Matrix Profile IV: Using Weakly Labeled Time Series to Predict Outcomes

Chin-Chia Michael Yeh
UC Riverside
myeh003@ucr.edu

Nickolas Kavantzas
Oracle Corporation
nickolas.kavantzas@oracle.com

Eamonn Keogh
UC Riverside
eamonn@cs.ucr.edu

ABSTRACT

In academic settings over the last decade, there has been significant progress in time series classification. However, much of this work makes assumptions that are simply unrealistic for deployed industrial applications. Examples of these unrealistic assumptions include the following: assuming that data subsequences have a single fixed-length, are precisely extracted from the data, and are correctly labeled according to their membership in a set of equal-size classes. In real-world industrial settings, these patterns can be of different lengths, the class annotations may only belong to a general region of the data, may contain errors, and finally, the class distribution is typically highly skewed. Can we learn from such weakly labeled data? In this work, we introduce SDTS, a scalable algorithm that can learn in such challenging settings. We demonstrate the utility of our ideas by learning from diverse datasets with millions of datapoints. As we shall demonstrate, our domain-agnostic parameter-free algorithm can be competitive with domain-specific algorithms used in neuroscience and entomology, even when those algorithms have been tuned by domain experts to incorporate domain knowledge.

1. INTRODUCTION

Much of the considerable progress in time series classification in recent years has ignored many of the pragmatic issues facing practitioners. To make progress, the community has typically manually contrived data to fit into the "flat file" format used in the machine learning community (i.e. ARFF format) [32]. The ready availability of such resources, including the UCR Time Series Archive [1] and the more general UCI Archive [13], has been a boon to researchers; however, it has isolated the academic community from the intricacies of time series classification as it presents itself in many industrial settings. To help the reader appreciate how the task-at-hand typically manifests itself in many industrial and medical settings, consider the two-dimensional time series shown in Figure 1. We will define this "learning from weakly labeled data" problem more formally in Section 3.

Proceedings of the VLDB Endowment, Vol. 10, No. 12

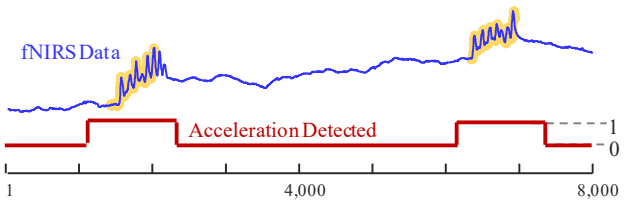

Figure 1: A two-dimensional time series. (*top*) A real-valued fNIRS time series from a patient. (*bottom*) A Boolean time series representing the detection of movement by the patient.

One dimension is a real-valued, functional Near-Infrared Spectroscopy (fNIRS) time series, and the other is a Boolean time series, which can be viewed as an "annotation" to the former. In a more general context, a '1' in this time series may represent a rare desirable or undesirable state. Here, it represents an undesirable patient movement that introduces artifacts into the recordings [26].

The weakly labeled time series learning task-at-hand reduces to the following:

> Suppose that we are given such training data ahead of time, but in the future, the Boolean time series will become unavailable (perhaps for some technical or privacy issue). Can we reconstruct the Boolean time series given just the real-valued signal?

In some domains, this task can be trivial. For example, suppose the real-valued time series is `Patient Temperature` (`PT`), and the Boolean time series is `HasFever` (`HF`). Then a simple threshold rule, If `PT` > 100.4° F then `HF` ←`TRUE`, would work, and we could robustly learn this rule even from a small dataset.

However, note that no such threshold-based rule would work for the example in Figure 1, where the height of the real-valued time series is unrelated to the Boolean value. Nevertheless, this toy problem does seem solvable based on alterative features. For example, the local variance of the time series seems to be higher at the relevant locations. However, in many datasets the variance, and/or other *statistical* features are also a poor indicator of the Boolean variable. In this work, we proposed to use *shape* features. As the zoom-in of the relevant sections demonstrates, shown in Figure 2, the local shape features may offer clues to the Boolean class labels.

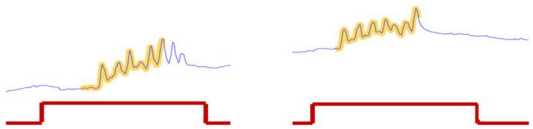

Figure 2: A zoom-in of where Figure 1 indicated the positive class for the Boolean time series (red). The approximately repeated shapes in the fNIRS time series (highlighted in yellow) are suggestive of a mechanism to solve the task-at-hand.

Similar problems where *both* time series are Boolean (or categorical) have been addressed in the literature [5]; however, the real-valued/Boolean task-at-hand here is significantly more difficult for the following reasons [25]:

- **Noisy Labels**: The Boolean annotation may be noisy. That is to say, it may have some false positives and/or false negatives. In our running example, an electrical spike in the recording device may give use an `Acceleration-Detected = TRUE` even if there was no actual movement by the patient.
- **Label Slop**: As hinted at in Figure 2, the Boolean labels may only be approximately aligned with the real-value patterns. This problem is common in manufacturing. It may be that the Boolean time series is some measure of quality (acceptable/unacceptable) that can only be measured after some time lag, for example by a once-a-shift stoichiometry test [9]. Therefore, a failed test can only be loosely associated with the entire last eight-hour period.
- **Class Skew**: In our running example, the `Acceleration-Detected` variable was `TRUE` about a quarter of the time. However, more generally, the minority Boolean class may be vanishingly rare. Again, this is typically true by definition. In medicine and industry, we often want to learn to detect events that we hope are vanishingly rare, such as epileptic fits or catastrophic overpressurization [12]. Thus, we expect that for a huge fraction of the time, a classifier will report "`class unknown`."
- **Scale**: We would like to (indeed, because of class skew and the rarity of targeted events, *need to*), be able to learn from very large datasets, with at least tens of millions of data points.
- **Multi-Scale Polymorphic Patterns**: Most research assumes that time series patterns are of fixed length [1]. However, there is no reason to expect this to be true in real world applications. For example, suppose the Boolean label `TRUE` denotes an unacceptable yield quality in a chemical process. This might have been caused by a `flow-rate` that is increasing too quickly, is oscillating as it increases, or is increasing in discrete steps due to a sticky valve etc. [22]. Not only do these single class root-cause patterns look different (they are *polymorphic*), they can be of very different lengths.

As the reader, will now appreciate, the data in the UCR and UCI archives are a poor proxy for learning from weakly labeled data on all the points above. While existing research on time classification tells us much about appropriate distance measures [1], the importance of data normalization etc., to the best of our knowledge, there is currently no system that tackle the challenges above.

We note that beyond high classification accuracy, our solution to this problem also has a very desirable side-effect. The classification dictionaries we learn can (at least in principle) sometimes tell us something unexpected about the data/domain. For example, that the Asian citrus psyllid insect has two modes of eating (Section 5.2) and humans react more strongly to images of *faces* than to images of *houses* (Section 5.3). We suspect this secondary use of our algorithm may actually be more important in many domains.

The rest of this paper is organized as follows. In Section 2 we discuss related work. Section 3 introduces the necessary definitions and notations. We introduce our algorithm, SDTS (Scalable Dictionary learning for Time Series) in Section 4 and provide a rigorous empirical evaluation in Section 5. Finally, in Section 6, we discuss limitations of our work, and offer directions for future work.

2. BACKGROUND AND RELATED WORK

The general literature on time series classification is vast; we refer the reader to [29] and the references therein. In the last decade, the majority of such research efforts consider *only* data from the UCR archive [1]. While this diverse set of datasets has been a useful resource to compare distance measures [29] and classification algorithms [1], it tends to mask the practical issues of real-world deployments. The format of the UCR Archive is the antitheses of our assumptions, which are enumerated in the last section. In all eighty-five datasets, the ground-truth labels are all correct, there is no label slop, the classes are highly balanced, and the sizes are relatively small (i.e., the training sets have an average of just 454 exemplars). It is unclear if the datasets are polymorphic[1], but each dataset only has patterns of a single fixed length.

The most limiting assumption of the literature is that the universe consists of K well defined classes, and everything belongs to one such class. However, as our assumptions presage, we assume the universe consists of K-1 well defined classes, but there is an `other` class that is ill-defined and unstructured, and moreover, the vast majority of objects are belong to the class `other`. As a result of these mismatched assumptions, to the best of our knowledge, there is no technique in the literature [1][29] we can apply to this problem.

There are a handful of research efforts that have noted the label slop problem in a slightly different context. The first work that specifically addresses the problem is [23]. They have cast the problem to the multi-instance learning framework by treating consecutive data points with uniform labels as *bags*. *Instances* are generated by first applying a sliding window within each bag, then conventional time series features are extracted within each sliding window. They use a multi-instance support vector machine to learn the correspondence between instances and labels. Recently, Guan et al. [7] has proposed a multi-instance learning graphical model based on Auto-Regressive Hidden Markov Model (ARHMM), which addresses the same problem. They improve upon [23] by explicitly modeling the temporal dynamics of time series using ARHMM.

[1] One of the current authors created or edited about one third of the datasets in the archive, and thus has some insights into this question. There are a handful of datasets that *are* polymorphic. For example, for `Gun-Point`, both classes are performed by two actors of very different heights and holstering styles. However, we believe that at least 90% of the datasets are *not* polymorphic.

There is a large body of work on *prognostics* and *precursor* search [10], some of which have goals that are similar to ours (see also Section 5.5). However, virtually all such work is highly domain specific. For example, [10] only considers a particular type of aviation evasive maneuvers, and [3] only investigates a single type of earthquake. Likewise, there is a vast body of work devoted *just* to the case when the time series comes from rotating machinery. The ability to inform/constrain an algorithm with first-principle models from aerodynamics, geology, or dynamics is clearly useful. However, it is contrary to our desire to have a parameter-free, domain-agnostic exploratory tool, that can work "out-of-the-box."

The core subroutine of our algorithm is *subsequence similarity search* [20], which we need to perform perhaps millions of times, in a (main memory) dataset that may also be millions of data points in length. This single fact may explain why we are the first to develop our rather straightforward algorithm. Until recently, the state-of-the-art for the similarity search task was the classic sliding-window similarity search, which must extract every subsequence, z-normalize it, then compute the distance [20]. While this can be accelerated in several ways (omitting the square root step of Euclidean distance, early abandoning etc. [20]), it is still $O(nm)$, with n the query length and m the dataset length. Note that it generally cannot be accelerated by caching the z-normalize subsequences, as this increases the memory footprint by a factor of n, and n may be in the thousands.

The MASS algorithm recently introduced by Mueen and colleagues has reduced the time needed for subsequence similarity search to $O(nlogn)$ [14]. Moreover, here the big O notation masks an at-least one order of magnitude additional difference. Unlike classic similarity search, the MASS algorithm has an extremely low constant factor. Moreover, it exploits FFT computation, which is the typically the most optimized algorithm in any software platform and is often accelerated by co-processors or other hardware optimizations. The practical implication of this is difficult to overstate. For example, in Section 5.3 we learn a model in 41 minutes, but this would have taken us at least many hours, perhaps days, if the state-of-the-art that that existed prior to MASS was used instead.

3. DEFINITIONS AND NOTATION

We begin by defining the data type of interest, *time series*:

Definition 1: A *time series* $T \in \mathbb{R}^n$ is a sequence of real-valued numbers $t_i \in \mathbb{R} : T = [t_1, t_2, \ldots, t_n]$ where n is the length of T.

We are not interested in the global properties of a time series, but in the local regions known as *subsequences*:

Definition 2: A *subsequence* $T_{i,m} \in \mathbb{R}^m$ of a T is a continuous subset of the values from T of length m starting from position i. Formally, $T_{i,m} = [t_i, t_{i+1}, \ldots, t_{i+m-1}]$.

The particular local property we seek to capture is repeating shapes, or *time series motifs*. In the general case, the most efficient way to locate time series motif is to compute the recently introduced *matrix profile* [33][34].

Definition 3: A *matrix profile* $P \in \mathbb{R}^{n-m+1}$ of a time series T is a meta time series that stores the z-normalized Euclidean distance between each subsequence and its nearest neighbor (within the all subsequence set of T), where n is the length of T and m is the given subsequence length.

The time complexity to compute P is $O(n^2)$ [34]. This maybe seems unscalable, but the following facts mitigate this. First, note that the time complexity is independent of m, the length of the subsequences. It is the dependence on m (the classic curse of dimensionality) that is the main limiting factor for other time series data mining algorithms [29].

Secondly, the matrix profile can be computed with an *anytime algorithm*, and in most domains, in just $O(nc)$ steps the algorithm converges to what would be the final solution (where c is a small constant) [33]. Finally, the matrix profile can be computed with GPUs, cloud computing, and other high-performance computing environments that make scaling to at least tens of millions of data points trivial.

Figure 3 shows the matrix profile of T_1. While the motif pair (red) is visually similar to the background random walk (black), the matrix profile still clearly reveals the locations of the motif pair.

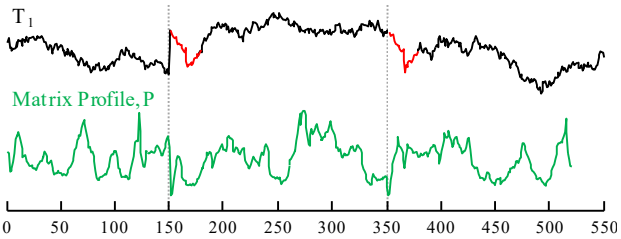

Figure 3: Matrix profile of T_1. The two lowest points on P correspond to the locations of embedded motif pair.

We are interested in the case which the real-valued time series T is accompanied by a Boolean time series.

Definition 4: Given a time series T, a Boolean *time series* $B \in \{0,1\}$ which annotated T is a sequence of binary values $b_i \in \{0,1\} : B = [b_1, b_2, \ldots, b_n]$ where n is the length of B and the length of T.

Note that in some domains, the Boolean time series may be produced natively, for example by a quality control technician annotating the yield quality as `accept/reject` [9], or by an attending physician annotating a patient's record as `tamponade/normal` [18]. However, in other domains it may be the case that the analysist could convert a real-valued time series into a Boolean time series with a simple thresholding rule. In fact, as shown in Figure 4, this was how we produced the annotation for our running example.

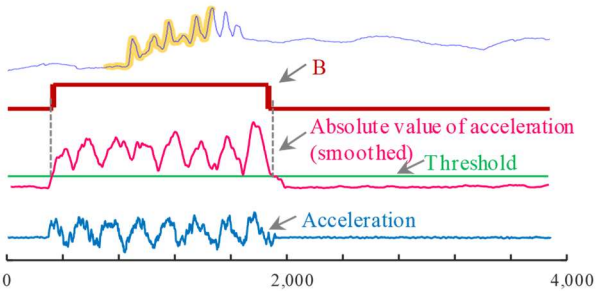

Figure 4: *bottom-to-top*) We took the acceleration from a fNIRS sensor and used it to produce a new time series containing the smoothed absolute value of acceleration. By thresholding this new vector, we produced the Boolean vector B that annotates the raw fNIRS (see Figure 1).

In many industrial domains the conversion may be even easier. For example, for a distillation column that is supposed to be able to produce at least 50 liters of material per second [12][22], we could convert the real-valued flow rate to a Boolean measure of quality by the trivial formula: $B_{\text{low-yield}} =$ `flow-rate` < 49.

Note that for consistency with the literature, we refer to the `TRUE` labels as *positive*, and the `FALSE` labels as *negative*, without any reference to the desirability of the state. For example, `chemical-leak` or `EEG-seizure` may be positive. Here *positive* just means the (typically rare) state we are attempting to predict.

Definition 5: The *weakly labeled time series problem* is the task of generating the binary time series B' of a given real-valued time series T' using knowledge (e.g., rules) acquired from the previously seen binary time series B and real-valued time series T.

Due to the class imbalance (and binary) nature of the problem, we use F_β-score instead of accuracy as the measure of success [19]. We can set β based on the relative importance of precision versus recall in the domain of interest. For example, β can be set to 2 in cases where false alarms can be tolerated, while a failed alarm is more critical.

Finally, we define the simple data structure that will allow us solve the problem-at-hand. We propose to solve the *weakly labeled time series problem* by automatically learning a *dictionary*.

Definition 6: A *dictionary* is a set of shapes $\boldsymbol{S}$ (possibly of different lengths), each with an associated threshold H. When used to monitor a new streaming time series T', B' is set to TRUE iff the current subsequence is within h_i of S_i (h_i is ith member of H and S_i is ith member of $\boldsymbol{S}$), else it remains FALSE.

In the next section, we will show how we can automatically learn such dictionaries from the data.

4. THE SDTS ALGORITHM

With all the definitions and notation specified, we are finally prepared to explain our algorithm. Since the weakly labeled time series problem is a learning/predicting type of task, we first introduce the dictionary learning algorithm in Section 4.1, and subsequently show how to predict with the learnt dictionary in Section 4.2.

4.1 Learning the Dictionary

Having defined the dictionary in the previous section, and motivated the use of the F_β-score to evaluate it, how can we find the best dictionary for a given dataset? Even if we confine the patterns in the dictionary to come from the data itself, and limit the maximum dictionary size, say to just five entries, the number of possible dictionaries exceeds a trillion for a modestly sized dataset. As outlined in Algorithm 1, we propose to use an optimized greedy search to construct the dictionary.

In line 1, each segment that is marked positive in time series T is extracted and concatenated to form another time series T'. This shorter time series T' will allow us to limit the search space for shape features to place in our dictionary. Since the objective of the algorithm is to find a set of shape features used to predict positive segments, all possible shape feature candidates (according to B) are contained in T'. Our reason for concatenating all of the positive time series snippets into a single time series is more than a bookkeeping device; it allows us to extract the maximum speed-up from the STOMP algorithm [34]. Figure 5 shows how the shorter time series T' is produced.

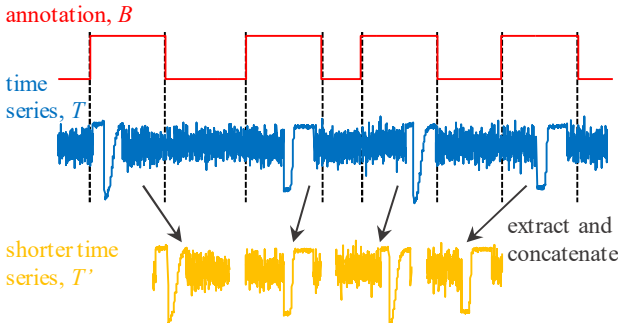

Figure 5: Positive segments are extracted and concatenated to form a shorter time series *T'* for matrix profile computation. We link positive segments together in our algorithm; the space between each segment is added for visual clarity. Recall that 'positive' just means Boolean *TRUE*, not necessary *desirable*.

In line 2, the matrix profile P of T' is computed (recall Definition 3). Because T' is generated by concatenating different segments of T, the discontinuity in time creates subsequences that do not exist in T (similarly to the pseudo word 'clean' formed in the concatenation of *Ora*clean*omaly*). To avoid considering such nonexistent subsequences as a shape candidate, subsequences that cross discontinuity are ignored when computing P, and their corresponding values in P are set to infinity. In line 3, a set of shape candidates $\boldsymbol{C}$ are selected based on their matrix profile values. For each positive segment in T', the subseqence with lowest matrix profile value is extracted and added to $\boldsymbol{C}$, because subsequences with lower matrix profile values are repeated with greater fidelity than others (by definition). Note: if there are a total of b positive segments, the size of $\boldsymbol{C}$ is also b. Figure 6 shows how the member of $\boldsymbol{C}$ is selected using P.

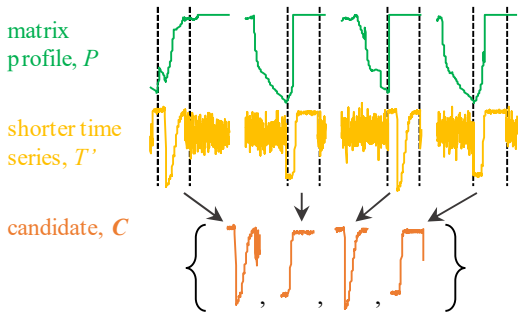

Figure 6: Candidate set *C* is selected from the shorter time series *T'* based on the matrix profile *P*. The subsequences with smaller values in *P* are selected and are added to *C*.

From lines 4 through 6, each shape feature C in $\boldsymbol{C}$ is individually evaluated by finding the threshold that optimizes the F_β-score when used to perform a prediction on T. Both the discovered threshold and corresponding F_β-score are stored in H_C and F respectively. The threshold is found efficiently by using the golden section search algorithm [30]. Although the thresholds found here are refined later in line 8 through 23, when the combination of shape features are considered, an initial set of thresholds is required as the initial condition for the coordinate ascent golden section search [30].

In lines 8 through 23, the final shape features are selected using greedy search. Each shape candidate C in $\boldsymbol{C}$ is tested by performing a prediction on $\boldsymbol{T}$ when used in conjunction with previously selected candidates in $\boldsymbol{S}$. It is important that we evaluate the candidate in the context of previously added patterns; otherwise, the dictionary may fill up with redundant patterns that are only slight variants of each other.

When testing a given candidate C, first we refine the threshold setting for each shape feature by using the golden section search algorithm in a coordinate ascent fashion; as using multiple shape feature may require less strict thresholds. In the inner loop (line s10 through 17), each candidate C is tested independently with the previously selected shape feature, and the best one is stored in $\boldsymbol{S_{bsf}}$. From lines 18 through 22, if $\boldsymbol{S_{bsf}}$ improves the F_β-score, $\boldsymbol{S_{bsf}}$ is added to $\boldsymbol{S}$. Otherwise, the greedy search is terminated. To ensure that the candidates are tested on a sufficient amount of validation data, the number of shape features is limited to half of the number of candidates. Finally, the selected shape feature $\boldsymbol{S}$ and associated threshold H are returned in line 24.

Algorithm 1: Dictionary Learning Algorithm.

Procedure *train*(T, B, m)
Input: time series T, annotation B, and subsequence length m
Output: dictionary (set of shape features $\boldsymbol{S}$ and thresholds H).

```
1  T' ← extractPositiveSegment(T, B)
2  P ← computeMP(T', m)                          // see Definition 3
3  C ← extractShapeCandi(T', P, m)
4  for each C in C
5     H_C, F ← findThresholdEvalF(C, T, B)
6  end for
7  f ← −∞, S ← ∅, H ← ∅
8  for i from 0 to |C|/2
9     f_bsf ← −∞, , S_bsf ← ∅, H_bsf ← ∅
10    for each (C, h) in (C, H_C)
11       S_new ← S ∪ C
12       H_new ← H ∪ h
13       H_new, f_new ← refineThresholdEvalF(S_new, H_new, T, B)
14       if f_new > f_bsf
15          f_bsf ← f_new, S_bsf ← S_new, H_bsf ← H_new
16       end if
17    end for
18    if f_bsf > f
19       f ← f_bsf, S ← S_bsf, H ← H_bsf
20    else
21       break
22    end if
23 end for
24 return S, H
```

To extend SDTS to allow dictionary elements of various lengths, we simply compute multiple matrix profiles using different settings of m in line 2 and combine extracted candidate from each individual matrix profile in line 3. Note that while Euclidean distances of different lengths time series are not commensurate, the F_β-scores derived from different lengths time series pattern *are* commensurate.

Users can simply provide a set of m to SDTS, and SDTS will automatically select shape features with the appropriate lengths. SDTS is not particularly sensitive to the setting of m, as we demonstrate in Figure 7. Given this, users can simply provide a coarse grid around the natural scale of the time series event. For example, if the user vaguely suspects that one hour is about the natural scale of the (sampled once a minute) data, the user can pass in a set of values for m such as [55, 60, 65] to bracket their intuition. The results of this search will almost certainly be as good as a search over increments of one second or finer.

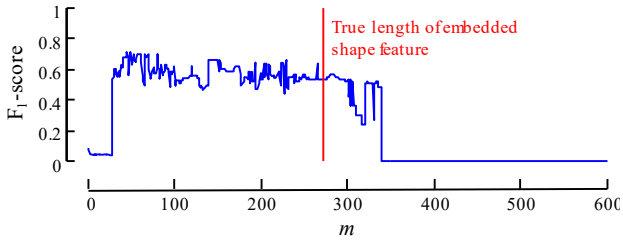

Figure 7: The performance of SDTS is relevantly insensitive to the settings of m. For an embedded pattern of length 275 (see section 5.1), the F1-score is about 0.6 for the large range of m greater than 50 and less than 300.

Beyond speed-up, there is an additional reason why the coarser search may be more desirable. We hope that our discovered rules will be examined (and perhaps edited) by the domain experts. Such experts are likely to feel more comfortable dealing with rules such as "*If you see this one hour-long valley in the temp reading…*", than the spuriously precise "*If you see this fifty-nine minute, thirty-seven second-long valley…*" [11].

4.2 Using the Learned Dictionary

Having learned the dictionary, applying it is straightforward; however, in Algorithm 2, we outline the details of its application for completeness.

In line 1, the predicted annotation B' is initialized as a zero vector of the same size as the input time series T'. From line 2 to line 9, we test each shape feature in the dictionary on T'. First, we compute the z-normalized Euclidean distance between a shape feature and each subsequences of the same length by using the MASS algorithm [14]. Next, from line 4 to line 8, we check each value in the distance vector D, and flag the subsequence as positive if its value is below the associated threshold h. Lastly, the predicted annotation B' is returned in line 10. The time complexity of the prediction algorithm is $O(|\boldsymbol{S}|\, n' \log n')$ as we perform MASS algorithm $|\boldsymbol{S}|$ times, and each MASS call takes $O(n' \log n')$, where n' is the length of T'.

Algorithm 2: Prediction Algorithm.

Procedure *predict*(T', $\boldsymbol{S}$, H)
Input: time series T' and dictionary (set of shape features $\boldsymbol{S}$ and thresholds H).
Output: B' predicted annotation

```
1  B' ← vector of zeros
2  for each (S, h) in (S, H)
3     D ← MASS(S, T')                            // see [14]
4     for i from 0 to length(D)-1
5        if D[i] < h
6           B'[i] ← 1
7        end if
8     end for
9  end for
10 return B'
```

The extension of the prediction algorithm to streaming time series monitoring is trivial. In line 3, instead of computing the z-normalized Euclidean distance between a shape feature to all subsequence in T', we simply compute the z-normalized Euclidean distance between the shape feature and the newly observed subsequence, and check the newly computed distance with the

associated threshold. Naïvely, this operation takes $O(m)$ each time the algorithm ingests a new point (where m is the length of shape feature). However, since the goal is to determine whether the resulting distance is below the threshold, techniques such as early abandoning and lower bounding [20] can be applied to speed up the computation. To concretely ground the computational demands, even if the dictionary contained one hundred shape features, each of length 1,000, it would be trivial to process a stream arriving at 500Hz, using off- the-shelf hardware.

5. EXPERIMENTAL EVALUATION

We begin by stating our experimental philosophy. We have designed all experiments in a manner such that they are easily reproducible. To this end, we have built a Web page [24] that contains all datasets and code used in this work, together with spreadsheets which contain the raw numbers.

Throughout the experiment section, we report the F_1-score as the single number measurement of success. We also report the wall clock time required, running on a desktop computer with Intel Core i7-6700K 4 GHz Quad-Core Processor. It is difficult to overstate the utility of the MASS algorithm in accelerating our learning algorithm. Where appropriate below, we will report the time taken if we eschew MASS, and resort to the second fastest known algorithm for Euclidean search [20]. Such times are necessarily estimated.

We begin our experiments with a synthetic dataset. Such tests are less compelling than the four diverse real-world case studies that follow it. However, the synthetic dataset allows us to "stress test" our algorithm, by varying the factors that make the task-at-hand challenging.

5.1 Stress Testing on a Synthetic Dataset

The TRACE dataset [21] is a synthetic dataset designed to model industrial processes that are *"..characterized by long periods of steady-state operation, intercalated by occasional shorter periods of a more dynamic nature in correspondence of either normal events, such as minor disturbances, planned interruptions or transitions to different operation states, or abnormal events, such as major disturbances, actuator failures, instrumentation failures, etc*.". These are all data characteristics that have been echoed back to Oracle by its IOT customers in the manufacturing and the oil-and-gas industries [17]. Testing on such synthetic data offers us the possibility of studying how the properties of the data and the domain affect our ability to learn.

We begin by performing a single experiment on a particular instantiation of the problem space; then, having calibrated our expectations, we vary each factor of the problem space one-by-one, while holding everything else constant to see how much that factor matters. The factors in question are:

- **Occurrence of positive events**: Varying this factor is similar to varying skewness between classes (and number of training example) in traditional binary classification.
- **Fraction of false positive labels**: Some segments without repeated shape features are marked as positive. Varying this factor allows us to examine the algorithm's robustness against noisy labels.
- **Fraction of false negative labels**: Some sections of the raw time series with conserved shape features are marked as negative. Similarly to the last factor, varying this factor allows us to examine the algorithm's robustness against noisy labels.
- **Amount of label slop**: This factor is unique to weakly labeled time series classification, and is measured by the fraction of each positive segment being irrelevant time series (i.e., time series other than embedded shape feature). Varying this factor allows us to examine the algorithm's ability to work against imprecise labels in time.

The default setup is as follows: 100 occurrences of positive events, 0 false positives, 0 false negative, and 0.7 label slop.

We have summarized the F_1-score, precision, and recall versus various settings in each factor in Figure 8, Figure 9, Figure 10, and Figure 11. Note that in each plot, only a single factor is varied while all the other factors are kept fixed. The synthetic data was generated by embedding TRACE patterns to random walk. Each set of experiments was repeated 16 times (with random walks generated by different seed), and the reported performances averaged 16 trials. Since the random walk for each set of experiments was generated independently, the performance of the default setups in each figure is slightly different, but within each plot, the numbers are commensurate as we vary the factors.

As shown in Figure 8, increasing the number of positive events benefits SDTS, since the number of shape feature candidates is directly proportional to the occurrence of positive events, and SDTS benefits from larger set of candidates to search over.

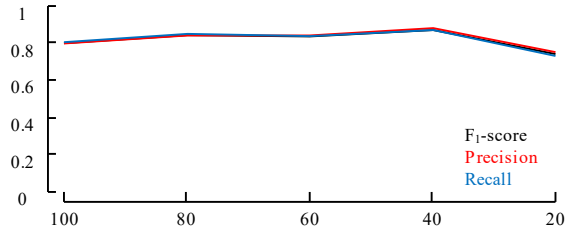

Figure 8: The performance of SDTS versus various settings of positive events occurrence. SDTS's performance suffers slightly when the number of positive events decreases.

Moreover, increasing the number of positive events can mitigate the issues associated with class imbalance. Since the length of training data is fixed, increasing the ratio of positive events reduces the preponderance of negative events; thus nudging the positive to negative ratio is closer to 1.

In contrast, SDTS's F_1-score, precision, and recall all suffer from the increase of false positives (i.e. mislabeled data) in the training data as demonstrated in Figure 9.

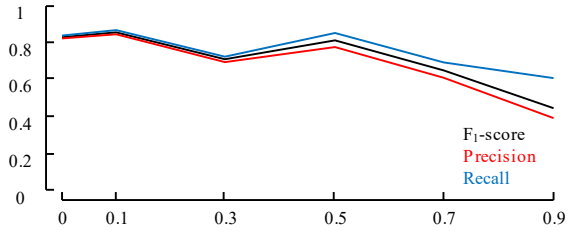

Figure 9: The performance of SDTS versus various settings of false positive fraction. Unsurprisingly, the performance decreases as the false positive fraction increases, but the degradation is slow and graceful.

It is unsurprising that the performance of the system degrades with increasing false positive labels. However, the performance of SDTS offers graceful degradation and does not fall dramatically, even when the fraction of false positive is as high as 0.5.

As shown in Figure 12, SDTS's F_1-score and recall suffer from the increase of false negatives. Yet, the precision is maintained at a relatively high value compared to the other two performance metrics.

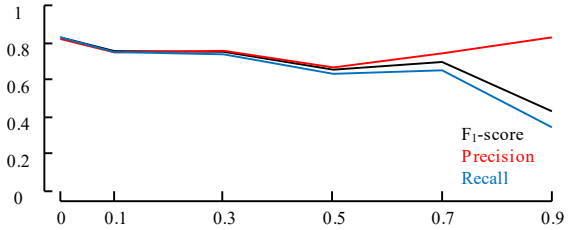

Figure 10: The performance of SDTS versus various settings of false negative fraction. Interestingly, the precision increases when the false negative fraction increases.

One possible explanation is that the false negatives force the dictionary learning algorithm to learn a tighter threshold, because the algorithm is trying to separate a captured (true) shape feature from an embedded shape feature (which is very similar to the captured shape feature) in negative segment. Similar to Figure 9, the F_1 -score of SDTS does not drastically decrease until the fraction of false negative is 0.7.

The experiment shown in Figure 11 suggests that SDTS's performance is only slightly impaired by large amounts of label slop. One possible reason is that SDTS is shift invariant. In other words, as long as the embedded shape feature is within the positive segment, SDTS would find the shape feature even if the ratio between noise and signal (the shape feature) is as large as 0.9.

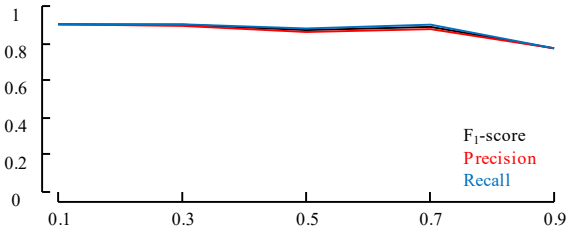

Figure 11: The performance of SDTS versus various settings of label slop amount. SDTS is not sensitive to increasing amounts of label slop.

5.2 Case Study I: Insect EPG

Insects that feed by ingesting plant fluids cause devastating damage to agriculture worldwide, primarily by transmitting pathogens of plants. As a concrete example, the Asian citrus psyllid (*Diaphorina citri*) shown in Figure 12.*top* a is vector of the pathogen causing citrus greening disease, and has already caused billions of dollars of damage to Florida's citrus industry in the last decade and is poised to do this same in California.

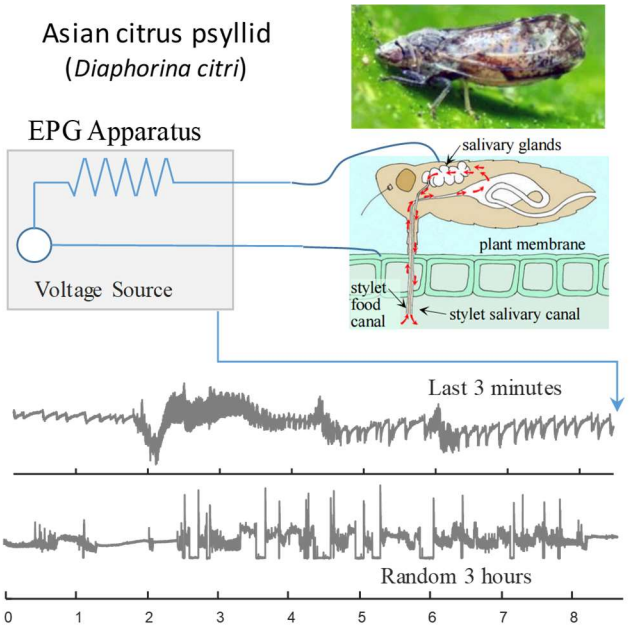

Figure 12: *top-to-bottom*) The Asian citrus psyllid can be connected to an EPG (Electrical Penetration Graph) apparatus, and have its behavior recorded. As the three minute, and three hour snippets show, this behavior is suggestive of structure, but nosily and complex.

As shown in Figure 12, the feeding processes required for successful pathogen transmission by psyllids can be recorded by monitoring voltage changes across an insect-food source feeding circuit. However, as [31] notes "*The output from such monitoring has traditionally been examined manually, a slow and onerous process*." While we do not wish to makes any claims of entomological significance, it is natural to ask if our ideas can be applied to such datasets.

We obtained a dataset recently made publicly available by the United States Department of Agriculture. While this dataset has been labelled by domain experts, as we show in Figure 13, it contains significant label slop, and is thus an ideal dataset to test SDTS robustness to that issue.

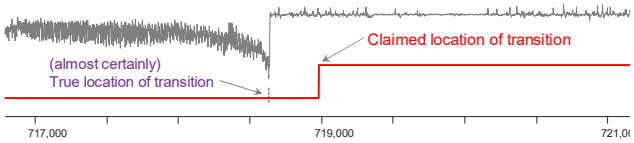

Figure 13: An original annotation of a transition from `stylet passage` to `non-probing` behavior [31]. Although we do not have access to the original data, it is virtually certain that this is an example of label slop (see Section 1).

We learn the model from one EPG recoding section of an insect feeding on *Corrizo* (a rootstock for citrus) and verify the learned model on another EPG recoding of feeding on the same citrus variation. While both experiments consider the same *species*, the Asian citrus psyllid, and the *individual* insects where different, thus we are testing the generalization ability of our algorithm.

To demonstrate our algorithm's ability to capture shape features from multiple classes, we treat both *phloem ingestion* and *xylem ingestion* as the positive class. SDTS is able to achieve a F_1-score of 0.78, a precision of 1.00, and a recall of 0.64. Figure 14 shows

the prediction result with the ground truth label. We can see that SDTS is capable of learning a model that gives no false positives despite some false negative.

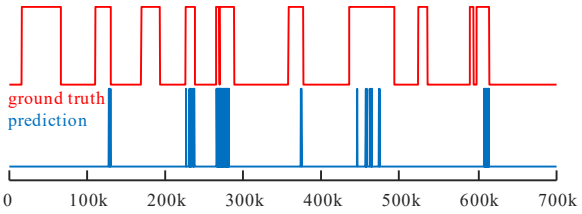

Figure 14: The annotation predicted by SDTS versus the ground truth annotation. The prediction of SDTS is not perfect, but it has no false positives.

Beyond the high accuracy achieved, we wonder if the dictionary learned is itself useful and/intuitive. We showed the model to Dr. Gregory Walker (Figure 15), who has not involved in collecting the data, but who has decades of experience in manually exploring EPG data. He noted "(the first two waveforms) *represent ingestion from two different apoplastic compartments such as xylem versus other extracellular space*." [28], which confirms that the two patterns are indeed polymorphic variants of a single behavior.

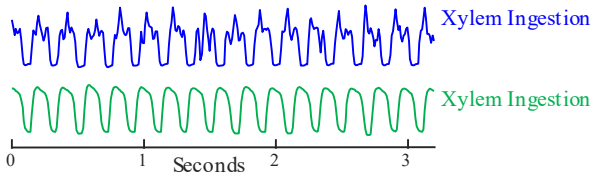

Figure 15: The first two patterns in the learned EPG model show that a single class can be highly polymorphic.

5.3 Case Study II: Neuroscience

The connection between visual perception of objects and neural activity in the visual cortical areas is a fundamental problem in neuroscience [15]. Recent work has shown that that electrical potential from the temporal lobe in humans contains sufficient information for spontaneous and near-instantaneous identification of a subject's perceptual state [15]. However, such efforts require an extraordinary amount of domain knowledge, data preprocessing, and algorithm tuning. Here, we will attempt to duplicate some fraction of the recent achievements, with our *completely* domain agnostic algorithm. To be clear, we are not claiming any medical significance or utility in this section. We are merely showing that, in a real-word, noisy, complex and massive electrocorticographic (ECoG) dataset (see Figure 16) created out of our control, we can robustly learn models that capture true structure in the data and allow (much) better-than-random guessing predictions on unseen data.

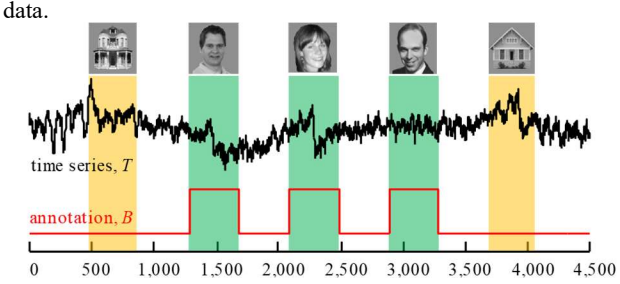

Figure 16: A small snippet of the electrocorticographic data used in our face discrimination experiment. The positive class is when the patient can see a `face`, and the negative class is when the patient is seeing either a `house` or `nothing`.

The ECoG data we consider was collected from an epileptic patient. Electrodes were placed directly on the patient's occipital lobe (the visual processing center for the mammalian brain). Fifty images of faces and fifty images of houses were shown to the patient in a random order, with 0.4-second pauses in-between. While fifty 1,000 Hz traces where recorded from various parts of the brain, for simplicity we consider only a single trace. Our task is to examine the traces to find patterns that indicate that the patient is seeing a `face`.

As noted on [15] "*face-selective* (time series patterns) *may have wide structural variation, with 'peaks' and 'troughs' that are very different in shape, latency, and duration*," making this a challenging task. It particular, we see this uncertainty in *latency* and *duration* as label slop.

We performed our experiment on subject 2. We partitioned the original time series into three sections (each section corresponding to different experiment runs) and performed three-fold cross validation similarly to [15]. To confirm that SDTS performs better than the *default rate* (random guessing in proportion to the prior probability of events), we repeated the experiment on the same data using a permutation test [16]. We generated the permuted labels by randomly shuffling the temporal location of the positive segments. In other words, a positive segment may or may not correspond to `face` in the false label. The experimental results suggest that SDTS is significantly better than random guessing (F_1-score of 0.47 vs. 0.21). This is a huge difference, and it is unsurprising that a two-sample t-test confirms the difference at a 5% level.

The time it takes SDTS to learn a model from the ECoG dataset was 41 minutes. If we replace MASS with the standard Euclidean distance subsequence-search technique, a sliding window that exacts the subsequences, z-normalizes them, and then compares the distance, this time grows to a few days. Interestingly, (as also noted in [20]), we found the time needed to z-normalize the subsequences dominates the time required for this operation.

5.4 Case Study III: Traffic Loop Sensor

To demonstrate that SDTS does not produce an unnecessarily complex dictionary for simple problems, we have applied SDTS on the much-studied Dodgers loop sensor dataset [4][8]. This dataset records the number of vehicles on the 101 North freeway off-ramp near Dodgers Stadium in Los Angeles for 25 weeks. The research community has performed a wide variety of time series data miming experiments on the dataset. The particular experiment we performed was weekend detection. In other words, in the accompany annotation of the training data, all the weekends were marked as positive while the weekdays were marked as negative. While this is a contrived problem, it is not trivial. As Figure 17 suggests, the data is noisy. Moreover, there are dropouts (random occasions when the sensor was offline), and several weekday holidays that might act as pseudo weekends. The data exhibits "bursts" when the Dodgers played a home game, which could be any day of the week. Finally, as the data spans a half year, and we learn from only the first twelve weeks, there is the possibly of concept drift as the seasons change.

Nevertheless, as Figure 17 shows, in general, a typical weekday traffic pattern *does* look different than typical weekend traffic, suggesting a simple model should suffice for accurately distinguishing the weekend from a weekday.

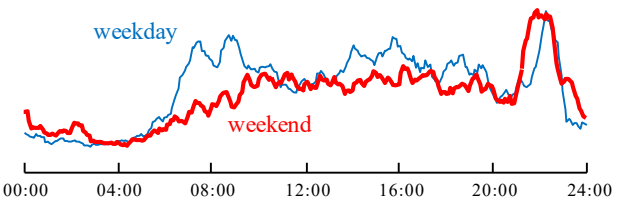

Figure 17: The major differences between weekend (red/bold) and weekday (blue/fine) patterns are the morning and evening rush hour 'bumps'.

To perform such an experiment, we trained the SDTS model on the first half of data. Then, we used the learned model on the second half. The result model is surprisingly simple. The model only contains one shape feature, corresponding to traffic density in the morning (more precisely, midnight to noon) of a Saturday. The captured feature is relevant as it can be used to differentiate a (relatively) quiet weekend morning from a busy weekday morning. Figure 18 shows the captured shape feature and how different it is from the traffic data from a weekday.

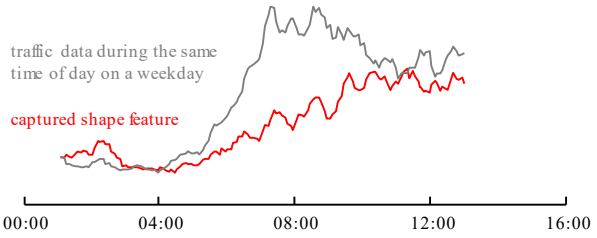

Figure 18: The capture shape feature is corresponding to weekend morning traffic.

Despite the learned model being sample, it can accurately detect weekend from the traffic data. Figure 19 shows how similar the predicted annotation is to the ground truth.

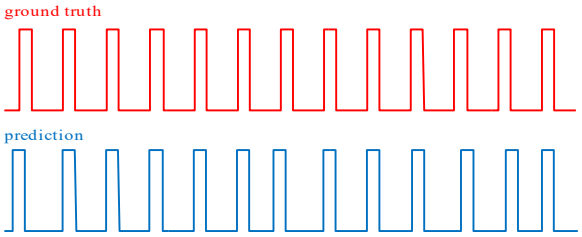

Figure 19: The ground-truth vs. prediction of the SDTS model. Of 13 weekends, the learned model perfectly annotates 9 of them. The other 4 weekends are slightly mislabeled in terms of their temporal locations (Falsely skipping the Saturday or mistakenly labeling Friday as a weekend day).

This is a good place to revisit one of our assumptions. Recall that we are searching for subsequences in the *z-normalized* space (Definition 3). Here, it might be imagined that we should not normalize the data, as the *absolute* values offer clues, with higher traffic volumes on weekdays. If this is really desired, it is trivial to achieve, as the MASS algorithm, and the Matrix Profile that is built upon it, can trivially be converted to an amplitude/offset sensitive algorithm by simply commenting-out some lines of code [14]. However, we claim that this is unlikely to ever be appropriate. Recall that in our motivating example shown in Figure 1, the *shape* was informative, but the change in *offset* (in this domain, "*wandering baseline*") was not. We argue that this is generally true, even in this apparent counterexample. For example, the absolute volume of cars could change due to nearby road maintenance, or even because of changes in the price of fuel; however, the overall shapes will remain near constant. In [20], the authors make a more detailed argument that virtually every task, in almost every dataset requires the normalization of subsequences.

5.5 Predicting the Future: A Tentative Case Study

As the experiments in the previous sections suggest, the ability of SDTS to predict the *current* state of the world can be useful in many domains. However, in many situations it is clearly more desirable and actionable to predict the *future* state of the world. Such shape features are called sometimes called "precursors" or "precursors signatures" (although the literature is inconsistent in its nomenclature [3][10]).

As Figure 20 suggests, it is trivial to generalize SDTS to allow the discovery of precursors.

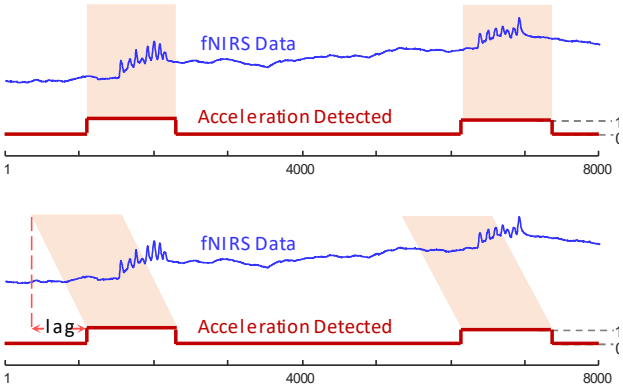

Figure 20: *top*) A visual reminder of the original setup for the weakly labeled classification problem (recall Figure 1). *bottom*) Generalizing the problem to a precursor setting simply requires compensating for the lag between the binary time series *B* and real-valued time series *T*.

All we need do to generalize from our typical consideration of "co-cursors" to precursors, is create a lag between the binary time series B and real-valued time series T (conversely, it may sometimes be more natural to speak of the *lead* time between T and B). As a practical matter, we can achieve this by simply removing the first L data points of B, where L is the length of the desired lag.

We may have some ideas of a reasonable value for L based on the domain. For example, for a small distillation column a lag of five minutes might be ambitious, but for a large distillation column, the inertia of the system may allow a lag of a few hours [9]. As it happens, this discussion of a domain dependent constraints may be moot. We will always want as much lead time as possible, and our proposed algorithm is fast enough to test expanding values of L until the scoring function is unable to find predictive patterns.

To test this idea, we have adapted a real dataset. This contrived experiment is not as interesting as the propriety real-world customer problem that inspired this work, but has the advantage that we can share all the data with the community.

As shown in Figure 21.*right*, the Sony AIBO is a small quadruped robot that comes equipped with a tri-axial accelerometer and a (very) low-resolution camera. This accelerometer measures data at a rate of 125 Hz. In Figure 21.*left*, we show two snippets of telemetry from the accelerometer's z-axis (the direction pointing skyward) as the robot walks on two different surfaces. As the reader will appreciate, the differences in gait due to the surface makes are

non-obvious, even after careful visual inspection, and seem swamped by natural viability and noise.

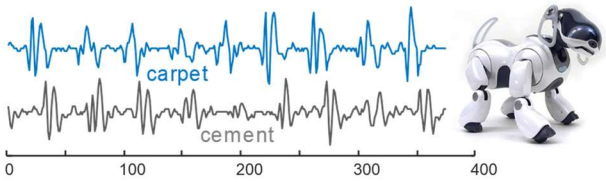

Figure 21: *left*) Two three-second snippets extracted from a Sony AIBO robot dog (*right*). The snippets show about three gait cycles.

The onboard camera and limited processing power do not lend themselves to complex image processing, but we can simply 'snap' a targeted color to the positive class. Finally, if we task our dog to walk backwards across the lab, we will produce a dataset that exactly models the setup in Figure 20.*bottom*, with series T being the accelerometer value, and B being `cement=TRUE` extracted from the video feed. The exact amount of *lag* depends on the angle of the robot's head. Again, while we acknowledge that this toy experiment is highly contrived, it *is* non-trivial, and is an excellent proxy for real-word problems in prognostics for manufacturing and transport.

Our dataset was created by interleaving the z-axis accelerometer time series of Sony AIBO surface recognition dataset [27]. The original dataset consists of accelerometer time series, corresponding to the robot walking on different surfaces (i.e., carpet, field, and cement) [27]. The goal of our experiment is to show that SDTS is capable of discovering *precursors* for an event of interest. Among the three classes provided by the Sony AIBO dataset [27], we picked "walking on cement" as the targeted event of interest.

We begin by splitting each of the time series into disjoint training and test splits. Then, we apply the following three steps independently to the training and test data.

1. `carpet` and `field` are concatenated together to make the problem more challenging.
2. `cement` is sliced into segments of various lengths.
3. The segments of `cement` are embedded into the `carpet-field` time series at multiple randomly selected locations.

Figure 22.*top* illustrate how the time series from various classes are put together. Note that the duration of the positive events, and the amount of interstitial time between them, are random and highly variable.

In order to generate the accompanying annotation for precursor discovery training data, we flag a small chunk (about 2 seconds in time) of the annotation time series as *positive* at the beginning of each `cement` regions. To make the situation conform to our assumptions, the positive segments for each `cement` regions has a lag relative to the actual starting point of `cement`, because our robot experiences a slight change in gait (due to walking on a different material), before visually confirming the change of surface. Figure 22.*bottom* shows an example of such annotation.

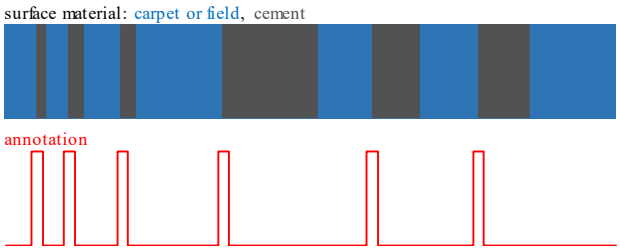

Figure 22: *top*) A representation of the surface walked upon by the robot. *bottom*) The annotation used to train SDTS for precursors of walking on "`cement`" has a slight lag, due to the delay between the robot experiencing the real-value stimulus, and seeing the positive label.

With the annotation for training data prepared in this fashion, we can simply apply the SDTS algorithm without any modification, to discover the precursor(s) for "walking on cement." Figure 23 shows the predicted annotation against the ground truth. The corresponding F_1-score is 0.63. There are a handful of false negatives, but all regions predicted as positive are indeed just prior to the robot seeing `cement`.

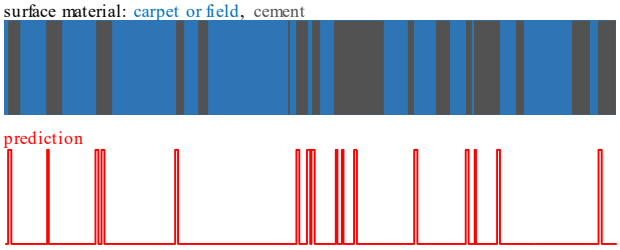

Figure 23: *top*) The surface walked on by the robot the ground truth for our predictions. *bottom*) The predictions made by our precursor model, found using SDTS.

In essence, this experiment shows that in principle, we can use SDTS to gain a little "lead-time" to predict upcoming events.

6. DISCUSSION AND CONCLUSIONS

Of the four case studies we considered, we believe that only *Traffic Loop Sensor* would be solvable by "eye", by the average person. The *Insect EPG* dataset appears to be at least partially solvable by humans, but only fully solvable by expert entomologists with decades of experience examining such data [28]. For both the *Robot Gait* and *Neuroscience* datasets, our algorithm offers truly superhuman performance. Even if we "cheat" by examining various sources of extra information, the differences discovered by our algorithm are too subtle for us to appreciate, much less duplicate or improve upon with human-coded rules.

In conclusion, we have introduced SDTS, a parameter-free domain agnostic algorithm for learning from weakly supervised datasets. We have made all code and data freely available to the community, to confirm, extend, and exploit our work [24].

Future work includes consideration of the multidimensional time series case, and allowing humans to interactively edit the learned models. We are also interested in the "cold-start" problem [6]. Could a model learned on one domain be used on similar domain, at least until enough data has been observed to allow relearning the model? In the industrial domain, this problem can arise if the production run for one object finishes, and a new production run for a similar device begins.

7. ACKNOWLEDGMENTS

Our thanks go out to all the donors of datasets and domain experts who offered advice. This research was funded by gifts from Oracle and NSF awards 1510741 and 1544969.

8. REFERENCES

[1] Bagnall, A., Lines, J., Hills, J., Bostrom, A. Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles. *IEEE Trans. Knowl. Data Eng*. 27(9): 2522-2535. 2015.

[2] Chen et al., The UCR time series classification archive. http://www.cs.ucr.edu/~eamonn/time_series_data/.

[3] Cheong, S. A. Extracting Earthquake Precursor Signatures Through Time Series Clustering, contributed talk, *Western Pacific Geophysics Meeting*, 23 June 2010, Taipei, Taiwan.

[4] Freeway Performance Measurement System (PeMS). http://pems.eecs.berkeley.edu/.

[5] Fung, P. and Church, K. K-vec: A New Approach for Aligning Parallel Texts. In *Proceedings of COLING 94*. pp. 1096–1102. Kyoto, Japan.1994.

[6] Gao, M., Tian, R., Wen, J., Xiong, Q., Ling, B., Yang, L. Item Anomaly Detection Based on Dynamic Partition for Time Series in Recommender Systems. *PLoS ONE* 10(8). 2015.

[7] Guan, X., Raich, R. and Wong, W. K. Efficient Multi-Instance Learning for Activity Recognition from Time Series Data Using an Auto-Regressive Hidden Markov Model. *In Proceedings of the 33rd International Conference on Machine Learning*. pp. 2330-2339. 2016.

[8] Ihler, A., Hutchins, J. and Smyth, P. Adaptive event detection with time-varying Poisson processes. *Proceedings of the 12th ACM SIGKDD Conference*. 2006.

[9] Jain, P. L. *Quality Control & Total Quality Management*. Tata Mcgraw Hill Publishing Co Ltd. 2001.

[10] Janakiraman, V. M., Matthews, B. L., Oza. N. C. Discovery of Precursors to Adverse Events using Time Series Data. *SDM* 2016: 639-647.

[11] Johansson, U., Niklasson, L., Köning, R. Accuracy vs. comprehensibility in data mining models. *Proceedings of the Seventh International Conference on Information Fusion*. Stockholm, Sweden. 2004. pp. 295–300.

[12] *Large Property Damage Losses in the Hydrocarbon Industry: The 100 Largest Losses 1974–2013*, Marsh, 2004.

[13] Lichman, M. UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. 2013.

[14] Mueen, A. et al. MASS: The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance. 2015. www.cs.unm.edu/~mueen/FastestSimilaritySearch.html

[15] Miller, K. J., Schalk, G., Hermes, D., Ojemann, J. G. and Rao, R. P. N. Spontaneous Decoding of the Timing and Content of Human Object Perception from Cortical Surface Recordings Reveals Complementary Information in the Event-Related Potential and Broadband Spectral Change. *PLOS Computational Biology*, 12. 2016.

[16] Ojala M. and Garriga, G. C. Permutation tests for studying classifier performance, *J Mach Learn Res*. 2010, vol. 11.

[17] Oracle Corporation. Driving Real-Time Insight: The Convergence of Big Data and the Internet of Things. *White paper*. 2016.

[18] Ozturk et al. Evaluation of non-surgical causes of cardiac tamponade in children at a cardiac surgery center. *Pediatr Int* 2014; 6:13–18.

[19] Powers, D. M W. Evaluation: From Precision, Recall and F-Measure to ROC. *Informedness, Markedness & Correlation. Journal of Machine Learning Technologies*. 2 (1): 37–63. 2011.

[20] Rakthanmanon T. et. al. Searching and mining trillions of time series subsequences under dynamic time warping. *KDD* 2012: 262-270.

[21] Roverso, D., Multivariate temporal classification by windowed wavelet decomposition and recurrent neural networks, in *3rd ANS Int'l Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface*, vol. 20, Washington, DC, USA, 2000.

[22] Sanders, E. *Chemical Process Safety. Learning from Case Histories*, 3rd ed., Elsevier , Oxford 2005.

[23] Stikic, M., Larlus, D., Ebert, S. and Schiele, B. Weakly supervised recognition of daily life activities with wearable sensors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(12), pp.2521-2537. 2011.

[24] Supporting website: http://www.cs.ucr.edu/~myeh003/weaklyLabeled/

[25] Sweeney, K. T., Ayaz, H., Ward, T. E., Izzetoglu, M., McLoone, S.F., Onaral, B. A. Methodology for Validating Artifact Removal Techniques for Physiological Signals. *IEEE Trans Info Tech Biomed* 16(5):918-926; 2012.

[26] Sweeney, K.., McLoone, S., Ward, T. The use of ensemble empirical mode decomposition with canonical correlation analysis as a novel artifact removal technique. *IEEE transactions on biomedical engineering* 60.1 (2013): 97-105.

[27] Vail, D. and Veloso, M. Learning from accelerometer data on a legged robot. *In Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*. 2004.

[28] Walker, G., Personal Correspondence. February 7, 2017.

[29] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E. J. Experimental comparison of representation methods and distance measures for time series data. *Data Min. Knowl. Discov.* 26(2): 275-309. 2013.

[30] Wikipedia contributors. Golden-section search. *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 26 Jan. 2017. Web. 6 Feb. 2017.

[31] Willett, D. S., George, J., Willett, N. S., Stelinski, L. L., Lapointe, S. L. Machine Learning for Characterization of Insect Vector Feeding. *PLoS Comput Biol* 12(11). 2016.

[32] Witten, I. H., Frank, E., Trigg, L., Hall, M., Holmes, G., Cunningham, S. J. Weka: Practical Machine Learning Tools and Techniques with Java Implementations. *Proceedings of the ICONIP99 Workshop on Emerging Knowledge Engineering and Connectionist-Based Information Systems*. pp. 192–196. 1999.

[33] Yeh, C.-C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D., F., Mueen, A., and Keogh, E. 2016. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets. *IEEE ICDM 2016*.

[34] Zhu, Y., Zimmerman, Z., Senobari, N., S., Yeh, C.-C. M., Funning, G., Mueen, A., Brisk, P., and Keogh, E. 2016. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. *IEEE ICDM 2016.*