UNIVERSITY OF CALIFORNIA
RIVERSIDE

Meaningful Rule Discovery and Adaptive Classification of Multi-Dimensional Time Series Data

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

Doctor of Philosophy

in

Computer Science

by

Mohammad Shokoohi-Yekta

March 2015

Dissertation Committee:

Dr. Eamonn Keogh, Chairperson
Dr. Rajiv Gupta
Dr. Marek Chrobak
Dr. Vagelis Hristidis

The Dissertation of Mohammad Shokoohi-Yekta 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. During my two and a half year research under his supervision, Dr. Keogh taught me how to do a good research, encouraged me to find interesting problems and solve them, and helped me with valuable suggestions. His brilliant research philosophy and precious insightful advice have given me strength throughout my PhD stage and will continue to benefit my entire research life. It is my great fortune to have him as my advisor. I truly believe he is the most knowledgeable person in his field and most supportive person I have ever seen.

I would also like to thank Dr. Rajiv Gupta, Dr. Marek Chrobak and Dr. Vagelis Hristidis who are my committee members, for their generous support and valuable comments; Dr. Khaleel Razak for being committee member in my oral-qualification exam. Also I would like to thank Dr. Rajiv Gupta for recommending me as the best TA which led to the Outstanding Teaching Assistant award in 2012.

My gratitude also goes to my colleagues in the data mining lab at UCR (names in random order), who offered me valuable help and friendship: Thanawin (Art) Rakthanmanon, Bilson Campana, Jesin Zakaria, Bing Hu, Yanping Chen, Nurjahan Begum, Liudmila Ulanova, Reza Rawassizadeh, Yan Zhu and Yuan Hao.

Moreover, I'm very grateful for all the financial support I received while I attended UCR: Dean's Distinguished Fellowship, GAANN Fellowship, Dissertation Year Program Fellowship and SDM travel award.

Finally, I would like to express my sincere gratitude to my family, especially to my parents for their constant love, patience and support. Without them, I could not have made this achievement; thanks to my wife, Ameneh Shakeri, for her constant support and inspirations; and to all my family members for always supporting me. I would also like to thank all my friends at UCR for being part of this wonderful journey.

DEDICATION

I dedicate my dissertation work to my wife and my lovely family. A special feeling of gratitude to my loving wife, Ameneh whom we recently got married, and my beloved parents, Mohsen and Zahra whose words of encouragement and push for tenacity ring in my ears. My sister Hamideh has never left my side and is very special. I'm so grateful and extremely happy to have Ameneh joining me along this journey. Even though we recently got married, she has been my best cheerleader for my doctorate defense.

ABSTRACT OF THE DISSERTATION

Meaningful Rule Discovery and Adaptive Classification of Multi-Dimensional Time Series Data

by

Mohammad Shokoohi-Yekta

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, March 2015
Dr. Eamonn Keogh, Chairperson

The ability to make predictions about future events is at the heart of much of science; so, it is not surprising that prediction has been a topic of great interest in the data mining community for the last decade. Most of the previous work has attempted to predict the future based on the current *value* of a stream. However, for many problems the actual values are irrelevant, whereas the *shape* of the current time series pattern may foretell the future. The handful of research efforts that consider this variant of the problem have met with limited success. In particular, it is now understood that most of these efforts allow the discovery of spurious rules. We believe the reason why rule discovery in real-valued time series has failed thus far is because most efforts have more or less indiscriminately applied the ideas of *symbolic* stream rule discovery to *real-valued* rule discovery. We feel that the lack of progress in this pursuit can be attributed to two related factors: the lack of effective algorithms for rule discovery in one dimensional time series, resulting in poor-quality and

random rules; less accurate classifiers built for multi-dimensional time series in order to make accurate predictions.

In recent years Dynamic Time Warping (DTW) has emerged as the distance measure of choice for virtually all time series data mining applications. For example, virtually all applications that process data from wearable devices use DTW as a core sub-routine. This is the result of significant progress in improving DTW's efficiency, together with multiple empirical studies showing that DTW-based classifiers at least equal (and generally surpass) the accuracy of all their rivals across dozens of datasets. Thus far, most of the research has considered only the one-dimensional case, with practitioners generalizing to the multi-dimensional case in one of two ways, *dependent* or *independent* warping. In general, it appears the community believes either that the two ways are equivalent, or that the choice is irrelevant.

In this dissertation, we strive to solve these problems. The contribution of this dissertation is as follows:

First, we show why the idea of applying *symbolic* stream rule discovery to *real-valued* rule discovery is not directly suitable for rule discovery in time series. Beyond our novel definitions/representations, which allow for meaningful and extendable specifications of rules, we further show novel algorithms that allow us to quickly discover high quality rules in very large datasets that accurately predict the occurrence of future events

Finally, we show that the two most commonly used multi-dimensional DTW methods can produce different classifications, and neither one dominates over the other. This seems to suggest that one should learn the best method for a particular application. However, we

will show that this is not necessary; a simple, principled rule can be used on a case-by-case basis to predict which of the two methods we should trust at the time of classification. Our method allows us to ensure that classification results are at least as accurate as the better of the two rival methods, and, in many cases, our method is significantly more accurate. We demonstrate our ideas with the most extensive set of multi-dimensional time series classification experiments ever attempted.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Prediction and forecasting have been a topic of great interest in the data mining community for the last decade. Most of the work in the literature has dealt with *discrete* objects, such as keystrokes (i.e. predictive text), database queries [18], medical interventions [31], web clicks, etc. However, prediction may also have great utility in *real-valued* time series.

The research community seems to have converged on the belief that Dynamic Time Warping (DTW) is remarkably hard to beat as a time series distance measure, across a host of domain applications, and a host of tasks; including clustering, classification and similarity search [42][58]. Moreover, the most often cited reason for *not* using DTW, its relatively high time complexity, has recently become a non-issue. In particular, amortized over a subsequence search or subsequence monitoring task, DTW is slower than Euclidean Distance by less than a factor of two [63]. As a practical matter, carefully optimized DTW is *much* faster than all but the most carefully optimized implementations of Euclidean Distance [35]. For example, a modern cell phone, using the state-of-the-art DTW subsequence monitoring algorithm [63], can easily process streams arriving at several thousand Hertz. However, such devices only *produce* data at about 100Hz.

In this dissertation, we propose an algorithm to discover meaningful rules from time series data in Chapter 2. We then propose an adaptive classification method for multi-dimensional time series which beats rival methods in Chapter 3. Finally, we give

conclusions in Chapter 4. In the following text, we show the detail of the motivations of each project.

1.1 Meaningful Rule Discovery in Time Series

Prediction and forecasting have been a topic of great interest in the data mining community for the last decade. Most of the work in the literature has dealt with *discrete* objects, such as keystrokes (i.e. predictive text), database queries [18], medical interventions [31], web clicks, etc. However, prediction may also have great utility in *real-valued* time series.

For concreteness we briefly consider two examples:

- Researchers in robotic interaction have long noted the importance of short-term prediction of human initiated forces to allow a robot to plan its interaction with a human. For example a recent paper notes the critical *"importance of the prediction of motion velocity and the anticipation of future perceived forces* [to allow the] *robot to anticipate the partner's intentions and adapt its motion*" [11].

- Doppler radar technology introduced in the last two decades has increased the mean lead time for tornado warnings from 5.3 to 9.5 minutes, saving countless lives [3]. But progress seems to have stalled, with 26% of tornados within the US occurring with no warning. McGovern et al. argue that further improvements will come not from new sensors, but from yet-to-be-invented algorithms that "*examine existing* (time series) *data for predictive rules*" [21].

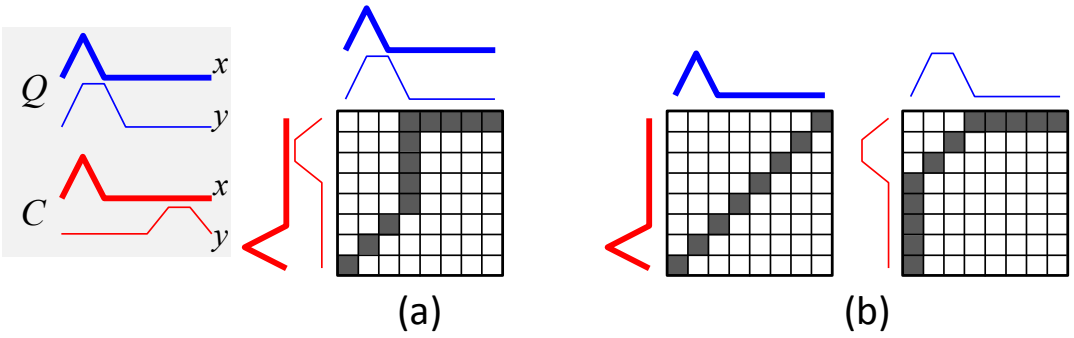
Most of the current work has attempted to predict the future based on the current *value* of a stream [20]. However, for many problems the actual values are irrelevant, but the *shape* of the current pattern may foretell the future. For clarity we call the former *forecasting*, and the latter, the subject of this dissertation, *rule-based prediction* (although the literature is inconsistent on this convention). There is an additional critical distinction between forecasting and rule-based prediction. Time series forecasting is typically *always-on*; it predicts values at every time step. In contrast, rule-based prediction *monitors* the incoming data at each time step, but only occasionally makes a prediction about an imminent occurrence of a pattern.

While *forecasting* is mature enough to have its own conferences and commercial software (SAS, IBM Cognos, etc.), the handful of research efforts to consider time series rule-based prediction have met with limited success. In particular, it is widely accepted that these efforts allow the discovery of spurious rules [14], including finding high confidence "rules" in *random walk* data. We believe that the reason why rule discovery in real-valued time series has failed thus far is that most efforts have more or less indiscriminately applied the ideas of *symbolic* stream rule discovery to *real-valued* rule discovery. In this dissertation, we argue that such ideas are not directly transferable to rule discovery in real-valued time series. Instead, we formulate a rule representation and a Minimum Description Length (MDL) inspired search strategy that evaluates candidate rules based on how well they can compress the data.

1.2 Classification of Multi-Dimensional Time Series

Virtually all attempts to improve time series classification in the last two decades have focused on the single-dimensional case, with the assumption that the generalization to the multi-dimensional case is trivial. There are two obvious ways DTW can be generalized to the multi-dimensional case: Figure 1 gives a visual intuition, which we formalize later in this work. For clarity, we refer to the two methods as DTW_D and DTW_I (with $_D$ standing for Dependent and $_I$ for Independent).

The vast majority of researchers seem to think that it makes no difference which method is used, as evidenced by the fact that they usually do not explicitly bother to *tell* the reader.



(a) $DTW_D(Q,C) = DTW(\{Q_x,Q_y\},\{C_x,C_y\}) = \mathbf{3.2}$

(b) $DTW_I(Q,C) = DTW(Q_x,C_x) + DTW(Q_y,C_y) = \mathbf{2.4}$

Figure 1: *top.left*) Two multi-dimensional time series. *a*) The DTW_D distance between them is 3.2. *b*) The DTW_I distance between them is 2.4. All elements of this visual key are formally defined below.

With some introspection we can see that there are actually several possibilities:

- There is no difference between DTW_D and DTW_I; they produce the same values for all time series.

However, we can immediately dismiss this possibility; as shown in Figure 1, the two methods generally produce different distance values, and thus *could* produce different class labels if classifying an object using the Nearest Neighbor (NN) algorithm.

The next possibility seems to be the one *implicitly* assumed by the community:

- DTW_D and DTW_I *can* produce different distance values, but this makes no difference in the classification accuracy.

As we shall show, this is not the case. The choice of DTW_D vs. DTW_I *can* make a significant difference in the classification accuracy.

Given that DTW_D and DTW_I can have different classification accuracies, one might then imagine that the following is the case:

- While DTW_D and DTW_I can produce different classification accuracies, it so happens that one of the two is always superior on all problems. If we could prove, or experimentally demonstrate this, we could "retire" the weaker measure.

This idea is tempting and has some precedents in similar situations in the literature. However, as we shall show, it is not the case. Datasets exist where DTW_D significantly outperforms DTW_I *and* vice-versa.

This would appear to be the end of the discussion. For a given problem, we can use cross-validation to determine which method to use, then simply hard-code it into our classifier. However, there are two reasons why this is *not* the last word. First, we do not have the luxury of cross-validation when we have very small training sets, a situation that

is very common when *cold-starting* a gesture recognition system or when labeled data is expensive. Secondly, we are not done moving down the hierarchy of possibilities. In particular:

- For any given domain, it may be that, on an individual *class-by-class*, or even *exemplar-by-exemplar* basis, DTW_D and DTW_I can produce different results, and that we could predict which of the two methods to trust at classification time.

This possibility is less intuitive than the others. It is not clear that the utility of the measures should vary within a single domain, and, if it did, correctly predicting which measure was most likely to have been correct on a case-by-case basis seems like an untenable undertaking.

In this work, we show for the first time that this last possibility is correct. The utility of DTW_D and DTW_I varies on an instance-by-instance basis, and our technique, DTW_A ($DTW_{Adaptive}$), can predict at run time with high accuracy in terms of which of them is more likely to be correct [66].

Before leaving this section, we will give a visual and initiative example of our claims. While we normally think of DTW in the context of "true" time series, it has also been used to classify (suitably represented) text, spectrographs, shapes [52], and, as shown in Figure 2, *colors* [75].

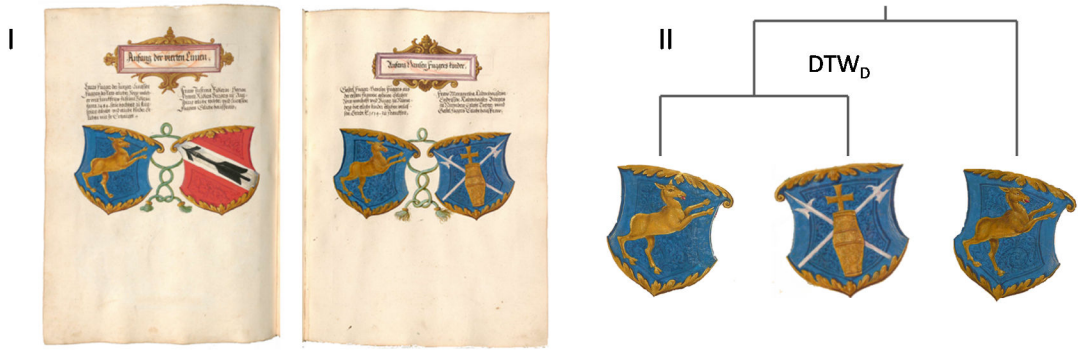


Figure 2: I) Two pages from a 16th century family history. The heraldic shield featuring the golden deer on a blue background is the family crest of Lucas Fugger. II) A clustering of three of the shields under DTWD shows an unintuitive result, the two examples of the Fuggers are not grouped together.

Because color is typically represented in a three dimensional RGB space, it naturally forms a multidimensional time series, as shown in Figure 3.

The two pairs of examples shown in Figure 3 are markedly different. In the pair of heraldic shields, each color needs to warp *independently*. A detailed, high-resolution examination of the images suggests why. While the blue background appears identical in each shield, the gold coloring of the deer is much darker in the uppermost example (this is easier to see in the large format images available at [76]). This difference is probably explained by the fact that the book took four years to produce, and maintaining exact hues over that time period would have been very difficult, especially with 16th century pigment technology.

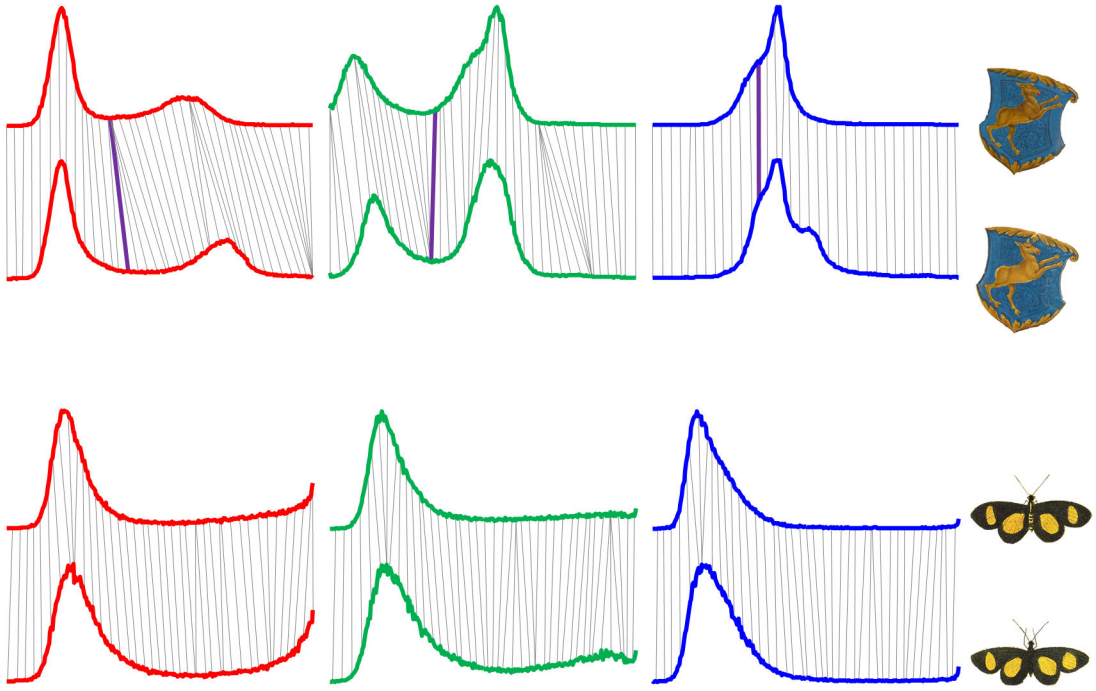


Figure 3: The color histograms of four objects taken from 16th century manuscripts. From left to right the red, green and blue channels are presented. Each channel has been independently aligned by DTW. *top)* The two examples from family crest of Lucas Fugger have radically different warpings in each of the three color channels, perhaps reflecting the fact that the book was created over a four year period. *bottom*) In contrast, the two butterfly examples have essentially identical alignments [37].

To make this clearer, we picked a single point just left of center on each channel of the lighter shield and recolored and thickened the hatch line that illustrates the warping. As we can see, in the blue channel the line is vertical, indicating no warping, in the green channel the line leans forward, and in the red channel the line leans backwards. This observation immediately explains the unintuitive clustering shown in Figure 2.*right*. By using DTW_D we forced all channels to warp in a single compromised way. If we simply use DTW_I we *do* obtain the correct clustering here.

This discussion seems to argue in favor of using DTW_I, at least for images. However, there are examples in which all color channels warp in the same way, as in the butterflies in Figure 3.*bottom* (see also Figure 5 of [75]). This happens if one image is simply sun-

faded, or it can be an artifact of the scanning process. In such situations we are better off using DTW_D which finds the best warping by pooling evidence from all three sources of information.

Our work has two implications for the time series research community: we free researchers/implementers from having to decide which technique to use for their problem; and, because *error*(DTW_A) will be *minimum*[*error*(DTW_D), *error*(DTW_I)], they can use our method safe in the knowledge that they did not choose the suboptimal method.

However, this greatly understates the case, as the correct inequality implied by our work is the more unintuitive *error*(DTW_A) ≤ *minimum*[*error*(DTW_D), *error*(DTW_I)]. That is to say, on some datasets our method can be significantly more accurate than *either* of the rival methods.

Chapter 2: Discovery of Meaningful Rules in Time Series Data

In this chapter, we will demonstrate an algorithm to discover meaningful rules in time series [27]. We break this chapter into eight sections. In Section 2.1 we review the background and the related work of this topic. We then illustrate the intuition behind rule discovery in Section 2.2. In Section 2.3 we show rules in real-valued data and formally define rules in Section 2.4. We discuss the Minimum Description Length, MDL, in Section 2.5. We discuss our rule discovery algorithm in Section 2.6. Experimental evaluation of our proposed method is shown in Section 2.7. Finally, we offer the conclusion of this part in Section 2.8.

2.1 Background and Related Work

In a sequence of papers culminating in [23], Park and Chu investigate a rule finding mechanism for time series. However, the algorithm is only evaluated for speed and then only on random walk data. No evidence was presented that the algorithm could actually find generalizable rules in time series.

Work by Wu and colleagues also use a piecewise linear representation to support rule discovery in time series. They tested their algorithm on real (financial) data, reporting approximately 68% "correctness of trend prediction" [32]. However, the authors graciously ran their algorithm on data provided by others and when they ran their algorithms on pure

random walk data, they again achieved approximately 68% correctness of trend prediction [33]. This suggests their original results did not outperform random guessing.

The most referenced time series rule-finding method in the literature is [6], which quantizes the data with K-means clustering of the entire training dataset and passes the (now) symbolic data over to a classic association rule discovery algorithm. The success of a rule is measured with a score called the J-measure. The method was used in several papers before it was shown that the J-measure gave the same significance to rules found in completely random data as to rules found in real data [14]. Later analyses by more than a dozen follow-up papers suggest that the problem is with the quantization step; in essence any technique that involves clustering *all* subsequences is doomed to produce cluster centers that are *independent* of the data [14]. In Section 2.7.8 we have compared our algorithm to the three highly cited rival methods.

For brevity, we forgo an in-depth review of MDL and instead direct the interested reader to [1][17]. We will discuss our use of MDL in Section 2.5.1.

2.2 The Intuition of Rule Discovery

It may be instructive to first consider the analogue problem of rule discovery in symbolic strings. Let us consider "The Raven", by Edgar Allan Poe. It begins:

Once upon a midnight dreary, while I pondered weak and …

What are the possible rules we might discover in this text? One possible rule is that the word "*door*" often follows the word "*chamber,*" a rule we can denote as:

chamber → door

The left side of the rule is the *antecedent* and the right side is the *consequent*. This rule is based on our observation that we see the phrase "*chamber door*" ten times in the text. We note that this is not a perfect rule; the word "*chamber*" appears once without been followed by "*door*" ("*...into the chamber turning...*"). Furthermore, it is important to note that the rule does not make the claim that all, or even many, occurrences of "*door*" are preceded by "*chamber*". In fact, there are four more examples of the word "*door*" in the text.

A major difference between text and time series is that the latter does not have a natural segmentation (i.e. spaces or periods), thus we are facing data that is more like this:

onceuponamidnightdrearywhileIponderedweak....

Given such a text, there are (language agnostic) algorithms that can segment the string into the original words [5], with varying degrees of accuracy. However, segmenting a *real-valued* time series into meaningful episodes is much more difficult. Furthermore, the problem is further complicated by the fact that, in most cases, the time series does not consist solely of discretely concatenated events. Rather, the events may be interspersed with filler symbols. For example, if we examine a motion capture of a sign language version of this poem there will be locations that do not correspond to discrete signs, but rather to transitions between signs. This will produce something rather like this:

oncexauponwamidnightmtdrearydwhileuIpponderediweak...

Finally, time series are inherently real-valued and as such, tests for equality are meaningless. This would be equivalent to our text string having some misspellings:

qncexauponwamidmightmtdreerydwgileuIpponderediweek...

The problem is now significantly more difficult than the original statement. We must generalize the antecedent to allow flexibility, perhaps by triggering the occurrence of a pattern that is within a certain threshold *t* distance under some suitable distance measure:

$$dist(\text{“}\textbf{chamber}\text{”}, \text{substring}) \leq t \rightarrow \textbf{door}$$

However, we are not done generalizing the rule model. The existence of misspellings in our data means that we may wish to accept similar consequents such as *poor* or *dooor* as successful predictions. Furthermore, we originally assumed that the consequent immediately followed the antecedent. However there may be some additional symbols between words. Thus we need to define a parameter, *maxlag*, which is the maximum number of characters between the end of the antecedent and the beginning of the consequent. For example, if *maxlag*, is set to two, then any of the below would be considered successful predictions:

...**chamberdoor**..., ...**chamber**z**door**..., ...**chamber**xy**door**...

but the following:

...**chamber**xzuv**door**...

is not a successful prediction because the lag between the antecedent and consequent is too long.

The *maxlag* parameter allows for meaningful falsifiable predictions. The prediction that “*this consequent will eventually occur*” is paradoxically both unfalsifiable and almost certainly true (if we wait long enough). We can now show our final rule format:

$$dist(\textbf{chamber}, \text{substr}_i) \leq t_1 \rightarrow dist(\textbf{door}, \text{substr}_j) \leq t_2,$$

$$j - (i + \rho - 1) \leq maxlag$$

This can be read as follows: "If we see a substring of length ρ that is within distance t_1 of the word *chamber*, then we fire the rule and expect to see a similar substring to word *door*, within a learned distance t_2, in the next *maxlag* time steps."

2.3 Moving to Real-Valued Data

We are now ready to begin to "port" our ideas to the real-valued time series that are of interest in this work. We will start with an example for which we know the ground truth and for which the reader has already developed some intuition. However, we note that we are *not* using external knowledge to help our algorithm, only to validate and explain it. As shown in Figure 4, we took an audio recording of the first four verses of "The Raven" (performed by an American male actor), and converted it to Mel-frequency cepstrum coefficient (MFCC) space, keeping just the second coefficient.

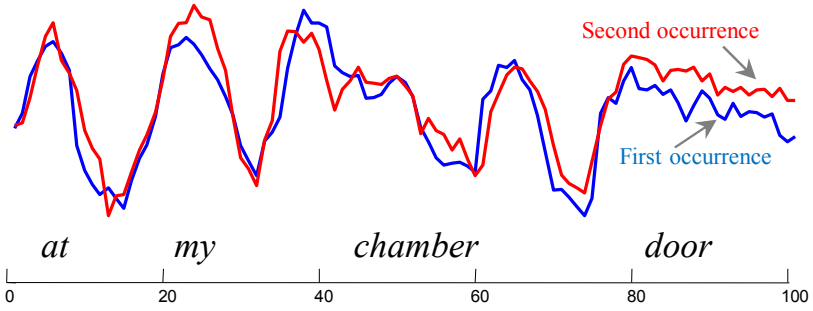


Figure 4: The motif pair discovered in the first 2,000 data points (20 seconds) of "The Raven". The shape corresponds to the utterance "...at my chamber door".

Using just the first 2,000 data points, which corresponds to the first verse of the poem, we found the pair of non-overlapping subsequences of length 100 (one second length in

the original data) that had the minimum distance to each other. Such a pair of subsequences is referred to as a *time series motif* and extensively studied in the literature [21][22].

The occurrence of such a highly conserved motif suggests *one* possible method for specifying rules. We could simply split the motif pattern in two, let the average of the left side be the antecedent, and let the average of the right side be the consequent. We need to set the *maxlag* and the threshold, t_1, parameters. For the moment, let us set the former to zero and the later to twice the distance between the antecedent motif prefixes. Figure 5 shows the rule.

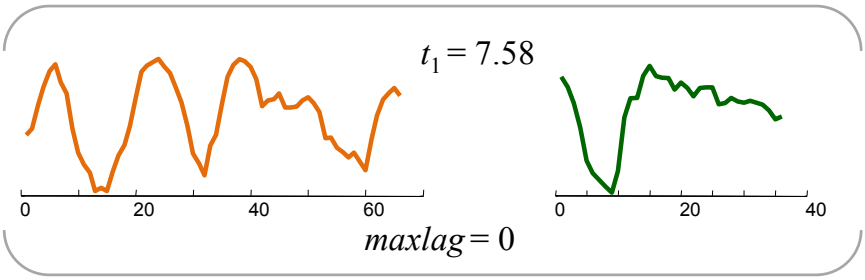


Figure 5: A rule learned (Figure 4) from the first 2,000 data points of the data. If the antecedent pattern (*left*) is matched to a subsequence in a stream that is within Euclidean distance of 7.58 to it, we predict the immediate occurrence of the consequent pattern (*right*).

We can immediately test this rule by running it on the remainder of *The Raven* data. The rule fires exactly three times and in every case it maps to an utterance of "***door.***" In this simple example, hard-coding the *maxlag* to zero is intuitive; however, we can easily imagine examples that need the flexibility of a larger *maxlag* constraint. Consider Figure 6 which shows accelerometer data collected from a device worn by a student at USC as he went about daily activities [24].

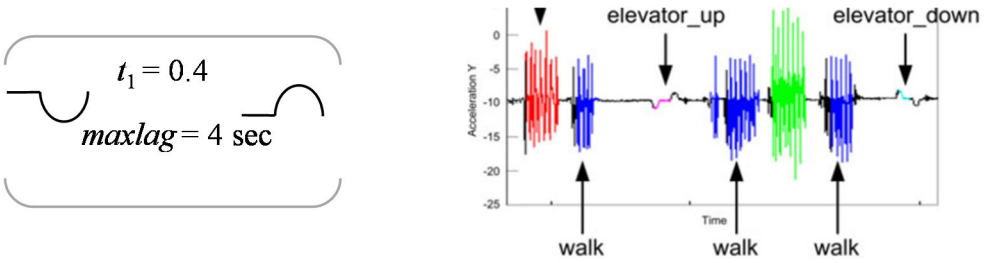


Figure 6: *left*) A rule for an accelerometer dataset encodes the fact that the initial acceleration "bump" of going up in an elevator must be eventually be matched by the elevator stopping at a floor. *right*) Real data from which this rule was learned [24].

This example shows a very easy rule to spot. The semicircular bump created by an elevator accelerating must eventually be matched by a bump in the opposite direction when the elevator brakes (the rule for elevators going *down* is similar, but with the consequent and antecedent swapped). The time lag between these two events is highly variable and depends on the number of floors serviced by the elevator.

2.4 Rule Framework

We are now in a position to present the definitions necessary to rigorously define our rule framework. First, we need to define a distance measure between two subsequences. While there are dozens of measures in the literature, recent empirical evidence suggests that Euclidean distance is very difficult to beat [8]. Furthermore, Euclidean distance is parameter-free, fast to compute, and is amiable to various data mining optimizations such as indexing and early abandoning computation [22]. We empirically considered other distance measures including DTW, Swale, Spade and EPR [6], none improved the accuracy of the rules (a finding consistent with [6]) and all required at least an order of magnitude more time.

We formally define a time series *antecedent* as a subsequence used to trigger a rule if it is similar to the current sliding window:

Definition 1: Assume we are monitoring a time series by continuously extracting the sliding window, W. Given a positive constant t (threshold), and an *antecedent* time series R_a, a binary flag `fired` is set to **`TRUE`** if $D(R_a, W) < t$.

Note that in order for a candidate antecedent to be even considered as a rule precursor, it must occur at least twice; we cannot generalize from single exemplars. This is essentially the definition of a *time series motif* [22]. In Section 6, we will exploit this in order to reduce our search space of antecedents and consequents.

In principle, the threshold, *maxlag*, and antecedent could be hand chosen by a domain expert. However, as we show later it is possible to find them automatically. As an antecedent is a precursor to an event, a predicted subsequence shape which follows an antecedent within a specified time (the *maxlag*) is called the antecedent's *consequent*:

Definition 2: A *consequent,* R_c, is a time series subsequence that is predicted to follow the detection of an *antecedent* within *maxlag* time steps.

The *maxlag* parameter encodes the fact that for a time series subsequence to be a meaningful consequent in a rule, it must occur within some acceptable time after the rule's antecedent has been detected. Without such a constraint on time, a consequent's occurrence may be coincidental.

Definition 3: The *maxlag* is the maximum number of time steps allowed between a detected *antecedent* and its *consequent*. In particular, if t_k is the last value in W, the moment

the rule is triggered, then the consequent must be derived from a subsequence of T, T_i, such that $0 \leq i - k - 1 \leq maxlag$.

With an *antecedent*, its *consequent*, the maximum expected *maxlag* delay between the two, and the *threshold* distance used to trigger a subsequence match, we have all the necessary components to specify a single *time series rule*:

Definition 4: A *time series rule, R,* is a 4-tuple of $\{R_a, R_c, maxlag, t\}$.

One obvious way to obtain an *antecedent* and *consequent* with a zero *maxlag* is to take a subsequence and split it:

Definition 5: The *Split Point* is a ratio in the range (0, 1) which indicates the end point of the *antecedent* and the beginning of the *consequent*.

Having defined time series rules and all supporting notation, we have just two more tasks. We need to formalize a scoring function to tell us how good a candidate rule is, and design an efficient search strategy.

2.5 Data Discretization

Because of our intention to use MDL to measure the relative merits of candidate rules, we must transform our real-valued time series into a discretized space [17]. After consideration of the many quantization options, we quantize the time series' real values into uniformly distributed bins. For a subsequence length ρ, we z-normalize all possible subsequences of that length and record the minimum and maximum values across the normalized subsequences. After attaining the global minimum value, *min*, and global

maximum value, *max,* across all subsequences, we set bin boundaries that are uniformly distributed between *min* and *max*. The resulting bin width is then: (*max* - *min*) / cardinality.

We can show the (lack) of effect that discretization has on time series with *classification* experiments, since the rule triggering step is essentially a classification problem. We conducted empirical tests on data from the UCR Archive [29]. For each dataset, we ran leave-one-out one-nearest-neighbor classification tests using uniform quantization with varying cardinalities. Table 1 provides a snapshot of the results.

Table 1: One-nearest-neighbor leave-one-out accuracy results on UCR datasets for various cardinalities.

Dataset	64-bit(raw) Cardinality:2^{64}	16-bit Cardinality: 65536	6-bit Cardinality: 64
50words	63.1%	63.1%	63.3%
CBF	85.2%	85.2%	85.2%
Beef	66.7%	66.7%	66.7%
ECG	88.0%	88.0%	88.0%
FaceAll	71.4%	69.6%	69.6%
Fish	78.3%	78.3%	77.7%
Lightning2	75.4%	75.4%	77.1%
OSULeaf	52.1%	52.1%	52.1%

It is demonstrated that a real-valued time series can be drastically reduced through discretization without significantly affecting the intrinsic information available. In fact, because cardinality reduction of the original data can reduce the effects of noise and outliers, we can sometimes see tiny improvements in accuracy. These results allow us to use MDL with little fear that we are throwing away valuable information. Which value of cardinality should we use? Empirically, if the value is anywhere in the range of [16, 65536], it makes no significant difference; we therefore use a cardinality of 16 throughout this work.

2.5.1. MDL Scoring

We begin with an important disclaimer. We claim only that our work in this section is *inspired by*, and in the *spirit of* MDL (and MML [30]). In particular, we have adopted (cf. [12]) and extended ideas may deviate slightly from the absolute purist's interpretation of MDL. Our goal here is to produce a pragmatic scoring function that works in the real world. We thus defer theoretical and philosophical discussions to an appropriate venue.

The intuition behind our scoring function is that if we make a good prediction, the consequent shape we predict will be similar to a subsequence that occurs within *maxlag* steps. We could quantify this similarity with Euclidean distance (essentially the *mean squared prediction error* used in forecasting [20]), however, the Euclidean distance does not allow us to compare the quality of consequents with different lengths. To make this clearer, let us return to our expository *text* example. Suppose we have to evaluate the following candidate rule: *dist*("***chamber***", substring) ≤ *t* → ***door***, which when fired makes a prediction of length four. When encountering this string:

*... bustabovehis**chamber**door**with**such**name**as**never**more…*

it achieves a hamming distance (a good analogue of Euclidean distance) of 0. Contrast this result with the following rule: *dist*("***chamber***", substring) ≤ *t* → ***doorwithlikename***, which when fired makes a prediction of length sixteen. While encountering the same string:

*... bustabovehis**chamber**doorwith**~~such~~**name**as**never**more…*

it achieves a hamming distance of four. Which of these two rules is better?

The former is an *exact but short* prediction; the latter is an *approximate but longer* and arguably more informative prediction. Unfortunately, simply normalizing for length does

not work here; while it is not commonly understood, the Euclidean distance between two subsequences of length ρ can actually *decrease* when we expand to length $\rho + 1$ due to the (re)normalization of the data. So not only is the effect of length not linear, it is not even monotonic.

Our solution to this problem, and the reason for the earlier digression into discretization of time series, is MDL [1][17]. For several decades MDL has been used to solve very similar problems in intrinsically discrete domains such as text, DNA, MIDI, etc. However, this application to time series rules is novel.

The intuition behind our use of MDL is to consider a candidate subsequence as a *hypothesis, H,* about a future event. This hypothesis (the bold/green line in Figure 7) has some cost, the number of bits it takes to store it. We denote this cost as the Description Length, *DL*. If we store the subsequences as simple integer arrays, we have $DL(H) = \text{length}(H) \times \log_2(\text{cardinality})$.

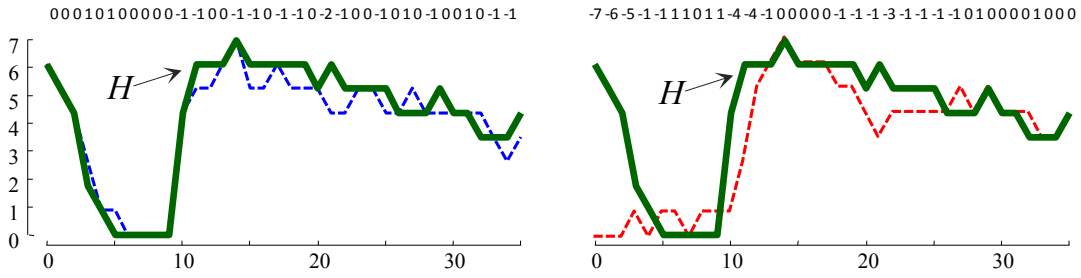


Figure 7: A hypothesis (green/bold) can be used to score subsequences by subtracting it from them (producing the small integers shown *top*) and encoding the difference vector with Huffman encoding. Intuitively, here the *left* sequence requires 57 bits, whereas the *right* sequence requires 84.

We then want to evaluate the quality of a predicted consequent by asking how well the prediction matched the future. We do this by asking, "Given our consequent *H*, what is the cost to encode the error of the actual match *m*?" We denote this as $DL(m \mid H)$, that is, the

description length of a matching subsequence *m*, *given* our hypothesized consequent *H*. We can measure this encoding cost by simply subtracting the consequent from the matching time series and encoding the difference vector efficiently. Thus the score of a candidate subsequence, *m*, with a hypothesis, *H*, is:

(1) $bit\text{-}save(m,H) = DL(m) - DL(m|H)$.

This idea is illustrated in Figure 7. Here we use a cardinality of just eight values for visual clarity.

This equation allows us to measure the relative predictive power of subsequences, *independent of their length*. In order to find rules in a training set, we must have at least two firings. This means that to evaluate the hypothesis we must measure how well it encodes a set, *M*, of at least two consequents.

(2) $total\text{-}bit\text{-}save(M,H) = -DL(H) + \sum_{m \in M} bit\text{-}save(m,H)$,

where the set *M* consists of all subsequences to be compressed with the consequent *H*.

2.6 Rule Discovery Algorithm

We are finally in a position to introduce our rule finding algorithm. In essence, it has two parts: 1) a scoring function and 2) a search algorithm which repeatedly invokes this scoring function while searching for high quality rules. As the scoring function is at the heart of our ideas, we will detail the intuition behind it next and then in Section 2.6.2 we will present our rule search method which utilizes this function.

We begin by noting that *all* experiments are reproducible. All experimental code and data are archived in perpetuity at (supporting page).

2.6.1. Rule Scoring

For clarity of presentation we begin by considering the case in which *maxlag* is constrained to be zero.

Our MDL scoring function is given two inputs: a candidate time series (like either *one* of the two time series in Figure 4) and an expected *maxlag* value (recall for the moment, it is hardcoded to zero). The function then returns three things: an antecedent, a consequent and the quality score of the resulting rule. Note that the antecedent concatenated with the consequent are simply the input time series, *R*, (much like Figure 5), however the split point is not known in advance. Table 2 illustrates the algorithm. Note that we propose a parameter-lite algorithm (Table 2), which is described by hierarchical functions that automatically generate most of the required inputs in Tables 3 to 6.

Table 2: Algorithm to score all rules that can be created from a single time series subsequence *R*, returning the antecedent, consequent and quality of the best rule derived from *R*

Procedure *find_Best_Rule* (***T***, ***R***) Input: A time series subsequence, ***R***, extracted from a time series, ***T***; Output: The antecedent (***a***), consequent (***c***) and quality score (***s***) of the best rule that can be derived from ***R***;	
1	**for** $i \leftarrow 1$ to 99 **do** //Test over all splitting points
2	$splitPoint \leftarrow i / 100$
3	$ruleScore(i) \leftarrow Best\_Rule\_Score\ (T, R, splitPoint)$ //Table 3
4	**end for**
5	$s \leftarrow \max\ (ruleScore)$
6	$sp \leftarrow$ find $(ruleScore == s) / 100$
7	$a \leftarrow R\ (1 : sp \times \text{Length}(R))$
8	$c \leftarrow R\ (sp \times \text{Length}(R) + 1 : \text{end})$
9	**Return** *a, c, s*

In lines 1 to 4 the algorithm iterates on all possible split points for the candidate time series, *R*, and calculates the quality score described in Table 3. In line 5 we find the maximum quality score (*s*). In lines 6 to 8 we find the split point corresponding to the maximum quality score and we split *R* to the antecedent (*a*) and the consequent (*c*). The procedure returns *a, c*, and *s*.

In Table 3 we describe *how* the rules are scored. For every antecedent that can be produced by *R*, we search for locations in *T* in which that rule would have fired. Given each firing, we "predict" the relevant consequent as a hypothesis *H* to explain the next |*c*| datapoints in *T*. We then calculate how many bits MDL could save using this prediction. If, as in Figure 7.*left,* our "prediction" was accurate we will save many bits. A less accurate prediction (Figure 7.*right*) will save fewer bits. The number of bits saved; summed over all firings (i.e. Eq. 2) is the score returned for the tentative rule.

Table 3: Algorithm to find the best instances of a rule

Procedure *Best_Rule_Score* (***T***, ***R***, ***sp***) Input: A time series subsequence, ***R***, extracted from a time series, ***T***; Split point for the antecedent/consequent, a number between zero and one, ***sp***; Output: Greatest possible bit-saves by predicting rule *R* in the time series *T*, *bestTotalBitSave;*	
1	*ac* ← *find_Antecedent_Candidates* (*T*, *R*, *sp*) //Table 4
2	*n* ← *find_Best_Number_of_Rule_Instances* (*T*, *R*, *sp, ac*) // Table 5
3	*bestTotalBitSave* ← *Rule_Bit_Saves* (*T*, *R*, *sp, n, ac*) // Table 6
4	**Return** *bestTotalBitSave*

Concretely, in line 1 we find the set of subsequences similar to the antecedent of *R*. In line 2 we learn a threshold for the distance that leads to the largest quality score for *R*. In line 3 the algorithm calculates the largest number of bits saved for the rule instances and finally returns that value as a quality score for the rule.

We find the set of subsequences similar to the antecedent of *R* in the subroutine described in Table 4. The first element of the set is the *antecedent* itself, the second element is the most similar subsequence to the *antecedent*, the next subsequence is the second closest and so on. This set includes all firings of *R* which need to be tested in Table 5.

Table 4: Algorithm to find a set of antecedent candidates

Procedure *find_Antecedent_Candidates* (***T***, ***R***, ***sp***) Input: A time series subsequence, ***R***, extracted from a time series, ***T***; Split point for the antecedent/consequent, a number between zero and one, ***sp***; Output: locations of antecedents in ***T*** ordered by distances from ***R***'s antecedent, ***ac***;	
1	*antecedentLength* ← Length(*R*) × *sp*
2	*antecedent* ← *R* (1:*antecedentLength*)
3	*Distances* ← Euclidean (*antecedent*, each subsequence in *T*)
4	*AntecedentDistances* ← sort (localMinimums (*Distances*))
5	*AntecedentCandidates* ← Locations (*AntecedentDistances)*
6	*ac* ← *AntecedentCandidates*
7	**Return** *ac*

In lines 1 and 2 we find the antecedent of the rule *R*. In line 3 we slide the antecedent across the entire time series, *T*, and calculate the Euclidean distances for each subsequence of the same length. In line 4 the algorithm finds the local minimums and sorts them according to their distances (ignoring *trivial matches* [22]). In line 5 we find the locations of the sorted distances in the time series *T* and finally the procedure returns a set of antecedent candidates sorted by their distances to the *antecedent* of *R*.

Recall that in Table 3 we search for locations in *T* in which that rule would have fired. However the number of firings clearly depends on the distance threshold we have chosen. A conservative (small) threshold is more likely to produce an accurate rule, but may miss opportunities when it *could* have fired and produce predictions that are at least better than random. We generally have no idea what a suitable threshold could be, fortunately we only

have to test |*AntecedentCandidates*| different values. In particular we just need to test all the values in the sorted list *AntecedentDistances*, as each new value ensures exactly one additional firing of the rule on our training data, this occurs in Table 5.

Table 5: Algorithm to discover the best number of rule instances to maximize the total number of bit-saves

Procedure *find_Best_Number_of_Rule_Instances* (***T***, ***R***, ***sp, ac***) Input: One instance of a rule, ***R***, extracted from a time series, ***T***; Split point for the antecedent/consequent, between zero and one, ***sp***; Locations of antecedents in ***T*** ordered by distances from ***R***'s antecedent, ***ac***; (i. e. *AntecedentCandidates*) Output: Best Number of instances of *R* to pick in the time series, ***n***;	
1	*totalBitSaves* (1) ← 0
2	*instances* ← 1
3	**while** (*totalBitSaves* is monotonically increasing) **do**
4	*instances* ← *instances* +1
5	*totalBitSaves*(*instances*)←*Rule_Bit_Saves*(*T*,*R*,*sp*,*instances*,*ac*)
6	**end while**
7	*bestBitSaves* ← max (*totalBitSaves*)
8	*n* ← find (*totalBitSaves* == *bestBitSaves*)
9	**Return** *n*

In lines 3 to 6 we iterate on the number of rule instances and each time calculate the total number of bit-saves as in Eq. 2. The loop terminates when the *totalBitSaves* starts decreasing. This use of a *greedy* approach to avoid searching all possible rules produces several orders of magnitude speedup, with little chance of missing a useful rule. In Section 2.6.2 we show that we can use the Euclidean distance as a heuristic to both guide the "rule test" order, and to tell us when we can abandon the rule test with a small, user-defined probability of missing the optimal answer (cf. Figure 8). We further justify a probabilistic early abandoning approach in our supporting webpage [34]. In line 7 we calculate the maximum number of total bit-saves and in line 8 we find the corresponding number of rule instances picked during the iteration in lines 3 to 6.

In the subroutine in Table 5 (and its invoking functions) we used Euclidean distance to process the data and *create* a large set of candidate rules with their observed outcomes on the training data. In Table 6 we move from Euclidean distance to MDL to *score* these rules.

We consider the *consequent* of *R* as a model/hypothesis and calculate the total number of bit-saves in order to predict other consequents. A larger number of bit-saves indicates more accurate predictions. After discovering antecedent candidates, we consider their following subsequences as consequents. The procedure then calculates the number of bits required to record the differences of the *consequent* saved as a model and the subsequences following antecedent candidates.

In lines 1 and 2 the algorithm finds the *consequent* of *R*. In line 3 we discretize the *consequent* into 16 values and we z-normalize it. We will use this *consequent* as the hypothesis therefore we exclude the antecedent of *R* in line 6. In lines 7 to 10 for all *n-1 AntecedentCandidates* we find their corresponding consequents.

In line 11 the algorithm discretizes and z-normalizes the corresponding consequents. In line 12 we calculate the number of bits required to record the consequents by using Huffman coding. In line 13 we use the *consequent* of R as a hypothesis and calculate the number of bits to save all other consequents by using MDL.

Table 6: Algorithm to score rule instances based on MDL

Procedure *Rule_Bit_Saves* (***T***, ***R***, ***sp, n, ac***) Input: A time series subsequence, ***R***, extracted from a time series, ***T***; Split point for the antecedent/consequent, between zero and one, ***sp***; The Number of instances of ***R*** to pick in the time series, ***n***; locations of antecedents in ***T*** ordered by distances from ***R***'s antecedent, ***ac;*** Output: ***totalBitSave***;	
1	*antecedentLength* ← Length(*R*) × *sp*
2	*consequent* ← *R* (*antecedentLength:end*)
3	Discretize and z-normalize (*consequent*)
4	*AntecedentCandidates* ← *ac*
5	*totalBitSave* ← 0
6	*antecedentsSelected* ← *AntecedentCandidates* (2 : *n*)
7	**for** *i* ← 1 to Length(*antecedentsSelected*) **do**
8	s_1 ← *AntecedentCandidates*(*i*) + *antecedentLength* +1
9	s_2 ← *AntecedentCandidates*(*i*) + Length(*R*)
10	*subConsequent* ← T (s_1 : s_2)
11	Discretize and z-normalize (*subConsequent*)
12	*subConsequentBits* ← Huffman (*subConsequent*)
13	*subConsequentMDLbits*←MDL(*subConsequent, consequent*)
14	*totalBitSave* ← *totalBitSave* + *subConsequentBits* -
15	*subConsequentMDLbits*
16	**end for**
17	**Return** *totalBitSave* – Huffman (*consequent*) // Eq. 2

In lines 14 and 15 our algorithm calculates the total number of bit-saves by subtracting the number of bits to record the consequents by using MDL from the number of bits to save the consequents by using Huffman coding (i.e. Eq. 2) and finally returns the total number of bit-saves, which tells us the quality of the rule.

2.6.2. Motif-Based Rule Searching

The previous section explained the rule *scoring* operator algorithm (Table 2), all that remains is to explain the *search* algorithm that uses this operator. In principle we could use a brute force search, testing *all* subsequences of ***T***. However this would be intractable. Fortunately we have an exploitable observation, a good rule candidate must be a time series

motif in *T,* and efficient algorithms for discovering the top K motifs in a time series are well-known [21][22]. Thus as illustrated in Table 7, we simply evaluate motifs from *T*, until we can claim the probability of finding a better rule is less than some small user-supplied threshold.

In lines 3 to 8 we iterate over the motifs to discover the best rule. In line 4 our procedure calls the subroutine motif_Discovery (*T*, *L*, *K*), which uses the MK motif discovery algorithm [22] to return the K^{th} best motif of length *L* in the time series *T*.

.

Table 7: Algorithm to discover rules in a time series

Procedure *Discover_Rules* (***T, L***) Input: A user provided time series, ***T***, and the rule length, ***L*;** Output: A set of discovered rules, ***Rules;***	
1	*Rules* ← [ ] *// Initialize rules to empty set*
2	*K* ← 1 *// Initialize which motif to consider*
3	**while** (*earlyAbandoning* is **`False`**) **do**
4	*motif* ← motif_Discovery (*T, L, K*) // MK algorithm
5	[*a, c, s*] ← *find_Best_Rule*(*T, motif*) // Table 2
6	*Rules*. add([*a, c, s*])
7	*K* ← *K* + 1
8	**end while**
9	**Return** best(*Rules*) //returns the rule with the maximum score, *s*

Note that by definition, the distance between each pair of motifs is non-decreasing in *K* [22]. We exploit this to create an *earlyAbandoning* function, described later in this section, to terminate the loop. In line 5 we pass the discovered motif to the rule scoring function (Table 2) which finds the best rule that can be derived from that motif. In line 6 we add the discovered rule to the set of existing rules and finally we return the rule which has the largest score *s* in line 9.

We have glossed over the termination condition for our algorithm (line 3). Here we describe it in more detail. Note that there is a strong relationship between Euclidean Distance (which motif discovery is *minimizing*) and bit-saves defined in Eq. 1 (which Table 2 is *maximizing*). To illustrate this we performed the following experiment. From the MFCC time series of a recitation of the poem "*The Dream within a Dream*", we randomly sampled 20,000 subsequence pairs of length 100 (1 sec of audio), denoting one subsequence H and the other m. We measured $\sqrt{(H-m)^2}$ and the $DL(m) - DL(m|H)$ (Eq. 1), and use the two values to create the scatterplot shown in Figure 8.

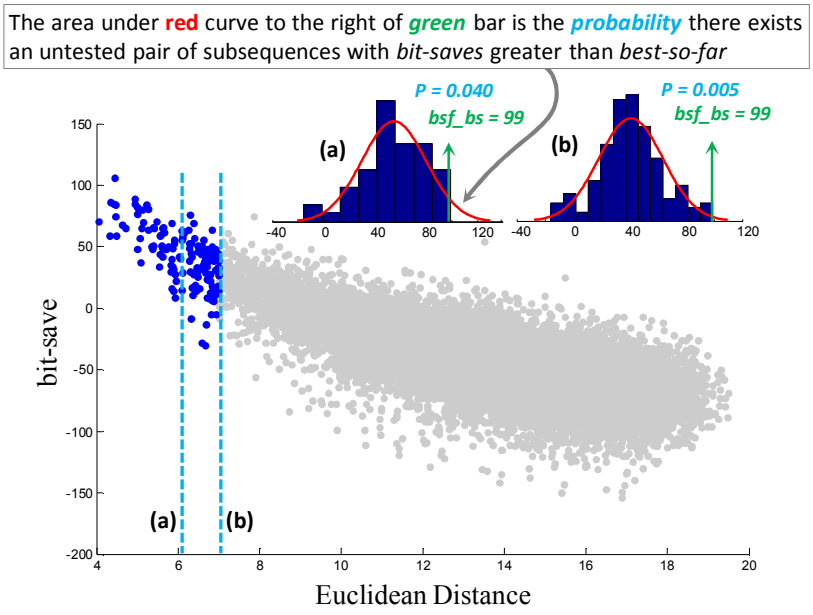


Figure 8: *bottom*) The empirical relationship between Euclidean and bit-save. *top*) As we search in Euclidean order (the x-axis order) from left to right, the expected value of the bit-save (the mean of the Gaussians) decreases.

The figure suggests we can use the Euclidean distance between motifs as a heuristic to tell us when we can abandon the motif discovery with a small, user-provided probability of missing the optimal answer. In essence, we propose to allow rule search in the form

"stop searching when there is only a one in a thousand chance that the current best-so-far is not the best rule."

Let $P_{bit\text{-}save}$(*best-so-far*) be the probability that the remaining pairs of subsequences in the Euclidean searching order (the x-axis ordering of Figure 8.*bottom*) contains a better rule than the rule represented by the current *best-so-far*. Concretely, we compute the bit-saves (Eq. 1) for the subsequences on the left side of the dash-line (a) in Figure 8.*bottom* to form the histogram shown at the top left of Figure 8.

The bit-saves property of the distribution can be realized by a Gaussian process (GP) [9]. The probability vector $\{\varphi_k\}$ is drawn from a GP as $\varphi_k \sim N(\mu_k, \sigma_k^2)$, where μ_k is the mean and σ_k^2 is the variance shown as the red "bell" curve. For example, the *best-so-far* bit-saves is 99 bits for both histograms in Figure 8.*top* and the $P_{bit\text{-}save}$(*best-so-far*) of the distribution changes from 0.040 to 0.005 from left to right as shown in Figure 8.*top*. The area below the red curve to the right of the *best-so-far* marker, is the probability that there exists an untested pair of subsequences with bit-saves greater than the *best-so-far* bit-saves. If $P_{bit\text{-}save}$(*best-so-far*) is less than the user threshold then we simply set the *earlyAbandoning* flag to be **`True`**, and the invoking search algorithm will terminate.

This method has a few assumptions; for example that a thin vertical "slice" of the scatterplot is Gaussian. These assumptions are empirically observed on most datasets (see [34]), and, more importantly, violations tend to result in a more *conservative* algorithm. That is the say, the algorithm may run a little longer, but will over-deliver on the requested probability of a true positive.

Our illustration in Figure 8 makes one assumption that *is* unwarranted, that total bit-saves come from *exactly* two of subsequences. Recall from Table 6 that in fact the total bit-saves come from *at least* two subsequences. We can easily generalize the *earlyAbandoning* function to account for this, but as it makes no empirical difference on the datasets we considered, for simplicity we ignore this idea in this work.

We conclude this section with a simple experiment to reinforce the intuition that motif distances are a good proxy for rules. We took every subsequence of length 100 (one sec) of the MFCC version of "The Raven" and recorded its distance to its nearest neighbor. The distribution of these distances is shown in Figure 9 with a few annotated examples. Note that one occurrence of the phrase "*...chamber door...*" has a very small distance to its nearest neighbor, which is naturally just another occurrence of the phrase. Similarly, the repeated phrases such as "*...the raven....*", "*...on the floor...*", etc., also have small distances to their nearest neighbors. In contrast, phrases featuring hapax legomena[1] such as "*caught*" or "*crest*" have a huge distance to their nearest neighbor. If we were attempting to find rules in the text space, unique words or phrases do not need to be considered since we clearly cannot generalize rules from a single example. Moreover, Zipf's law tells us that about half the words in an English text are hapax legomena [16], and an even larger proportion of *phrases* must be unique. This observation is for *text* and, as Figure 9 hints, it is also true for most *real-valued* time series.

[1] A *hapax legomena* is a word that appears only once in a body of text.

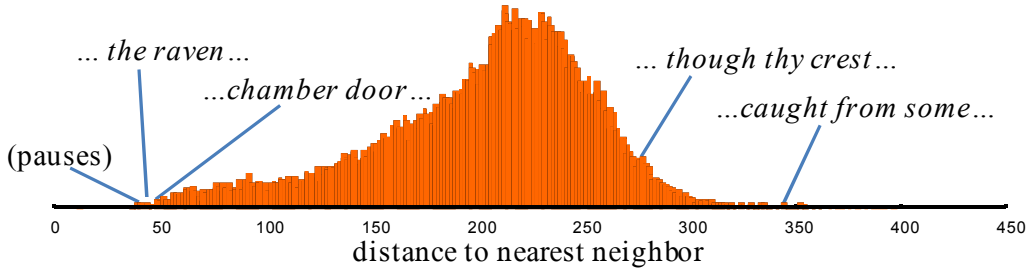


Figure 9: Distribution of nearest neighbor distances for one second snippets of the audio (in MFCC space) of "The Raven."

2.6.3. Generalizing to Allow a Maxlag

Our rule discovery algorithm described in Section 2.6.1 assumes a zero maxlag. To generalize to the arbitrary maxlag value case, we just need to slightly modify the "algorithm to score rule instances based on MDL" (Table 6). All other algorithms in our approach (Tables 2, 3, 4, 5 and 7) remain unchanged. While maxlag is allowed, we should consider a maxlag interval to search the consequent after the split point. The highlighted section of Table 8 shows the modifications of Table 6 which allows a non-zero maxlag in our rule discovery algorithm.

In line 10 we allow a maxlag value after the split point (*subConsequent*) to search for a subsequence in ***T*** closest to the *consequent*. In lines 11 to 14 we slide the *consequent* through each subsequence of the *subConsequent* and find the closest subsequence to the *consequent*. The remainder of the algorithm is the same as Table 6. In Section 2.7.2 we conduct an experiment which requires a non-zero maxlag.

Table 8: Algorithm to score rule instances based on MDL (allowing maxlag) Differs from Table 6, only in lines 10 to 14

Procedure *Rule_Bit_Saves* (***T***, ***R***, ***sp, n, ac, mlag***)
Input: A time series subsequence, ***R***, extracted from a time series, ***T***; Split point for the antecedent/consequent, between zero and one, ***sp***; The Number of instances of ***R*** to pick in the time series, ***n***; locations of antecedents in ***T*** ordered by distances from ***R***'s antecedent, ***ac;*** Maxlag allowed between the antecedent and consequent, ***mlag***;
Output: ***totalBitSave***;

```
1   antecedentLength ← Length(R) × sp
2   consequent ← R (antecedentLength:end)
3   Discretize and z-normalize (consequent)
4   AntecedentCandidates ← ac
5   totalBitSave ← 0
6   antecedentsSelected ← AntecedentCandidates (2 : n)
7     for i ← 1 to Length(antecedentsSelected) do
8       s1 ← AntecedentCandidates(i) + antecedentLength +1
9       s2 ← AntecedentCandidates(i) + Length(R)
10      subConsequent ← T (s1 : s1 + mlag)     // considering maxlag
11      consequentDist ← Euclidean (consequent, each subsequence
12                                  of subConsequent)
13      conseqLoc ← find (consequentDist == min(consequentDist))
14      subConsequent ← T (s1 + conseqLoc : s2 + conseqLoc)
15      Discretize and z-normalize (subConsequent)
16      subConsequentBits ← Huffman (subConsequent)
17      subConsequentMDLbits←MDL(subConsequent, consequent)
18      totalBitSave ← totalBitSave + subConsequentBits -
19                     subConsequentMDLbits
20    end for
21  Return totalBitSave – Huffman (consequent)  // Eq. 2
```

2.7 Experimental Evaluation

To ensure that our experiments are reproducible, we have built a website which contains all data/code/raw spreadsheets for the results, in addition to more experiments that are omitted here for brevity [34]. The visualization of the rules suffers from space limitations/BW formatting; we encourage the reader to view high-resolution color versions at [34].

We provide two sources of evaluation for *quality*. In some cases, as in "The Raven" example above, we show the rules are meaningful by considering the annotation available by external labels of some kind. In the more general case we use the Euclidean distance between our predicted consequent and the F matching locations where the rule fired, a value we denote as F_{error} (this is essentially the root-mean-squared error). Because this number is difficult to interpret by itself, we do the following: On the same testing set, using the same consequent, we fire the rule *randomly F* times and measure the Euclidean distance between our predicted consequent and the F random locations. We denote this value as R_{error} (which is averaged over 1,000 random runs). Our reported measure of quality then is just $Q = \frac{F_{error}}{R_{error}}$. Values close to one suggest our rules are no better than random guessing and values significantly less than one indicate that we are finding true structure in the data. For all experiments, except where otherwise stated, the maxlag parameter is set to zero.

For completeness we do compare to some other methods (cf. Section 2.7.8). However, as noted in the introduction it is widely accepted that purported solutions do not perform above chance levels [6][23][32][33].

2.7.1. Finding Rules in Bird Vocalization

We consider the task of finding rules in Zebra finch vocalizations (in MFCC space). Such rules may help weight in on the "nature vs. nurture" debate [15], but more importantly for us here, show that we can learn robust accurate rules from complex and noisy datasets.

The vocal learning lab at Hunter College provided recordings of Zebra finches singing (~one minute) every ten days, from day 40 to 100 (post hatching). Starting from day 40,

we split data into a train (first 30-sec) and test set. Our algorithm finds several high quality rules, one of them shown in Figure 10.

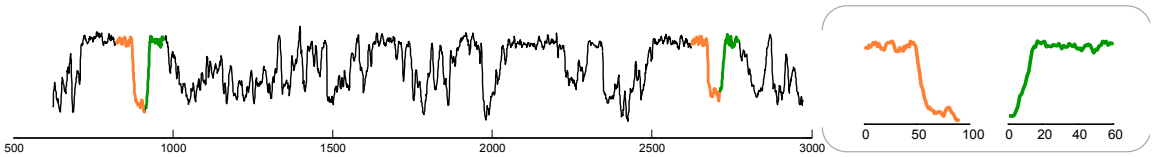


Figure 10: *left*) 25 seconds of zebra finch vocalization from the day-40 training data set. The discovered locations (orange/**bold**) are used to find a rule (*right*).

The rule shown in Figure 10.*right* looks plausible, but does it generalize to the test set? In Figure 11 we show one rule firing on an excerpt of the test set.

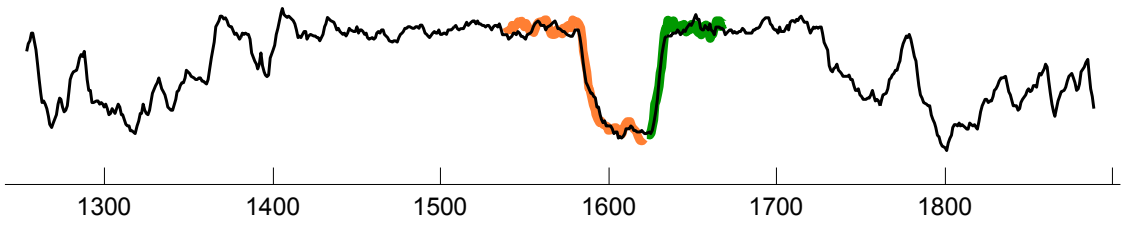


Figure 11: The rule learned in Figure 10 fires (bold/orange) on the test set of day 40 (only an excerpt is shown). The *Q* (cf. Section 2.7) for the fired rule is 0.33, suggesting high accuracy.

It is interesting to ask if the discovered rule generalizes over time, or if the bird's song evolves (i.e. *concept drift*). To test this we apply the learned rule in Figure 10 to the singing of the zebra finch on day 50.

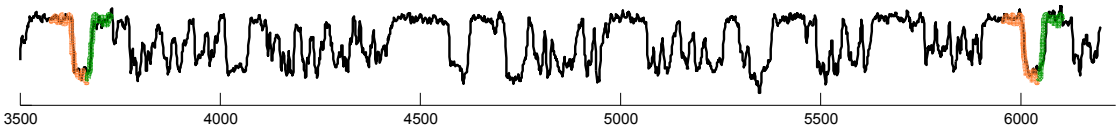


Figure 12: The rule learned in Figure 10 is applied to the same Zebra finch ten days later. The *Q* for the left and right instances are 0.19 and 0.40 respectively.

The low *Q*-scores in Figure 12 indicate that the rule discovered on day 40 (Figure 10) still generalizes. In Figure 13 we repeat the same experiment for day 100.

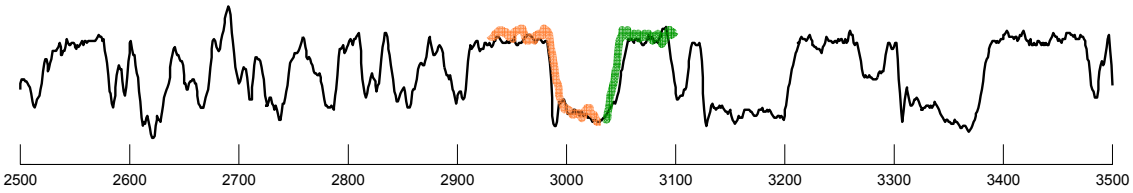


Figure 13: The rule learned in Figure 10 is applied to the singing of the same Zebra finch sixty days later.

Here we find that while the rule does predict the future much better than chance, the song seems to have undergone some modifications. This finding is consistent with the literature which suggests that young birds vocally improvise until about 90 days, after which the song "crystallizes" [2]. In [34] we show many addition experiments in this domain, and allow the reader to actually *hear* the data/rules.

2.7.2. Finding Rules in Energy Disaggregation

A home-based intelligent energy conservation system needs to know what appliances (or loads) are being used in the home and when they are being used in order to provide intelligent feedback or to make decisions that can reduce costs. The AMPds, Almanac of Minutely Power dataset, contains one year of such data that includes eleven measurements at one-minute intervals for twenty-one sub-meters [19].

For our experiments we consider a single meter into which the Fridge, Dishwasher, Clothes Washer and Clothes Dryer are plugged in to. We run our rule discovery algorithm on the first month of the data and find the rule shown in Figure 14.*right* as one of the top rules. Then we apply it to our test set and show one of the firings in Figure 14.*left.*

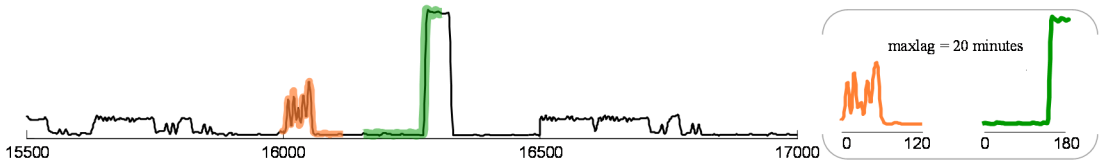


Figure 14: *right*) One of the top rules learned from the first month of the AMPds data set. *left*) One firing of the learned rule in the right which contains a 20 minute lag between its antecedent and consequent.

The antecedent and consequent in Figure 14.*right* correspond to the **Clothes Washer** and **Clothes Dryer** respectively. The discovered rule in Figure 14.*right* can be interpreted as: *when the tenant uses the **Clothes Washer**, after a while they will run the **Clothes Dryer**.* It is obvious that the tenant may immediately run the dryer or may spend some time doing something else first. Therefore in this case a non-zero maxlag must be allowed. For example the rule fired in Figure 14.*left* contains a 20 minute gap between its antecedent and consequent. Other firings of the rule contained different values. In order to assure we capture most of the firings, we allowed a 100 minute maxlag. Our algorithm to allow maxlag is described in Section 2.6.3.

An examination of the data suggests that our algorithm did not report any false positives for this experiment; however we *did* observe some true negatives. Most of these omissions may be attributed to the fact (as visually hinted at in Figure 14.*right*) that the **Clothes Washer** patterns are complicated and polymorphic. That is to say, the patterns depend on many settings of the washing machine (whites/colors/delicates, wash/rinse/spin etc). Automatically generalizing to handle such situations is ongoing work.

2.7.3. Finding Rules in an Activity Data Set

We consider a benchmark data set that contains daily activity telemetry [25]. The data is comprised of recordings of four subjects wearing seven inertial measurement units

(IMUs), performing sequentially a pre-defined set of activities. Each subject created five recordings, which we randomly divided into disjoint train/test partitions. In this experiment, we consider only the data generated by the IMU mounted on the right upper arm (near the bicep). Figure 15 shows one of the top rules learned from a recording of a subject.

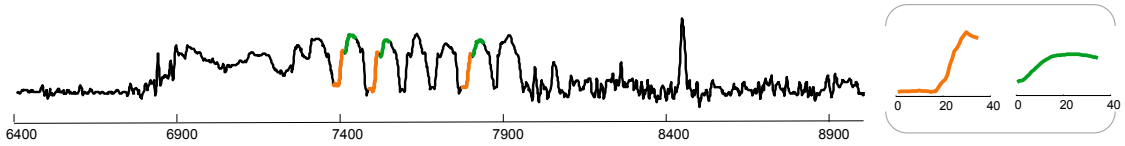


Figure 15: *left*) A subsequence from a recording of a subject that contains a rule discovered by our algorithm. The discovered locations (orange/**bold**) are generalized by our algorithm into a rule (*right*).

To test if the discovered rule generalizes to other instances of the activity, we applied the discovered rule in Figure 15 to other recordings of the same subject. Figure 16 shows the rule firings found in the test recording.

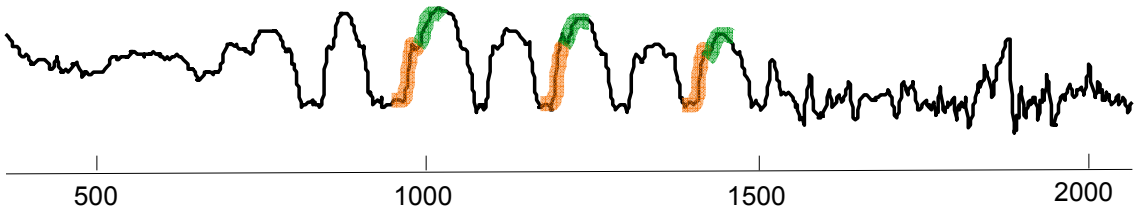


Figure 16. Three instances of the rule shown in Figure 15 were discovered in the test set. The Q for the three instances from left to right are 0.22, 0.10 and 0.41 respectively.

According to the labels provided with the dataset, the rule discovered is a part of the activity: *drinking from a cup while standing*. However, this information is not enough to fully interpret the rule. Thus, to better understand the rule, we reproduced the data by having an actor wear an IMU on the same part of the body. We recorded the actor drinking from a cup using both the IMU (at 100Hz) and a camera to capture simultaneous video.

Figure 17.*top* shows some stills from the video (The complete video can be found at [34]). The time series of the complete activity from the IMU is shown in Figure 17.*bottom*.

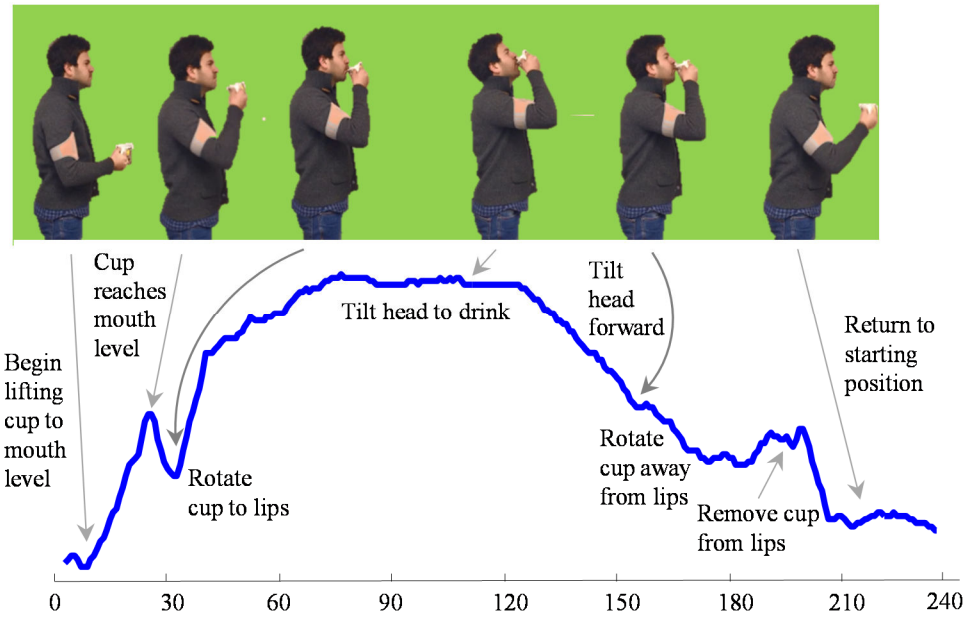


Figure 17: *bottom*) The IMU data of an actor drinking from a cup. *top*) Stills from a video aligned with the IMU.

Our data is very similar to the original benchmark data (our actor may have a different physique, mannerisms etc), and gives us some hints to understand the rule we discovered. As shown in Figure 17, the rule appears to describe the first half part of the drinking activity: *a lifting of the cup to the mouth is immediately followed by a slightly tilting head to drink from the cup.*

2.7.4. Finding Rules in NASA Telemetry Data

The NASA valve data set consists of 36 events of interleaved nominal and erroneous solenoid voltage measurements recorded from Marrotta series MPV-41 valves as they are tested in a laboratory [10]. Figure 18 shows one of the top rules learned from this time series where the first peak in Figure 18.*left* is that of a failed solenoid.

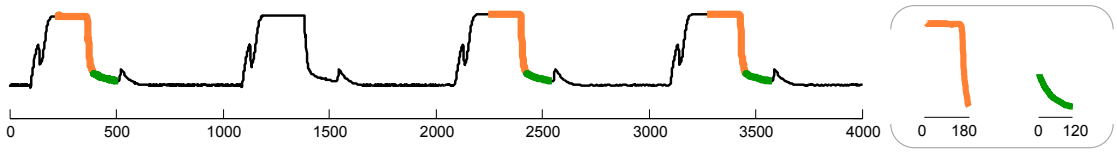


Figure 18: *left*) A snippet of the NASA data. *right*) The first ranked rule learned which characterizes a nominal discharge.

We applied the discovered rule in Figure 18 to the test set and found the rule fires four times. Figure 19 shows the rule instances fired on the test set.

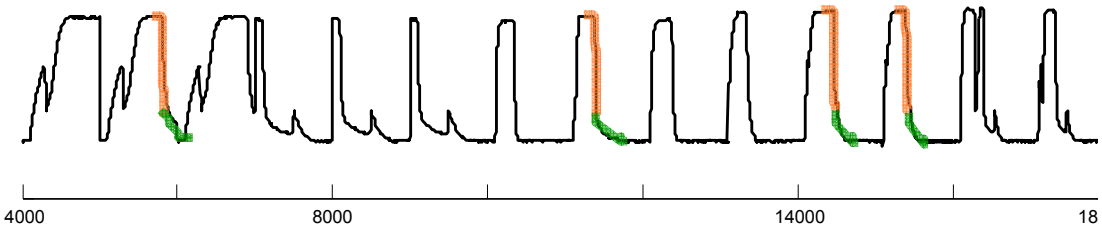


Figure 19: The rule discovered in Figure 18 fired on the test set.

This rule appears to describe a normal solenoid discharge event: *a rapid decrease in the current is immediately followed by a slight ramp and gradual, complete discharge*.

Because of the variety of malfunction events in contrast to the homogeneity of normal solenoid readings in this data set, this rule learned from successful tests achieves the highest MDL score as well as very low average Q-value of 0.10. Note that the rule *fails* to fire in several locations in Figure 19. According to the domain experts [10], most of the non-firing locations correspond to a valve assembly miss-cycled for which the solenoid still experienced a nominal discharge. Apart from these outliers, all normal solenoid trials were detected with this rule. Thus, in a sense, we can imagine using the *negation* of the rule firing as an anomaly detector.

2.7.5. Finding Rules in Insect Behavior

Insects in the order *Homoptera* feed on plants using their stylet (feeding tube) to suck out sap. This behavior is damaging to the plants, causing billions of dollars of damage to crops each year [13]. Many research labs use an Electrical Penetration Graph (EPG), the apparatus sketched in Figure 20, to record such insect behavior.

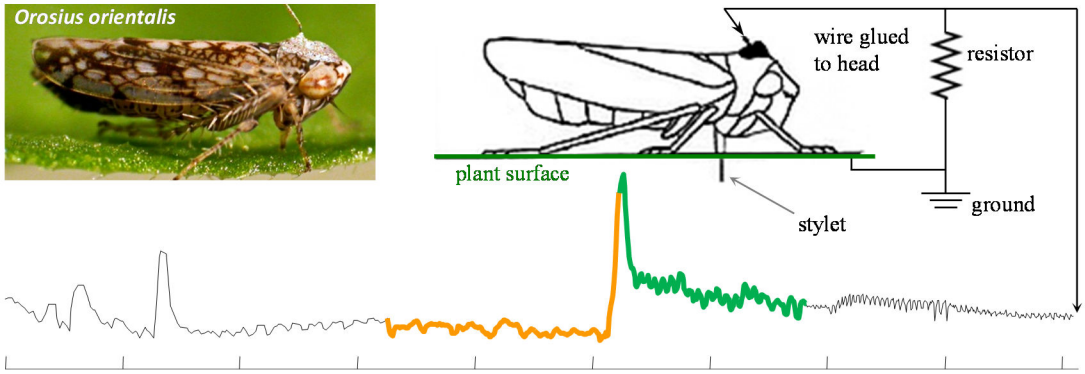


Figure 20: *left*) A tethered brown leafhopper. *right*) A schematic diagram of the apparatus. *bottom*) A rule discovered by our algorithm (shown in a small snippet of training data for context).

We examined a ten minute snippet of data and found several very high-scoring rules. To see if the rules generalized to *unseen* data we evaluated them on holdout data. For example, the rule shown in Figure 20, fired three times on holdout data, resulting in a mean Q of 0.35. Figure 21 shows two firings of the rule.

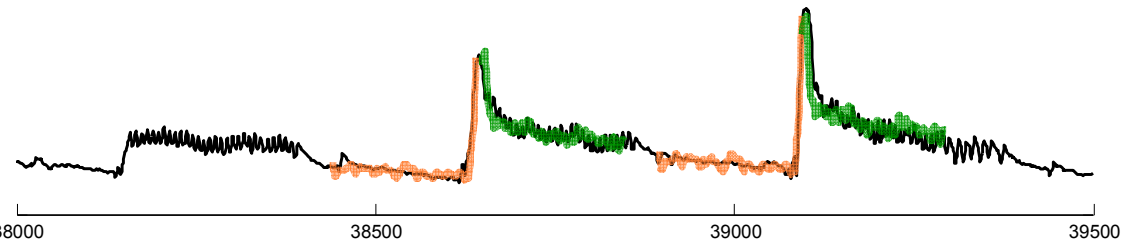


Figure 21: The rule discovered in Figure 20 fired on holdout data, the Q for the two instances are 0.39 and 0.35 respectively.

Dr. Elaine Backus, a co-inventor of the EPG apparatus, was kind enough to interpret the rule for us. It can be read as: IF *we see a probing behavior* (pause, then dramatic

increase) THEN *we will see pathway behavior* (oscillations indicating search for vascular tissues).

While this rule is comprised of simple shapes, in Figure 22 we show additional rules discovered that show that the antecedent and/or consequent can be even more complex.

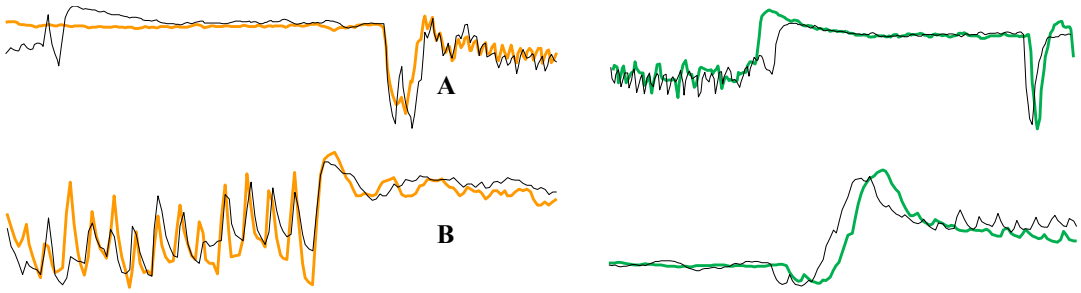


Figure 22: Two high confidence rules discovered in EPG data. Rule B has been interpreted as: IF *we see periods of phloem ingestion followed by a pause* THEN *we will see salivation/tasting*.

2.7.6. An Important Sanity Check

We conducted a sanity check experiment that is very simple, but would nevertheless had demonstrated the problems with the approaches in [6][32] (as [14] also demonstrated, but in a different context). We reran all the experiments above, making a single change, which was to replace the data with *random walk data*. In no such case does our algorithm find any rules. This finding bolsters our confidence that our scoring function is valid.

2.7.7. Time Complexity

If Maxlag is set to zero, then the time complexity for our algorithm is O($n$log$n$). Allowing a Maxlag increases this to O($n$log$n$ × |$maxlag$|). In essence, the time required by our algorithm is dominated by the speed of motif discovery, which fortunately has received a lot of attention in recent years [22][28]. We do not include explicit timing experiments

because in general the time needed for rule discovery is inconsequential. For example, the insect EPG data took several months to collect, so the few minutes our algorithm needed to find rules is not likely to be a burden. Likewise, the Zebra finch data reflects years of painstaking work, so the few minutes our algorithm needs is simply negligible.

2.7.8. On Comparisons to Rival Methods

At least two of the most referenced methods for rule discovery in time series operate on a Piecewise Linear Approximation (PLR) representation of the data [23][32]. In this section we go beyond simply comparing to these methods, we will argue that *any* method based on PLR is very unlikely to ever be able to find rules in the general case. Note that while the papers in question are heavily cited [23][32], these are high-level citations, we are not aware of any paper that actually uses either work to find rules. Thus our claim does not contradict any deeply held opinion of the community.

Recall that [23], was only evaluated for *speed*. No evidence was presented that the algorithm could actually find generalizable rules in time series. Also recall that while [32] reported approximately 68% "*correctness of trend prediction*" on real financial data [32] they also got approximately 68% correctness of trend prediction on *random walk* data [33]. This suggests their original results did not outperform random guessing (at least on the domains tested).

Our claim is based on the simple observation that while PLR is a very useful compression technique for time series, it is *not good* at capturing the semantic patterns in the data. To demonstrate this, let us revisit the data shown in Figure 7 in this dissertation. We have converted this dataset into PLR representations with varying levels of precision

and plotted it in Figure 23. While [23][32] use fast approximate techniques, we take the time to use the optimal segmentation algorithm.

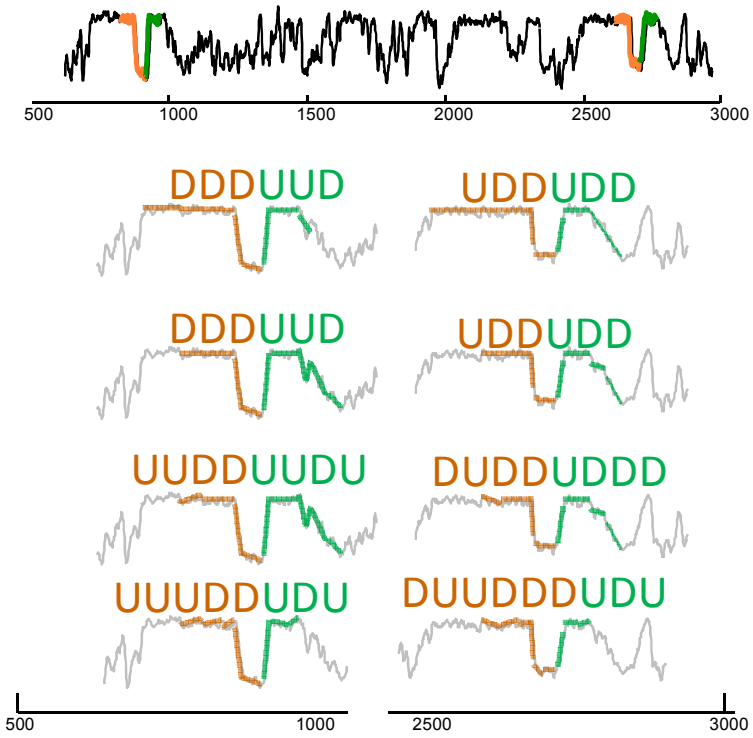


Figure 23: *top*) The time series shown in Figure 7. *Bottom*) The two sections containing the rule discovered by our algorithm converted into PLR with various levels of approximation, annotated by strings indicating the signs of the slope of the segments, **U**p or **D**own.

Let us consider a simple rule finding algorithm based on PLR, very similar in spirit to [23][32]. We can measure the slope of the line segments, denote positive slopes as **U**, and non-positive slopes as **D**. We could then look for rules in the form of

$$DDD \rightarrow UUD$$

That is to say, IF we see a segment going **U**p, followed by two segments going **D**own, followed by a final Segment going **U**p, THEN we will see an **U**p-**D**own pattern.

As plausible as this idea seems, it will not work on the data in Figure 23. The problem is that two subsequences can have a very small Euclidian distance, but have very different

segmentations, including the *number* of segments. For the latter issue, we could force two regions to have the same number of segments, but that does not solve the problem.

Note that the problem of similar subsequences resulting in different symbolic words *seems* to get slightly better as we use a coarser approximation. However, with the coarser approximation we lose the power to discriminate among very different subsequences. For example, about 1/64 of the subsequences in *random walk* have the pattern ***DDDUUD***, but we are wise in not attaching any significance to this fact.

More generally, we make the following observations.

1. Two time series can be arbitrarily *different*, yet have the *same* PLA representation.

2. Two time series can be differ only by an arbitrarily small epsilon, yet have completely different PLA representations.

We regard the latter fact as the death knell of any attempt to find rules using PLA. While the first point is a well understood implication of dimensionality reduction in general, the latter point is less well appreciated, so we will give a visual existence proof.

Let use construct a time series A, consisting of one hundred '0's, {0,0,0,0,..., then a linear rise to 100, ...0,1,2,3,4,.., 98,99,100,.. then one hundred '100's. The resulting time series is shown in Figure 24.*top* in a red/**bold** line.

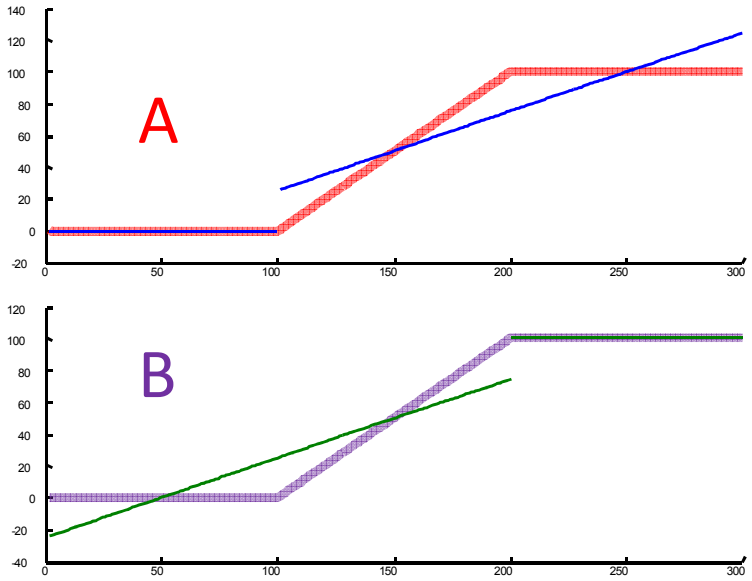


Figure 24: A demonstration of the brittleness of PLA for understanding the semantics of a time series.

We then make a copy of A, changing the first value to '1' instead of '0'. We call this time series B, and show it in Figure 24.*bottom*. Note that at this resolution it is not possible to see the difference.

We now segment A and B using the optimal segmentation algorithm, specifying two segments, again Figure 24 shows the results. Imagine we have a rule discovery algorithm trying to reason with just the PLA representation. Such an algorithm is convinced that while A is constant in the region from $A_{[1:100]}$, B has a significant positive slope in that region. If the rule engine is ignoring the actual slopes, and only considering *where* "events" happen, it would reason that there is an "event" at about location 100 in A, but B is unchanging here.

The reader may argue that this problem is due to the fact that we have forced too low a dimensionality on the time series here. We *did* use low a dimensionality simply for visual clarity, it is trivial to create examples with very low residual error that have the same

problem. In any case, allowing more segments allows the problem of *fragmentation*. Consider again the finest (highest dimensionality) segmentation shown in Figure 23, its output, UUUDDUDU vs. DUUDDDUDU, is hardly conducive to pattern matching or rule discovery.

It is important to note that these issues are not pathological things that can happen, but are almost never observed. They are observed very *frequently* in Insect Behavior, NASA Telemetry Data, Bird Vocalization and Energy Disaggregation, and several times in the Activity Data Set.

The reader may wonder if there is a fix for these issues. There *may* be, but to the best of our knowledge there is currently none in the literature. We note that these issues (and more we omitted for brevity) where pointed out almost two decades ago in [26].

The previous page and a half was a rather long preamble, to help explain the experimental results:

We compare our algorithm to the two (very different) PLA based approaches in [23] and [32]. To be as fair to them as possible we tested over many combinations of reasonable parameters, using both human-guided and brute force search for the best parameters. For Bird Vocalization, Energy Disaggregation NASA Telemetry Data and Activity Data Set, the best results for both approaches had Q values at the default rate (consistent with random guessing). For Insect Behavior, we showed the five highest ranked rules to our expert, shuffled together with 15 completely *random* rules, and asked her to pick out the three she thought most likely to represented true insect behavior. She noted that none of the 20 looked plausible, but when force to pick three she chose completely *random* rules.

We also compared to [6], because is it the most highly cited paper (by a huge margin) to have the words "time series" or "temporal" and "rules" in its title or abstract. However, we note that since its publication, at least 6 independent researcher have claimed that it *can not* work [7][4][14]. We do not wish to weigh in on this, we simply fatefully reimplemented the algorithm and tested it. The algorithm requires several parameters, including the *K* in *K*-means. Here we simply exhaustively tested every K from 2 to length of time series divided by two, and report only the *best* value. For Bird Vocalization, NASA Telemetry Data and Activity Data Set, the best results had *Q* values at about the default rate. For Energy Disaggregation, the result was always significantly *below* the default rate, perhaps due to fact that most of these data contains no noise.

The algorithm ranks rules by J-measure. For Insect Behavior, we again we showed the five highest ranked rules to our expert, shuffled together with 15 completely *random* rules (different to the 15 random rules noted above), and asked her to pick out the three she thought most likely to represented true insect behavior. Once again she noted that none of the 20 looked plausible, but when force to pick three she chose two *random* rules and one that had achieved the 4th highest J-measure, a result consistent with chance.

2.8 Conclusions

We have introduced a technique for finding rules in time series which leverages of recent advances in time series motifs discovery to provide a tractable search. Our novel application of MDL to time series rule discovery allows us to meaningfully rank and compare varied length rules, and rules with different levels of "support". Our rule

representation is expressive enough to allow rules with different length antecedents/consequents/lags/firing thresholds, but at the same time does not require extensive human intervention or tweaking.

There are many avenues for future work. On some datasets, Dynamic Time Warping may be more robust than the Euclidean distance, adapting to the concept drift that will be inevitable in some applications, and for some domains scalability to massive datasets remains an issue.

It may be possible to generalize the rule representation to allow more expressive logical connectives, i.e.

$$D(R_a, W_1) < t_1 \texttt{ AND } D(R_b, W_2) < t_2 \rightarrow R_c \text{, or}$$

$$D(R_a, W_1) < t_1 \texttt{ OR } D(R_b, W_2) < t_2 \rightarrow R_c$$

However this would require significantly more training data, and perhaps additional efforts to guard against overfitting. Such flexibility would allow a rule to consider antecedents from two different sources. For example a certain pattern observed in `humidity` `AND` other pattern observed in `temperature` might predict a particular future pattern in `humidity`.

Finally, unlike time series classification or clustering [29], there are currently no standard benchmarks for time series rule discovery. We plan to repair this omission, and invite the community to donate additional challenging datasets for which the ground truth is known, and archiving them [34].

Chapter 3: Generalizing Dynamic Time Warping to the Multi-Dimensional Case Requires an Adaptive Approach

Virtually all attempts to improve time series classification in the last two decades have focused on the single-dimensional case, with the assumption that the generalization to the multi-dimensional case is trivial. There are two obvious ways DTW can be generalized to the multi-dimensional case. For clarity, we refer to the two methods as DTW_D and DTW_I (with $_D$ standing for Dependent and $_I$ for Independent).

The vast majority of researchers seem to think that it makes no difference which method is used, as evidenced by the fact that they usually do not explicitly bother to *tell* the reader.

Our work has two implications for the time series research community: we free researchers/implementers from having to decide which technique to use for their problem; and, because *error*(DTW_A) will be *minimum*[*error*(DTW_D), *error*(DTW_I)], they can use our method safe in the knowledge that they did not choose the suboptimal method.

However, this greatly understates the case, as the correct inequality implied by our work is the more unintuitive *error*(DTW_A) ≤ *minimum*[*error*(DTW_D), *error*(DTW_I)]. That is to say, on some datasets our method can be significantly more accurate than *either* of the rival methods.

This chapter is organized as follows: In Section 3.1, we introduce the definitions and notations used in this chapter. We discuss the intuition behind the two possible methods

for calculating DTW, *dependent* and *independent,* in Section 3.2. We then illustrate our proposed idea in Section 3.3. We present a detailed empirical evaluation of our ideas in Section 3.4. The related work and background is discussed in Section 3.5. Finally, in Section 3.6, we offer conclusions and directions for future work.

3.1 Definitions and Background

We present the definitions of key terms that we use in this work. For our task at hand, each object in the dataset is a *time series*.

***Definition* 1**: A *Time Series* $T = t_1, t_2, \ldots, t_n$ is an ordered set of real values. The total number of real values is equal to the length of the time series. A dataset $\boldsymbol{D} = \{T_1, T_2, \ldots, T_M\}$ is a collection of M such time series.

We are interested in multi-dimensional time series:

***Definition* 2**: *Multi-Dimensional Time Series* (MDT) consist of M individual time series ($M \geq 2$) where each time series has n observations:

$Q_1 = q_{1,1}, q_{2,1}, q_{3,1}, \ldots q_{n,1}$

$Q_2 = q_{1,2}, q_{2,2}, q_{3,2}, \ldots q_{n,2}$

...

$Q_M = q_{1,M}, q_{2,M}, q_{3,M}, \ldots q_{n,M}$

If we wish to compare two time series, we could use the ubiquitous Euclidean distance. However, the DTW distance subsumes the Euclidean distance as a special case and has been shown to be significantly more accurate in virtually all domains [42][63]. Unlike the Euclidean distance's strict *one-to-one* alignment, DTW allows a *one-to-many* alignment, as

illustrated in Figure 1 To align sequences using DTW, an n-by-n matrix is constructed with the (i^{th}, j^{th}) element being the squared Euclidean distance $d(q_i, c_j)$ between the points q_i and c_j. A warping path P is a contiguous set of matrix elements defining a mapping between Q and C. The t^{th} element of P is defined as $p_t = (i,j)_t$, so we have:

$$P = p_1, p_2, \ldots, p_t, \ldots, p_T \quad n \le T \le 2n - 1$$

The warping path that defines the alignment between the two time series is usually subject to several constraints: the warping path must start and finish in diagonally opposite corner cells of the matrix, the steps in the warping path are restricted to adjacent cells, and the points in the warping path must be monotonically spaced in time. In addition, virtually all practitioners using DTW also constrain the warping path in a global sense by limiting how far it may stay from the diagonal [42][58]. A typical constraint is the Sakoe-Chiba Band which states that the warping path cannot deviate more than R cells from the diagonal [42][58][63]. This constraint prevents pathological warpings (for example, a single heartbeat mapping to ten heartbeats) and is at the heart of the LB_{Keogh} lowerbounding technique, which is used in virtually all speedup techniques for DTW [35][42].

While there are exponentially many warping paths that satisfy the above conditions, we are only interested in the path that minimizes the warping cost:

$$(1) \qquad DTW(Q,C) = \min\{\sqrt{\sum_{t=1}^{T} p_t}\}$$

This path can be found using dynamic programming to evaluate the following recurrence, which defines the cumulative distance $D(i,j)$ as the distance $d(i,j)$ found in the current cell and the minimum of the cumulative distances of the adjacent elements [54][61]:

(2) $D(i,j) = d(q_i, c_j) + \min\{D(i-1, j-1), D(i-1, j), D(i, j-1)\}$ in which: $d(q_i, c_j) = (q_i - c_j)^2$

While this recursive function is elegant and can be tersely implemented, in practice the community uses an *iterative* algorithm, which is faster and amiable to various early abandoning optimizations [35][42]. Moreover, the iterative algorithm implementation only constructs and considers a single column of the matrix at a time and, thus, has a space complexity of just O(n).

The Euclidean distance between two sequences is a special case of DTW, where the t^{th} element of P is constrained such that $p_t = (i,j)_t$, $i = j = t$. This review of DTW is necessarily brief; we refer the interested reader to [35][42][54][61] for more details.

3.1.1. Generalizing to the Multi-Dimensional Case

The DTW distance, as formalized in Eq. 2, is applicable to only single-dimensional time series, leaving open the question of how to extend it to the multi-dimensional time series (MDT) case. Consider both Q and C as two M-dimensional time series; we show two possible approaches for doing this, DTW_I and DTW_D:

***Definition* 3**: DTW_I is the cumulative distances of all dimensions independently measured under DTW. If $DTW(Q_m, C_m)$ is defined as the DTW distance of the m^{th} dimension of Q and the m^{th} dimension of C, we can write DTW_I as:

$$DTW_I(Q,C) = \sum_{m=1}^{M} DTW(Q_m, C_m) \quad (3)$$

In Eq. 3, each dimension is considered to be independent, and DTW is allowed the freedom to warp each dimension independently of the others. The case when M is two was shown in Figure 1.*b*.

We can also compute the multi-dimensional DTW in a manner that forces all dimensions to warp identically, in a single warping matrix. In other words, the independence of dimensions is no longer allowed, and we assume mutual dependence between all dimensions. We define DTW_D as:

***Definition* 4**: DTW_D is calculated in a similar way to DTW for single-dimensional time series (Eq. 2), except that we redefine $d(q_i, c_j)$ as the cumulative squared Euclidean distances of M data points instead of the *single* data point used in the more familiar one-dimensional case. Formally, if $q_{i,m}$ is the i^{th} data point in the m^{th} dimension of Q and $c_{j,m}$ is the j^{th} data point in the m^{th} dimension of C, we replace $d(q_i, c_j)$ in (Eq. 2) with:

$$d(q_i, c_j) = \sum_{m=1}^{M} (q_{i,m} - c_{j,m})^2 \quad (4)$$

To make our distance measure invariant to scale and offset, we need to *z-normalize* each dimension of the time series before computing their DTW distance. As demonstrated in [51], even tiny differences in scale and offset rapidly swamp any similarity in shape. Note that this allows us to meaningfully compute either variant of the multi-dimensional DTW, even if the individual dimensions are not commensurate or are in different units, such as *accelerations* and *rotations*.

Using both DTW_D and DTW_I distance measures to classify a time series exemplar, T, four different cases may occur:

1. T gets correctly classified by both DTW_I and DTW_D.
2. T gets misclassified by both DTW_I and DTW_D.
3. T gets classified correctly by DTW_I but misclassified by DTW_D.

4. *T* gets classified correctly by DTW_D but misclassified by DTW_I.

We are only interested in cases 3 and 4. We call such exemplars *iSuccess* and *dSuccess*, respectively:

***Definition* 5**: *iSuccess* is the set of time series exemplars that are classified correctly under DTW_I but misclassified under DTW_D.

***Definition* 6**: *dSuccess* is the set of time series exemplars that are classified correctly under DTW_D but misclassified under DTW_I.

Having reviewed the necessary formal definitions, we are now in a position to introduce our observations about the relative merits of DTW_D and DTW_I.

3.2 Observations

We begin with some informal notation. We say a dataset is "in *D*" if we expect DTW_D to achieve higher accuracy and "in *I*" if we anticipate DTW_I will be more accurate. In the introduction we claimed that there are datasets in which we expect DTW_D to outperform DTW_I and vice versa. A natural question to ask is under what conditions we can expect each of these methods to be superior. As we shall see, one of the fundamental contributions of this work is to make this question moot by producing an algorithm that is always *at least as good* as the better choice. Nevertheless, it is instructive to ask and attempt to answer this question.

Assume that the data in question corresponds to an *event*. An event could be an arrhythmic heartbeat, the writing of the letter 'Z,' a golf swing, a bird call, a self-driving car parallel parking, etc. Further assume that we have multi-dimensional time series

recordings of such events. It is possible that each dimension is simply recording two views of the same physical phenomena. For example, consider the MFCC coefficients of the bird call shown in Figure 25.

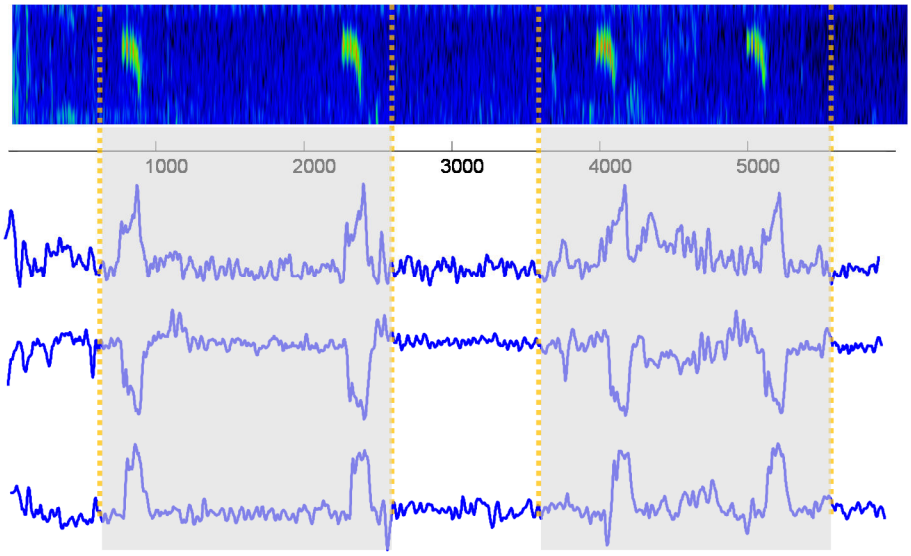


Figure 25: Three coefficients from the MFCC space of a Southern Chestnut-Tailed Antbird (*Myrmeciza hemimelaena*). This bird's call is typically transcribed as "*klee-klee*" [45]; thus, the above shows two calls, the second being significantly briefer.

It is clear that while the coefficients are (somewhat) independent in the Y-axis values they can take on, they are *not* independent in the time axis. In the second bird call all the relevant peaks and valleys move by *exactly* the same amount in the time axis. Because of this structure, we strongly expect that this dataset is in D. In contrast, consider the *event* shown in Figure 26 of a cricket umpire signaling `TV-Replay`, in which the umpire traces a rectangle representing a television screen.

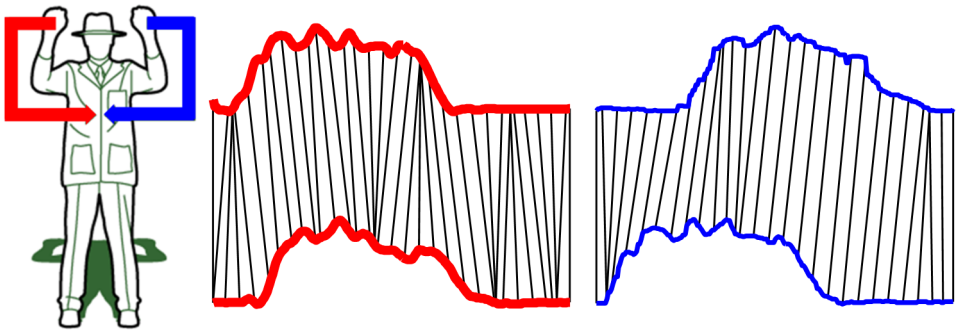


Figure 26: (*left*) A cricket umpire signaling "`TV-Replay`." (*right*) The Dynamic Time Warping distance between two time series of the X-axis from the right and left hand.

Here we have measured a multi-d time series that consists of the X-axis acceleration of sensors worn on each wrist. Note that, in contrast to the bird call example, the two time series are very unlikely to be *perfectly* dependent in how they evolve in the time axis. Try as he might, the umpire could not move both hands in a *perfectly* symmetric fashion. There are at least two possible and non-exclusive sources of difference:

- **Lag**: Here, we imagine that the umpire favors his dominant hand, and the other hand follows at a more or less constant speed, but a fraction of a second behind.

- **Loose Coupling**: Here, the *event* does cause two or more things to happen, both of which are recorded as a time series, but there is more freedom in the performance of the event. For example, if the *event* is the umpire wishing to signal a Leg-Bye, he will tap his raised knee with this hand. However, while he typically uses his dominant hand to do this, he may touch *either* knee. Moreover, his "free" hand may rest by his side, or he may raise it, and even waive it slightly, to drawn attention to the fact he is making a signal. This variability in performance means that two wrist-worn sensors are only very loosely coupled for this event.

In the next section, we will explicitly test the effects of these factors with some experiments. To do this we consider a dataset that we are sure is in *D*, and then we synthetically add increasing amounts of lag and loose coupling to see if this would move the dataset into *I*.

We consider a *handwriting* dataset, which we are confident is in *D*. Because we are considering the X and Y accelerations of the point of the writing tip of a pen, the two dimensions are *physically* coupled. Here, an *event* is the production of one of the twenty-six lower-case letters. We estimate the error rate of classification by randomly dividing the 5,000 objects into a stratified 1,000/4,000 train test split thirty times and reporting the average error rate.

It is critical to note that, in this dataset, if we were considering only the one-dimensional case, our synthetic modifications of the data would make essentially no difference to the distances DTW returns, or the overall error rate. As shown in Figure 27, even a huge change in the lag makes almost no difference to the single-dimensional DTW case.

Thus, all effects shown in the next two sections are due not to the effects of modifying the data objects per se, but to the effect this has on DTW_D and DTW_I.

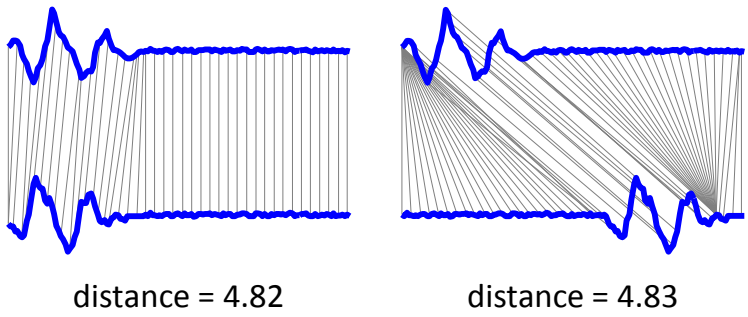


Figure 27: For the one-dimensional case, adding lag to one of the time series can change the warping drastically, without changing the distance by a significant amount.

3.2.1. The Effect of Lag

We induce lag in one of two ways. First, for each object we add *Random Lag* of *K* by shifting *just* the Y-axis by an amount randomly chosen from the range [0, *K*]. Second, we add a *Fixed Lag* variant by shifting *just* the Y-axis by increasing values of *K*. For clarity, in this variant all objects will have the same lag of exactly *K*. In Figure 28 we show the effect of varying the amount of *Random Lag* from zero to forty.

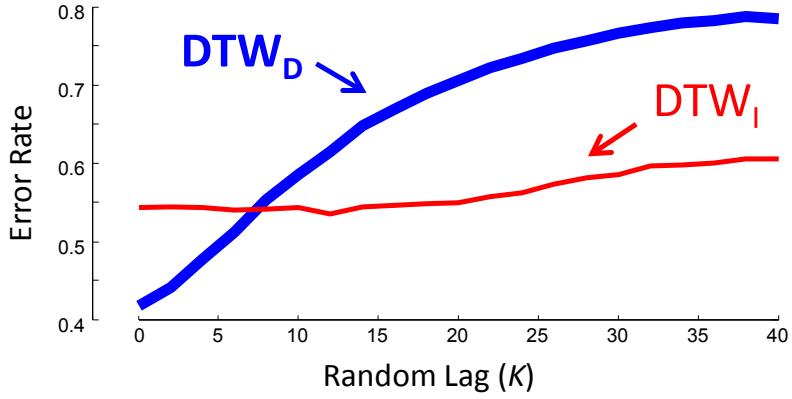


Figure 28: The effect of adding *Random Lag* on the classification accuracy of DTW_D and DTW_I.

In Figure 29 we show the effect of varying the amount of *Fixed Lag*, again from zero to forty.

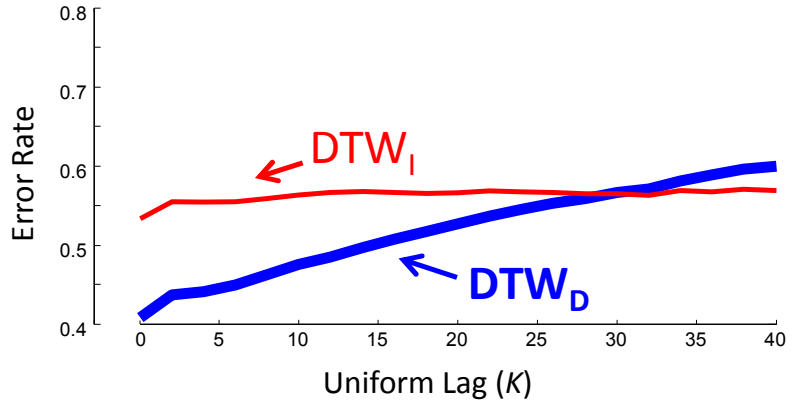


Figure 29: The effect of adding *Fixed Lag* on the classification accuracy of DTW_D and DTW_I.

If we consider just the values at either end of the range, this provides us with the first experimental evidence that datasets exist that are strongly in *D*, along with datasets (albeit *contrived* here) that are strongly in *I*. More generally, the trend lines in the figure confirm our claim that *Lag* is one of the elements responsible for datasets falling in *D* or *I*. Note that the effects are not small; the wrong choice of DTW_D or DTW_I here can almost double the error rate.

3.2.2. The Effect of Loose Coupling

We create synthetic loose coupling by adding increasing amounts of random warping to just the Y-axis of each exemplar. For brevity, and to enhance the flow of the presentation, we relegate the explanation (and the actual code) of how we do this to the supporting website [76]. However, we note that the modified data is plausible and realistic data. For example, if we use it to regenerate the original letters, they are readable and believable as a person's handwriting. Figure 30 shows the effect of increasingly loose coupling.

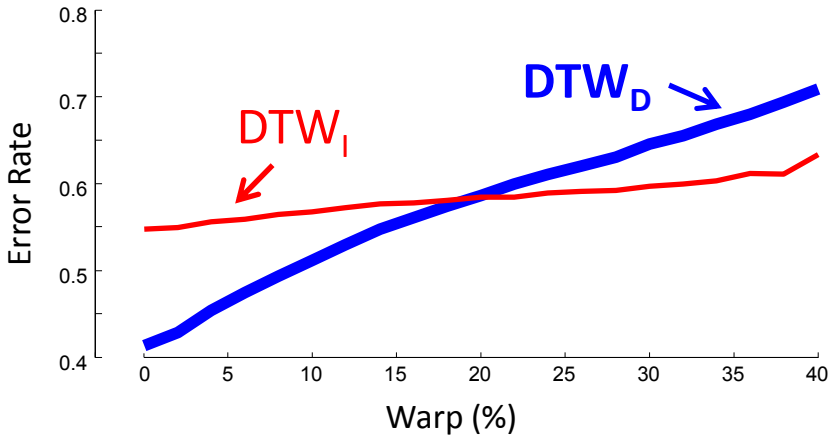


Figure 30: The effect of adding *Fixed Warping* on the classification accuracy of DTW_D and DTW_I.

Once again, these results provide us with evidence that some datasets are in *D* and some are in *I*, and that loose coupling can be a reason for this division.

3.2.3. Implication of Observations

At first blush we might interpret the above results as implying that all datasets lie on a spectrum between being strongly in *D* and strongly in *I*. If true, then the only task left to us is to discover where on the spectrum a dataset falls so that we can use the correct technique.

However, this idea has two difficulties. For some datasets we may not have enough training data to learn whether we are in *D* or in *I* with high confidence. The second issue is simply that the assumption that all *datasets* lie on such a spectrum misses a crucial point. It is possible that the suitability for DTW_D or DTW_I occurs at a *class-by-class* level, or even an *exemplar-by-exemplar* level, not at a *dataset-by-dataset* level.

It is easy to imagine such examples. Suppose we have accelerometers on both wrists of a tennis player, and our classification task is to label data into the following shot types {`serve` | `forehand` | `lob` | `other`}. For many exemplars we might expect DTW_I to work best, since the hands are generally loosely coupled in tennis. However, for some classes, such as the backhand, most players use a two-handed grip, temporarily coupling the two accelerometers. This would give us a *class-by-class*-level difference in the suitability of the warping technique. Moreover, some players, notably French professional Jo-Wilfried Tsonga, switch between one-handed and two-handed back-hand shots during the game. This would give us an *exemplar-by-exemplar* level difference in the suitability of the warping technique.

3.2.4. Further Discussion

The reader may wonder why we need DTW_D at all. It appears that DTW_D is just a special case of DTW_I, and therefore unnecessary. In other words, if both warping paths created by DTW_I happen to be the same, the results are logically equivalent to DTW_D. Since there is nothing preventing this from happening, one might imagine that DTW_D is simply subsumed as a special case of DTW_I, and the results above are an error or anomaly of some kind.

The reason why we need both DTW_D and DTW_I is subtle and underappreciated. When we perform a DTW_I calculation and find it produces a relatively small distance, it *may* have achieved this with radically different warpings for each axis.

In contrast, DTW_D must use the same warping for *both* dimensions. Thus, in a sense, the fact that it could achieve a small distance, *in spite* of the same warping constraint, is *extra* evidence of similarity.

We can illustrate this with the simple experiment shown in Figure 31. Here we have three 2-dimensional time series. Subjectively we would surely group them {{**A**,**B**},**C**}, as **B** is created by simply copying **A**, shifting the "patterns" one place to the right, and adding a tiny "bump" in the 3rd value of **B**. Nevertheless, if we cluster these objects with DTW_I, we get an extraordinarily unintuitive result, suggesting {{**A**,**C**},**B**}.

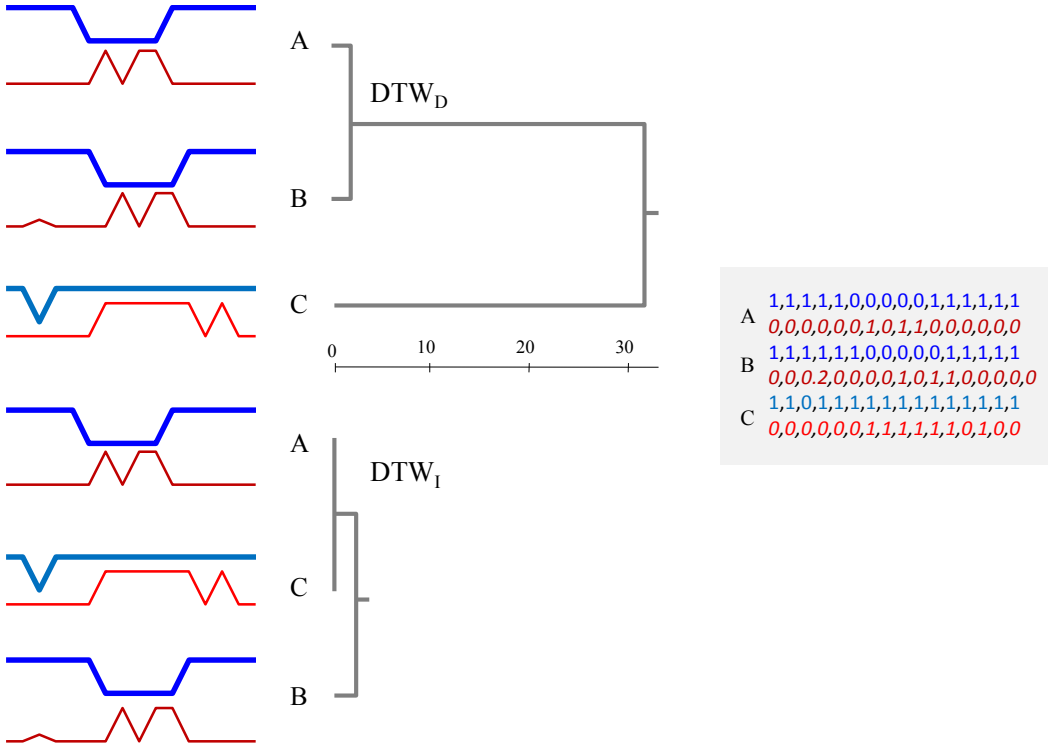


Figure 31: Three 2-dimensional time series (raw values shown in inset) single-linkage hierarchically clustered using DTWD (*top*) and DTWI (*bottom*). Here DTWD produces an intuitive clustering, closely linking A and B, whereas DTWI uses its greater representational flexibility to produce a bizarre result, finding A to be *identical* to C.

This may be easier to appreciate with an analogy. Imagine we have two distance measures, NAME_D and NAME_I, that measure the distance between a target person's name (say, "Anne Price") and the set of names we find in a small address book. The underlying measure we assume is a string edit distance. The NAME_I function calculates the most similar first and last names *independently*, perhaps finding "*Anne* Smith" and "Bob *Price*" in the address book to produce a distance of zero. In contrast, the NAME_D function calculates the distances *dependently*, that is to say, from the *same* individual. Imagine that NAME_I reports a distance of one, having found a person called '*Ann***a** *Price*'. While the

latter distance of one is greater than the former distance of zero, we would almost certainly be more impressed by the similarity of the latter.

The example illustrated in Figure 31 shows DTW_D outperforming DTW_I so forcefully that the reader may now wonder why we need DTW_I at all. As previously noted, DTW_D uses the same warping path for all dimensions to measure their distances. Therefore, DTW_D may not be a suitable distance measure for instances with lag, simply because of using only one warping path (recall the three diverse warping paths for each of the color channels shown in Figure 3.*top*). For exemplars including lag between dimensions, DTW_I is capable of measuring the distances independently and being invariant to the lag. We illustrate this in Figure 32.

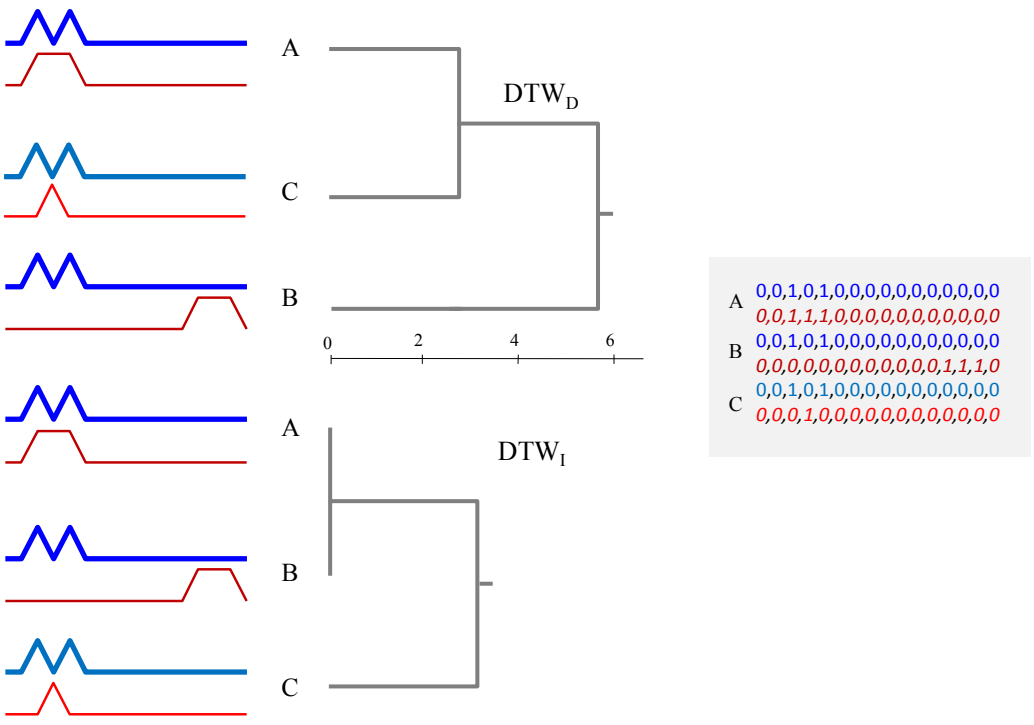


Figure 32: Three 2-dimensional time series (raw values shown in inset) single-linkage hierarchically clustered using DTWD (*top*) and DTWI (*bottom*). Here DTWI produces an intuitive clustering, closely linking A and B, whereas DTWD uses only one warping path to measure distances therefore producing a bizarre result, finding A to be *identical* to C. Compare to Figure 31.

Here instances **A** and **B** should belong to the same cluster since both are the same, except **B** includes a large lag in the second dimension (illustrated in red). As shown in Figure 32, DTW_D clusters **A** and **C** into the same subtree, simply because there is no lag in the second dimension of **C**. However, DTW_I correctly clusters **A** and **B** the same because the lag in the second dimension of **B** is ignored by DTW_I.

The reader may wonder if our observations are true but irrelevant, as we have only demonstrated our claims for contrived data up to this point. In our experimental section we give compelling evidence on several real-world datasets, but here we give a visually intuitive example. In Figure 33 we show two dimensions of the telemetry from an oil refinery. In the first day shown, these two time series have near-perfect correlation, suggesting a very tight coupling. However, in the first half of the next day a connecting valve is closed, and the two time series become almost completely uncoupled.

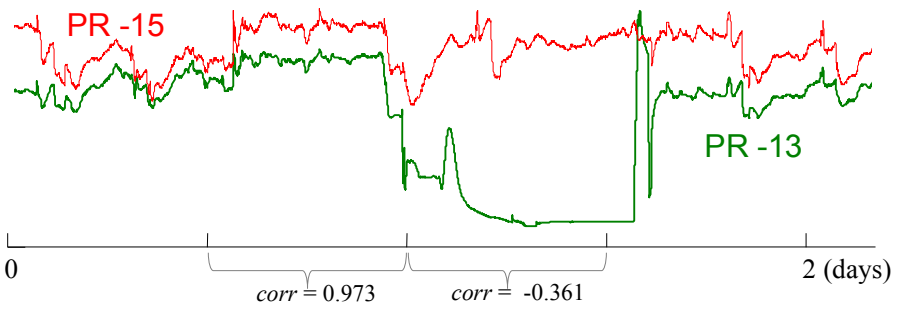


Figure 33: Two dimensions from a two-day sequence from the telemetry of a Delayed Coker. In the last twelve hours of Day One, the two pressure readings are tightly coupled, but in the first twelve hours of Day Two, they are almost completely uncoupled.

More generally, we have made similar observations in many datasets in scientific and medical domains. It seems quite common that the strength and type of *relationship*

(*correlation* is not quite the right word here) between two time series can ebb and flow over time, and neither DTW_I nor DTW_I is always best.

Finally, before continuing, we must discount a solution that may have occurred to the reader, perhaps based on visual inspection of Figure 1. One might imagine that a large dissimilarity between the warping paths in DTW_I could indicate that the exemplar comparison in question is best suited to DTW_I. However, this is not the case. In general, it is possible that two warping paths could be arbitrarily different, but reflect *identical* DTW distances. In Figure 34 we illustrate this with a toy example. While this example uses symmetric data to elucidate our point, this observation is more generally true.

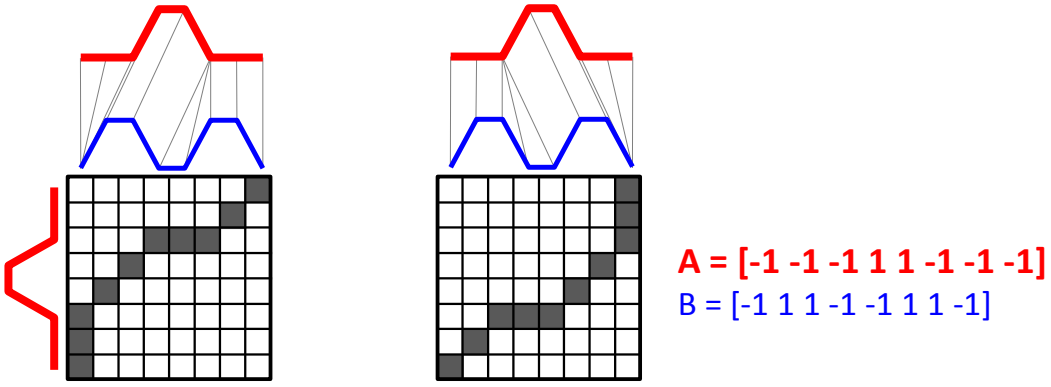


Figure 34: An example of two warping paths that are very different, but reflect *identical* DTW distances of 2.80.

3.3 Proposed Framework

In essence, our task reduces to a meta-classification problem. Given an instance to classify, we must first decide whether it is "an object best classified by DTW_I" or "an object best classified by DTW_D." More formally:

Problem Statement: Given that we are using NN-DTW to classify an exemplar Q, and that we have discovered the nearest neighbor to Q under both DTW_I and DTW_D, if the

classes of the two nearest neighbors differ, predict the distance function most likely to be correct.

Our problem statement is superficially similar to a "gating network" in a Mixture of Experts (ME), a technique frequently used in neural networks and some other classifiers [74]. Using the divide and conquer principle, several "experts" that are either regression functions or classifiers are created such that they specialize in a particular region of the input space. The gating network defines those regions where the individual expert opinions are trustworthy, and uses a probabilistic model to combine the expert's opinions.

There are many variants of ME (see [74] and references therein). However, in contrast to most versions, we have a much narrower task. We have exactly two experts, and we are "weighting" their opinions only in a strictly binary sense.

We propose the following solution to our problem. Offline, on the training data, we will compute a *threshold*. At query time, we will compute a score *S*, and choose which method to trust based on the value of *S* relative to the *threshold*. In order to best explain our framework, we first explain *how* our classification model works, given that a *threshold* has been learned from the *trainData.* Later, in Section 3.3.2*,* we consider the task of *learning* the *threshold*. If we have a lack of labeled data, we outline two possible approaches. We can either learn the *threshold* from a different dataset, from the same or similar domain, or we can simply hardcode the *threshold* to one, which gives us much of the benefit of our observation. We have explored both ideas in [76].

3.3.1. Adaptive Classifier for MDT

Table 9 outlines our classification algorithm. In line 1 the algorithm finds the nearest neighbor distance in the training set for Q under DTW_D, *minD*. In line 2 we find the nearest neighbor distance under DTW_I, *minI*. In line 3 the procedure divides *minD* by *minI*, which is our scoring function, *S*. In lines 4 to 8 the algorithm compares our scoring function, *S*, to the previously learned *threshold*. If *S* is greater than the *threshold*, we believe that *Q* is most likely in *I* and thus return DTW_I as the distance measure for classification, whereas if *S* is less than or equal to the *threshold*, we predict that *Q* is most likely in *D,* and the function returns DTW_D.

Table 9: Adaptive classification algorithm

Procedure *adaptive_Classifier* (***Q, trainData, threshold***) Input: A time series query, *Q*, the labeled data, *trainData, a threshold*; Output: An adaptive distance measure to classify *Q*, DTW_A;	
1	*minD* ← Nearest_Neighbor_Distance_D (*Q*, *trainData*);
2	*minI* ← Nearest_Neighbor_Distance_I (*Q*, *trainData*);
3	*S* ← *minD* / *minI*;
4	**if** *S* > *threshold*
5	$DTW_A \leftarrow DTW_I$;
6	**else**
7	$DTW_A \leftarrow DTW_D$;
8	**end if**
9	**Return** DTW_A

We formally define our scoring function as:

$$S = \frac{Nearest\ Neighbor\ Distance\ under\ DTW_D}{Nearest\ Neighbor\ Distance\ under\ DTW_I + \epsilon} \quad (5)$$

The *epsilon* in the denominator is to prevent division by zero, a value that is theoretically possibility but never observed. While *S* can change in the range of [ε, ∞], in practice we find its value to always be in the range of [0.5, 2].

Having explained our simple classification algorithm, all that remains is to explain how we set the threshold (and *why*). In fact, hardcoding the threshold to a value of exactly *one* works very well and allows the algorithm in Table 9 to beat the rival methods. However, we can further improve the accuracy by tuning the threshold, so in the next section we will explain how that is done.

3.3.2. Learning the Adjusted Threshold

In order to learn the *threshold* we use in Table 9, we first need to identify the *iSuccess* (*def.* 5) and *dSuccess* (*def.* 6) exemplars in the training set using cross validation (Table 11). As shown in Table 10, we consider four cases based on whether *iSuccess* and *dSuccess* are empty sets or not.

Table 10: Learning the adjusted threshold

	Procedure *Learn_Threshold* (*trainData*) Input: Labeled data, *trainData*; Output: Adjusted threshold, *threshold*;
1	[*S_iSuccess, S_dSuccess*] ← *find_Scores* (*trainData*);
2	**if** (*S_iSuccess* == ϕ && *S_dSuccess* == ϕ)
3	*threshold* ← 1;
4	**else if** (*S_iSuccess* == ϕ && *S_dSuccess* != ϕ)
5	*threshold* ← max(*S_dSuccess*) ;
6	**else if** (*S_iSuccess* **!=** ϕ && *S_dSuccess* == ϕ)
7	*threshold* ← min(*S_iSuccess*) ;
8	**else if** (*S_iSuccess* **!=** ϕ && *S_dSuccess* != ϕ)
9	*threshold* ← Decision_Tree (*S_iSuccess*, *S_dSuccess*);
10	**end if**

In line 1 we run the subroutine in Table 11 to find all the *S* scores for *iSuccess* and *dSuccess*, and then we consider four cases on the two sets. Line 2 is the case in which both sets are empty, so the problem (at least the training data) is independent of *D* or *I*, and

picking either DTW_I or DTW_D will make no difference in classifying the data. Therefore, we assign the value one, an arbitrary number, to the *threshold* in line 3. We note that this case is possible, but we never observed it.

In line 4 we test to see if *S_iSuccess* is empty and *S_dSuccess* is non-empty. If so, the dataset is almost certainly in *D*, and we need to set the *threshold* such that the *S* score for all *dSuccess* exemplars will be less than the *threshold*. Therefore, in line 5 the *threshold* gets assigned to the maximum value of *S_dSuccess*.

The opposite case, in which *S_dSuccess* is empty and *S_iSuccess* is non-empty (line 6), offers evidence that the dataset is in *I,* and we need to set the *threshold* such that the *S* score for all *iSuccess* exemplars will be greater than the *threshold*. We ensure this (in line 7) by assigning the *threshold* to the minimum value of *S_iSuccess*.

In practice, the three cases above are rare, and in lines 8 to 10 we find the *threshold* for the most common case in which both *S_iSuccess* and *S_dSuccess* are non-empty sets. The best *threshold* is a value that maximizes the total number of *S_iSuccess* with values greater than the *threshold* and *S_dSuccess* with values less than the *threshold*. Finding such a *threshold* is essentially the decision tree problem of maximizing the *information gain* by finding the optimal split point. For more details on information gain, we refer the interested reader to [73][60].

Recall the function *find_Scores* (*trainData*) called in Table 10. The function uses cross validation to find the two sets, *iSuccess* and *dSuccess,* then calculates the *S* scores (Eq. 5) for all their exemplars. The algorithm is described in Table 11. In line 1 we apply cross validation to the entire *trainData*. In line 2 we calculate the nearest neighbor distance under

DTW_D for each exemplar, and in line 3 we do the same under DTW_I. In lines 4 to 7, if the exemplar in the *trainData* is classified correctly under DTW_D and misclassified under DTW_I, the *S* score (Eq. 5) is calculated and gets added to *S_dSuccess*. In line 8 to 11, if the exemplar is misclassified under DTW_D and classified correctly under DTW_I, we calculate the *S* score and add it to *S_iSuccess*.

Table 11: An algorithm to find *iSuccess* and *dSuccess* and compute *S* scores for all their exemplars

```
Procedure  find_Scores (trainData)
Input:     Labeled data, trainData;
Output:    S scores for iSuccess and dSuccess, S_iSuccess and S_dSuccess;
 1  for   n ← 1 to size(trainData)
 2     minD ← Nearest_Neighbor_Distance_D (trainData(n),trainData);
 3     minI ← Nearest_Neighbor_Distance_I (trainData(n), trainData);
 4     if   (trainData(n).label == Nearest_Neighbor_D ().label  &&
 5              trainData(n).label != Nearest_Neighbor_I ().label )
 6           S_dSuccess.add (minD / minI);
 7     end if
 8     if   (trainData(n).label != Nearest_Neighbor_D ().label  &&
 9              trainData(n).label == Nearest_Neighbor_I ().label )
10           S_iSuccess.add (minD / minI);
11     end if
12  end for
```

3.3.3. The Intuition behind Our Scoring Function, S

In this section we explain the intuition behind our scoring function (Eq. 5), introduced in Section 3.3.1. We will show how the ratio of the nearest neighbor distance under DTW_D (*minD*) and DTW_I (*minI*) is capable of discriminating multi-dimensional time series in *I* from exemplars in *D*. For any time series in *I,* each dimension is at least somewhat independent; therefore, exemplars that are from the same class and are warped/shifted

independently in each dimension exist in the training set. Figure 35.*top* shows a time series in *I*, Q_I and its nearest neighbor in the training set, C_I.

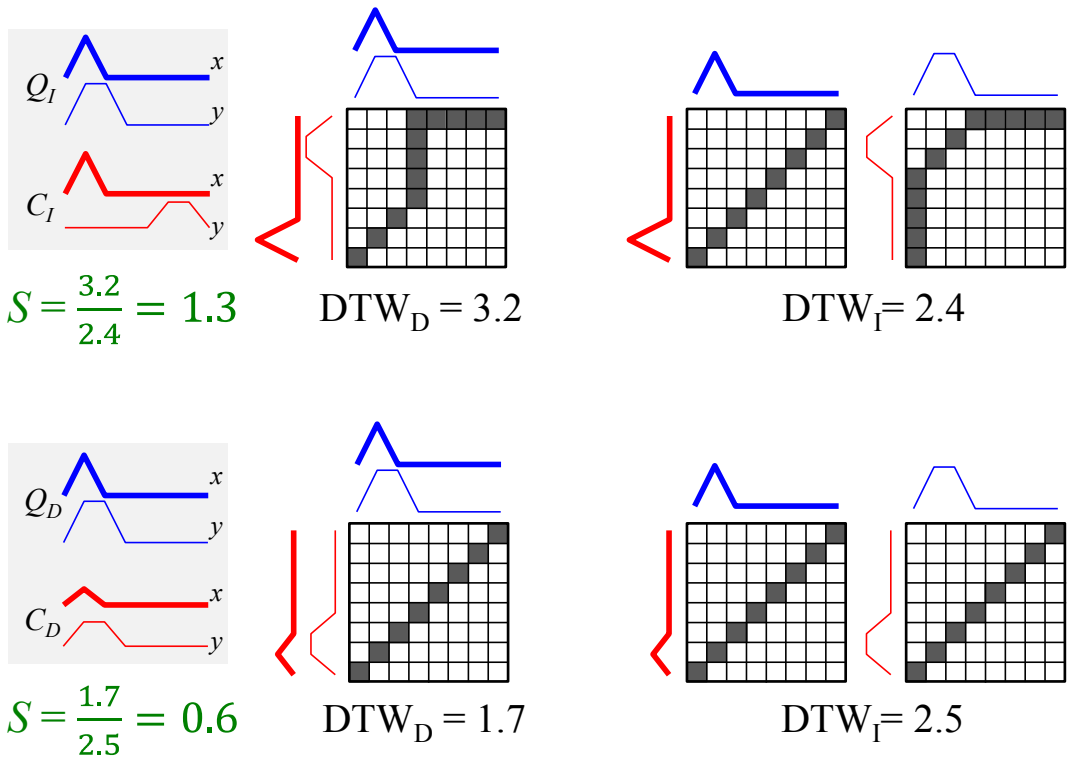


Figure 35: *top*) Computing the *S* score for a two-dimensional time series in *I*. *bottom*) *S* score calculation of a two-dimensional time series in *D*.

Note that in our proposed algorithm in Section 3.3.1 the nearest neighbors under DTW_I and DTW_D are not necessarily the same; however, for simplicity of presentation we consider C_I as the nearest neighbor under both DTW_I and DTW_D. Since Q_I is in *I* and its dimensions are warped independently, the DTW_I distance will be less than or equal to the DTW_D simply because DTW_I is allowed the freedom to find the nearest distance independently in each dimension. In Figure 35.*top*, DTW_I calculates the distance in two different paths, whereas DTW_D has only one path to pick which is a combination of the

two paths in DTW_I and eventually produces a larger distance. For any instance in *I*, *minD* is larger and *minI* gets smaller; therefore, *minD*/*minI* tends to be larger.

In Figure 35.*bottom* we show an instance, Q_D, which is in *D*. In this case the nearest neighbor in the training set, C_D, will be an exemplar in which both dimensions are dependently warped. In such a case, the warping path for both dimensions in DTW_I are the same as, and similar to, the path in DTW_D. In contrast to the previous case, DTW_I does not take advantage of different warping paths in order to achieve a lower distance score compared to DTW_D. However, we show for the same warping path, the distance under DTW_I is larger than the DTW_D distance. Since DTW_D and DTW_I both take the same path, we can compare their cumulative distances in a meaningful way.

If $q_{i,m}$ is the i^{th} data point in the m^{th} dimension of Q_D and $c_{j,m}$ is the j^{th} data point in the m^{th} dimension of C_D, for the two-dimensional case in Figure 35.*bottom,* we can show the following:

$$DTW_D(Q_D, C_D) = \sqrt{\sum_{i,j=1}^{n} [(q_{i,1} - c_{j,1})^2 + (q_{i,2} - c_{j,2})^2]} \prec$$

$$\sqrt{\sum_{i,j=1}^{n} (q_{i,1} - c_{j,1})^2} + \sqrt{\sum_{i,j=1}^{n} (q_{i,2} - c_{j,2})^2} = DTW_I(Q_D, C_D)$$

Accordingly, for a time series in *D*, *minD* is smaller and *minI* gets larger; therefore, *minD*/*minI* tends to be smaller. We considered a two-dimensional time series here and assumed that for a query in *I*, the path in DTW_I and DTW_D are exactly the same; however,

we can simply generalize the above illustration to dimensions greater than two, and for queries in *I*, different but similar paths for both DTW_I and DTW_D.

We have shown that our scoring function, *S* (cf. Eq. 5), tends to produce larger values for queries in *I* and smaller values for queries in *D*, as illustrated above; thus, our scoring function is capable of discriminating time series in *I* from exemplars in *D*. We will demonstrate the effectiveness of our scoring function with extensive experiments in the next section.

3.4 Experiments

We have designed all our experiments to ensure that they are *very* easy to reproduce. A supporting webpage [76] contains *all* the code, datasets, and raw data spreadsheets used in this work. Moreover, although this work is completely self-contained, the webpage contains additional experiments for the interested reader.

In addition to comparing to DTW_D and DTW_I, we also compare to the classification using each *individual* dimension, which we refer to using the notation DTW(*1st*), DTW(*2nd*), etc.

It is important to note that all experiments use *exactly* the same base algorithm, one nearest neighbor, and *exactly* the same train/test splits. Thus, any differences in results can be attributed solely to the distance measure used.

It is known that the warping window width can slightly affect the classification accuracy. As this issue is orthogonal to our work, we simply set the warping window constraint for DTW to be 20% for all experiments [42].

3.4.1. Recognition of Cricket Umpire Signals

Cricket is a very popular game in British Commonwealth countries. The game requires an umpire to signal different events in the game to a distant scorer/bookkeeper. The signals are communicated with motions of the hands. For example, `No-Ball` is signaled by touching each shoulder with the opposite hand, and `TV-Replay`, a request for an off-field review of the video of a play, is signaled by miming the outline of a TV screen (*cf.* Figure 26). A complete dictionary of signals can be found in [76].

The dataset introduced in [53] consists of four umpires performing twelve signals, each with ten repetitions. The data, recorded at a frequency of 184Hz, was collected by placing 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-dimensional MDT from the two accelerometers and we can combine any number of them to create a multi-dimensional classification problem. Figure 36 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.

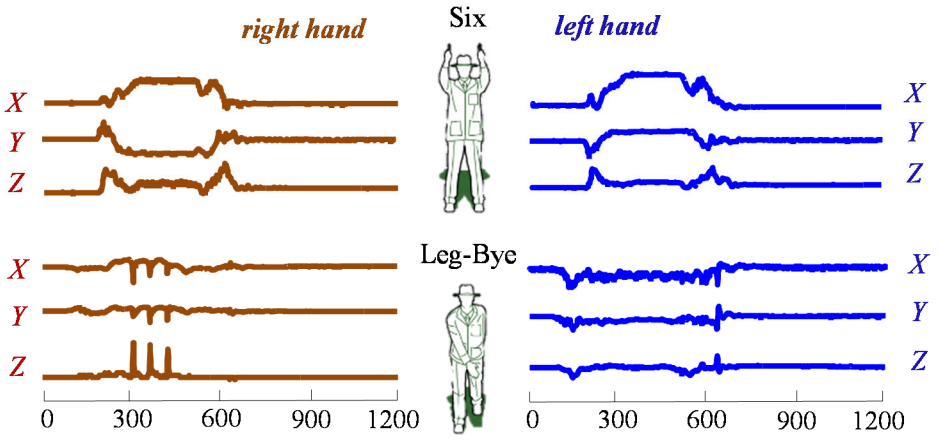


Figure 36: *X, Y* and *Z* acceleration data from the right hand (*left*), a representation of the umpire's body position (*center*), and the *X, Y* and *Z* acceleration data from the left hand, for the two umpire signals `Six` and `Leg-Bye`.

We used 40% of the data for training and use the rest as testing data. The classification results using various combinations of dimensions are shown in Table 12.

Table 12: Classification error rates on the cricket data

Data	DTW($_{1st}$)	DTW($_{2nd}$)	DTW_I	DTW_D	DTW_A
$X_{right\_}X_{left}$	0.26	0.27	0.13	0.20	**0.11**
$Y_{right\_}X_{left}$	0.17	0.27	0.07	0.04	**0.03**
$X_{right\_}Y_{left}$	0.26	0.14	0.07	0.10	**0.06**
$Y_{right\_}Y_{left}$	0.17	0.14	0.04	0.03	**0.03**
$Z_{right\_}Z_{left}$	0.18	0.18	0.07	0.06	**0.04**
$Z_{right\_}X_{left}$	0.18	0.27	0.07	0.04	**0.04**

Note that *all* combinations support our original claims that neither DTW_D nor DTW_I dominates the other, and that on all datasets, DTW_A is *at least as* accurate as the better of DTW_D and DTW_I, and often *more* accurate.

Above we considered only *pairs* of dimensions, however, the results generalize for any-sized subsets of dimensions. Obviously, adding more dimensions does not guarantee improved accuracy. In [47] the authors outline a strategy for choosing *which* dimensions

to add to an MDT. Note, however, that this issue is completely orthogonal to our contributions; Table 12 suggests that whatever set of dimensions you choose, you are better off with DTW_A than any other method.

3.4.2. Accelerometer-Based Gesture Recognition

There is increasing interest in using gesture commands for interacting with and controlling external devices. The results in [50] suggest that different people often have different interpretations of even simple gestures, and thus it is necessary to *learn* personalized classifiers on an individual basis.

A widely used benchmark dataset, introduced in [56], consists of the performances of the gestures shown in Figure 37 by eight participants. To produce realistic variability in performance, the data was collected on multiple days over three weeks. On each day the participant held a Nintendo Wii remote and repeated each of the eight gestures ten times.

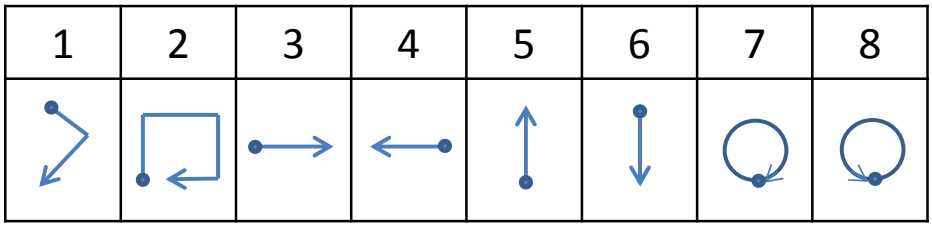


Figure 37: Gesture vocabulary adopted from [50]. The dot denotes *the start* and the arrow *the end* of the gesture.

The dataset consists of 4,480 gestures in total: 560 for each participant. The accelerometer has three axes (*X*, *Y* and *Z*); thus, we have a three-dimensional MDT form, and we can combine them to create a two or three multi-dimensional classification problem. We combined every pair of dimensions to create all possible two-dimensional

time series and combined all three for the three-dimensional case. The classification results are shown in Table 13.

Table 13: Error rates for gesture recognition

Data	DTW(*1st*)	DTW(*2nd*)	DTW(*3rd*)	DTW_I	DTW_D	DTW_A
X_Y	0.34	0.48	-	0.23	0.18	**0.18**
X_Z	0.34	0.41	-	0.13	0.13	**0.12**
Y_Z	0.48	0.41	-	0.26	0.24	**0.23**
X_Y_Z	0.34	0.48	0.41	0.11	0.09	**0.08**

As before, the results support our claim that DTW_A is *at least as* accurate as the better of DTW_D or DTW_I.

3.4.3. Word Recognition from Articulatory Movement Data

Silent "speech" recognition may potentially facilitate oral communication in people with severe voice impairments, for example, after laryngectomy, a surgical removal of larynx due to the treatment of cancer [70][71][72]. Silent speech recognition is to recognize words or sentences from non-audio data (e.g., tongue and lip movement data) [71]. An Electromagnetic Articulograph (EMA) [75] is an apparatus used to measure the movement of the tongue and lips during speech. The motion tracking using EMA is registered by attaching small sensors on the surface of the articulators (e.g., tongue and lips). The spatial accuracy of motion tracking using EMA AG500 is 0.5 mm [75]. We consider the EMA dataset in [70] which contains data collected from multiple native English native speakers producing 25 words. Twelve sensors were used in data collection, each providing *X*, *Y* and *Z* time-series positions with a sampling rate of 200 Hz. As shown in Figure 38 the sensors are located on the forehead, tongue; from tip to back in the midline, lips and jaw. The three

head sensors (Head Center, Head Right, and Head Left) attached on a pair of glasses were used to calculate head-independent movement of other sensors. Tongue sensors were named T1, T2, T3, and T4, from tip to back. For more details about the data collection procedure and description, please refer to [70].

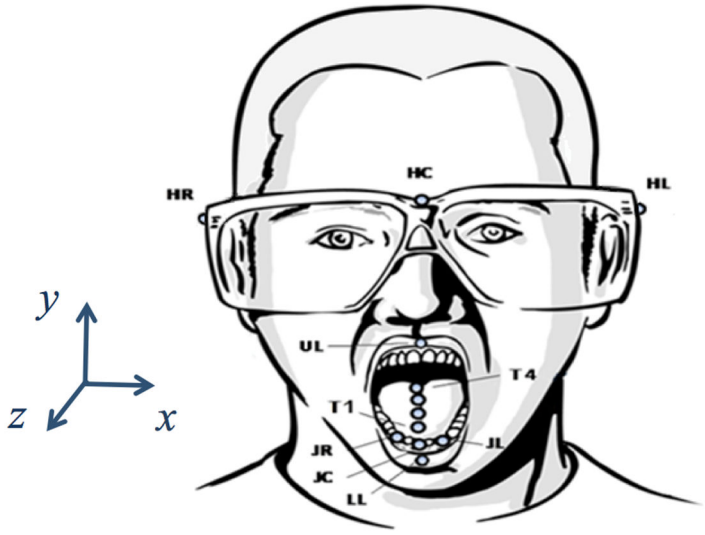


Figure 38: The coordinate system and sensor locations on a participant's forehead, tongue, lips, and jaw in data collection using EMA. Labels are described in text.

Of the total of 36 available dimensions, for brevity and simplicity, we show only some random combinations of dimensions extracted from the sensors on the tongue tip (*T1*), the upper lip (*UL*) and lower lip (*LL*). The classification results are shown in Table 14.

Table 14: Classification error rates on the continuous articulatory movement dataset

Data	DTW(*1st*)	DTW(*2nd*)	DTW(*3rd*)	DTW_I	DTW_D	DTW_A
$T1_Z\_UL_X$	0.34	0.59	-	0.25	0.31	**0.25**
$T1_Y\_UL_Y$	0.38	0.55	-	0.22	0.15	**0.13**
$T1_Z\_LL_Z$	0.34	0.47	-	0.17	0.10	**0.10**
$T1_Z\_LL_Y$	0.34	0.48	-	0.20	0.12	**0.11**
$T1_Y\_LL_Y$	0.38	0.48	-	0.21	0.14	**0.14**
$T1_Y\_T1_Z$	0.38	0.34	-	0.24	0.15	**0.15**
$T1_X\_T1_Y\_T1_Z$	0.32	0.38	0.34	0.15	0.10	**0.10**
$T1_X\_T1_Y\_UL_Y$	0.32	0.38	0.55	0.16	0.18	**0.14**
$T1_Y\_T1_Z\_LL_X$	0.38	0.34	0.67	0.12	0.08	**0.08**
$T1_Z\_LL_X\_LL_Y$	0.34	0.67	0.48	0.14	0.12	**0.10**

Yet again, the results support our claim that DTW_A is *at least as* accurate as *the better* of DTW_D or DTW_I.

3.4.4. Revisiting the Semi-Synthetic Data

We reconsider the handwriting data set used in Section 3.2. Recall that the data is *real*, but manipulated in ways such that it changed from being in *D* to being in *I*. In Figure 39 we revisit these problems, this time using DTW_A. Once again these experiments offer strong support for our claims about DTW_A dominating DTW_I and DTW_D.

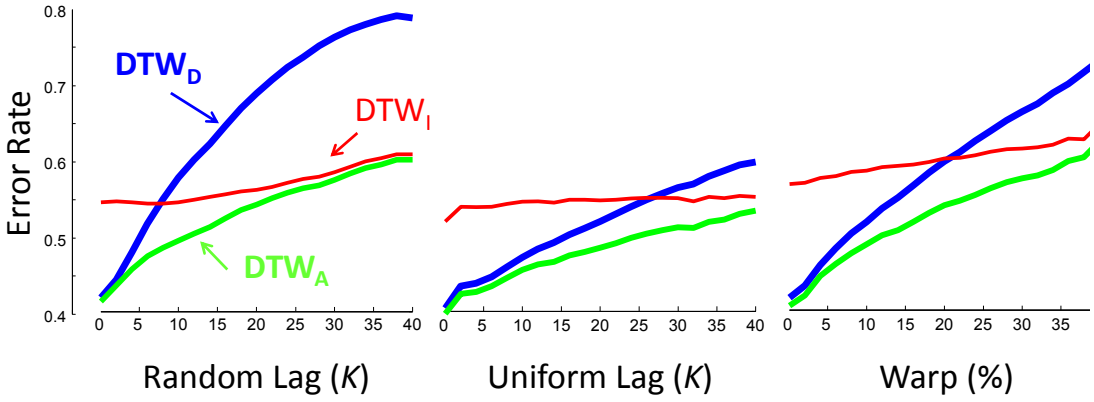


Figure 39: The experiments shown in Figure 28, Figure 29 and Figure 30, revisited using the DTW_A technique.

3.4.5. Human Activity Recognition Using Smart Watches

Providing accurate and exploitable information on human activity has become an active field in pervasive computing [9][55][43]. Activity recognition using a smart watch has the potential to be highly useful in modern healthcare by monitoring the patients' activities and automatically reporting summaries to healthcare providers [64]. In order to detect a specific set of gestures or behaviors during daily activities we can simply use a *rejection threshold,*

which classifies the target gestures from (the much larger space of) non-target activities [48]. However, in order to detect such gestures/behaviors we need to know the best distance measure to classify the gestures. For this purpose we designed a simple experiment. We asked two users to wear a Samsung Gear 2 and execute 100 performances of eight different gestures. We collected the accelerometer data (*X*, *Y* and *Z*) with a sampling rate of 50 Hz. The eight gestures performed by users and a sample accelerometer data is shown in Figure 40.

Figure 40: *left*) Sample accelerometer data (*X, Y* and *Z*) of a gesture. *middle*) A Samsung Gear 2 used to collect activity data. *right*) the eight different gestures considered in our experiments.

We combined every pair of dimensions to create all possible two-dimensional time series and combined all three for the three-dimensional case. The classification results are shown in **Table 15**.

Table 15: Error rates for activity detection using a smart watch

Data	DTW(*1st*)	DTW(*2nd*)	DTW(*3rd*)	DTW_I	DTW_D	DTW_A
X_Y	0.43	0.46	-	0.25	0.17	**0.15**
X_Z	0.43	0.50	-	0.28	0.15	**0.12**
Y_Z	0.46	0.50	-	0.33	0.15	**0.15**
X_Y_Z	0.43	0.46	0.50	0.22	0.12	**0.12**

As with previous experiments, the results support our claim that DTW_A is *at least as* accurate as the better of DTW_D or DTW_I.

3.4.6. Learning the Threshold with Sparse Training Data

The reader may wonder if it is possible to learn the *threshold* if we have little labeled data to work with. For example, this is a common situation when beginning to use a new gesture-based system (the so-called "cold start" problem).

As noted in Section 3.3.1 in this dissertation, simply hardcoding the threshold to a value of one gives us much of the benefit of our observation. However, tuning the threshold **does** help, and we want to obtain the best possible accuracy.

Without claiming to have completely solved this problem, we outline one possible approach here. Our idea is that we can learn the *threshold* from a different dataset from the same or similar domain. In essence this is a simple form of *transfer learning*.

For example, if a company releases a smartwatch with gesture recognition capabilities, a good "universal" threshold could be learned and set at the factory. This would allow the system to work well "out-of-the-box," and possibly be refined and personalized over time.

We have conducted an experiment to demonstrate this idea. For the *Gesture Recognition* dataset in Section 3.4.2 of this dissertation, we combined all three dimensions of *X*, *Y* and *Z* from the accelerometer and created two completely disjointed datasets, G_1 and G_2. We learned the *threshold* from G_1 and used the same *threshold* to classify G_2. The results are shown in **Table 16**.

Table 16: The first and second row correspond to results for classifying G_2 with the threshold learned from G_2 and G_1, respectively

Data	DTW(*1st*)	DTW(*2nd*)	DTW(*3rd*)	DTW_I	DTW_D	DTW_A
X_Y_Z (1)	0.30	0.45	0.37	0.08	0.06	**0.05**
X_Y_Z (2)	0.30	0.45	0.37	0.08	0.06	**0.06**

The results tentatively suggest that by adopting the threshold value from a different dataset in the same domain, we can achieve approximately the same accuracy we would have achieved by learning the threshold from a (large sample) of the native dataset.

3.4.7. What Causes a Time Series to be in *D* or *I* ?

For all of the experiments considered above, the data sets included a mixture of exemplars in *I* and *D*. If this was not true, DTW_A could not have had a lower error-rate. However, an interesting question we have glossed over thus far is *what* causes an individual time series exemplar to be in *D* or *I*? Is it an intrinsic property of the individual exemplar itself or a property of the exemplar in relation to a particular data set? We conducted a wide-ranging investigation of the exemplars' characteristics in various domains to see if any feature(s) stands out as a discriminator of time series in *I* vs. in *D*. We considered the correlation, complexity, Euclidean distance, Minimum Description Length, etc. of dimensions and did not find any useful discriminator.

We have designed a simple experiment which strongly suggests (at least for the data set considered) that the existence of exemplars in *I* or *D* strongly depends on the entire data set. A time series, which is in *I,* may later be in *D* if we use a different training set for classifying that item. In addition, just the size of the training set can have a significant impact on whether exemplars fall in *I* or *D*.

Once again we revisit the handwriting data set we used in Section 3.2. In all iterations we randomly sample ten exemplars from each of twenty-six classes as the test set (a total of 260 items). In the first iteration we randomly pick only a single instance for each class and define it as the train set. Then we classify the test set using DTW_D and DTW_I separately and also count the number of items in *iSuccess* and *dSuccess* (cf. definitions 5 and 6). In the second iteration we randomly pick two instances for each class as the train set and repeat the same steps. We continue the iterations until we reach seventeen exemplars per class. The results are shown in Figure 41.

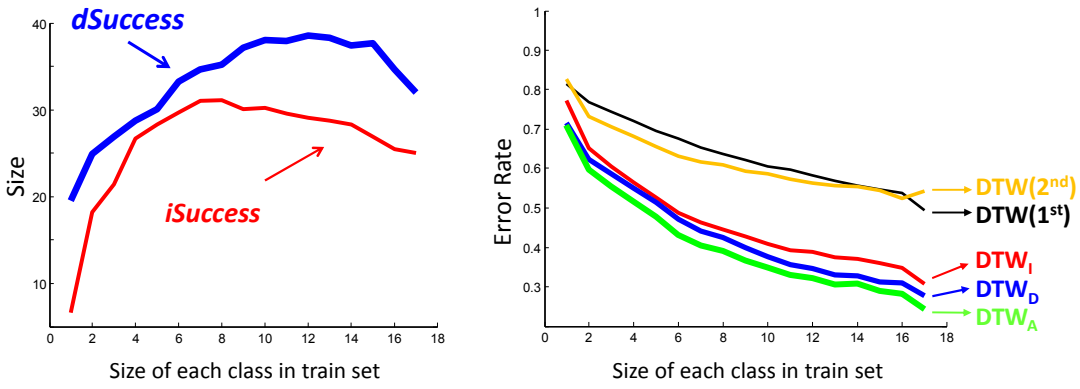


Figure 41: *left*) The number of elements in *iSuccess* and *dSuccess* for train sets of different sizes. *right*) The effect of data set size on accuracy of classification.

We believe we can interpret Figure 41 as follows. First (and only incidentally here), Figure 41.*right* supports our claim that DTW_A is *at least as* accurate as *the better* of DTW_D or DTW_I, as the green curve for DTW_A dominates all the other approaches for the entire range of training data sizes. Note that all five approaches have almost the same (very high) error-rate when the training set is very small. This is simply because there is very little space for them to differ. In the limit, had we started one value to the left, with *zero* data, all approaches would have had the *exact* same error-rate, the default rate.

As the training dataset gets larger, all approaches benefit, just as we expect. However, DTW_A benefits the most. This is because as the training dataset gets larger, there is a greater possibility that some objects to be classified can benefit from choosing the more suitable of the two multi-dimensional variants of DTW. To see this more clearly, in Figure 41.*left* we measured the number of objects in *iSuccess* and *dSuccess* for the experiments shown in Figure 41.*right*. We see that for small train sets the number of items in *iSuccess* and *dSuccess* are low. However, when the size of the train set increases, the number of instances in *iSuccess* or *dSuccess* begins to increase in spite of the fact that the size of the test sets remains constant.

There is one observation in Figure 41.*left* that we have to explain. After some point, the number of items in *iSuccess* and *dSuccess* begins to *decrease*. Why is this? The reason is that, for large enough training sets, there is a greater chance that the nearest neighbor, under *both* DTW_I and DTW_D, will be the same (true) class. This will make our DTW_A unnecessary (but *not* harmful to accuracy). A similar effect has been shown for the error-rates of (one dimensional) DTW vs. ED classifiers (See Figure 1 of [67]). For small datasets, DTW and ED often make different decisions about which item is the nearest neighbor, but as the datasets get larger they tend to agree more and more often, eventually converging to the same error-rate [67][62].

In a sense, these observations seem to cast limits on the utility of our proposed ideas. DTW_A will be no better (but critically, no *worse*) in the case that the training dataset is pathologically small or is arbitrarily large. However, the situation in between is clearly the most common. For example, in virtually all gesture recognition systems it is assumed that

the user is willing to provide at least five to ten examples of a gesture so that we can estimate variability of performance [50][56][46][69][35][68]. But, clearly, we do not expect an individual user to provide one thousand labeled examples of a gesture.

3.5 Related Work

We have deferred a discussion of related work until now when the reader can appreciate the nature of our contributions. While there are hundreds of research efforts that use DTW in a multi-dimensional setting [45][50][53][56][59][70], we are not aware of any work that discusses the relative merits of DTW_I and DTW_D, or even explicitly notes that they are alternatives. The vast majority of researchers seem to think that it makes no difference which method is used, as evidenced by the fact that they usually do not bother to explicitly *tell* the reader [56][35][35][49][68]. In paper [56] they define DTW as a dynamic programming algorithm, which calculates the matching cost and finds the corresponding shortest path. However, it is not clear how they generalize it to the multi-dimensional case. A handful of papers do mention that there exist two ways of computing dynamic time warping in multi-dimensional time series. For example, the authors in [59] choose DTW_D for classifying satellite images because they argue that satellite images have dependent dimensions in their time series. Other papers, such as [46][35][44], use DTW_D without pointing out the alternative approach of DTW_I. The authors in [69][40][57] use other methods similar to DTW_I such as adding up all dimensions and dealing with a single dimension time series. For instance, [40] applies DTW to the data obtained from the sum of all channels in different dimensions. The authors in [69] normalize and smooth each

dimension and then use the total difference among dimensions to find the best synchronization with the regular DTW algorithm.

The ubiquity of multi-dimensional time series, especially given the recent explosion of interest in wearable devices, has produced significant research in speeding up DTW [63], choosing which subset of dimensions to use [47], choosing a setting for the warping window constraint [42], etc. However, all such work is orthogonal to (and compatible with) our contributions. We have created an annotated bibliography of which papers use DTW_I vs. DTW_D [76].

As we noted above, there is significant research in "gating networks" and related techniques for choosing which classifier to use on a given region of the input space [74]. However, to the best of our knowledge, these ideas have never been applied to time series and are only superficially related to the task at hand.

3.6 Conclusions

In this work we demonstrate for the first time that of the two obvious ways to do multi-dimensional NN-DTW classification, neither is always superior. We show that the differences are not trivial, as the wrong choice can *double* the error rate. We introduce a simple algorithm that can pick the method that is most likely to predict the correct class on a case-by-case basis. Our algorithm is simple to implement, and its overhead is inconsequential in terms of both time and space.

For concreteness we have confined our remarks and empirical demonstrations to *classification* problems, but note that distance measures are at the heart of many time series

data mining tasks, including clustering, summarization, motif discovery, and many forms of anomaly detection. In future work we will expand our consideration of our ideas to these tasks.

Finally, in this work we have focused on intuitively explaining our observations/ideas and showing strong empirical evidence for them. However, we plan to revisit our work with a more theoretical framework and prove several useful properties of DTW_A.

Chapter 4: Conclusions

We have introduced a technique for finding rules in time series which leverages of recent advances in time series motifs discovery to provide a tractable search. Our novel application of MDL to time series rule discovery allows us to meaningfully rank and compare varied length rules, and rules with different levels of "support". Our rule representation is expressive enough to allow rules with different length antecedents/consequents/lags/firing thresholds, but at the same time does not require extensive human intervention or tweaking. In addition to single dimensional time series, we also proposed an accurate classifier for the multi-dimensional case which can be used towards prediction and discovering rules.

In this dissertation we demonstrate for the first time that of the two obvious ways to do multi-dimensional NN-DTW classification, neither is always superior. We show that the differences are not trivial, as the wrong choice can *double* the error rate. We introduce a simple algorithm that can pick the method that is most likely to predict the correct class on a case-by-case basis. Our algorithm is simple to implement, and its overhead is inconsequential in terms of both time and space.

For concreteness we have confined our remarks and empirical demonstrations to *classification* problems, but note that distance measures are at the heart of many time series data mining tasks, including clustering, summarization, motif discovery, and many forms of anomaly detection. In future work we will expand our consideration of our ideas to these tasks.

In this dissertation we have focused on intuitively explaining our observations/ideas and showing strong empirical evidence for them. However, we plan to revisit our work with a more theoretical framework and prove several useful properties of DTW_A.

Finally, I would like to share some useful research philosophy that we observe during this dissertation process:

- For any bad idea, there is a dataset that makes it look good. Therefore, in all our work, we strive to test on *many* diverse real world datasets.
- Reproducibility is our religion. To encourage the adoption and extension of our ideas, we are making all code and data *freely available* at our webpage [34][76].

Bibliography

Chapters 1&2

[1] Barron, A., Rissanen, J., and Yu, B., The minimum description length principle in coding and modeling. IEEE Trans. Information Theory, vol. 44, no. 6. 1998.

[2] Brainard, M. S. & Doupe, A. J. Auditory feedback in learning and maintenance of vocal behaviour. Nature Rev. Neurosci. 1, 31–40 (2000).

[3] Brotzge, J. and Erickson, S., Tornadoes without NWS warning. Weather Forecasting, 25, 159-172. 2010.

[4] Chen, J. R.: Making Subsequence Time Series Clustering Meaningful. ICDM 2005: 114-121

[5] Cohen, P.R and Adams, N.M. An Algorithm for Segmenting Categorical Time Series into Meaningful Episodes. ICAIDA 2001, p.198-207, 2001.

[6] Das, G., Lin, K., Mannila, H., Renganathan, G., Smyth, P. Rule Discovery from Time Series. KDD 1998.

[7] Denton, A., Besemann, C., Dorr, D. H.: Pattern-based time-series subsequence clustering using radial distribution functions. Knowl. Inf. Syst. 18(1): 1-27 (2009)

[8] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E.J., Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB 1(2): 1542-1552. 2008.

[9] Dudley, R. Sample Functions of the Gaussian Process. The Annals of Probability, Vol. 1, No. 1, pp. 66-103, 1973.

[10] Ferrell, B., and Santuro, S., NASA shuttle valve data. http://cs.fit.edu/~pkc/nasa/data/. 2005.

[11] Gribovskaya, E., Kheddar, A., and Billard, A. Motion Learning and Adaptive Impedance for Robot Control during Physical Interaction with Humans. ICRA 2011.

[12] Hu, B., Rakthanmanon, T., Hao, Y., Evans, S., Lonardi, S., Keogh, E. Discovering the Intrinsic Cardinality and Dimensionality of Time Series Using MDL. ICDM 2011: 1086-91

[13] Jin, S. et al. 2012. Characterization of EPG waveforms for the tea green leafhopper on tea plants and their correlation with stylet activities. Journal of Insect Physiology. 58:35-44

[14] Keogh, E., Lin, J. Clustering of time-series subsequences is meaningless: implications for previous and future research. Knowl. Inf. Syst. 8(2). 2005. 154-177.

[15] Kojima, S. and Doupe, A. (2011). Social performance reveals unexpected vocal competency in young songbirds. Proc Natl Acad Sci USA. Vol 108 pp 1687-92.

[16] Kornai, A., Mathematical Linguistics, Springer. 2008.

[17] Grunwald, P.D., Myung, I. J. Advances in Minimum Description Length Theory and Applications. 2003.

[18] Li, G., Ji, S., Li, C., and Feng, J. Efficient type-ahead search on relational data: a TASTIER approach. SIGMOD Conference 2009: 695-706.

[19] Makonin, S., Popowich, F., Bartram, L., Gill, B., and Baijic, I. AMPds: A Public Dataset for Load Disaggregation and Eco-Feedback Research. Electrical Power and Energy Conference (EPEC), 2013 IEEE, pp. 1-6, 2013.

[20] Makridakis, S., Wheelwright, S., and Hyndman, R. J., Forecasting: methods and applications. New York: John Wiley & Sons. ISBN 0-471-53233-9. 1998.

[21] McGovern, et al. Identifying Predictive Multi-Dimensional Time Series Motifs: An application to severe weather prediction. Data Mining and Knowledge Discovery. 2010.

[22] Mueen, A., Keogh, E., Zhu, Q., Cash, S. and Westover, B. Exact Discovery of Time Series Motif. SDM 2009.

[23] Park, S., and Chu, S.W. Discovering and Matching Elastic Rules from Sequence Databases. Fundam. Inform. 47, 75-90. 2001.

[24] Parnandi, A., Le, K., Vaghela, P., Kolli, A., Dantu, K., Poduri, S., and Sukhatme, G. Coarse In-building Localization with Smartphones. Mobiecase 2009.

[25] Ricardo C., et al. The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition, Pattern Recognition Letters, 2013.

[26] Shatkay, H., Zdonik, S. B.: Approximate Queries and Representations for Large Data Sequences. ICDE 1996: 536-545

[27] Shokoohi-Yekta, M., Chen, Y., Campana, B., Hu, B., Zakaria, J., and Keogh, E. Discovery of Meaningful Rules in Time Series. SIGKDD (under review), 2015.

[28] Tanaka, Y., Iwamoto, K., and Uehara, K. Discovery of time-series motif from multi-dimensional data based on MDL principle. Machine Learning. 58(2), 2005.

[29] UCR Time Series, Classification and Clustering. http://www.cs.ucr.edu/~eamonn/time_series_data/

[30] Wallace, C and Dowe, D. Minimum message length and Kolmogorov complexity. The Computer Journal - CJ, vol. 42, no. 4, pp. 270-283, 1999.

[31] Weiss, S., Indurkhya, N., and Apte, C., Predictive Rule Discovery from Electronic Health Records. ACM IHI, 2010.

[32] Wu, H., Salzberg, B., and Zhang, D., Online Event-driven Subsequence Matching over Financial Data Streams, SIGMOD Conference, 2004: 23-34.

[33] Wu. H (2005). Personal email communication.

[34] Project URL: https://sites.google.com/site/ruleDiscovery

Chapters 1&3

[35] J. Aach and G. M. Church. *Aligning gene expression time series with time warping algorithms.* Bioinformatics 17, no. 6 (2001): 495-508.

[36] A. Akl, S. Valaee. *Accelerometer-Based Gesture Recognition via Dynamic–Time Warping, Affinity Propagation, & Compressive Sensing*. Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on, pp. 2270-2273. IEEE, 2010.

[37] Seba, Albertus. *Locupletissimi rerum naturalium thesauri accurata descriptio, et iconibus artificiosissimis expressio, per universam physices historiam : Opus, cui, in hoc rerum genere, nullum par exstitit* (1734)

[38] A. Al-Jawad, M. Reyes Adame, M. Romanovas, M. Hobert, W. Maetzler, M. Traechtler, K. Moeller and Y. Manoli, *Using multi-dimensional dynamic time warping for TUG Test instrumentation with inertial sensors.* 2012 IEEE Conference on, pp. 212-218. IEEE, 2012.

[39] I. Assent, M. Wichterich, R. Krieger, H. Kremer, T. Seidl: *Anticipatory DTW for Efficient Similarity Search in Time Series Databases*. PVLDB 2(1): 826-837 (2009).

[40] M. Bashir and J. Kempf. *Reduced Dynamic Time Warping for Handwriting Recognition Based on Multi-Dimensional Time Series of a Novel Pen Device*. International Journal of Intelligent Systems and Technologies, WASET 3, no. 4 (2008): 194.

[41] Das Ehrenbuch der Fugger (The secret book of honour of the Fugger) -BSB Cgm 9460, Augsburg, ca. 1545 - 1548 mit Nachträgen aus späterer Zeit.

[42] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh. 2008. *Querying and mining of time series data: experimental comparison of representations and distance measures*. PVLDB 1, 2, 1542-52.

[43] M. Ermes, J. Parkka, J. Mantyjarvi, and I. Korhonen. *Detection of daily activities and sports with wearable sensors in controlled and uncontrolled conditions*. Information Technology in Biomedicine, IEEE Transactions on 12, no. 1 (2008): 20-26.

[44] R. Fernandes de Mello and I. Gondra, *Multi-Dimensional Dynamic Time Warping for Image Texture Similarity*. In Advances in Artificial Intelligence-SBIA 2008, pp. 23-32. Springer Berlin Heidelberg, 2008.

[45] Field Guide to the Songbirds of South America The Passerines - Mildred Wyatt-Wold Series in Ornithology(2009) Robert S. Ridgely, Guy Tudor. University of Texas Press.

[46] N. Gillian, R. Benjamin Knapp, S. O'Modhrain, *Recognition Of Multivariate Temporal Musical Gestures Using N-Dimensional Dynamic Time Warping*. NIME11 (2011).

[47] B. Hu, Y. Chen, J. Zakaria, L. Ulanova, E. Keogh: *Classification of Multi-dimensional Streaming Time Series by Weighting Each Classifier's Track Record*. ICDM 2013: 281-290

[48] B. Hu, Y. Chen, and E. J. Keogh. *Time Series Classification under More Realistic Assumptions*. In SDM, pp. 578-586. 2013.

[49] N. Kale, J. Lee, R. Lotfian and R. Jafari . *Impact of Sensor Misplacement on Dynamic Time Warping Based Human Activity Recognition using Wearable Computers.* Proceedings of the conference on Wireless Health, p. 7. ACM, 2012.

[50] J. Kela, P. Korpipaa, J. Mantyjarvi, S. Kallio, G. Savino, L. Jozzo and D. Marca. *Acceleromter-based gesture control for a design environment*, Personal and Ubiquitous Computing, vol. 10, no. 5, pp. 285-299, 2006.

[51] E. Keogh and S. Kasetty, *On the need for Time Series Data Mining Benchmarks: a survey and empirical demonstration*, Proc. ACM KDD 2002, pp. 102-111.

[52] E. J. Keogh, L. Wei, X. Xi, S. H. Lee, M. Vlachos: *LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures*. VLDB 2006: 882-893

[53] 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.

[54] J. B. Kruskall, and M. Liberman, *The symmetric time warping algorithm: From continous to discrete. In Time Warps, String Edits and Macromolecules*. Addison-Wesley, 1983.

[55] O. D. Lara and M. A. Labrador. *A survey on human activity recognition using wearable sensors*. Communications Surveys & Tutorials, IEEE 15, no. 3 (2013): 1192-1209.

[56] J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya and V. Vasudevan, *uWave: Accelerometer-based personalized gesture recognition and its applications*, IEEE intl. Conf. Pervasive Computing and Communication (PerCom), 2009.

[57] D. McGlynn and M. G. Madden. *An Ensemble Dynamic Time Warping Classifier with Application to Activity Recognition.* Research and Development in Intelligent Systems XXVII, pp. 339-352. Springer London, 2011.

[58] P. Papapetrou, V. Athitsos, M. Potamias, G. Kollios, and D. Gunopulos. *Embedding-based subsequence matching in time-series databases*. ACM TODS 36, 3, 17, 2011.

[59] F. Petitjean, J. Inglada and P. Gancarski. *Satellite Image Time Series Analysis under Time Warping*. IEEE Transactions on Geoscience and Remote Sensing, vol. 50-. 8, pp. 3081-95, 2012.

[60] J. R. Quinlan, (1986). *Induction of Decision Trees*. Machine Learning 1: 81-106, Kluwer Academic Publishers.

[61] L. Rabiner, and B. Juang, *Fundamentals of speech recognition*. Englewood Cliffs, N.J, Prentice Hall, 1993.

[62] C. A. Ratanamahatana and E. Keogh. (2004). *Everything you know about Dynamic Time Warping is Wrong*. Third Workshop on Mining Temporal and Sequential Data, in conjunction with the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), August 22-25, 2004 - Seattle, WA

[63] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria and E. J. Keogh: *Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping*. TKDD 7(3): 10 (2013)

[64] R. Rawassizadeh, B.A. Price., M. Petre. Wearables: *Has the age of Smartwatches finally arrived?*. in Communication of ACM, 2015. Vol 58, Issue 1

[65] Y. Sakurai, C. Faloutsos, and M. Yamamuro. 2007. *Stream monitoring under the time warping distance*. ICDE, 1046-55.

[66] M. Shokoohi-Yekta, J. Wang and E. Keogh. *On the Non-Trivial Generalization of Dynamic Time Warping to the Multi-Dimensional Case*. SIAM SDM 2015 (to appear).

[67] J. Shieh, E. J. Keogh: *iSAX: disk-aware mining and indexing of massive time series datasets*. Data Min. Knowl. Discov. 19(1): 24-57 (2009)

[68] J. Tang. *Extracting Commands From Gestures: Gesture Spotting and Recognition for Real-time Music Performance*. PhD diss., Carnegie Mellon University, 2013.

[69] G.A. Ten Holta, M.J.T. Reindersa and E.A. Hendriks. *Multi-Dimensional Dynamic Time Warping for Gesture Recognition.* Thirteenth annual conference of the Advanced School for Computing and Imaging. Vol. 300. 2007.

[70] J. Wang, A. Balasubramanian, L. Mojica de la Vega, J. R. Green, A. Samal, and B. Prabhakaran. *Word recognition from continuous articulatory movement time-series data using symbolic representations*, ACL/ISCA Interspeech Workshop on Speech and Language Processing for Assistive Technologies, Grenoble, France, 119-127 (2013).

[71] J. Wang, A. Samal and J. R. Green. *Preliminary test of a real-time, interactive silent speech interface based on electromagnetic articulograph*, ACL/ISCA Workshop on Speech and Language Processing for Assistive Technologies, Baltimore, MD, 38-45, 2014.

[72] J. Wang, A. Samal, J. R. Green and F. Rudzicz. *Whole-word recognition from articulatory movements for silent speech interfaces*, Proc. of Interspeech, Portland, OR, 1327-30, 2012.

[73] L. Ye and E. J. Keogh. *Time Series Shapelets:A New Primitive for Data Mining*. KDD 2009.

[74] S. E. Yuksel , J. N. Wilson and P. D. Gader. *Twenty years of mixture of experts*, IEEETrans. Neural Netw. Learn. Syst., vol. 23, no. 8, pp.1177 -1193 2012.

[75] Y. Yunusova, J. R. Green and A. Mefferd. *Accuracy assessment for AG500 electromagnetic articulograph*. Journal of Speech, Language, and Hearing Research, 52, 547-555, 2009.Q. Zhu, E. J. Keogh: *Mother Fugger: Mining Historical Manuscripts with Local Color Patches*. ICDM 2010: 699-708

[76] Project webpage: https://sites.google.com/site/dtwAdaptive, While under review, files are pass protected by `DMKD2015`