Data Min Knowl Disc (2015) 29:358–399
DOI 10.1007/s10618-014-0345-2

Using the minimum description length to discover the intrinsic cardinality and dimensionality of time series

Bing Hu · Thanawin Rakthanmanon · Yuan Hao · Scott Evans · Stefano Lonardi · Eamonn Keogh

Received: 27 December 2012 / Accepted: 9 January 2014 / Published online: 15 February 2014
© The Author(s) 2014. This article is published with open access at Springerlink.com

Abstract Many algorithms for data mining or indexing time series data do not operate directly on the raw data, but instead they use alternative representations that include transforms, quantization, approximation, and multi-resolution abstractions. Choosing the best representation and abstraction level for a given task/dataset is arguably the most critical step in time series data mining. In this work, we investigate the problem of discovering the natural intrinsic representation model, dimensionality and alphabet cardinality of a time series. The ability to automatically discover these intrinsic features has implications beyond selecting the best parameters for particular algorithms, as characterizing data in such a manner is useful in its own right and an important subroutine in algorithms for classification, clustering and outlier discovery. We will frame

Responsible editor: Geoffrey I. Webb.

B. Hu (✉) · T. Rakthanmanon · Y. Hao · S. Lonardi · E. Keogh
Department of Computer Science & Engineering, University of California,
Riverside, Riverside, CA 92521, USA
e-mail: bhu002@ucr.edu

T. Rakthanmanon
e-mail: rakthant@ucr.edu

Y. Hao
e-mail: yhao002@ucr.edu

S. Lonardi
e-mail: stelo@cs.ucr.edu

E. Keogh
e-mail: eamonn@cs.ucr.edu

S. Evans
GE Global Research, Niskayuna, NY, USA
e-mail: evans@ge.com

the discovery of these intrinsic features in the Minimal Description Length framework. Extensive empirical tests show that our method is simpler, more general and more accurate than previous methods, and has the important advantage of being essentially parameter-free.

Keywords Time Series · MDL · Dimensionality reduction

1 Introduction

Most algorithms for indexing or mining time series data operate on higher-level representations of the data, which include transforms, quantization, approximations and multi-resolution approaches. For instance, Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT), Adaptive Piecewise Constant Approximation (APCA) and Piecewise Linear Approximation (PLA) are models that all have their advocates for various data mining tasks and each has been used extensively (Ding et al. 2008). However, the question of choosing the best abstraction level and/or representation of the data for a given task/dataset still remains. In this work, we investigate this problem by discovering the natural intrinsic model, dimensionality and (alphabet) cardinality of a time series. We will frame the discovery of these intrinsic features in the Minimal Description Length (MDL) framework (Grünwald et al. 2005; Kontkanen and Myllym 2007; Pednault Pednault 1989; Rissanen et al. 1992). MDL is the cornerstone of many bioinformatics algorithms (Evans et al. 2007; Rissanen 1989), and has had some impact in data mining, however it is arguably underutilized in *time series* data mining (Jonyer et al. 2004; Papadimitriou et al. 2005).

The ability to discover the intrinsic dimensionality and cardinality of time series has implications beyond setting the best parameters for data mining algorithms. For instance, it can help characterize the nature of the data in a manner that is useful in its own right. It can also constitute an important sub-routine in algorithms for classification, clustering and outlier discovery (Protopapas et al. 2006; Yankov et al. 2008). We illustrate this idea in the following example in Fig. 1, which consists of three unrelated datasets.

The number of unique values in each time series is, from top to bottom, 14, 500 and 62. However, we might reasonably claim that the *intrinsic* alphabet cardinality is

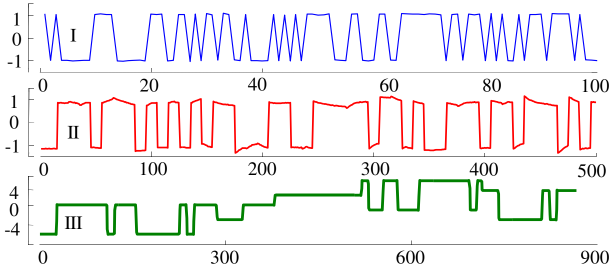


Fig. 1 Three unrelated industrial time series with low intrinsic cardinality. (*I*) Evaporator (channel one). (*II*) Winding (channel five). (*III*) Dryer (channel one)

instead 2, 2, and 12, respectively. As it happens, an understanding of the processes that produced this data would perhaps support this claim (Keogh et al. 2006). In these datasets, and indeed in many real-world datasets, there is a significant difference between the actual and intrinsic cardinality. Similar remarks apply to dimensionality.

- Before we define more precisely what we mean by actual versus intrinsic cardinality, we should elaborate on the motivations behind our considerations. Our objective is generally not simply to save memory:[1] if we are wastefully using eight bytes per time point instead of using the mere three bytes required by the intrinsic cardinality, the memory space saved is significant; however, memory is getting cheaper, and is rarely a bottleneck in data mining tasks. Instead, there are many other reasons why we may wish to find the true intrinsic model, cardinality and dimensionality of the data. For example, there is an increasing interest in using specialized hardware for data mining (Sart et al. 2010). However, the complexity of implementing data mining algorithms in hardware typically grows super linearly with the cardinality of the alphabet. For example, FPGAs usually cannot handle cardinalities greater than 256 (Sart et al. 2010).
- Some data mining algorithms benefit from having the data represented in the lowest meaningful cardinality. As a trivial example, consider the time series: ..0, 0, 1, 0, 0, 1, 0, 0, 1. We can easily find the rule that a ‘1’ follows two appearances of ‘0’. However, notice that this rule is not apparent in this string: ..0, 0, 1.0001, 0.0001, 0, 1, 0.000001, 0, 1 even though it is essentially the same time series.
- Most time series indexing algorithms critically depend on the ability to reduce the dimensionality (Ding et al. 2008) or the cardinality (Lin et al. 2007) of the time series (or both Assent et al. 2008; Camerra et al. 2010) and search over the compacted representation in main memory. However, setting the best level of representation remains a “black art.”
- In resource-limited devices, it may be helpful to remove the spurious precision induced by a cardinality/dimensionality that is too high. We elaborate on this issue by using a concrete example below.
- Knowing the intrinsic model, cardinality and dimensionality of a dataset allows us to create very simple outlier detection models. We simply look for data where the parameters discovered in new data differ from our expectations learned on training data. This is a simple idea, but it can be very effective as we show in our experimental section.

1.1 A concrete example

For concreteness, we present a simple scenario that shows the utility of understanding the intrinsic cardinality/dimensionality of data. Suppose we wish to build a time series classifier into a device with a limited memory footprint such as a cell phone, pacemaker or “smartshoe” (Vahdatpour and Sarrafzadeh 2010). Let us suppose we have only 20kB available for the classifier, and that (as is the case with the benchmark dataset, TwoPat

[1] However, Sect. 1.1 shows an example where this *is* useful.

Keogh et al. 2006) each time series exemplar has a dimensionality of 128 and takes 4 bytes per value.

One could choose decision trees or Bayesian classifiers because they are space efficient; however, recent evidence suggests that nearest neighbor classifiers can be difficult to beat for time series problems (Ding et al. 2008). If we had simply stored forty random samples in the memory for our nearest neighbor classifier, the average error rate over fifty runs would be a respectable 58.7 % for a four-class problem. However, we could also down-sample the dimensionality by a factor of two, either by skipping every second point, or by averaging pairs of points (as in SAX, Lin et al. 2007), and place eighty reduced-quality samples in memory. Or perhaps we could instead reduce the alphabet cardinality by reducing the precision of the original four bytes to just one byte, thus allowing 160 reduced-fidelity objects to be placed in memory. Many other combinations of dimensionality and cardinality reduction could be tested, which would trade reduced fidelity to the original data for more exemplars stored in memory. In this case, a dimensionality of 32 and a cardinality of 6 allow us to place 852 objects in memory and achieve an accuracy of about 90.75 %, a remarkable improvement in accuracy given the limited resources. As we shall see, this combination of parameters can be found using our MDL technique.

In general, testing all of the combinations of parameters is computationally infeasible. Furthermore, while in this case we have class labels to guide us through the search of parameter space, this would not be the case for other unsupervised data mining algorithms, such as clustering, motif discovery (Lin et al. 2002), outlier discovery (Chandola et al. 2009; Vereshchagin and Vitanyi 2010; Yankov et al. 2008), etc.

As we shall show, our MDL framework allows us to automatically discover the parameters that reflect the intrinsic model/cardinality/dimensionality of the data without requiring external information or expensive cross validation search.

2 Definitions and notation

We begin with the definition of a time series:

Definition 1 A *time series T* is an ordered list of numbers. $T = t_1, t_2, ..., t_m$. Each value t_i is a finite precision number and m is the length of the *time series T*.

Before continuing, we must justify the decision of (slightly) quantizing the time series. MDL is only defined for discrete values,[2] but most time series are real-valued. The cardinality of a set is defined as the measure of the number of elements of the set. In math, discrete values have a finite cardinality, and real numbers have an infinite cardinality. When dealing with values stored in a digital computer, this distinction can be problematic, as even real numbers must limited to a finite cardinality. Here we simply follow the convention that for very high cardinalities numbers can be considered essentially real-valued, thus we need to cast the "effectively infinite" 2^{64} cardinality we typically encounter into a more obvious discrete cardinality to allow MDL to be applied.

[2] The closely related technique of MML (Minimum Message Length Wallace and Boulton 1968) *does* allow for continuous real-valued data. However, here we stick with the more familiar MDL formulation.

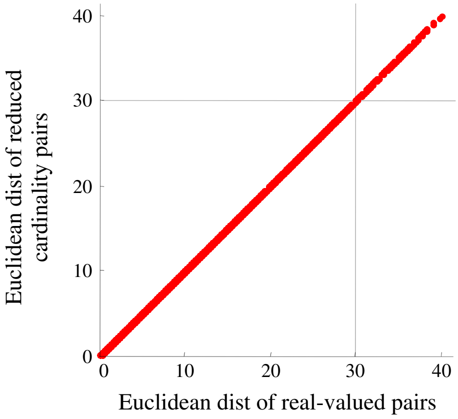


Fig. 2 Each point on this plot corresponds to a pair of time series: the x-axis corresponds to their Euclidean distance, while the y-axis corresponds to the Euclidean distance between the 8-bit quantized representations of the same pair

The obvious solution is to reduce the original number of possible values to a manageable amount. Although the reader may object that such a drastic reduction in precision must surely lead to a loss of some significant information, this is not the case. To illustrate this point, we performed a simple experiment. From each of the 20 diverse datasets in the UCR archive (Keogh et al. 2006) we randomly extracted one hundred pairs of time series. For each pair of time series we measured their Euclidean distance in the original high dimensional space, and then in the quantized 256-cardinality space, and used these pairs of distances to plot a point in a scatter plot. Figure 2 shows the results.

The figure illustrates that all of the points fall close to the diagonal, and thus the quantization makes no perceptible difference. Beyond this subjective visual test, we also reproduced the heavily cited UCR time series classification benchmark experiments (Keogh et al. 2006), replacing the original data with the 256-cardinality version. For all cases the difference in classification accuracy was less than one tenth of 1 % (full details are at www.cs.ucr.edu/~bhu002/MDL/MDL.html). Based on these considerations, in this work we reduce all of the time series data to its 256 cardinality version by using a discretization function:

Definition 2 A *discretization* function normalizes a real-valued time series T into b-bit discrete values in the range $[-2^{\mathrm{b}-1}, 2^{\mathrm{b}-1}-1]$. The discretization function used in this manuscript is as follows:

$$Discretization_b(T) = round\left(\frac{T - \min}{\max - \min}\right) * (2^b - 1) - 2^{b-1}$$

where *min* and *max* are the minimum and maximum values in T, respectively.[3]

Given a time series T, we are interested in estimating its *minimum description length*, i.e., the smallest number of bits it takes to represent it.

[3] This slightly awkward formula is necessary because we use the symmetric range [−128, 127]. If we use range [1, 256] instead we get a more elegant: $Discretization(T) = round\left(\frac{(T-min)}{(max-min)}\right) * (2^{\mathrm{s}} - 1) + 1$.

Definition 3 A *description length DL* of a time series T is the total number of bits required to represent it. When Huffman coding is used to compress the time series T, the description length of the time series T is defined by:

$$DL\left(T\right) = \left|HuffmanCoding\left(T\right)\right|$$

In the current literature, the number of bits required to store the time series depends on the idiosyncrasies of the data format or hardware device, not on any intrinsic properties of the data or domain. Here we are instead interested in knowing the minimum number of bits to exactly represent the data, i.e., the *intrinsic* amount of information in the time series. The general problem of determining the smallest program that can reproduce the time series, known as Kolmogorov complexity, is not computable (Li 1997). However, the Kolmogorov complexity can be approximated by using general-purpose data compression methods, like Huffman coding (Grünwald et al. 2005; Vereshchagin and Vitanyi 2010; Zwally and Gloersen 1977). The (lossless) compressed file size is an upper bound to the Kolmogorov complexity of the time series (De Rooij and Vitányi 2012).

Observe that in order to decompress the data $HuffmanCoding(T)$, the Huffman tree (or the symbol frequencies) is needed, thus the description length could be more precisely defined as $DL(T) = |HuffmanCoding(T)| + |HuffmanTree(T)|$. One could use a simple binary representation to encode the Huffman tree, however the most efficient way to encode it is to assume that the tree in the canonical form (see, e.g., Chapter 2 of Witten et al. 1999). In a canonical Huffman code all the codes for a given length are assigned their values sequentially. Instead of storing the structure of the Huffman tree explicitly, only the lengths of the codes (i.e., the number of bits) are required. Since the longest possible Huffman code over 2^b symbols is at most $2^b - 1$ bits long (when symbol frequencies are Fibonacci numbers), the number of bits necessary to transmit its length is at most b. Thus, the number of bits necessary to represent $HuffmanTree(T)$ in canonical form can be bounded by $O(b2^b)$.

We disregarded the cost Huffman tree in our formulation, because in practice

- The size of the tree is negligible compared to the number of bits required to represent the time series because $m \gg b$.
- The size of $|HuffmanTree(T)|$ has very low variance, and thus can be regarded as a "constant" term. This is especially true when comparing similar models, for example, a model with a dimensionality of 10 to a model with a dimensionality of nine or eleven. When comparing vastly different models, for example a model with a dimensionality of ten with a model with a dimensionality of one hundred, the differences of the sizes of the relevant Huffman trees are greater, but this difference is dwarfed by the bit saving gained by discovering the true dimensionality.

In the extensive experiments in Sect. 4 we found there is no measureable difference in outcome of the formulations with or without the cost of $|HuffmanTree(T)|$ included, thus we report only the simpler formulation.

One of the key steps in finding the intrinsic cardinality and/or dimensionality requires one to convert a given time series to another representation or model, e.g., by using DFT or DWT. We call this representation a *hypothesis*:

Definition 4 A *hypothesis* H is a representation of a discrete time series T after applying a transformation M.

In general, there are many possible transforms. Examples include DWT, DFT, APCA, and Piecewise Linear Approximation (PLA), among others (Ding et al. 2008). Figure 8 shows three illustrative examples, DFT, APCA, and PLA. In this paper, we demonstrate our ideas using these three most commonly used representations, but our ideas are not restricted to these time series models (see Ding et al. 2008 for a survey of time series representations).

Henceforth, we will use the term *model* interchangeably with the term *hypothesis*.

Definition 5 A *reduced description length* of a time series T given hypothesis H is the number of bits used for encoding the time series T, exploiting information in the hypothesis H, i.e., DL $(T|H)$, and the number of bits used for encoding H, i.e., DL (H). The reduced description length is defined as:

$$DL(T, H) = DL(H) + DL(T|H)$$

The first term DL (H) is called the *model cost* and represents the number of bits required to store the hypothesis H. For instance, the model cost for the PLA would include the bits needed to encode the mean, slope and length of each linear segment.

The second term, DL $(T|H)$, called the *correction cost* (in some works it is called the *description cost* or *error term*) is the number of bits required to rebuild the entire time series T from the *given* hypothesis H.

There are many possible ways to encode T given H. Perhaps the simplest way is to store the differences (i.e., the difference vector) between T and H: one can easily reconstruct exactly the time series T from H and the difference vector. Thus, we simply use $DL(T|H) = DL(T - H)$.

We will demonstrate how to calculate the reduced description length in more detail in the next section.

3 MDL modeling of time series

3.1 An intuitive example of our basic idea

For concreteness, we will consider a simple worked example comparing two possible dimensionalities of data. Note that here we are assuming a cardinality of 16, and a model of APCA. However, in general we do not need to make such assumptions. Let us consider a sample time series T of length 24:

$$T = \mathbf{1\ 1\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 11\ 12\ 12\ 12\ 12\ 11\ 11\ 10\ 10\ 9\ 7}$$

Figure 3 illustrates a plot of T.

We attempt to model this data with a single constant line, a special case of APCA. We begin by finding the mean of *all* of the data, which (rounding in our integer space) is eight. We can create a hypothesis H_1 to model this data, which is shown in Fig. 4. It

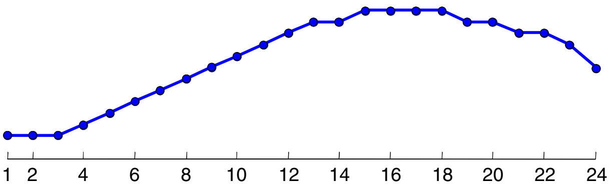


Fig. 3 A sample time series T that will be used as a running example in this section

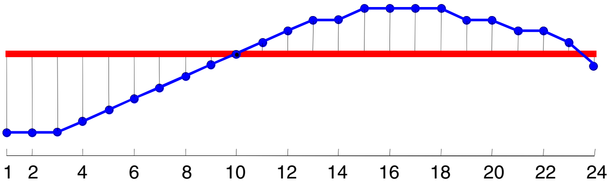


Fig. 4 Time series T (*blue/fine*), approximated by a one-dimensional APCA approximation H_1 (*red/bold*). The error for this model is represented by the *vertical lines* (Color figure online)

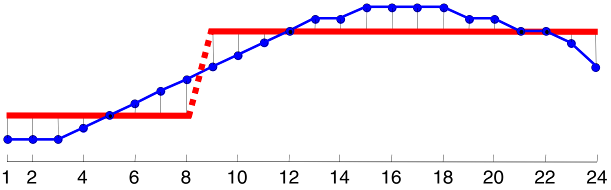


Fig. 5 Time series T (*blue/fine*), approximated by a two-dimensional APCA approximation, H_2 (*red/bold*). *Vertical lines* represent the error (Color figure online)

is simply a constant line with a mean of eight. There are 16 possible values this model *could* have had. Thus, $DL\ (H_1) = 4$ bits.

Model H_1 does not approximate T well, and we must account for the error.[4] The errors e_1, represented by the length of the vertical lines in Fig. 4, are:

$$e_1 = \mathbf{7\ 7\ 7\ 6\ 5\ 4\ 3\ 2\ 1\ 0\ -1\ -2\ -3\ -3\ -4\ -4\ -4\ -4\ -3\ -3\ -2\ -2\ -1\ 1}$$

As noted in Definition 5, the cost to represent these errors is the correction cost; this is the number of bits encoding $e_1$using Huffman coding, which is 82 bits. Thus, the overall cost to represent $T$with a one-dimensional model or its reduced description length is:

$$DL\ (T,\ H_1) = DL\ (T|H_1) + DL\ (H_1)$$
$$DL\ (T,\ H_1) = 82 + 4 = 86 \text{ bits}$$

We can now test to see if hypothesis H_2, which models the data with *two* constant lines, could reduce the description length. Figure 5 shows the *two* segment approximation lines created by APCA.

[4] The word *error* has a pejorative meaning not intended here; some authors prefer to use *correction cost*.

As we expect, the error e_2, shown as the vertical lines in Fig. 5, is smaller than the error e_1. In particular, the error e_2 is:

$$e_2 = \mathbf{2\ 2\ 2\ 1\ 0\ {-1}{-2}{-3}\ 3\ 2\ 1\ 0\ {-1}\ {-1}\ {-2}\ {-2}\ {-2}\ {-2}\ {-1}\ {-1}\ 0\ 0\ 1\ 3}$$

The number of bits encoding e_2 using Huffman coding or the correction cost to generate the time series T given the hypothesis H_2, $DL\ (T|H_2)$, is 65 bits. Although the correction cost is smaller than one-dimensional APCA, the model cost is larger. In order to store *two* constant lines, *two* constant numbers corresponding to the height of each line and a pointer indicating the end position of the first line are required. Thus, the reduced description length of model H_2 is:

$$DL\ (T, H_2) = DL\ (T|H_2) + DL\ (H_2)$$
$$DL\ (T, H_2) = 65 + 2 * \log_2 (16) + \lceil \log_2 (24) \rceil = 78 \text{ bits}$$

Because we have $DL\ (T, H_2) < DL\ (T, H_1)$, we prefer H_2 for our data.

We are not done yet: we should also test H_3, H_4, H_5, etc., corresponding to 3, 4, 5, etc. piecewise constant segments. Additionally, we could also test alternative models corresponding to different DFT or PLA representations and test different cardinalities. For example, suppose we had been given T_2 instead:

$$T_2 = \mathbf{0\ 0\ 0\ 0\ 4\ 4\ 4\ 4\ 4\ 0\ 0\ 0\ 0\ 8\ 8\ 8\ 8\ 8\ 8\ 12\ 12\ 12\ 12\ 12}$$

Here, if we tested multiple hypotheses as to the cardinality of this data, we would hope to find that the hypothesis H_4^C that attempts to encode the data with a cardinality of just 4 would result in the smallest model.

A review of the following facts may help make our contributions more intuitive. The quality of an approximate representation of a time series is measured by the *reconstruction error* (Ding et al. 2008; Keogh and Kasetty 2003). This is simply the Euclidean distance between the model and the raw data. For example, in Fig. 4 we see the reconstruction error of the single segment model is 18.78, and the reconstruction error of the two segment model in Fig. 5 is just 8.42. This suggests a general truth; for the APCA, PLA, and DFT approximations of time series, it is *always* the case that the d-dimensional model has a greater or equal reconstruction error than $d + 1$ dimensional model (Ding et al. 2008; Palpanas et al. 2008). Note that this is only true on average for the DWT, SAX, IPLA, PAA approximations (Ding et al. 2008). However, even for these representations violations to this rule are very rare and minor.

The reader may have noted that in the example discussed, the range of the error vector (i.e. {$\texttt{max}(e_i)$-$\texttt{min}(e_i)$}) also decreased, from 12 (7 to −4) in the former case, to just 7 (3 to −3) in the latter. This is not necessarily the case; the relevant algorithms are minimizing the *global* error of the model, not the *maximum* error on any individual segment/coefficient, and one can certainly construct synthetic datasets for which this is not the case. However, it is *almost* always the case, and necessarily so. Recall that as d approaches m, this range approaches zero. Further note that this range is an upper bound for the number of unique values that the compression algorithm must encode in the description length.

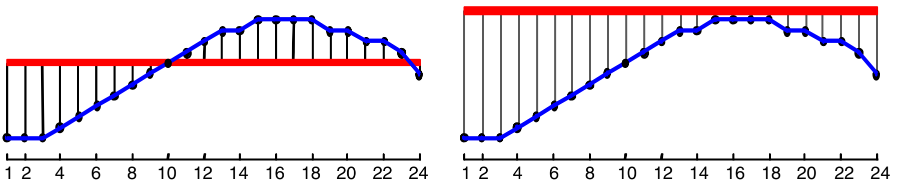


Fig. 6 *Left* The figure shown in Fig. 4 contrasted with an attempt to approximate the raw data with a constant segment that clearly has too great a mean value (*right*). Note that while the number of repeated residuals ("errors") is identical in both cases, the magnitude of the residuals is much greater in the latter case. It is this unnecessarily large magnitude that tells us this is a poor choice of an approximation

We note that this tight relationship between Euclidean distance and MDL has been observed/exploited before. In Fig. 12 of Rakthanmanon et al. (2012), Rakthanmanon et al. shows a scatterplot illustrating the extraordinary high correlation between the Euclidean distance of pair of subsequences, and the MDL description length when using one subsequence to encode the other (using essentially the same MDL formulation that we use here). Thus we can see the naturalness of using MDL to score our model choice, which is solely concerned with minimizing the Euclidean distance between the model and the original data.

We have just one more issue to address before moving on. We had glossed over this issue to enhance the flow of the presentation above. Consider Fig. 6 which contrasts the original single-segment approximation shown in Fig. 4 with an alternative single-segment approximation.

Intuitively, the alternative is much worse, vastly overestimating the mean of the original data. However, on what basis could MDL make this distinction? If our MDL formulation considered the Y-axis values to be *categorical* variables then there would be no reason to prefer either model.

However, note that the sum of the magnitude of the residuals is much greater in for Fig. 6—right. This is true by definition, as using the mean minimizes this value. However, nothing in our model description length *explicitly* accounts for this. An obvious solution to this issue is to encode a term that accounts for the *range* of numbers required to be modeled in the description length, in addition to their entropy. This issue is unique to *ordinal* data, and does not occur with *categorical* data. For example, when dealing with categorical data, there is no cost difference between say $s_{\mathrm{x}} = \mathbf{a\,a\,a\,b}$, and $s_{\mathrm{y}} = \mathbf{m\,m\,m\,n}$. However, in our domain there *is* a significant difference between say $e_{\mathrm{x}} = \mathbf{1\,1\,1\,2}$, and $e_{\mathrm{y}} = \mathbf{3\,3\,3\,4}$, because the latter condemns us to consider values in a $\log_2(4)$ range in the description length for the model, whereas the former allows us to only consider values in the smaller $\log_2(2)$ range.

In principle, this term *is* included in the size of $|HuffmanTree(T)|$, but as we noted above, we ignore this term in our model. The problem with Huffman coding is code words in Huffman coding can only have an integer number of bits. Thus the size of $|HuffmanTree(T)|$, does not distinguish between alternative models if we shift the mean up or down a few values. Arithmetic coding can be viewed as a generalization of Huffman coding, effectively allowing non-integer bit lengths. For this reason it tends to offer significantly better compression for small alphabet sizes, and we should expect a good hypothesis to have a small alphabet size by definition. In Fig. 7 we show the

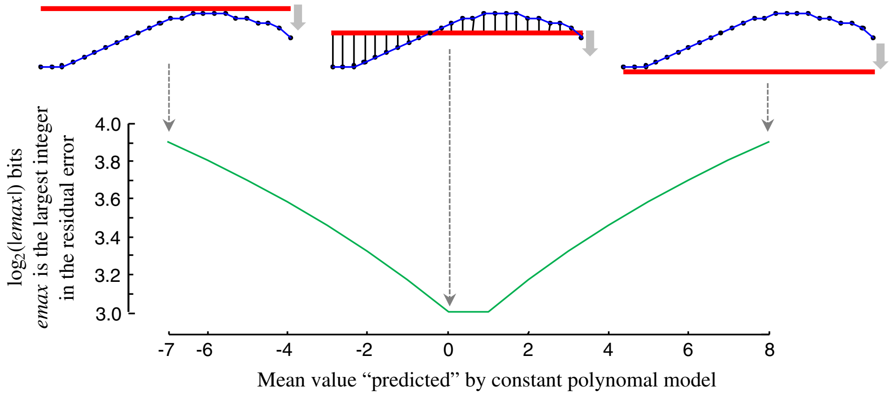


Fig. 7 The $\log_2$ of the range of the residual errors for all possible single constant polynomial models of the data introduced in Fig. 3. Note that the model that minimizes this value (with a tie) is also the model that minimizes the residual error

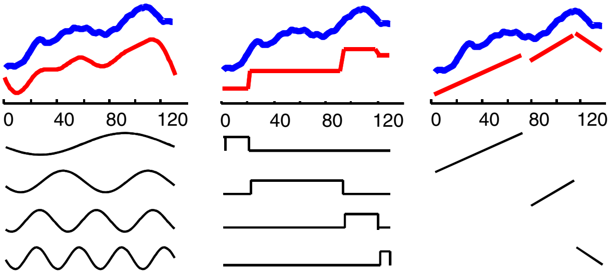


Fig. 8 A time series T shown in *bold/blue* and three different models of it shown in *fine/red*: from *left to right*: DFT, APCA, and PLA (Color figure online)

effect of considering fractional bits for this problem. Note that the factional bits have a narrow range of 3 to 4, and the Huffman encoding does not make any distinction here.

The reader can now appreciate our why "solution" to this issue was to simply ignore it. Because the underlying dimensionality reduction algorithms we are using (APCA, DFT, PLA) are attempting to minimize the residual error,[5] they are also implicitly minimizing the range of residuals. As shown by Fig. 7, if we explicitly added a term for the range of residuals it would have no effect, as the dimensionality reduction algorithm has already minimized it.

We have shown a detailed example using APCA. However, essentially all of the time series representations can be encoded in a similar way. As shown with three representative examples in Fig. 8, essentially all of the time series models consist of a set of basic functions (i.e., coefficients) that are linearly combined to produce an approximation of the data.

[5] DFT does minimize the residual error at any desired dimensionality given its set of basis functions. For both APCA and PLA, while there are algorithms that can minimize the residual error, they are too slow to use in practice. We use greedy approximation algorithms that are known to produce near optimal results (Keogh et al. 2011; Rakthanmanon et al. 2012).

As we apply our ideas to each representation, we must be careful to correctly “charge” each model for the number of parameters used in the model. For example, each APCA segment requires the mean value and length, whereas PLA segments require the mean value, segment length and slope. Each DFT coefficient can be represented by the amplitude and phase of each sine wave; however, because of the complex conjugate property, we get a “free” coefficient for each one we store (Camerra et al. 2010; Ding et al. 2008). In previous comparisons of the indexing performance of various time series representations, many authors have given an unfair advantage to one representation by counting the cost to represent an approximation incorrectly (Keogh and Pazzani 2000). The ideas in this work explicitly assume a fair comparison. Fortunately, the community seems to have become more aware of this issue in recent years (Camerra et al. 2010; Palpanas et al. 2008).

In the next section we give both the generic version of the MDL model discovery for time series algorithms and three concrete instantiations for DFT, APCA, and PLA.

3.2 Generic MDL for time series algorithms

In the previous section, we used a toy example to demonstrate how to compute the reduced description length of a time series with a competing hypothesis. In this section, we will show a detailed generic version of our algorithm, and then explain our algorithm in detail how we apply our algorithm to the three most commonly used time series representations.

Our algorithm not only discovers the intrinsic cardinality and dimensionality of an input time series, but it can also be used to find the right model or data representation for a given time series. Table 1 shows a high-level view of our algorithm for discovering the best model, cardinality, and dimensionality which will minimize the total number of bits required to store the input time series.

Because MDL is the core of our algorithm, the first step is to quantize a real-valued time series into a discrete-valued (but still fine-grained) time series, T (line 1). Next, we consider each model, cardinality, and dimensionality one by one (lines 3–5). Then, a hypothesis H is created based on the selected model and parameters (line 6). For example, a hypothesis H, shown in Fig. 5, is created when the model M = APCA, cardinality $c = 16$, and dimensionality $d = 2$; note that, in that case, the length of the input time series was $m = 24$.

The reduced description length is finally calculated (line 7), and our algorithm returns the model and parameters that minimize the reduced description length for encoding T (lines 8–13).

For concreteness, we will now consider three specific versions of our generic algorithm.

3.3 Adaptive Piecewise Constant Approximation

As we have seen Sect. 3.1, an APCA model is simple; it contains only constant segments. The pseudo code for APCA, shown in Table 2, is very similar to the generic

Table 1 Generic MDL algorithm for time series

```
Algorithm: Generic MDL algorithm for time series
Input:  TS:  time series
Output: intrinsic_model:  intrinsic model
        intrinsic_card :  intrinsic cardinality
         intrinsic_dim  :  intrinsic dimensionality

 1.T = Discretization(TS)
 2.bsf = ∞
 3.for all M in {APCA,PLA,DFT,MIXTURE}
 4.    for all cardinality c
 5.      for all dimensionality d
 6.          H = ModelRespresentation(T,M,c,d)
 7.         total_cost = DL(H)+ DL(T|H)
 8.         if (bsf > total_cost)
 9.             bsf = total_cost
 10.            intrinsic_model = M
 11.            intrinsic_card  = c
 12.            intrinsic_dim   = d
 13.         end if
 14.     end for
 15.    end for
 16. end for
```

algorithm. First of all, we quantize the input time series (line 1). Then, we evaluate all cardinalities from 2 to 256 and dimensionalities from 2 to the maximum, which is half of the length of the input time series TS (lines 3–4). Value m denotes the length of the input time series.

Note that if the dimensionality were more than $m/2$, some segments would contain only one point. Then, a hypothesis H would be created using the values of cardinality c and dimensionality d, as shown in Fig. 5, where $c = 16$ and $d = 2$. The model contains d constant segments, so the model cost is the number of bits required for storing d constant numbers, and $d - 1$ pointers to indicate the offset of the end of each segment (line 6). The difference between T and H is also required to rebuild T. The correction cost is computed; then the reduced description length is calculated from the combination of the model cost and the correction cost (line 7). Finally, the hypothesis that minimizes this value is returned as an output of the algorithm (lines 8–13).

Table 2 Our algorithm specific to APCA

```
Algorithm: Intrinsic Discovery for APCA
Input:  TS (time series)
Output: intrinsic_card ; intrinsic_dim

1. T = Discretization(TS)
2. bsf = ∞
3.    for  c = 2:256
4.       for d = 2 to m/2
5.           H = APCA(T,c,d)
6.           model_cost = $d*\log_2 c+(d-1)*\log_2 m$
7.           total_cost = model_cost + DL(T|H)
8.           if (bsf > total_cost)
9.               bsf = total_cost
10.              intrinsic_card  = c
11.              intrinsic_dim   = d
12.          end if
13.      end for
14.   end for
```

3.4 Piecewise Linear Approximation

An example of a PLA model is shown in Fig. 8—right. In contrast to APCA, a hypothesis using PLA is more complex because each segment contains a line of any slope, instead of a constant line in APCA. The algorithm used to discover the intrinsic cardinality and dimensionality for PLA is shown in Table 3, which is similar to the algorithm for APCA, except for the code in line 5 and 6.

A PLA hypothesis H is created from the external module *PLA* (line 5). To represent each segment in hypothesis H, we record the starting value, ending value, and the ending offset (line 6). The slope is not kept because storing a real number is more expensive than log_2c.

The first two values are represented in cardinality c and thus log_2c bits are required for each of them. We also require log_2m bits to point to any arbitrary offset in T. Thus, the model cost is shown in line 6. Finally, the reduced description length is calculated and the best choice is returned (lines 8–13).

3.5 Discrete Fourier Transform

A data representation in DFT space is simply a linear combination of sine waves, as shown in Fig. 8–left. Table 4 presents our algorithm specific to DFT. After we quantize

Table 3 Our algorithm specific to PLA

```
Algorithm: Intrinsic Discovery for PLA
Input:  TS (time series)
Output: intrinsic_card; intrinsic_dim

1. $T$ = Discretization (TS)
2. bsf = $\infty$
3. for   $c$ = 2:256
4.    for $d$ = 2 to $m/2$
5.         $H$ = PLA (T,c,d)
6.         model_cost = 2*$d$*$\log_2 c$ +($d$-1)*$\log_2 m$
7.         total_cost = model_cost + DL($T|H$)
8          if (bsf > total_cost)
9.            bsf = total_cost
10.           intrinsic_card = $c$
11.           intrinsic_dim = $d$
12         end if
13.   end for
14. end for
```

the input time series to a discrete time series T (line 1), the external module DFT is called to return the list of sine wave coefficients that represent T. The coefficients in DFT are a set of complex conjugates, so we store only half of all coefficients which contain complex numbers without their conjugate, called `half_coef` [line 5]. When `half_coef` is provided, it is trivial to compute their conjugates and obtain all original coefficients.

Instead of using all of `half_coef` to regenerate T, we test using subsets of them as the hypothesis to approximately regenerate T, incurring an approximation error. We first sort the coefficients by their absolute value (line 6). We use top-d coefficients as the hypothesis to regenerate T by using `InverseDFT` (line 8). For example, when `d=1` we use only the single most important coefficient to rebuild T, and when `d=2` the combination of top-two sine waves are used as a hypothesis, etc. However, it is expensive to use 16 bits for each coefficient by keeping two complex numbers for its real part and imaginary part. Therefore, in line 7, we reduce those numbers to just c possible values (cardinality) by rounding the number to the nearest integer in a space of size c, and we also need a constant number of bits (32 bits) for the maximum

Table 4 Our algorithm specific to DFT

```
Algorithm: Intrinsic Discovery for DFT
Input:     TS (time series)
Output:    intrinsic_card; intrinsic_dim

1. $T$ = Discretization (TS)
2. bsf = ∞
3. for  $c$ = 2:256
4.    for $d$ = 2 to $m/2$
5.         half_coef = DFT($T$)
6.         sorted_coef = SortByPolar(half_coef)
7.         round_coef = Round(sorted_coef, $c$)
8          $H$ = InverseDFT(round_coef(1:$n$))
9.         model_cost = 2*$d$*$\log_2 c$+d*$\log_2(m/2)$)+32
10.        total_cost = model_cost + DL($T|H$)
11.        if (bsf > total_cost)
12.            bsf = total_cost
13.            intrinsic_card = $c$
14.            intrinsic_dim  = $d$
15.        end if
16.   end for
17. end for
```

and minimum value of both the real parts and the imaginary parts. Hence, the model contains top-d coefficients whose real (and imaginary) parts are in a space of size c. Thus, the model cost and the reduced description length are shown in lines 9 and 10.

For simplicity we placed the external modules APCA, PLA, and DFT inside two for-loops; however, to improve performance, they should be moved outside the loops.

3.6 A mixed polynomial degree model

For a given time series T, we want to know the representation that can minimize the reduced description length for T. We have shown how to achieve this goal by applying the MDL principle to three different models (APCA, PLA and DFT). However, for

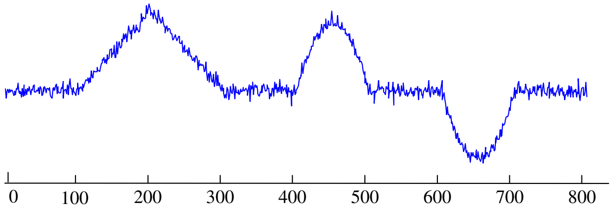


Fig. 9 A toy example of a time series that has more than one state

some complex time series, using only *one* of the above models may not be sufficient to achieve the most parsimonious representation, as measured by bit cost, or by our subjective understanding of the data (Lemire 2007; Palpanas et al. 2008). It has been shown that averaged over *many* highly diverse datasets; there is not much difference among the different representations (Palpanas et al. 2008). However, it is possible that within a single dataset, the specific model used could make a significant difference. For example, consider each of the two time series that form the trajectory of an automobile as it drives through Manhattan. These time series are comprised of a combination of straight lines and curves. We could choose just one of these possibilities, either representing the automobile's turns with many piecewise linear segments, or representing the long straight sections with a degenerate "curve." However, a mixed polynomial degree model is clearly more natural here.

For clarity, we show a toy example that can benefit from a mixed polynomial degree model in Fig. 9. It is easy to observe that there are constant, linear and quadratic patterns in this example. In Sects. 4.9 and 4.10, we further demonstrate the utility of our ideas on real datasets (Keogh et al. 2011; Lemire 2007; National Aeronautics and Space Administration 2011).

Several works propose that it may be fruitful to use a *combination* of different models within one time series (Keogh et al. 2011; Lemire 2007; Palpanas et al. 2008). For example, Lemire (2007) proposes a mixed model wherein the polynomial degree of each interval in one time series can vary. The polynomial degree can be zero, one, two or higher. The goal of Lemire (2007) is to minimize the Euclidean error between the model and the original data for a given number of segments. However, note that Lemire (2007) requires the user to state the desired dimensionality, something we obviously wish to avoid. Minimizing the Euclidean error between the model and the original data *is* a useful objective function for some tasks, but it is not necessarily the same as discovering the intrinsic dimensionality, which is our stated goal. In the following, we show that our proposed algorithm returns the intrinsic model by minimizing the reduced description length using MDL. Moreover, our algorithm is essentially parameter-free.

We propose a mixed model framework using MDL that optimizes a mixture of constant, linear and quadratic representations for different local regions of a single time series. In this case, the operator space of the segmentation algorithm (Table 6) becomes larger. Table 5 shows a high-level view of the algorithm. Lines 1–4 are similar to the algorithm for APCA and PLA. The function in line 5 is the return for the d segments with the hypothesis H, the model cost and the starting point of each

Table 5 Our algorithm specific to the mixed polynomial degree model

```
Algorithm: Intrinsic Model Discovery for the mixed polynomial degree representations
Input: TS (time series)
Output: intrinsic_card; intrinsic_dim

1.  $T$ = Discretization (TS)
2.  bsf = ∞
3.  for  $c$ = 2:256
4.     for $d$ = 2 to $m/2$
5.        segment_info= bottom_up_mixed($T$,$c$,$d$)     // See Table 6
6.        $H$ = sum(segment_info.$H$)
7.        model_cost = sum(segment_info.model_cost) + ($d$-1)*$\log_2 m$
8.        total_cost = model_cost +DL($T|H$)
9.       if (bsf > total_cost)
10.        bsf = total_cost
11.        intrinsic_card = $c$
12.        intrinsic_dim  = $d$
13.      end if
14.   end for
15. end for
```

segment. Table 6 illustrates how the bottom-up mixed polynomial degree algorithm works in detail. Each segment is represented by a different polynomial degree to minimize the reduced description length. The model costs for constant, linear and quadratic representations are $1\log_2 c$ bits, $2{*}\log_2 c$ bits and $3{*}\log_2 c$ bits, respectively. For example, if c is 256, the model costs for the above three representations are 8 bits, 16 bits and 24 bits, respectively. In line 7, In addition to the total model cost of all of the segments, the model cost of the whole time series needs to use extra bits to store the starting point of each segment. Observe that the model cost for a segment is independent of the length of the segment. More specifically, the model cost for each segment is only determined by the polynomial degree of the representation and the cardinality c.

Table 6 shows the bottom-up mixed polynomial degree model algorithm. By choosing the minimum description costs as the objection function, the algorithm shown in Table 6 is a generalization of the bottom-up algorithm for generating the PLA introduced in Keogh et al. (2011). There are two main differences between our bottom-up mixed model algorithm and the bottom-up algorithm described in Keogh et al. (2011). The first is a minor pragmatic point: instead of using two points in the finest possible approximation, the algorithm shown in Table 6 uses three points. This is because when

Table 6 Bottom-up mixed polynomial degree model algorithm

```
Algorithm: Bottom-up algorithm for mixed polynomial degree model
Input:  TS (time series), c, d
Output: Seg_TS

1. T = Discretization (TS)
2. for i = 1:3:length(T)
3.   Seg_TS = concat(Seg_TS, T([i:i+2]))
4. end
5. for  i = 1: length(Seg_TS) - 1
     //Find the merging cost of each pair of segments
6.    merge_cost(i) = calculate_MDL_cost(merge(Seg_TS(i), Seg_TS(i+1)),c);
7. end
8. while length(segment) > d
9.    ind = min(merge_cost)                      // Find cheapest pair to merge
10.   Seg_TS(i) = merge(Seg_TS(ind), Seg_TS(ind+1)))   // Merge them
11.   delete(Seg_TS(ind+1))                           // Update records
12.   merge_cost(i)= calculate_MDL_cost(Seg_TS(i),  Seg_TS(i+1)),c)
13.   merge_cost(i-1)=calculate_MDL_cost(merge(Seg_TS(i-1), Seg_TS(i)),c)
14. end
```

the polynomial degree of the representation is two, the number of points by using this approximation must be at least three. Second, instead of using Euclidean distance as the objective function, the algorithm in Table 6 uses MDL cost. The algorithm calculates the MDL costs for three degrees of polynomial degree representations for a segment. The polynomial degrees are zero, one and two, respectively. Next, it chooses the one that can minimize the cost (the description length). The algorithm begins by creating the finest possible approximation of the input time series. So for a length of n time series, there are $n/3$ segments after this step, as shown in Table 6, lines 2–4. Then the cost of merging each pair of adjacent segments is calculated, as shown in lines 5–7. To minimize the merging cost for the two inputsegments, this calculate_MDL_cost function calculates the MDL costs for three kinds of polynomial degree representations, and then chooses the minimum one as the merging cost (line 6). After this step, the algorithm iteratively merges the lowest cost pair until a stopping criterion is met. In this scenario, the stopping criterion is the input number of segments. This means that the algorithm will not terminate as long as the current number of segments is larger than the input number of segments.

It is important to note that, similar to the algorithm (Keogh et al. 2011), our algorithm is greedy in the sense that once two regions have been joined together in a single segment, they will remain together in that segment (which may get larger as it is iteratively joined with other segments). There are only *join* operators; there are no *split* operators. However, if a region in our algorithm is initially assigned to a polynomial

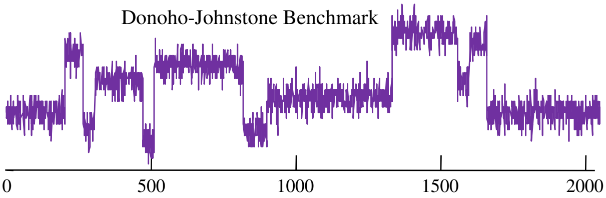


Fig. 10 A version of the Donoho–Johnstone block benchmark created 10 years ago and downloaded from Sarle (1999)

of a particular degree, this does not mean it cannot later be subsumed into a larger segment of a different degree. In other words, a tiny region that locally may consider itself, say, linear has the ability to later become part of a constant or quadratic segment as it obtains a more “global” view.

4 Experimental evaluation

To ensure that our experiments are *easily* reproducible, we have set up a website which contains all data and code, together with the raw spreadsheets of the results (www.cs.ucr.edu/~bhu002/MDL/MDL.html). In addition, this website contains additional experiments that are omitted here for brevity.

4.1 A detailed example on a famous problem

We start with a simple sanity check on the classic problem specifying the correct time series model, cardinality and dimensionality, given an observation of a corrupted version of it. While this problem has received significant attention in the literature (Donoho and Johnstone 1994; Salvador and Chan 2004; Sarle 1999), our MDL method has two significant advantages over existing works. First, there is no explicit parameter to set, whereas most other methods require several parameters to be set. Second, MDL helps to specify the model, cardinality and dimensionality, whereas other methods typically only consider the model and/or dimensionality.

To eliminate the possibility of data bias (Keogh and Kasetty 2003) we consider a ten-year-old instantiation (Sarle 1999) of a classic benchmark problem (Donoho and Johnstone 1994). In Fig. 10 we show the classic Donoho–Johnstone block benchmark. The underlying model used to produce it consists of 12 piecewise constant sections with Gaussian noise added.

The task is challenging because some of the piecewise constant sections are very short and thus easily dismissed during a model search. Dozens of algorithms have been tested on this time series (indeed, on this *exact* instance of data) in the last decade: which should we compare to? Most of these methods have several parameters, in some cases as many as six (Firoiu and Cohen 2002; García-López and Acosta-Mesa 2009). We argue that comparisons to such methods are inappropriate, since our *explicit* aim is to introduce a parameter-free method. The most cited *parameter-free* method addressing this problem is the L-Method (Salvador and Chan 2004). In essence, the

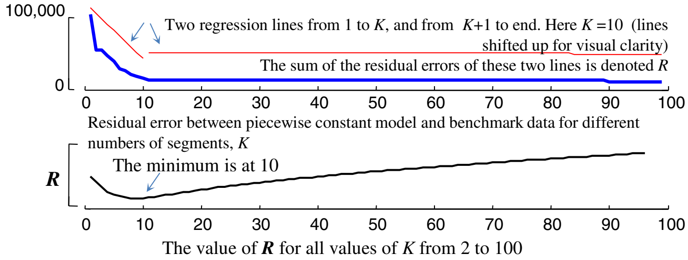


Fig. 11 The knee-finding L-Method. *Top* A residual error versus size-of-model curve (*blue/bold*) is modeled by all possible pairs of regression lines (*red/light*). Here, just one possibility is shown. *Bottom* The location that minimizes the summed residual error of the two regression lines is given as the optimal "knee" (Color figure online)

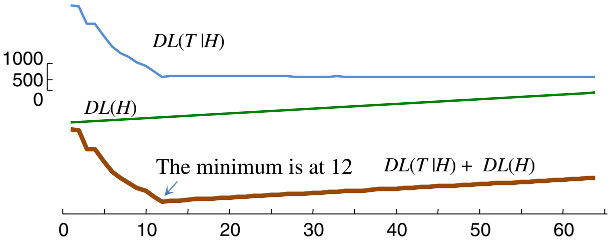


Fig. 12 The description length of the Donoho–Johnstone block benchmark time series is minimized at a *dimensionality* corresponding to 12 piecewise constant segments, which is the correct answer (Sarle 1999)

L-Method is a "knee-finding" algorithm. It attempts to explain the residual error vs. size-of-model curve using all possible pairs of two regression lines. Figure 11—top shows one such pair of lines, from *one to ten* and from *eleven to the end*. The location that produces the minimum sum of the residual errors of these two curves, ***R***, is offered as the optimal model. As we can see in Fig. 11—bottom, this occurs at location ten, a reasonable estimate of the true value of 12.

We also tested several other methods, including a recently-proposed Bayesian Information Criterion-based method that we found predicted a too coarse four-segment model (Zhao et al. 2008). No other parameter-free or parameter-lite method we found produced *intuitive* (much less *correct*) results. We therefore omit further comparisons in this paper (however, many additional experiments are available at www.cs.ucr.edu/~bhu002/MDL/MDL.html).

We solve this problem with our MDL approach. Figure 12 shows that of the 64 different piecewise constant models it evaluated, MDL selected the 12-segment model, which is the *correct* answer.

The figure above uses a cardinality of 256, but the same answer is returned for (at least) every cardinality from 8 to 256.

Beyond outperforming other techniques at the task of finding the correct *dimensionality* of a model, MDL can also find the intrinsic *cardinality* of a dataset, something for which methods (Salvador and Chan 2004; Zhao et al. 2008) are not even defined. In Fig. 13 we have repeated the previous experiment, but this time fixing the dimen-

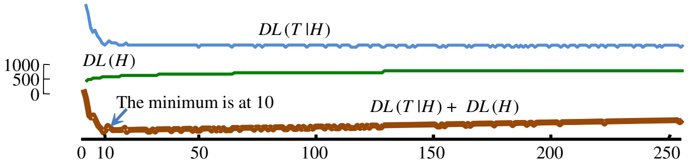


Fig. 13 The description length of the Donoho–Johnstone block benchmark time series is minimized with a *cardinality* of 10, which is the true cardinality (Sarle 1999)

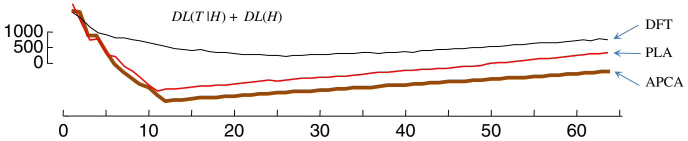


Fig. 14 The description length of the Donoho–Johnstone block benchmark time series is minimized with a piecewise constant model (APCA), not a PLA or Fourier representation (DFT)

sionality to 12 as suggested above, and testing all possible cardinality values from 2 to 256.

Here MDL indicates a cardinality of ten, which is the *correct answer* (Sarle 1999). We also re-implemented the most referenced *recent* paper on time series discretization (García-López and Acosta-Mesa 2009). The algorithm is stochastic, and requires the setting of five parameters. In one hundred runs over multiple parameters we found it consistently underestimated the cardinality of the data (the mean cardinality was 7.2).

Before leaving this example, we show one further significant advantage of MDL over existing techniques. Both Salvador and Chan (2004) and Zhao et al. (2008) try to find the optimal dimensionality, *assuming* the underlying model is known. However, in many circumstances we may not know the underlying model. As we show in Fig. 14, with MDL we can relax even this assumption. If our MDL scoring scheme is allowed to choose over the cross product of model = {APCA, PLA, DFT}, dimensionality = {1 to 512} and cardinality = {2 to 256}, it correctly chooses the right model, dimensionality *and* cardinality.

4.2 An example application in physiology

The *Muscle* dataset studied by Mörchen and Ultsch (2005) describes the muscle activation of a professional inline speed skater. The authors calculated the muscle activation from the original EMG (electromyography) measurements by taking the logarithm of the energy derived from a wavelet analysis. Figure 15—top shows an excerpt. At first glance it seems to have two states, which correspond to our (perhaps) naive intuitions about skating and muscle physiology.

We test this binary assumption by using MDL to find the model, dimensionality and cardinality. The results for the model and dimensionality are objectively correct, as we might have expected given the results in the previous section, but the results for cardinality, shown in Fig. 16—left, are worth examining.

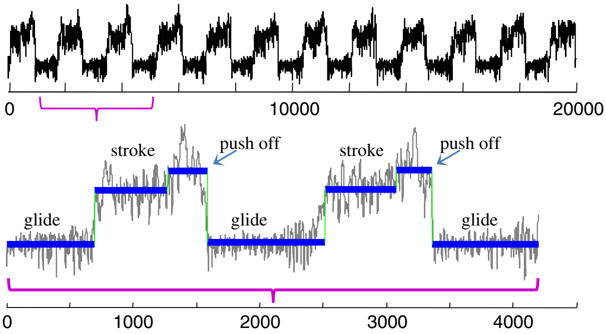


Fig. 15 *Top* An excerpt from the Muscle dataset. *Bottom* A zoomed-in section of the Muscle dataset which had its model, dimensionality and cardinality set by MDL

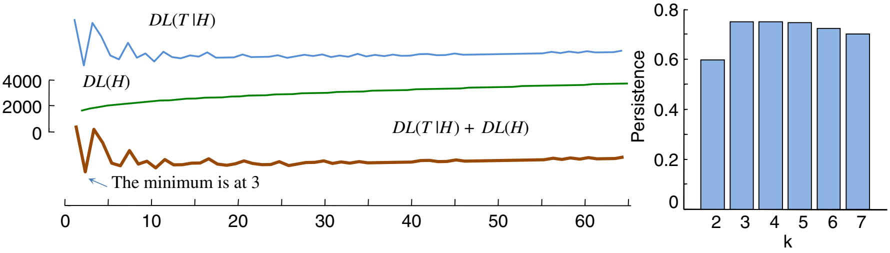


Fig. 16 *Left* The description length of the muscle activation time series is minimized with a cardinality of three, which is the correct answer. *Right* The Persist algorithm, using the code from Mörchen and Ultsch (2005), predicts a value of four

Our MDL method suggests a cardinality of three. Glancing back at Fig. 15—bottom shows why. At the end of the *stroke* there is an additional level corresponding to an additional *push-off* by the athlete. This feature was noted by physiologists who worked with Mörchen and Ultsch (2005). However, their algorithm weakly predicts a value of four.[6] Here, once again we find the MDL can outperform this latter approach, even though Mörchen and Ultsch (2005) acknowledges that their reported result is the best obtained after some parameter tuning using additional data from the same domain.

4.3 An example application in astronomy

In this section (and the one following) we consider the possible utility of MDL scoring as an anomaly detector. Building an anomaly detector using MDL is very simple. We can simply record the best model, dimensionality and/or cardinality predicted for the training data, and then test on future observations that have significantly different learned parameters. We can illustrate this idea with an example in astronomy. We begin by noting that we are merely demonstrating an additional possible application of our ideas. We are only showing that we can *reproduce* the utility of existing works.

[6] The values for $k = 3$, 4 or 5 do not differ by more than 1 %.

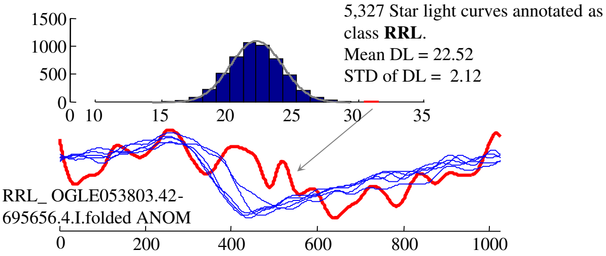


Fig. 17 *Top* The distribution of intrinsic dimensionalities of *star light curves*, estimated over 5,327 human-annotated examples. *Bottom* Three typical examples of the class RRL, and a high intrinsic dimensionality example, labeled as an outlier by Protopapas et al. (2006)

However note that our technique is at least as fast as existing methods (Protopapas et al. 2006; Rebbapragada et al. 2009), and does not require any training data or parameter tuning, an important advantage for exploratory data mining.

Globally there are hundreds of telescopes covering the sky and constantly recording massive amounts of astronomical data (Protopapas et al. 2006). Moreover, there is a worldwide effort to digitize tens of millions of observations recorded on formats ranging from paper/pencil to punch cards over the last hundred years. Having humans manually inspect all such observations is clearly impossible (Malatesta et al. 2005). Therefore, outlier detection can be useful to catch anomalous data, which may indicate an exciting new discovery or just a pedestrian error. We took a collection of 1,000 hand-annotated RRL variable stars (Protopapas et al. 2006; Rebbapragada et al. 2009), and measured the mean and standard deviation of the DFT dimensionality, which turned out to be 22.52 and 2.12, respectively. As shown in Fig. 17—top, the distribution is Gaussian.

We then took a test set of 8,124 objects, known to contain at least one anomaly, and measured the intrinsic DFT dimensionality of all of its members, and discovered that one had a value of 31. As shown in Fig. 17—bottom, the offending curve looks different from the other data, and is labeled *RRL_OGLE053803.42-695656.4.I.folded* ***ANOM***. This curve *is* a previously known anomaly. In this case, we are simply able to reproduce the anomaly finding ability of previous work (Protopapas et al. 2006; Rebbapragada et al. 2009). However, we achieved this result without extensive parameter tuning, and we can do so *very* efficiently.

4.4 An example application in cardiology

In this section we show how MDL can be used to mine ECG data. Our intention is not to produce a definitive method for this domain, but simply to demonstrate the utility and generality of MDL. We conducted an experiment that is similar in spirit to the previous section. We learned the mean and standard deviation of the DFT dimensionality on 200 normal heartbeats, finding them to be 20.82 and 1.70, respectively. As shown in Fig. 18—top, the distribution is clearly Gaussian. We used these learned values to monitor the rest of the data, flagging any heartbeats that had a dimensionality that

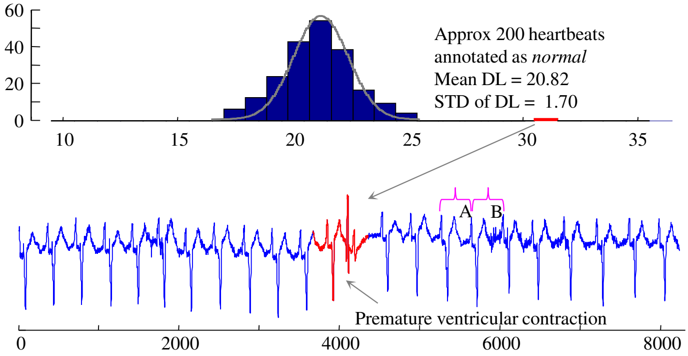


Fig. 18 *Top* The distribution of intrinsic dimensionalities of individual heartbeats, estimated over the 200 normal examples in record 108 of the MIT BIH Arrhythmia Database (*bottom*)

was more than three standard deviations from the mean. Figure 18—bottom shows a heartbeat that was flagged by this technique.

Once again, here we are simply reproducing a result that could be produced by other methods (Yankov et al. 2008). However, we reiterate that we are doing so without any parameter tuning. Moreover, it is interesting to note when our algorithm does *not* flag innocuous data (i.e., produces false positives). Consider the two adjacent heartbeats labeled A and B in Fig. 18—bottom. It happens that the completely normal heartbeat B has significantly more noise than heartbeat A. Such non-stationary noise presents great difficulties for distance-based and density-based outlier detection methods (Yankov et al. 2008), but MDL is essentially invariant to it. Likewise, the significant wandering baseline (not illustrated) in parts of this dataset has no medical significance and is ignored by MDL, but it is the bane of many EEG anomaly detection methods (Chandola et al. 2009).

4.5 An example application in geosciences

Global-scale Earth observation satellites such as the Defense Meteorological Satellite Program (DMSP) Special Sensor Microwave/Imager (SSM/I) have provided temporally detailed information about the Earth's surface since 1978, and the National Snow and Ice Data Center (NSIDC) in Boulder, Colorado makes this data available in real time. Such archives are a critical resource for scientists studying climate change (Picard et al. 2007). In Fig. 19, we show a brightness temperature time series from a region in Antarctica, using SSM/I daily observations over the 2001–2002 austral summer.

We used MDL to search this archive for low complexity annual data, reasoning that low complexity data might be amenable to explanation. Because there is no natural starting point for a year, for each time series we tested every possible day as the starting point. The simplest time series we discovered required a piecewise constant dimensionality of two with a cardinality of two, suggesting that a very simple process created the data. Furthermore, the model discovered (piecewise constant) is somewhat surprising, since virtually all climate data is sinusoidal,

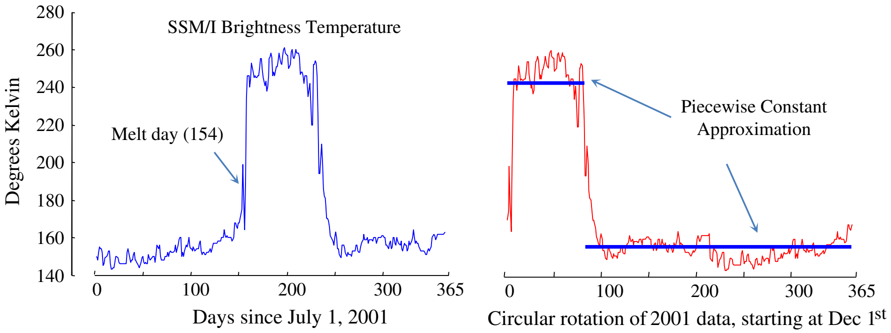
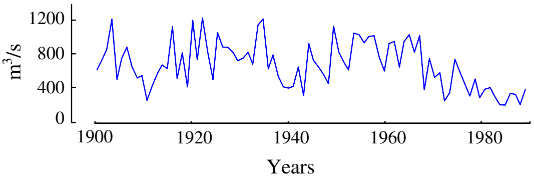


Fig. 19 *Left* A time series of temperatures in a region of Antarctica. *Right* Of the hundreds of millions of such time series archived at NSIDC, this time series (and a few thousand more) is unusual in that it has a very low complexity, being best modeled with just two linear segments

Fig. 20 A time series showing the annual discharge rate of Senegal River from the year 1903 to 1988



reflecting annual periodicity; thus, we were intrigued to find an explanation for the data.

After consulting some polar climate experts, the following explanation emerges. For most of the year the location in question is covered in snow. The introduction of a small amount of *liquid* water will significantly change the reflective properties of the ground cover, allowing the absorption of more heat from the sun, thus producing more liquid water in a rapid positive feedback cycle. This explains why the data does not have a sinusoidal shape or a gradual (say, linear) rise, but a fast phase change, from a mean of about 155 Kelvin to a ninety-day summer of about 260 Kelvin.

4.6 An example application in hydrology and environmental science

In this section we show two applications of our algorithm in hydrological and environmental domains.

The first application is the hydrology data studied by Kehagias (2004) describing the annual discharge rate of the Senegal River. This data was measured at Bakel station from the year 1903 to 1988 (Kehagias 2004), as shown in Fig. 20.

The authors of Kehagias (2004) reported the optimal segmentation occurs when the number of segments is five using a Hidden Markov Model (HMM)-based segmentation algorithm, as shown in Fig. 21—top.

As illustrated in Fig. 21—bottom, we get a similar plot by *hard coding* the number of segments to five, using the MDL-based APCA algorithm shown in Table 2. However, as shown in Fig. 22, our MDL algorithm actually predicts *two* as its intrinsic

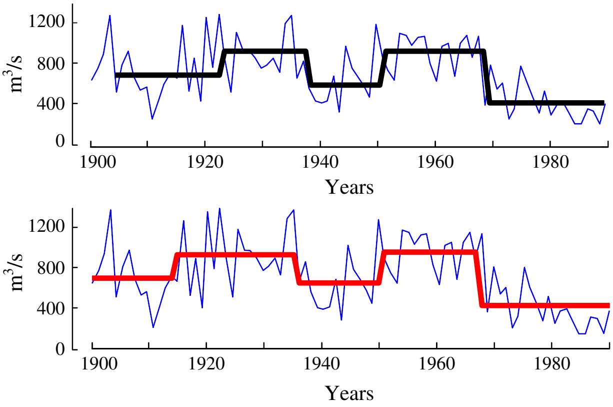


Fig. 21 *Top* The *blue/light line* is Senegal River data. The *black/bold line* is the segmentation result found in Section 5.1 of Kehagias (2004). *Bottom* We obtained the *red/bold line* by hard coding the number of segments to five using the MDL algorithm (Color figure online)

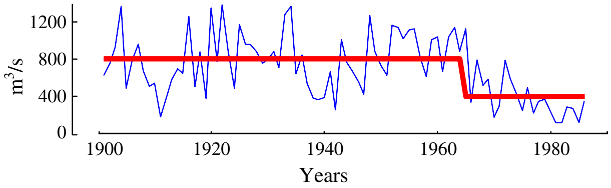


Fig. 22 Our MDL algorithm predicts that the intrinsic dimensionality of the annual discharge rate of the Senegal River is two. The approximation is shown in *red/bold* (Color figure online)

dimensionality of data in Fig. 20. Note that there is no ground truth for this problem.[7] Nevertheless, for the Donoho–Johnstone block benchmark dataset that has ground truth in Sect. 4.1, we have correctly predicted its intrinsic dimensionality, and we would argue that the two-segment solution shown in Fig. 22 is at least subjectively as plausible as the five segment solution.

Below we consider an application of our algorithm in a similar domain in environmental data. The data we consider is the time series of the annual global temperature change from the year 1700 to 1981 (Kehagias 2004), as shown in Fig. 23.

In Kehagias (2004) the authors suggest that the optimal segmentation occurs when the number of segments is four, as shown in Fig. 24—top.

As illustrated in Fig. 24—bottom, using the algorithm in Table 2, we obtain a very similar plot by hard coding the number of segments to four. As before there is no external ground truth explanation as to why the optimal segmentation of this global annual mean temperature time series should be four. However, as shown in Fig. 25, our MDL algorithm predicts that the intrinsic dimensionality of the global temperature

[7] In Kehagias (2004) authors claimed that to obtain the optimal segmentation, the number of segments should be five. This claim is very subjective, simply because this “optimal” segmentation is with respect to the total deviation from segment means. Moreover, there is no hydrological interpretation of the five segments with regard to the real data.

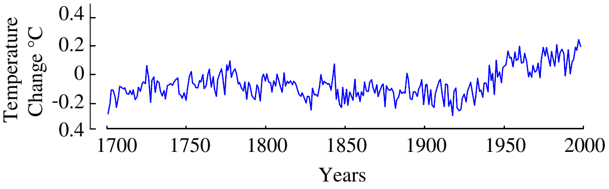


Fig. 23 A time series showing the annual global temperature change from the year 1700 to 1981

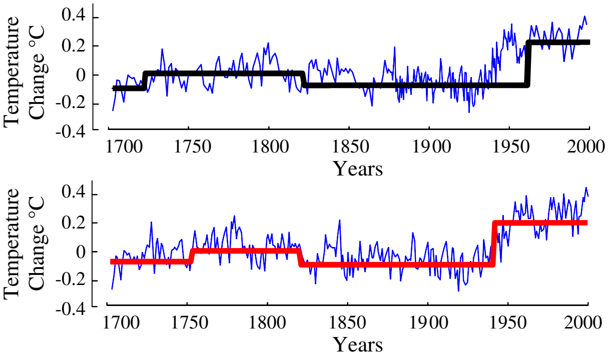


Fig. 24 *Top* The *blue*/*light line* is the annual global temperature change. The *black*/*bold line* is the segmentation result found in Section 5.2 of Kehagias (2004). *Bottom* We obtained a similar but slightly different model, as shown in the *red*/*bold line*, by hard coding the number of segments to four using the MDL algorithm (Color figure online)

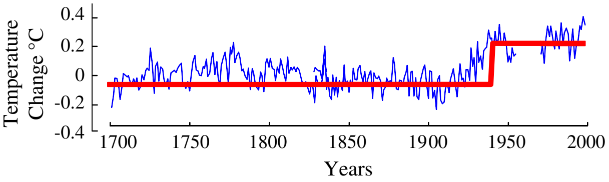


Fig. 25 Our MDL algorithm obtains *two* as the intrinsic dimensionality of the time series for the global annual mean temperature

change data is *two*. Here, there is at least some tentative evidence to support our two segment model. Research done by Linacre and Geerts (2011), National Aeronautics and Space Administration (2011) and US Environmental Protection Agency (2011) suggests that there was a global temperature rise between 1910 and 1940. To be more precise, this period of rapid warming was from 1915 to 1942 (Linacre and Geerts 2011; US Environmental Protection Agency 2011). Moreover, there were no significant temperature changes after 1700 other than during this rapid warming period.

4.7 An example application in biophysics

In Bronson et al. (2009), the authors also proposed an HMM-based approach (distinct from, but similar in spirit to that described in Kehagias (2004) and discussed in

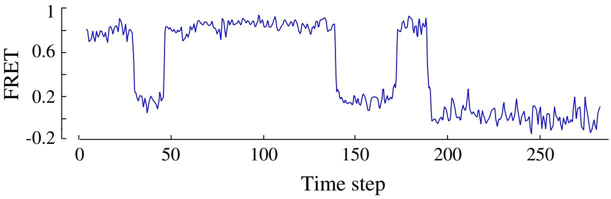


Fig. 26 A representative smFRET trace from Bronson et al. (2009) and vbFRET Toolbox (2012)

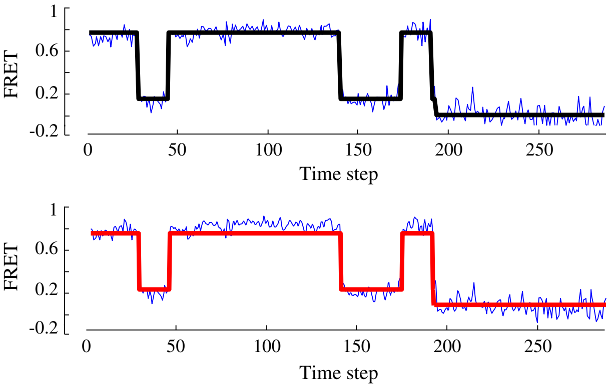


Fig. 27 *Top* The time series in *blue* from Fig. 26 is predicted to have three states (Bronson et al. 2009; vbFRET Toolbox 2012). The approximation is shown in *black*/*bold*. *Bottom* Our algorithm also finds three as the intrinsic cardinality. Piecewise constant approximation is shown in *red*/*bold* (Color figure online)

the previous section) to segment the time series from single-molecule Förster resonance energy transfer (smFRET) experiments. Figure 26 shows a time series from the smFRET experiment (Bronson et al. 2009; vbFRET Toolbox 2012).

The authors in Bronson et al. (2009) noted that there are biophysical reasons to think that the data is intrinsically piecewise constant, but the number of states is unknown. Their method suggests that there are three states in the above time series, as shown in Fig. 27—top). We obtain the same results, as our algorithm finds that the intrinsic cardinality for the data in Fig. 26 is also three using the algorithm in Table 2. Figure 27—bottom illustrates our approach. However, there are several parameters in the HMM-based approach used by Bronson et al. (2009). Moreover, their approach iteratively finds the number of states with the maximum likelihood, which results in a very slow algorithm. In contrast, our algorithm is parameter-free and significantly faster. Note that we do not even have to have the assumption (made by Bronson et al. 2009) that the data is piecewise constant. The mixed polynomial algorithm introduced in Sect. 3.6 considered and discounted a linear, quadratic or mixed polynomial model to produce the model shown in Fig. 27—bottom.

4.8 An example application in prognostics

In this section, we demonstrate our framework's ability to aid in clustering problems.

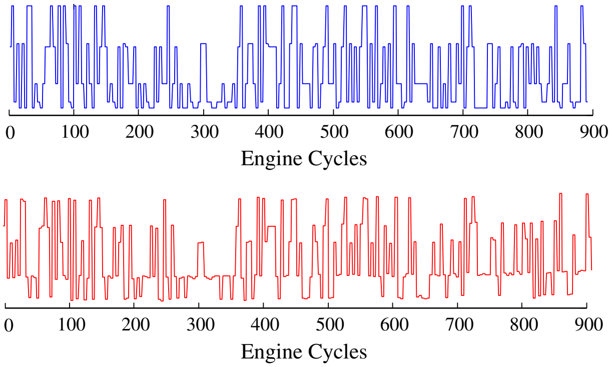


Fig. 28 *Top* An example of an operational variable in the PHM08 dataset. *Bottom* An example of a non-operational variable in the PHM08 dataset

Recently, the field of prognostics for engineering systems has attracted a huge amount of attention due to its ability to provide an early warning for system failures, forecast maintenance as needed, and estimate the remaining useful life of a system (Goebel et al. 2008; Prognostics Center of Excellence, National Aeronautics and Space Administration (NASA) 2012; Vachtsevanos et al. 2006; Wang et al. 2008; Wang and Lee 2006). Data-driven prognostics are more useful than model-driven prognostics, since a model-driven prognostic requires incorporating a physical understanding of the systems (Goebel et al. 2008). This is especially true when we have access to large amounts of data, a situation that is becoming more and more common.

There may be thousands of sensors in a single engineering system. Consider, for example, a typical oil-drilling platform that can have 20,000–40,000 sensors on board (IBM 2012). All of these sensors stream data about the health of the system (IBM 2012). Among the huge number of variables, there are some variables called operational sensors that have a substantial effect on system performance. In order to do a prognostic analysis, first the operational variables should be filtered from the non-operational variables that are just responding to the operational ones (Wang et al. 2008).

We analyzed the Prognostics and Health Management Challenge (PHM08) dataset which contains data from 233 different engines (PHM Data Challenge Competition 2008; Prognostics Center of Excellence, National Aeronautics and Space Administration (NASA) 2012). Each engine has data from around 900 engine cycles for one aircraft. Each engine cycle represents one aircraft flying from one destination to another. Figure 28 implies that the data from the operational variable and the non-operational variable are visually very similar.

The defined task here is to cluster the operational variables and non-operational variables into two groups. For ease of exposition, we only consider one variable in each group. Nevertheless, our framework can be easily extended to multivariate problems. We calculate the intrinsic cardinality and the reduced description length for one operational variable and one non-operational variable from all of the 233 engines. One marker in Fig. 29 represents one variable from one engine.

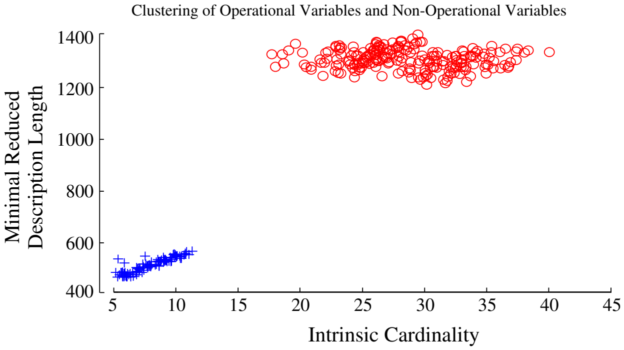


Fig. 29 The *blue/cross markers* represent operational variables. The *red/circle* markers represent non-operational variables. The variables from 233 engines are analyzed in the plot (Color figure online)

Although data from the two kinds of variables look very similar (Fig. 28), our results in Fig. 29 show that there is a significant difference between them: the operational variables lie in the lower left corner of Fig. 29 with low cardinalities and small reduced description lengths. In contrast, the non-operational variables lie in the upper right corner of Fig. 29 with high cardinalities and large reduced description lengths. This implies that the data from the operational variables is relatively 'simple' compared to the data from the non-operational variables, since the intrinsic cardinalities and reduced description lengths of the data from the operational variables are relatively small. This result was confirmed by a Prognostics expert: the hypothesis for filtering out the operational variables is that data from operational variables tends to have simpler behavior, since there are only several crucial states for the engines (Heimes and BAE Systems 2008; PHM Data Challenge Competition 2008; Prognostics Center of Excellence, National Aeronautics and Space Administration (NASA) 2012; Wang et al. 2008; Wang and Lee 2006). Note that in our experiment we did not need to tune any parameters, while most of the related literature for this dataset use multi-layer perceptron neural networks (Heimes and BAE Systems 2008; Wang et al. 2008; Wang and Lee 2006), which have the overhead of parameter tuning and are prone to overfitting.

4.9 Testing the mixed polynomial degree model

In Sect. 3.6 we introduced an algorithm for finding mixed polynomial degree models for a time series. In this section, we demonstrate the application of our proposed algorithm to the synthetic time series shown in Fig. 9. As shown in Fig. 30, we calculated its intrinsic dimensionality using the algorithms in Tables 5 and 6. The minimum cost occurs when the dimensionality is eight.

Figure 31 shows the data and its intrinsic mixed polynomial degree representations. In Fig. 31—bottom, we observed that there are four constant segments, two linear segments and two quadratic segments, which correctly reflects how we constructed this toy data.

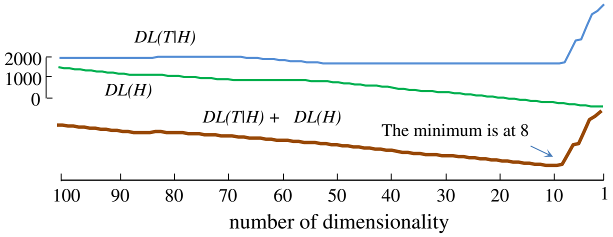


Fig. 30 The description length of the synthetic time series shown in Fig. 9 minimizes when the dimensionality is eight, which is the intrinsic dimensionality

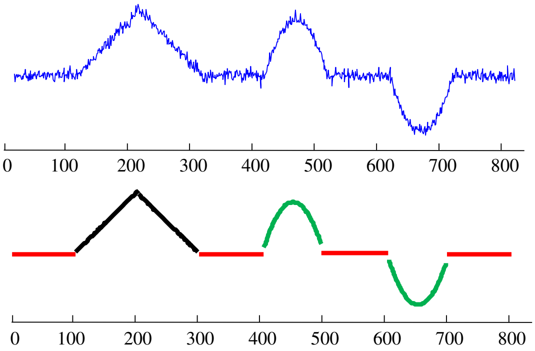


Fig. 31 *Top* A toy time series shown in Fig. 9 has constant, linear and quadratic segments. *Bottom* data in *top* is represented by a mixed polynomial degree model. The segments are brushed with *different colors* according to the polynomial degree of the representations. *Red* indicates a constant representation. *Black* indicates a linear representation and *green* indicates a quadratic representation (Color figure online)

4.10 An example application in aeronautics

Having demonstrated the mixed polynomial degree model on a toy problem, we are now ready to consider a real-world dataset. The Space Shuttle dataset contains time series produced by the inertial navigation system error correction system, as shown in Fig. 32.

Using only one approximation to represent the time series like the ones in Fig. 32 does not achieve a natural segmentation, since the data itself is intrinsically composed of more than one state. We applied the mixed polynomial degree algorithm in Tables 5 and 6 to the data shown in Fig. 32. The algorithm returns different polynomial degrees for different segments, as demonstrated in Fig. 33. We observe that there are three different states in both of the two time series shown in Fig. 32. An understanding of the processes that produced this data seems to support this result (Keogh et al. 2006).

4.11 Quantifiable experiments

We conclude this section with a set of *quantifiable* experiments that explicitly allow us to demonstrate the robustness of our algorithm to various factors that can cause it to fail.

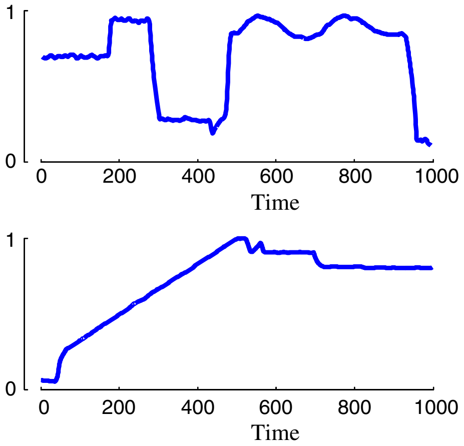


Fig. 32 *Top* One snippet of a space shuttle time series that clearly has more than one state. *Bottom* Another space shuttle time series that has more than one state

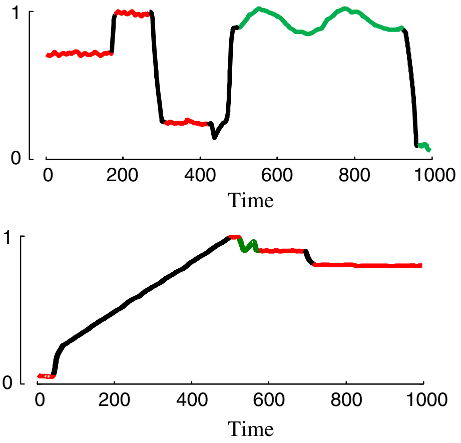


Fig. 33 The data shown in Fig. 32 after we applied our mixed polynomial degree segmentation. The segments are brushed with *different colors* according to the polynomial degree of the representations. *Red* indicates a constant representation. *Black* indicates a linear representation and *green* indicates a quadratic representation (Color figure online)

In every case, we push our algorithm *passed* its failure point, and by archiving *all* data (www.cs.ucr.edu/~bhu002/MDL/MDL.html) we establish baselines for researchers to improve on our results. We are particularly interested in measuring our algorithms sensitivity to:

- *Noise* The results shown in Sects. 4.1 and 4.2 suggest that our framework is at least somewhat robust to noise, but it is natural to ask at what point it breaks down, and how gracefully it degrades.
- *Sampling rate* For many applications the ubiquity of cheap sensors and memory means that the data is sampled at a rate higher than any practical application needs. For example, in the last decade most ECG data has gone from being sampled at 256Hz to sampling rates of up to 10KHz, even though there is little evidence that this aids analysis in any way. Nevertheless, there are clearly situations in which the data may be sampled at a lower rate than the ideal, and again we should consider how gracefully our method degrades.

- *Model assumptions* While we have attempted to have our algorithm as free of assumptions/parameters as possible, we still must specify the model class (es) to search over, i.e. DFT, APCA, and PLA. Clearly even if we had noise-free data, the data may not be exactly a “platonic idea” created from pure components of our chosen model. Thus we must ask ourselves how much our model assumptions can be violated before our algorithm degrades.

To test these issues we created modifications of the Donoho–Johnstone block benchmark (Sarle 1999). Our version is essentially identical to the version shown in Fig. 10, except it initially has no noise. We call this initial prototype signal P. After adding various distortions/modifications to the data, we can measure the success of our algorithm in three ways:

- The *Root-Mean-Square-Error* (RMSE). This is the average of the sum of squared differences between P and the predicted output of our algorithm. This is essentially the mean of square lengths of the gray hatch lines shown in Fig. 5. While zero clearly indicates a perfect recreation of the data, the absolute value of RMSE otherwise has little intuitive value. However the *rate* at which it changes due to a distortion is of interest here. This measure is shown with blue lines in Fig. 34.
- *Correct cardinality prediction* This is a binary outcome, either our algorithm predicted the correct cardinality or it did not. This is shown with black lines in Fig. 34.
- *Correct dimensionality prediction* This is also a binary outcome, either our algorithm predicted the correct dimensionality or it did not. Note that we only count a correct prediction of dimensionality if *every* segment endpoint is within three data points of the true location. This measure is shown with red lines in Fig. 34.

As shown in Fig. 34 our strategy is to begin with an easy case for our algorithm and progressively add more distortion until *both* the cardinality and dimensionality predictions fail. Concretely:

- In Fig. 34—*top* we start with the initial prototype signal P which has no noise, and we add noise until the signal-to-noise (SNR) ratio is -4.0. The SNR is calculated according to the standard equation in Signal to Noise Ratio (http://en.wikipedia.org/wiki/Signal-to-noise_ratio). As we can see, the cardinality prediction fails at a SNR of about -1.7, and the dimensionality prediction shortly thereafter.
- In Fig. 34—*middle* we start with the initial prototype signal P with an SNR of -0.22, which is the published DJB data with the *medium-noise* setting (Sarle 1999). We progressively resample the data from 2048 datapoints (the original data) down to just 82 datapoints. Both the cardinality and dimensionality predictions fail as we move from 300 to 320 datapoints.
- In Fig. 34—*bottom* we again start with the initial prototype signal P with an SNR of -0.22, this time we gradually add a global linear trend from 0 to 0.45, as measured by the gradient of the data. Both the dimensionality and cardinality predictions fail as the gradient is increase pasted 0.3.

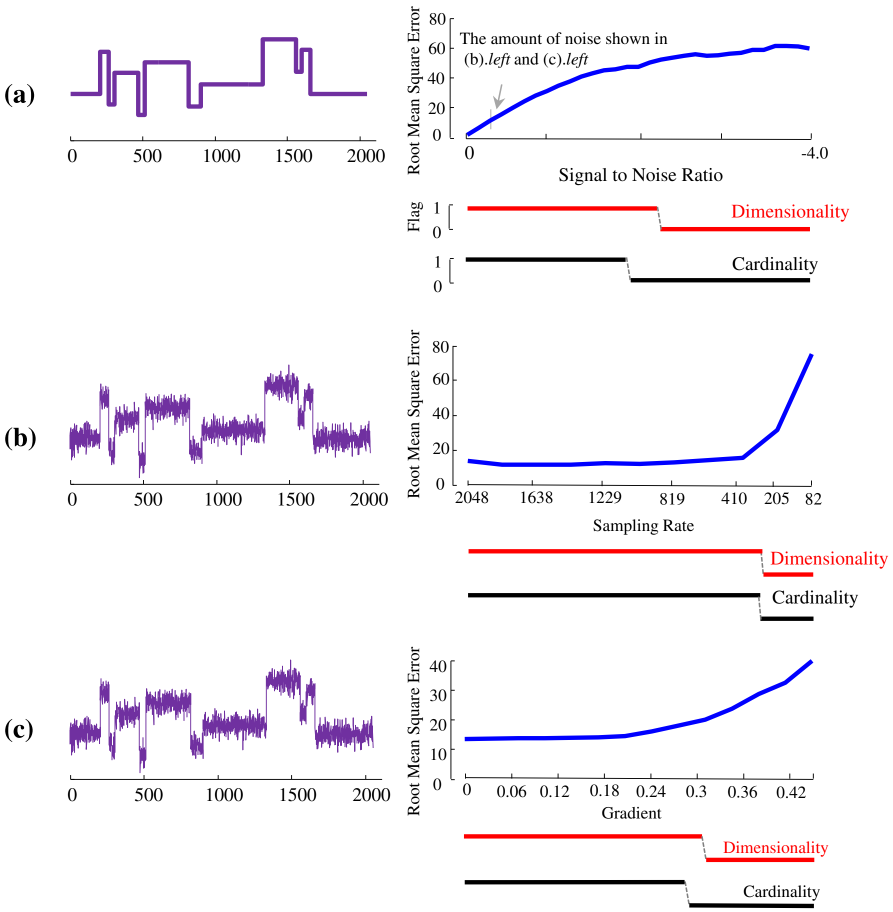
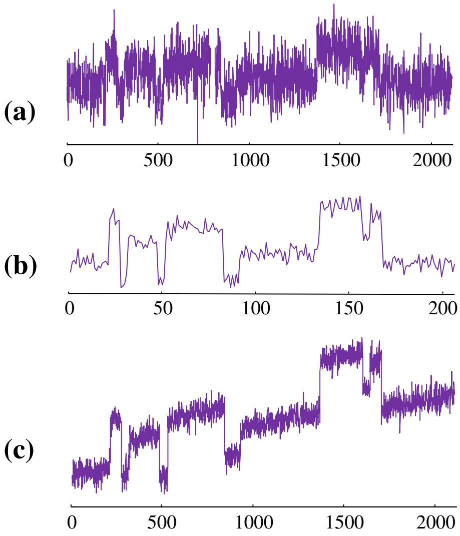


Fig. 34 The robustness of our algorithm to various distortions added to the DJB data. In **a**—*left* we show the DJB data with no noise, and in **a**—*right* we plot the RMSE between **a**—*left* and the corrupted versions. In **b**—*left* we start with the same data shown in Fig. 10 (the noise level in **b**—*left* is also marked in pointed out in (**a**)—*right*). In **b**—*right*, we show the RMSE between (**b**)—*left* and the downsampled versions of the data. In **c**—*left* we again start with the data used in Fig. 10, and in **c**—*right* we plot the RMSE between (**c**)—*left* and the linear trend added versions

The results in Fig. 34 suggest our algorithm is quiet robust to these various distortions. To give the reader a better appreciation of *when* our algorithm fails, in Fig. 35 we show the most corrupted version of the signals for which our algorithm correctly predicted *either* the cardinality or the dimensionality.

It is important to reiterate that the experiment that added a *linear* trend to the data but only considered a *constant* model was deliberately testing the mismatch between assumptions and reality. In particular if we repeat the technique shown in Fig. 14, of testing over *all* models spaces in {DFT, APCA, PLA}, our algorithm does correctly predict the data in Fig. 35c as consisting of piecewise linear segments, and still correctly predicts the cardinality *and* the dimensionality.

Fig. 35 The three most corrupted versions of the Donoho–Johnstone block for which our framework makes a correct prediction of *either* the cardinality or the dimensionality. **a** The nosiest example, **b** the example with the lowest sampling rate, **c** the example with the greatest linear trend added



5 Time and space complexity

The *space* complexity of our algorithm is linear in the size of the original data. The *time* complexity of the algorithms that use APCA and PLA as the representation in Tables 2 and 3 is O (m^2) and the time complexity of the algorithm using DFT as the approximation in Table 4 is O $(m\log m)$. Although we have two for-loops in the above three tables, the for-loops just add constant factors; they do not increase the degree of the polynomial to the time complexity. This is because outer for-loop is the range of cardinalities c from 2 to 256 and the inner for-loop is the range of the dimensionalities d from 2 to 64.

Our framework achieves the time complexity of O (m^2). Note that the data in Sect. 4.5 were obtained over a year and the datasets in Sect. 4.6 were obtained over more than 80 years. Thus compared to how long it takes to *collect* the data, our algorithm's *execution time* (a few seconds) is inconsequential for most applications.

Nevertheless, we can use the following two methods to speed up the search by pruning the search space of combinations of every$c$ and d that are very unlikely to be fruitful.

First, there are nested for-loops in Tables 2, 3 and 4. It appears that we have to calculate the MDL cost for every combination of each c and d, thus the results will form a 2D matrix. However, instead of finding the MDL cost from the every combination of each c and d, we can just calculate the MDL cost in a very small subset of the matrix, in particular, just one row and one column. This works as follows, for a given time series, we first calculate its intrinsic dimensionality given a fixed cardinality of 256. Secondly, with the intrinsic dimensionality in hand, we scan a range of cardinality from 2 to 256 to find out the intrinsic cardinality. We illustrate the intuition as to why searching only one row and one column of the matrix are generally sufficient for finding the intrinsic cardinality and dimensionality. Consider the Donoho–Johnstone benchmark

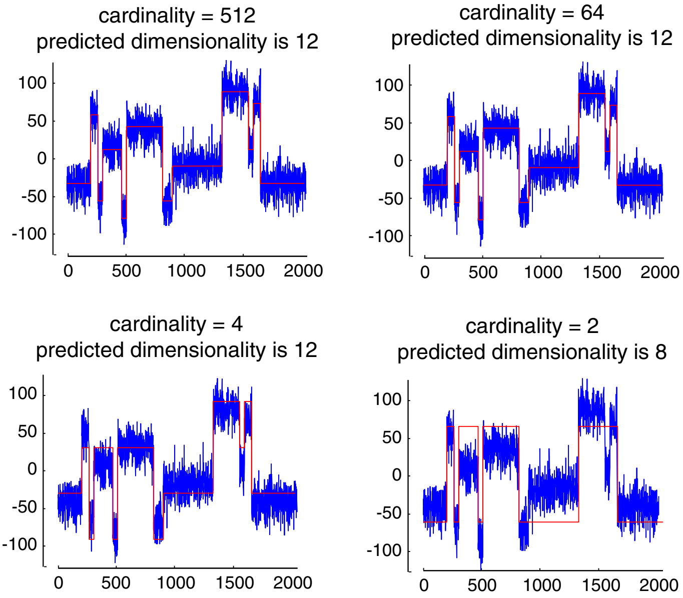


Fig. 36 A comparison of the effect from differing cardinalities on our framework's ability to discover the correct intrinsic dimensionality of DJB data. For any cardinality from 512 to 4, the discovered intrinsic dimensionality does not change. Only when the cardinality is set to a pathologically low three or two (*bottom right*) does the cardinality value affect the predicted dimensionality

(DJB) data as an example in Fig. 36, there is no need to search over the whole matrix, because changing the cardinality from 512 to 4 does not produce different predicted dimensionalities.

In order to calculate the intrinsic dimensionality for DJB, we first fix the cardinality at 256, then find the MDL cost with dimensionality from range 2 to 64 (the inner for-loop in Table 2). In this example, the time complexity for finding the intrinsic dimensionality is O (m^2). After we discover the intrinsic dimensionality is 12, we hardcode the dimensionality at 12, then calculate the MDL cost with cardinality ranging from 2 to 256. Thus, the time complexity for finding the intrinsic cardinality is O (m^2). Using with this method, there is no need to calculate the MDL cost for every combination of c and d.

Second, we further can optimize the algorithm by caching the results from the PLA/APCA/DFT approximations. We can do the DFT/PLA/APCA decomposition *once* at the finest granularity, and cache the results, leaving only a loop that performs efficient calculations on integers with Huffman coding. After this optimization, the time taken for our algorithms is O (m^2) without any constant factors for using PLA and APCA approximation. Figure 37 demonstrates the comparison of running time using APCA and our MDL framework. As Fig. 37 shows, after caching the results for PLA, the *overhead ratio* for calculating the MDL costs is relatively small and decreases for larger dataset. This is because the overhead is dominated by the Huffman coding, whose time complexity is only O $(m \log m)$.

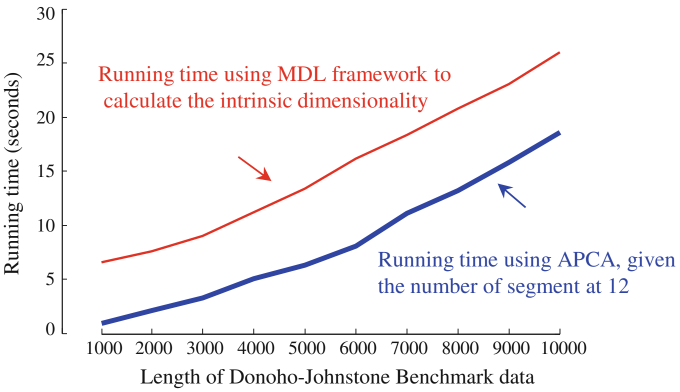


Fig. 37 The running time comparison between our MDL based approach (*red/fine*) and the APCA (*blue/bold*) approximation for DJB dataset. The x axis is the length of different instantiations of the DJB data (Color figure online)

6 Discussion and related work

We took the opportunity to show an initial draft of this manuscript to many respected researchers in this area, and this paper greatly benefits from their input. However, many researchers passionately argued often mutually exclusive points related to MDL that we felt were orthogonal to our work and irrelevant distractions from our claims. We will briefly address these points here.

The first issue is *who* should be credited with the invention of the basic idea we are exploiting, that the shortest overall two-part message is most likely the correct explanation for the data. Experts in complexity theory advocate passionately for Andrey Kolmogorov, Chris Wallace, Ray Solomonoff, Jorma Rissanen or Gregory Chaitin, etc. Obviously, our work is not weighing in on such a discussion, and we refer to Li (1997) as a good neutral starting point for historical context. We stand on the shoulders of *all* such giants.

One researcher felt that MDL models could only be evaluated in terms of the *prediction* of future events, not on post-hoc *explanations* of the models discovered (as we did in Fig. 15, for example). However, we *have* carried out prediction experiments. For example, in the introduction we used our MDL technique to predict which of approximately 700 combinations of settings of the cardinality/dimensionality/number of exemplars would produce the most accurate classifier under the given constraints. Clearly the 90.75 % accuracy we achieved significantly outperforms the default settings that gave only 58.70 %. However, a brute force search shows that our *predicted* model produced the *best result* (three similar settings of the parameters tied with the 90.75 % accuracy). Likewise, the experiment shown in Fig. 18 can be cast in a prediction framework: “predict which of these heartbeats a cardiologist is most likely to state is abnormal.” To summarize, we do not feel that the prediction/explanation dichotomy is of particular relevance here.

There are many works that use MDL in the context of real-valued time series. However, our parameter-free method *is* novel. For example, Firoiu and Cohen (2002) uses MDL to help guide a PLA segmentation of time series; however, the method also uses *both* hybrid neural networks *and* hidden Markov models, requiring at least six parameters to be set (and a significant amount of computational overhead). Similarly, Molkov et al. (2009) use MDL in the context of neural networks, inheriting the utility of MDL but also inheriting the difficulty of learning the topology and parameters of a neural network.

Likewise, the authors of Davis et al. 2008 use MDL to "find breaks" (i.e., segments) in a time series, but their formulation uses a genetic algorithm which requires a large computational overhead and the careful setting of seven parameters. Finally, there are now several research efforts that use MDL for time series (Vatauv 2012; Vespier et al. 2012) that were inspired by the original conference version of this work (Hu et al. 2011).

There are also examples of research efforts using MDL to help cluster or carry out motif discovery in time series; however, to the best of our knowledge, this is the first work to show a completely parameter-free method for the discovery of the cardinality/dimensionality/model of a time series.

7 Conclusions

We have shown that a simple, yet powerful methodology based on MDL can robustly identify the intrinsic model, cardinality and dimensionality of time series data in a wide variety of domains. Our method has significant advantages over existing methods in that it is more general and is essentially parameter-free. We have further shown applications of our ideas to resource-limited classification and anomaly detection. We have given away all of our (admittedly very simple) code and datasets so that others can confirm and build on our results from www.cs.ucr.edu/~bhu002/MDL/MDL.html.

A reader may assert that our claim to be parameter-free is unwarranted because: we "choose" to use a binary computer instead of say a ternary computer,[8] we use Huffman coding, not Shannon–Fano coding and we hard code the maximum cardinality of time series to 256. However, a pragmatic data miner will still see our work as being a way to explore time series data, free from the need to have to *adjust* parameters. In that sense our work is truly parameter-free.

In addition to the above, we need to acknowledge other shortcomings and limitations of our work. Our ideas, while built upon the solid theoretical foundation of MLD, are *heuristic*, we have not proved any properties of our algorithms. Moreover, our method is essentially a *scoring* function; as such it will inherit any limitations of the *search* function used (cf. Table 3). For example while there is an optimal algorithm for finding the cheapest (in the sense of lowest root-mean-squared error) PLA of a time series given any desired d, this algorithm is too slow for most practical purposes and thus we (and virtually all the rest of the community) must content ourselves with an approximate PLA construction algorithm (Keogh et al. 2011).

[8] Of course, no commercial ternary computers exist, however they are at least a logical possibility.

Acknowledgments This project was supported by the Department of the United States Air Force, Air Force Research Laboratory under Contract FA8750-10-C-0160, and by NSF grants IIS-1161997. We would like to thank the reviewers for their helpful comments, which have greatly improved this work.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

Assent I, Krieger R, Afschari F, Seidl T (2008) The TS-Tree: Efficient Time Series Search and Retrieval. In: EDBT. ACM, New York

Bronson JE, Fei J, Hofman JM, Gonzalez RL, Wiggins CH (2009) Learning rates and states from biophysical time series: a Bayesian approach to model selection and single-molecule FRET data. Biophys J 97:3196–3205

Camerra A, Palpanas T, Shieh J, Keogh E (2010) *i*SAX 2.0: indexing and mining one billion time series. In: International conference on data mining

Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. ACM Comput Surv 41:3

Davis RA, Lee TCM, Rodriguez-Yam G (2008) Break detection for a class of nonlinear time series models. J Time Ser Anal 29:834–867

De Rooij S, Vitányi P (2012) Approximating rate-distortion graphs of individual data: experiments in Lossy compression and denoising. IEEE Trans Comput 61(3):395–407

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. In: VLDB, Auckland, pp 1542–1552

Donoho DL, Johnstone IM (1994) Ideal spatial adaptation via wavelet shrinkage. J Biometrika 81:425–455

Evans SC et al (2007) Microrna target detection and analysis for genes related to breast cancer using MDL compress. EURASIP J Bioinform Syst Biol 1–16

Firoiu L, Cohen PR (2002) Segmenting time series with a hybrid neural networks hidden Markov model. In: Proceedings of 8th national conference on artificial Intelligence, p 247

García-López D, Acosta-Mesa H (2009) Discretization of time series dataset with a genetic search. In: MICAI. Springer, Berlin, pp 201–212

Goebel K, Saha B, Saxena A (2008) A comparsion of three data-driven techniques for prognostics. In: Failure prevention for system availability, 62th meeting of the MFPT Society, pp 119–131

Grünwald PD, Myung IJ, Pitt MA (2005) Advances in minimum description length: theory and applications. MIT, Cambridge

Heimes FO, BAE Systems (2008) Recurrent neural networks for remaining useful life estimation. In: International conference on prognostics and health management

Hu B, Rakthanmanon T, Hao Y, Evans S, Lonardi S, Keogh E (2011) Discovering the intrinsic cardinality and dimensionality of time series using MDL. In: ICDM

International Business Machiness (IBM) (2012) Harness the power of big data. www.public.dhe.ibm.com/common/ssi/ecm/en/imm14100usen/IMM14100USEN.PDF. Accessed 7 Nov 2012

Jonyer I, Holder LB, Cook DJ (2004) Attribute-value selection based on minimum description length. In: International conference on artificial intelligence

Kehagias Ath (2004) A hidden Markov model segmentation procedure for hydrological and enviromental time series. Stoch Environ Res Risk Assess 18:117–130

Keogh E, Chu S, Hart D, Pazzani M (2011) An online algorithm for segmenting time series. In: KDD

Keogh E, Kasetty S (2003) On the need for time series data mining benchmarks: a survey and empirical demonstration. J Data Min Knowl Discov 7(4):349–371

Keogh E, Pazzani MJ (2000) A simple dimensionality reduction technique for fast similarity search in large time series databases. In: PAKDD, pp 122–133

Keogh E, Zhu Q, Hu B, Hao Y, Xi X, Wei L, Ratanamahatana CA (2006) The UCR time series classification /clustering. www.cs.ucr.edu/~eamonn/time_series_data/

Kontkanen P, Myllym P (2007) “MDL histogram density estimation. In: Proceedings of the eleventh international workshop on artificial intelligence and statistics

Lemire D (2007) A better alternative to piecewise linear time series segmentation. In: SDM

Li M (1997) An introduction to Kolmogorov complexity and its applications, 2nd edn. Springer, Berlin

Lin J, Keogh E, Lonardi S, Patel P (2002) Finding motifs in time series. In: Proceedings of 2nd workshop on temporal data mining

Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing SAX: a novel symbolic representation of time series. J DMKD 15(2):107–144

Linacre E, Geerts B (2011) Resources in atmospheric science, 2002. http://www-das.uwyo.edu/~geerts/cwx/notes/chap15/global_temp.html. Accessed 1 Dec 2011

Malatesta K, Beck S, Menali G, Waagen E (2005) The AAVSO data validation project. J Am Assoc Variable Star Observ (JAAVSO) 78:31–44

Molkov YI, Mukhin DN, Loskutov EM, Feigin AM (2009) Using the minimum description length principle for global reconstruction of dynamic systems from noisy time series. Phys Rev E 80:046207

Mörchen F, Ultsch A (2005) Optimizing time series discretization for knowledge discovery. In: KDD

National Aeronautics and Space Administration (2011) GISS surface temperature analysis. http://data.giss.nasa.gov/gistemp/. Accessed 1 Dec 2011

Palpanas T, Vlachos M, Keogh E, Gunopulos D (2008) Streaming time series summarization using user-defined amnesic functions. IEEE Trans Knowl Data Eng 20(7):992–1006

Papadimitriou S, Gionis A, Tsaparas P, Väisänen A, Mannila H, Faloutsos C (2005) Parameter-free spatial data mining using MDL. In: ICDM

Pednault EPD (1989) Some experiments in applying inductive inference principles to surface reconstruction. In: IJCAI, pp 1603–1609

PHM Data Challenge Competition (2008). phmconf.orgjOCS/index.php/phm/2008/challenge

Picard G, Fily M, Gallee H (2007) Surface melting derived from microwave radiometers: a climatic indicator in Antarctica. Ann Glaciol 47:29–34

Protopapas P, Giammarco JM, Faccioli L, Struble MF, Dave R, Alcock C (2006) Finding outlier light-curves in catalogs of periodic variable stars. Monthly Not R Astron Soc 369:677–696

Prognostics Center of Excellence, National Aeronautics and Space Administration (NASA) (2012). ti.arc.nasa.gov/tech/dash/pcoe/prognostic-data-repository/. Accessed 7 Nov 2012

Project URL. www.cs.ucr.edu/~bhu002/MDL/MDL.html. This URL contains all data and code used in this paper, as well as many additional experiments omitted for brevity

Rakthanmanon T, Keogh E, Lonardi S, Evans S (2012) MDL-based time series clustering. Knowl Inf Syst 33(2):371–399

Rebbapragada U, Protopapas P, Brodley CE, Alcock CR (2009) Finding anomalous periodic time series. Mach Learn 74(3):281–313

Rissanen J (1989) Stochastic complexity in statistical inquiry. World Scientific, Singapore

Rissanen J, Speed T, Yu B (1992) Density estimation by stochastic complexity. IEEE Trans Inf Theory 38:315–323

Salvador S, Chan P (2004) Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In: International conference on tools with artificial intelligence, pp 576–584

Sarle W (1999) Donoho–Johnstone benchmarks: neural net results. ftp.sas.com/pub/neural/dojo/dojo.html

Sart D, Mueen A, Najjar W, Niennattrakul V, Keogh E (2010) Accelerating dynamic time warping subsequence search with GPUs and FPGAs. In: IEEE international conference on data mining, pp 1001–1006

Signal to Noise Ratio. http://en.wikipedia.org/wiki/Signal-to-noise_ratio

US Environmental Protection Agency (2011) Climate Change Science. www.epa.gov/climatechange/science/recenttc.html. Accessed 6 Dec 2011

Vachtsevanos G, Lewis FL, Roemer M, Hess A, Wu B (2006) Intelligent fault diagnosis and prognosis for engineering systems, 1st edn. Wiley, Hoboken

Vahdatpour A, Sarrafzadeh M (2010) Unsupervised discovery of abnormal activity occurrences in multi-dimensional time series, with applications in wearable systems. In: SIAM international conference on data mining

Vatauv R (2012) The impact of motion dimensionality and bit cardinality on the design of 3D gesture recognizers. Int J Hum–Comput Stud 71(4):387–409

vbFRET Toolbox (2012) www.vbFRET.sourceforge.net. Accessed 8 Nov 2012

Vereshchagin N, Vitanyi P (2010) Rate distortion and denoising of individual data using Kolmogorov complexity. IEEE Trans Inf Theory 56(7):3438–3454

Vespier U, Knobbe A, Nijssen S, Vanschoren J (2012) MDL-based analysis of time series at multiple time-scales. Lecture notes in computer science (LNCS), vol 7524. Springer, Berlin

Wallace CS, Boulton DM (1968) An information measure for classification. Comput J 11(2):185–194

Wang T, Lee J (2006) On performance evaluation of prognostics algorithms. In: Proceedings of MFPT, pp 219–226

Wang T, Yu J, Siegel D, Lee J (2008) A similarity-based prognostics approach for remaining useful life estimation of engineered systems. In: International conference on prognostics and health management

Witten H, Moffat A, Bell TC (1999) Managing gigabytes compressing and indexing documents and images. Morgan Kaufmann, San Francisco

Yankov D, Keogh E, Rebbapragada U (2008) Disk aware discord discovery: finding unusual time series in terabyte sized datasets. Knowl Inf Syst 17(2):241–262

Zhao Q, Hautamaki V, Franti P (2008) Knee point detection in BIC for detecting the number of clusters. In: ACIVS, vol 5259, pp 664–673

Zwally HJ, Gloersen P (1977) Passive microwave images of the polar regions and research applications. Polar Rec 18:431–450