UNIVERSITY OF CALIFORNIA
RIVERSIDE

Mining Time Series Data: Moving from Toy Problems to Realistic Deployments

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Bing Hu

December 2013

Dissertation Committee:

Dr. Eamonn Keogh, Chairperson
Dr. Stefano Lonardi
Dr. Gianfranco Ciardo

The Dissertation of Bing Hu is approved:

_____

_____

_____

Committee Chairperson

University of California, Riverside

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincerest gratitude to my advisor Dr. Eamonn Keogh for the invaluable guidance, supervision, and generous support during my doctoral study. I deeply appreciate that in summer 2010, he has me as one of his Ph.D students, which has totally changed my life. During my three and a half year's Ph.D study, I am blessed with Dr. Keogh's priceless insightful advice, brilliant research philosophy and his valuable time. It is very fortunate that I have him as my advisor. Thanks a lot!

I humbly thank Dr. Stefano Lonardi and Dr. Gianfranco Ciardo who are my committee members, for their generous support and valuable comments. Dr. Michalis Faloutsos and Dr. Ertem Tuncel for being committee members in my oral-qualification exam. I also want to thank Ms. Amy Ricks for all the helpful advice whenever I approached her.

I express gratitude to my colleagues in the data mining lab at UCR (names in random order), who offered me valuable help and friendship: Gustavo Batista, Thanawin (Art) Rakthanmanon, Abdullah Mueen, Bilson Campana, Qiang Zhu, Xiaoyue Wang, Yanping Chen, Mohammad Shokoohi-Yekta, Jesin Zakaria, Nurjahan Begum and Liudmila Ulanova. Also I would like to thank Denisa Duma for fruitful discussion for the Multi-dimensional classification paper.

Finally, I would like to gratefully thank my family for always being there with me, providing constant inspirations. Without my parents' support, I would not be me like this. I also would like to thanks all my friends at UCR for being part of this wonderful journal.

ABSTRACT OF THE DISSERTATION

Mining Time Series Data: Moving from Toy Problems to Realistic Deployments

by

Bing Hu

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2013
Dr. Eamonn Keogh, Chairperson

Data mining and knowledge discovery has attracted a lot of research interest in the last decade. Although there is extensive research in this area, we argue that most of the work is not as useful, since the datasets that they are dealing with and the methods that they proposed to solve the problems are more like 'toy examples' compared to the much more complicate real-world scenario. We have observed the following two problems that widely exist in most of data mining research. First, parameters will hurt the potential of spreading the ideas in the research community. In a lot of works, there are usually several parameters to tune in the proposed method. We claim that the parameter turning can kill the usefulness of an algorithm and reduce the number of citations. Second, the prevalently existed assumptions about the data further limit their application to solve the real-world problem. We strive to mitigate the above two problems. The contribution of this dissertation is as follows:

First, we demonstrate a parameter free framework using MDL to discover the intrinsic features of the data. With the intrinsic cardinality and dimensionality of the time series, we can further understand the underlying meaning of the data, before consulting the domain experts. In addition, the intrinsic features can be used as dimensionality reduction and have huge applications in the various lower bounding techniques. Second, we show a time series classification framework that has none of the prevalent assumptions. We propose to use the data editing technique to automatically build a data dictionary. In addition, our classification framework has the capability to say 'I do not know' at a certain point when classifying the incoming queries that does not belong to any concept in the training data. Our results show that a small fraction of all the data can achieve even better classification results than using all the data. In the last, we propose a dynamically weighted multi-dimensional classification framework, which can smartly choose the weight of each data dimension. The results over extensive datasets from various domains show that our framework is more accurate and robust to the occluded data.

List of Figures

Figure 1: *left*) A snippet from a two-lead polysomnogram. *right*) At certain times, V5R becomes noisy while V5 remains almost unaffected. At other times (not shown), we see these roles reversed. 8

Figure 2: Two snippets of gyroscope data (110Hz) from a physical activity dataset [95]. Activities denoted `rope-jumping` (red/left) and `ascending-stairs` (purple/right) are more obvious from the wrist and shoe sensors, respectively..... 9

Figure 3: Three unrelated industrial time series with low intrinsic cardinality. I) Evaporator (channel one). II) Winding (channel five). III) Dryer (channel one)..... 12

Figure 4: 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..... 17

Figure 5: A sample time series *T* that will be used as a running example in this section 21

Figure 6: Time series *T* (blue/fine), approximated by a one-dimensional APCA approximation *H1* (red/bold). The error for this model is represented by the vertical lines 22

Figure 7: Time series *T* (blue/fine), approximated by a two-dimensional APCA approximation, *H2* (red/bold). Vertical lines represent the error 22

Figure 8 : left) The figure shown in Figure 6 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 24

Figure 9 : The log2 of the range of the residual errors for all possible single constant polynomial models of the data introduced in Figure 5. Note

that the model that minimizes this value (with a tie) is also the model that minimizes the residual error..... 25

Figure 10: 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 26

Figure 11 : A toy example of a time series that has more than one state 34

Figure 12 : A version of the Donoho-Johnstone block benchmark created ten years ago and downloaded from [46] 39

Figure 13 : The knee-finding L-Method. *top*) A residual error vs. 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"..... 41

Figure 14 : The description length of the Donoho-Johnstone block benchmark time series is minimized at a dimensionality corresponding to twelve piecewise constant segments, which is the *correct* answer [46]..... 41

Figure 15 : The description length of the Donoho-Johnstone block benchmark time series is minimized with a cardinality of ten, which is the true cardinality [46] 42

Figure 16: The description length of the Donoho-Johnstone block benchmark time series is minimized with a piecewise constant model (APCA), not a piecewise linear model (PLA) or Fourier representation (DFT)..... 43

Figure 17 : *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..... 43

Figure 18: *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 [32], predicts a value of four..... 44

Figure 19: *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 [40] 46

Figure 20: *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*) 46

Figure 21: *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 48

Figure 22: A time series showing the annual discharge rate of Senegal River from the year 1903 to 1988 49

Figure 23: *top*) The blue/light line is Senegal River data. The black/bold line is the segmentation result found in Section 5.1 of [19]. *bottom*) We obtained the red/bold line by hard coding the number of segments to five using the MDL algorithm 49

Figure 24: 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 50

Figure 25: A time series showing the annual global temperature change from the year 1700 to 1981 50

Figure 26: *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 [19]. *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 51

Figure 27: Our MDL algorithm obtains *two* as the intrinsic dimensionality of the time series for the global annual mean temperature 51

Figure 28: A representative smFRET trace from [2][54] 52

Figure 29: *top*) The time series in blue from Figure 28 is predicted to have three states [2][54]. 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 53

Figure 30: *top*) An example of an *operational* variable in the PHM08 dataset. *bottom*) An example of a *non-operational* variable in the PHM08 dataset 54

Figure 31: 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 55

Figure 32: The description length of the synthetic time series shown in Figure 11 minimizes when the dimensionality is eight, which is the intrinsic dimensionality 56

Figure 33: *top*) A toy time series shown in Figure 11 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..... 56

Figure 34: *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 57

Figure 35: The data shown in Figure 34 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..... 58

Figure 36: 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 Figure 12. (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 Figure 12, and in (c).*right* we plot the RMSE between (c).*left* and the linear trend added versions 61

Figure 37: 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……62

Figure 38: 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 64

Figure 39: The running time comparison between our MDL based approach (red/fine) and the APCA (blue/bold) approximation for Donoho-Johnstone benchmark dataset. The x axis is the length of different instantiations of the DJB data 65

Figure 40: A snippet of BIDMC Congestive Heart Failure Database ECG - Record-08 [68]. (a) is weakly-labeled data, which exhibits both *extraneous* data, a section of recording when the machine was not plugged in, and *redundant* data (only one pair of redundancies are shown in bold (red/green). (b) A minimally redundant set of representative heartbeats (a *data dictionary*) could be used as training data 72

Figure 41: A snippet of BIDMC Congestive Heart Failure Database ECG: Record-03 [68]. Note that this section of ECG data exhibits more variability than the data in Figure 40. 74

Figure 42: *left*) A toy example data dictionary which was condensed from a large dataset. These seven subsequences in data dictionary A span the concept space of the bulls/bears problem. *right*) Note that if we had a distance measure that was invariant to linear scaling, we could further reduce data dictionary A to data dictionary B 76

Figure 43: *left*) A data dictionary learned from a 15-class ECG classification problem (just class 01 is shown here). At first glance, the two exemplars seem redundant apart from their (irrelevant) phases. *right*) By using the *Euclidean* distance between the two patterns we can see that the misalignment of the beats would cause a large error. The problem solved by using the *Uniform Scaling* distance [87]..... 77

Figure 44: *top*) A snippet of BIDMC Congestive Heart Failure Database ECG data: Record-08 [68]. *bottom*) the distance vector of an incoming

query. The nearest neighbor and its distance of q is colored in red/bold 81

Figure 45: top) A snippet of BIDMC Congestive Heart Failure Database ECG data: Record-08 [68]. bottom) the extracted subsequence has twice the query length 88

Figure 46: The *green*/left histogram contains the nearest neighbor distances of correctly classified queries for the ECG data used in Section 3.3.2. The *red*/right histogram shows nearest neighbor distances for queries from the other class 90

Figure 47: Anytime algorithms are interruptible after initialization. This plot shows the result quality increases with computation time 90

Figure 48: The classification error rates for D from $D_{0.39\%}$ to $D_{14.2\%}$ for the physical activity dataset [95] 97

Figure 49: The *pink*/*green(bold)* curves are train/test error rates obtained when we replaced *Euclidean* distance with *Uniform Scaling* distance 98

Figure 50: Two examples of rejected queries. Both queries contain significant amount of noise 98

Figure 51: The classification error rates for D from $D_{0.28\%}$ to $D_{5.82\%}$ for BIDMC Congestive Heart Failure Database [68] 100

Figure 52: The *pink*/*green(bold)* curves are train/test error rates obtained when we replaced *Euclidean* distance with *Uniform Scaling* distance 100

Figure 53: The classification error rates for D from $D_{0.17\%}$ to $D_{5.32\%}$ for [67] 101

Figure 54: The *blue*/*brown(bold)* curves are train/test error rates obtained when we replaced *Euclidean* distance with *Uniform Scaling* distance. Note the other curves are taken from Figure 53 for comparison purposes 102

Figure 55: Classification accuracy of complexity as an index in the anytime classifier on constant query streams with different arrival rates for datasets in Section 3.3.1 to 3.3.3. 103

Figure 56: The red dot/blue triangle represent sensors mounted in wrist/shoe, respectively. *left*) A two dimensional time series (T_1, T_2), T_1 from a

sensor on the wrist and T_2 from a sensor on the shoe. *right*) A query q with two dimensions (q_1 and q_2), will find their nearest neighbors in T_1 and T_2 , respectively. 109

Figure 57: The performance of four classifiers (a), (b), (c), and (d) on four activities. In each classifier, the height of the bar is the confidence score for each activity. 111

Figure 58: The distributions of nearest neighbor distances for true positives (green/left) and false positives (red/right) in the classification of activity running using data from wrist (top) and activity rope-jumping using data from shoe (bottom).. 114

Figure 59: A snippet of BIDMC Congestive Heart Failure Database ECG, Record-03. (a) WT, which exhibits both extraneous and redundant data. There are two types of anomalous heartbeats (V, S) and normal beat (N) in WT. (b) A minimally redundant set of representative heartbeats (a data dictionary) could be used as training data. 118

Figure 60. left) A snippet of sound spectrum and MFCCs from 2 to 5 for the East Brazilian Pygmy Owl. right) A snippet of sound spectrum and MFCCs from 2 to 5 for the Common Potoo. 131

Figure 61: *x*, *y*, *z* acceleration data from right hand (*brown*) and left hand (*blue*) for two signals `Six` and `Leg Bye`.. 133

Figure 62. Visualization of the six gesture classes. This figure from [136] is used with permission. 135

Figure 63. a) Modified Wii Remotes embedded in specially designed utensils. b) A subject is preparing salad. This figure is used with permission from [148]. 136

List of Tables

Table 1 : Generic MDL algorithm for time series 28

Table 2 : Our algorithm specific to APCA 29

Table 3 **:** Our algorithm specific to PLA 31

Table 4 : Our algorithm specific to DFT 33

Table 5 : Our algorithm specific to the mixed polynomial degree model 36

Table 6 : Bottom-up mixed polynomial degree model algorithm 38

Table 7: Classification Algorithm using Data Dictionary 79

Table 8: Nearest Neighbor Search within a Time Series 80

Table 9: Classification of Training Data 86

Table 10: Anytime Nearest Neighbor Classification Algorithm 91

Table 11: using Complexity as an Index 93

Table 12: Adjusted Confidence Classification Algorithm 123

Table 13: Learning the Confidence Score 125

Table 14: Classification Results on the Physical Activity Data for ACV and Seven Straw Men 129

Table 15: Classification Results on the Cricket Data 134

Table 16: Classification Results on the Gesture Data 135

Table 17: Classification Results on the Kitchen Data 136

Table of Contents

List of Figures……vi

List of Tables……xii

Chapter 1: Introduction 1

1.1 Discover the Intrinsic Features…. 1

1.2 Doing More Realistic Research…….……2

1.3 Multi-Dimensional Time Series Classification…….6

Chapter 2: Discovering the Intrinsic Cardinality and Dimensionality of Time Series using MDL..... 11

2.1 Introduction..... 11

2.1.1. A Concrete Example 14

2.2 Definitions and Notation..... 15

2.3 MDL Modeling of Time Series..... 21

2.3.1. An Intuitive Example of Our Basic Idea..... 21

2.3.2.Generic MDL for Time Series Algorithms 27

2.3.3.Adaptive Piecewise Constant Approximation…….....……...……29

2.3.4 Piecewise Linear Approximation…...……..……..…..……30

2.3.5 Discrete Fourier Transform……..…….……..……..31

2.3.6 A Mixed Polynomial Degree Model……..…….....…33

2.4 Experimental Results 39

2.4.1 A Detailed Example on a Famous Problem……..……..……..39

2.4.2 An Example Application in Physiology……...……43

2.4.3 An Example Application in Astronomy……..44

2.4.4 An Example Application in Cardiology……46

2.4.5 An Example Application in Geosciences…….47

2.4.6An Example Application in Hydrology and Environmental Science……..…….49

2.4.7 An Example Application in Biophysics……...…….52

2.4.8 An Example Application in Prognostics……...……53

2.4.9 Testing the Mixed Polynomial Degree Model…….....…….56

2.4.10 An Example Application in Aeronautics……..……..57

2.4.11 Quantifiable Experiments…….……..58

2.5Time and Space Complexity……...……..62

2.6Discussion and Related Work……...……65

2.7Conclusions……...……67

Chapter 3: Time Series Classification under More Realistic Assumptions……...69

3.1 Definitions and Notation……..70

3.1.1 A Discussion of Data Dictionaries…….73

3.1.2 An Additional Insight on Data Redundancy…….……75

3.1.3 On the Need for a Threshold……..……78

3.2 Algorithms……...……..…..78

3.2.1 Classification Using A Data Dictionary……..……..78

3.2.2 Building the Data Dictionary…….…….…...81

3.2.3 Learning the Threshold Distance…….……..89

3.2.4 Anytime Classification using Complexity As An Index……90

3.2.5 Uniform Scaling Technique……93

3.3 Experimental Evaluation……..93

3.3.1 An Example Application in Physiology……..……..95

3.3.2 An Example Application in Cardiology……..……..99

3.3.3 An Example Application in Daily Acitivies……101

3.3.4 Speed Up The Search Using Complexity As Index…….....102

3.4 Conclusion and Future Work……...……106

Chapter 4: Classification of Multi-Dimensional Streaming Time Series by Weighting each Classifier's Track Record…….....……107

4.1 Notation and Background……..……107

4.1.1 Basic Time Series Definitions……..……107

4.1.2 Supporting Confidence-Based Classification…….…….…..109

4.1.3 Supporting Distance-Based Classification……..…..112

4.1.4 Allowing Real World Deployment……...……116

4.2 Related Work……119

4.2.1 Relationship to Ensemble Methods……..……120

4.2.2 The Adjusted Confidence vs. the Weight in Weighted Voting……..121

4.3 Algorithms……..……..122

4.3.1 Classification of Multi-Dimensional Time Series using the Adjusted Confidence Scores……..122

4.3.2 Learning the Confidence Score……124

4.3.3 Learning the Adjusted Confidence Score…… 125

4.4 Experiments……...……126

4.4.1 Physical Activity Data……128

4.4.2 Avian Audio Data……..……130

4.4.3 Recognition of Cricket Umpire Signals……..132

4.4.4 Gesture Recognition……..……..134

4.4.5 Kitchen Activity Data……135

4.4.6 Robustness to Irrelevant Features…….……..136

4.5 Conclusion…….……137

Chapter 5: Conclusion…….…….137

Bibliography……140

Chapter 1: Introduction

Time series data are being generated at an unprecedented scale and rate from almost every application domain, e.g. medical and biological experimental observations, streaming data generated from the various sensors, daily prices in the stock market, etc. In the last decade, there is dramatically increasing research in query and mining time series data. However, we have observed the following two problems that widely exist in most of data mining research. First, parameters will hurt the potential of spreading the ideas in the research community. In a lot of works, there are usually several parameters to tune in the proposed method. We claim that the parameter turning can kill the usefulness of an algorithm and reduce the number of citations. Second, the prevalently existed assumptions about the data further limit their application to solve the real-world problem.

In this dissertation, we strive to mitigate the above two problems from the following three aspects. First, in Chapter 2 we demonstrate a parameter free framework to discover the intrinsic features of time series. Second, we illustrate how to do time series classification under more realistic assumptions in Chapter 3. In the last, we extend the framework in Chapter 3 to a multi-dimensional classification framework.

In the following text, we show the detail of the motivations of each project.

1.1 Discover the Intrinsic Features

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 Chapter 2, 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 sub-routine in algorithms for classification, clustering and outlier discovery. We will frame the discovery of these intrinsic features in the Minimal Description Length (MDL) 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.

1.2 Doing More Realistic Research

In virtually all time series classification research, long time series are processed into short equal-length "template" sequences that are representative of the class. For example, individual and complete gait cycles for biometric classification[62][72][79][88], individual and complete heartbeats for cardiological classification [70][82], individual and complete gestures for gesture recognition [111], etc. In most cases, the segmentation of long time series into these idealized snippets is done by hand [72][79][88][90]. However, for many real-world problems this either cannot be done, or only done with great effort [76][91][96].

As a concrete example, consider the famous Gun/Point problem [84][103], which has appeared in at least one hundred works [71][86][93]. To create this dataset, the original authors [102][103] used a metronome that signaled every three seconds to cue both the actor's behavior and the start/stop of the recording apparatus [102]. This allowed the extraction of perfectly aligned data, containing *all* of the target behavior and *only* the target behavior. Unsurprisingly, dozens of papers report less than 10% classification error rate on this problem. However, does such an error rate reflect our abilities with real-world data?

Such contriving of time series datasets seems to be the norm. For example, [111] notes, "*one subject performed one trial of an action* (in exactly) *10 seconds*." and [95] tells us that human editors should carefully discard "*all transient activities between performing different activities.*" Likewise, a recent paper states: *"We assume that the trajectories are segmented in time such that the first and last frames are already aligned* (and) *the resulting model has the same length*" [108]. Note that these authors are to be commended for stating their assumptions so concretely. In many cases, no such statements are made, but we suspect that similar "massaging" of the data has occurred.

We believe that such contriving of the data has led to unwarranted optimism about how well we can classify real-time series data streams. For real-world problems, we cannot always expect the *training* data to be so idealized, and we certainly cannot expect the *testing* data to be so perfect.

A more realistic idea for data gathering is to capture data "in the wild" as in [67][98][104], etc. However, this opens the problem of data editing and cleaning. For

example, a one-hour trace of data labeled "*walking*" will almost certainly contain non-representative subsequences, such as the subject pausing at a crosswalk, or introducing a temporary asymmetry into her gait as she answers her phone. The current solution to preprocess such data requires human intervention to examine and edit such traces, and keeping data that demonstrate the sought-after variability (walking `uphill`, `downhill`, `level`, walking `fast`, `normal`, `slow`), while discarding data that is atypical of the class.

Moreover, in virtually all time series classification research, the data must be arranged to have equal length [108]. For example, in the world's largest collection of time series datasets, the UCR classification archive, all forty-five time series datasets contain *only* equal-length data [84].

Most of the literature assumes that all objects to be classified belong to exactly one of two or more well-defined classes. For example, in the Gun/Point problem, every one of the instances is *either* a gun-aiming *or* a finger-pointing (unarmed) behavior. However, the vast majority of normal human actions are clearly neither. How well do current techniques work when most of the data is *not* from the well-defined classes?

The fourth and final unrealistic assumption is that queries to be classified are presented at equal time intervals. For example, if we know a system will produce queries ten times a second, we can then plan the hardware resources needed, and the maximum size of the training set. However, in many real world systems the available time for classification is not known a priori and may change as a consequence of external circumstances [105]. For example, for some ECG classification systems, the individual

beats are detected, and then passed to the classification system. Given that human heart rates vary from about 40 to 200 beats per minute, the query arrival rate can range between 0.6Hz to 3.3Hz[1]. The classification of flying insects can be fruitfully considered a time series problem and there the arrival rates can vary by at least four orders of magnitude [65][80]. If we plan only for the *fastest* possible arrival rate, then we may be forced to invest in computational resources that are unused 99.99% of the time, or to only consider a tiny training dataset, when 99.99% of the time we could have searched a larger dataset.

To summarize, much of the progress in time series classification from streams in the last decade is almost certainly optimistic, given that most of the literature implicitly or explicitly assumes one or more of the following:

- Copious amounts of perfectly aligned atomic patterns can be obtained [79][109][111].
- The patterns are all of equal length [79][84][88][96][104].
- Every item that we attempt to classify belongs to exactly one of our well-defined classes [76][84][96][103].

The queries arrive at a constant rate that is known ahead of time.

In Chapter 3, we demonstrate a time series classification framework that does not make *any* of these assumptions.

[1] Note that only *some* ECG classification systems do beat extraction then classification [74]. Many researchers believe that robust beat extraction can be a harder problem than classification itself (cf. Figure 40 and Figure 41), and thus present *every* subsequence extracted by a sliding window for classification. This is the approach we consider in Section 3.3

Our approach requires only very weakly-labeled data, such as "This ten-minute trace of ECG data consists mostly of arrhythmias, and that three-minute trace seems mostly free of them", removing assumption (1). Using this data we automatically build a "data dictionary", which contains only the minimal subset of the original data to span the concept space. This is because the data dictionary can contain, say, one example of walking fast, one example of walking normal, etc. This mitigates assumption (2).

As a byproduct of building this data dictionary, we learn a rejection threshold, which allows us to address assumption (3). A query item further than this threshold to its nearest neighbor is assumed to be in the other class. We show that using the Uniform Scaling distance measure [87] instead of Euclidean distance also addresses assumption (2). Finally, we introduce a novel technique to search the data dictionary in an *anytime* manner [105], allowing us to handle dynamic arrival rates and addressing assumption (4).

1.3 Multi-Dimensional Time Series Classification

Extensive research on time series classification in the last decade has produced fast and accurate algorithms for the single-dimensional case. However, the increasing prevalence of inexpensive sensors has reinforced the need for algorithms to handle multi-dimensional time series. For example, modern smartphones have at least a dozen sensors capable of producing streaming time series, and hospital-based (and increasingly, home-based) medical devices can produce time series streams from more than twenty sensors. The two most common ways to generalize from single to multi-dimensional data are to use all the streams or just the single best stream as determined at training time. However,

as we show here, both approaches can be very brittle. Moreover, neither approach exploits the observation that different sensors may be considered "experts" on different classes. In this work, we introduce a novel framework for multi-dimensional time series classification that weights the class prediction from each time series stream. These weights are based not only on each stream's previous track record on the class it is currently predicting, but also on the distance from the unlabeled object. As we demonstrate with extensive experiments on real data, our method is more accurate than current approaches and particularly robust in the face of concept drift or sensor noise.

Many physiological, medical, and scientific processes produce copious amounts of Multi-Dimensional Time series (*MDT*) data [133][140][151][156]. If we need to classify patterns manifest on just a *single* (independent) stream from an *MDT*, there is strong evidence that the simple nearest neighbor algorithm should be the algorithm of choice [123][129][132]. However, in many cases, the *m* individual time series in the *MDT* may reflect different views of the same underlying phenomena we want to classify. For example, we may have two different leads recording an ECG (Figure 1) or several gyroscopes on a Body Area Network (*BAN*) (Figure 2). In such a case, how should we use information from multiple sensors? The obvious choices are:

ALL: Use all *m* time series [151]. In this category, we include efforts that transform all *m* time series into a new space, using SVD [155] or Markov models [157], etc.

BEST: Use only the *single* best time series, which is either found empirically or suggested by domain knowledge [130]. In many research efforts the latter is probably done as a matter of course and reported *fait accompli* without discussion.

SUB: Use the *best* subset of the time series that is either found empirically or suggested by domain knowledge [128][133][146][149][155].

Note that while **SUB** includes **ALL** and **BEST** as special cases, the latter two choices are usually made without an effort to evaluate other possible subsets.

There are two reasons why we believe that none of the above is the ideal solution for the task at hand.

First, consider the two-lead ECG snippets shown in Figure 1. below. Here, we want to classify *myocardial ischemias* in this patient to correlate them with (independently recorded) sleep states. While the example shown in Figure 1.*left* could be classified from *either* the V_5 or V_{5R} lead, other examples are much more subtle and benefit from using *both* leads. However, suppose we use **ALL**, pooling evidence from both leads, then later on if *either* of them becomes noisy or disconnected (a very common occurrence [120][130]), we will do very poorly.

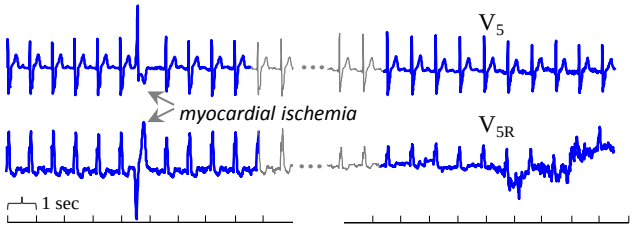


Figure 1: *left*) A snippet from a two-lead polysomnogram. *right*) At certain times, V_{5R} becomes noisy while V_5 remains almost unaffected. At other times (not shown), we see these roles reversed.

The second reason why most of the current approaches are sub-optimal is even more intuitive. The best subset of time series to use is almost always *class*-dependent. To see this, consider the *BAN* data shown in Figure 2. As we might expect, `rope-jumping` activities can be more easily classified using data from a sensor on the wrist than using

data from a sensor on the shoe. Conversely, to classify `ascending-stairs` behavior, using data from a sensor on the shoe is more accurate than using data from a sensor on the wrist. This can be easily explained if we imagine how the body moves during these behaviors.

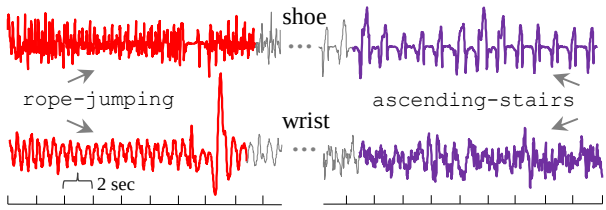


Figure 2: Two snippets of gyroscope data (110Hz) from a physical activity dataset [95]. Activities denoted `rope-jumping` (red/left) and `ascending-stairs` (purple/right) are more obvious from the wrist and shoe sensors, respectively.

In this work, we introduce a novel framework to address these two observations. At classification time, each sensor is polled for its vote on the class label. However, the vote is weighted by the sensor's self-reported confidence in its prediction. This self-reported confidence is based on two factors:

Confidence-based classification: the sensor's expertise on the class in question. This element is independent of the object to be classified. The expertise simply reflects that a sensor should not be confident in predicting one class if it was mostly wrong when it predicted this class during the training phase.

Distance-based classification: the similarity of the object to be classified and the examples seen during the training phase should be considered. This element reflects the fact that a sensor should not be confident in predicting *any* class if the object to be classified is significantly different than exemplars encountered during training.

As we shall demonstrate, by taking into account these two factors, we can make *MDT* classification both more accurate and more robust.

Chapter 2:

Discovering the Intrinsic Cardinality and Dimensionality of Time Series using Minimum Description Length

In this chapter, we will demonstrate a framework to discover the intrinsic features that has implications beyond selecting the best parameters for particular algorithms. We break the chapter into seven sections. Section 2.1 illustrates the motivation of the work. We give the definitions and intuitions of the algorithm in Section 2.2. Section 2.3 demonstrates the detail of the proposed algorithms. We demonstrate an extensive experimental evaluation of the proposed algorithm in Section 2.4. Section 2.5 clarifies the time and space complexity of our framework. Section 2.6 discusses the related work. In the last, we offer the conclusion of our work in Section 2.7.

2.1 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 [7] . 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 [13][24][36][43]. MDL is the cornerstone of many bioinformatics algorithms [9][42], and has had some impact in data mining, however it is arguably underutilized in *time series* data mining [18][35].

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 [40][58]. We illustrate this idea in the following example in Figure 3, which consists of three unrelated datasets.

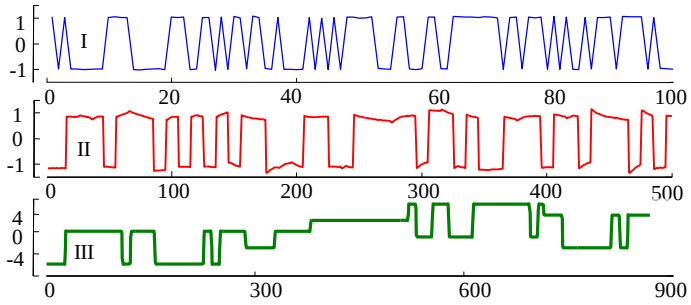


Figure 3: Three unrelated industrial time series with low intrinsic cardinality. I) Evaporator (channel one). II) Winding (channel five). III) Dryer (channel one)

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 instead 2, 2, and 12, respectively. As it happens, an understanding of the processes that produced

this data would perhaps support this claim [23]. 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[2]: 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 [47]. 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 [47].

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 [7] or the cardinality [28] of the time series (or both [1][3]) and search

[2] However, Section 2.1.1 shows an example where this *is* useful.

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.

2.1.1. 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"[50]. Let us suppose we have only 20kB available for the classifier, and that (as is the case with the benchmark dataset, TwoPat [23]) 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 [7]. 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 [28]), 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 [29], outlier discovery [4] [52][58], 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.

2.2 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[3], 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.

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 twenty diverse datasets in the UCR archive [23] 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 4 shows the results.

[3] The closely related technique of MML (Minimum Message Length [55]) *does* allow for continuous real-valued data. However, here we stick with the more familiar MDL formulation.

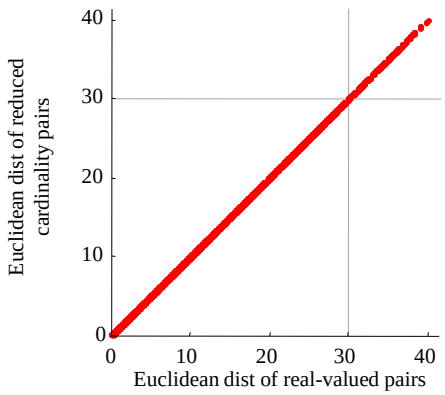


Figure 4: Each point on this plot corresponds to a pair of time series: the x-axis corresponds to their Euclidean distance, while the y-axis corresponds to the Euclidean distance between the 8-bit quantized representations of the same pair

The 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 [23], replacing the original data with the 256-cardinality version. For all cases the difference in classification accuracy was less than one tenth of one percent (full details are at [61]). 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 $[-2^{b-1}, 2^{b-1}-1]$. 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[4].

[4] 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: $Discretization(T) = round\left(\frac{T-min}{max-min}\right)*\left(2^8-1\right)+1$.

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.

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:

$$DL\,(T) = |\,HuffmanCoding\,(T)\,|$$

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 [26]. However, the Kolmogorov complexity can be approximated by using general-purpose data compression methods, like Huffman coding [13][52][60]. The (lossless) compressed file size is an upper bound to the Kolmogorov complexity of the time series [6].

Observe that in order to decompress losslessly *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. For each node, starting at root (1) if leaf, output "1" + character (byte), (2) If not leaf, output bit "0", then encode both children (*left*, then *right*) the same way recursively. The number of bits required to store the Huffman tree depends on the number of symbols (2^b) in the discretization of the time

series. There are two reasons why (for simplicity) we do not consider the cost Huffman tree in our formulation:

In practice the size of the tree is negligible compared to the number of bits required to represent the time series.

In practice 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 ten 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 Section 2.4 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 Discrete Wavelet Transform (DWT), Discrete Fourier Transform (DFT), Adaptive Piecewise Constant Approximation (APCA), and Piecewise Linear Approximation (PLA), among others [7].

Figure 10 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 [7] 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 ($T \mid H$), and the number of bits used for encoding H, i.e., DL (H). The reduced description length is defined as:

$$DL\ (T, H) = DL\ (H) + DL\ (T \mid H)$$

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 Piecewise Linear Approximation would include the bits needed to encode the mean, slope and length of each linear segment.

The second term, $DL\ (T \mid H)$, 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 $DL\ (T \mid H) = DL\ (T - H)$.

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

2.3 MDL Modeling of Time Series

2.3.1. 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 5 illustrates a plot of T.

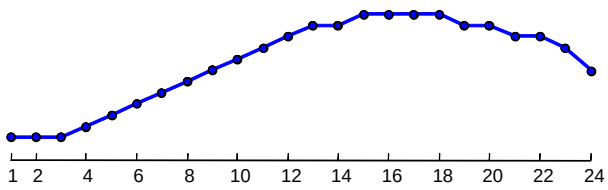


Figure 5: A sample time series T that will be used as a running example in this section

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 Figure 6. It 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.

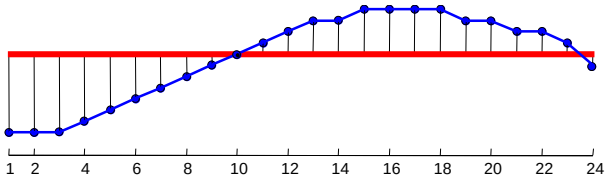


Figure 6: Time series T (blue/fine), approximated by a one-dimensional APCA approximation H_1 (red/bold). The error for this model is represented by the vertical lines

Model H_1 does not approximate T well, and we must account for the error[5]. The errors e_1, represented by the length of the vertical lines in Figure 6**,** 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 7 shows the *two* segment approximation lines created by APCA.

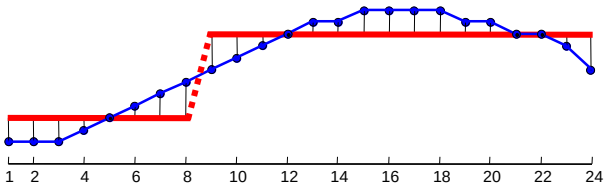


Figure 7: Time series T (blue/fine), approximated by a two-dimensional APCA approximation, H_2 (red/bold). Vertical lines represent the error

[5] 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 Figure 7, is smaller than the error e_1. In particular, the error e_2 is:

e_2 = **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, $DL\ (T \mid H_2)$, 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 hypothesis 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 levels of DFT or PLA representation and test different cardinalities. For example, suppose we had been given T_2 instead:

T_2 = **0 0 0 0 4 4 4 4 4 0 0 0 0 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.

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 Figure 8 which contrasts

the original single-segment approximation shown in Figure 6 with an alternative single-segment approximation.

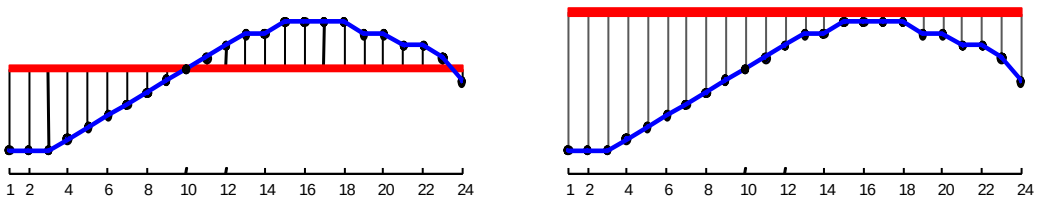


Figure 8 : left) The figure shown in Figure 6 contrasted with an attempt to approximate the raw data with a constant segment that clearly has too great a mean value (right). Note that while the number of repeated residuals ("errors") is identical in both cases, the magnitude of the residuals is much greater in the latter case. It is this unnecessarily large magnitude that tells us this is a poor choice of an approximation

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 Figure 8.*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_x = **a a a b** , and s_y = **m m m n** . However, in our domain there *is* a significant difference between say e_x = **1 1 1 2** , and e_y = **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 Figure 9, we show the 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.

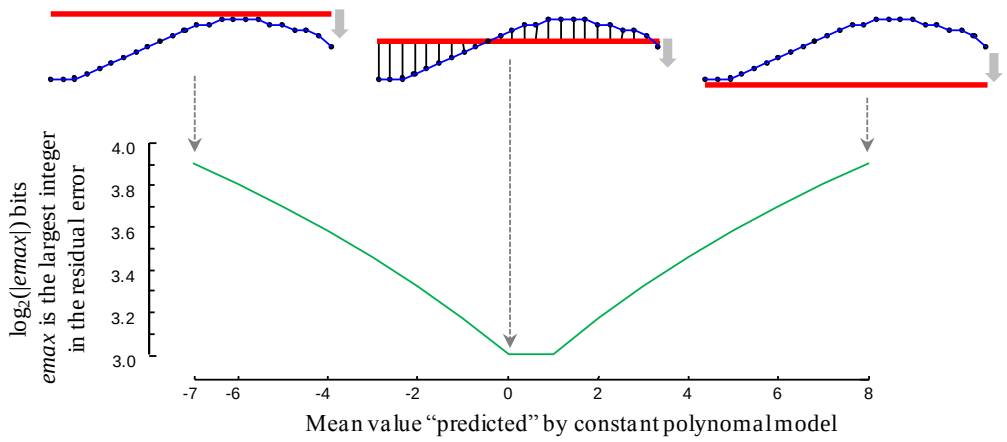


Figure 9 : The $\log_2$ of the range of the residual errors for all possible single constant polynomial models of the data introduced in Figure 5. Note that the model that minimizes this value (with a tie) is also the model that minimizes the residual error.

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[6], they are also implicitly minimizing the range of residuals. As shown by Figure 9, 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 Figure 10, 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.

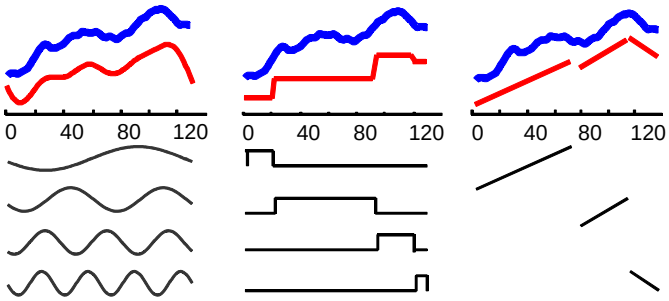


Figure 10: A time series *T* shown in bold/blue and three different models of it shown in fine/red: from left to right: DFT, APCA, and PLA

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

[6] 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 [21][34].

property, we get a "free" coefficient for each one we store [3][7]. 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 [20]. 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 [3] [34].

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.

2.3.2. 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 (line 3-5). Then, a

hypothesis *H* is created based on the selected model and parameters (line 6). For example, a hypothesis *H*, shown in Figure 7, 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.

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
```

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

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* (line 8-13).

2.3.3. Adaptive Piecewise Constant Approximation

As we have seen the example in Section 2.3.1, 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 algorithm.

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*log2c+(d-1)*log2m
 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
```

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 (line 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 Figure 7, 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 (line 8-13).

2.3.4. Piecewise Linear Approximation

An example of a PLA model is shown in Figure 10.*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 $\log_2 c$.

The first two values are represented in cardinality c and thus $\log_2 c$ bits are required for each of them. We also require $\log_2 m$ 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 (line 8-13).

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 = ∞
3. for   c = 2:256
4.    for d = 2 to m/2
5.         H = PLA (T,c,d)
6.         model_cost = 2*d*log2c +(d-1)*log2m
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
```

2.3.5. Discrete Fourier Transform

A data representation in DFT space is simply a linear combination of sine waves, as shown in Figure 10.***left***. Table 4 presents our algorithm specific to DFT. After we

quantize 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 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.

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*log2c+d*log2(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
```

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.

2.3.6. 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 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 [27][34]. It has been shown that averaged over *many* highly diverse datasets, there is not much difference among the different representations [34]. 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 Figure 11. It is easy to observe that there are constant, linear and quadratic patterns in this example. In Sections 2.4.9 and 2.4.10, we further demonstrate the utility of our ideas on real datasets [21][27][33].

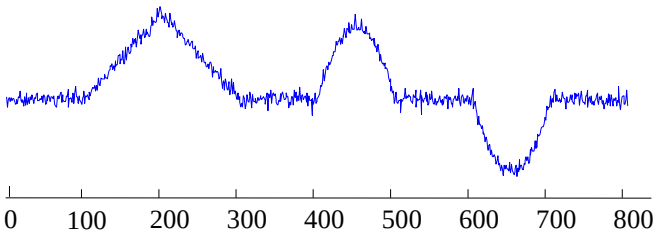


Figure 11 : A toy example of a time series that has more than one state

Several works propose that it may be fruitful to use a *combination* of different models within one time series [21][27][34]. For example, [27] 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 [27] is to minimize the Euclidean error between the model and the original data for a given number of segments. However, note that [27] 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 to 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 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 $\log_2 c$ 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 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)*log2m
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
```

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 Piecewise Linear Approximation (PLA) introduced in [21]. There are two main differences between our bottom-up mixed model algorithm and the bottom-up algorithm described in [21]. 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 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**,** line 2 to line 4. Then the cost of merging each pair of adjacent segments is calculated, as shown in line 5 to line 7. To minimize the merging cost for the two input segments, 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 [21], 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 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.

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
```

2.4 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 [61]. In addition, this website contains additional experiments that are omitted here for brevity.

2.4.1. 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 [8][44][46], our MDL method has two significant advantages over existing works. First, there are 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 [22] we consider a ten-year-old instantiation [46] of a classic benchmark problem [8]. In Figure 12, we show the classic Donoho-Johnstone block benchmark. The underlying model used to produce it consists of twelve piecewise constant sections with Gaussian noise added.

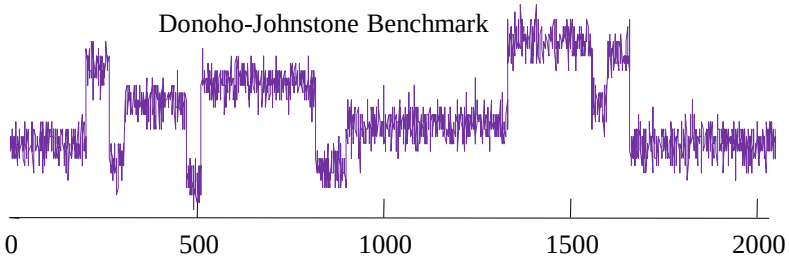


Figure 12 : A version of the Donoho-Johnstone block benchmark created ten years ago and downloaded from [46]

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 [10][11]. 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 [44]. In essence, the 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 13***.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 Figure 13***.bottom***, this occurs at location ten, a reasonable estimate of the true value of twelve.

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 [59]. 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 [61]).

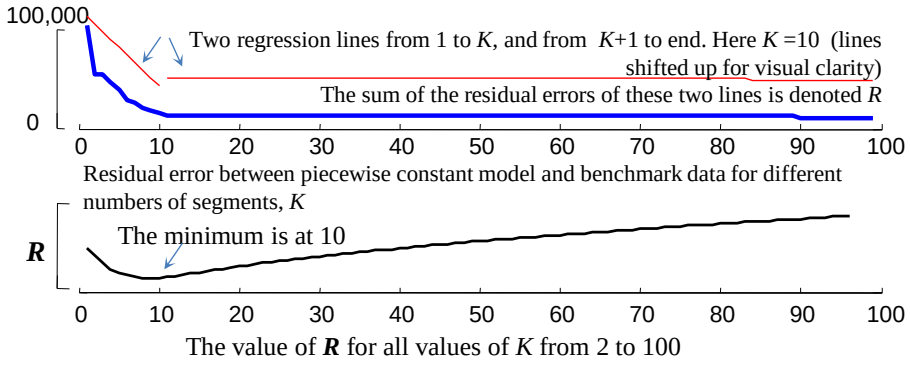


Figure 13 : The knee-finding L-Method. *top*) A residual error vs. size-of-model curve (blue/bold) is modeled by all possible pairs of regression lines (red/light). Here, just one possibility is shown. *bottom*) The location that minimizes the summed residual error of the two regression lines is given as the optimal "knee"

We solve this problem with our MDL approach. Figure 14 shows that of the 64 different piecewise constant models it evaluated, MDL selected the twelve-segment model, which is the *correct* answer.

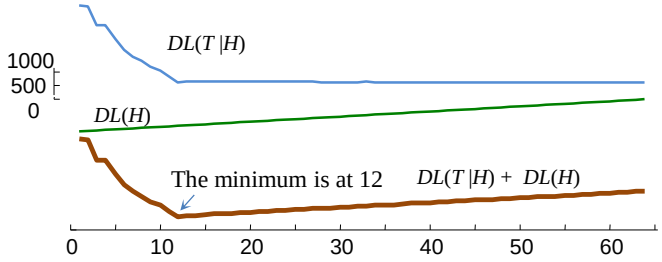


Figure 14 : The description length of the Donoho-Johnstone block benchmark time series is minimized at a dimensionality corresponding to twelve piecewise constant segments, which is the *correct* answer [46]

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 [44][59] are not even defined. In we have repeated the

previous experiment, but this time fixing the dimensionality to twelve as suggested above, and testing all possible cardinality values from 2 to 256.

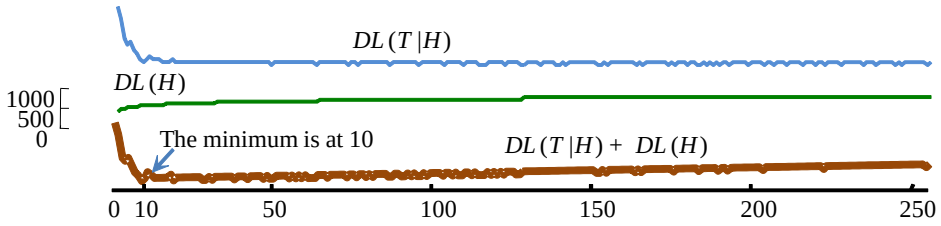


Figure 15 : The description length of the Donoho-Johnstone block benchmark time series is minimized with a cardinality of ten, which is the true cardinality [46]

Here MDL indicates a cardinality of ten, which is the *correct answer* [46]. We also re-implemented the most referenced *recent* paper on time series discretization [11]. 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 [44][59] 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 Figure 16, 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.

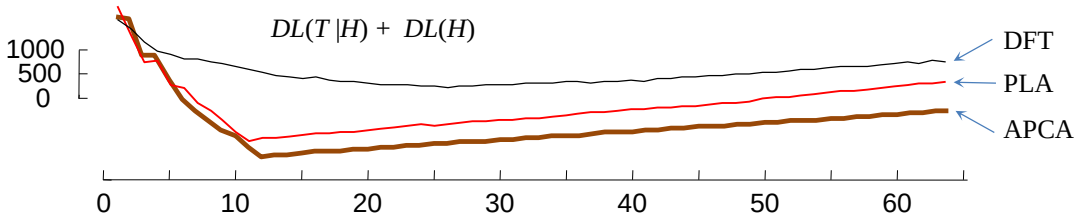


Figure 16: The description length of the Donoho-Johnstone block benchmark time series is minimized with a piecewise constant model (APCA), not a piecewise linear model (PLA) or Fourier representation (DFT)

2.4.2. An Example Application in Physiology

The *Muscle* dataset studied by Mörchen and Ultsch [32] 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 17.*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.

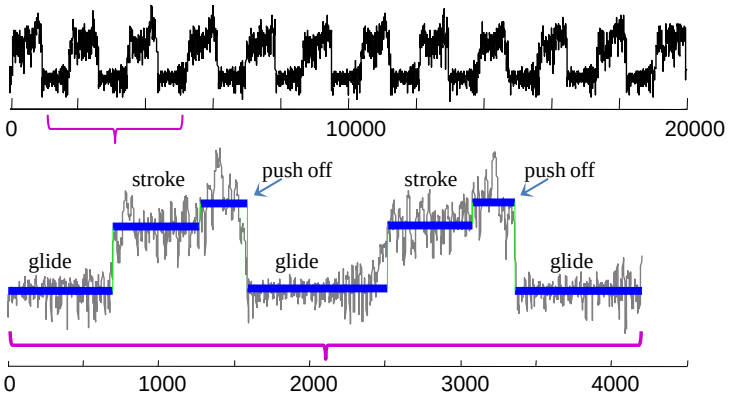


Figure 17 : *top*) An excerpt from the Muscle dataset. *bottom*) A zoomed-in section of the Muscle dataset which had its model, dimensionality and cardinality set by MDL

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 Figure 18.*left*, are worth examining.

Our MDL method suggests a cardinality of three. Glancing back at Figure 17.*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 [32]. However, their algorithm weakly predicts a value of four[7]. Here, once again we find the MDL can outperform this latter approach, even though [32] acknowledges that their reported result is the best obtained after some parameter tuning using additional data from the same domain.

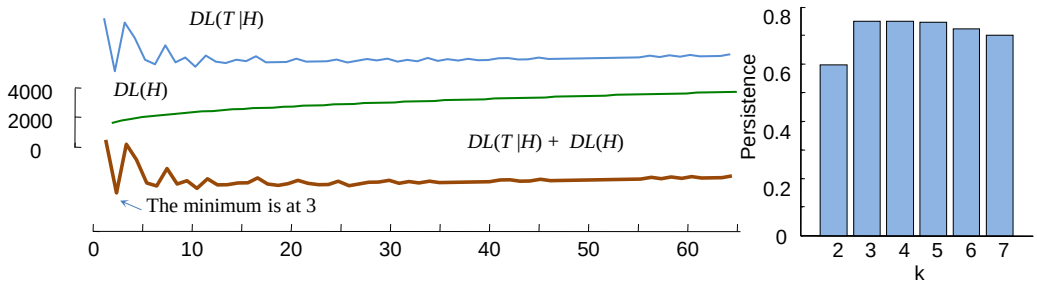


Figure 18: *left*) The description length of the muscle activation time series is minimized with a cardinality of three, which is the correct answer. *right*) The Persist algorithm, using the code from [32], predicts a value of four

2.4.3. 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

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

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. However note that our technique is at least as fast as existing methods [40][41]., 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 [40]. 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 [30]. 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 [40] [41], 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 Figure 19.*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 Figure 19.*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 [40][41]. However, we achieved this result without extensive parameter tuning, and we can do so *very* efficiently.

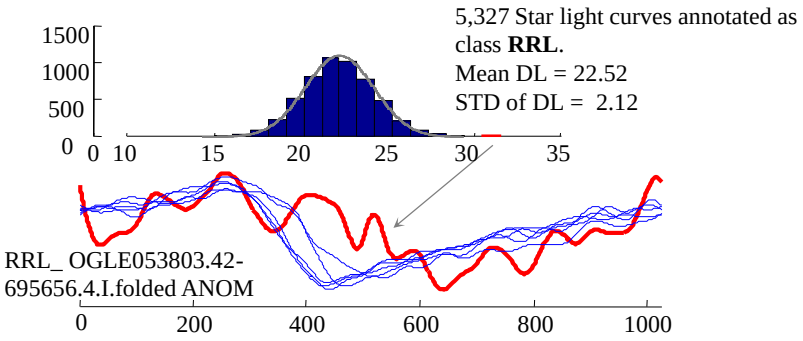


Figure 19: *top*) The distribution of intrinsic dimensionalities of star light curves, estimated over 5,327 human-annotated examples. *bottom*) Three typical examples of the class RRL, and a high intrinsic dimensionality example, labeled as an outlier by [40]

2.4.4. 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 Figure 20.*top*, the distribution is clearly Gaussian

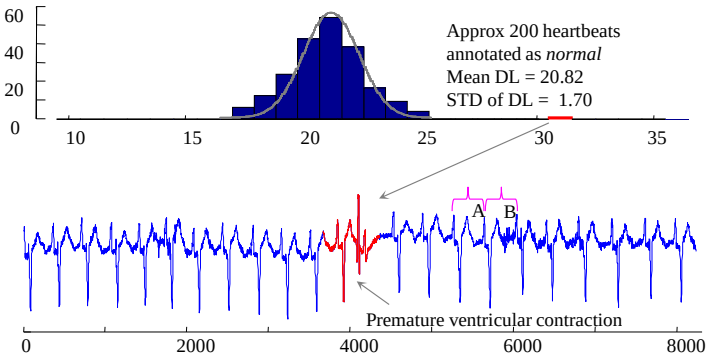


Figure 20: *top*) The distribution of intrinsic dimensionalities of individual heartbeats, estimated over the 200 normal examples in record 108 of the MIT BIH Arrhythmia Database (*bottom*)

. We used these learned values to monitor the rest of the data, flagging any heartbeats that had a dimensionality that was more than three standard deviations from the mean. Figure 20**.***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 [58]. 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 Figure 20.*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 [58], 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 [4].

2.4.5. 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 [37]. In Figure 21**,** 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, reflecting annual periodicity; thus, we were intrigued to find an explanation for the data.

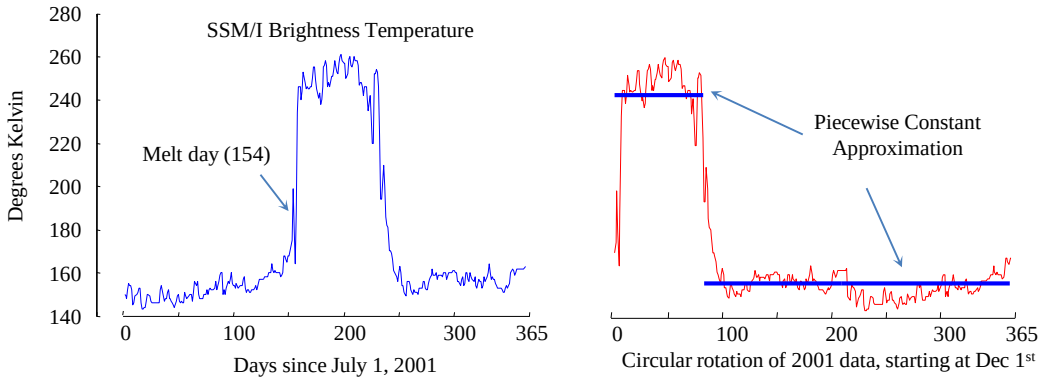


Figure 21: *left*) A time series of temperatures in a region of Antarctica. *right*) Of the hundreds of millions of such time series archived at NSIDC, this time series (and a few thousand more) is unusual in that it has a very low complexity, being best modeled with just two linear segments

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.

2.4.6. 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 [19] describing the annual discharge rate of the Senegal River. This data was measured at Bakel station from the year 1903 to 1988 [19], as shown in Figure 22.

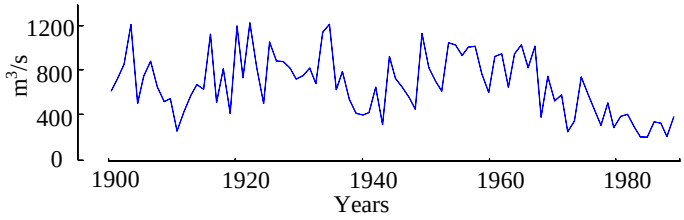


Figure 22: A time series showing the annual discharge rate of Senegal River from the year 1903 to 1988

The authors of [19] reported the optimal segmentation occurs when the number of segments is five using a Hidden Markov Model (HMM)-based segmentation algorithm, as shown in Figure 23.*top*.

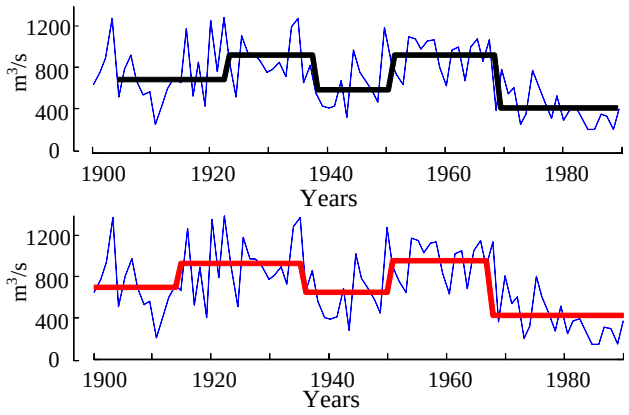


Figure 23: *top*) The blue/light line is Senegal River data. The black/bold line is the segmentation result found in Section 5.1 of [19]. *bottom*) We obtained the red/bold line by hard coding the number of segments to five using the MDL algorithm

As illustrated in Figure 23.*bottom*, we get a similar plot by *hard coding* the number of segments to five, using the MDL-based Adaptive Piecewise Constant Approximation algorithm shown in Table 2. However, as shown in Figure 24, our MDL algorithm actually predicts *two* as its intrinsic dimensionality of data in Figure 22. Note that there is no ground truth for this problem[8]. Nevertheless, for the Donoho-Johnstone block benchmark dataset that has ground truth in Section 2.4.1, we have correctly predicted its intrinsic dimensionality, and we would argue that the two-segment solution shown in Figure 24 is at least subjectively as plausible as the five segment solution.

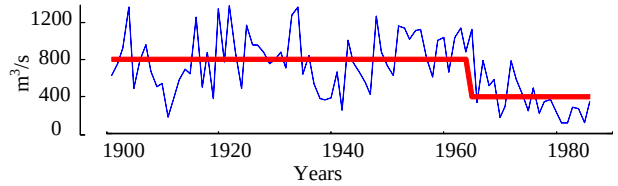


Figure 24: Our MDL algorithm predicts that the intrinsic dimensionality of the annual discharge rate of the Senegal River is *two*. The approximation is shown in red/bold

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 [19], as shown in Figure 25.

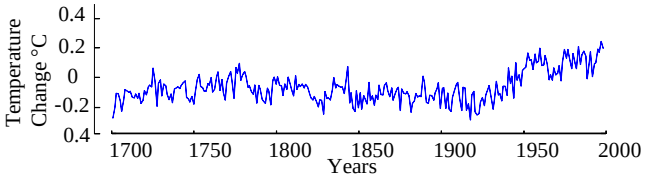


Figure 25: A time series showing the annual global temperature change from the year 1700 to 1981

[8] In [19] 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.

In [19] the authors suggest that the optimal segmentation occurs when the number of segments is four, as shown in Figure 26.*top*.

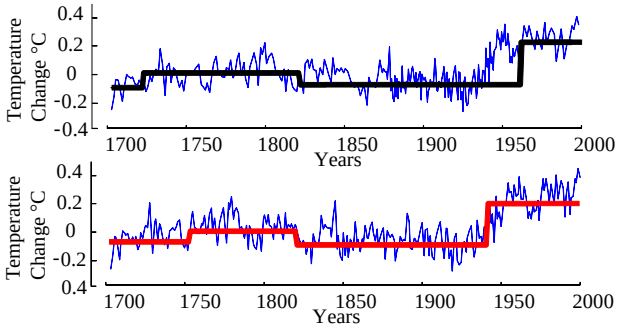


Figure 26: *top*) The blue/light line is the annual global temperature change. The black/bold line is the segmentation result found in Section 5.2 of [19]. *bottom*) We obtained a similar but slightly different model, as shown in the red/bold line, by hard coding the number of segments to four using the MDL algorithm

As illustrated in Figure 26.*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 Figure 27**,** our MDL algorithm predicts that the intrinsic dimensionality of the global temperature change data is *two*.

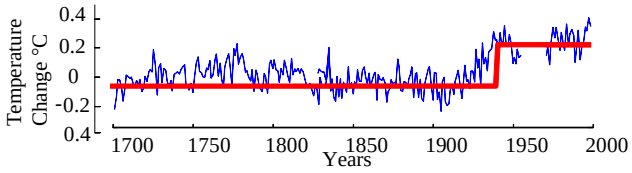


Figure 27: Our MDL algorithm obtains *two* as the intrinsic dimensionality of the time series for the global annual mean temperature

Here, there is at least some tentative evidence to support our two segment model. Research done by [25] [33] and [48] 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 [25][48]. Moreover, there were no significant temperature changes after 1700 other than during this rapid warming period.

2.4.7. An Example Application in Biophysics

In [2], the authors also proposed an HMM-based approach (distinct from, but similar in spirit to that described in [19] and discussed in the previous section) to segment the time series from single-molecule Förster resonance energy transfer (smFRET) experiments. Figure 28 shows a time series from the smFRET experiment [2][54].

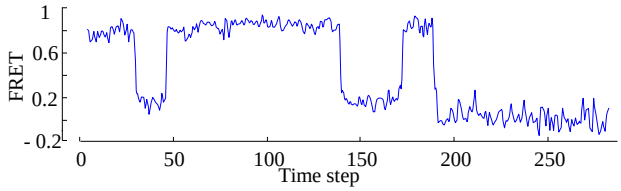


Figure 28: A representative smFRET trace from [2][54]

The authors in [2] 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 Figure 29.*top*). We obtain the same results, as our algorithm finds that the intrinsic cardinality for the data in Figure 28 is also three using the algorithm in Table 2. Figure 29. *bottom*) illustrates our approach. However, there are several parameters in the HMM-based approach used by [2]. 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 [2]) that the data is piecewise constant. The mixed

polynomial algorithm introduced in Section 2.3.6 considered and discounted a linear, quadratic or mixed polynomial model to produce the model shown in Figure 29.*bottom*).

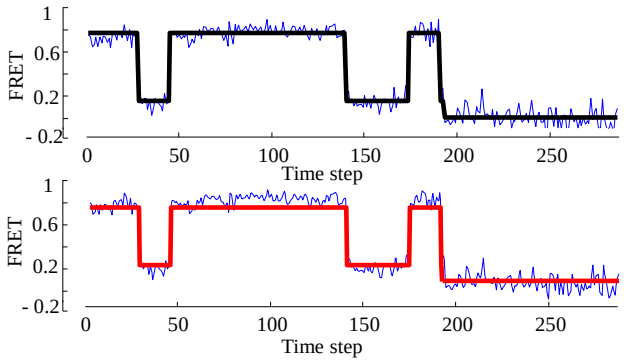


Figure 29: *top*) The time series in blue from Figure 28 is predicted to have three states [2][54]. The approximation is shown in black/bold. *bottom*) Our algorithm also finds three as the intrinsic cardinality. Piecewise constant approximation is shown in red/bold

2.4.8. An Example Application in Prognostics

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

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 [12][39][49][56][57]. Data-driven prognostics are more useful than model-driven prognostics, since a model-driven prognostic requires incorporating a physical understanding of the systems [12]. 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 to 40,000 sensors on board [16]. All of these sensors stream data about the health of the system [16]. 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 [56].

We analyzed the Prognostics and Health Management Challenge (PHM08) dataset which contains data from 233 different engines [38][39]. 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 30 implies that the data from the operational variable and the non-operational variable are visually very similar.

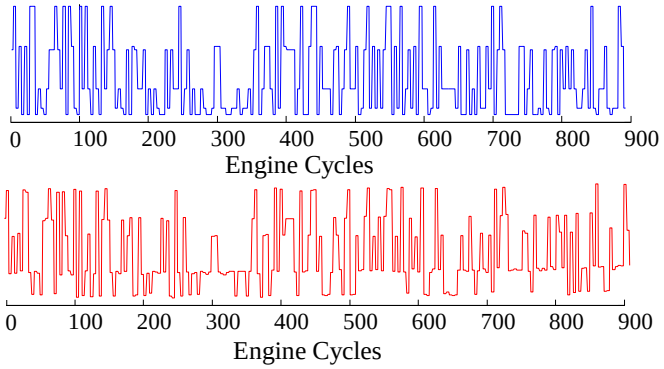


Figure 30: *top*) An example of an *operational* variable in the PHM08 dataset. *bottom*) An example of a *non-operational* variable in the PHM08 dataset

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 Figure 31 represents one variable from one engine.

Although data from the two kinds of variables look very similar (Figure 30), our results in Figure 31 show that there is a significant difference between them: the operational variables lie in the lower left corner of Figure 31 with low cardinalities and small reduced description lengths. In contrast, the non-operational variables lie in the upper right corner of Figure 31 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 [14][38][39][56][57]. 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 [14][56][57], which have the overhead of parameter tuning and are prone to overfitting.

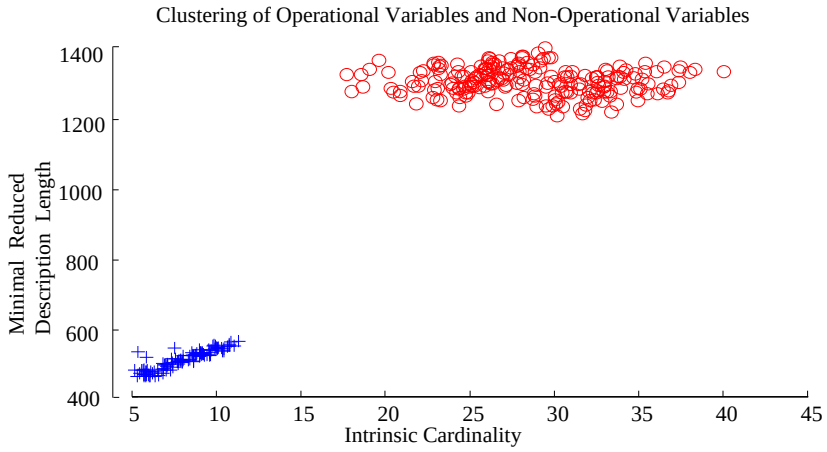


Figure 31: The blue/cross markers represent operational variables. The red/circle markers represent non-operational variables. The variables from 233 engines are analyzed in the plot

2.4.9. Testing the Mixed Polynomial Degree Model

In Section 2.3.6 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 Figure 33. As shown in Figure 32, we calculated its intrinsic dimensionality using the algorithms in Table 5 and Table 6. The minimum cost occurs when the dimensionality is eight.

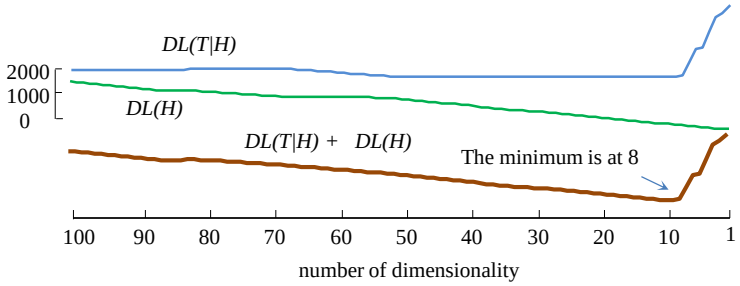


Figure 32: The description length of the synthetic time series shown in Figure 11 minimizes when the dimensionality is eight, which is the intrinsic dimensionality

Figure 33 shows the data and its intrinsic mixed polynomial degree representations. In Figure 33 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.

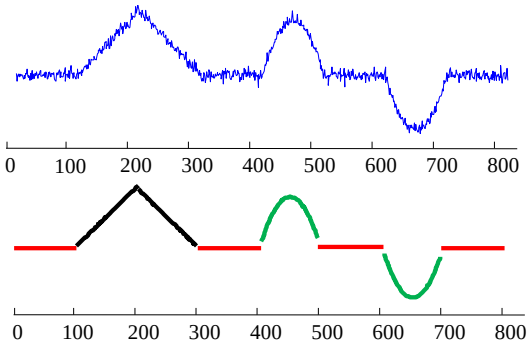


Figure 33: *top*) A toy time series shown in Figure 11 has constant, linear and quadratic segments. *bottom*) data in *top)* is represented by a mixed polynomial degree model. The segments are brushed with different colors according to the polynomial degree of the representations. Red

indicates a constant representation. Black indicates a linear representation and green indicates a quadratic representation

2.4.10. 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 Figure 34.

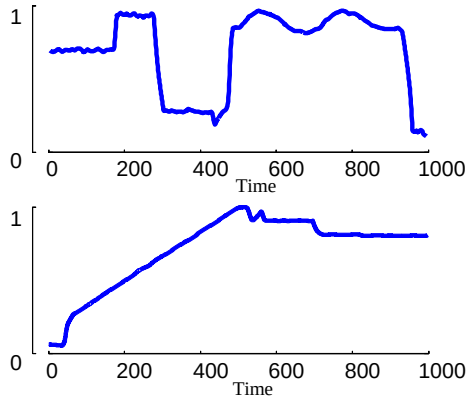


Figure 34: *top*) One snippet of a space shuttle time series that clearly has more than one state. *bottom*) Another space shuttle time series that has more than one state

Using only one approximation to represent the time series like the ones in Figure 34 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 Table 5 and Table 6 to the data shown in Figure 34. The algorithm returns different polynomial degrees for different segments, as demonstrated in Figure 35**.**

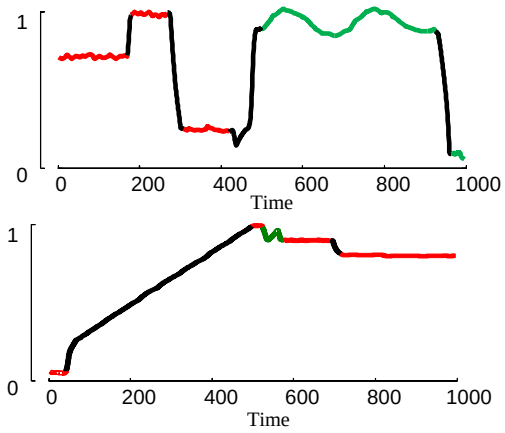


Figure 35: The data shown in Figure 34 after we applied our mixed polynomial degree segmentation. The segments are brushed with different colors according to the polynomial degree of the representations. Red indicates a constant representation. Black indicates a linear representation and green indicates a quadratic representation

We observe that there are three different states in both of the two time series shown in Figure 34. An understanding of the processes that produced this data seems to support this result [23].

2.4.11. 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. In every case, we push our algorithm *passed* its failure point, and by archiving *all* data [61] 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 Sections 2.4.1 and 2.4.2 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 [46]. Our version is essentially identical to the version shown in Figure 12, 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 Figure 7. 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 Figure 36.

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 Figure 36.

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 Figure 36.

As shown in Figure 36 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 Figure 36.*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 [45]. As we can see, the cardinality prediction fails at a SNR of about -1.7, and the dimensionality prediction shortly thereafter.

In Figure 36.*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 [46]. 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 Figure 36.*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 zero 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.

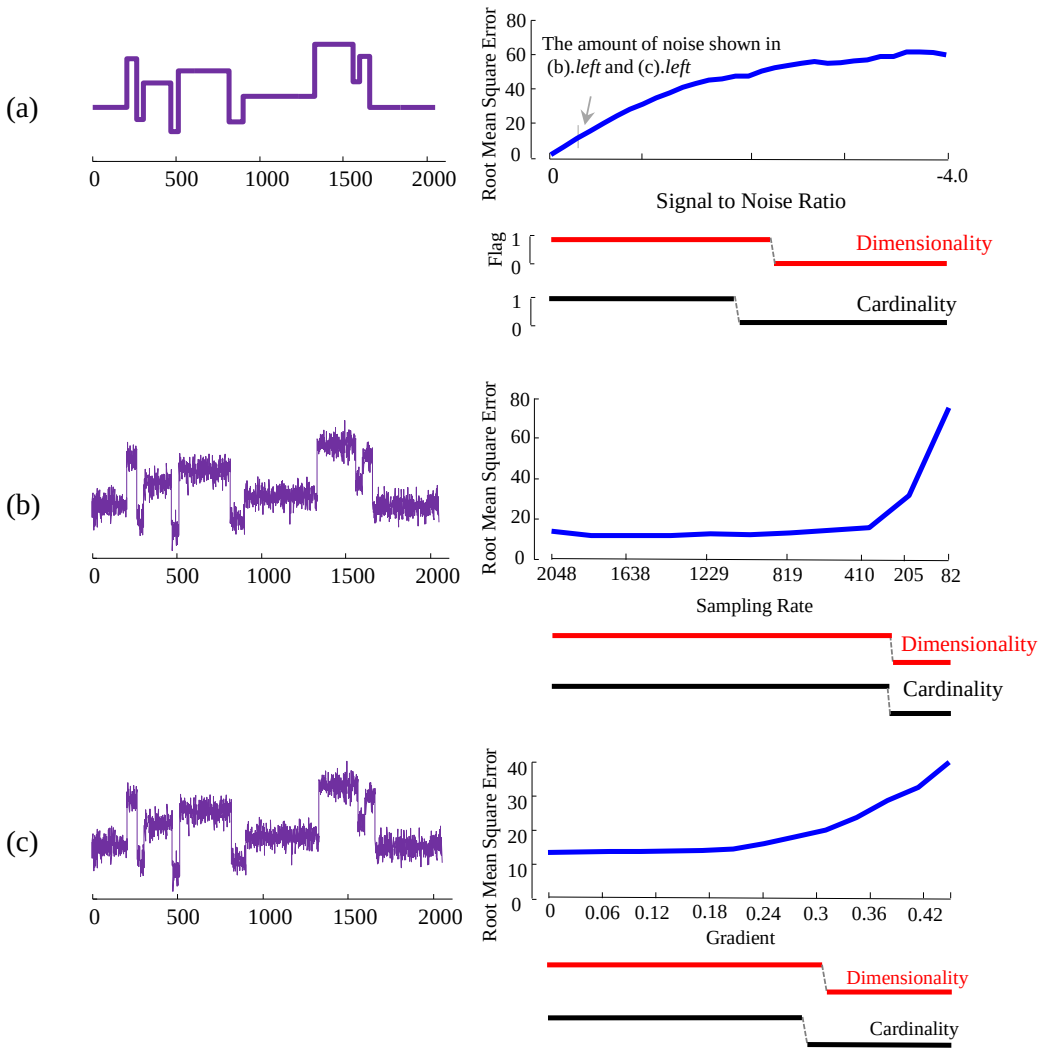


Figure 36: The robustness of our algorithm to various distortions added to the DJB data. In (a).*left* we show the DJB data with no noise, and in (a).*right* we plot the RMSE between (a).*left* and the corrupted versions. In (b).*left* we start with the same data shown in Figure 12. (the noise level 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 Figure 12, and in (c).*right* we plot the RMSE between (c).*left* and the linear trend added versions

The results in Figure 36 suggest our algorithm is quiet robust to these various distortions. To give the reader a better appreciation of *when* our algorithm fails, in Figure

37 we show the most corrupted version of the signals for which our algorithm correctly predicted *either* the cardinality or the dimensionality.

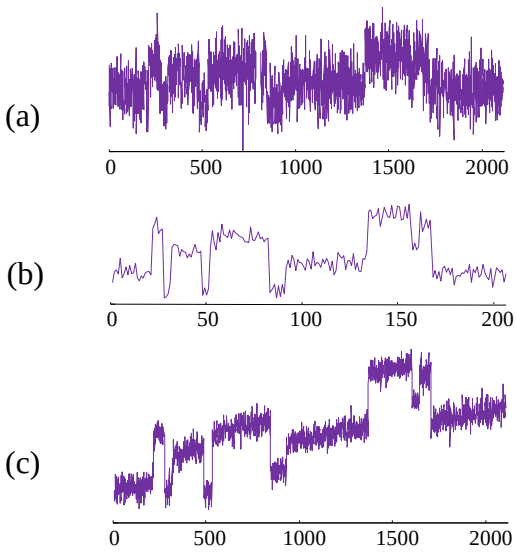


Figure 37: The three most corrupted versions of the Donoho-Johnstone block for which our framework makes a correct prediction of *either* the cardinality or the dimensionality. (a) The nosiest example, (b) the example with the lowest sampling rate, (c) the example with the greatest linear trend added

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 Figure 16, of testing over *all* models spaces in {DFT, APCA, PLA}, our algorithm does correctly predict the data in Figure 37(c) as consisting of piecewise linear segments, and still correctly predicts the cardinality *and* the dimensionality.

2.5 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 Table

2 and Table 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 Section 2.4.5 were obtained over a year and the datasets in Section 2.4.6 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 Table 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 (DJB) data

as an example in Figure 38, there is no need to search over the whole matrix, because changing the cardinality from 512 to 4 does not produce different predicted dimensionalities.

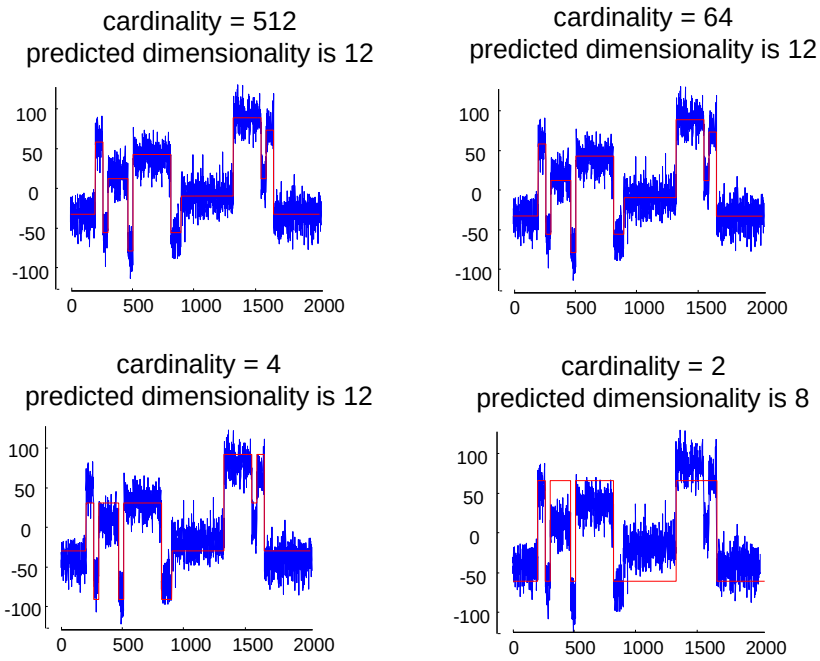


Figure 38: A comparison of the effect from differing cardinalities on our framework's ability to discover the correct intrinsic dimensionality of DJB data. For any cardinality from 512 to 4, the discovered intrinsic dimensionality does not change. Only when the cardinality is set to a pathologically low three or two (*bottom right*) does the cardinality value affect the predicted dimensionality

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 39 demonstrates the comparison of running time using APCA and our MDL framework. As Figure 39 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$m$).

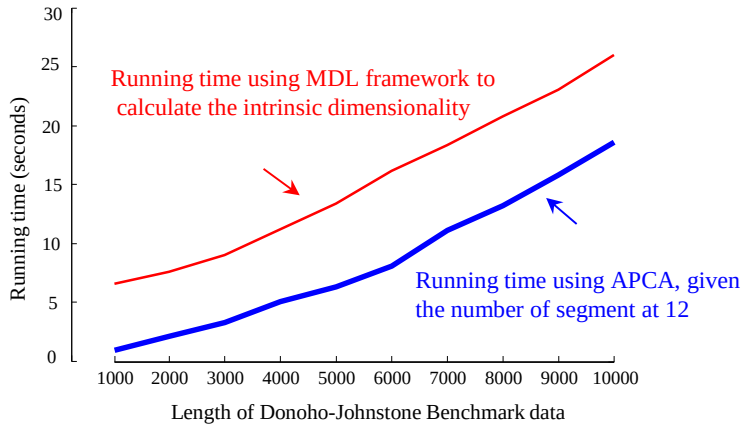


Figure 39: The running time comparison between our MDL based approach (red/fine) and the APCA (blue/bold) approximation for Donoho-Johnstone benchmark dataset. The x axis is the length of different instantiations of the DJB data

2.6 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 [26] 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 Figure 17, 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 Figure 20 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, [10] 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, [31] 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 [5] 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 [51][53] that were inspired by the original conference version of this work [15].

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.

2.7 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 [61].

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[9], 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 [21].

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

Chapter 3:

Time Series Classification under More Realistic Assumptions

Most literature on time series classification assumes that the beginning and ending points of the pattern of interest can be correctly identified, both during the training phase and later deployment. In this work, we argue that this assumption is unjustified, and this has in many cases led to unwarranted optimism about the performance of the proposed algorithms. As we shall show, the task of correctly extracting individual gait cycles, heartbeats, gestures, behaviors, etc., is generally much more difficult than the task of actually classifying those patterns. We propose to mitigate this problem by introducing an alignment-free time series classification framework. The framework requires only very weakly annotated data, such as "in this ten minutes of data, we see mostly normal heartbeats...," and by generalizing the classic machine learning idea of data editing to streaming/continuous data, allows us to build robust, fast and accurate classifiers.

We demonstrate on several diverse real-world problems that beyond removing unwarranted assumptions and requiring essentially no human intervention, our framework is both significantly faster and significantly more accurate than current state-of-the-art approaches.

This chapter is organized as follows: In Section 3.1, we introduce definitions and notation used in this chapter. Note that although some of the terms (i.e. time series) have

already been defined in Chapter 2: in order to make each chapter self-contained, we still redefine the term in each chapter. In Section 3.2.1, we show how classification is achieved with our data dictionary model. In Section 3.2.2, we illustrate how to actually *learn* the data dictionary by utilizing data editing techniques [94][101][107][110]. Section 3.2.3 demonstrates how our framework learns the threshold distances. We demonstrate the algorithm to remove the forth assumption by using the algorithm introduced in Section 3.2.4. In Section 3.3, we present a detailed empirical evaluation of our ideas. We discuss related work in Section 3.3.5. Finally, in Section 3.4, we offer conclusions and directions for future work.

3.1 Definitions and Notation

We begin with the definition of *time series*:

Definition 6: *Time Series*: T = $t_1, \ldots t_m$ is an ordered set of *m* real-valued variables.

We are only interested in *local* properties of a time series, thus we confine our interest to *subsequences*:

Definition 7 : *Subsequence*: Given a time series T of length m, a subsequence S_k of T is a sampling of length $n \leq m$ of contiguous position from T with starting position at *k*, $S_k = t_k, \ldots t_{k+n-1}$ for $1 \leq k \leq m-n+1$.

The extraction of subsequences from a time series can be achieved by use of a *sliding window*:

Definition 8 : *Sliding Window*: Given a time series T of length *m*, and a user-defined subsequence length of *n*, all possible subsequences can be extracted by

sliding a window of size *n* across T and extracting each subsequence, S_k. For a time series T with length *m*, the number of all possible subsequences of length *n* is *m*-*n*+1.

For concreteness, we take the step of explicitly defining training data, as our definition of *training data* explicitly removes the assumptions inherent in most works [72][79][84][88][96][104][108].

Definition 9 : *Training Data*: A Training Data C is a collection of the *weakly-labeled* time series annotated by behavior/state or some other mapping to the ground truth.

By *weakly-labeled* we simply mean that each long data sequence has a single global label and not lots of local labeled pointers to every beginning and ending of individual patterns, e.g., individual gestures. There are two important properties of such data that we must consider:

Weakly-labeled training data may contain *extraneous/irrelevant sections*. For example, after a subject reaches down to turn on an ankle sensor to record her gait, there may be a few seconds before she actually begins to walk [104]. Moreover, during the recording session, the subject may pause to shop, or jump to avoid a puddle. It seems very unlikely that such recordings could *avoid* having such spurious data. Note that this claim is not mere speculation; we observed this phenomenon in the first few seconds of the BIDMC Congestive Heart Failure dataset [68] as shown in Figure 40, and similar phenomena occur in all the datasets we examined.

Weakly-labeled training data will almost certainly contain significant redundancies. While we want lots of data in order to learn the inherent variability of the concept we wish to learn, significant redundancy will make our classification algorithms slow when deployed. Consider Figure 40 once more. Once we have a single normal heartbeat, say pattern A, then there is little utility in adding any of the 14 or so other very similar patterns, including pattern B. However, to robustly learn this concept (beats belonging to Record-08), we must add either example of the Premature Ventricular Contraction (PVC).

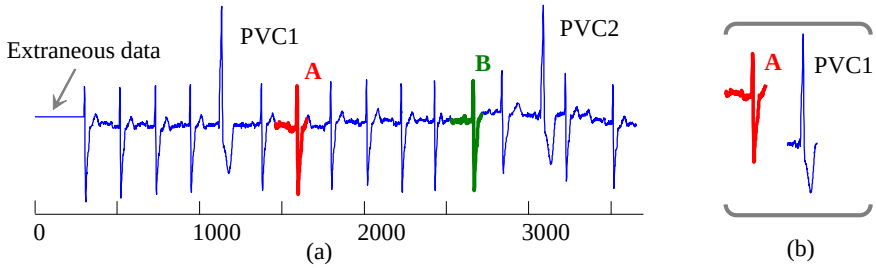


Figure 40: A snippet of BIDMC Congestive Heart Failure Database ECG - Record-08 [68]. (a) is weakly-labeled data, which exhibits both *extraneous* data, a section of recording when the machine was not plugged in, and *redundant* data (only one pair of redundancies are shown in bold (red/green). (b) A minimally redundant set of representative heartbeats (a *data dictionary*) could be used as training data

Rather than these large *weakly-labeled* training datasets, we desire a smaller "smart" training data subset that does not contain *spurious* data, while maintaining coverage of the target concept by having one (ideally, *exactly* one) instance of each of the many ways the targeted behavior is manifest. For example, from the training data shown in Figure 40, we want just one PVC example and just one example of a normal heartbeat (perhaps *either* **A** *or* **B**). However, we do not want to require costly human effort to obtain this. While the time series shown in Figure 40 would be fairly easy to edit by hand, it is only 0.16% of the full ECG dataset we consider in Section 3.3.2. Therefore, our objective is to

build this idealized subset of the training data automatically. We begin by defining it more concretely as a *data dictionary.*

Definition 10 : A *Data Dictionary* **D** is a (potentially very small) "smart" subset of the training data. We allow an input parameter *x*, where *x* is the percentage of the training data C used in data dictionary **D**. The range of *x* is (0,100%], and a dictionary with the percentage *x* of the original data is denoted as $\mathbf{D}_x$.

As the *Data Dictionary* is at the heart of our contribution, we will take the time to discuss it in detail.

3.1.1. A Discussion of Data Dictionaries

As defined above, there are a huge number of possible data dictionaries for any percentage *x*, as any random subset of C satisfies the definition. However, we obviously wish to create one with some desirable properties.

Clearly, the classification error rate obtained from using just **D** should be no worse than that obtained from using all the training data. We do not wish to sacrifice accuracy. As we shall show, this is a surprisingly easy objective to achieve. In fact, as we shall show later, the classification error rate using a judiciously chosen **D** is generally significantly lower than using all of C. This is because the data dictionary contains less spurious –and therefore, potentially *misleading*–data.

Another desirable property of **D** is that it be a very small percentage of the training data. This is to allow real-time deployment of the classifier, especially on resource limited devices (embedded devices, smartphones, etc. [67][75]). This requirement may be seen as conflicting with the above classification *error rate* requirement; however, again

we will show that in most real-world problems we can judiciously throw away more than 95% of C to obtain a $\mathbf{D}_{5\%}$ that is *at least* as accurate as using all the data in C.

Note that the number of subsequences within each class in **D** may be different. That is to say, our algorithm for building **D** is *not* round-robin; rather the algorithm adaptively adds more subsequences to cover the more "complicated" classes of **D**. For example, the ECG data from Record-08 shown in Figure 40 is relatively simple. In contrast, the ECG of Record-03 shown in Figure 41 has a more complicated trace, and at least four kinds of beats (normal, S, PVC and Q). Therefore, we might expect the number of subsequences for Record-03 in **D** to be greater than that for Record-08, something that is empirically borne out in our experiments (Section 3.3).

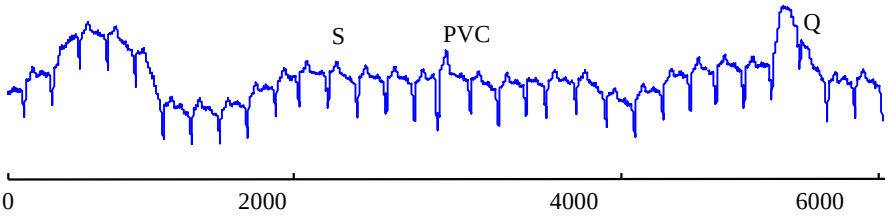


Figure 41: A snippet of BIDMC Congestive Heart Failure Database ECG: Record-03 [68]. Note that this section of ECG data exhibits more variability than the data in Figure 40.

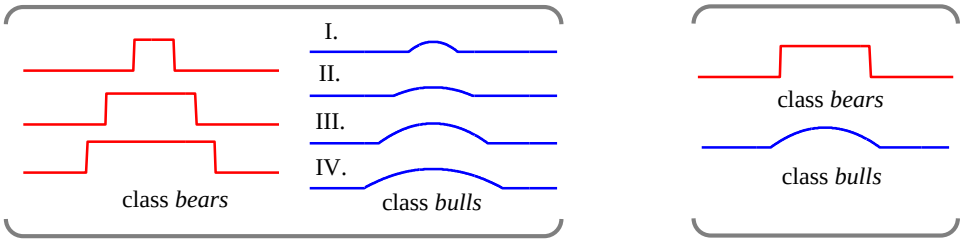
Finally, there is the question of what value we should set *x* to. In fact, we can largely bypass this issue by providing an algorithm that produces a "spectrum" of data dictionaries in the range of $x = (0,100\%]$, together with an estimate of their error rate on unseen data. The user can examine this error rate vs. value-of-*x* curve to make the necessary trade-offs. Note that these data dictionaries are "nested", that is to say, for any value of *x* we have $\mathbf{D}_x \subseteq \mathbf{D}_{x+\varepsilon}$. Thus, we can consider our data dictionary creation algorithm an *anyspace* algorithm [110].

Given the above considerations, how can we build the best data dictionary? As we will later show, we can heuristically search the space of data dictionaries using the simple algorithm in Section 3.2.2.

3.1.2. An Additional Insight on Data Redundancy

Based on our experience with real-world time series problems, we noted the following: in many cases, **D** contains many patterns that appear to be simply (linearly) rescaled versions of each other. For clarity, we illustrate our point with a synthetic example in Figure 42; however, we will later show some real examples.

This situation is a consequence of our requirement that data dictionary **D** has the most representative subsequences of training data C. For example, if one class contains examples of `walk`, we hope to have at least one representative of each type of `walk`—perhaps one example of a `leisurely-amble`, one example of a `normal-paced-walk`, one example of a `brisk-walk`, etc. It is important to note that in this example, the three walking styles are *not* simply linearly rescaled versions of each other. They have different foot strike patterns, and thus produce different prototypical time series templates [69][92]. Nevertheless, *within* each sub-class of `walk`，there may also be a need to allow some linear rescaling of the time series. Using the *Euclidean* distance our search algorithm can achieve this by attempting to ensure that the data dictionary contains each gait pattern over a range of speeds. This is what our toy example in Figure 42 illustrates.



left) Data dictionary A *right*) Data dictionary B

Figure 42: *left*) A toy example data dictionary which was condensed from a large dataset. These seven subsequences in data dictionary A span the concept space of the bulls/bears problem. *right*) Note that if we had a distance measure that was invariant to linear scaling, we could further reduce data dictionary A to data dictionary B

For example, when reducing a dataset of daily human activities, we may have to extract examples of a `brisk- walk` at 6.0km/h, 6.1km/h, 6.2km/h, etc. However, by generalizing from the *Euclidean* distance to the *Uniform Scaling* distance [87], we allow our algorithm to keep just one example of the `walk`, and *still* achieve coverage of the target concept by using a flexible measure *instead* of lots of data. The *Uniform Scaling* distance is a simple generalization of the *Euclidean* distance that allows limited invariance of the length of the patterns being matched [87]. The maximum amount of linear scaling allowed is a user-defined parameter [87]. As we later show, allowing just a small amount of scaling, say 25%, can greatly improve accuracy.

To see this in a real dataset, consider Figure 43.*left*, which shows one of fifteen classes that was processed into a data dictionary in an experiment we performed in Section 3.3.2. At first glance, the two patterns seem redundant[10], violating one of the requirements stated above.

[10] Note the fact that the two patterns are out of phase does not make them non-redundant, as at query time only queries half their length are used, and they are sliding across the entire length of the patterns. Details in Section 3.3.2.

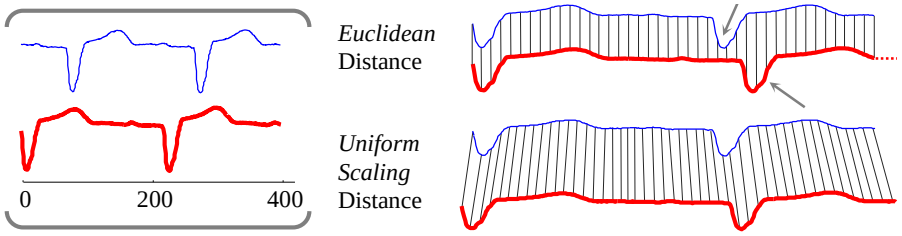


Figure 43: *left*) A data dictionary learned from a 15-class ECG classification problem (just class 01 is shown here). At first glance, the two exemplars seem redundant apart from their (irrelevant) phases. *right*) By using the *Euclidean* distance between the two patterns we can see that the misalignment of the beats would cause a large error. The problem solved by using the *Uniform Scaling* distance [87]

Instead of having two similar but different scaled patterns, just a single pattern is kept using the *Uniform Scaling* distance. We have found that using the *Uniform Scaling* distance allows us to have a significantly smaller data dictionary. In Figure 43, we could delete either one of the two patterns and cover the space of possible heartbeats from Record-01. For example, in Figure 42, we could further delete patterns I, II and IV and still cover the space of possible "*bulls*".

However, beyond reducing the size of data dictionaries (thus speeding up classification), there is an additional advantage of using *Uniform Scaling*; it allows us to achieve a lower error rate. How is this possible? It is possible because we can generalize to patterns *not seen* in the training data.

Imagine the training data does contain some examples of gaits at speeds from 6.1 to 6.5km/h. As noted above, if the data dictionary has enough examples to cover this range of speeds, we should expect to do well. However, suppose the unseen data contains some walking at 6.7km/h. This is only slightly faster than we have seen in the training data, but the *Euclidean* distance is very sensitive to such changes [87]. Using the *Uniform Scaling* distance allows us to generalize our labeled example at 6.5km/h to the brisker 6.7km/h

instance. This idea is more than speculation. As we show in Section 3.3, using the *Uniform Scaling* distance does produce a significantly lower error rate.

3.1.3. On the Need for a Threshold

As noted above, the training set may have extraneous data. Likewise, in most realistic deployment scenarios, we expect some (often *most*) of the data to be classified as the `other` class. In these cases, we wish our algorithm to label the objects as such. To achieve this, the data dictionary must have a distance threshold `r` beyond which we reject the query as unclassifiable (i.e., the `other` class). As we will show, we can learn this threshold as we build the dictionary.

3.2 Algorithms

In order to best explain our framework, we first assume a data dictionary with the appropriate threshold has already been created and begin by explaining how our classification model works. Later, in Section 3.2.2, we revisit the more difficult task of learning the data dictionary.

3.2.1. Classification Using A Data Dictionary

Our classification model requires just a data dictionary with its accompanying threshold distance, `r`.

For an incoming object to be classified `q`, we classify it with the data dictionary using the classic nearest neighbor algorithm [107]. In Table 7, we show how to determine the class membership of this query, including the possibility that this query does not belong

to any class in this data dictionary. For our purposes, there are exactly two possibilities of interest:

If the query's nearest neighbor distance is larger than the threshold distance, we say this query does not belong to any class in this data dictionary (line 12).

If the query's nearest neighbor distance is smaller than the threshold distance, then it is assigned to the same class as its nearest neighbor (line 14).

The algorithm begins by initializing the `bsf` distance to infinity and the predicted `class_label` to `NaN` in lines 1 and 2. From lines 3 to 9, we find the nearest neighbor of the query `q` in data dictionary **D**. The subroutine `NN_search` (shown in Table 8) returns the nearest neighbor distance of `q` within a time series. If the nearest neighbor distance within a time series in line 4 is smaller than the `bsf`, then in lines 6 and 7 we update the `bsf` and the `class_label`.

Table 7: Classification Algorithm using Data Dictionary

Input:	**D**, a data dictionary that has N classes; The total number of time series in D is k **r**, a threshold distance of D **q**, a query
Output:	**The class membership of q**, including the possibility of a special class 'other'
1	bsf = ∞; //initialize the best-so-far distance
2	class_label = NaN;
3	***for*** i = 1 to k
4	dist = NN_search(q, D(i));
5	***if*** dist < bsf
6	bsf = dist;
7	class_label = class of D(i);
8	***endif***
9	***endfor***
10	NN_dist = bsf;
11	***if*** NN_dist > r
12	***return*** q belongs to 'other' class;
13	***elseif*** NN_dist <= r
14	***return*** q belongs to 'class_label'th class;
15	***endif***

From lines 11 to 15, we compare the nearest neighbor distance to the threshold distance `r`. If the nearest neighbor distance is smaller than `r`, then this query belongs to the same class as its nearest neighbor. Otherwise, this query does not belong to any class within this data dictionary and is thus classified as the `other` class.

As we show in Table 7 line 4, the function `NN_search` is slightly different from the classic nearest neighbor search algorithm [84]. `NN_search` returns not only the nearest neighbor distance of a query, but also a distance vector that contains distances between the query and *all* the possible subsequences in a time series. This distance vector is not exploited at classification time, but as we show in Section 3.2.2, it is exploited when building the data dictionary. For concreteness, we briefly discuss the `NN_search` function in Table 8 below.

Table 8: Nearest Neighbor Search within a Time Series

Input:	**q**, a query **T**, a time series
Output:	**dist_vector**, a vector that contains distances between q and all possible subsequences in T
	NN_dist, the nearest neighbor distance
1	w = set of all possible subsequences in T;
2	dist_vector = zeros(1,|w|);
3	***for*** i = 1 to |w|
4	dist_vector(i) = distance(q,w(i));
5	***endfor***
6	NN_dist = minimum(dist_vector);
7	***return*** dist_vector ;
8	***return*** NN_dist ;

In line 1, using a sliding window (cf. Definition 8), we extract all the subsequences of the same length as the query. From lines 3 to 5, the distances between `q` and all the possible subsequences are calculated. We calculate the nearest neighbor distance in line 6. Note that in line 4, the distance could be *Euclidean* distance [84], or *Uniform Scaling* distance [87], etc. We will revisit this choice in Section 3.3.

In addition to finding the nearest neighbor, this function also returns a distance vector. This additional information is exploited by the dictionary building algorithm discussed later in Section 3.2.2. Figure 44.*bottom* shows an example of such a distance vector.

Having demonstrated how the classification model works in conjunction with the data dictionary, we are in position to illustrate how to build the data dictionary, which is a more difficult task.

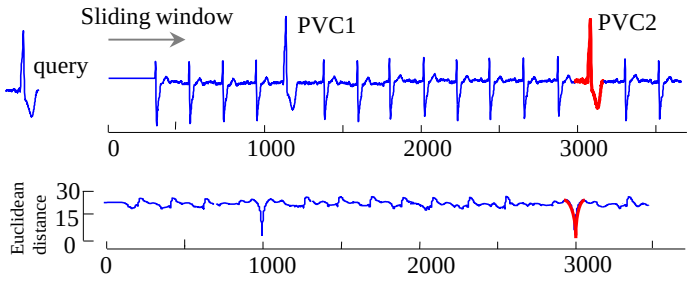


Figure 44: *top*) A snippet of BIDMC Congestive Heart Failure Database ECG data: Record-08 [68]. *bottom*) the distance vector of an incoming query. The nearest neighbor and its distance of `q` is colored in red/bold

3.2.2. Building the Data Dictionary

As discussed in Section 3.1, we want to build the data dictionary automatically. Using human effort to *manually* edit the training data into a data dictionary is clearly not a realistic solution: as it is not scalable to large datasets and invites human bias into the process.

Before introducing our dictionary-building algorithm, we will show a worked example on a toy dataset in the *discrete* domain. We use a small discrete domain example simply because is it easy to write intuitively; our real goal remains large real-valued time series data.

A. The intuition behind data dictionary building

Suppose we have a training dataset that contains two classes, C_1 and C_2:

C_1 = { d**pace**kfjkl**walk**fl**walk**kl**pace**dalyutek**walk**sfj}

C_2 = { jhjh**leap**ashl**jump**okdjkl**leap**hf**leap**fj**jump**acgd}

In this toy example, the data is *weakly-labeled*. The colored/ bolded text is for the reader's introspection only; it is *not* available to the algorithm. Here the reader can see that in C_1, there appears to be two ways a shorter subsequence query might belong to this class; if it contains the word ***pace*** or ***walk***. This is similar to the situation shown in Figure 40 where a query will be classified to the class of Record-08 if it contains pattern **A** or pattern PVC.

We want to know whether any incoming queries belong to either class in this training data or not. In our proposed framework, we search *just* the data dictionary.

Recall that one of the desired properties of the data dictionary is that it contains a minimally redundant set of patterns that is representative of the training data. In this example for C_1, these are clearly the substrings ***pace*** and ***walk***. Likewise for C_2, ***leap*** and ***jump*** seem to completely define the class. Thus, the data dictionary **D** should be the following:

D = C_1:{ ***pace*** ; ***walk*** }; C_2: { ***leap*** ; ***jump***}, `r` = 1

Consider now two incoming queries *ieap* and *kklp*. The former is a noisy version of a pattern found in our dictionary, but as it is within our rejection threshold of (hamming) distance `r` of 1, it is correctly labeled as C_2. In contrast, *kklp* has a distance of 3 to its nearest neighbor in **D**, so it is correctly rejected.

Note that had we attempted to classify against the raw data rather than the dictionary, the query *kklp* would have been classified as C_1 (it appears in the middle of ..l**walk**kl**p**aced.). This misclassification is clearly contrived, but it *does* happen frequently in the real data. Consider the flat section of time series at the beginning of Figure 41. As noted above, it is *extraneous* data, due to a temporary disconnection of the sensor. However, many other patients' ECG traces also have these flat sections, but clearly that does not mean we should classify them as belonging to patient Record-08.

In our example, we have considered two separate queries; however, a closer analogue of our real-valued problem is to imagine an endless stream that needs to be classified:

.. ttgpacedgrteweerjumpwalkflqrafertwqhafhfahfahfbseew..

Up to this point we have not explained how we built our toy dictionary. The answer is simply to use the results of leaving-one-out classification to score candidate substrings. For example, by using leaving-one-out to classify the first substring of length 4 in C_1 *dpac*, it is incorrectly classified as C_2 (it matches the middle of ..**ump**acgd.. with a distance of 1). In contrast, when we attempt to classify the second substring of length 4 in C_1, *pace*, we find it is correctly classified. By collecting statistics about which substrings are often used for correct predictions, but rarely used for wrong predictions, we find that the four substrings shown in our data dictionary emerge as the obvious choices. This basic idea is known as *data editing* [94][97][107]. In the next section, we formalize this idea, and generalize it to *real*-valued data streams.

B. Building the data dictionary

The high-level intuition behind building the data dictionary is to use a ranking function to score every subsequence in C. These "scores" rate the subsequences by their *expected utility* for classification of future unseen data. We use these scores to guide a greedy search algorithm, which iteratively selects the best subsequence and places it in **D**. How do we know this utility? We simply estimate it by cross validation, e.g. looking at the classification error rate and some additional information as explained below.

As previously hinted, our algorithm iteratively adds subsequences to the data dictionary. Each iteration has three steps. In Step 1, the algorithm scores the subsequences in C. In Step 2, the highest scoring subsequence is extracted and placed in **D**. Finally, in Step 3, we identify all the queries that cannot be correctly classified by the current **D**. These incorrectly classified items are passed back to Step 1 to re-score the subsequences in C.

There is an important caveat. Once we have removed the best subsequence in Step 2, the scores of all the other subsequences may change in the next iteration. To return to our running example in Figure 40, *either* subsequence **A** and **B** would rank highly. However once we have placed one, say **A**, in **D**, there is little utility in adding **B**, since having **A** in **D** is sufficient to correctly classify similar patterns in Step 3. Thus we expect the scores of **B** will be low in the next iteration, given that the correctly classified queries by the current **D** will not be used to re-score C in the next iteration.

The process iterates until we run out of subsequences to add to **D** or the unlikely event of *perfect* training error rate having been achieved. In the dozens of problems we

have considered, the training error rate plateaus well before 10% of the training data has been added to the data dictionary.

Below we consider each step in detail.

Step 1 : In order to rank every point in the time series, we use the leaving-one-out classification algorithm[11]. However, we do not want to use *just* the classification error rate to score the subsequences. Imagine we have two subsequences S_1 and S_2, either of which is found to correctly predict 70% of the queries tested with them. Either appears to be a good candidate to add to **D**. However, suppose that in addition to being close enough to many objects with the *same* class label (*friends*), allowing its 30% error rate, further suppose that S_1 is also very close to many objects with *different* class labels (*enemies*). If S_2 keeps a larger distance from its enemy class objects, it is a much better choice for inclusion in **D**.

This idea, that instead of using just the error rate of classification, you must also consider the relative distance to "*friends*" and "*enemies*" has been investigated extensively in the field of data editing [97][107].

Given a query length *l*, we randomly choose a query `q` from the training data C[12]. In Table 9, lines 2 and 3, we first split the training data into two parts, Part A (*friends* only) and Part B (*enemies* only). Using the `NN_search` algorithm in Table 8, we find *nearest neighbor friend* in Part A (lines 5 to 13) and *nearest neighbor enemy* (lines 14 to 22) in Part B.

[11] Where tractably is an issue, we may sample a subset of the queries.

[12] We defer the discussion on how to choose a query length to Section 3.4.

In lines 23 to 27, the nearest neighbor *friend* distance and the nearest neighbor *enemy* distance are compared. If the nearest neighbor *friend* distance is smaller than the nearest neighbor *enemy* distance, we discover all the distances of the query `q` in Part A that are also smaller than the *nearest neighbor enemy* distance. Such subsequences are *likely true positives*. That is to say, our confidence that these subsequences can produce correct classifications of unseen data has increased.

Table 9: Classification of Training Data

Input:	**C**, the training data
Output:	**likely true/false positive subsequences**
1	q = a randomly selected subsequence in C;
2	A = *friends* ;
	//all the time series in C that have the same class as q, q is removed from A;
3	B = *enemies* ; // all the time series in C that have different class from q;
4	dists_A = []; dists_B = [];
5	bsf = ∞; //initialize the best-so-far distance
6	***for*** i = 1 to |A|
7	[dist_vector, NN_dist] = NN_search(q, A(i));
8	***if*** NN_dist < bsf
9	bsf = NN_dist;
10	***endif***
11	dists_A = [dists_A ; dist_vector];
12	***endfor***
13	NN_friend_dist = bsf; // nearest neighbor distance in same class
14	bsf = ∞; //initialize the best-so-far distance
15	***for*** j = 1 to |B|
16	[dist_vector, NN_dist] = NN_search(q, B(j));
17	***if*** NN_dist < bsf
18	bsf = NN_dist;
19	***endif***
20	dists_B = [dists_B ; dist_vector];
21	***endfor***
22	NN_enemy_dist = bsf; // nearest neighbor distance in different class
23	***if*** NN_friend_dist < NN_enemy_dist
24	likely_true_positives = find(dists_A < NN_enemy_dist)
25	***elseif*** NN_friend_dist >= NN_enemy_dist
26	likely_false_positives = find(dists_B < NN_friend_dist)
27	***endif***

Similarly, if the nearest neighbor *friend* distance is larger than the nearest neighbor *enemy* distance, we find all the distances of the query `q` in Part B that are also smaller than the *nearest neighbor friend* distance. We call the corresponding subsequences *likely false positives*.

Given the *likely true/false positives* found in Table 9, we are now in a position to discuss how to rank them.

By utilizing the simple rank function introduced in [107], we generalize an algorithm that gives positive score to *likely true positives* and negative score to the *likely false positives*.

$$rank(S) = \sum_{k} \begin{cases} 1, & likely\ true\ positives \\ -2/(num\_of\_class - 1), & likely\ false\ positives \\ 0, & other \end{cases} \quad (1)$$

Note that subsequences that are not used to classify any queries (correctly or not) get a zero score. Using a large number of queries, we compute a score vector for every time series in C. We denote rank(S) as the score for a subsequence S in the time series.

In the next step, we demonstrate how to extract the current best subsequence using the score vectors.

Step 2: We extract the highest scoring subsequence and place it in **D**. We demonstrate this step by using the example in Figure 45. Suppose in one of the iterations in Step 1, the starting point of the red/bold heartbeat has the highest score. We therefore need to extract this heartbeat. Because the *Euclidean* distance is very sensitive to even slight misalignments, and our scoring function is somewhat "blurred" as to its exact location in the x-axis. Extracting *exactly* the subsequence with query length *l* would be very brittle. Therefore, we "pad" the chosen subsequence some time series from the left and to the right, in particular with the *l*/2 data points to either side.

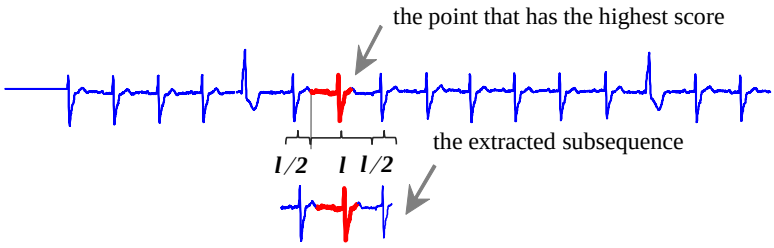


Figure 45: top) A snippet of BIDMC Congestive Heart Failure Database ECG data: Record-08 [68]. bottom) the extracted subsequence has twice the query length

Note that there is a slight difference between the first iteration and the subsequent iterations. Before the first iteration, **D** is empty. After the first iteration, **D** should contain exactly one subsequence from each class. This is the smallest **D** logically possible. Therefore, instead of splitting C to the *friends* part and the *enemies* part, the algorithm finds the most representative subsequence in *each* class in Step 1, and then adds them into **D** in Step 2.

After the first iteration, we extract only the one subsequence that holds the highest score in C and add it into **D**. Thus, the class sizes in **D** can be skewed, as the algorithm adds more exemplars to the more diverse/complicated classes. While we are iteratively building **D**, the size of C becomes smaller, as the extracted subsequence is removed from C.

Step 3: The algorithm examines the quality of the current **D** by doing classification using all the queries. The queries that are correctly classified by the current **D** will not be used to re-score C in the next iteration Step 1, since the current **D** is sufficient to correctly classify them. Only the misclassified queries will proceed back to Step 1 to re-score C. In Step 3, we redo classification experiments on D using all the queries, since the correctly classified queries in $\mathbf{D}_x$ may become misclassified in $\mathbf{D}_{x+\varepsilon}$.

After building a data dictionary for a training data, our last obligation is to learn the distance threshold.

3.2.3. Learning the Threshold Distance

After the data dictionary is built, we learn a threshold to allow us to reject future queries, which do not belong to any of our learned classes. We begin by recording a histogram of the nearest neighbor distances of testing queries that are *correctly* classified using **D**, as shown in Figure 46. Next, we compute a similar histogram for the nearest neighbor distances of queries, which should *not* have a valid and meaningful match within **D** (i.e., the `other` class). Where can we get such queries? In the example shown in Figure 46, we simply used gesture data as the `other` class, knowing *gestures* should not match a set of *heartbeats*. Note that it is occasionally possible that a gesture will match a heartbeat by coincidence; but our approach is robust to such spurious matches so long as they are relatively rare. If external datasets are in short supply, we can also simply permute subsequences of **D** to produce the `other` class, for example flipping heartbeats upside-down and backwards.

Given the two histograms, we choose the location that gives the equal-error-rate as the threshold (about 7.1 in Section 3.3.1).However, based on their tolerance to false negatives, users may choose a more liberal or conservative decision boundary.

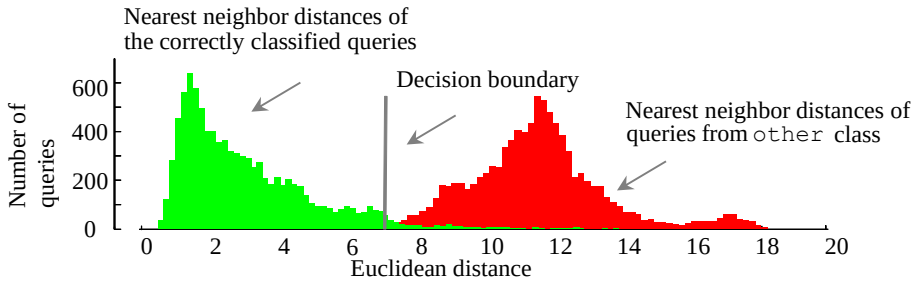


Figure 46: The *green*/left histogram contains the nearest neighbor distances of correctly classified queries for the ECG data used in Section 3.3.2. The *red*/right histogram shows nearest neighbor distances for queries from the `other` class

3.2.4. Anytime Classification using Complexity As An Index

A. Anytime classification

Anytime classification algorithm is the algorithm that sacrifices the quality of experimental results for faster running time [77][105][112]. The algorithm becomes interruptible after a short time of initialization. Figure 47 demonstrates the tradeoff between the quality of experimental result and the computation time.

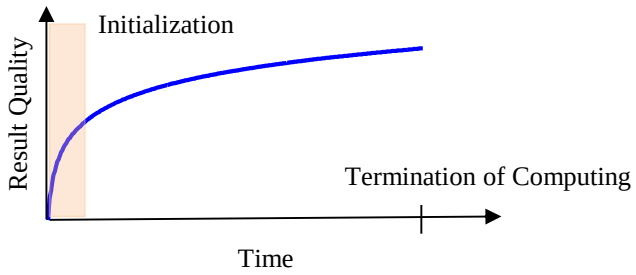


Figure 47: Anytime algorithms are interruptible after initialization. This plot shows the result quality increases with computation time

Anytime classification algorithm can mitigate the assumption that the arriving time of queries is known ahead of time, since the computation can be interrupted any time after a short time of initialization. Table 10 demonstrates how the anytime classification framework works. The algorithm begins by initializing the bsf distance to infinity and the

predicted class_label to NaN in lines 1 and 2. In line 3, we extracted all the possible subsequences in the training data in a specific order. Note that the order of all the subsequences can be defined by different methods. For example, in Table 8 line 1, all possible subsequences are extracted in a left-to-right order using a sliding window. Thus the search process in Table 8 is sequential search. In contrast, in Section B. , we propose to index all the possible subsequences using complexity of each subsequence. We show our proposed indexing method in Table 11. From line 4, we start to calculate the distance between q and each subsequence. If the distance is smaller than the bsf in line 6 is smaller than the bsf, then in lines 7 and 8 we update the bsf and the class_label. From line 10 and line 12, if the stopFlag is true, then the computation will be stopped and return the current class_label associated with the bsf distance.

Table 10: Anytime Nearest Neighbor Classification Algorithm

Input:	**T, the training data** q, a query stopFlag = 0, the value of stopFlag can be changed during the computation process;
Output:	**The class membership of q;**
1	bsf ← ∞; // initialize the best-so-far distance
2	class_label ← NaN;
3	subs ← some_order_of_all_possible_subsequences(T);
4	**for** i ← 1 to |subs|
5	dist ← distance(q,subs(i));
6	**if** dist < bsf
7	bsf = dist;
8	class_label = subs(i).class;
9	**end**
10	**if** stopFlag == 1
11	return class_label;
12	**end**
13	**end**

B. Using complexity as an index

We propose to use complexity as an index to speed up the search within the data dictionary. Every subsequence is indexed according to its complexity. The complexity of

a time series can be calculated by different methods, such as Kolmogorov complexity [89], variants of entropy [62][64],etc. There are several desirable properties of a complexity measure [66], such as,

- Low time and space complexity;
- Few parameters, ideally none;
- Intuitive and interpretable;

Given the above consideration, we propose to use one complexity measure shown in equation (2), which has O (1) space and O (n) time complexity. More importantly, this complexity measure has a natural interpretation with zero parameter.

$$CE(q) = \sqrt[2]{\sum_{i=1}^{n-1}(q_i - q_{i+1})^2} \qquad (2)$$

We are not claiming this is the optimal indexing approach for speed up. We want to show an existence proof of an indexing technique that can mitigate the assumption (4). Table 11 demonstrates the indexing algorithm. In line 1, we calculate the complexity of the incoming query. We extract all the possible subsequences in line 2 using a sliding window. The sliding window length is the same length as the query. This extraction is the same as the one in line 1 in Table 8. From line 3 to line 6, we calculate the absolute difference of the complexity between the query and each subsequence. Last we sort all the complexity differences with an ascending order in line 7. After the sorting, the subsequence, which is closer to the query in terms of complexity, will have a higher rank. In another word, in Table 10 line 3, the subsequence that has a similar complexity as the

query does, will have a higher priority for calculating its distance between the incoming query.

Table 11: using Complexity as an Index

```
Input:      T, training data
            q, an incoming query
Output:     indexing_Order,The indexing using complexity for all the
            subsequences in T
1   CE_q ← CE(q); // equation (2)
2   |subs|← all_possible_subsequence(T);
3   for i ← 1 to |subs|
4       CE_subs(i) ← CE(subs(i));
5       diff_CE(i) ←  abs|CE_subs(i) – CE_q|;
6   end
7   indexing_Order = ascend_sort(diff_CE);
```

3.2.5. Uniform Scaling Technique

Finally, we can trivially replace the *Euclidean* distance with *Uniform Scaling*[13] distance in the above data dictionary building and threshold learning process [87]. We choose the maximum scaling factor based on the variability of time series in the domain at hand, see discussion in Section 3.3. A naive implementation of *Uniform Scaling* would be slow, but [87] shows that it can be computed in essentially the same time as *Euclidean* distance.

3.3 Experimental Evaluation

We begin by discussing our experimental philosophy. To ensure that our experiments are easily reproducible, we have built a website, which contains all the datasets and code [114]. In addition, this website contains additional experiments, which are omitted here

[13] The reader may ask why not Dynamic Time Warping? Empirically, we tried it and it does not help. Moreover, we should *not* expect it to help this problem [114].

for brevity. Our experimental results support our claim that using *only* the data dictionary is more accurate and faster than using *all* the available training data.

We compare our algorithm with several widely used rival approaches. The most widely used rival approach extracts feature vectors from the data and reports the best result among multiple models [67][98][104]. In addition, we compare with the obvious strawman of using *all* the training data, which is just a special case of our framework, in which all the training data is used (i.e. $\mathbf{D_{100\%}}$).

To support our claim that the real-world streaming data is not as clean as the contrived datasets used in most literature, we report the percentage of the rejected queries produced by the learned threshold and show some examples[14].

We report the error rate using both *Euclidean* distance and *Uniform Scaling* distance to support our claim that the latter can be very useful for time series classification problems.

While we are ultimately interested in the *testing* error rate, we also report the *training* error rate, as this can be used to predict the best size of the data dictionary for a given problem. However, for completeness, we build and test the data dictionary $\mathbf{D}_x$ for every value of x, from the smallest logically possible size to whatever value minimizes the holdout error rate (this is generally much less than $x = 10\%$).

The reader may object that error rate is not the correct measure here. Imagine that our rejection threshold is so high that we reject 999 of 1,000 queries, and just happen to get one classified object correct. In this case, reporting a 0% error rate would be dubious at

[14] Due to space limitations, we only show the rejected queries in the first case study. See [114] for examples of rejected queries from the other case studies.

best. This is of course what precision/recall and similar measurements are designed to be robust to. However, in all our case studies, our rejection rate is much less than 10%, so reporting just the error rate is reasonable, and allows us to present more visually intuitive figures. Moreover, we will show experiments where we consider the correctness of rejections made by our algorithm.

Finally, we defer experiments that consider the scalability of dictionary building to [114], noting in passing that this is done offline, and that in any case we can do this faster than real-time. In other words, we can learn the dictionary for an hour heartbeats in much less than one hour.

3.3.1. An Example Application in Physiology

We consider a physical activity dataset containing eight subjects performing activities such as: `normal-walking`, `walking-very-slow`, `descending-stairs`, `cycling`, and `inactivity` (an umbrella term for lying-in-bed/sitting-still/standing-still), etc [95]. Approximately eight hours of data at 110Hz was collected from wearable sensors on the subjects' wrist, chest, and shoes.

For simplicity of exposition, we consider only a *single* time series, recording the roll-axis from the sensor placed in the subjects' shoe. However, our algorithm trivially extends to multi-dimensional data (examples appear at [114]). Note that although our algorithm only uses a *single* axis from the sensor, we demonstrate that our results are significantly better than rival algorithms that use all *three*-axis data (roll, pitch and yaw) from the same sensor [104].

We randomly choose 60% of the data as training data, and treat the rest as testing data. In Figure 48, we show the training/testing error rates as our algorithm grows **D** from the smallest logically possible size (about 0.39% of all the training data) to the point where it is clear that our algorithm can no longer improve. Although our algorithm bottoms out earlier in the plot, we wish to demonstrate that the output is very smooth over a wide range of values.

We compare with the widely-used rival approach [67][104], which extracts signal features from the sliding windows. For fairness to this method, we used their suggested window size [104], and tested *all* of the following classifiers: K-nearest neighbors (K=5), SVM, Naïve Bayes, boosted decision trees and C4.5 decision tree [67][98][104]. The *best* classification result is 0.364 achieved by the C4.5 decision tree.

For the commonly used strawman of using all the training data, the testing error rate is 0.221. However, our framework equals this testing error rate using only 1.6% (i.e. $\mathbf{D}_{1.6\%}$) of the training data and obtains the significantly lower error rate of 0.152 at $\mathbf{D}_{8.3\%}$. Moreover, given that we are using only about one-twelfth the data, we are able to classify the data about twelve times faster.

Our algorithm is clearly highly competitive, but does it owe its performance to choice of *which* subsequences are placed in **D** by our algorithm? To test this, we built another **D** by *randomly* extracting subsequences from C. As Figure 48 also shows, our systematic method for ranking subsequences is significantly better than *random* selection.

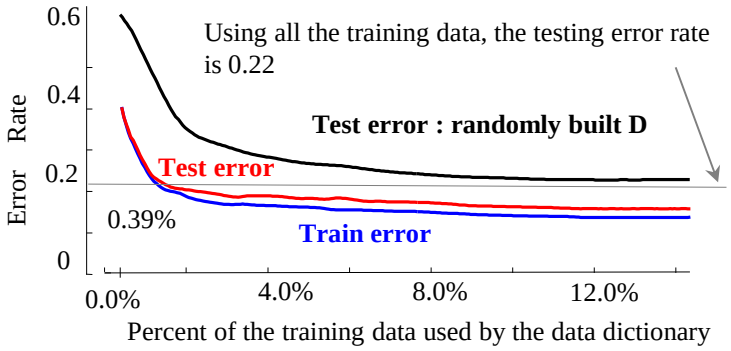


Figure 48: The classification error rates for D from $D_{0.39\%}$ to $D_{14.2\%}$ for the physical activity dataset [95]

A final observation about these results is that the training error rate is a very good predictor of the test error rate. As Figure 48 shows, the training error is only *slightly* optimistic.

We are now ready to test our claim that *Uniform Scaling* (c.f. Section 3.1.2) can help in datasets containing signals acquired from human behavior/physiology. We repeated the experiments above under the exact same conditions, except we replaced *Euclidean* distance with *Uniform Scaling* distance in both the training and testing phases.

Based on studies of variability for human locomotion [62][69][92], we chose a maximum scaling factor of 15%; that is to say, queries are tested at every scale from 85% to 115% of their original length. *Uniform Scaling* obtains a 0.085 testing error rate at $\mathbf{D_{8.1\%}}$, significantly better than *Euclidean* distance, as shown in Figure 49.

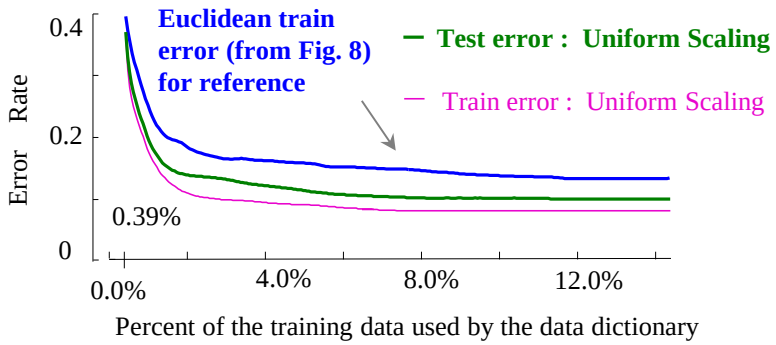


Figure 49: The *pink/green(bold)* curves are train/test error rates obtained when we replaced *Euclidean* distance with *Uniform Scaling* distance

We learned a threshold distance of 14.5 for $\mathbf{D}^{15}$. With this threshold, our algorithm rejects 9.5% of the testing queries. In Figure 50, we see that the vast majority of rejected queries do belong to the `other` class and are thus correctly rejected.

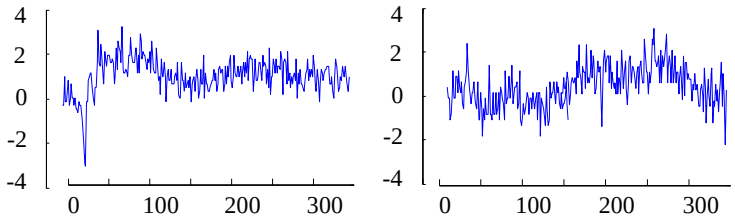


Figure 50: Two examples of rejected queries. Both queries contain significant amount of noise

We do not present formal numerical results for the rejected queries, as the weakly-annotated format of the original data does not provide the label of the objects with certainty.

This dataset draws from sporting activities. We also consider a similar but independent dataset [67], which considers more quotidian activities such as `tooth-brushing` etc. We achieve near identical improvements on this dataset, thus we relegate a discussion of it to [114].

[15] Experimental results show that the threshold distances for **D** built with *Euclidean* distance and *Uniform Scaling* distance are almost identical. Therefore, we only report one threshold distance.

3.3.2. An Example Application in Cardiology

We apply our framework to a large ECG dataset: the BIDMC Congestive Heart Failure Database [68]. The dataset includes ECG recordings from fifteen subjects with severe congestive heart failure. The individual recordings are each about 20 hours in duration, sampled at 250 Hz.

Ultimately, the medical community wants to classify patient-independent *types* of heartbeats. However, in this experiment, we classify *individuals'* heartbeats. This is simply because we are able to obtain huge amounts of labeled data this way. Note that as hinted at in Figure 41, the data *is* complex and noisy. Moreover, a single (unhealthy) individual may have many different types of beats. Cardiologist Helga Van Herle from USC informs us this is a perfect proxy problem.

We use a randomly selected 150 minutes of data for training, and 450 minutes of data for testing.

In Figure 51, we show the training/testing error rates as our algorithm grows the data dictionary from the smallest possible size ($\mathbf{D}_{0.28\%}$) to the point where it is clear that our algorithm can no longer improve.

Note that the testing error rate is 0.102 using the strawman of using *all* the training data, which is significantly better than the default error rate 0.933. However, our framework duplicates this error rate using only 2.1% (i.e. $\mathbf{D}_{2.1\%}$) of the training data**,** and obtains the much lower error rate of 0.076 at $\mathbf{D}_{4.5\%}$. From Figure 51 we again see that our method for building dictionaries is much better than random selection.

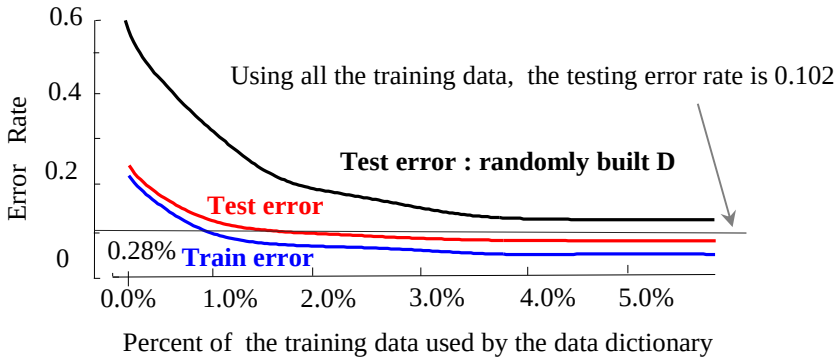


Figure 51: The classification error rates for D from $D_{0.28\%}$ to $D_{5.82\%}$ for BIDMC Congestive Heart Failure Database [68]

We again test the *Uniform Scaling* distance instead of *Euclidean* distance in both the training/testing phases. Based on studies of variability for human heartbeats [68][73] and advice from a cardiologist, we chose a maximum scaling factor of 25%. In Figure 52, *Uniform Scaling* obtains a 0.035 testing error rate at $\mathbf{D_{4.6\%}}$, significantly better than using the *Euclidean* distance.

As illustrated in Figure 46, the threshold distance for **D** is 7.1. With this threshold, the algorithm rejects 4.8% of the testing queries. Once again, these rejections (which can be seen at [114]) all seem like reasonable rejections due to loss of signal or extraordinary amounts of noise/machine artifacts.

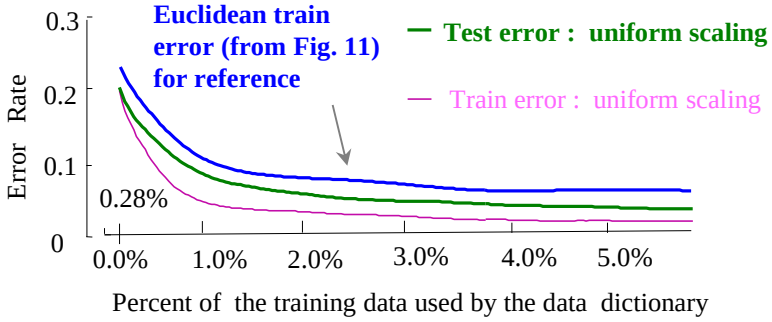


Figure 52: The *pink/green(bold)* curves are train/test error rates obtained when we replaced *Euclidean* distance with *Uniform Scaling* distance

3.3.3. An Example Application in Daily Acitivies

Finally, we apply our framework to a widely studied benchmark dataset that contains 20 subjects performing approximately 30 hours of daily activities [67], such as: `running`, `stretching`, `scrubbing`, `vacuuming`, `riding-escalator`, `brushing-teeth`, `walking`, `bicycling`, etc. The data was sampled at 70 Hz. We randomly chose 50% of the data as training data, and treated the rest as testing data.

In Figure 53, we show the training/testing error rates as our algorithm grows the data dictionary from the smallest size ($\mathbf{D_{0.17\%}}$) to the point where it is clear that our algorithm no longer improves. The *use-all-the-training-data* strawman [67][98][104], has a testing error rate of 0.237; however, we duplicate this error rate at $\mathbf{D_{1.1\%}}$ and obtain the significantly lower error rate of 0.152 at $\mathbf{D_{3.8\%}}$.

We also compare with the widely used rival approach discussed in Section 3.3.1 [67][104]. The *best* result is error rate of 0.314 achieved by C4.5 decision tree [114].

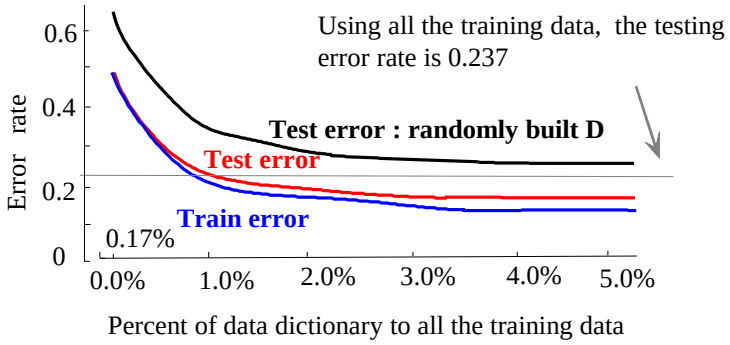


Figure 53: The classification error rates for D from $D_{0.17\%}$ to $D_{5.32\%}$ for [67]

In Figure 54, we show that using *Uniform Scaling* distance again beats *Euclidean* distance, obtaining a mere 0.091 testing error rate at $\mathbf{D_{4.6\%}}$. The threshold learned for **D** is 13.5, which rejects 6.3% of the testing queries [114].

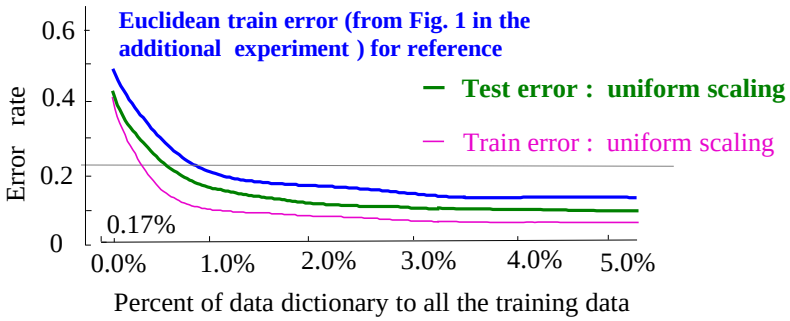


Figure 54: The *blue/brown(bold)* curves are train/test error rates obtained when we replaced *Euclidean* distance with *Uniform Scaling* distance. Note the other curves are taken from Figure 53 for comparison purposes

3.3.4. Speed Up The Search Using Complexity As Index

To evaluate the performance of our proposed indexing method, we simulate the classification of queries with varying arrival rates `k`. For the purpose of generality over all datasets, the arrival rates is modeled in equation (3) as a function of the number of all the subsequences `|subs|` in the data dictionary [105]. This is because using the concrete numerical values (e.g. the frequency of the data generated at 250Hz) may not always be meaningful or applicable, due to the wide variability in dataset characteristics: number of available exemplars, number of classes, etc.

$$\text{ArrivalTime}(k) = |subs| * k, \quad 0.1 \leq k \leq 1 \qquad (3)$$

For k = 1, the arrival rate of the streaming queries is exactly the time needed to calculate all the data dictionary, which is the same amount of time for the sequential search in Table 8. For k = 0.1, the arrival time of the streaming queries is only one-tenth the time of calculating the entire data dictionary.

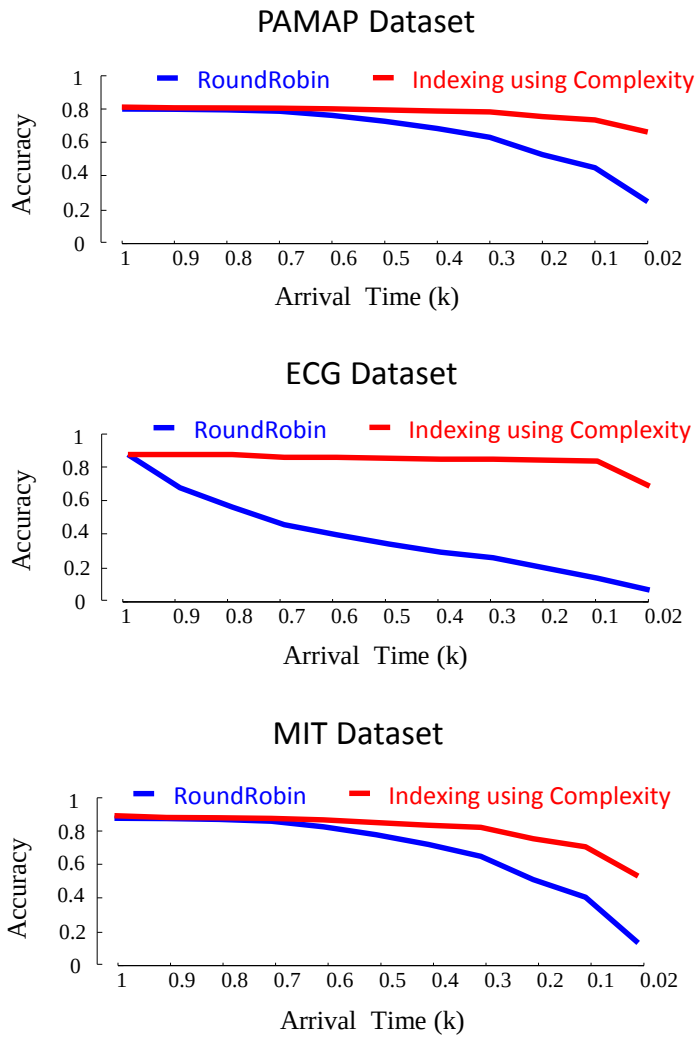


Figure 55: Classification accuracy of complexity as an index in the anytime classifier on constant query streams with different arrival rates for datasets in Section 3.3.1 to 3.3.3.

3.3.5. Related Work

There is significant literature on time series classification [67][71][75][93][103][106] both in the data mining community and beyond. However, almost all of these works make the three assumptions we relaxed in this work, and are thus orthogonal to the contributions here. Our algorithm can be seen as building a data dictionary of primitives for the very long streaming/continuous time series [99][100]. Other works have also done

this, such as [100], but they use significant amount of human effort to *hand-edit* the time series into patterns. In contrast, we build dictionaries automatically, with no human intervention.

In the following, we show the widely existence of the unrealistic assumptions in literature.

Many publications assume a large number of perfectly aligned atomic patterns are available. Our proposed concepts of *weakly-labeled* data and the data dictionary do not require the well-processed patterns. However, some researchers either derive *non-trivial* algorithms to extract such patterns from the original raw data or interpose the data generation process to produce such patterns. For example, [79] notes, "…*it is desirable to identify the boundaries of single gait cycles, or steps, and process them individually*..." However, the task of segmenting the data can be more difficult than classifying them. In [79], the authors also admit that, "*Finding gait cycle boundaries requires identification of landmark features in the waveforms that occur each cycle. Natural gait variation and differences between normal and pathological gait make this task non-trivial*."

The widely existed missing data phenomenon further increase the difficulty to the extraction of perfectly aligned atomic patterns. Our proposed *weakly-labeled* data can significantly mitigate this problem. However, researchers often "clean" datasets before publicly release them [82]. This is a noble idea, but one that perhaps shields the community from the realities of real-world deployment. Indeed, authors have been critiqued for releasing less ideal data. For example, authors in [88] criticize the UC-

Berkeley WARD dataset [111] by noting "*part of the sensed data is missed due to battery failure*".

There are many examples of human intervention of the data generation procedure to produce the perfectly aligned data. For example, [113] has a very rigid data generation process, by noting that "*When the subject was asked to perform a trial of one specific activity, an observer standing nearby marked the starting and ending points of the period of the activity performed.* " In addition, the subject was asked to repeat each activity multiple times. However, in the real-world scenario, the human does not perform the daily activities in this way.

Another widely existed unrealistic assumption is that the patterns to be classified are all of equal length [79][84][88][96][104]. The most famous and widely used time series benchmark is the UCR archive [84] . All the forty-seven datasets are well preprocessed and are of equal length. However, in reality, patterns can be of different lengths. For example, the human heart rate can be different. People can walk at different speed, etc. Authors in [88] observed, "*It is clearly visible that despite the normalization steps taken, there is still considerable variation within the same gesture type from the same person.*"

The assumption that exists in almost all the time series classification literature is that they assume every item to be classified belongs to exactly one of the well-defined classes [76][84][96][103]. Here we use a simple example to demonstrate the widely existence of this assumption. For example, in [96], authors report the classification result of seven daily activities, `lie`, `row`, `bike`, `sit/stand`, `run`, `nordic-walk`, `walk`. However, in reality, there are much more human activates than the mentioned above. For

example, `hand-shake`, `push-the-door`, etc. If the query with a concept other than the seven concepts, their classifier will still mistakenly report a class label. In our proposed framework, we use a rejection threshold to prevent this problem.

3.4 Conclusion and Future Work

We introduced a novel framework that requires only very weakly-labeled data and removes the unjustified assumptions made in virtually all time series classification research. We demonstrated over several large, real-world datasets that our method is significantly more accurate than several common strawman algorithms. Moreover, with less than one tenth of the original data kept in D, we are at least ten times faster at classification time.

Our algorithm has just one parameter, the length of queries. In our activity datasets, we simply used the original authors values [67][95], and for ECGs we used a cardiologist's suggestion. By changing these suggested values we empirically found that we are not sensitive to this parameter. Nevertheless in future work, we plan to learn it from the data.

Chapter 4:

Classification of Multi-Dimensional Streaming Time Series by Weighting Each Classifier's Track Record

Although there is extensive research on time series classification, the problem of multi-dimensional time series classification is still understudied. In this chapter we demonstrate a proposed framework with classification of multi-dimensional time series data. This chapter is organized as follows. We first introduce the notations and intuition behind our framework in Section 4.1. We will defer the discussion of related work in Section 4.2, when the reader's intuition for the domain has been developed. Section 4.3 explains how our novel voting framework works. In Section 4.4, we provide an extensive evaluation of our ideas with several real-world datasets from diverse domains. Finally, we offer conclusions and directions for future work in Section 4.5.

4.1 Notation and Background

In this section, we describe the definitions and intuition of our framework. We begin with the basic definitions.

4.1.1. Basic Time Series Definitions

We begin with the definition of a *time series*:

Definition 11: A time series $T = \{t_1,t_2,...,t_n\}$ is a continuous sequence of n real-valued numbers.

The recent ubiquity of inexpensive sensors, for example, in smartphones or medical devices, has led to greater interest in *multi-dimensional* time series [142][151][156]. We define *multi-dimensional time series* (*MDT*) as follows:

Definition 12: A *multi-dimensional time series MDT* = $\{T_1; T_2;\ldots T_m\}$consists of *m* time series T_i, which are synchronously recorded streams.

For convenience in this work, we refer to each dimension in *MDT* as a *stream* or a *sensor*, where there is no ambiguity.

There is near unanimous consensus that the *nearest neighbor (NN)* classifier is the best option for time series data [123][129][132]. Thus, this is our classifier of choice. In order to use the *nearest neighbor classifier* in classification of *MDT*, we must slightly generalize from ubiquitous *single* time version [123][129][132]. We define the *nearest neighbor classifier* in the classification of *MDT* as follows:

Definition 13: The *nearest neighbor classifier* for an *MDT* is an algorithm that for each dimension q_i in an incoming *MDT* query $q = \{q_1;q_2;\ldots q_m\}$ finds its nearest neighbor only in the corresponding dimension T_i from the *MDT* training data $\{T_1; T_2;\ldots T_m\}$. The class label of q is determined by a combination of the nearest neighbor results for q_i.

Hereafter, when we refer to a *classifier*, we mean a single *nearest neighbor classifier* operating on a *single* dimension in *MDT*.

As shown in Figure 56, the query q_i from a given dimension only finds its nearest neighbor in the respective dimension T_i in training data; the query q_i does not find its nearest neighbor in any other dimension T_j.

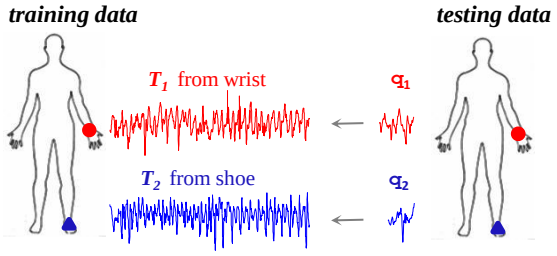


Figure 56: The red dot/blue triangle represent sensors mounted in wrist/shoe, respectively. *left*) A two dimensional time series (T_1, T_2), T_1 from a sensor on the wrist and T_2 from a sensor on the shoe. *right*) A query q with two dimensions (q_1 and q_2), will find their nearest neighbors in T_1 and T_2, respectively.

4.1.2. Supporting Confidence-Based Classification

As noted above, rather than using an approach that uses the **ALL**, **BEST,** or **SUB** streams of an *MDT*, we propose to evaluate and exploit the expertise of each data stream. In other words, for each time series stream in *MDT*, we have an individual nearest neighbor classifier, and a (dynamically determined) combination of classifier's predictions is used as the overall class prediction.

At query time, each classifier tells us not only *what* it predicts for the class label, but also how *confident* it is in its prediction. Our central claim in this work is that by judiciously considering these confidence-annotated predictions, we can outperform all the obvious rival methods. While similar ideas (*weighted voting* [121][127][131][160], *Bayesian classification* [121]) exist in the literature for general classification (cf. Section 4.2), the application and adaption to the unique structure of time series data we present in this work is novel.

Our technique opens several questions, the most immediate of which is how to learn each classifiers' expertise?

The expertise of each classifier could be labeled by domain experts. For example, a clinician may know that an ECG from electrodes placed on the right of the sternum (S_5) are generally better for recognizing *atrial flutter*, whereas data from the patient's back (V_7, V_8, V_9) tends to be better for detecting *myocardial infraction* [120][126]. However, experts are expensive. Thus, we will create a framework that automatically *learns* the expertise of each classifier directly from the training data. As our framework requires that each classifier must report a score indicating how confident it is for its predicted class label, we define *confidence score* as follows:

Definition 14: A *confidence score C* with range [0, 1] is a self-reported confidence of a classifier on its prediction result. Numbers closer to 1 indicate higher confidence in prediction.

Before we demonstrate how we learn and use the confidence scores in Section 4.3, we show an intuitive example to demonstrate that the expertise of classifiers *does* vary. In Figure 57, we show the confidence score of classifiers learned for various human activities (more details in Section 4.4) in a heavily cited benchmark dataset for human activity recognition [140].

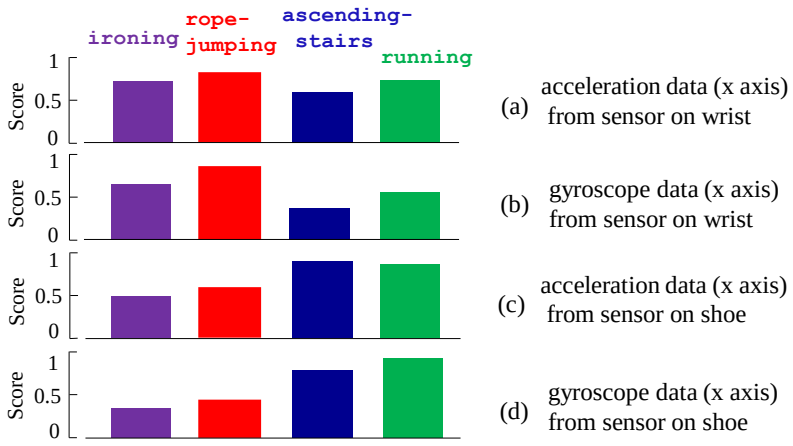


Figure 57: The performance of four classifiers (a), (b), (c), and (d) on four activities. In each classifier, the height of the bar is the confidence score for each activity.

Note that classifier (b) which tracks a sensor on the wrist has high confidence in the upper body activities `ironing` and `rope-jumping`[16], but has relatively low confidence in `ascending-stairs` and `running`, which are clearly lower body activities. Conversely, classifiers (c) and (d), which are embedded in the participant's shoes, have the opposite expertise.

If there are p concepts to be learned, then we must learn a *confidence score vector* $C\_vector = [C_1, C_2, \ldots C_p]$ for *each* stream in *MDT*. Each element C_i represents how confident the classifier is when predicting the i^{th} class label.

Accordingly, for an *MDT* comprised of m streams, our framework learns a *confidence score matrix* $C\_matrix = \{C\_vector_1; C\_vector_2; \ldots C\_vector_m\}$ with row number m and column number p for the m classifiers. This, in essence, is what Figure 57 illustrates.

[16] We classify `rope-jumping` as *upper body* because the participant may have variable footwork, skipping on left, right or both legs; however their wrist action has very low variability.

4.1.3. Supporting Distance-Based Classification

The confidence scores in Figure 57 are learned offline in *training* phase. However, as noted in the introduction, we have an additional observation we plan to exploit, and this observation requires adjustment of confidence in the *testing* (or *deployment*) stage. Our observation is that an individual stream classifier should not be confident predicting *any* class if the object being classified is significantly different than the exemplars encountered during training. This problem was hinted at in Figure 1 and was observed in nearly all of the case studies in Section 4.4. A common trivial reason for this occurring is that a battery dies on one sensor, and thus the time series to be classified is just a constant line. This effect is very commonly seen in medicine when one lead is unplugged or falls off the patient. Moreover, the sensor failure problem has been frequently observed in the literature. For example, a recent paper states: "…*part of the sensed data is missed due to battery failure*…" [159].

Furthermore, there are other possible reasons why the testing data might differ from the training data. If one of the concepts we learned with high confidence is `ascending-stairs`, we may find the new behaviors to be classified range from near identical time series patterns to more and more distorted patterns. This is because we may encounter data from a user that is tired, or wearing new shoes, or carrying heavy groceries, or encountering fresh snow etc. In these cases, even if the time series is still closer to `ascending-stairs` than any other class, the relevant classifier should signal a more tentative class prediction.

In Figure 58, we show a concrete example to demonstrate the importance of integrating the nearest neighbor distance with the confidence score. This is real-world data which we have slightly contrived for clarity. For simplicity, assume that there is an *MDT* with two dimensions. Further assume that at query time there is an incoming query `q` with two dimensions (q_1 and q_2). We want to determine the class membership of `q` using the confidence score approach.

Consider a case when we discover that among a dozen possible classes, q_1 and q_2 report that their nearest neighbors are different, say `running` and `rope-jumping`, respectively. (If they had agreed on a class label, then our prediction would have just been the agreed upon that label.) Given our observation about the confidence scores, we can break the tied vote by trusting the more confident classifier, which in this case was `running` with a confidence score 0.82.

However, as shown in Figure 58, this may not be the optimal decision. While q_1 is a little closer to `running` than q_2 is to `rope-jumping`, neither is particularly similar to its class prototypes. We simply do not have much experience with such objects. Nevertheless, if we take into account the learned distributions of nearest neighbor distances for the two classifiers, we find that the probability of being a true positive for q_2 is much higher than q_1. In Section 4.3.3, we formalize this visual intuition of how we adjust the prior knowledge – the confidence score − to a posterior probability by integrating the nearest neighbor distance as the new evidence. We discover that the prediction `rope-jumping` from the second classifier has a higher confidence score after this adjustment, which is the correct answer in this example.

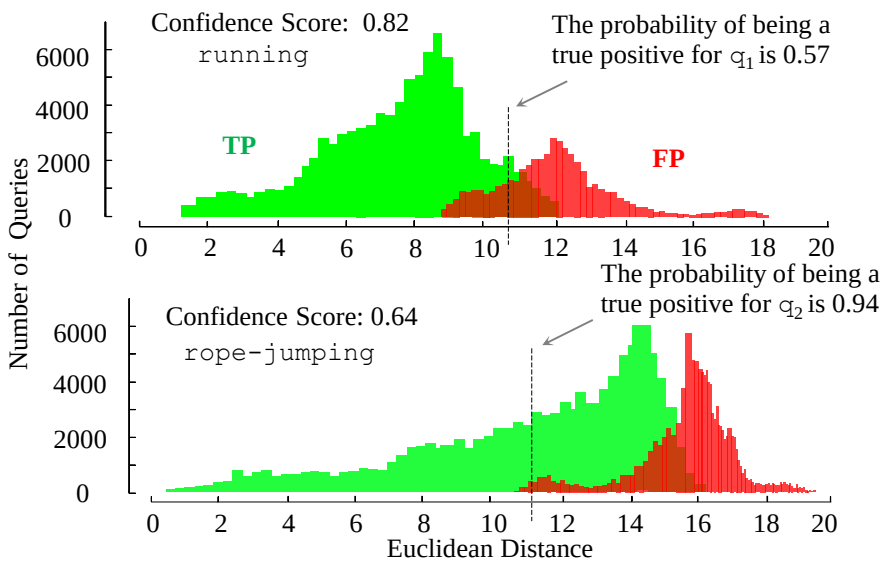


Figure 58: The distributions of nearest neighbor distances for true positives (green/left) and false positives (red/right) in the classification of activity running using data from wrist (top) and activity rope-jumping using data from shoe (bottom).

Note that the above observation only makes an overall difference in accuracy if there is variability in the distributions observed in each class. Empirically, we find that this is almost always the case for real-world problems. Some classes are intrinsically simple; for example there is only so much variability possible in say, `running`. However, some behaviors such as `ironing` are much more amiable to individual idiosyncrasies. Moreover, variability in equipment or clothing being ironed will also tend to produce distributions with greater means and larger standard deviations.

In summary, simply voting with the confidence score learned in the training phase may be sub-optimal, unless the testing data is *exactly* like the training data, an unlikely eventuality.

To take into account the above observation, we define the *adjusted* confidence score as follows:

Definition 15: The *adjusted* Confidence score (*adC)* with range [0, 1] is a score that subsumes the confidence score (c.f. Definition 16) by incorporating information about the distance between testing objects and the training objects as measured at query time.

In Section 4.3.3, we show how we adjust the confidence score in a principled way by combining the nearest neighbor distance at query time using Bayes theorem [121]. If the query is not similar to *any* class that the classifier learned, the *adC* for predicting the label of this query should be very low.

We use *ACV* as the abbreviation for our algorithm *Adjusted Confidence Vote*, which incorporates these observations.

As we have shown in Figure 58, the nearest neighbor distance distributions of true positives and false positives for each class play an important role in adjusting the confidence score at query time. We define *distributions of nearest neighbor distances* as:

Definition 17: The *distributions of nearest neighbor distances* (*DN*) are two distributions; one is the distribution for nearest neighbor distances of the true positives and another one is for the false positives. For each concept that a classifier learns during the training phase has two such *distributions*.

In Definition 18, we showed that our algorithm learns *C* in the training phase by evaluating the classification performances for each classifier. During this process, we also store the *distributions of nearest neighbor distances*. For each classifier, we have a vector of distributions $DN\_vector = [DN_1, DN_2, \ldots DN_p]$ with length p.

4.1.4. Allowing Real World Deployment

Recently, it has been noted that much of the literature on time series classification implicitly or explicitly makes unjustified assumptions that limit the applicability of the proposed algorithms to real-world scenarios [129][130]. These assumptions are:

Large amounts of perfectly aligned *atomic* patterns can be obtained [122][129][132]. That is to say, the algorithms assume they will only be given whole and complete heartbeats/gait cycles/atomic behaviors, with no extra spurious leading or trailing data.

The patterns are all of *equal length* [130][132][147][152]. For example, in the world's largest collection of time series datasets, the UCR classification archive, all forty-five time series datasets contain *only* equal-length data [132].

All patterns presented to the classifier will belong to one of two or more well-defined classes, that is to say, there is no possibility for the classifier to label an object as `unknown` [123][132].

These unrealistic assumptions are violated by most real-world datasets. In particular at least one assumption is violated by *all* five datasets we consider in Section 4.4. Thus, while we know a lot about the ability of various classification paradigms on the datasets found in the UCR archive [132], based on the hundred-plus research efforts that report results on it [152], we know a lot less about how well these ideas will perform in a real-world deployment.

Thus, while it is not strictly necessary to demonstrate our novel observations, in this work we will follow the lead of [129] and introduce our framework in a way that does not

make such unwarranted assumptions about the data. The next two definitions are required to remove these assumptions.

We define the *weakly-labeled training data* as follows:

Definition 19: *weakly-labeled training data* (*WT)* is a collection of the *weakly-labeled* time series annotated by behavior/state or some other mapping to the ground truth.

This is best understood by contrast to strongly-labeled training data (i.e. *all* of the UCR datasets [132]). Strongly-labeled data presents objects with explicitly labeled sections. For example, in the Kitchen Activity Dataset we consider in Section 4.4.5, someone has taken the effort to annotate the precise moment that the various atomic food preparation activities begin and end. In contrast, in *WT* data, we are given data labeled like this: "*in these two minutes of data there are some examples of* `chopping`." This is clearly a more realistic and scalable way to annotate data and our efforts are made with these more assumptions in mind.

There are two important properties of *WT* that we must consider and which are illustrated in Figure 59.*a*.

First, *WT* will generally contain *extraneous/irrelevant* sections. For example, when recoding ECG data, a section of recoding is clearly *extraneous* when the machine was not plugged in, as shown by the "flatline" in Figure 59.*a*. Similar phenomena occur in all the datasets we examined. Second, *WT* will almost certainly contain significant *redundancies*. Consider Figure 59.*a* again. Once we have a single normal heartbeat, say pattern $\mathbf{N_1}$ (**N**ormal beat), then there is little utility in adding additional examples of the

same type of ECG in the training data. Rather, what we should add into the training data are representative examples of other types of heartbeats, in this case, one example of the pattern **S** (**S**upraventricular Ectopic Atrial) and one example of the pattern $\mathbf{V_1}$ or $\mathbf{V_2}$ (Premature **V**entricular Contraction).

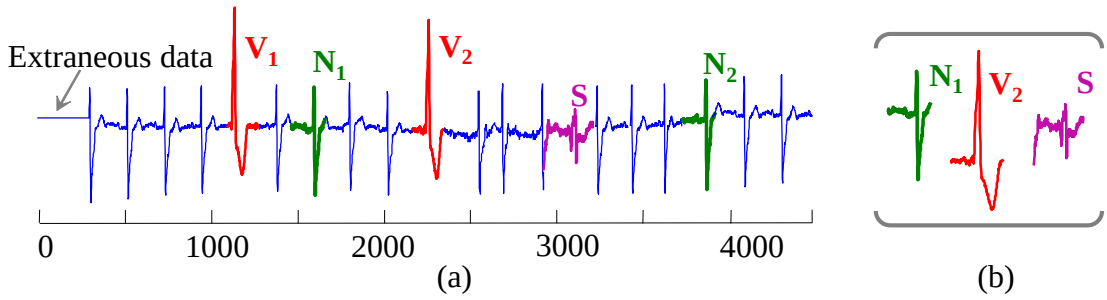


Figure 59: A snippet of BIDMC Congestive Heart Failure Database ECG, Record-03. (a) WT, which exhibits both extraneous and redundant data. There are two types of anomalous heartbeats (V, S) and normal beat (N) in WT. (b) A minimally redundant set of representative heartbeats (a data dictionary) could be used as training data.

Rather than having the redundant data in *WT*, we desire a smaller but smarter training subset that does not have the spurious data, while still covering the target concepts. For example, Figure 59.*b* consists of just one example of **N**, **V** and **S**. We define the minimally redundant training subset as a *data dictionary*:

Definition 20: A data dictionary *D* is a (potentially very small) "smart" subset of *WT*, while covering all the *p* concepts in *WT*.

Note that in our simple example in Figure 59.*b*, it happens to be the case that one example from each of {**N**, **S**, **V**} suffices to cover the concept space. This does not have to be the case; for example, the class **V** could be polymorphic and we may need to have multiple examples of it in order to represent its variability.

While *D* could be manually built by domain experts, again we note that human domain expertise is expensive. The framework in [129] demonstrates how to build *D* automatically using a simple data editing technique, which removes data redundancy while retaining just enough examples of the concept to cover the space of its "variability."

There are two important properties of *D*. First, the classification accuracy obtained from using just *D* is generally much higher than that obtained from using all *WT* [129]. This may be a little surprising, as we generally think *more is better* when it comes to data. Recall that *D* is designed so that it does not contain spurious data. If we have voluminous spurious data, then there is a high probability that some of it will be close to an exemplar from a different class, reducing classification accuracy.

The second important property of *D* is its size. In most real-world settings, *D* is a very small fraction of *WT*, perhaps only one-hundredth its size. This allows real-time deployment on resource limited devices (embedded devices, smartphones, etc. [119]).

For an *MDT* with *m* dimensions, our framework must learn *m* data dictionaries for the respective dimension.

4.2 Related Work

Since the adjusted confidence score is at the heart of our contribution, we will take the time to discuss its relationship with the related work.

4.2.1. Relationship to Ensemble Methods

We are finally in a position to clarify the relationship between our algorithm *ACV* and the ensemble methods that it superficially resembles, for example, Boosting or Bagging [117][127][131][145][160].

In brief, our approach is different from such ensemble methods in the sense that we do *not* generate redundant classifiers that later combine for prediction [160].

The common approach in the first step of Boosting and Bagging is that they both generate multiple base classifiers in order to produce diverse "views" of the data [160]. However, we do not generate classifiers. Instead, we perform the classic *nearest neighbor classifier* on each *single* data stream as an individual classifier. For example, the most famous algorithm in Boosting is AdaBoost [160]. In order to focus more on the training examples that are "difficult" to classify, AdaBoost iteratively generates different classifiers to focus on the training examples that are incorrectly classified. However, our *ACV* framework does not generate classifiers to adapt the data.

The last step of ensemble methods performs a combination of the votes from the base classifiers [160]. Our contribution of the novel voting scheme using *adC* was informed by this combination step. In particular, using *just C* is similar to the weight in weight voting. (There is still a difference between using just *C* and the weight in weighted voting, which we clarify in 4.2.2). However, we augment weighted voting by adjusting the weight with the similarity measure at query time.

In Section 4.4, we show using several large datasets from diverse domains that *ACV* framework beat the most popular voting methods: majority weight and weighted voting [160].

4.2.2. The Adjusted Confidence vs. the Weight in Weighted Voting

In general, our voting framework is similar to weighted voting [121][127][131][144][160]. Since *adC* is an augmentation of *C*, we first clarify the difference between *C* (Definition 14) and the "weight" in weighted voting.

In weighted voting, the key decision is how to choose the weight [160]. To obtain the best performance, the general intuition is that the weights should be chosen in proportion to the performance of individual classifiers [160]. Our confidence score *C* has a similar intuition in the sense that both of them are chosen based on the performance of the classifiers in the *training* phase. However, unlike the weights in weighted voting, which are chosen based on the *overall* performance of the classifier, *C* is calculated based on the performance of the classifier for each *individual* class in the classifier.

In other words, for all the classes considered by a classifier, there is a corresponding *C* for each class. As shown in Section 4.1.2, instead of having a single weight for each classifier as the weighted voting does, we have a $C\_vector = [C_1, C_2, \ldots C_p]$ for each classifier.

As *adC* is an augmentation of *C*, it can also be seen as an extension of the strategy of the weighted voting algorithm [121][127][131][160]. The modification lies in the fact that *adC* is the *posterior* probability by combining the new evidence (nearest neighbor distance at query time) and a *prior* knowledge (the confidence score) using Bayes

theorem [124]. In contrast, the weighted voting only uses *prior* knowledge, in particular the past performance of the classifiers [160].

As we will show in Table 12, our framework takes the predicted label with the highest posteriori probability, the highest sum of *adC*. The optimality of using the maximum posteriori estimation together with Naïve Bayes over other approaches has already been proven [115][124][158]. These optimality results require that we treat the multiple data streams as independent of each other. This may be an unrealistic assumption, but it has been shown that Naïve Bayes is surprisingly robust to violations of this assumption [124][158]. The experimental results in Section 4.4 will show that our *ACV* approach is more accurate and robust than all the rival methods.

4.3 Algorithms

In order to best explain our framework, we first explain how our classification model works *given* that the confidence scores of an *MDT C_matrix*, the distributions of nearest neighbor distances *DN_matrix*, and the data dictionaries *D_matrix* for *WT* have already been created. Later, in Section 4.3.2 and 4.3.3, we revisit the task of *learning* them.

4.3.1. Classification of Multi-Dimensional Time Series using the Adjusted Confidence Scores

For an incoming *m*-dimensional unlabeled object q, we classify each dimension with the corresponding classifier in *D_matrix* using the classic *one nearest neighbor* algorithm [132]. For each class p_j, we sum the *adC* of each classifier that assigned class p_j to query q. The class with the highest sum is returned as the class prediction for q.

In Table 12, we explain the algorithm in more detail. We begin in line 1 by initializing all of the *m adC* from the *m* classifiers to zeros. In line 3, we calculate the nearest neighbor for each dimension of `q` with the corresponding classifier in *D_matrix*. To be clear, each dimension is considered completely independently of all others.

The function `One_NN_search` is simply the classic one nearest neighbor algorithm discussed in Definition 13 [132]. We omit details of the function `One_NN_search`, as it is well known [123][151]. Note that while the distance measure used in line 3 could be any measure [123], we only consider the Euclidean distance， as it has been shown to be an extremely competitive measure [129][132][154]. In line 4, the algorithm computes the adjusted confidence score calculated by equation (1) in Section 4.3.3.

Table 12: Adjusted Confidence Classification Algorithm

Input	***C_matrix***, a confidence score matrix that contains *p* columns and *m* rows(from *m* classifiers) ***DN_matrix***, distributions of nearest neighbor distances ***D_matrix***, a matrix that has *m* data dictionaries ***q***, a query with *m* dimensions
Output	***The class membership of*** *q* ***score***, the total confidence for the prediction

1
2
3
 	
4
5
6
	adC_vector ← zeros(1,m);
	for i ← 1 to m
	[NN_labels(i),NN_dist] ← One_NN_search(q{i},*D_matrix*{i});
	// NN_labels is a vector with m elements for the m classifiers
	adC_vector(m) ← calculate_adC(NN_labels(i),NN_dist);
	endfor
	[class_label,score]←
	class with highest sum(adC vector,NN labels);

Note that if we only use the confidence score retrieved from *C_matrix* without any adjustment, *ACV* degenerates to the weighted voting algorithm scheme [160]. (Although to our knowledge, this has never been done for *time series* before.) However, as we

argued in Section 4.1, we need to augment this confidence score with the observed nearest neighbor distances.

We take the class label that has the highest sum of adjusted confidence scores.

Having demonstrated how the classification model works in conjunction with *C_matrix* and *DN_matrix*, we are now in a position to illustrate how to *learn* them.

4.3.2. Learning the Confidence Score

We learn the confidence scores by evaluating the classification performance for each classifier during the training phase. As a byproduct of this, we also obtain *DN* for every class in all the classifiers, which we use to calculate the probability of being a true positive given the nearest neighbor distance at query time. To be concrete, for each class, we use the precision [150] of the classification as the confidence score.

In Table 13, we show how we learn *C_matrix* and *DN_matrix*. Note that we randomly split the *weakly-labeled* training data into two parts. We learn the data dictionaries from one fold using the framework in [129] and treat another fold as holdout data.

We first randomly sample a large number of queries from the holdout data in line 1. From lines 2 to 10, we calculate the *C_vector* and *DN_vector* for each classifier. In line 3, we calculate the classification result for queries in the i^{th} classifier. Then the algorithm retrieves the *DN* (i.e. `NN_true` and `NN_false`) from `NN_dists` in lines 5 to 6. Line 7 shows that the algorithm adds *DN* to *DN_matrix*. In line 8 the algorithm calculates the precision for the classification result as the confidence score for the j^{th} class in the i^{th} classifier.

Table 13: Learning the Confidence Score

Input	***D_matrix***, The number of classes in each *D* is *p*; ***Holdout_WT***, holdout data in the *WT*
Output	***C_matrix***, confidence score matrix contains *m* confidence vectors for the *m* classifiers; ***DN_matrix***, the distributions *NN* distances
1	qs ← a large number of multi-dimensional queries randomly sampled from *Holdout_WT*
2	***for*** i ← 1 to *m*
3	[NN_labels,NN_dists]← One_NN_search(qs(i),MD(i)); // perform classification for the i^{th} classifier
4	***for*** j ← 1 to *p*
5	NN_true ← NN_dists for true positives in j^{th} class
6	NN_false ← NN_dists for false positives in j^{th} class // DN is NN_true and NN_false
7	DN_matrix(i,j) ← [NN_true,NN_false];
8	C_matrix(i,j)← calculate_precision(NN_true,NN_false)
9	***endfor***
10	***endfor***

In the next section, we illustrate how we adjust the confidence score at query time by combining the nearest neighbor distance and the confidence score in a principled manner.

4.3.3. Learning the Adjusted Confidence Score

The adjusted confidence score is the confidence score augmented by integrating the nearest neighbor distance at query time.

The Bayes theorem is the optimal model to learn the adjusted confidence score [124]. This is because the adjusted confidence score is the posterior probability by combining the new evidence (nearest neighbor distance at query time) and the prior knowledge (the confidence score). We denote the following:

pl : predicted nearest neighbor label

tl: true label

dist: the nearest neighbor distance calculated at query time

The adjusted confidence score is calculated as follows:

$$P(pl = tl \mid dist) = \frac{P(dist \mid pl = tl) \times P(pl = tl)}{P(dist)}$$

$$= \frac{P(dist \mid pl = tl) \times P(pl = tl)}{P(dist \mid pl = tl) \times P(pl = tl) + P(dist \mid pl \neq tl) \times P(pl \neq tl)} \quad (1)$$

In the above equation, $P(pl = tl)$ is the confidence score that we have learned using algorithm in Table 13. We can easily calculate $P(dist \mid pl = tl)$ given *DN* and the *dist* with density estimation.

4.4 Experiments

We begin by stating our experimental philosophy. To ensure that our experiments are easily reproducible, we have built a website which contains all the datasets and code [161]. In addition, this webpage contains further experiments which are omitted here for brevity.

Before listing the seven straw men that we compare to, we note that in addition we have compared our approach with many other widely-used rival classification frameworks, in particular SVM, boosted decision trees and the C4.5 decision tree [140][157][160]. The best result among these is achieved by C4.5 decision tree; however it is still not competitive with results produced by our algorithm *ACV*. Thus for clarity and brevity we relegate these results to our website [161]. We do not claim this as a novel finding, the superiority of nearest neighbor methods over eager leaning methods for time series has been noted by many others [123][152].

We test on five datasets (*plus* another three we relegated to [161]), which we believe is the largest set of *MDT* datasets ever considered in a single work. In particular, more

than 90% of the papers on this problem test on exactly *one* dataset [128][130][140][149] [156][159].

For the purpose of comparison, we list the seven straw men we use. Note that each straw man has been used in at least one recent paper. We begin by explaining **ALL**, **BEST**, and S**UB** in more detail.

For **ALL**, we calculate the sum of the distance[17] between query q with *m* dimensions and the *m* classifiers and then find the one with the minimum distance as the nearest neighbor of q.

For **BEST**, at query time, we use only the *one* classifier that has the best performance in the training phase [149].

For SUB, in the training phase, we perform a heuristic greedy search over all the classifiers until the accuracy starts to decrease [128][133][146][155].

In addition to **ALL**, **BEST**, and **SUB**, there are four other obvious rival approaches in the literature that we need to compare:

Minimum Distance Vote: choose the class label of the classifier that reported the minimum distance among the nearest distances from all the classifiers [151].

Majority Vote: choose the most commonly predicted class label [160]. (Technically, this is a "plurality" and not a "majority," but we will use the common term).

Random Vote: at classification time, randomly choose a classifier and take its class prediction [160].

[17] We considered other variants, including summing the *squared* distances, etc. Our chosen variant was empirically the best method that used all dimensions.

Weighted Vote: choose the class label with the highest sum of the weights from the classifiers that agreed on that class label. The weight is learned purely based on the past performance of the classifiers on the training data [160].

4.4.1. Physical Activity Data

We consider a physical activity dataset containing 36 axis synchronous measurements from three Inertial Measurement Units (IMUs) located on the wrist, chest and ankle. This dataset has eight subjects performing activities such as: `ironing`, `rope-jumping`, `running`, `folding laundry`, `ascending-stairs`, etc [140]. Approximately eight hours of data at 110Hz was collected.

We performed the following experimental procedure. We randomly chose 40% of the dataset as training data, and treat the rest as testing data. A data dictionary matrix *D_matrix* that contains less than ten percent of all the training data is learned using the framework in [129].

Note that in all of our case studies, our experiment are *subject independent* evaluation, which is considered much harder than subject *dependent* evaluation [136] [140][149].

As shown in Table 14, our *ACV* approach achieved a classification error rate of 0.05. In contrast, the original authors of the data reported an overall classification error rate at 0.10 [140][149]. While these two results are not exactly commensurate, the evaluation procedure in [140][149] would be expected to produce *higher* accuracy based on their split sizes. Their method extracts signal features from sliding windows and reports the

best result after testing the feature vectors with all the popular classification algorithms using Weka [149].

Table 14: Classification Results on the Physical Activity Data for ACV and Seven Straw Men

Algorithms	Accuracy: Original Data	Accuracy: Occluded Data
ALL [119][133]	0.19	0.16
BEST [136][140]	0.72	0.63
SUB [2][133][146]	0.78	0.64
Minimum NN dists[151]	0.59	0.58
Random[160]	0.51	0.47
Majority Vote [160]	0.84	0.76
Weighted Vote [160]	0.89	0.77
Adjusted Confidence Vote	0.95	0.94

Moreover, Table 14 shows that the *ACV* method beats all seven straw men by a significant margin.

Recall that the strongest motivation for our ideas is to produce a framework that is robust for missing (or "occluded") data. Our claim is that such missing data is very common, but researchers often "clean" datasets before publicly releasing them. This is a noble idea, but one that perhaps shields the community from the realities of real-world deployments. Indeed, authors have been critiqued for releasing less than idealized data; For example, authors in [159] criticize the UC-Berkeley WARD Dataset [156] by noting"part of the sensed data is missed due to battery failure".

While we have seen multiple real examples of occluded data, to allow systematic testing we must synthetically occlude data. Let us revisit this widely studied dataset [140] as an example (Later datasets had a similar treatment.) There are 36 data streams, arranged in 12 triplets. For example, there are x, y, z axes for the acceleration data, and roll, pitch, and yaw for gyroscope data. We perform our occlusion experiments by

simulating sensor failure of one triplet at a time. For each of the three streams, in a randomly chosen triple, we toss a fair coin to decide if we should replace it with either a straight line or a sine wave. We report the average performance by testing all the 12 cases. In Table 14, rightmost column, we show the classification result for these data occlusion experiments.

As we can observe, ACV also achieves the highest accuracy for occluded data. Among the seven straw men, the Majority Vote and Weighted Vote methods return competitive results in using original data. However, when it comes to data with occlusion, the performance of these two algorithms drops precipitously. This is because data in the testing phase is different from data used in the training phase. While only one tenth of the data is different (by definition), this is enough to make a drastic difference in their performance.

In contrast, our ACV approach is relatively robust for data with occlusion, since ACV carefully factors in the nearest neighbor distances in the testing phase.

Given the relatively poor performance of the five straw men on both the original and occluded data (shown in gray in Table 14), we omitted the results for these approaches in the rest of this work. Instead, we put the results of a complete comparison on the supporting webpage [161].

4.4.2. Avian Audio Data

Audio classification typically begins by extracting acoustic features such as Mel-Frequency Cepstral Coefficients (MFCCs) from audio signals [118][138]. MFCCs

represent the speech amplitude spectrum in a compact form by transforming the audio data into thirteen coefficients[18].

In most algorithms that use the MFCCs for speech recognition, researchers either use one coefficient or use all the coefficients [118][138][153]. As noted above, it is our claim that *both* these choices may result in poor performance. To see this, we consider two species, *East Brazilian Pygmy Owl* (*Glaucidium minutissimum*) and *Common Potoo* (*Nyctibius griseus*) as examples. As shown in Figure 60, it is clear that for the *Owl*, the patterns (***green***/bold) exhibited in the third and fourth coefficients (*red*) are much clearer than the ones in the second and fifth coefficients. While for *Potoo*, the patterns in the third and fourth coefficients seem random.

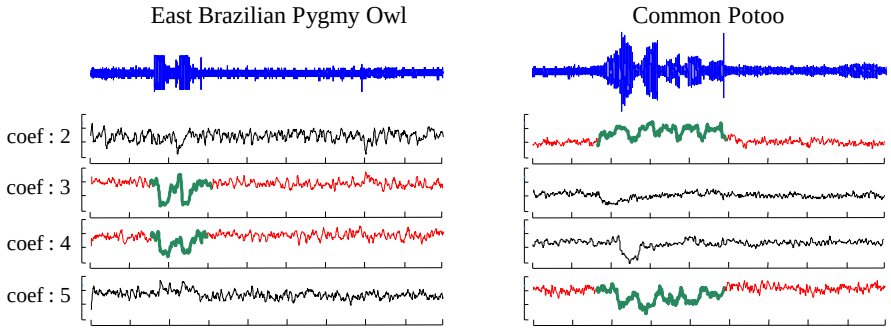


Figure 60. left) A snippet of sound spectrum and MFCCs from 2 to 5 for the East Brazilian Pygmy Owl. right) A snippet of sound spectrum and MFCCs from 2 to 5 for the Common Potoo.

Clearly, it is not a trivial task to automatically identify which coefficients are most useful for which species, even for experienced avian bio-acousticians. Moreover, even within a single species, the bird calls in the testing phase may be subtly different from in

[18] Usually the top thirteen coefficients are used for audio analysis. The first coefficient is a normalized energy parameter, which is not used for speech recognition [138].

the training phase, as the inevitable background noise may affect different coefficients in different ways.

Thus, we see this domain as an ideal candidate for our ideas and treat the twelve coefficients as an *MDT*.

Xeno-canto is a large data pool of bird sound files with the aim of sharing bird sounds. Avian audio files are uploaded by volunteers from all over the world [153]. We randomly chose four hours of audio data from four species of birds to perform a classification experiment. The four species are *East Brazilian Pygmy Owl*, *Common Potoo*, *Dusky Capped Flycatcher* (*Myiarchus tuberculi*fer), and *Acadian Flycatcher* (*Empidonax virescens*). We have placed the original audio files and the extracted MFCCs time series on the supporting webpage [161].

In the bird sound datasets, we do not need to explicitly perform experiment with occlusion because of the natural variability of the bird sounds, recorded—in some cases—years and hundreds of miles apart [153].

Our dataset consists of approximately eighteen hundred bird calls. We randomly chose 40% of the data as training data and treated the rest as testing data. The classification accuracy using our *ACV* approach is 0.87, while for Majority Vote and Weighted Vote, the classification accuracy is 0.66 and 0.79, respectively.

4.4.3. Recognition of Cricket Umpire Signals

Cricket is a very popular game in British Commonwealth countries. An umpire signals different events in the game to a distant scorer/book-keeper. The signals are communicated with motions of the hands. For example, `No Ball` is signaled by

touching each shoulder with the opposite hand. A complete list of signals can be found in [137].

The dataset in [134] consists of four umpires performing twelve signals. There are four umpires performing each signal ten times. The data with frequency 184Hz was collected by placing two accelerometers on the wrists of the umpires. Each accelerometer has three synchronous measures for three axes (*x*, *y* and *z*). Thus, we have a six dimension *MDT* from the two accelerometers. Figure 61 shows the data for two example signals, `Six` and `Leg Bye`. To signal `Six`, the umpire raises both hands above his head. `Leg Bye` is signaled with a hand touching the umpire's raised knee three times.

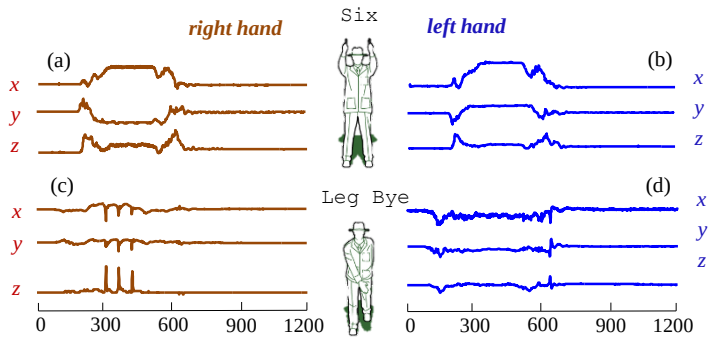


Figure 61: *x*, *y*, *z* acceleration data from right hand (*brown*) and left hand (*blue*) for two signals `Six` and `Leg Bye`.

We randomly chose 40% of the data as the training data and treated the rest as testing data. The classification results are shown in Table 15. As noted in Section 4.4.1, since the Majority Vote and Weighted Vote methods return the most competitive results among the seven straw men, we only list the results using these two straw men due to space limitations. However, we reiterate that we have put the full result on the supporting webpage [161].

To produce *real-world* occluded data, we had two experienced officials perform the twelve cricket signals under the same experimental conditions as in [134]. By contriving a battery failure, we arranged that one subject had a sensor failure on the left hand and the other subject had a sensor failure on the right hand. We added this data to the original data in [134].

As we can see in the result for occluded data in the rightmost column of Table 15, our *ACV* approach is significantly more robust to sensor failure than Majority Vote or Weighted Vote. Moreover, for original data [134], *ACV* once again achieves the highest accuracy.

Table 15: Classification Results on the Cricket Data

Algorithms	Accuracy Original Data	Accuracy Occluded Data
Majority Vote [160]	0.88	0.71
Weighted Vote [160]	0.92	0.78
Adjusted Confidence Vote	0.96	0.93

4.4.4. Gesture Recognition

Almost all modern smartphones are equipped with multiple sensors (i.e. acceleration sensors, gyroscopes, etc.). This has inspired dozens of research efforts on creating gesture recognizers for mobile devices [136].

The dataset provided in [136] is rapidly becoming a benchmark in this domain. The data was created by fifteen subjects wearing iPhones on their wrists to create six hand gestures as shown in Figure 62. Each participant provided each gesture fifteen times. There are six dimensions comprised of 3-axis acceleration data and 3-axis gyroscope data recorded at a frequency of 80Hz.

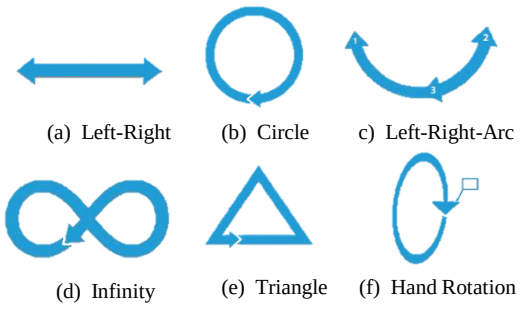


Figure 62. Visualization of the six gesture classes. This figure from [136] is used with permission.

We randomly chose 40% of the data as the training data and treated the rest as the testing data. Using the same method discussed in Section 4.4.1, we randomly choose half of the testing subjects for the occluded data experiment. The comparison in Table 16 shows that our *ACV* approach obtains the highest accuracy in both cases of using the original data and data with occlusion.

Table 16: Classification Results on the Gesture Data

Algorithms	Accuracy Original Data	Accuracy Occluded Data
Majority Vote [160]	0.86	0.67
Weighted Vote [160]	0.89	0.74
Adjusted Confidence Vote	0.97	0.93

4.4.5. Kitchen Activity Data

A recent European effort in assisting elderly people to live more independently [148] has investigated technology to support activities in the kitchen, including automatic guidance while cooking and cleaning. Sensors embedded into kitchen utensils provide continuous data streams while being used. This provides an ideal test bed to demonstrate our framework. The first major dataset released [125][148] has four Wii-remote instrumented utensils to collect acceleration data, as shown in Figure 63. Twenty subjects performed seven hours of a recipe for a mixed salad preparation [125]. There are eleven

classes, including `peeling`, `slicing`, `scraping`, `chopping`, etc. The data was recorded at a frequency of 40Hz.

(a) (b)

Figure 63. a) Modified Wii Remotes embedded in specially designed utensils. b) A subject is preparing salad. This figure is used with permission from [148].

Since in this dataset there are only three axes, we cannot use the same method with occluded data. Instead, we perform our occlusion experiments by simulating sensor failure of one axis at a time. By randomly choosing 40% data as training data and the rest as testing, we obtain the classification result as shown in Table 17. Once again, our *ACV* approach obtains the highest accuracy in both cases.

Table 17: Classification Results on the Kitchen Data

Algorithms	Accuracy Original Data	Accuracy Occluded Data
Majority Vote [160]	0.74	0.54
Weighted Vote [160]	0.84	0.76
Adjusted Confidence Vote	0.92	0.88

4.4.6. Robustness to Irrelevant Features

To demonstrate the robustness of our approach, we repeated the experiments above with an interesting modification. We added time series with no relation to the class into the data.

Let us consider the cricket dataset as an example. Originally, the dataset is an *MDT* with six dimensions. However, we added another six dimensions of random walk data to

the original data. To be clear, none of the explicit algorithms "know" which, if any, dimensions are irrelevant.

We repeated the experiment shown in Table 15 with the modified dataset. Both the Majority Vote and Weighted Vote are quite brittle to the additional irrelevant data, as their classification accuracies drop steeply to 0.69 and 0.78, respectively. However, our *ACV* approach obtains an accuracy of 0.95, barely affected by the irrelevant features. This is very important advantage when exploiting new domains in which we may have poor intuitions as to *which* features are useful.

4.5 Conclusion

Building on the general techniques of weighted voting [121][160] and Bayesian classification [121], and extending the techniques of "realistic assumptions" dictionary-based classification [129], we have introduced a novel voting framework for accurate classification of multi-dimensional time series. We demonstrated on several large, real-world datasets from diverse domains that our approach is significantly more accurate and robust than rival approaches. In particular, we have shown that our framework is very robust to missing data and irrelevant, a problem that frequently occurs in the real world [156]. Finally, we have given away *all* code and data to allow others to confirm, extend and use our work [161].

Chapter 5:

Conclusion

Time Series data is growing fast, especially in this 'Big Data' era. Although there is extensive research on time series data mining, in this dissertation we argue that most of the work is not as useful, since the datasets that they are dealing with and the way that they solved the problems are more like 'toy examples' compared to the much more complicate situation in the real-world scenario. We have observed the following two problems that widely exist in most of data mining research. First, parameters will hurt the potential of spreading the ideas in the research community. In a lot of works, there are usually several parameters to tune in the proposed method. We claim that the parameter turning can kill the usefulness of an algorithm and reduce the number of citations. Second, the prevalently existed assumptions about the data further limit their application to solve the real-world problem. We strive to mitigate the above two problems.

In this context, the contributions of this dissertation are as follows.

- We demonstrate a parameter free framework using MDL to discover the intrinsic features of the data. With the intrinsic cardinality and dimensionality of the time series, we can further understand the underlying meaning of the data, before consulting the domain experts. In addition, the intrinsic features can be used as dimensionality reduction and have huge applications in the various lower bounding techniques.

- We show a time series classification framework that has none of the prevalent assumptions. We propose to use the data editing technique to automatically build a data dictionary. In addition, our classification framework has the capability to say ‘I do not know’ at a certain point when classifying the incoming queries that does not belong to any concept in the training data. Our results show that a small fraction of all the data can achieve even better classification results than using all the data.
- We illustrate the limitations of the current multi-dimensional classification framework. Using **ALL**, **SUB**, **BEST** of the data cannot generate the optimal results. We propose a dynamically weighted multi-dimensional classification framework, which can smartly choose the weight of each data dimension. The results over extensive datasets from various domains show that our framework is more accurate and robust to the occluded data.

Bibliography

Chapter 2

[1] I. Assent, R. Krieger, F. Afschari, and T. Seidl. "The TS-Tree: Efficient Time Series Search and Retrieval." *EDBT*, 2008

[2] J.E. Bronson, J.Fei, J.M. Hofman, R.L.Gonzalez and C.H. Wiggins, "Learning Rates and States from Biophysical Time Series: A Bayesian Approach to Model Selection and Single-Molecule FRET Data," Biophysical Journal, vol 97, pp 3196-3205, 2009

[3] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. " *i*SAX 2.0: Indexing and Mining One Billion Time Series, "*International Conference on Data Mining*, 2010

[4] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly detection: A survey," *ACM* Comput. Surv. 41, 3, 2009

[5] R.A. Davis, T.C.M. Lee, and G. Rodriguez-Yam. " Break Detection for a Class of Nonlinear Time Series Models ," *J. of Time Series Analysis*, 29, 834-867, 2008

[6] S. De Rooij and P. Vitányi. "Approximating Rate-Distortion Graphs of Individual Data: Experiments in Lossy Compression and Denoising, " *IEEE Transactions on Computers*, 2006

[7] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. " Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures, " *VLDB*, pp. 1542-1552, 2008

[8] D.L. Donoho and I. M. Johnstone. "Ideal Spatial Adaptation via Wavelet Shrinkage." *Journal of Biometrika* 81, pp. 425-455, 1994

[9] S.C. Evans et al. " Microrna Target Detection and Analysis for Genes Related to Breast Cancer Using MDL Compress." *EURASIP J. Bioinform. Syst. Biol.*, pp. 1-16, 2007

[10] L. Firoiu and P. R. Cohen. " Segmenting Time Series with A Hybrid Neural Networks Hidden Markov Model. " *Proc. 8th Nat. Conf. Artif. Intell*, p.247, 2002

[11] D. García-López and H. Acosta-Mesa. "Discretization of Time Series Dataset with a Genetic Search." *MICAI*, pp. 201-212, 2009

[12] K. Goebel, B. Saha and A. Saxena, "A Comparsion of Three Data-Driven Techniques for Prognostics," *Failure prevention for system availability, 62th meeting of the MFPT Society*, pp119-131, 2008

[13] P.D. Grünwald, I.J. Myung, and M.A. Pitt, "Advances in Minimum Description Length: Theory and Applications ," *MIT Press*, 2005

[14] F.O. Heimes and BAE Systems, "Recurrent Neural Networks for Remaining Useful Life Estimation," *International Conference on Prognostics and Health Management*, 2008

[15] B.Hu, T. Rakthanmanon, Y. Hao, S. Evans, S. Lonardi and E. Keogh, "Discovering the Intrinsic Cardinality and Dimensionality of Time Series using MDL," *ICDM*, 2011

[16] International Business Machiness (IBM), "Harness the Power of Big Data," accessed on Nov 7, 2012.

[17] *public.dhe.ibm.com/common/ssi/ecm/en/imm14100usen/IMM14100USEN.PDF*.

[18] I. Jonyer, L. B. Holder, and D. J. Cook, " Attribute-Value Selection Based on Minimum Description Length," *International Conference on Artificial Intelligence*, 2004

[19] Ath. Kehagias, "A Hidden Markov Model Segmentation Procedure for Hydrological and Enviromental Time Series, " *Stochastic Environmental Resea*, 41, 2004

[20] E. Keogh and M. J. Pazzani, "A Simple Dimensionality Reduction Technique for Fast Similarity Search in Large Time Series Databases ," *PAKDD*, pp.122-133, 2000

[21] E. Keogh, S. Chu, D. Hart and M. Pazzani, " An Online Algorithm for Segmenting Time Series," KDD, 2011

[22] E. Keogh and S. Kasetty. "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration, " *Journal of Data Mining and Knowledge Discovery*, pp.349-371, 2003

[23] E. Keogh, Q. Zhu, B. Hu, Y. Hao , X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR Time Series Classification /Clustering Homepage: *www.cs.ucr.edu/~eamonn/time_series_data*/, 2006

[24] P. Kontkanen and P. Myllym. "MDL histogram density estimation," *Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*, 2007

[25] E. Linacre and B. Geerts, *www-das.uwyo.edu/~geerts/cwx/notes/chap15/global_temp.html*, Resources in atmospheric science, 2002. (Accessed 1th Dec 2011)

[26] M. Li and P. Vitanyi. "An Introduction to Kolmogorov Complexity and Its Applications," *2nd Ed, Springer*, 1997

[27] D. Lemire, "A Better Alternative to Piecewise Linear Time Series Segmentation," *SDM*, 2007

[28] J. Lin, E. Keogh, L. Wei, and S. Lonardi. "Experiencing SAX: A Novel Symbolic Representation of Time Series," *Journal of DMKD* 15, 2, pp. 107-144, 2007

[29] J. Lin, E. Keogh, S. Lonardi, and P. Patel. " Finding Motifs in Time Series". *In Proc. of 2nd Workshop on Temporal Data Mining*, 2002

[30] K. Malatesta, S. Beck, G. Menali, and E. Waagen. "The AAVSO Data Validation Project," *Journal of the American Association of Variable Star Observers (JAAVSO) 78*, pp. 31–44, 2005

[31] Y.I. Molkov, D. N. Mukhin, E. M. Loskutov, and A. M. Feigin, " Using the Minimum Description Length Principle for Global Reconstruction of Dynamic Systems from Noisy Time Series," *Phys. Rev. E* 80, 046207, 2009

[32] F. Mörchen and A. Ultsch. "Optimizing Time Series Discretization for Knowledge Discovery," *KDD*, 2005

[33] National Aeronautics and Space Administration, (Accessed Dec 1, 2011). *http://data.giss.nasa.gov/gistemp/*, *GISS surface temperature Analysis*, 2011

[34] T. Palpanas, M. Vlachos, E. Keogh, and D. Gunopulos. "Streaming Time Series Summarization Using User-Defined Amnesic Functions," *IEEE Trans. Knowl. Data Eng*. 20, 7, pp. 992-1006, 2008

[35] S. Papadimitriou, A. Gionis, P. Tsaparas, A. Väisänen , H. Mannila and C. Faloutsos . "Parameter-Free Spatial Data Mining using MDL," *ICDM*, 2005

[36] E.P.D. Pednault. "Some Experiments in Applying Inductive Inference Principles to Surface Reconstruction," *IJCAI*, pp. 1603-1609, 1989

[37] G. Picard, M. Fily, and H. Gallee. "Surface Melting Derived from Microwave Radiometers: A Climatic Indicator in Antarctica," *Annals of Glaciology*, 47, pp.29 – 34, 2007

[38] PHM data challenge competition, http://www.phmconf.orgjOCS/index.php/phm/2008/challenge, 2008.

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

[40] P. Protopapas, J. M. Giammarco, L. Faccioli, M. F. Struble, R. Dave, and C. Alcock. " Finding Outlier Light-Curves in Catalogs of Periodic Variable Stars," *Monthly Notices of the Royal Astronomical Society*, 369, pp. 677–696, 2006

[41] U. Rebbapragada, P. Protopapas, C. E. Brodley, and C. R. Alcock, "Finding Anomalous Periodic Time Series," *Machine Learning* 74, 3, pp. 281-313, 2009

[42] J. Rissanen. "Stochastic Complexity in Statistical Inquiry," World Scientific, Singapore, 1989

[43] J. Rissanen, T. Speed and B. Yu. "Density Estimation by Stochastic Complexity," *IEEE Trans. On Information Theory*, 38, 315-323, 1992

[44] S. Salvador and P. Chan. "Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms," *International Conference on Tools with Artificial Intelligence*, pp. 576-584, 2004

[45] Signal to Noise Ratio, *http://en.wikipedia.org/wiki/Signal-to-noise_ratio*

[46] W. Sarle, Donoho-Johnstone Benchmarks: Neural Net Results, *ftp://ftp.sas.com/pub/neural/dojo/dojo.html*, 1999

[47] D. Sart, A. Mueen, W. Najjar, V. Niennattrakul, and E. Keogh. "Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs," *IEEE International Conference on Data Mining*, pp. 1001- 1006, 2010

[48] US Environmental Protection Agency, *www.epa.gov/climatechange/science/recenttc.html*, Climate Change Science, accessed on Dec 6, 2011

[49] G. Vachtsevanos, F.L. Lewis, M. Roemer, A. Hess and B. Wu, "Intelligent Fault Diagnosis and Prognosis for Engineering Systems," Wiley, 1st edition, 2006

[50] A. Vahdatpour and M. Sarrafzadeh. "Unsupervised Discovery of Abnormal Activity Occurrences in Multi-dimensional Time Series, with Applications in Wearable Systems," *SIAM International Conference on Data Mining*, 2010

[51] R. Vatauv, "The Impact of Motion Dimensionality and Bit Cardinality on the design of 3D Gesture Recognizers", International Journal of Human-Computer Studies, 2012

[52] N. Vereshchagin and P. Vitanyi. "Rate Distortion And Denoising Of Individual Data Using Kolmogorov Complexity," IEEE Trans. *Information Theory* 56, 7, pp. 3438–3454, 2010

[53] U. Vespier, A. Knobbe, S. Nijssen and J. Vanschoren, "MDL-Based Analysis of Time Series at Multiple Time-Scales," Lecture Notes in Computer Science (LNCS), Volume 7524, 2012

[54] vbFRET toolbox, *vbFRET.sourceforge.net*, accessed on Nov8,2012

[55] C.S. Wallace and D. M. Boulton. "An Information Measure for Classification," *Computer Journal* 11, 2, pp.185-194, 1968

[56] T. Wang, J. Yu, D. Siegel and J. Lee, "A Similarity-Based Prognostics Approach for Remaining Useful Life Estimation of Engineered Systems," *International Conference on Prognostics and Health Management*, 2008

[57] T. Wang and J. Lee, "On Performance Evaluation of Prognostics Algorithms," *Proceedings of MFPT*, pp 219-226, 2006

[58] D. Yankov, E. Keogh, and U. Rebbapragada. "Disk Aware Discord Discovery: Finding Unusual Time Series in Terabyte Sized Datasets," *Knowl. Inf. Syst*. 17, 2, pp. 241-262, 2008

[59] Q. Zhao, V. Hautamaki, and P. Franti. "Knee Point Detection in BIC for Detecting the Number of Clusters", *ACIVS*, 5259, pp. 664–673, 2008

[60] H.J. Zwally and P. Gloersen. "Passive Microwave Images of the Polar Regions and Research Applications", *Polar Records* 18, pp. 431-450, 1977

[61] 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

Chapter 3

[62] S.L.G Andino, et al, Measuring the complexity of time series: an application to neurophysiological signals. Human Brain Mapping, 11(1), pages 46-57, 2000

[63] K. Aspelin, Establishing Pedestrian Walking Speeds. Portland State University. www.usroads.com/journals/p/rej/9710/re971001.htm, retrieved 2009-08-24.

[64] W. Aziz and M. Arif, Complexity analysis of stride interval time series by threshold dependent symbolic entropy, EJAP, 98 (1), pages 30-40, 2006

[65] G. Batista, E. Keogh, A, Mafra-Neto and E. Rowton, Sensors and Software to Allow Computational Entomology, an Emerging Application of Data Mining, SIGKDD demo, 2011.

[66] G. Batista, X. Wang and E. Keogh, A Complexity-Invariant Distance Measure for Time Series, SDM, 2011.

[67] L. Bao and S.S. Intille, Acitivity Recognition from User-Annotated Acceleration Data, In Proc' of the 2nd International Conference on Pervasive Computing, pages1-17, 2004.

[68] The BIDMC Congestive Heart Failure Database, *www.physionet.org/physiobank/database/chfdb/*

[69] G.A. Cavagna, N.C. Heglund and C.R. Taylor, *Mechanical work in terrestrial locomotion: two basic mechanisms for minimizing energy expenditure*, Journal of Physiology 233(5): R243-R261, 1977.

[70] P.de Chazal, M. O'Dwyer, and R. B. Reilly, *Automatic classification of ECG heartbeats using ECG morphology and heartbeat interval features*, IEEE Trans. Biomed. Eng., vol. 51, pp. 1196-06, Jul.2004.

[71] L. Chen, M. T. Özsu and V.Oria, *Robust and fast similarity search for moving object trajectories* , In Proc' of the ACM SIGMOD, 2005.

[72] CMU Graphics Lab Motion Capture Database, *mocap.cs.cmu.edu/*, retrieved 2012-04-24.

[73] Electrocardiography,*en.wikipedia.org/wiki/Electrocardiography*.

[74] M. Faezipour, A. Saeed, S. Bulusu, M. Nourani, H. Minn and L. Tamil, *A patient-adaptive profiling scheme for ECG beat classification*. IEEE Transactions on Information Technology in Biomedicine 14(5), p1153-1165, 2010.

[75] D. Gafurov, K. Helkala and T. Søndrol, *Biometric Gait Authentication Using Accelerometer Sensor*, Journal of Computers , (1) 6, 2006.

[76] D. Gafurov and E. Snekkenes, *Towards Understanding the Uniqueness of Gait Biometric*, 8th IEEE International Conference on Automatic Face & Gesture Recognition, 2008.

[77] J.Grass and S. Zilberstein, Anytime Algorithm Development Tools. Technical Report. UMI Order Number:

[78] UM-CS-1995-094., University of Massachusetts.

[79] M.A. Hanson, H.C. Powell Jr, A.T. Barth, J. Lach, M.B.C, Brown, *Neural Network Gait Classification for On-Body Inerital Sensors*, In Proc' of the 2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks, 2009.

[80] Y. Hao, Y. Chen, J. Zakaria, B. Hu, T. Rakthanmanon and E. Keogh, *Towards Never-Ending Learning from Time Series Streams*, SIGKDD, 2013.

[81] B. Hu, Y. Chen and E. Keogh, *Time Series Classification under More Realistic Assumptions*, SDM, 2011

[82] B. Hu, Y. Chen, J. Zakaria, L.Ulanova and E. Keogh, *Classification of Multi-Dimensional Streaming Time Series by Weighting each Classifier's Track Record*, ICDM, 2013

[83] B. Hu, T. R Rakthanmanon, Y. Hao, S. Evans, S. Lonardi, and E. Keogh, *Discovering the Intrinsic Cardinality and Dimensionality of Time Series using MDL*, ICDM, 2011.

[84] E. Keogh, Q. Zhu, B. Hu, Y. Hao , X. Xi, L. Wei, and C. A.

[85] Ratanamahatana. The UCR Time Series Classification/Clustering Homepage: *www.cs.ucr.edu/~eamonn/time_series_data/,* 2006.

[86] E. Keogh, S. Lonardi and C. Ratanamahatana, *Towards Parameter-Free Data Mining* , In Proc' of the tenth ACM SIGKDD, 2004.

[87] E. Keogh, T. Palpanas, V.B. Zordan, D. Gunopulos and M. Cardle, *Indexing Large Human-Motion Databases*, VLDB, 2004.

[88] P. Koch, W. Konen and K. Hein, *Gesture Recognition on Few Training Data using Slow Feature Analysis and Parametric Bootstrap* , IJCNN, 2010.

[89] M. Li, P. Vitanyi, *An introduction to Kolmogorov complexity and its applications*, Second Edition, Springer Verlag, 1997

[90] J. Lester, T. Choudhury, N. Kern, G. Borriello and B. Hannaford, *A Hybrid Discriminative/Generative Approach for Modeling Human Activities*, IJCAI, 2005.

[91] J. Liu, K. Yu, Y. Zhang and Y. Huang, *Training Conditional Random Fields Using Transfer Learning for Gesture Recognition*, ICDM,2010

[92] T.A. McMahon, G.C. Cheng, *The mechanics of running : How does stiffness couple with speed*, Journal of Biomechanics, Vol 23, 1990.

[93] M. Morse and J.M. Patel, *An Efficient and Accurate Method for Evaluating Time Series Similarity*, Proc SIGMOD, 2007.

[94] V. Niennattrakul, E. Keogh and C.A. Ratanamahatana, *Data Editing Techniques to Allow the Application of Distance-Based Outlier Detection to Streams,* ICDM, 2010.

[95] PAMAP, Physical Activity Monitoring for Aging People, *www.pamap.org/demo.html* , retrieved 2012-05-12.

[96] J. Pärkkä, M. Ermes, P. Korpipää, J. Mäntyjärvi, J. Peltola, and I.Korhonen, *Activity classification using realistic data from wearable sensors*, IEEE Trans. Inf. Tech. Biomed., vol. 10, pp. 119-28, 2006.

[97] E. Pekalska, R.P.W. Duin and P. Paclík, *Prototype selection for dissimilarity-based classifiers*, Pattern Recognition, 39, 2006.

[98] C.Pham, T. Plötz, P. Olivier, *A dynamic time warping approach to real-time activity recognition for food preparation*, In Proc' of the First international joint conference on Ambient intelligence, 2010.

[99] M. Raptis, D. Kirovski, and H. Hoppes, *Real-Time Classification of Dance Gestures from Skeleton Animation*, In Proc'of the ACM SIGGRAPH symposium on Computer animation, 2011.

[100] M. Raptis, K. Wnuk, and S. Soatto, *Flexible Dictionaries for Action Recognition*, In Proc' of the 1st International Workshop on Machine Learning for Vision-based Motion Analysis, 2008.

[101] T.Rakthanmanon, E. Keogh, S. Lonardi, and S. Evans. *Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data*. ICDM 2011.

[102] C.A. Ratanamahatana (2012). Personal communcation. May 2012.

[103] C.A. Ratanamahatana and E. Keogh, *Making Time-series Classification More Accurate Using Learned Constraints*, SDM, 2004.

[104] A. Reiss and D. Stricker, *Introducing a Modular Activity Monitoring System*, 33th International EMBC, 2011.

[105] J.Shieh and E. Keogh, *Polishing the Right Apple:Anytime Classification also Benefits Data Streams with Constant Arrival Times*, ICDM, 2010.

[106] J. Song and D. Kim, *Simultaneous Gesture Segmentation and Recognition based on Forward Spotting Accumulative HMM*, In Proc' of the 18th ICPR, 2006.

[107] K. Ueno, X. Xi, E. Keogh and D. Lee, *Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining*, ICDM, 2010.

[108] J. Usabiaga, G. Bebis, A. Erol, M. Nicolescu, *Recognizing simple human actions using 3D head movement*, Computational Intelligence, 23(4), 2007.

[109] R.D. Vatavu, *The Effect of Sampling Rate on the Performance of Template-based Gesture Recognizers*, Proc of ICMI, 2011.

[110] L. Ye, X. Wang, E. Keogh and A. Mafra-Neto, *Autocannibalistic and Anyspace Indexing Algorithms with Applications to Sensor Data Mining*, SDM, 2009.

[111] A.Y. Yang, A. Giani, R. Giannatonio, K. Gilani, etc. *Distributed Human Action Recognition via Wearable Motion Sensor Networks*, *www.eecs.berkeley.edu/~yang/software/WAR/index.html*

[112] S. Zilberstein and S. Russell, Approximate reasoning using anytime algorithms. In Imprecise and Approximate Computation. Kluwer Academic Publishers, 1995

[113] M. Zhang, A.A. Sawchuk, *USC-HAD: A Daily Activity Recognition using Wearable Sensors*. ACM International Conference on Ubiquitous Computing (UbiComp) Workshop on Situation, Activity and Goal Awareness(SAGAware), 2012

[114] Project URL: sites.google.com/site/sdm13realistic/

Chapter 4

[115] J. Aldrich, R.A. Fisher and the making of maximum likelihood 1912-1922, Statistical Science, 12(3), 1922.

[116] P. Bartlett, Y. Freund, W. Lee and R. Schapire, Boosting the Margin: A New Explanation for the Effective of Voting methods, The Annals of Statistics, vol(26),1998.

[117] E. Bauer and R. Kohavi, An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting and Variants, Journal of Machine Learning, 1998.

[118] F. Briggs, R. Raich and X. Fern, Audio Classification of Bird Species: a Statistical Manifold Approach, ICDM, 2009.

[119] L. Bao and S.S. Intille, Activity Recognition from User-Annotated Acceleration Data, 2nd International Conference on Pervasive Computing, 2004.

[120] E. Braunwald, Heart Disease: A Textbook of Cardiovascular Medicine, Ninth Edition, 2011.

[121] C.M. Bishop, M. Svensén, Bayesian Hierarchical Mixtures of Experts, Procs of 19th Conference on Uncertainty in Artificial Intelligence, 2003.

[122] Y. Chen, B. Hu, E. Keogh and G. E. P.A Batista, DTW-D, Time Series Semi-Supervised Learning from a Single Example, KDD, 2013

[123] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang and E. Keogh, Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures, PVLDB 1(2): 1542-1552 , 2008.

[124] P. Domingos and M. Pazzani, Beyond Independence: Condition for the Optimality of thhe Simple Bayesian Classifier, Machine Learning, vol(29),p(103-137), 1997.

[125] Digital Interaction at Culture Lab, *di.ncl.ac.uk/publicweb/AmbientKitchen/*, accessed on Jan 9, 2013.

[126] Electrocardiography, *en.wikipedia.org/wiki/Electrocardiography*

[127] Y. Freund and R. Schapire, A Short Introduction to Boosting, Journal of Japanese Society for Artificial Intelligence, vol(14), 1999.

[128] S. Günnemann, I. Färber, K. Virochsiri, and T. Seidl, Subspace Correlation Clustering: Finding Locally Correlated Dimensions in Subspace Projections of the Data, KDD, 2012.

[129] B. Hu, Y. Chen and E. Keogh, Time Series Classification under More Realistic Assumptions, *SDM*, 2013.

[130] Y. Hu, S. Palreddy and W. Tompkins, A Patient-Adaptable ECG Beat Classifier using a Mixture of Experts Approach, IEEE Transactions on Biomedical Engineering, vol(44),2007.

[131] M. Jordan and R. Jacobs, Hierarchical Mixtures of Experts and the EM Algorithm, A.I.Memo No.1440, C.B.C.L.Memo. No.83, 1993.

[132] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei and C.A. Ratanamahatana. The UCR Time Series Classification/Clustering Homepage: www.cs..ucr.edu/~eamonn/time_series_data/, 2006.

[133] H. Kremer, S. Günnemann, A. Held and T. Seidl, Mining of Temporal Coherent Subspace Clusters in Multivariate Time Series Databases, *PAKDD*, 2012.

[134] M.H. Ko, G. West, S.Venkatesh and M. Kumar. Online context recognition in multisensory system using dynamic time warping. *In Intelligent Sensors, Sensor Networks and Information Processing Conference*, 2005.

[135] J. Kolter and M. Maloof, Dynamic Weighted Majority: A New Ensemble Method for Tracking Concept Drift, *ICDM*, 2003.

[136] S. Kratz and M. Rohs, A $3 Gesture Recognizer – Simple Gesture Recognition for Devices Equipped with 3D Acceleration Sensors, *IUI*, 2010.

[137] Lord's, The Home of Cricket, *www.lords.org/laws-and-spirit/laws-of-cricket/laws/*, accessed on Feb 5th, 2013.

[138] P. Mermelstein, Distance measures for speech recognition, psychological and instrumental, *Pattern Recognition and Artificial Intelligence*, 1976.

[139] M. Miller and A. Stoytchev, Hierarchical Voting Experts: An Unsupervised Algorithm for Hierarchical Sequence Segmentation, ICDL, 2008.

[140] PAMAP, Physical Activity Monitoring for Aging People,

[141] *www.pamap.org/demo.html*, retrieved 2012-05-12

[142] J. Pärkkä, M. Ermes, P. Korpipää, J. Mäntyjärvi, J. Peltola, and I. K Korhonen, Activity classification using realistic data from wearable sensors, *IEEE Trans. Inf. Tech. Biomed*., vol. 10, pp. 119-28, 2006.

[143] R.E. Schapire, and Y. Singer, Improved Boosting Algorithms using Confidence-rated Predictions, Journal of Machine Learning, 1999.

[144] W. Street and Y. Kim, A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification, KDD, 2001.

[145] D. Optiz and R. Maclin, Popular Ensemble Methods: An Empirical Study, Journal of Artificial Intelligence Research, vol(11),1999.

[146] D. Optitz, Feature Selection for Ensembles, AAAI, 1999

[147] M. Radovanović,A. Nanopoulos and M. Ivanović, Time Series Classification in Many Intrinsic Dimensions, *SDM*, 2010.

[148] C.Pham and P.Olivier, Slice & Dice: Recognizing Food Preparation Activities using Embedded Accelerometers, *Procs of the European Conference on Ambient Intelligence*, 2009.

[149] A. Reiss and D. Stricker, Introducing a Modular Activity Monitoring System, *33rd IEEE EMBS*, 2011.

[150] C.J. van Rijsbergen, Information Retrieval, London, GV, 2nd Edition, 1979, ISBN 0-408-70929-4

[151] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos and E. Keogh, Indexing Multi-Dimensional Time Series with Support for Multiple Distance Measures, *KDD*, 2003.

[152] X. Xi, E. Keogh, C. Shelton, L.Wei and C. Ratanamahatana, Fast Time Series Classification Using Numerosity Reduction, *ICML*, 2006.

[153] Xeno-canto, Sharing Bird Sounds from around the World, *www.xeno-canto.org/*, accessed on Feb 6, 2013.

[154] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. Keogh, Experimental comparison of representation methods and distance measures for time series data. *DMKD*, vol26(2), 2013.

[155] H. Yoon, K. Yang, C. Shahabi, Feature Subset Selection and Feature Ranking for Multivariate Time Series, *IEEE Trans. Knowl. Data Eng.* 17(9): 1186-1198 , 2005.

[156] A.Yang, A. Giani, R. Giannatonio, K. Gilani, Distributed Human Action Recognition via Wearable Motion Sensor Networks, *Journal of Ambient Intelligence and Smart Environments*, 2009.

[157] J.Yin and Q. Yang, Integrating Hidden Markov Models and Spectral Analysis for Sensory Time Series Clustering, *ICDM*, 2005.

[158] H. Zhang, The Optimality of Naïve Bayes, *AAAI, FLAIRS Conference*, 2004.

[159] M. Zhang and A.A. Sawchuk, *USC-HAD:* A Daily Activity Dataset for Ubiquitous Activity Recognition Using Wearable Sensors, *UbiComp*, 2012.

[160] Z. Zhou, Ensemble Methods: Foundations and Algorithm, Chapman and Hall/CRC, 1st edition, 2012.

[161] Project webpage : *sites.google.com/site/kddmdtbing/*