grow to large values around the edges of a sample when the sample is normalized to the interval $[-1, 1]$. This should benefit our learning algorithm when the support is sparse at the edges of the support, says the suggestion.

Experiment 4: Normalization

Hypothesis: Normalizing the sample will improve the performance of MDL when the distribution over the support is non-uniform.

Source: Lorenz attractor

Noise: normal (Gaussian) with variance $\sigma^2 = 1$.

Support: The interval of the support is $[-1, 1]$. As in Experiment 3, the distribution is normal around 3 centers with $\sigma^2 = 0.83$. They are situated at $x = \{-0.667, 0, 0.667\}$.

Sample: one sample of 300 points.

Test set: i.i.d., 3,000 points.

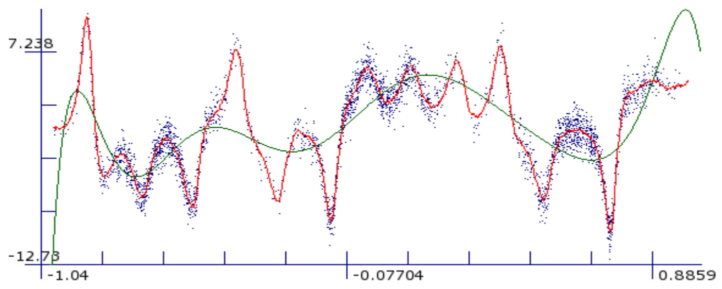
The conditions are the same as in Experiment 3 except for the support interval which is now normalized to $[-1, 1]$. The graphs in Figure 12 show only small changes when compared with Experiment 3 which resemble the usual small changes in the graphs due to the random choice of the points. These changes cannot be attributed to the normalization. All 3 methods predict about the same optima, the same regions of good generalization and the same points of overfitting as they did in Experiment 3. MDL in particular shows no improvement whatsoever.

Conclusion. Normalization does not improve the performance of MDL. It does not influence the generalization error at all. Polynomials are completely indifferent to normalization.

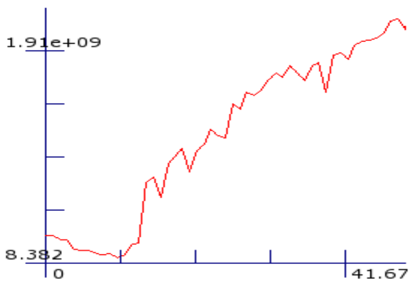
3.4 Different types of noise

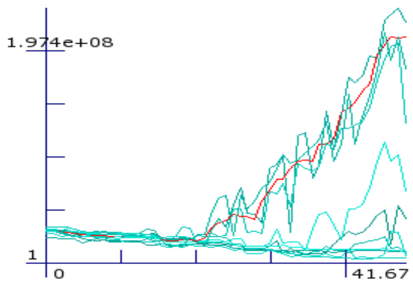
Not only the distribution over the support is part of a problem but also the noise that pollutes the signal. If the true distribution of the noise is unknown it is common practice to assume a Gaussian distribution. As explained in Section 1.5 on page 13, the mean squared error between the sample and the least square polynomial is taken as the variance of the distribution. The Gaussian distribution is the maximum entropy distribution for a given variance. It gives the lowest log likelihood to all its members of high probability and minimizes the effects of a wrong prediction.

Figure 12: Normalized Lorenz attractor with sparse support



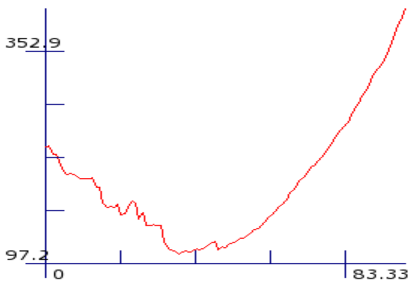
Lorenz attractor, 300 point training sample and 3,000 point test sample concentrated around 3 centers and the correct 10-degree polynomial. The points on the x-axis lie in the interval $[-1, 1]$.

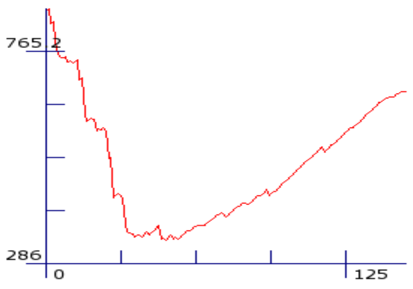




Left: analysis for degree 0–50, optimum at 10 degrees, good generalization in the range 4–11 and overfitting from 14 degrees onwards.

Right: cross-validation for the same problem. Predicts the optimum at 11, good generalization in the range 6–22 and overfitting from 23 degrees onwards.





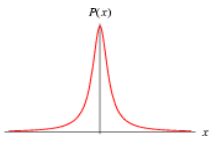
Left: mixture MDL predicts the optimum at 37 degrees, good generalization in the range 16–56 and overfitting from 79 degrees onwards.

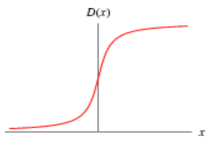
Right: Rissanen's original version for degree 0–150, predicts the optimum at 50, good generalization in the range 21–130 and no of overfitting.

How will MDL perform if the noise has a distribution around the source function that is not Gaussian? And what if the distribution of the noise has no calculable variance? Is is still safe to assume a Gaussian distribution over the noise? To test this, the application provides for a number of different distributions over the noise—the Gaussian (normal) distribution, the uniform distribution, the exponential distribution and the Cauchy distribution. Like the other distributions the Cauchy distribution is symmetric around the mean and non-increasing. But it has no calculable variance. Its density function is

Cauchy disribution

$$P(x) = \frac{s}{\pi\left((x-\mu)^2+s^2\right)}\,, \tag{44}$$





where μ is the mean and s the distance between the mean and the point that has half the probability of the mean.

The Cauchy distribution is very helpful in studying the problem of far outliers, single points that show an exceptional large deviation from the sample mean. It is common practice to eliminate such outliers from the sample before applying other statistical methods. They usually do not contribute to the results but rather hinder the analysis. But in a multi-dimensional problem space where points are concentrated around an unknown attractor it might be impossible to detect the far outliers and we want a method for model selection to be able to take account of them.

Experiment 5: different types of noise

Hypothesis: MDL gives good predictions under different types of noise but might have problems with the Cauchy distribution which has no calculable variance.

Source: Thom map, which itself contains extreme values.

Noise: Normal, uniform and exponential with $\sigma^2 = 1$. Cauchy with $s = 0.03$ and $s = 0.1$.

Support: uniform over the interval [0, 10].

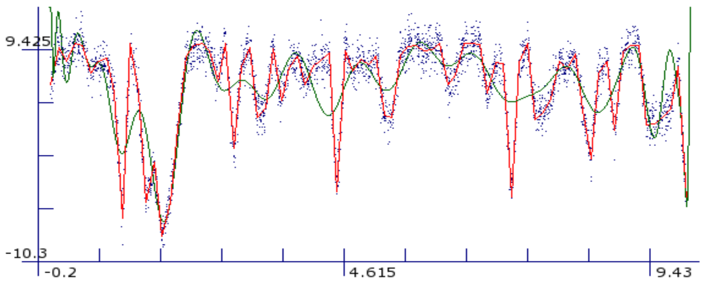
Sample: four samples of 300 points each.

Test set: one i.i.d. test set for every training sample, 3,000 points each.

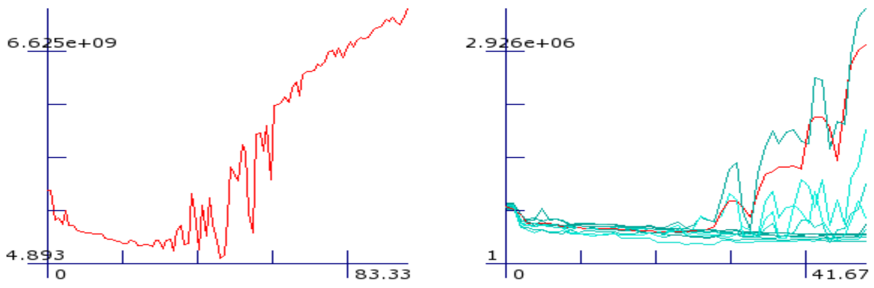
To make the problem realistic we will use a source that itself contains strong deviations from the mean: the Thom map[9]. The Thom map used in the experiments has 5 different regions that go to extreme values, four of which are

[9] See appendix 5 on page 63 for further details on this process.

Figure 13: Thom map and Gaussian noise

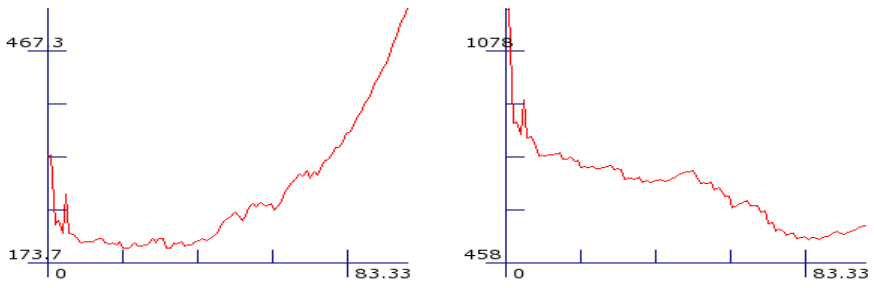


Thom map, 300 point training, 3,000 point test sample and the correct 35-degree polynomial. The points are drawn with Gaussian noise of $\sigma^2 = 1$,



Left: the analysis for degree 0–100 shows the optimum at 35 degrees, good generalization in the range 2–39 and overfitting from 57 degrees onwards. There is a significant global minimum at 49 degrees, outside of the region of good generalization.

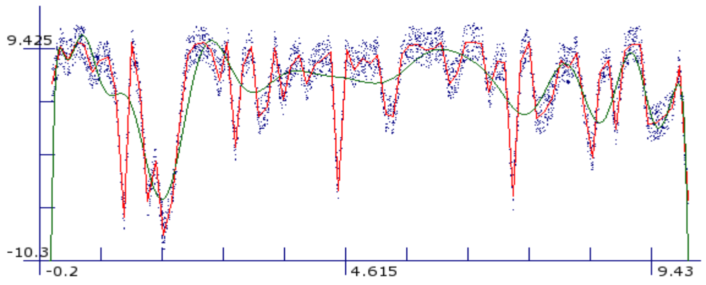
Right: cross-validation, optimum at 22, good generalization in the range 2–29 and overfitting from 35 degrees onwards.



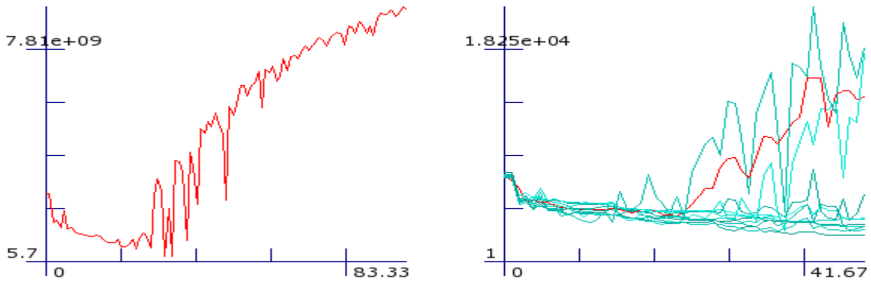
Left: mixture MDL predicts the optimum at 21, good generalization in the range 5–64 and overfitting from 78 degrees onwards.

Right: Rissanen's original version predicts the optimum at 84 degrees, good generalization anywhere from 6 degrees onwards and no overfitting.

Figure 14: Thom map and uniform noise

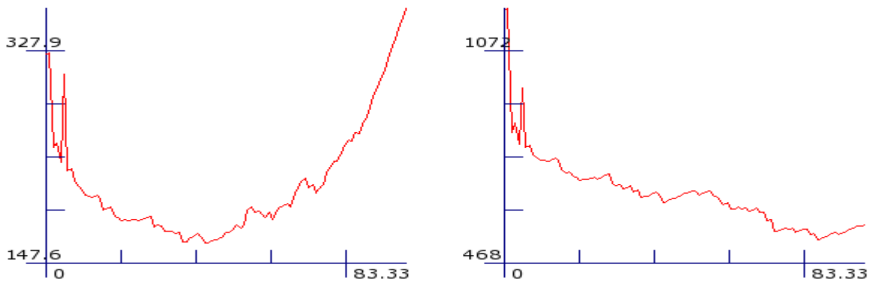


Thom map, 300 point training sample, 3,000 point test sample and the correct 25-degree polynomial. The noise is uniform with variance $\sigma^2 = 1$,



Left: analysis for degree 0–100, optimum at 25 degrees, good generalization in the range 2–29 and overfitting from 51 degrees onwards. Compared to the situation with Gaussian noise of the same variance the critical points are lowered by about 10 degrees.

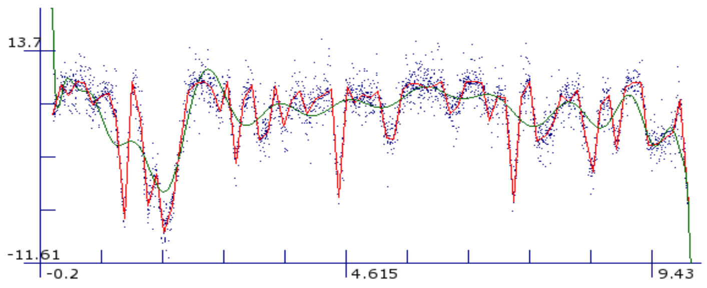
Right: cross-validation, optimum at 24, good generalization in the range 2–27 and overfitting from 30 degrees onwards.



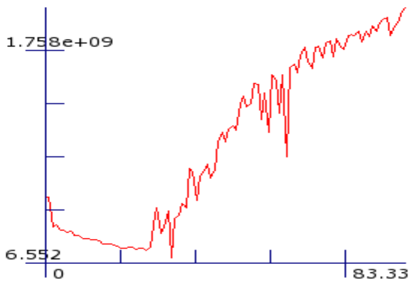
Left: mixture MDL predicts the optimum at 38 degrees, good generalization in the range 6–66 and overfitting from 80 degrees onwards.

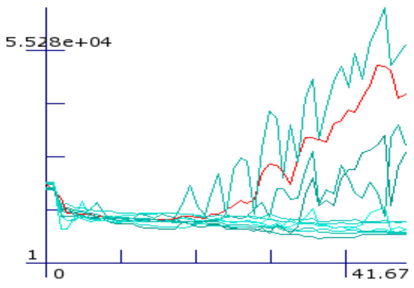
Right: Rissanen's original version predicts an optimum at 91 degrees, good generalization anywhere from 6 degrees onwards and no overfitting.

Figure 15: Thom map and exponential noise



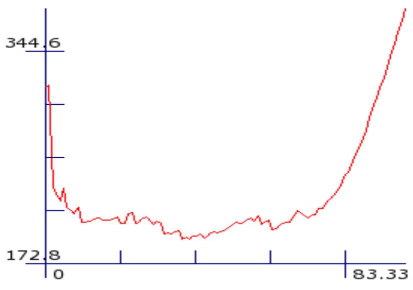
Thom map, 300 point training and 3,000 point test sample and the correct 30-degree polynomial. The noise is exponential with variance $\sigma^2 = 1$,

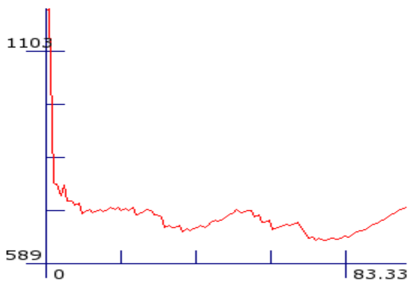




Left: analysis for degree 0–100, optimum at 30 degrees, good generalization in the range 2–30 and overfitting from 43 degrees onwards, very similar to the uniform distribution.

Right: cross-validation, optimum at 17 degrees, good generalization in the range 2–29 and overfitting from 30 degrees onwards.

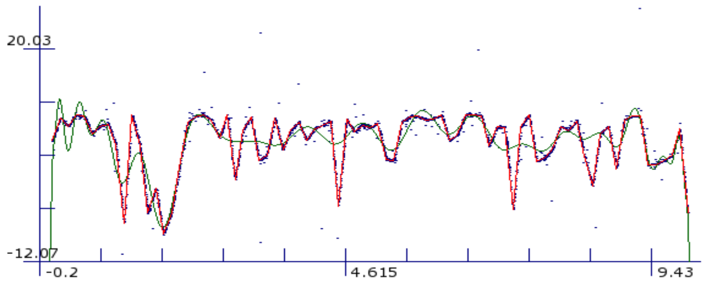




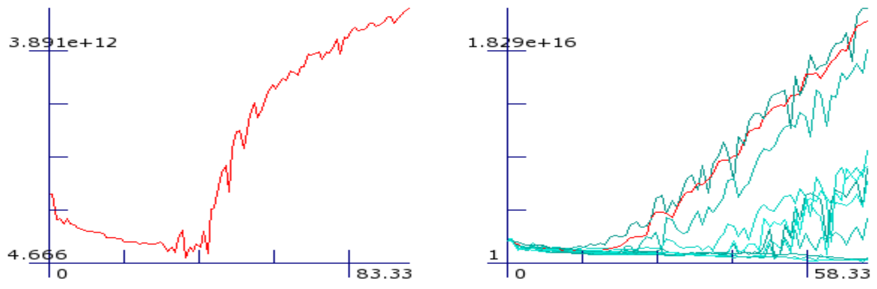
Left: mixture MDL predicts the optimum at 38 degrees, good generalization in the range 2–85 and overfitting from 93 degrees onwards.

Right: Rissanen's original version predicts the optimum at 75 degrees, good generalization from 2 degrees onwards and not overfitting.

Figure 16: Thom map and Cauchy noise of $s = 0.03$

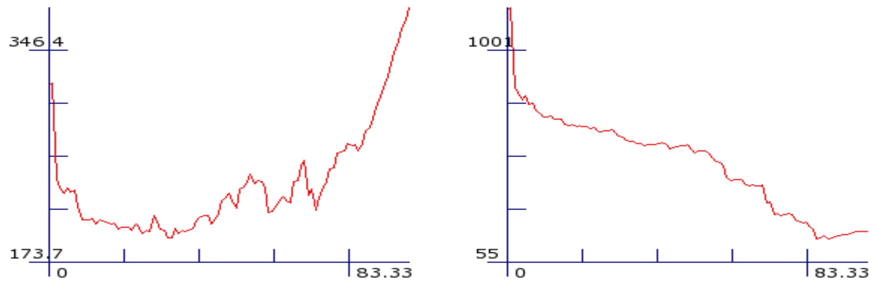


Thom map, 300 point training, 3,000 point test sample and the correct 38-degrees polynomial. The noise is Cauchy with $s = 0.03$,



Left: analysis for degree 0–100, optimum at 38 degrees, good generalization in the range 2–26 and overfitting from 47 degrees onwards.

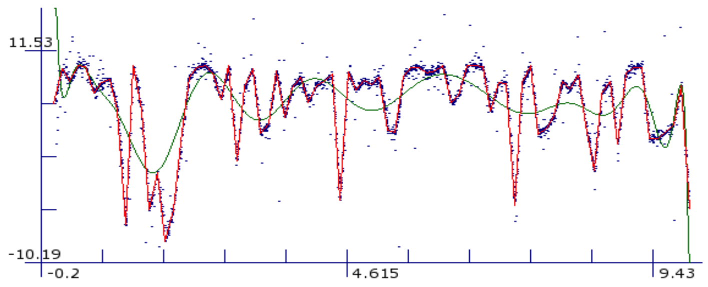
Right: cross-validation, optimum at 18 degrees, good performance in the range 3–22 and overfitting from degree 24 onwards.



Left: mixture MDL predicts the optimum at 38 degrees, good generalization in the range 3–79 and overfitting from 94 degrees onwards.

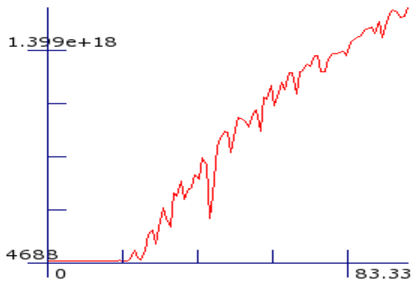
Right: Rissanen's original version predicts the optimum at 86 degrees, good generalization from 16 degrees onwards and no overfitting.

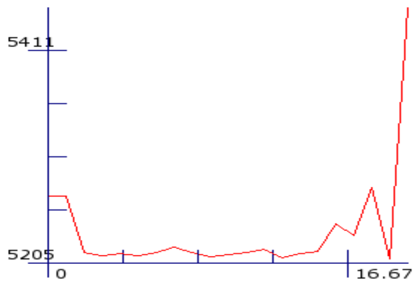
Figure 17: Thom map and Cauchy noise of $s = 0.1$



Thom map, 300 point training, 3,000 point test sample and the correct 13-degree polynomial. The noise is Cauchy with $s = 0.1$,

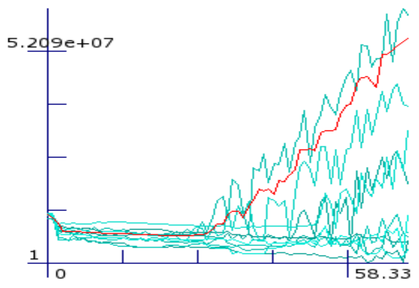
While in all the other figures all samples were completely visible, in this case there are some points that lie far outside the plotted area. The most extreme point has a value of 3935 which is 250 times larger than the upper limit of the plot.

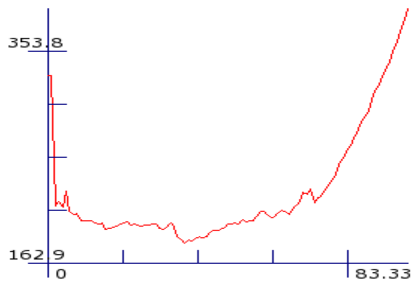




Left: analysis for degree 0–100, optimum at 13 degrees, good generalization in the range 2–17 and overfitting from 20 degrees onwards.

Right: magnification of the range 0–20 of the analysis





Left: cross-validation, optimum at 27 degrees, good generalization in the range 2–34 and overfitting from 39 degrees onwards.

Right: mixture MDL, optimum at 38 degrees, good generalization in the range 2–85 and overfitting from 93 degrees onwards. This is not much different from the result on samples drawn with $s = 0.1$.

Rissanen's original version is not shown. It predicts the optimum at 74 degrees, good generalization from 2 degrees onwards and no overfitting.

very narrow. Polynomials converge fast with this source and it can be described well by polynomials of 50 degrees and more, catching all its alternations. Small samples might witness the extreme regions only with a single point, if at all. A method that simply ignores outliers will be in the disadvantage.

The experiments in Figure 13–16 show the performance of cross-validation and MDL on the different types of noise. Rissanen's original version cannot give meaningful predictions on this source. It consistently overfits by many degrees and never predicts any overfitting in the range for which it was tested.

Mixture MDL is not much influenced by the type of noise. It gives essentially the same predictions throughout. Cross-validation gives different results for different types of noise but that does not result in overall better predictions. For the one distribution where mixture MDL really overfits, Cauchy with $s = 0.1$, cross-validation fails as well. For the same distribution but with a training sample of 700 points (not shown in the figures) both methods give excellent predictions. But even in this case the predictions of mixture MDL are not really different from those for the other types of noise.

Conclusion. We can conclude that mixture MDL performs acceptable under most types of noise, though it does not really take account of them. Its predictions are so general that the nature of a particular distribution doesn't really make a difference. But this is exactly why we assume the Gaussian distribution when the true distribution is unknown: it minimizes the effect of a wrong assumption.

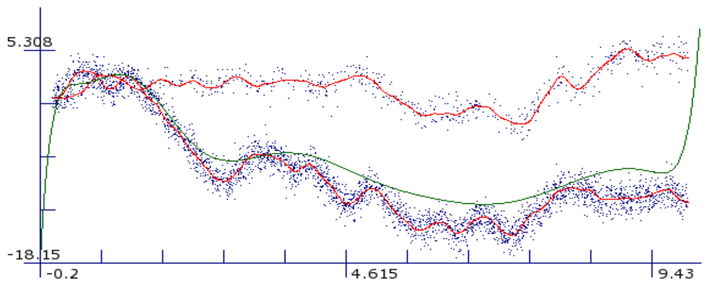
The Cauchy distribution has no calculable variance and all methods fail on small samples from this distribution, as expected. But most points that were drawn by the Cauchy distributions of the experiments lie much closer to the source function than the points that were drawn by the other distributions of this experiment. It is hard to believe that a correct prediction should be impossible under such circumstances and we hope for future improvement.

3.5 Disturbance by foreign signals

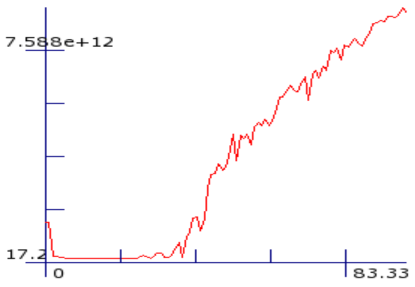
The distributions of the noise in Experiment 5 were rather simple. They were all symmetric around the mean and non increasing on both sides of it. There are many sorts of noise which do not share these properties. This last experiment will look at the performance of MDL when the points that witness the target function were polluted by some other function that we are not interested in. This is a very common phenomenon in signal processing. Usually the target signal has to be filtered from a host of foreign signals before it is available for further analysis. This is similar to *denoising* [Ris00].

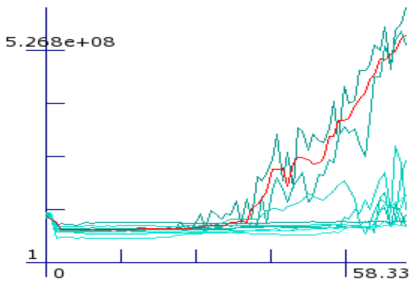
It is almost impossible to filter out all foreign signals. They often leave a weak trace behind. As MDL was able to deal with the different types of noise of Experiment 5 we expect that it will also be able to perform well when the points are polluted by some other function. Appendix 5 on page 62 gives some details on the noisy pendulum that is used as the source function for this experiment.

Figure 18: Multiple sources



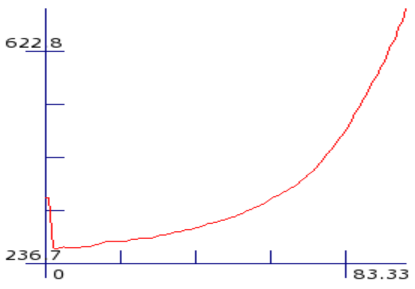
Two pendula (the lower one is the target and the upper one is foreign) 300 point training sample, 3,000 point test sample and the correct 13-degree polynomial. The upper pendulum is witnessed by 17% of the points, the lower pendulum by 83%.

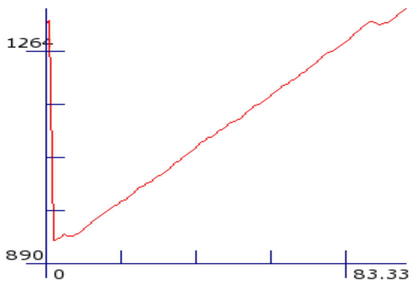




Left: analysis for degree 0–100, optimum at 13 degrees, good generalization in the range 2–39 and overfitting from 44 degrees onwards.

Right: cross-validation predicts the optimum at 7, good generalization in the range 2–39 and overfitting from 40 degrees onwards.





Left: mixture MDL predicts the optimum at 3 degrees, good generalization in the range 2–46 and overfitting from 62 degrees onwards.

Right: Rissanen's original version for degree 0–100, predicts the optimum at 2 degrees, good generalization in the range 2–47 and overfitting from 97 degrees onwards.

Experiment 6: Disturbance by foreign signals

Hypothesis: MDL will perform well when the data is disturbed by an external function.

Source: a time series produced by a noisy pendulum.

Noise: a second pendulum that accounts for 17% of the data. In addition, the measurements on both pendula are disturbed by a normal distribution with variance $\sigma^2 = 1$.

Support: uniform over the interval [0, 10].

Sample: one sample of 300 points.

Test set: i.i.d., 3,000 points.

We admit that the definition of this learning problem is a bit arbitrary. We could as well have said that we want to learn a source that consists of two functions or that the distribution around the source function has two maxima and changes along the x-axis. The setup of this experiment would fulfill any of those definitions.

The generalization analysis in Figure 18 shows a very clear region of overfitting. The borders are sharp and the generalization error is almost constant throughout the basin. Cross-validation reproduces this exactly. Both versions of MDL predict an optimum that has a low generalization error. They don't reproduce the flatness of the basin. Instead, both methods predict a continuing increase in the generalization error from the optimum onwards. Also, both methods include degrees that overfit in their region of good generalization.

Conclusion. Cross-validation again proved that the sample contains all the critical information. MDL did not reproduce it, though both methods made good predictions on the optimum.

Different setups of the experiment with different sample sizes and different ratios for the distribution of the noise over the two functions showed similar results. The basin of the generalization analysis is very flat and cross-validation gives accurate predictions. Both versions of MDL usually predict a good optimum but include degrees that overfit in their regions of good generalization.

4 Discussions & conclusions

4.1 The results of the experiments

Rissanen's original two-part version. The only experiments where Rissanen's original version achieved good results were the simple sinus curve and the pendulum of the last experiment. On everything else it more or less failed. Of course, for each problem the penalty term can be changed and two-part MDL will then give good results. No penalty term is known that shows results as universally good as those of cross-validation. But that is what we expect from a reliable prediction method.

Mixture MDL. Liang and Barron's minimax version of mixture MDL is much stronger than Rissanen's original version. It is capable of taking account of a host of source functions, be it the sinus wave, the step function, the Lorenz attractor, the Thom map or two instances of a pendulum. It is stable under different types of measurement noise. But it fails when the distributions over the support is not uniform.

Liang and Barron's algorithm makes use of a small subset of the training set to normalize the otherwise improper prior distribution. The choice of this subset does have a noticeable effect on the predictions. Especially when the predictions show a number of competing minima of similar depth, different choices for the subset will pronounce different minima as global. To stabilize the predictions even further it is recommended to apply the algorithm several times and to take the average.

Smoothing. The predictions of mixture MDL for individual models seem to be more reliable than those of cross-validation. There is not much need for smoothing or other methods that would stabilize the results. In particular, false minima are rare. Nevertheless it would be worth while to do additional research and look whether smoothing can improve the predictions of MDL.

Different types of noise. For mixture MDL we found that different types of measurement noise all lead to the same predictions. Originally we assumed the distribution over the noise to be Gaussian and it seems that the maximum entropy property of this distribution has saved mixed MDL from making bad predictions.

The independent method (cross-validation). Cross-validation proved as an independent method that all problems were learnable, including those of different types of distributions over the support. The training sample contained all the information that was necessary to give a correct prediction. This shows that there is still much to be improved before MDL can be proclaimed as a universal learning method.

The i.i.d. assumption. Remember that our basic and only assumption about training samples and future samples was that they are identically independently distributed. This is a very powerful assumption. It implies that membership in the training sample and the future test sample is completely random. We do not believe that a learning algorithm that cannot generalize from a training sample to an i.i.d. test sample whenever that is possible can rightfully be called a universal learning algorithm.

But what if we relax even the i.i.d. assumption. Can MDL still give good predictions? What if we allow the measurement noise to change between the training and the test phase? Experiment 5 has shown that mixture MDL, combined with the minimax approach, is capable of doing exactly that. The changes in noise did not really effect its predictions. Whether the noise was Gaussian, uniform, exponential or Cauchy, the predictions were more or less the same, and they were also more or less correct. An important conclusion is that MDL is capable of learning even if we relax the i.i.d. assumption.

4.2 The structure of the per model predictions

The experiments have shown that there is more to say about the quality of a learning method than only how close it comes to the optimum. A good method for model selection should give reliable estimates on the generalization error of individual models.

The structure of the curve of the generalization analysis in itself contains information on the source that can be exploited.

This generalization curve looks definitely very different for a sinus wave, for the step function, or for the Lorenz attractor. Samples that originate from the same signal share many details in the generalization curve. Especially mixture MDL captures these details in a way that was not anticipated. It would be interesting to study how these curves relate to the Kolmogorov structure function of the samples.

Based on those cases where the minimax version of mixture MDL does prevent overfitting we can formulate the following hypothesis. Let $s = \{(x_1, y_1) \dots (x_n, y_n)\}$ be our training sample and let $\hat{m}_k$ be the corresponding k dimensional least square model. We predict that the length of the mixture code $-\log p(s|k) + \log k$ is a good approximation of the expected log generalization error of $\hat{m}_k$ on an i.i.d. sample $s' = \{(x_1, y_1) \dots (x_m, y_m)\}$, a sample that stems from the same distribution as our training sample and that is independently identically distributed.

$$-\log p(s|k) + \log(k) \;\approx\; \lim_{m\to\infty} \log \frac{1}{m} \sum_{i=1}^{m} \big(\hat{m}_k(x_i) - y_i\big)^2 + O(1). \tag{45}$$

The constant term $O(1)$ should depend on the model family but not on the samples s and s'. This is a significant extension of the theory. Normally, it is only predicted that Minimum Description Length minimizes the generalization error. The structur of individual model predictions is not considered.

4.3 The utility of MDL

The experiments of this thesis have shown that mixture MDL is in principle able to minimize the generalization error, especially when confronted with small training samples. While this statement is effectively limited to the problem space that was actually studied—polynomial models and regression in the two-dimensional plane—the problems within this space were as broad a selection as possible. And with the fact in mind that polynomial models suffer very badly from overfitting we may assume that MDL will work well on other models and learning problems as well. [GLV00] gives examples of other problems where MDL was successful: modeling a robot arm and recognition of on-line handwritten characters.

As a general purpose method. We are now in a position to answer the question from page 5 where we introduced the three definitions of a good model. Can we minimize the generalization error and randomness deficiency by one and the same method? The answer is "yes". MDL can minimize both the generalization error and randomness deficiency. It can filter, compress and predict future data. It can predict well even when the training sample is very small.

As a default learning method. And can we recommend MDL as the method of choice for the imaginary problems on page 2? At the current state of the theory the answer must be "no". Cross-validation does a better job and should be preferred. The bad performance for non-uniform support effectively ruled out MDL as a default learning method.

Technical shortcomings. While we are convinced that MDL is a powerful and universal method, we also acknowledge the fact that it does not bring with it all the merits that we hoped for. It turns out to be rather difficult to estimate the complexity of a model. Compared to cross-validation, it requires considerably more theoretical knowledge to understand and implement mixture MDL. And while the version of cross-validation that we used in the experiments can be adapted without difficulty to any type of model, we cannot do so for Barron and Liang's algorithm. An extension to ARMA models seems possible, an extension to neural networks is unlikely. This is not the type of general prediction method we want to teach to undergraduate students in the applied sciences or to implement as the default learning method for an autonomous robot. But it is certainly an option for the embedded system industry. Once an algorithm has been found that can correctly estimate the complexity of a given model family it can easily be implemented, provided it is reasonably fast and efficient.

4.4 Evaluation of the application

The *Statistical Data Viewer* was first of all intended for scientific research. But throughout the design great care was taken to give students easy access to the theory as well. Therefor the application must be evaluated twice, as a research tool and as a teaching tool. Up to now only a handful of researchers and students have tried the application and the results are preliminary.

The research tool. As a research tool, the application was a success, beyond the expectations. The object oriented setup of experiments resulted in an efficient and robust experimental environment. The speed and ease of use of the interface not only allowed us to conduct the broad range of experiments we had in mind but uncovered phenomena unknown to us and allowed us to explore them immediately. It turned an abstract and remote theory into a tangible object of study.

A particularly nice experience was the use of the application in discussions. Changes to the setup of an experiment could easily be followed by the discussion partner, the results are immediate and clear, there are few misunderstandings. This allows for a rapid and fruitful exchange of ideas.

The teaching tool. As a teaching tool the *Statistical Data Viewer* was also successful. The concepts in use are clear, the problems can be easily isolated and the whole problem space is at once accessible to the student. Little instructions and no programming is required and it is almost impossible to fail to set up a meaningful experiment.

Objects vs. functions. The *Statistical Data Viewer* uses an object oriented approach to mathematics and tries to transfer the control structure to the graphical representations of the objects. This stands in contrast to the common mathematical packages. They are function oriented and rely heavily on scripting. This was never experienced as a draw back. There is no reason to assume that mathematical packages have to be function oriented simply because that is the way most mathematical concepts are defined. The function oriented approach of mathematics is nothing but a notational convention that can change with the media of communication.

The transfer of control to the graphical representations did also not lead to any problems. The amount of time needed to understand the application and to set up the first meaningful experiment can be counted in minutes, not in hours or days. Even experienced users of the common mathematical packages need considerable time to setup a meaningful experiment, and due to the complexity of the scripts there always remains a comparatively high chance of errors in the definitions of the experiment.

The *Statistical Data Viewer* does not exclude scripting, though. Any extension of the functionality will have to be programmed. The important difference is that the definition of a mathematical function and the use of it in experiments is strictly separated.

4.5 Further research

The application has indeed proven to give good access to a very abstract theory, to help researchers gain valuable new insight and to give outsiders easy access to the complex problems involved. There are concrete plans to extend the *Statistical Data Viewer* to more dimensional data and to add other important families of models: splines, ARMA models, Markov chains, wavelets, neural networks and decision trees. There are variants of MDL and other methods for model selection that were not discussed in this thesis.

There are also aspects of model selection that want to be made more precise: the effects of smoothing, the precise nature of the critical points, , research on the structure of the predictions, and taking account of arbitrary cost functions[10]. Especially the structure of the predictions promises to be an area of fruitful research as little is known about them and as it seems to be very difficult to get hold on them without an application like the *Statistical Data Viewer*.

On the application level the graphics need to be further integrated into the control structure of the application. The XML interfaces need to be improved and the distribution of calculations over a network of computers should be supported. The programming interfaces as a whole should be simplified.

But the most challenging task seems to be the acceptance of an experimental visualization tool by researchers that work on fundamental theory.

4.6 Theory & visualization

The preface mentioned the notorious inaccessibility of statistics and complexity theory. Part of the problem is the limited use of suitable research tools that give tangible results. The *Statistical Data Viewer* proves that this is not a technical problem. Even a master student can build a reasonable application.

I personally have the feeling that the importance of experimental visualization tools is not well understood in statistics and complexity theory. For example, many of the otherwise excellent lecture books that I have seen during my study of Artificial Intelligence in Groningen missed some basic facts about overfitting that are immediately revealed by the *Statistical Data Viewer*. Those same misunderstandings repeatedly emerged when I discussed the results of my project with leading experts. They include:

- ignorance of the fact that overfitting can and should be avoided even when the model family is a bad choice for a given learning problem. Cross-validation has proven throughout that this is possible and we have already discussed in Section 1.2 that we cannot always choose a family that is optimal for a given problem.
- confusion over what a good model is. Many experts firmly believe that a model of the same family and degree as the original signal will always minimize the generalization error. This was discussed in Section 1.3.
- confusion over the definition of real overfitting. This was discussed in Section 2.4.
- the extend to which the original signal and a model that heavily overfits overlap. In all the experiments conducted during the work on this thesis, a model that heavily overfits still follows the original signal very closely

[10] Currently only the mean squared error is used. See Section 2.2 on page 24 for a discussion of cost functions.

> for the most part of the support range of the x-axis. In contrast, most text books that I have seen suggest that the overfitting model and the original signal should deviate almost everywhere.

Also, the theory group in which I did this research found it difficult to agree that experimental visualization tools can help theory in many ways. As put by Paul Vitányi, "If you can see it, it's not theory." According to such views, deduction is the only legitimate tool to navigate the oceans of fundamental theory. Fortunately, in his fundamental work on undecidability [Göd31], Kurt Gödel has once and for all shown that you can always make true statements about a theory strong enough to contain arithmetic that cannot be proven in that theory. There are examples of powerful mathematical hypotheses that sprung up from experiments and that gained considerable momentum without being proven: the Riemann hypothesis [Rie59][11], unproven since 1859, and the Shimura-Taniyama conjecture [Shi71][12] which stayed unproven for forty years. Experimental tools like the *Statistical Data Viewer* can reveal important properties of MDL that can be very difficult or even impossible to deduce from the theory.

We conclude this argument with a scene from Berthold Brecht's play *Life of Galileo* [Bre95]:

> Some scholars from the university of Florence visit Galileo Galilei at his home. He wants them to look through his modern telescope and see the moons of Jupiter. They are inconsistent with the Ptolemaic model of the universe. The scholars refuse. If Galilei cannot give a formal proof for the existence of such moons, why should they believe his telescope?

[11] The Riemann hypothesis states that the complex zeta function

$$\zeta(x) \;=\; \frac{1}{\Gamma(x)}\int_0^\infty \frac{u^{x-1}}{e^u-1}\,du \qquad (46)$$

has infinitely many nontrivial roots and that all of them have real part 1/2. Among others, the Prime Number Theorem depends on this.

[12] Yutaka Taniyama conjectured in 1960 that every elliptic curve defined over the rational field is a factor of the Jacobian of a modular function field. The Shimura-Taniyama conjecture was finally proven in 1994 through a tremendous effort by A.J. Wiles, thereby proving Fermat's theorem as well [Wil95].

5 Summary

Theory. Model selection plays an important part in machine learning and in artificial intelligence in general. It is used in many advanced applications. A central problem to model selection is overfitting. It can be measured as the generalization error on test samples. But minimizing the generalization error is not the only definition of a good model. Resemblance of the original function and minimum randomness deficiency are others. While we are not interested in resemblance of the original function we do want to know if minimum randomness deficiency and minimizing the generalization error can be combined.

Kolmogorov complexity is a powerful mathematical tool. It is a basic concept in statistics and computer science. It cannot be computed but it is highly useful for proving bounds and existence for weaker notions of complexity. One of the motives that brought about the conception of Kolmogorov complexity was data prediction. This has finally resulted in the theory of MDL for model selection. MDL selects a model that minimizes the combined complexity of model and data, known as the two-part code. It has been proven that MDL minimizes randomness deficiency [VV02].

Early attempts to reliably estimate model complexity were not very successful. Mixture MDL abandoned the two-part approach altogether and concentrated on the complexity of the data given the degree of the model, not a particular model. In [BL02] A. Barron and F. Liang have combined this version of MDL with the minimax approach, looking at data complexity from a worst case perspective.

Experimental verification. One of the prominent problems in statistics is the availability of appropriate data. Another is the limitations that scripting imposes on the choice of experiments. To efficiently map the performance of MDL on as broad a selection of problems as possible we introduced our own application, the *Statistical Data Viewer*. While intended for the advanced scientist, it has an interface that is easy enough to use to allow uninitiated students to explore and comprehend the problems of model selection. Its interactive plots and sophisticated editor allow for a fast and efficient setup and execution of controlled experiments.

All the relevant aspects of model selection are implemented. The problem space consists of two-dimensional regression problems and the models are polynomials. To objectively measure the generalization error we use i.i.d. test samples. To compare MDL to an independent successful method for model selection we implemented cross-validation. The application can easily be extended to other problems, models and methods. This thesis includes a manual and some details of the implementation.

Already in the early experiments a number of critical points emerged that are essential to the evaluation of a method for model selection: 1) the origin or 0-degree model, 2) the optimum model, 3) the region of good generalization: better than half way between the origin and the optimum, 4) false minima and 5) the point of real overfitting where the generalization error becomes forever worse than at the origin.

The experiments described in this thesis include 1) a single frequency sinus wave with Gaussian noise, 2) the step function with Gaussian noise, 3) a non-uniform support for the Lorenz attractor, plus Gaussian noise, 4) normalization of the same problem to the support interval $[-1, 1]$, 5) the Thom map plus different types of noise—Gaussian, uniform, exponential, Cauchy—and 6) two noisy pendula plus Gaussian noise to simulate multiple sources.

Conclusions. The original version of MDL as proposed by Rissanen failed for almost all problems. Mixture MDL on the other hand proved to be of almost equal strength with cross-validation. The only problem where it consistently failed was a non-uniform distribution over the support. Cross-validation proved that the samples contained enough information for correct predictions and we attribute the failure to a weakness of the theory that will hopefully be cured.

Another point of concern was smoothing. Cross-validation was of little use without it. The generalization analysis would also have benefitted from it as it would have removed the false minima and would have resulted in more reliable criteria that the other methods would have to fulfill. But it would have obscured the excellent quality of the per model predictions of mixture MDL for low degree models. Rissanen's original version and mixture MDL did not need it.

The minimax version of mixture MDL proved to be very indifferent to various types of noise. If we assume that the training and the test sample are i.i.d. this falls short of the optimum. On the other hand, if we relax our assumption that training and test sample are i.i.d. we are presented with an even stronger theory: mixture MDL is capable of good prediction even if the distribution over the measurement noise changes.

Where MDL was able to prevent overfitting we observed that the per model predictions accurately reflected the per model generalization error. This led us to formulate the hypothesis that mixture code length is a good approximation of the expected log generalization error on samples that are i.i.d. distributed with regard to the training sample.

The experiments of this thesis and some tests by potential users have shown that applications like the *Statistical Data Viewer* are a real alternative to the common mathematical packages. There is a general need in modern tools for statistical research and there are concrete plans to extend the application. The biggest challenge is to convince the research community to cooperate. Researchers working on fundamental theory tend to view such visualization tools as non-theoretic.

A The random processes

The *Statistical Data Viewer* uses a dynamic range of randomized time series to create interesting two-dimensional learning problems. Most of them are common in nature. They include sinus waves, fractals and the step function.

In all graphs the horizontal x-axis shows the time t and the vertical y-axis shows the value y as a function of t. When more than one variable changes with time, only the y value is visualized in the graph and considered for model selection.

Autoregression. Autoregressive time series are particularly common in nature. The value y_t at time t of such a series is the weighted sum

$$y_t = a_0 + \sum_{i=1}^{n} a_i \, y_{t-i} + \epsilon \tag{47}$$

over n previous values of y. ϵ is the system noise. The *Statistical Data Viewer* allows to specify six different parameters a_0–a_5 but three parameters is usually quite enough. Figure 19 shows an example with $a_1 = 0.5$, $a_2 = 0.5$. The other parameters are equal to zero:

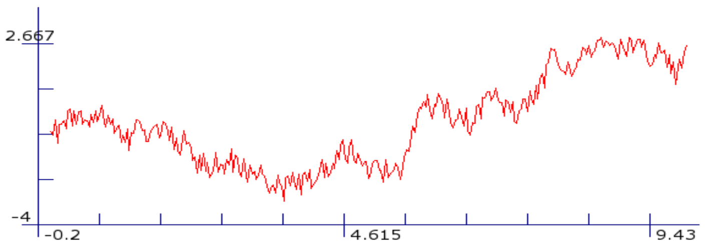


Figure 19: Autoregression

Sinus wave. The sinus wave is also very common. Five frequencies can be specified, each with an individual offset. The resulting function is

$$f(x) = \sum_{i=1}^{5} \sin(f_i \, x + o_i) \tag{48}$$

The example in Figure 20 uses zero offsets and the four frequencies $f_1 = 1.05$, $f_2 = 0.8$, $f_3 = 0.55$ and $f_4 = 0.15$:

Lorenz attractor. The famous Lorenz attractor is a self similar object. It is also a time series. E.N. Lorenz discovered it when he was working on models of the weather [Lor63]. Its evolution is governed by the equations

$$\begin{aligned} y_t &= a\,(z_{t-1} - y_{t-1}) + \epsilon \\ z_t &= b y_{t-1} - z_{t-1}\, w_{t-1} + \epsilon \\ w_t &= y_{t-1}\, z_{t-1} - c\, y_{t-1}) + \epsilon \end{aligned} \tag{49}$$

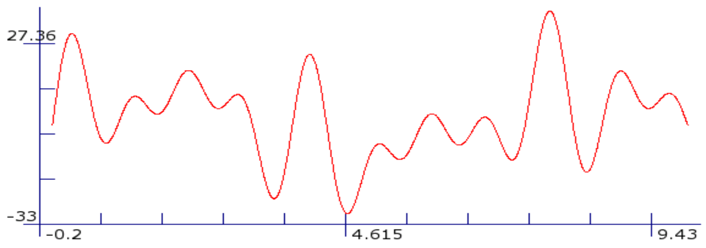


Figure 20: Sinus wave

z and w are not shown in the graph and are not considered in the experiments. The default values are $a = 10$, $b = 28$, $c = 2.667$:

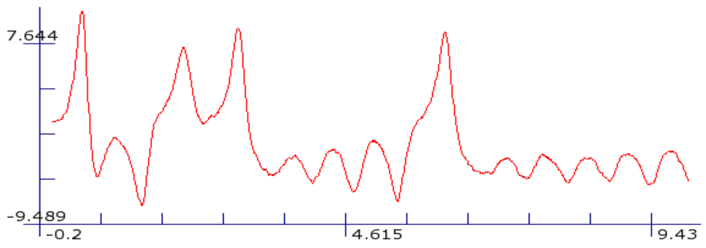


Figure 21: Lorenz attractor

Pendulum. The movement of a noisy pendulum with orbit o and frequency f is defined as

$$y_t \;=\; \sin\left(y_{t-2}\right) - c\, y_{t-1} + f\, \cos\left(o\,(t-2)\right) + \epsilon \tag{50}$$

The example in Figure 22 has $c = 0.2$, $f = 0.5$ and $o = 0.67$.

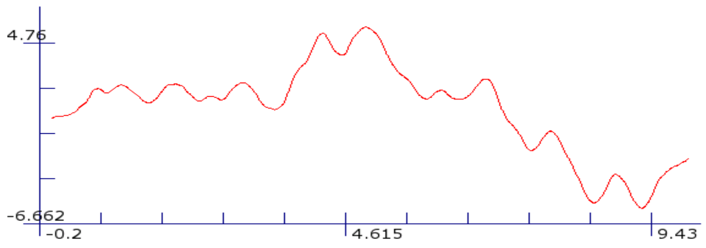


Figure 22: Pendulum

Step function. The step function oscillates n times between two values. The example in Figure figure-step-function has $n = 10$ and oscillates between minus

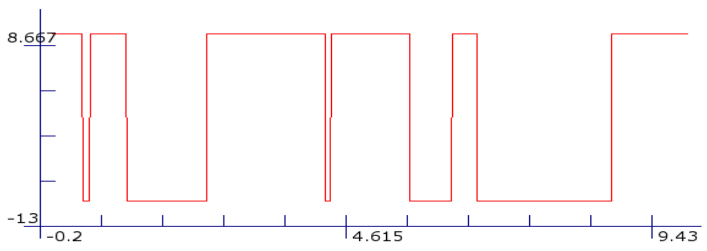


Figure 23: Step function

ten and ten. The points where the function switches between values are chosen at random. The same non-zero seed will produce the same step function.

Thom map. The Thom map is also a time series. R. Thom [Tho93] discovered it when he searched for a simple discrete equivalent to the Lorenz equations which were defined for continous time.

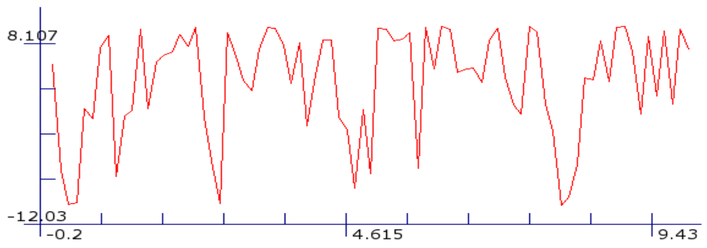


Figure 24: Thom map

$$\begin{aligned} y_t &= a\, y_{t-1} + b\, z_{t-1} + \epsilon \\ z_t &= c\, y_{t-1} + d\, z_{t-1} + \epsilon \end{aligned} \tag{51}$$

The z-axis is not shown in the graph and is not considered in the experiments. The example in Figure 24 has $a = 0.5$, $b = 0.3$, $c = 0.3$ and $d = 0.4$.

B Core algorithms

Least square fitting. The method for linear regression used in the *Statistical Data Viewer* is least square fitting based on Gaussian elimination. A k-degree polynomial has $k+1$ parameters:

$$f_k(x) \; = \; a_0 + a_1 x + \cdots + a_k x^k \tag{52}$$

The squared error between such a polynomial and an n points sample $\{(x_1, y_1) \ldots (x_n, y_n)\}$ is

$$\sigma^2 \; = \; \sum_{i=1}^{n} \Big(y_i - (a_0 + a_1 x_i + \cdots + a_k x_i^k) \Big)^2, \tag{53}$$

the partial derivatives of which are

$$\begin{aligned}
\frac{\partial \sigma^2}{\partial a_0} &= -2 \sum_{i=1}^{n} \left[y_i - (a_0 + a_1 x_i + \cdots + a_k x_i^k) \right] \; = \; 0 \\
\frac{\partial \sigma^2}{\partial a_1} &= -2 \sum_{i=1}^{n} \left[y_i - (a_0 + a_1 x_i + \cdots + a_k x_i^k) \right] x \; = \; 0 \\
\frac{\partial \sigma^2}{\partial a_k} &= -2 \sum_{i=1}^{n} \left[y_i - (a_0 + a_1 x_i + \cdots + a_k x_i^k) \right] x^k \; = \; 0
\end{aligned} \tag{54}$$

which can be rewritten in matrix form as

$$\begin{bmatrix} n & \sum x & \ldots & \sum x^k \\ \sum x & \sum x^2 & \ldots & \sum x^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum x^k & \sum x^{k+1} & \ldots & \sum x^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum y \\ \sum xy \\ \vdots \\ \sum x^k y \end{bmatrix} \tag{55}$$

The left part of (55) is a Vandermonde matrix and it is accessible under this name in the application. A Vandermonde matrix can be calculated in $O(k\,n)$ time while the number of multiplications per term x^y does not exceed $\log y$, minimizing the rounding error [PFTV92].

Actually, since all algorithms are slowed down by higher precision, the time complexities have to be calculated as a function not of k but of $k \log d$ with d the precision in bits or digits. This is neglected for the sake of readability.

The next step in the algorithm is to put the $k \times k$ Vandermonde matrix into echelon or lower triangular form where all values below the diagonal are zero. During the triangularization all operations on rows of the matrix are applied to the same rows of the solution vector to keep the them consistent. The result:

$$\begin{bmatrix} 1 & v_{0,1} & \dots & v_{0,k} \\ 0 & 1 & \dots & v_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_k \end{bmatrix} \tag{56}$$

with $v_{n,m}$ the values at row n, column m after triangularization and s_n the value of the solution vector after triangularization. Triangularization by Gaussian elimination can be done in $O(k^3)$ time. Once the matrix is in triangular form we store it to calculate any polynomial of $\leq k$ degree at any precision in $O(k^2)$. During the process of Gaussian elimination a vector is set aside by which the determinant of any Vandermonde matrix $\leq k$ can be obtained in $O(k)$, a time saving byproduct which is important for mixture MDL.

The parameters of the r-degree polynomial can be read recursively from the triangular matrix, starting with the highest parameter which is equal to s_r of the right hand solution vector. The recursive formula for parameter p_{r-s} with $s \leq r$ is

$$p_{r-s} \; = \; s_{r-s} - \sum_{i=1}^{s} [v_{r-s,r-i} \times p_{r-i}] \tag{57}$$

Rounding problems. Computation and triangularization of the Vandermonde matrix has to be done at a precision high enough to safeguard against obvious rounding errors but low enough to guarantee a fast performance. As a general rule, a precision of $2k$ digits is used for k-degree matrixes, which works well for every matrix of $k \leq 150$. This is way above the 70–100 degrees recommended for most operations because of slow performance beyond that point.

When we calculate a parameter vector of precision $d < 2k$ it is of great importance where in the algorithm we round off. The choices are:

Late rounding. The vector is calculated at the $2k$ digit precision of the matrix. Only the final result is rounded to d digits.

Early rounding. Each parameter is rounded instantly to d digits and then used to calculate the other parameters.

Before reading on the interested reader is suggested to answer for him or herself which method gives the better result.

I asked four experts in statistics and algorithms and all of them gave the wrong answer. Late rounding does *not* give the better result. I implemented both versions and found that early rounding reduces the number of digits needed to achieve the same performance by a factor of 2. The error of a parameter vector of d digits precision obtained with late rounding is roughly equal to that of a vector of $d/2$ digits obtained with early rounding.

More than anything else, this example should make it clear that the size of an actual implementation cannot be used to estimate the complexity of a model.

References

[Aka73] H. Akaike. Information theory and an extension of the maximum likelihood princple. In B.N. Petro and F. Csaki, editors, *Second International Symposium on Information Theory*, pages 267–281, Budapest, 1973. Akademiai Kaido.

[BL02] Andrew Barron and Feng Liang. Exact Minimax Strategies for Predictive Density Estimation, Data Compression and Model Selection. Presented at the International Symposium on Information Theory 2002 in Lausanne, Switserland, June 2002.

[Bre95] Berthold Brecht. *Life of Galileo.* Arcade Publishing, New York, December 1995. Written 1938–39, originally published 1955 in East Berlin.

[Cha66] Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *J. Assoc. Comput. Mach.*, 13:547–569, 1966.

[Cha69] Gregory J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *J. Assoc. Comput. Mach.*, 16:145–159, 1969.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., New York, 1991.

[Fis25] R.A. Fisher. Theory of statistical estimation. *Proc. Cambridge Phil. Society*, 22:700–725, 1925.

[GLV00] Qiong Gao, Ming Li, and Paul Vitányi. Applying MDL to learn best model granularity. *Artificial Intelligence*, 2000.

[Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[Grü00] Peter Grünwald. Maximum Entropy and the Glasses You Are Looking Through. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI 2000)*, Stanford, CA, July 2000.

[Kol65] Andrei Nikolaevich Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–7, 1965.

[KST82] Daniel Kahneman, Paul Slovic, and Amos Tversky. *Judgment under Uncertainty: Heuristics and Biases.* Cambridge University Press, Cambridge, England, April 1982.

[Lev74] L.A. Levin. Laws of information conservation (non-growth) and aspects of the foundation of probability theory. *Problems Inform. Transmission*, 10:206–210, 1974.

[Lor63] Edward N. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–148, 1963.

[LV97] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, Berlin, second edition, 1997.

[MP69] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambrige, MA, 1969.

[PFTV92] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. Vandermonde Matrices and Toeplitz Matrices. In *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, pages 82–89. Cambridge University Press, Cambridge, England, second edition, 1992.

[Rie59] Georg Friedrich Bernhard Riemann. On the number of primes less than a given magnitude. *Monatsberichte der Berliner Akademie*, November 1859.

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

[Ris00] Jorma Rissanen. MDL denoising. *IEEE Trans. Information Theory*, 46(7):2537–2543, 2000.

[Rue89] David Ruelle. *Chaotic Evolution and Strange Attractors.* Cambridge University Press, Cambridge, England, 1989.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[Shi71] Goro Shimura. On elliptic curves with complex multiplication as factors of Jacobian varieties. *Nagoya Math. J*, 43:199–208, 1971.

[Sol64] Ray Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Information and Control*, 7:1–22, 224–254, 1964.

[Tho93] René Thom. *Structural Stability and Morphogenesis: An Outline of a General Theory of Models.* Addison-Wesley, Reading, MA, 1993.

[Tur36] Alan M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proc. London Math. Soc., Ser. 2*, 42:230–265, 1936.

[VV02] Nikolai Vereshchagin and Paul Vitányi. Kolmogorov's Structure Functions with an Application to the Foundations of Model Selection. *Proc. 47th IEEE Symp. Found. Comput. Sci. (FOCS'02)*, 2002.

[Wil95] Andrew John Wiles. Modular elliptic curves and Fermat's Last Theorem. *Annals of Mathematics*, 141:443–551, 1995.

Index