Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels

Shaghayegh Gharghabi, Yifei Ding, Chin-Chia Michael Yeh, Kaveh Kamgar, Liudmila Ulanova, Eamonn Keogh
Department of Computer Science and Engineering
University of California, Riverside
{sghar003, yding007, myeh003, kkamg001, lulan001}@ucr.edu eamonn@cs.ucr.edu

***Abstract*— Unsupervised semantic segmentation in the time series domain is a much-studied problem due to its potential to detect unexpected regularities and regimes in poorly understood data. However, the current techniques have several shortcomings, which have limited the adoption of time series semantic segmentation beyond academic settings for three primary reasons. First, most methods require setting/learning many parameters and thus may have problems generalizing to novel situations. Second, most methods implicitly assume that all the data is segmentable, and have difficulty when that assumption is unwarranted. Finally, most research efforts have been confined to the batch case, but online segmentation is clearly more useful and actionable. To address these issues, we present an algorithm which is domain agnostic, has only one easily determined parameter, and can handle data streaming at a high rate. In this context, we test our algorithm on the largest and most diverse collection of time series datasets ever considered, and demonstrate our algorithm's superiority over current solutions. Furthermore, we are the first to show that semantic segmentation may be possible at superhuman performance levels.**

Keywords—Time Series; Semantic Segmentation; Online Algorithms;

I. Introduction

The ubiquity of sensors and the plunging cost of storage has resulted in increasing amounts of time series data being captured. One of the most basic analyses one can perform on such data is to segment it into homogenous regions. We note that the word "segmentation" is somewhat overloaded in the literature. It can refer to the approximation of a signal with piecewise polynomials [13], or the division of a time series into internally consistent regimes. For clarity, this latter task is sometimes called "semantic segmentation" [1][31]; where there is no danger of confusion, and we will refer to it as *segmentation*. Sometimes, it can be fruitful to see segmentation as a special type of clustering with the additional constraint that the elements in each cluster are contiguous in time.

The utility of segmentation is myriad. For example, if one can segment a long time series into k regions (where k is a small), then it may be sufficient to show only k short representative patterns to a human or a machine annotator in order to produce labels for the entire dataset. Moreover, as an exploratory tool, sometimes we can find unexpected and actionable regularities in our data.

While there are many techniques for segmentation [1][15] [17][19][25], they all have one or more limitations that have prevented their utilization in real world settings. This observation has motivated us to introduce FLOSS (Fast Low-cost Online Semantic Segmentation), a novel algorithm which, to the best of our knowledge, is unique in offering all the following features:

- **Domain Agnosticism**: Most techniques in the literature are implicitly or explicitly suited to a *single* domain, including motion capture [1][16], motion capture of *upper-body only* [3], electroencephalography [15], music [28], automobile trajectories [11], or electrical power demand [25]. For example, the detailed survey in [17] notes that for almost all methods "*some prior knowledge of the nature of the motion is required*." In contrast, FLOSS is a domain agnostic technique that makes essentially no assumptions about the data.
- **Streaming**: Many segmentation algorithms are only defined for batch data [1][15]. However, a streaming segmentation of data may provide *actionable* real-time information. For example, it could allow for medical intervention [30] or a preemptive repair to a machine that has entered a failure mode [22].
- **Real World Data Suitability**: Most techniques assume that every region of the data belongs to a well-defined semantic segment. However, that may not be the case. Consider a wrist-worn accelerometer data worn by an athlete working out at a gym. Examined at the scale of tens of seconds, there will be many well-defined homogenous regions of behavior, corresponding to various repetitions on the apparatus (see Fig. 1). However, it is probable that there are many minutes of behavior that accumulated while the athlete was waiting her turn to use a machine. These periods may be devoid of structure. Any model that insists on attempting to explain *all* of the data may be condemned to poor results. In contrast, FLOSS can effectively mark these difficult sections as "don't know".

Beyond introducing the FLOSS algorithm, we claim the following contributions to the literature:

- Most research efforts in this domain test on limited datasets [1][15]. The authors of [19] and [32] are both to be commended for considering *three* datasets, but they are exceptional, considering *one* dataset is the norm. In contrast, we test on a data repository of thirty-two datasets from diverse domains. We believe that this free public archive will accelerate progress in this area, just as the TREC datasets have done for text retrieval and the UCR archive has done for time series *classification* [8].

- While classification, clustering, compression etc. all have formal and universally accepted metrics to assess progress and allow meaningful comparison of rival methods, the evaluation of segmentation algorithms has often been anecdotal [17]. Evaluation often reduces to the authors asking us to visually compare the output of their algorithm with the ground truth. While there is nothing wrong with visually compelling examples or anecdotes, it is clearly desirable to have more formal metrics. In [19], the authors adapt precision/recall, but in some contexts, this is unsuitable for semantic segmentation. In Section III-D, we introduce a metric that allows us to meaningfully score segmentations given some external ground truth.

The rest of this paper is organized as follows. In Section II, we provide a summary of the background and related work along with the necessary definitions. In Section III-A, we introduce a batch algorithm for semantic segmentation before generalizing it to the streaming case in Section III-C. Section 0 illuminates a detailed quantitative and qualitative evaluation of our ideas. Finally, in Section IV, we offer conclusions and directions for future work.

II. Background and Related Work

In this section, we introduce all the necessary definitions and notations and consider related work. Because the term segmentation is so overloaded, even in the limited context of time series, we also explicitly state what we are *not* attempting to do in this work.

Note that for clarity and brevity, our definitions and algorithms only consider the one-dimensional cases; however the generalizations to the multidimensional case are trivial, and we have archived multidimensional examples and code in [4].

A. Definitions

Here, we introduce the necessary definitions and terminology, beginning with the definition of a time series:

Definition 1: A time series $\boldsymbol{T} = t_1, t_2, t_3, \ldots, t_n$ is a continuous ordered sequence of real values in equally spaced time intervals of length *n*.

Our segmentation algorithm will exploit the similarity of local patterns within $\boldsymbol{T}$, called *subsequences*:

Definition 2: A *subsequence* $\boldsymbol{T}_{i,L}$ of a $\boldsymbol{T}$ is a continuous subset of the values from $\boldsymbol{T}$ of length L starting from position i. $\boldsymbol{T}_{i,L} = t_i, t_{i+1}, \ldots, t_{i+L-1}$, where $1 \leq i \leq n\text{-}L+1$.

The time series $\boldsymbol{T}$ is ultimately recorded, because it is measuring some aspect of a system $\boldsymbol{S}$ (perhaps *indirectly* measuring the phenomenon in some instances).

Definition 3: A system $\boldsymbol{S}$ is a physical or logical process containing two or more discrete states separated by one or more boundaries $\boldsymbol{b}$.

We further discuss and justify our assumption that $\boldsymbol{S}$ is intrinsically discrete in Section III.

The algorithms we present are built on the recently introduced Matrix Profile (MP) representation as well as the STAMP and STAMPI (the online variation) algorithms used to compute it [31]. We briefly review of these in the next section.

B. Matrix Profile Background

STAMP is a time series all-pairs one-nearest-neighbor search (also known as *similarity join*) algorithm that leverages the Fast Fourier Transform for speed and scalability. The input parameters are the time series data $\boldsymbol{T}$ and a subsequence length L, where L is the desired length of the time series pattern to search for. For output, it returns two vectors, *MPValues* and *MPIndex*, both of which are the same length of $\boldsymbol{T}$ and can be seen as annotating it. At the index *i* of the data structure…

- *MPValues*, is the Euclidean distance of the subsequence $\boldsymbol{T}_{i:i+L}$ to its nearest neighbor elsewhere in $\boldsymbol{T}$. To prevent *trivial matches*, where the subsequence matches to itself, an exclusion region is enforced such that the distance between $\boldsymbol{T}_{i:i+L}$ and any subsequence begins at $[i - {}^{L}/_{2}: i + {}^{L}/_{2}]$ and is assumed to be *infinity*.
- *MPIndex* is the location of *i*'s nearest neighbor in $\boldsymbol{T}$. Note that in general, this nearest neighbor information is not symmetric, *i*'s nearest neighbor may be *j*, but *j*'s nearest neighbor may be *k*.

This review is necessarily brief, so we refer the reader to the original paper for more details [31].

C. What FLOSS is Not

Even within the narrow context of time series analytics, the term *segmentation* is overloaded; thus, it is necessary to explicitly explain some tasks we are *not* addressing.

Change point detection is a method for detecting various changes in the statistical properties of time series, such as the mean, variance, or spectral density. A good representation of the literature on this problem is surveyed in detail in a recent paper in [1]. In contrast to change point detection, we are interested in regimens that are defined by changes in the *shapes* of the time series subsequences, which can change without any obvious effect on the statistical properties.

Similar to our stated goals, recent work on change point detection has begun to stress the need to be parameter-free and have few assumptions [21]. However, scalability is rarely a priority; therefore, a typical dataset considered in this domain is a few hundred data points. This indicates that human inspection is often a competitive algorithm. However, due the scale of the data we wish to consider and the necessity to detect regime changes where they would be difficult to discern visually on the screen, an algorithm that is competitive with or even surpasses human inspection is necessary.

Another interpretation of segmentation refers to Piecewise Linear Approximation (PLA). The goal is to approximate a time series $\boldsymbol{T}$ with a more compact representation by fitting k piecewise polynomials using linear interpolation or linear regression, while minimizing the error with respect to the original ***T*** [11] [29]. Success here is measured in terms of root-mean-squared-error, and it does not indicate any semantic meaning of the solution.

Finally, we are not interested in segmenting *individual* phrases/gestures/phonemes etc. This type of work is almost always heavily domain dependent and requires substantial training data [3]. For example, there is a significant amount of work that attempts to segment the time series equivalent of the string *nowthatchersdead* to produce *now thatchers dead* (and not *now that chers dead*). In contrast, we are interested in segmenting at a higher level, which would be the equivalent of segmenting an entire book into chapters or themes.

D. *Related Work*

Hidden Markov Models (HMMs) have been successfully used to segment discrete strings. Examples of this include segmenting a DNA strand into coding and non-coding regions, and the efforts to use HMMs in the real-valued space (but they are almost always tied to a single domain, such as seismology [7]). We have considered and dismissed HMMs for several reasons. To use HMMs with real-valued time series, we must set at least two parameters, the level of cardinality reduction (the number of states to discretize to) and the level of dimensionality reduction (the number of values to average) [7]. This is in addition to specifying the HMM architecture, which is tricky even for domain experts [7] and contrary to our hope for a domain agnostic algorithm.

The work that most closely aligns with our goals is Autoplait [19], which segments time series using Minimum Description Length (MDL) to score alterative HMMs of the data. This work also stresses the need for domain independence and few parameters. The most significant limitation of Autoplait is that it is only defined for the batch case. It would not be trivial to convert it to handle streaming data. This approach requires discrete data, which is obtained by an equal division of the range bound by the smallest and largest value seen. In the streaming case, wandering baseline or linear drift ensures that at some point all the incoming values are greater (or smaller) than the values the model can process. This is surely not unfixable, but it is also not simple to address, and it is only one of the many issues that must be overcome to allow an Autoplait variant handle streaming data.

The authors of Autoplait (and various subsets thereof) have many additional papers in this general space. However, to the best of our understanding, none of them offer a solution for the task-at-hand. For example while StreamScan is a streaming algorithm [20], the authors note the need to train it: "*we trained several basic motions, such as 'walking', 'jumping*'..", and the algorithm has at least six parameters.

III. SEMANTIC SEGMENTATION

We are finally able to formally define the task-at-hand. Assume we have a system ***S***, which can be in two or more discrete states. Examples of such systems include:

- The heart of a patient recovering from open heart surgery. The patient's heart may be in the state of *tamponade* or *normal* [9].
- A music performance may be envisioned a system that moves between the states of *intro*, *verse*, *chorus*, *bridge*, and *outro* [28].
- Fractional Distillation of petrochemicals consists of cycles of *heating*, *vaporizing*, *condensing*, and *collecting* [24].
- An exercise routine often consists of *warm-up*, *stretching*, *resistance training*, and *cool-down*. This special case of treating human behavior as a switching linear dynamic *system* (SLDS) [27] has become an increasingly popular tool for modeling human dynamics [6][26].

We can monitor most of these systems with sensors. For the cases mentioned above, a photoplethysmograph, a microphone, a thermocouple, and a wrist mounted accelerometer (smartwatch) are obvious choices. In most cases, we would expect the time series from the sensors to reflect the current state of the underlying system. This understanding allows us to produce the following definition of the problem regarding the time series semantic segmentation task:

Definition 4: Given a time series ***T***, monitoring some aspect of a system ***S***, infer the boundaries ***b*** between changes of state.

We recognize that this definition makes some simplifying assumptions. Some systems are not naturally in discrete states, but may be best modelled as having a degree of membership to various states. For example, *Hypokalemia*, a disease where the heart system is deficient in potassium, is often diagnosed by examining ECGS for increased amplitude and width of the P-wave [30]. Hypokalemia can manifest itself continuously at any level from mild to severe. In fact, our example of tamponade is one of the few intrinsically discrete heart conditions. Nevertheless, many systems *do* switch between discrete classes, and these are our domains of interest. Moreover, even though hypokalemia can *change* continuously, in practice it often changes fast enough (in response to intravenous or oral potassium supplements) to be detectible as a regimen change in a window of ten minutes, and we can easily support windows of this length.

Note that even in systems that do have some mechanism to "snap" the system to discrete behaviors, there is often another ill-defined "other" class. For example, consider the short section of time series shown in Fig. 1. Here the need for precise movements forces the exercise repetitions to be highly conserved. However, there is no reason the transitions between the repetition sets need to be conserved.

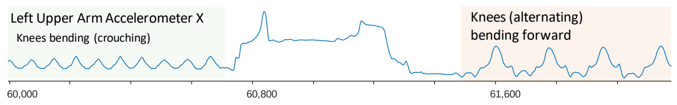

Fig. 1: A snippet of time series collected during an exercise routine. Both the first and last third are well-defined motions, but the section in the middle is less structured, representing a transition between apparatuses.

Similar remarks apply to many other domains. In fact, in many cases, the majority of the data examined may consist of ill-defined and high entropy regions. Note that we cannot use these observations to conclude that the underlying system is not in any state. It may simply be the case that the view given by our sensor is not adequate to make this a determination. For example, a sensor on the ankle will help distinguish between the states of walking and running, but it will presumably offer little information when the system (the human) is toggling between *typing* and *mouse-use*.

A. *Introducing FLUSS*

We begin by introducing FLUSS (Fast Low-cost Unipotent Semantic Segmentation), an algorithm that extends and modifies the (unnamed) algorithm hinted at in [31]. Later, in Section III-C, we will show how we can take this intrinsically batch algorithm and make it a streaming algorithm.

The task of FLUSS is to produce a companion time series called the Arc Curve (AC), which annotates the raw time series with information about the likelihood of a regime change at each location. We also need to provide an algorithm to examine this Arc Curve and decide how many (if any) regimes exist. We consider that issue separately in Section III-B.

FLUSS takes both a time series $\boldsymbol{T}$ and a user provided subsequence length in as input, and outputs an AC vector of length n, where at each index i contains the number of “arcs” that cross over i. We define an “arc” as follows. The ith entry in the *MPIndex* vector contains a positive integer j, which indicates the nearest neighbor location. So, for the ith entry, containing a positive integer j, the nearest neighbor for the time series subsequence beginning at index i is the time series subsequence beginning at index j. We can visualize each entry pair (i,j) as an arc drawn from location i to j. The spatial layout of the arcs along with the number of “arc” crossing over of each index i is summarized by the Arc Curve. Specifically, index i of the Arc Curve contains a non-negative integer indicating the number of arcs that cross over i. Fig. 2 below illustrates this notation.

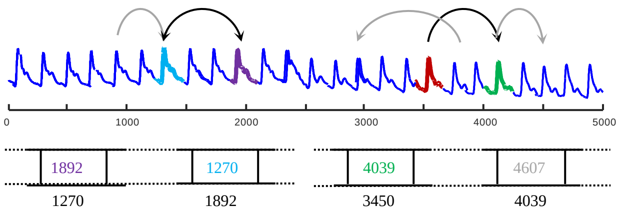

Fig. 2: Selected arcs illustrated with the corresponding Matrix Profile indices indicated. Note that nearest neighbor subsequences indices can be symmetric, e.g., 1270 and 1892, but this is not generally true. A subsequence’s nearest neighbors can be located to the left or to the right.

Note that every index has exactly one arc leaving it; however, each index may have zero, one, or multiple arcs pointing to it. We define the Arc Curve more formally below:

Definition 5: The Arc Curve (AC) for a time series $\boldsymbol{T}$ of length n is itself a time series of length n containing non-negative integer values. The ith index in the AC specifics how many nearest neighbor arcs from the *MPIndex* spatially cross over location i.

Now, we can state the intuition of our segmentation algorithms.

Our Overarching Intuition: Suppose a time series $\boldsymbol{T}$ has a regime change at location i. We would expect few arcs to cross i, as most subsequences will find their nearest neighbor *within* their host regime. Thus, the height of the Arc Curve should be the lowest at the location of the boundary between the change of regimes/states.

In Fig. 3, we show the AC plot for the dataset shown in Fig. 2, which will be used as a running example.

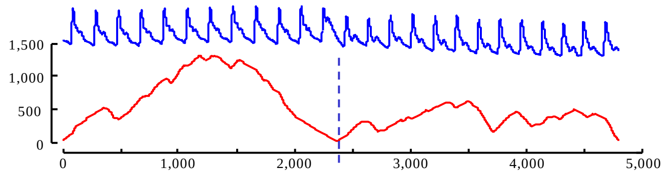

Fig. 3: *top*) The ABP of a reclining male. At time 2400, he was rotated into a standing position. *bottom*) The AC plot for this dataset shows a clear valley at time of system change.

We consider the Arterial Blood Pressure (ABP) of a healthy volunteer resting on a medical tilt table [12]. At time 2400, the table was tilted upright, invoking a response from the homeostatic reflex mechanism. While the figure above hints at the utility of FLUSS, it also highlights a weakness. Note that while the Arc Curve has a satisfyingly low value at the location of the regime change, it also has low values at both the leftmost and rightmost edges. This occurs because there are simply fewer candidate arcs that can cross a given location at the edges. We need to compensate for this bias, or we are likely to report false positives near the edges.

This compensation is easy to achieve. We begin by imagining the case where there is *no* locality structure in the time series under consideration; for example, imagine we are examining a random vector. Under such circumstances, we would expect the arcs from each subsequence to point to an effectively random location. Given this null case, with no structure, what would an Idealized Arc Curve (IAC) look like? With a little introspection, we can see that, as shown in Fig. 4, it would be an inverted parabola with its height ½n (we relegate the derivation of this fact to [4]).

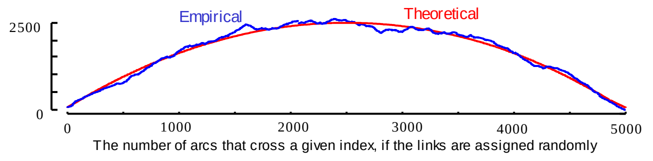

Fig. 4: The Idealized Arc Curve (IAC) for a time series with no localized similarity structure is an inverted parabola with a height ½n. An empirical curve shows close agreement. As we will see later, it is actually a special case of beta (2, 2, a, c).

To compensate for the edge effect bias in the Arc Curve, for each location i, we consider the actual number of observed arc crossings relative to the number of expected arc crossings predicted by our parabolic model (1), to obtain the Corrected Arc Crossings (CAC):

$$\mathrm{CAC}_i = \min\left(\frac{\mathrm{AC}_i}{\mathrm{IAC}_i}, 1\right) \tag{1}$$

The min function is used to keep the CAC bounded between 0 and 1 in the logically possible (but never empirically observed) case that $AC_i > IAC_i$.

This normalized and bounded measure is useful because it allows the following:

- Commensurate comparisons across streams monitored at different sampling rates.
- The possibility to learn domain specific *threshold* values. For example, suppose we learn in ECG training data, that for a patient in an ICU recovering from heart surgery, a CAC value less than 0.2 is rarely seen unless the patient has cardiac tamponade. Now we can monitor and alert for this condition.

In Fig. 5, we show the CAC for our running example. Note that the issue of the edge bias of AC has been resolved, and the curve minimizes at the correct location of 2400.

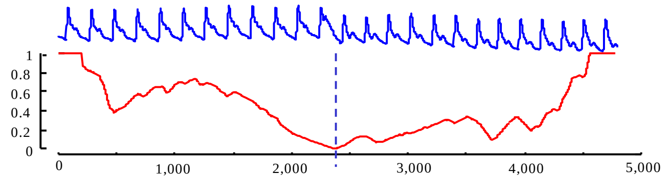

Fig. 5: (contrast with Fig. 3). *top*) Our running ABP example. *bottom*) The CAC minimizes in the correct place and avoids the "edge-effect" false positives of the AC curve.

Before continuing, we will demonstrate the invariance of the CAC to several issues commonly encountered during real-world uses. We recomputed the CAC for our running example after modifying the original data in several ways, including:

- Downsampling from the original 250 Hz to 125 Hz (red).
- Reducing the bit depth from 64-bit to 8-bit (blue).
- Adding a linear trend of ten degrees (cyan).
- Adding 20dB of white noise (black).
- Smoothing, with MATLAB's default settings (pink).
- Randomly deleting 3% of the data, and filling it back in with simple linear interpolation (green).

As Fig. 6 suggests for this example, the CAC is quite robust in regard to these issues, and the minimum value of the CAC still occurs in the same place.

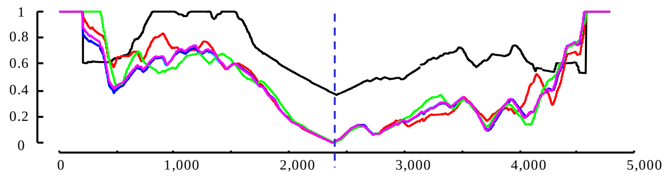

Fig. 6: The CAC computed for our running example after it was distorted in various ways (best viewed in color; the key is above). Compare it to Fig. 5.

The only distortion to appreciably change the CAC is noise; however, as Fig. 7 shows, we added an enormous amount of noise, and we still found the correct segmentation.

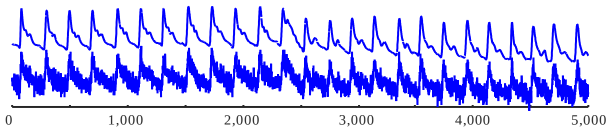

Fig. 7: A comparison of the original TiltABP (*top*) and the data with twenty dB of noise added (*bottom*).

We have shown that the CAC is robust to many variations of time series data, and are now ready to fully explain the algorithm for obtaining the CAC. While the construction of the CAC is straightforward, given the discussion above, we formalize it on TABLE I for clarity.

In lines 1 to 2, we obtain the length of the *MPIndex* and zero initialize three vectors. Next, we iterate over the *MPIndex* to count the number of arcs that cross over index *i* in lines 3 through 7. This information is stored in *nnmark*. Then, we iterate over *nnmark* and cumulatively sum its values consecutively for each index *i*. The cumulative sum at *i* is stored in AC_i. This is accomplished in lines 10 to 13. Finally, in lines 15 to 18, we normalize AC with the corresponding parabolic curve to obtain the CAC.

TABLE I: Algorithm for constructing CAC

```
Procedure CAC (MPIndex, L)
Input-  MPIndex: the matrix profile index for the time series of interest
        L: the subsequence length
Output- CAC: a corrected Arc Curve
1   n = length(MPIndex)
2   AC = CAC = nnmark = zero initialize array of size n
3   for i=1:n
4     j = MPIndex[i]
5     nnmark[min(i,j)] = nnmark[min(i,j)] + 1
6     nnmark[max(i,j)] = nnmark[max(i,j)] - 1
7   end
8
9   numArcs = 0
10  for i=1:n
11    numArcs = numArcs + nnmark[i]
12    AC[i] = numArcs
13  end
14
15  IAC = parabolic curve of length n and height ½ n
16  CAC = min (AC/IAC, 1)
17  Set the L length in the beginning and end of CAC to 1
18  return CAC
```

B. *Extracting Regimes from the CAC*

With our CAC defined, we are now ready to explain how we extract the locations of the regime changes from the CAC. Our basic regime extracting algorithm requires the user to input *k*, the number of regimes. This is similar to many popular clustering algorithms, such as *k*-means, which require the user to input the *k* number of clusters. However, later we will show a technique to remove the need to specify *k* when given some training data to learn from.

As suggested in Fig. 5, a small value for the lowest "valley" at location *x* is strong evidence of a regime change at that location. This is based on the intuition that significantly fewer number of arcs would cross location *x* if *x* is a boundary point between two discrete states [31]. Note that this intuition is somewhat asymmetric. A large value for the lowest valley indicates that there is no evidence of a regime change, not that there is positive evidence of no regime change. This is a subtle distinction, but it is worth stating explicitly.

At a high level, the regime extracting algorithm (REA) searches for k lowest "valley" points in the CAC. However, we need to avoid the trivial minimum; if x is the lowest point, then it is almost certain that either x+1 or x-1 is the second lowest point. To avoid this, FLUSS does not return the k minimum values. Instead, it obtains one minimum "valley" value at location x. Then, FLUSS sets up an exclusion zone surrounding x. For simplicity, we have defined the zone as five times the subsequence length before and after x. This exclusion zone is based on an assumption that regimes will have multiple repetitions; FLUSS is not able to segment single gesture patterns. With the first exclusion zone in place, FLUSS repeats the process described above until all k boundary points are found.

While this algorithm is obvious and intuitive, for concreteness, we formally outline in TABLE II.

TABLE II: REA: Algorithm for Extracting Regimes

```
Procedure ExtractRegimes  (CAC, numRegimes, L)
Input- CAC: a corrected Arc Curve
       numRegimes: number of regime changes
       L: length of the subsequence
Output- locRegimes: the locations of the regimes

1   locRegimes = empty array of length numRegimes
2   for i=1:numRegimes
3      locRegimes(i) = indexOf(min(CAC))
4      Set exclusion zone of 5*L
5   end
6   return locRegimes
```

C. Introducing FLOSS

In the previous sections, we have shown that at least in our running example, FLUSS can detect changes of regimes in batch datasets. Now we consider the streaming case, in which we maintain the CAC over a sliding window; an example of this could be the last ten minutes of a patient recovering from heart surgery. In principle, this seems simple. At every time stamp, we need to ingress the newly arriving point, and egress the oldest point, updating all the arcs in the Matrix Profile index and adjusting the CAC as needed. However, there is a large asymmetry in the time complexity for ingress and egress.

- **Ingress**: When the new point arrives, we must find its nearest neighbor in the sliding window, and determine whether any item currently in the sliding window needs to change *its* nearest neighbor to the newly arrived subsequence. Using the MASS algorithm, this takes just O($n$log$n$) [23].
- **Egress**: When a point is ejected, we must update all subsequences in the sliding window that currently point to that departing subsequence (if any). This is a problem, because while pathological unlikely, almost *all* subsequences could point to the disappearing subsequence. This would force us to do O(n^2) work, forcing us to recompute the Matrix Profile [31].

This issue would not exist if the arcs in the Matrix Profile only went in one direction, to a previous time. In that case, when we egress a data point, for the corresponding subsequence being removed:

- As the arcs only go to a previous time, we do not have to delete arcs that point to it, since it does not have one.
- As for the arcs that point away from it, we could delete that arc by removing the first element in the matrix profile index in O(1).

This would indicate that the overall time to maintain the 1-Direction on CAC (CAC_{1D}) would be only O($n$log$n$) for ingress plus O(1) for egress, for a total of O($n$log$n$).

However, this begs the question, would using the CAC_{1D} yield similar results to using the CAC? To test this, we begin by computing the empirical one directional IAC (IAC_{1D}). The empirical IAC_{1D} is shown with the theoretical original (bi-directional) IAC in Fig. 8.

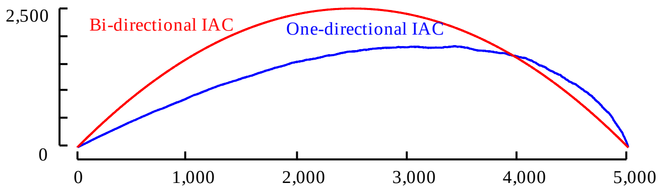

Fig. 8: The one-directional IAC (IAC_{1D}) is a right skewed distribution with shorter height than the original IAC.

Compared to the original IAC, IAC_{1D} has a somewhat similar shape, but it is shorter and skewed to the right. The skewness is caused by the fact that it is more likely for arcs to cross later in time, since all the arcs are pointing forward in time. By theoretical modeling/visual inspection [4], we claim the distribution of IAC_{1D} can be modeled by a beta distribution. The empirical IAC_{1D} and a sample generated by the beta distribution is shown in Fig. 9. Note that in retrospect, we can see the parabolic curve of Fig. 4 as a special case of the beta distribution with $\alpha = \beta = 2$.

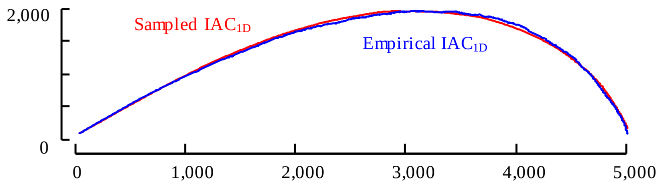

Fig. 9: The empirical IAC_{1D} is modeled closely by a beta distribution.

As a result of this difference, IAC_{1D} is used instead of IAC when computing CAC_{1D}. Below, we computed the CAC_{1D} on our running example, as shown in Fig. 10.

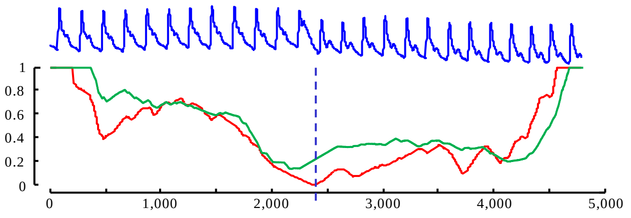

Fig. 10: *top*) Our running ABP example. *bottom*) the CAC (red) from Fig. 5 and the CAC_{1D} (green).

As we can see, the CAC_{1D} minimum value is very close to that of the CAC. Moreover, this is a short “toy” example. As the datasets get larger, the two curves become essentially indistinguishable.

D. *Scoring Function*

Most of the evaluation of segmentation algorithms has been largely anecdotal (see [17] for a detailed survey), and indeed we also show visually convincing examples in Section 0. However, because of the scale of the experiments, i.e. examining thirty-two datasets, we need to have a scoring metric.

Many research efforts have used the familiar precision/recall or measures derived from them. However, as [17] points out, this presents a problem. Suppose the ground truth for transition between two semantic regimes is at location 10,700. If an algorithm predicts the location of the transition at say 10,701, should we score this as a success? What about say 10,759? To mitigate this brittleness, several authors have independently suggested a “Temporal Tolerance” parameter to bracket the ground truth [17]. However, this only slightly mitigates the issue. Suppose we bracket our running example with a range of 100, and reward any prediction in the range 10,700 ± 100. Would we penalize an algorithm that predicted 10,801, but reward an algorithm that predicted 10,800?

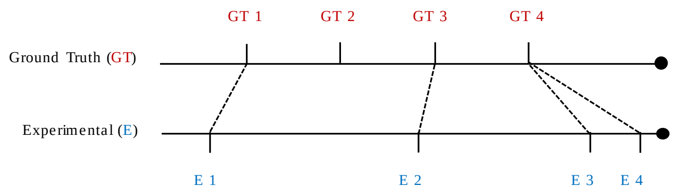

Fig. 11: An example of our scoring function in action. The top line illustrates the locations of the ground truth locations (GT1, GT2, GT3, GT4), and the bottom line illustrates the boundary locations (E1, E2, E3, E4) reported by an algorithm. Note that multiple proposed boundary points may be mapped to a single ground truth point.

Another issue we face in creating a scoring function is rewarding a solution that has *k* boundaries predictions, in which most of the predictions are good, but just one (or a few) are poor. If we insist on a one-to-one mapping of the predictions with the ground truth, we over-penalize any solution for missing one boundary while accurately detecting others (a similar matching issue is understood in many biometric matching algorithms).

TABLE III: Scoring Function Algorithm

```
Procedure ScoreRegimes(locRegimes, gtRegimes, n)
Input-  locRegimes: extracted regimes
        gtRegimes: ground truth regimes
        n: length of the time series
Output- score: [0,1], with 0 being the best score
1  sumDiff = 0
2  numRegimes = length(gtRegimes)
3  for i=1:numRegimes
4     Find the gtRegimes[j] closest to locRegimes[i]
5     diff = | locRegimes[i] – gtRegimes[j] |
6     sumDiff = sumDiff + diff
7  end
8  score = sumDiff/n
9  return score
```

Our solution is visually explained in Fig. 11, and formally outlined in TABLE III. It gives 0 as the best score, and 1 as the worst. The function sums the distances between the ground truth boundary points and the boundary points suggested by an algorithm. Dividing that sum by the product of the number of segments and the length of the time series to normalize the range to [0, 1].

EXPERIMENTAL EVALUATION

We begin by stating our experimental philosophy. We have designed all experiments such that they are easily reproducible. To this end, we have built a web page [4] that contains all of the datasets and code used in this work as well as the spreadsheets containing the raw numbers and some supporting videos. The thirty-two segmentation test datasets we created will be archived in perpetuity at [4], independent of this work. We hope the archive will grow as the community donates additional datasets.

E. *Datasets*

We created an extremely diverse collection of test datasets. The biological datasets include time series taken from humans, birds, rats, pigs, and insects. The mechanical datasets include data taken from robots and electrical power demand (both from a single building and an entire city). The datasets fall into three categories:

- **Synthetic**: There is one completely synthetic dataset, which is mostly for calibration and sanity checks.
- **Real**: The majority of our datasets are real. In most cases, the ground truth boundaries are confidently known because of *external* information. For example, for the Pulsus Paradoxus datasets [9], the boundaries were determined by the attending physician viewing the patient’s Echocardiogram.
- **Semi-Real**: In some cases, we contrived real data to have boundaries. For example, we took calls from a single species of bird that were recorded at different locations (thus they were almost certainly different *individuals*) and concatenated them. Thus, we expect the change of *individual* to also be a change of *regime*.

As Fig. 12 suggests, some of the boundaries are obvious visually. However, we can also see that many are so subtle that finding the boundary is a non-trivial challenge for humans, including domain experts (in fairness, these are only snippets, the excluded data would probably give the human more context).

For brevity, we omit further discussion of these datasets. However, we have created a visual key, which gives detailed information on the provenance of each dataset and placed it in perpetuity at [4].

We set the only parameter, the subsequence length *L*, by visual inspection. We set it to be about one period length (i.e. one heartbeat, one gait cycle, etc.). As we will show in Section IV-0, our algorithm is *not* sensitive to this choice.

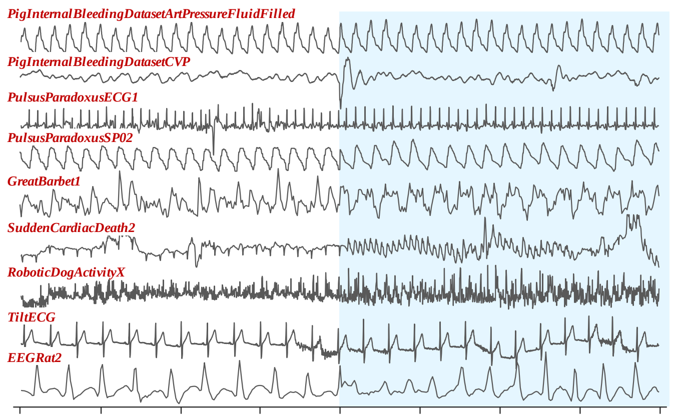

Fig. 12: Snippets (not the full traces) from a random selection of the test datasets. The snippets are centered on a boundary (change of background color). There are no axes in this figure, as the data are at different scales.

F. *Rival Methods*

The most obvious rival method is Autoplait [19]. As noted above, while there are dozens of segmentation algorithms, it is often difficult or unfair to compare to them because:

- They are designed only for a limited domain; thus if they are not competitive, it might be because they are just not suited for some or most of the diverse datasets considered.
- They require setting many parameters; thus if they are not competitive, it might be because we tuned the parameters poorly.
- The code is not publicly available; if they are not competitive, it might be because of our unconscious *implementation bias* [14].

In contrast to all of the above, Autoplait is domain agnostic, parameter-free, and the authors make their high-quality implementation freely available. Autoplait segments time series by using MDL to recursively test if a region is best modeled by one HMM or two (this is a simplification of this innovative work, we refer the interested reader to refer to [19]).

After confirming that we had the code working correctly by testing over the authors' own datasets and some toy datasets, we found that Autoplait only produced a segmentation on 12 out of our 32 test datasets. The underlying MDL model is too conservative. To fix this issue, for every dataset we carefully hand-tuned a parameter *W*, which we used to reduce the weight of their Cost(***T***|***M***), making the splits "cheaper," encouraging the production of *k* regimes. This is the only change we made to the Autoplait code. With this change, most, but not all, datasets produced a segmentation. We found that we could perfectly replicate the results in the original Autoplait paper, on the authors own chosen benchmark datasets. However, because these datasets are not very challenging, we confine these results to our supporting webpage [4].

We also compared it to the HOG_{1D} algorithm [32], which has similar goals/motivations to FLOSS, but is batch only.

G. *Case Study: Hemodynamics*

In this case study, we revisit our running example in more depth. Recall that in Section III-A, we suggested that in some domains it may be possible to use training data to learn a value for the CAC score that indicates a change of regime, and we expect that value to generalize to unseen data from the same domain. To test this notion, we consider the Physiologic Response to Changes in Posture (PRCP) dataset [12].

The PRCP dataset consists of continuously monitored Arterial Blood Pressure (ABP) of ten healthy volunteers (five of each sex). During sessions lasting approximately one hour each, the subject's posture was changed in two ways; by rotating the medical tilt table they rested on, or by asking the subject to arise from the reclined table under their own power. Each of these posture-change events (just 'events' in the below) was separated by five minutes in the resting supine position. Because the timing of these events was carefully recorded, this dataset offers an objective ground truth for regime change. As Fig. 13 shows, this data is quite complex.

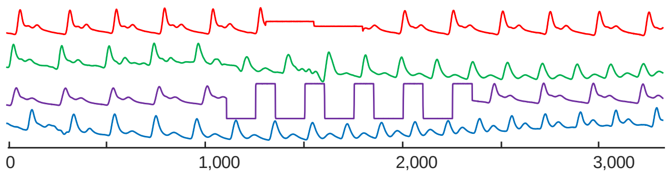

Fig. 13: Four random examples of "no-regime-change." Even these regions of "no change" include sensor artifacts, wandering baseline, noise, and short disconnection artifacts.

To avoid cherry picking, we chose the first subject (in the original archive), a male, to use as the training data. Likewise, to avoid parameter tuning, we Googled "normal resting bpm." The first result, from the Mayo Clinic, suggested "60 to 100bpm," so we set the subsequence length to 187.5, which at 250hz corresponds to the mean of these values.

As we are attempting to learn from only negative examples, we selected 20 regions, each one-minute long (possibly with overlaps) from the regions that do not include any event. For our testing data, we selected 140 negative and 60 positive (regions that straddle an event) from the remaining 9 traces.

We ran FLUSS on the twenty training objects and recorded the minimum CAC value encountered. As shown in Fig. 14 (left), the mean value was 0.671 with a standard deviation 0.194. Using the classic statistical-process-control heuristic, we set the threshold for the testing phase to the mean minus three standard deviations, or 0.089. As we can see in Fig. 14 (right), this gives us an accuracy of 93.5%, with one false negative and twelve false positives.

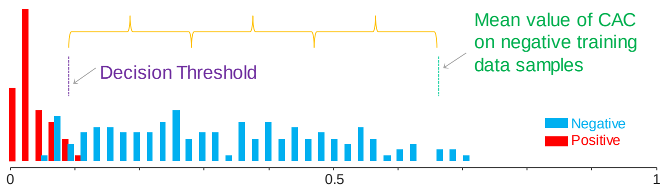

Fig. 14: The positive and negative holdout data can be classified by a simple decision threshold; the mean of the training negative examples minus three standard deviations.

Note that we cannot guarantee here that the false positives are really "false." Independent of the externally imposed interventions, the subject may have induced a regime change by just thinking about a stressful situation [18]. Further note that we could have improved these results significantly with a little work. For example, we could have tuned *L*, the only parameter, we could have built a separate model for females, or for overweight individuals, we could have removed noise or wandering baseline (a common practice for such data) etc. Nevertheless, this experiment bodes well for our claim that we can learn a domain dependent threshold for flagging regime changes, and then it will generalize to unseen data.

H. *User Study: Super-human Performance*

As noted above, the evaluation of semantic segmentation algorithms has often been anecdotal and visual [17]. In essence, many researchers overlay the results of the segmentation on the original data, and we invite the reader to confirm that it matches human intuition [1][5][17][19]. While we are not discounting the utility of such sanity checks (see Fig. 15), by definition, such demonstrations can only offer evidence that the system is par-human [2]. It is natural to wonder if semantic segmentation can achieve super-human performance. To test this, we performed a small user-study. We asked graduate students in a data mining class to participate. Participation was voluntary and anonymous; however, to ensure that the participants were motivated to give their best effort, a cash prize was given for the best performance.

The study was conducted as follows. The participants were briefed on the purpose and meaning of semantic segmentation and where shown some simple annotated examples (This briefing is archived in [4]). Then they were given access to an interface that showed twelve random examples[1], in a serial fashion from the archive discussed in Section IV-E. The interface allowed the participants to explore the data at their leisure, then click on the screen to denote their best guess as to the location of the regime change.

Because our scoring function is fine-grained, we only count a method as *wining* if its score is less than half the score of its rival. Otherwise, we report a *tie*. TABLE IV summarizes the outcomes.

TABLE IV: The Performance of Fluss vs. Humans

	FLUSS	**Best Human**	**Ave Human**
Mean Score	0.013	0.011	0.120
win | lose | draw over FLUSS	NA	2 | 4 | 6	0.81 | 9.5 | 2.0

While the scale of this experiment was modest, these results suggest that we are at, or are approaching, super-human performance for semantic segmentation of time series.

I. *Comparisions to Rival Methods*

Despite our best efforts, we could not get the original Autoplait algorithm to produce any segmentation on 20 of our 32 test datasets. We counted this as a "loss" for "Autoplait_Classic." By carefully adapting the algorithm (see Section IV-F), we could get Autoplait to produce a segmentation on thirteen additional datasets ("Autoplait_Adapted"). On the datasets it did predict segmentations for, sometimes it predicted too many or too few segments. In those cases, we allowed both versions to "cheat." If it predicted too few segments, we took only the closest matches, and gave it all the missing matches with no penalty. If it predicted too many segments, we only considered the best interpretation of a subset of its results without penalizing the spurious segments. In contrast, HOG_{1D} only refused to produce a segmentation on 2 of our 32 datasets. For the rest, it was able to produce the required *k* splits.

Recall that we tested twenty-two humans on a subset of the data. We invited the best scoring individual to segment all of the data. Finally, for calibration, similarly to the default-rate of classification, we considered a random algorithm, which was allowed 100 attempts at guessing the segmentation and reports the average of all attempts. As we noted above, we set the only parameter, the subsequence length *L*, to be about one period length before conducting any experiments. The performance of FLUSS is shown in TABLE V.

TABLE V: The Performance of four rivals compared to FLUSS

	$\text{Autoplait}_{\text{Classic}}$	**$\text{Autoplait}_{\text{Adapted}}$**	**HOG_{1D}**	**Best Human**	**Random**
win | lose | draw over FLUSS	3 | 26 | 3	3 | 25 | 4	8 | 15 | 9	11 | 9 | 12	0 | 32 | 0

A post-mortem analysis showed that if we had instead chosen between ¼ to ½ a period length, we would have cut the number of wins by all rivals by more than half. Nevertheless, these results strongly support our claim of the superiority of FLUSS.

J. *Robustness of FLUSS to the only Parameter Choice*

The performance of FLUSS is highly robust to the choice of its only parameter, the subsequence length *L*. To demonstrate this, we consider two random datasets, *TiltABP* and *DutchFactory*, changing the subsequence length to span an order of magnitude, from 100 to 400 in TiltABP and from 25 to 250 in DutchFactory. As shown in Fig. 15, this variation of subsequence length has insignificant effect on the segmentation.

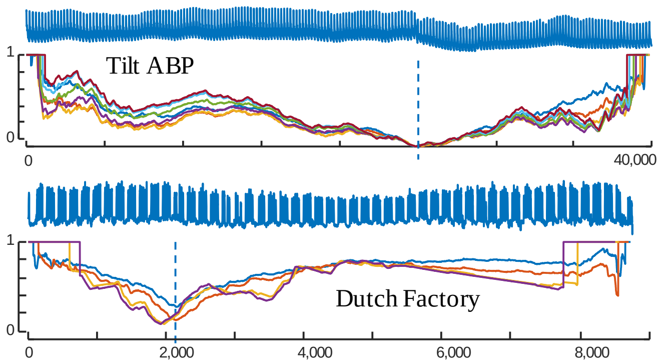

Fig. 15: The CAC computed for (*top*) TiltABP with *L* = {100, 150, 200, 250, 300, 350, 400} and (*bottom*) DutchFactory for *L* = {25, 50, 200, 250}. Even for this vast range of values for *L*, the output of FLUSS is essentially unchanged.

K. *The Speed and Utility of FLOSS*

Our evaluation of FLOSS is brief, since it essentially inherits all the qualities of FLUSS but allows for online segmentation. However, to demonstrate the speed and utility of FLOSS, we performed the following experiment. We

[1] We only considered a subset of twelve from the full dataset to be respectful of the participant's time and attention.

considered the Y-axis shoe acceleration from Subject 3 from the PAMAP dataset's [26] outdoor activity. The data is 270,000 data points sampled at 100Hz, giving us 45 minutes of wall-clock time. We used FLOSS to maintain a sliding window of the last 20-seconds of the subject's behavior, using a 65 subsequence length (suggested by [26]). We discovered:

- It took us only 73.7 seconds to process the data; thus, we can process the data about 36 times faster than in real time.
- In a post-hoc sanity check, we examined the three lowest values of the CAC in this trace. By comparing the locations to the ground truth provided, we discovered that the Top-3 regimes changes correspond exactly to the following transitions: Normal-Walking|Transient-Activities, Nordic-Walking|Transient-Activities and Running|Transient-Activities.

IV. Summary and Future Work

We have introduced a fast, domain-independent, online segmentation algorithm and have shown its utility and its versatility by applying it to dozens of diverse datasets. We further demonstrated that our algorithm is insensitive to the value of the only input parameter. Moreover, we have made all code and data freely available to the community to confirm, extend, and exploit our work. Finally, we are interested in applications of our ideas, for example, to learning from weakly labeled-data [10].

Acknowledgements:

We gratefully acknowledge funding from NSF 1510741 and MERL labs.

References

[1] Aminikhanghahi, S. and Cook, D.J., 2016. A survey of methods for time series change point detection. *Knowledge and Information Systems*, pp.1-29.

[2] Anon. Progress_in_artificial_intelligence. Retrieved January 19, 2017 from en.wikipedia.org/wiki/Progress_in_artificial_intelligence.

[3] Aoki, T., Lin, J., Kulić, D. and Venture, G., 2016,. Segmentation of human upper body movement using multiple IMU sensors. In Engineering in Medicine and Biology Society pp. 3163-66.

[4] Authors. (2017). Supporting website for this paper https://sites.google.com/site/onlinesemanticsegmentation/

[5] Bouchard, D. and Badler, N., 2007, September. Semantic segmentation of motion capture using laban movement analysis. In *International Workshop on Intelligent Virtual Agents* (pp. 37-44). Springer Berlin.

[6] Bregler, C., 1997. Learning and Recognizing Human Dynamics in Video Sequences. In *Proc. Int. Conference on Computer Vision and Pattern Recognition, 1997*.

[7] Cassisi, C., et. al. 2016. Probabilistic Reasoning Over Seismic Time Series: Volcano Monitoring by Hidden Markov Models at Mt. Etna. *Pure and Applied Geophysics*.

[8] Chen, Y., et al. 2015. The UCR Time Series Classification Archive. URL www.cs.ucr.edu/~eamonn/time_series_data/

[9] Chuttani, K., et al., 1994. Diagnosis of cardiac tamponade after cardiac surgery: relative value of clinical, echocardiographic, and hemodynamic signs. *American Heart Journal*, *127*(4), pp. 913-918.

[10] Hao, Y., Chen, Y., Zakaria, J., Hu, B., Rakthanmanon, T. and Keogh, E., 2013, August. Towards never-ending learning from time series streams. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 874-882). ACM.

[11] Harguess, J. and Aggarwal, J.K., 2009, November. Semantic labeling of track events using time series segmentation and shape analysis. In *Image Processing (ICIP), 2009 16th IEEE International Conference on* (pp. 4317-4320).

[12] Heldt, T., Oefinger, M.B., Hoshiyama, M. and Mark, R.G., 2003, September. Circulatory response to passive and active changes in posture. In IEEE *Computers in Cardiology, 2003* (pp. 263-266).

[13] Keogh, E., Chu, S., Hart, D. and Pazzani, M., 2004. Segmenting time series: A survey and novel approach. *Data mining in time series databases*, *57*, pp.1-22.

[14] Keogh, E., Kasetty, Sh., 2003. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration, *Data Mining and knowledge discovery*, 7(4), pp.349-371.

[15] Lainscsek, C., et. al., 2013. Non-linear dynamical analysis of EEG distinguishes patients with Parkinson's disease from healthy individuals. *Frontiers in neurology*, 4, 200.

[16] Lan, R. and Sun, H., 2015. Automated human motion segmentation via motion regularities. *The Visual Computer*, *31*(1), pp.35-53.

[17] Lin, J.F.S., Karg, M. and Kulić, D., 2016. Movement Primitive Segmentation for Human Motion Modeling: A Framework for Analysis. *IEEE Transactions on Human-Machine Systems, 46(3)*, pp.325-339.

[18] Maschke, G.W. and Scalabrini, G.J., 2005. (URL, Retrieved 12-12-16) The lie behind the lie detector. *Antipolygraph. org*.

[19] Matsubara, Y., Sakurai, Y. and Faloutsos, C., 2014, June. Autoplait: Automatic mining of co-evolving time sequences. In *Proceedings of the 2014 ACM SIGMOD* (pp. 193-204).

[20] Matsubara, Y., Sakurai, Y., Ueda, N. and Yoshikawa, M., 2014, December. Fast and exact monitoring of co-evolving data streams. In *2014 IEEE International Conference on Data Mining* (pp. 390-399).

[21] Matteson, D.S. and James, N.A., 2014. A nonparametric approach for multiple change point analysis of multivariate data. *Journal of the American Statistical Association*, *109*(505), pp.334-345.

[22] Molina, J.M., Garcia, J., Garcia, A.B., Melo, R. and Correia, L., 2009,. Segmentation and classification of time-series: Real case studies. In *International Conference on Intelligent Data Engineering and Automated Learning* (pp. 743-750).

[23] Mueen, A., Viswanathan, K., Gupta, C. K., and Keogh, E., 2015. The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance, URL: www.cs.unm.edu/~mueen/FastestSimilaritySearch.html

[24] Nishino, J., Itoh, M., Ishinomori, T., Kubota, N. and Uemichi, Y., 2003. Development of a catalytic cracking process for converting waste plastics to petrochemicals. *Journal of Material Cycles and Waste Management*, *5*(2), pp.89-93.

[25] Reinhardt, A., Christin, D. and Kanhere, S.S., 2013, November. Predicting the power consumption of electric appliances through time series pattern matching. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings* (pp. 1-2). ACM.

[26] Reiss, A. and Stricker, D., 2012. Introducing a new benchmarked dataset for activity monitoring. *In 16th International Symposium on Wearable Computers (ISWC)*, 2012, pages 108– 109. IEEE, 2012.

[27] Pavlovic, V., Rehg, J.M. and MacCormick, J., 2000, March. Learning switching linear models of human motion. In *NIPS* (Vol. 2, p. 4).

[28] Serra, J., Müller, M., Grosche, P. and Arcos, J.L., 2014. Unsupervised music structure annotation by time series structure features and segment similarity. *IEEE Transactions on Multimedia*, *16(5)*, pp.1229-1240.

[29] Wang, P., Wang, H. and Wang, W., 2011,. Finding semantics in time series. In *Proc' of 2011 ACM SIGMOD International Conference on Management of Data*, pp. 385-96.

[30] Weiner, I.D. and Wingo, C.S., 1997. Hypokalemia--consequences, causes, and correction. *Journal of the American Society of Nephrology*, *8*(7), pp.1179-1188.

[31] Yeh, C.-C. M., et. al., 2016. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. *IEEE ICDM 2016*.

[32] Zhao, J., Itti, L., 2016. Decomposing Time Series with application to Temporal Segmentation, *In Proceedings of the IEEE Winter Conference on Applications of Computer Vision* (WACV), Lake Placid, NY, pp.1-9.