Discovery of Meaningful Rules in Time Series

Mohammad Shokoohi-Yekta Yanping Chen Bilson Campana Bing Hu Jesin Zakaria Eamonn Keogh
University of California, Riverside
{mshok002, ychen053, bcampana, bhu002, jzaka001, eamonn} @cs.ucr.edu

ABSTRACT

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. In this work, we show why these ideas are 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.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Application – *Data Mining*

Keywords

Rule Discovery, Prediction, Motif Discovery, Time Series

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 [16], medical interventions [28], 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*" [10].

KDD'15, August 11 – 14, 2015, Sydney, NSW, Australia

DOI: http://dx.doi.org/10.1145/2783258.2783306

- 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*" [19].

Most of the current work has attempted to predict the future based on the current *value* of a stream [18]. 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 paper, *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 [12], 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 work, 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.

2. BACKGROUND AND RELATED WORK

In a sequence of papers culminating in [21], 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" [29]. 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 [30]. This suggests their original results did not outperform random guessing.

The most referenced time series rule-finding method in the literature is [5], 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 [12]. 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 [12]. In Section 7.7 we have compared our algorithm to the three highly cited rival methods.

For brevity, we forgo an in-depth review related work here, referring the reader to an expanded version of this paper [31]. For more background on MDL we direct the interested reader to [1][15]. We will discuss our use of MDL in Section 5.1.

3. 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 [4], 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:

oncexauponwamidnightmtdrearydwhileulpponderediweak...

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:

qncexauponwamidmightmtdreerydwgileulpponderediweek...

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:

*...**chamber**door...*, *...**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."

3.1 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 1, 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.

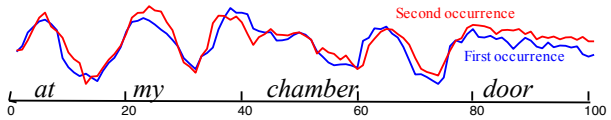

Figure 1. The motif pair discovered in the first 2,000 data points (20 seconds) of "The Raven". The shape corresponds to the utterance "*...at my chamber door*".

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 [19][20].

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 2 shows the rule.

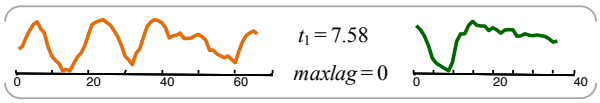

Figure 2. A rule learned (Figure 1) from the first 2,000 data points of the "The Raven". If the antecedent pattern (*left*) is matched to a subsequence in a stream that is within Euclidean distance of 7.58 to it, we predict the immediate occurrence of the consequent pattern (*right*).

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 3 which shows accelerometer data collected from a device worn by a student at USC as he went about daily activities [22].

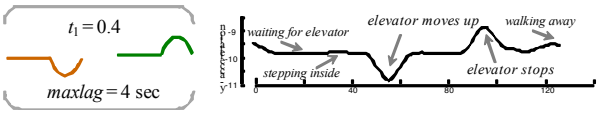

Figure 3. *left*) A rule for an accelerometer dataset encodes the fact that the initial acceleration "bump" of going *up* in an elevator must be eventually be matched by the elevator stopping at a floor. *right*) Real data from which this rule was learned [22].

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.

4. THE 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 [7]. Furthermore, Euclidean distance is parameter-free, fast to compute, and is amiable to various data mining optimizations such as indexing and early abandoning computation [20]. 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* [20]. 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.

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 [15]. After consideration of the many quantization options, we quantize the time series' real values into uniformly sized 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 sized 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 [26]. 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.

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.

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%

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.

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 [27]). In particular, we have adopted (cf. [11]) 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 [18]), 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) $\leq t \rightarrow$ **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) $\leq t \rightarrow$ **doorwithlikename**, which when fired makes a prediction of length sixteen. While encountering the same string:

*... bustabovehis**chamber**door**with**~~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][15]. 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 4) 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 $DL(H) = \text{length}(H) \times \log_2(\text{cardinality})$.

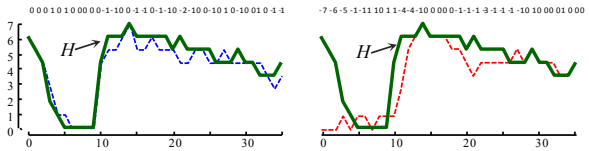

Figure 4. A hypothesis (green/bold) can be used to score subsequences by subtracting it from them (producing the small integers shown *top*) and encoding the difference vector with Huffman encoding. Intuitively, here the *left* sequence requires 57 bits, whereas the *right* sequence requires 84.

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 4. 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*.

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 6.2 we will present our rule search method which utilizes this function.

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 1) 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 2), 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 ← 1 to 99 do                    //Test over all splitting points
2     splitPoint ← i / 100
3     ruleScore(i) ← Best_Rule_Score (T, R, splitPoint) //Table 3
4  end for
5   s ← max (ruleScore)
6   sp ← find (ruleScore == s) / 100
7   a ← R (1 : sp × Length(R))
8   c ← R (sp × Length(R) + 1: 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 4.*left,* our "prediction" was accurate we will save many bits. A less accurate prediction (Figure 4.*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* [20]). 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 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 5). We further justify a probabilistic early abandoning approach in our supporting webpage [31]. 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      s1 ← AntecedentCandidates(i) + antecedentLength +1
9      s2 ← AntecedentCandidates(i) + Length(R)
10     subConsequent ← T (s1 : s2)
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.

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 [19][20]. 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 [20] 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* [20]. 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 5.

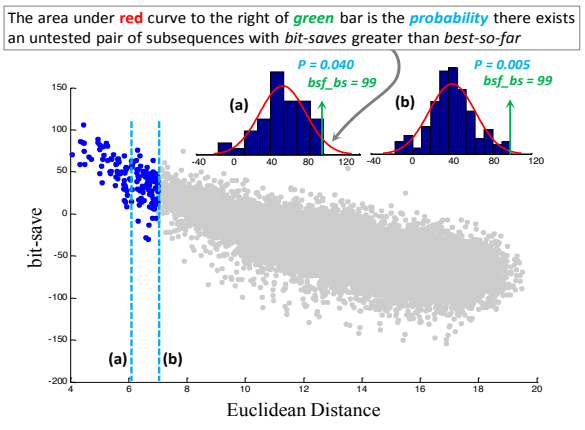

Figure 5. *bottom*) The empirical relationship between Euclidean and bit-save. *top*) As we search in Euclidean order (the x-axis order) from left to right, the expected value of the bit-save (the mean of the Gaussians) decreases.

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 5.*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 5.*bottom* to form the histogram shown at the top left of Figure 5.

The bit-saves property of the distribution can be realized by a Gaussian process (GP) [8]. 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 5.*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 5.*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 [31]), and 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 5 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 6 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 [14], and an even larger proportion of *phrases* must be unique. This observation is for *text* and, as Figure 6 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.

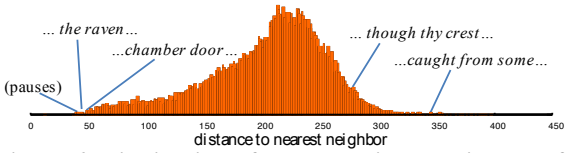

Figure 6. Distribution of nearest neighbor distances for one second snippets of the audio (in MFCC space) of "The Raven."

6.3 Generalizing to allow a Maxlag

Our rule discovery algorithm described in Section 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 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
```

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 many experiments that are omitted here for brevity [31]. The visualization of the rules suffers from space limitations/BW formatting; we encourage the reader to view high-resolution color versions at [31].

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.

We compared our work to the three most obvious and widely cited rival methods. None perform above chance levels, therefore for brevity and clarity we push the details of these comparisons to the expanded version of our paper in [31].

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 weigh in on the "nature vs. nurture" debate [13], but here, simply 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 7.

Figure 7. *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 7.*right* looks plausible, but does it generalize to the test set? In Figure 8 we show one rule firing on an excerpt of the test set.

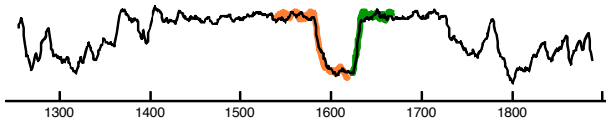

Figure 8. The rule learned in Figure 7 fires (bold/orange) on the test set of day 40 (only an excerpt is shown). The *Q* (cf. Section 7) for the fired rule is 0.33, suggesting high accuracy.

It is interesting to ask if the discovered rule generalizes over time, as the bird's song evolves (i.e. *concept drift*). To test this we apply the learned rule in Figure 7 to the zebra finch song from day 50.

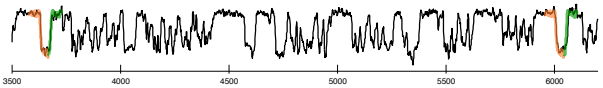

Figure 9. The rule learned in Figure 7 is applied to the same Zebra finch ten days later. The *Q* for the left and right instances are 0.19 and 0.40 respectively.

The low *Q*-scores in Figure 9 indicate that the rule discovered on day 40 (Figure 7) still generalizes. In Figure 10 we repeat the same experiment for day 100.

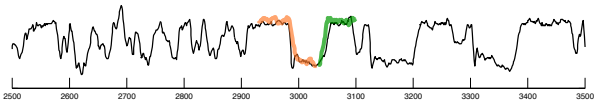

Figure 10. The rule learned in Figure 7 is applied to the singing of the same Zebra finch sixty days later.

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 [31] we show many addition experiments in this domain, and allow the reader to actually *hear* the data/rules.

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 [17].

For our experiments we consider a single meter into which the Fridge, Dishwasher, Clothes Washer and Clothes Dryer are plugged in. We run our rule discovery algorithm on the first month of the data and find the rule shown in Figure 11.*right* as one of the top rules. Then we apply it to our test set and show one of the firings in Figure 11.*left.*

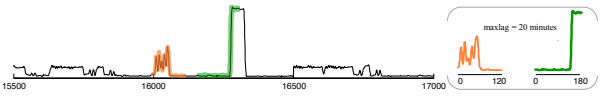

Figure 11. *right)* One of the top rules learned from the first month of the AMPds data set. *left*) One firing of the learned rule in the *right* which contains a 20 minute lag between its antecedent and consequent.

The antecedent and consequent in Figure 11.*right* correspond to the **Clothes Washer** and **Clothes Dryer** respectively. The discovered rule in Figure 11.*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 11.*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 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 11.*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, rinse/spin etc). Automatically generalizing to handle such situations is ongoing work.

7.3 Finding Rules in an Activity Data Set

We consider a benchmark data set that contains daily activity telemetry [23], of four subjects wearing seven inertial measurement units (IMUs). Each subject created five recordings,

which we randomly divided into disjoint train/test partitions. We consider only the data from the right upper arm. Figure 12 shows one of the top rules learned from a recording of a subject.

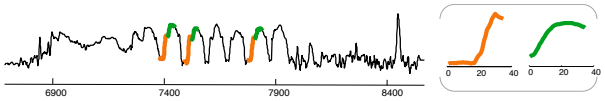

Figure 12. *left*) A subsequence from a recording of a subject that contains a rule discovered by our algorithm. The discovered locations (orange/bold) are generalized by our algorithm into a rule (*right*).

To test if the discovered rule generalizes to other instances of the activity, we applied the discovered rule in Figure 12 to other recordings of the same subject. Figure 13 shows the rule firings found in the test recording.

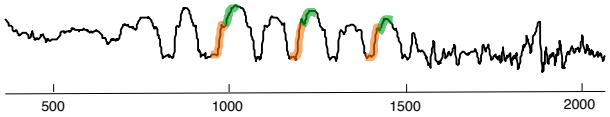

Figure 13. Three instances of the rule shown in Figure 12 discovered in the test set. The *Q* for the three instances from left to right are 0.22, 0.10 and 0.41 respectively.

According to the labels provided with the dataset, the rule discovered is a part of the activity: *drinking from a cup while standing*. 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 14.*top* shows some stills from the video. The time series of the complete activity from the IMU is shown in Figure 14.*bottom*.

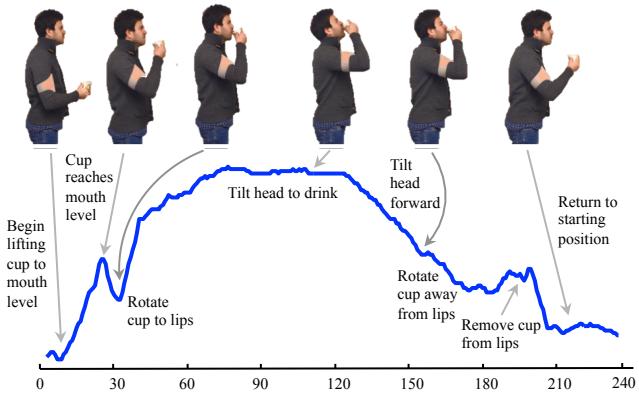

Figure 14. *bottom*) The IMU data of an actor drinking from a cup. *top*) Stills from a video aligned with the IMU.

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 14, 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.*

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 [9]. Figure 15 shows one of the top rules learned from this time series where the first peak in Figure 15.*left* is that of a failed solenoid.

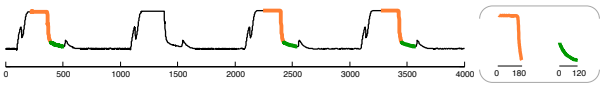

Figure 15. *left*) A snippet of the NASA data. *right*) The first ranked rule learned which characterizes a nominal discharge.

We applied the discovered rule in Figure 15 to the test set and found the rule fires four times. Figure 16 shows the rule instances fired on the test set.

Figure 16. The rule discovered in Figure 15 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 16. According to the domain experts [9], 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, we can imagine using the *negation* of the rule firing as an anomaly detector.

7.5 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 [5][29] (as [12] 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.

7.6 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 [20][25]. 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.

7.7 On Comparisons to Rival Methods

We compared our algorithm to the two (very different) Piecewise Linear Approximation (PLA) based approaches in [21] and [29], and also the highly cited paper [5]. 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 all data sets in Section 7, the best results for all approaches had Q values, measure of quality, at the default rate (consistent with random guessing). Due to space limitations, we push the details of these comparisons to the expanded version of our paper in [31].

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. We have also demonstrated our method by comparing it to the three most widely cited rival methods and show how we make much more accurate predictions [31].

There are many avenues for future work. On some datasets, Dynamic Time Warping, in single or multi-dimensional cases, may be more robust than the Euclidean distance, adapting to the concept drift that will be inevitable in some applications [24], 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.

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

However this would require significantly more training data to guard against overfitting. Such flexibility would allow a rule to consider antecedents from two different sources. Finally, unlike time series classification [26], there are currently no standard benchmarks for time series rule discovery. We plan to repair this omission, and invite the community to donate challenging datasets for which the ground truth is known, and archiving them [31].

Acknowledgements

We would like to thank all the donors of the datasets. We further wish to acknowledge funding from NSF IIS-1161997 II and a gift from Samsung Research.

9. REFERENCES

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

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

[6] 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)

[7] Ding, H., et al. Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB 1(2): 1542-1552. 2008.

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

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

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

[11] Hu, B., et al. Discovering the Intrinsic Cardinality and Dimensionality of Time Series Using MDL. ICDM 2011:

[12] 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.

[13] 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.

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

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

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

[17] 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.

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

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

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

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

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

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

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

[25] 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.

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

[27] Wallace, C., and Dowe, D. Minimum message length and Kolmogorov complexity. Computer Journal vol. 42-4, 1999.

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

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

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

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