Towards Never-Ending Learning from Time Series Streams

Yuan Hao*, Yanping Chen*, Jesin Zakaria, Bing Hu, Thanawin Rakthanmanon#, Eamonn Keogh
Department of Computer Science & Engineering, #Kasetsart University
University of California, Riverside
{yhao,ychen053,eamonn}@cs.ucr.edu

ABSTRACT

Time series classification has been an active area of research in the data mining community for over a decade, and significant progress has been made in the tractability and accuracy of learning. However, virtually all work assumes a one-time training session in which labeled examples of all the concepts to be learned are provided. This assumption may be valid in a handful of situations, but it does not hold in most medical and scientific applications where we initially may have only the vaguest understanding of what concepts can be learned. Based on this observation, we propose a never-ending learning framework for time series in which an agent examines an unbounded stream of data and occasionally asks a teacher (which may be a human or an algorithm) for a label. We demonstrate the utility of our ideas with experiments in domains as diverse as medicine, entomology, wildlife monitoring, and human behavior analyses.

Categories and Subject Descriptors

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

Keywords

Never-Ending Learning, Classification, Data Streams, Time Series

1. INTRODUCTION

Virtually all work on time series classification assumes a one-time training session in which multiple labeled examples of all the concepts to be learned are provided. This assumption is sometimes valid, for example, when learning a set of gestures to control a game or novel HCI interface. However, in many medical and scientific applications, we initially may have only the vaguest understanding of what concepts need to be learned. Given this observation, and inspired by the Never-Ending Language Learning (NELL) research project at CMU [6], we propose a time series learning framework in which we observe streams forever, and we continuously attempt to learn new (or drifting) concepts.

Our ideas are best illustrated with a simple visual example. In Figure 1, we show a time series produced by a light sensor at Soda Hall in Berkley. While the sensor will produce data forever, we can only keep a fixed amount of data in a buffer. Here, the daily periodicity is obvious, and a more careful inspection reveals two *very* similar patterns, annotated A and B.

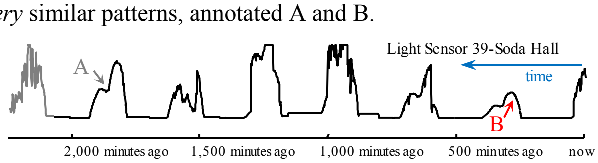

Figure 1: The light sensors at Soda Hall produce a never-ending time series, of which we can cache only a small subset main memory.

*Joint first authors.

As we can see in Figure 2.*left* and Figure 2.*center*, these patterns are even more similar after we z-normalize them [8]. Suppose that the appearance of these two similar patterns (or "motif") causes an agent to query a teacher as to their *meaning*.

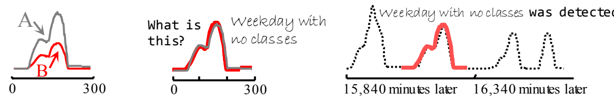

Figure 2: *left*) A "motif" of two patterns annotated in Figure 1 aligned to highlight their similarity. *center*) We imagine asking a teacher for a label for the pattern. *right*) This allows us to detect and classify a new occurrence eleven days later.

This query could be implemented in a number of ways; moreover, the teacher need not necessarily be human. Let us assume here that an email is sent to the building supervisor with a picture of the patterns and any other useful metadata. If the teacher is willing to provide a label, in this case Weekday with no classes, we have learned a concept for this time series, and we can monitor for future occurrences of it.

An important generalization of the above is that the time series may only be a *proxy* for another much higher dimensional streaming data source, such as video or audio. For example, suppose the classrooms are equipped with surveillance cameras, and we had conducted our monitoring at a finer temporal resolution, say seconds. We could imagine that our algorithm might notice a novel pattern of short-lived but dramatic spikes in light intensity. In this case we could send the teacher not the *time series* data, but some short *video clips* that bracket the events. The teacher might label the pattern Camera use with flash. This idea, that the time series is only a (more tractable) proxy for the real stream of interest, greatly expands the generality of our ideas, as time series has been shown to be a useful proxy of audio, video, text, networks, and a host of other types of data [5].

This example elucidates our aims, but suggested a wealth of questions. How can we detect repeated patterns, especially when the data arrives at a *much* faster rate, and the probability of two patterns from a rare concept appearing close together is very small? Assuming the teacher is a finite or expensive resource, how can we optimize the set of questions we might ask of it/him/her, and how do we act on this feedback?

The rest of this paper is organized as follows. In Section 2, we briefly discuss related work before explaining our system architecture and algorithms in Section 3. We provide an empirical evaluation on a host of diverse domains in Section 4, and in Section 5, we offer conclusions and directions for future work.

2. RELATED WORK

The task at hand requires contributions from, and an understanding of, many areas, including: frequent item mining [7], time series classification [8], hierarchical clustering, crowdsourcing, active learning [20], semi-supervised learning, etc. It would be impossible to consider all these areas with appropriate depth in this work; thus, we refer the reader to [13] where we have a detailed bibliography of the many research efforts we draw from.

However, it would be remiss of us not to mention the groundbreaking NELL project lead by Tom Mitchell at CMU [6], which is the inspiration for the current work. Note, however, that the techniques used by NELL are informed by very different assumptions and goals. NELL is learning *ontologies* from *discrete* data that it can crawl *multiple* times. In contrast, our system is learning prototypical time series *templates* from *real-valued* data that it can only see *once*.

The work closest in spirit to ours in the *time series* domain is [3]. Here, the authors are interested in a human activity inference system with an application to psychiatric patient monitoring. They use time series streams from a wrist worn sensor to detect *dense motifs*, which are used in a periodic (every few weeks) retrospective interview/assessment of the patient. However, this work is perhaps best described as a *sequence* of *batch* learning, rather than a true *continuous* learning system. Moreover, the system requires at least seven parameters to be set and significant human intervention. In contrast, our system requires few (and relatively non-critical) parameters, and where humans are used as teachers, we limit our demands of them to providing labels only.

3. ALGORITHMS

The first decision facing us is which base classifier to use. Here, the choice is easy; there is near universal agreement that the special structure of time series lends itself particularly well to the nearest neighbor classifier [8][14][18]. This only leaves the question of which *distance measure* to use. There is increasing empirical evidence that the best distance measure for time series is either Euclidean Distance (ED), or its generalization to allow time misalignments, Dynamic Time Warping (DTW) [8]. DTW has been shown to be more accurate than ED on some problems; however, it requires a parameter, the *warping window width*, to be carefully set using training data, which we do not have.

Because ED is parameter-free, computationally more tractable, allows several useful optimizations in our framework (triangular inequality etc.), and works *very* well empirically [8][18], we use it in this work. However, nothing in our overarching architecture specifically precludes other measures.

3.1 Overview of System Architecture

We begin by stating our assumptions:

- We assume we have a never-ending[1] data stream **S**.

S could be an audio stream, a video stream, a text document stream, multi-dimensional time series telemetry, etc. Moreover, **S** could be a combination of any of the above. For example, all broadcast TV in the USA has simultaneous video, audio, and text.

- Given **S**, we assume we can record or create a real-time *proxy* stream *P* that is "parallel" to **S**.

P is simply a single time series that is a low-dimensional (and therefore easy to analyze in real time) proxy for the higher dimensional/higher arrival rate stream **S** that we are interested in. In some situations, *P* may be a *companion* to **S**. For example, in [4], which manually attempts some of the goals of this work, **S** is a night-vision camera recording sleeping postures and *P* is a time series stream from a sensor worn on the wrist of the sleeper. In other cases, *P* could be a *transform* or low-dimensional projection of **S**. In one example we consider, **S** is a stereo audio stream recorded at 44,100Hz, and *P* is a single channel 100Hz Mel-frequency cepstral coefficient (MFCC) transformation of it. Note that our framework includes the possibility of the special case where **S** = *P*, as in Figure 1.

- We assume we have access to a teacher (or *Oracle* [20]), possibly at some cost.

The space of possible teachers is large. The teacher may be *strong*, giving only correct labels to examples, or *weak*, giving a set of probabilities for the labels. The teacher may be *synchronous*, providing labels on demand, or *asynchronous*, providing labels after a significant delay, or at fixed intervals.

Given the sparseness of our assumptions and especially the generality of our teaching model, we wish to produce a very general framework in order to address a wealth of domains. However, many of these domains come with unique domain specific requirements. Thus, we have created the framework outlined in Figure 3, which attempts to divorce the domain dependent and domain independent elements.

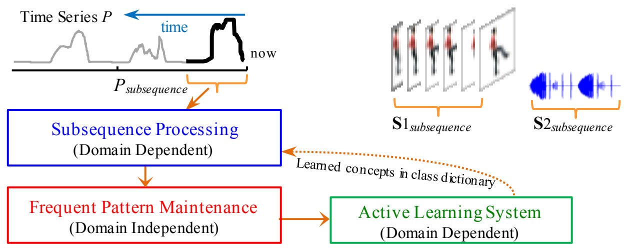

Figure 3: An overview of our system architecture. The time series *P* which is being processed may actually be a proxy for a more complex data source such as audio or video (*top right*).

Recall that *P* itself may be the signal of interest, or it may just be a proxy for a higher dimensional stream **S**, such as a video or audio stream, as shown in Figure 3.*top.right*.

Our framework is further explained at a high level in Table 1. We begin in Line 1 by initializing the class dictionary, in most cases just to `empty`. The dictionary format is defined in Section 3.2. We then initialize a dendrogram of size *w*. We will explain the motivation for using a dendrogram in Section 3.4. This dendrogram is initialized with random data, but as we shall see, these random data are quickly replaced with subsequences from *P* as the algorithm runs.

After these initialization steps, we enter an infinite loop in which we repeatedly extract the next available subsequence from the time series stream *P* (Line 4), then pass it to a module for *subsequence processing*. In this unit, domain dependent normalization may take place (Line 5), and we will attempt to classify the subsequence using the class dictionary. If the subsequence is not classified and is regarded as valid (cf. Section 3.3), then it is passed to the frequent pattern maintenance algorithm in Line 6, which attempts to maintain an approximate history of all data seen thus far. If the new subsequence is similar to previously seen data, this module may signal this by returning a new 'top' motif. In Line 7, the active learning module decides if the current top motif warrants seeking a label. If the motif is labeled by a teacher, the current dictionary is updated to include this now *known* pattern.

Table 1: The Never-Ending Learning Algorithm

```
Algorithm: Never_Ending_Learning(S,P,w)
1  dict ← initialize_class_dictionary
2  global dendro = create_random_dendrogram_of_size(w)
3     For ever
4        sub  ← get_subsequence_from_P(S,P)
5        sub  ← subsequence_processsing(sub, dict)
6        top  ← frequent_pattern_maintenance(sub)
7        dict ← active_learning_system(top, dict)
8     End
```

[1] For our purposes, a "never-ending" stream may only last for days or hours. The salient point is the contrast with the *batch* learning algorithms that the vast majority of time series papers consider [8].

In the next four subsections, we expand our discussion of the class dictionary and the three major modules introduced above.

3.2 Class Dictionaries

We limit our representation of a class concept i to a triple containing: a prototype time series, C_i; its associated threshold, T_i; and $Count_i$, a counter to record how often we see sequences of this class. As shown in Figure 4.*right*, a class dictionary is a set of such concepts, represented by M triples.

Unlabeled objects that are within T_i of class C_i under the Euclidean distance are classified as belonging to that class. Figure 4.*left* illustrates the representational power of our model. Note that because a single class could be represented by two or more templates with different thresholds (i.e. *Weekend* in Figure 4.*right*), this representation can in principle approximate any decision boundary. It has been shown that for time series problems this simple model can be *very* completive with more complex models [14], at least in the case where both C_i and T_i are carefully set.

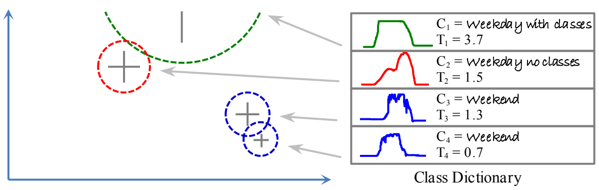

Figure 4: An illustration of the expressiveness of our model.

It is possible that the volumes that define two *different* classes could overlap (as C_1 and C_2 slightly do above) and that an unlabeled object could fall into the intersection. In this case, we assign the unlabeled object to the nearest center. We reiterate that this model is adopted for *simplicity*; nothing in our overall framework precludes more complex models, using different distance measures [8], using logical connectives [18], etc.

As shown in Table 1-Line 1 our algorithm begins by initializing the class dictionary. In most cases it will be initialized as `empty`; however, in some cases, we may have some domain knowledge we wish to "prime" the system with. For example, as shown in Figure 5, our experience in medical domains suggests that we should initialize our system to recognize and ignore the ubiquitous flatlines caused by battery/sensor failure, patient bed transfers, etc.

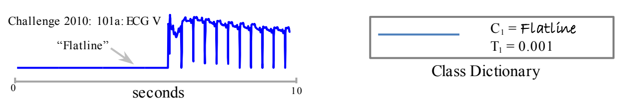

Figure 5: *left*) Sections of constant "flatline" signals are so common in medical domains that it is worth initializing the medical dictionaries with an example (*right*), thus suppressing the need to waste a query asking a teacher for a label for it.

Whatever the size of the *initial* dictionary, it can only increase by being appended to by the active learning module, as suggested in Line 7 of Table 1 and explained in detail in Section 3.5.

3.3 Subsequence Processing

Subsequence processing refers to any domain specific preprocessing that must be done to prepare the data for the next stage (frequent pattern mining). We have already seen in Figure 1 and Figure 2 that z-normalization may be necessary [8]. More generally, this step could include downsampling, smoothing, wandering baseline removal, taking the derivative of the signal, filling in missing values, etc. In some domains, very specialized processing may take place. For example, for ECG datasets, robust beat extraction algorithms exist that can detect and extract full individual heartbeats, and as we show in Section 4.2, converting from the time to the frequency domain may be required [2].

As shown in Table 2-Line 3, after processing, we attempt to classify the subsequence by comparing it to each time series in our dictionary and assigning its class label to its nearest neighbor, if and only if it is within the appropriate threshold. If that is the case, we increment the class counter and the subsequence is simply discarded without passing it to the next stage.

Table 2: The Subsequence Processing Algorithm

```
Algorithm: sub = subsequence_processsing(sub,dict)
1  sub ← domain_dependent_processing(sub)
2  [dist,index] ← nearest_neighbor_in_dictionary(sub,dict)
3  if dist < T_index          // Item can be classified
4    disp('An instance of class ' index ' was detected!')
5    count_index ← count_index + 1
6    sub ← null;              // Return null to signal that no
7  end                        // further processing is needed
```

Assuming the algorithm processes the subsequence and finds it is unknown, it passes it onto the next step of frequent pattern maintenance, which *is* completely domain independent.

3.4 Frequent Pattern Maintenance

As we discuss in more detail in the next section, any attempt to garner a label must have some cost, even if only CPU time. Thus, as hinted at in Figure 1/Figure 2, we plan to only ask for labels for patterns which appear to be repeated with some minimal fidelity. This reflects the intuition that a repeated pattern probably reflects some conserved concept that *could* be learned.

The need to detect repeated time series patterns opens a host of problems. Note that the problem of maintaining *discrete* frequent items from unbounded streams in bounded space is known to be unsolvable in general, and thus has opened up an active area of research in approximation algorithms for this task [7]. However, we have the more difficult task of maintaining *real-valued* and high dimensional frequent items. The change from discrete to real-valued causes two significant difficulties.

- **Meaningfulness**: We never expect two real-valued items to be equal, so how can we define a *frequent* time series?
- **Tractability**: The high dimensionality of the data objects, combined with the inability to avail of common techniques and representations for *discrete* frequent pattern mining (hashing, graphs, trees, and lattices [7]) seems to bode ill for our hopes to produce a highly tractable algorithm.

Fortunately, these issues are not as problematic as they may seem. Frequent item mining algorithms for discrete data must handle million-plus Hertz arrival rates [7]. However, most medical/human behavior domains have arrival rates that are rarely more than a few hundred Hertz. Likewise, for *meaningfulness*, a small Euclidean distance between two or more time series tells us that a pattern has been (approximately) repeated.

We begin with the intuition of our solution to these problems. For the moment, imagine we can relax the space and time limitations, and that we could buffer *all* the data seen thus far. Further imagine, as shown in Figure 6, that we could build a dendrogram for all the data. Under this assumption, frequent patterns would show up as dense subtrees in the dendrogram.

Given this intuition, we have just two problems to solve. The first is to produce a concrete definition of "unusually dense subtree." The second problem is to efficiently maintain a dendrogram in constant space with unbounded streaming data.

While our constant space dendrogram can only approximate the results of the idealized ever-growing dendrogram, we have good reason to suspect this will be a good approximation. Consider the dense subtree shown in Figure 6; even if our constant space algorithm discards any two of the four sequences in this clade, we would *still* have a dense subtree of size two that would be sufficient to report the existence of a repeated pattern. We will revisit this intuition with more rigor below.

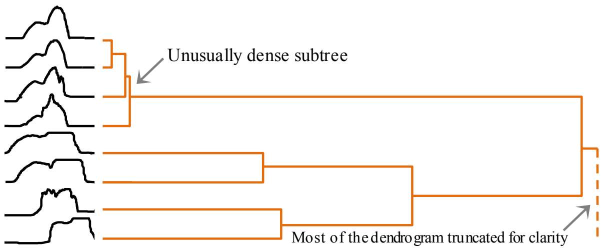

Figure 6: A visual intuition of our solution to the frequent time series subsequence problem. The elements in a dense subtree (or clade) can be seen as a frequent pattern.

We will maintain a dendrogram of size *w* in a buffer, where *w* is as large as possible given the space or (more likely) time limitations imposed by the domain. At most once per each time step[2], the Subsequence Processing Module will hand over a subsequence for consideration. After this happens a subsequence from the dendrogram will be randomly chosen to be discarded in order to maintain constant space. At all times, our algorithm will maintain the top most significant patterns in the dendrogram, and it is *only* this top-1 motif that will be visible to the active learning module discussed below.

In order to define *most significant motif* more concretely, we must first define one parameter, *MaxSubtreeSize*. The dense subtree shown in Figure 6 has four elements; a dense subtree may have fewer elements, as few as two. However, what should be the maximum allowed number of elements? If we allow the maximum to be a significant fraction of *w*, the size of the dendrogram, we can permit pathological solutions, as a subtree is only dense relative to the rest of the tree. Thus, we define MaxSubtreeSize to be a small constant. Empirically, the exact value does not matter, so we simply use six throughout this work.

We calculate the significance of the top motif in the following way. Offline, we take a sample time series from the domain in question and remove existing patterns by permuting the data. We use this "patternless" data to create multiple dendrograms with the same parameters we intend to monitor *P* under. We examine these dendrograms for all possible sizes of subtrees from two to MaxSubtreeSize, and as shown in Figure 7 we record the mean and standard deviation of the heights of these subtrees.

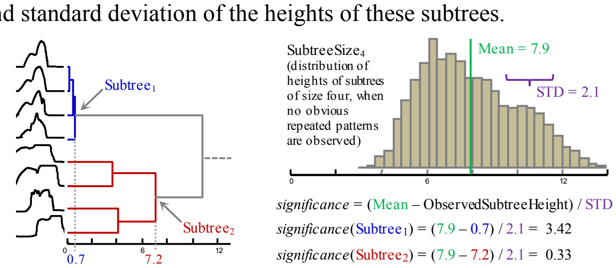

Figure 7: *left*) The (partial) dendrogram shown in Figure 6 has its subtrees of size four ranked by density. *right*) The *observed* heights of the subtrees are compared to the *expected* heights given the assumption of no patterns in the data.

These distributions tell us what we should expect to see if there are *no* frequent patterns in the new data stream *P*, as clusters of frequent patterns will show up as unusually dense subtrees. These distributions allow us to examine the subtrees of the currently maintained dendrogram and rank them according to their *significance*, which is simply defined as the number of standard deviations less than the mean is the height of the ancestor node. Thus, the *significance* of subtree$_i$, which is of size *j* is:

$$significance(Subtree_i) = \frac{mean(AllSubtreeSize_j.height) - Subtreet_i.height}{STD(AllSubtreeSize_j.height)}$$

For example, in Figure 7.*right*, we see that Subtree$_1$ has a score of 3.42, suggesting it is much denser than expected. Note that this measure makes differently-sized subtrees commensurate.

There are two issues we need to address to prevent pathological solutions.

- **Redundancy**: Consider Figure 7.*left*. If we report Subtree$_1$ as the most significant pattern, it would be fruitless to report a contained subtree of size two as the next most significant pattern. Thus, once we find the i^{th} most significant subtree, all its descendant and ancestor nodes are excluded from consideration for the i^{th}+1 to *K* most significant subtrees.
- **Overflow**: Suppose we are monitoring an accelerometer on an individual's leg. If she goes on a long walk, we might expect that single gait cycles might *flood* the dendrogram, and diminish our ability to detect other behaviors. Thus, we allow any subtree in the current list of the top *K* to grow up to MaxSubtreeSize. After that point, if a new instance is inserted into this subtree, we test to see which of the MaxSubtreeSize + 1 items can be discarded to create the tightest subtree of size MaxSubtreeSize, and the outlying object is discarded.

In Table 3, we illustrate a high level overview of the algorithm.

Table 3: Frequent Pattern Maintenance Algorithm

```
Algorithm: top = frequent_pattern_maintenance(sub)
if sub == null                          // If null was passed in,
  top ← null; return;                   // do nothing, return null
else
  dendro ← insert(dendro,sub)  // |dendro| is now w + 1
  top ← find_most_significant_subtree(dendro)
  dendro ← discard_a_leaf_node(dendro) // back to size w
end
```

Our frequent pattern mining algorithm has only a single value, *w* the number of objects we can keep in the buffer, which affects its performance. This is not really a free *parameter*, as *w* should be set as large as possible, *given* the more restrictive of the time *or* space constraints. However, it is interesting to ask how large *w* needs to be to allow successful learning. A detailed analysis is perhaps worthy of its own paper, so we will content ourselves here with a brief intuition. Imagine a version of our problem, simplified by the following assumptions. One in one hundred subsequences in the data stream belong to the same pattern; everything else is random data. Moreover, assume that we can unambiguously recognize the pattern the moment we see any two examples of it. Under these assumptions, how does the size of *w* affect how long we expect to wait to discover the pattern? Figure 8 shows this relationship for several values of *w*.

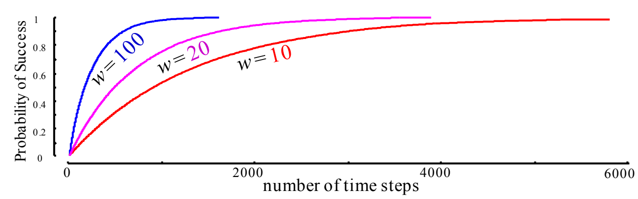

Figure 8: The average number of time steps required to find a repeated pattern with a desired probability for various values of *w*. All curves end when they reach 99.5%.

[2] Recall from Section 3.3 that the Subsequence Processing Module may choose to *discard* a subsequence rather than pass it to Frequent Pattern Maintenance.

If w is set to ten, we must wait about 5,935 time steps to have at least a 99.5% chance of finding the pattern. If we increase w by a factor of ten our wait time does decrease, but only by a factor of 3.6. In other words, there are rapidly diminishing returns for larger and larger values of w. These results are borne out by experiments on real datasets (cf. Section 4). A pathologically small value for w, say w = 2, will almost never stumble on a repeated pattern. However, once we make w large enough, we can easily find repeated patterns, and making w larger again makes no perceptible difference. The good news is that "large enough" seems to be a surprisingly small number, of the order of a few hundred for the many diverse domains we consider. Such values are easily supported by off-the-shelf hardware or even smartphones. In particular, *all experiments* in this paper are performed in real time on cheap commodity hardware.

Finally, we note that there clearly exist real-world problems with extraordinarily rare patterns that would push the limits of our current naive implementation. However, it is important to note that our description was optimized for clarity of presentation and brevity, *not* efficiency. We can take advantage of recent research in online [1] and incremental [19] hierarchical clustering to bring the cost per time step down to O(w).

3.5 Active Learning System

The active learning system which exploits the frequent patterns we discovered must be domain dependent. Nevertheless, we can classify two broad approaches depending on the teacher (oracle) available. Teachers may be:

- **Strong Teachers** which are assumed to give correct and unambiguous class labels. Most, but not all, strong teachers are humans. Strong teachers are assumed to have a significant cost.
- **Weak Teachers** which are assumed to provide more tentative labels. Most, but not all, weak teachers are assumed to be algorithms; however, they could be input of a crowdsourcing algorithm or a classification algorithm that makes errors but performs above the default rate.

The ability of our algorithm to maintain frequently occurring time series opens a plethora of possibilities for active learning. Two common frameworks for active learning are Pool-Based sampling and Stream-Based sampling [20]. In Pool-Based sampling, we assume there is a pool of unlabeled data available, and we may (at some cost) request a label for some instances. In Stream-Based sampling, we are presented with unlabeled examples one at a time and the learner must decide whether or not it is worth the cost to request its label. Our framework provides opportunities that can take advantage of both scenarios; we are *both* maintaining a pool of instances in the dendrogram *and* we also see a continuous stream of unlabeled data.

Because this step is necessarily domain dependent, we will content ourselves here with giving real world examples and defer creating a more general framework to future work.

Given our dictionary-based model, the only questions that remains are *when* we should trigger a query to the teacher, and *what action* we should take given the teacher's feedback.

3.5.1 When to trigger queries

Different assumptions about the teacher model and its associated costs can lead to different triggering mechanisms [20]. However, most frameworks can reduce to questions of how frequently we should ask questions. A conservative questioner that only asks questions rarely may miss opportunities to learn concepts, whereas an aggressive questioning policy will accumulate large costs and will frequently ask questions about data that are unlikely to represent any concept.

For any given domain, we assume that the teacher will tell us how many queries on average they are willing to answer in a given time period. For example, our cardiologist (c.f. Section 4.2) is willing to answer two queries per day from a system recording a healthy adult patient undergoing a routine sleep study, but twenty queries per day from a system monitoring a child in an ICU who has had recent increase in her SOFA score [10].

Let SR be the sampling rate of P, and QR be the mean number of seconds between queries that the teacher is willing to tolerate. We can then calculate the trigger threshold as:

$$trigger\ threshold\ = -\,probit(1/(SR * QR))$$

Where *probit* is the standard statistical function. We defer a detailed derivation to [13]. This equation assumes the distributions of heights of subtrees (e.g. Figure 7.*right*) are approximately Gaussian, a reasonable assumption when $j \ll w$.

3.5.2 Learning a concept: Strong teacher case

In Table 4, the active learning system begins by comparing the *significance* (c.f. Section 3.4) of the top motif to this user supplied *trigger threshold*. If the motif warrants bothering the teacher, the `get_labels` function is invoked. The exact implementation of this is domain dependent, requiring the teacher to examine images, short audio or video snippets, or in one instantiation we discuss below, the bodies of insects, and provide labels for these objects. Once the labels have been obtained, then in Line 5 the dictionary is updated.

We have two tasks when updating the dictionary. First we must create the concept C_i; we can do this by either averaging the objects in the motif or choosing one randomly. Empirically, both perform about the same, which is unsurprising since the variance of the motif must be very low to pass the *trigger threshold*. The second thing we must do is decide on a value for threshold T_i. Here we could leverage off a wealth of recent advances in One-Class Classification [9]; however, for simplicity we simply set the threshold T_i to three times the `top` subtree's height. As we shall see, this simple idea works so well that more sophisticated ideas are not warranted, at least the domains we investigated.

Table 4: The Active Learning Algorithm

```
Algorithm:dict = active_learning_system(top,dict)
1  if (significance(top)<trigger threshold) // The subtree is not
2     dict ← dict; return;                  // worth investigating
3  elseif in_strong_teacher_mode
4    labels ← get_labels(top)
5    dict ← update_dictionary(dict,top,labels)
6  else
7    spawn_weak_learner_agent(top)
8  end
```

3.5.3 Learning a concept: Weak teacher case

A weak teacher can leverage off side information. For concreteness, we will give an illustration that closely matches an experiment we consider in Section 4.6; however, we envision a host of possible variants (hence our insistence that this phase be domain dependent). As illustrated in Figure 9.*top*, we can measure the X-axis acceleration on the wrist of the subject as he works with various tools. Moreover, RFID tags mounted on the tools can produce binary time series which record which tools are close to the user's hand, although these binary sensors clearly cannot encode any information about whether the tool is being used or carried or cleaned, etc. At some point, our active learning algorithm is invoked in weak teacher mode with pattern C_1, which happens (although we do not know this) to correspond to an axe swing.

The weak teacher simply waits for future occurrences of the pattern to be observed, and then, as shown in Figure 9.*middle*, immediately polls the binary sensors for clues as to C_1's label.

In the example shown in Figure 9.*bottom*, after the first detection of C_1, we have one vote for `Axe`, one for `Cat`, and zero for `Bar`. However, by the third detection of C_1, we have seen three votes for `Axe`, one for Bar, and one for Cat. Thus, we can compute that the most likely label for C_1 is `Axe`, with a probability of 0.6 = 3 / (3 + 1 +1).

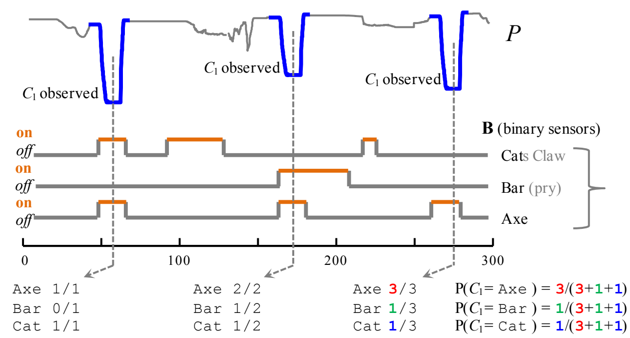

Figure 9: An illustration of a weak teacher. *top*) A stream *P* in which we detect three occurrences of the pattern C_1. *middle*) At the time of detection we poll a set of binary sensors to see which of them are active. *bottom*) We can use the frequency of associations between a pattern and binary "votes" to calculate probabilities for C_1's class label.

This simple weak teaching scheme is the one we use in this work and we empirically evaluate it in Section 4.6. However, we recognize that more sophisticated formulations can be developed. For example, our approach assumes that the binary sensors are mostly in the *off* position. A more robust method would look at the prior probability of a sensor's state and the dependence between sensors. Our point here is simply to provide an existence proof of a system that can learn without human intervention.

Finally, note that the sensors polled do not have to be *natively* binary. They could be normally real-valued; for example, an accelerometer time series can be discretized to binary {*has moved in the last 10-sec*, *has not moved in the last 10-sec*}.

4. EXPERIMENTS

We begin by noting that all code and data used in this paper, together with additional details and many additional experiments, are archived in perpetuity at [13].

While true never-ending learning systems are our ultimate goal, here we content ourselves with experiments that last from minutes to days. Our experiments are designed to demonstrate the vast range of problems we can apply our framework to.

We do not consider the effect of varying *w* on our results. As noted in Section 3.4, once it is set to a reasonable value (typically around 250), its value makes almost no difference and we can process streams with such values in real-time for all the problems considered below.

Because our system discards subsequences randomly, where possible, we test each dataset 100 times and report the average performance. For each class, we report the number of times the class is learned as well as the average precision and recall [22]. To compute the average precision and recall, we count in each run the number of true positives, false positives, and false negatives *after* the class is first added to the dictionary.

4.1 Activity Data

We begin with a short but visually intuitive domain, the activity dataset of [24]. This dataset consists of a 13.3 minute 10-fps video sequence (thresholded to binary by the original authors) of an actor performing one of eight activities. From this data, the original authors extracted 721 optical flow time series. We randomly chose just *one* of these time series to act as *P*, with **S** being the original video.

We set our trigger threshold to 3.5, which is the value that we expect to spawn about three requests for labels on each run, and we assume a label is given after a delay of ten seconds (Figure 10.*left* shows the first query shown to the teacher on the first run.

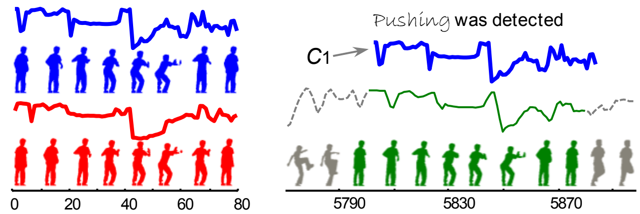

Figure 10: *left*) A query shown to the user during a run on the activity dataset; the teacher labeled it Pushing and a new concept C_1 was added to the dictionary. *right*) About 9.6 minutes later, the classifier detected a new example of the class.

The teacher labeled this Pushing, and the concept was inserted into the dictionary. About 9.6 minutes later, this classifier correctly claimed to spot a new example of this class, as shown in Figure 10.*right*.

This dataset has the interesting property that the actor starts in a canonical pose and returns to it after completing the scripted action at eight-second intervals. This means that we can permute the data so long as we only "cut and paste" at multiples of eight seconds. This allows us to test over one hundred runs and smooth our performance estimates.

Averaged over one hundred runs, we achieved an impressive 41.8% precision and 87.96% recall on the **`running`** concept. On some other concepts, we did not fare so well. For example, we only achieved 19.87% precision and 51.01% recall on the **`smoking`** concept. However, this class has much higher variability in its performance, and recall that we only used a *single* time series of the 721 available for this dataset.

4.2 Invasive Species of Flying Insects

Recently, it has been shown that it is possible to accurately classify the *species*[3] of flying insects by transforming the faint audio produced by their flight into a periodogram and doing nearest neighbor time series classification on this representation [2]. Figure 11 demonstrates the practicality of this idea.

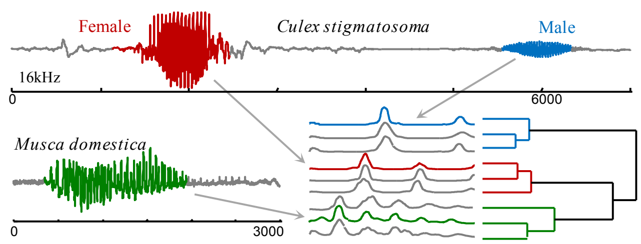

Figure 11: *top*) An audio snippet of a female *Cx. stigmatosoma* pursued by a male. *bottom left*) An audio snippet of a common house fly. *bottom right*) If we convert these sound snippets into periodograms we can cluster and classify the insects.

This allows us to classify *known* species, for example, species we have raised in our lab to obtain training data. However, in many insect monitoring settings we are almost guaranteed to encounter some unexpected or invasive species; can we use our framework to detect and classify them? At first blush, this does

[3] And for some sexually dimorphic species such as mosquitoes, the *sex*.

not seem possible. The **S** data source is a high quality audio source, and while entomologists could act as our teachers, at best they could recognize the sound at the family level, i.e. some kind of Apoidea (bee). We could hardly expect them to recognize which of the 21,000 or so species of bee they heard.

We had considered augmenting **S** with HD video, and sending the teacher short video clips of the novel insects. However, many medically and agriculturally important insects are tiny; for example, some species of *Trichogramma* (parasitic wasps) are just 0.2 mm, about the size of the period at the end of this sentence.

Our solution is to exploit the fact that some insect traps can *physically* capture the flying insects themselves and record their time of capture [17]. Thus, the **S** data source is audio snippets of the insects as they flew into the trap *and* the physical bodies of insects. Naturally, this causes a delay in the teaching phase, as we cannot digitally transmit **S** to the teacher but must wait until she comes to physically inspect the trap once a day.

Using insects raised from larvae in our lab, we learned two concepts: *Culex stigmatosoma* male ($C_{stig♂}$) and female ($C_{stig♀}$). These concepts are just the periodograms shown in Figure 11 with the thresholds that maximized cross-validated accuracy.

With the two concepts now hard coded into our dictionary, we performed the following experiments. On day one we released 500 *Cx. stigmatosoma* of each sex, together with two members of an *invasive species*. If we cannot detect the invasive species, we increase their number for the next day, and try again until we do detected them, After we detected the invasive species, the next day we released 500 of them with 500 *Cx. stigmatosoma* of each sex and measured the precision/recall of detection for all three classes. We repeated the whole procedure for three different species to act as our invasive species. Table 5 shows the results.

Table 5: Our Ability to Detect then Classify Invasive Insects

Number of insects before detection		Precision / Recall		
invasive species name	triggered	*invasive species*	$C_{stig♂}$	$C_{stig♀}$
Aedes aegypti ♂	3	**0.91 / 0.86**	0.88/0.94	0.96/0.92
Culex tarsalis ♂	3	**0.57 / 0.66**	0.58/0.78	1.00/0.95
Musca domestica ♂ and ♀	7	**0.98 / 0.73**	0.99/0.95	0.96/0.94

Recall that the results for $C_{stig♂}$ and $C_{stig♀}$ test only the representational power of the dictionary model, as we learned these concepts offline. However, the results for the three invasive species *do* reflect our ability to learn rare concepts (just 3 to 7 sub-second occurrences in 24 hours), and having learned these concepts, we tested our ability to use the dictionary to accurately detect further instances. The only invasive species for which we report less than 0.9 precision is *Cx. tarsalis* ♂, which is a sister species of the *Cx. stigmatosoma*, and thus it is not surprising that our precision falls to a (still respectable) 0.57.

4.3 Long Term Electrocardiogram

We investigated BIDMC Dataset ch07, a 20-hour long ECG recorded from a 48-year old male with severe congestive heart failure [11][12]. This record has 17,998,834 data points containing 92,584 heartbeats. As shown in Table 6, the heartbeats have been independently classified into five types.

Table 6: The ground truth frequencies of beats in BIDMC$_{ch}$07

Name	Abbreviation	Frequency (%)
Normal	N	97.752
R-on-T Premature Ventricular Contraction	r	1.909
Supraventricular Premature or Ectopic Beat	S	0.209
Premature Ventricular Contraction	V	0.104
Unclassifiable Beat	Q	0.025

In Figure 12, we can see this data has both intermittent noise and a wandering baseline; we did *not* attempt to remove either.

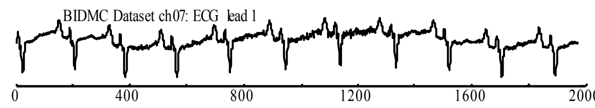

Figure 12: A small snippet (0.0065%) of BIDMC$_{ch}$07 Lead 1.

Let us consider a single test run. After 45 seconds, the system asked for a label for the pattern shown in Figure 13.*left*. Our teacher, Dr. Criley[4], gave the label Normal(N). Just two minutes later, the system asked for a label for the pattern shown in Figure 13.*center*; here, Dr. Criley annotated the pattern as R-on-T PVC (r).

These two requests happened so quickly that the attending physician that hooked up the ECG apparatus will be in the same room and able to answer the queries directly. The next request for a label does not occur for another 9.5 hours, and we envision it being sent by email to the teacher. As shown in Figure 13.*right*, our teacher labeled it PVC (V).

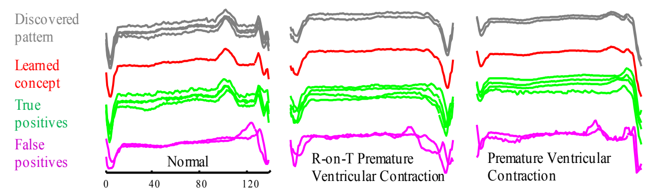

Figure 13: *left to right*) Three patterns discovered in our ECG experiment. *top to bottom*) The motif discovered and used to query the teacher. The learned concept. Some examples of true positives. Some examples of false positives.

In this run, the class (S) was also learned, but just thirty minutes before the end of the experiment. We did not discover class (Q); however, it is *extremely* rare and as hinted at by its name (Unclassifiable Beat), *very* diverse in its appearance.

Because the data has been independently annotated beat-by-beat by an algorithm, we can use this ground truth as a virtual teacher and run our algorithm 100 times to find the average precision and recall, as shown in Table 7. We note, however, that our cardiologist examined some of the "false positives" of our algorithm and declared them to be *true* positives, suggesting that some of the annotations on the original data are incorrect. In fairness, [12] notes the data was "*prepared using an automated detector and has not been corrected manually*." Thus, we feel the numerical results here are pessimistic.

Table 7: Results on BIDMC$_{ch}$07

Class	Detection Rate	Precision	Recall
Normal (N)	100%	0.9978	0.9948
R-on-T PVC (r)	100%	0.9147	0.8080
Supraventricular (S)	100%	0.5028	0.4141
PVC (V)	100%	0.2342	0.6775
Unclassifiable (Q)	0%	-	-

Beyond the objectively correct cardiac dysrhythmias discovered by our system, we frequently found our algorithm has the ability to surprise us. For example, after eighteen minutes of monitoring BIDMC-chf07-lead 2 [12], the algorithm asked for a label for the extraordinary repeated pattern shown in Figure 14.

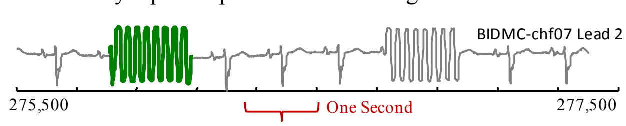

Figure 14: A pattern (green/bold) shown with surrounding data for context, discovered in lead 2 of BIDMC$_{ch}$07.

[4] Dr. John Michael Criley, MD, FACC, MACP is Professor Emeritus at the David Geffen School of Medicine at UCLA.

The label given by the teacher, Dr. Criley, was "interference from nearby electrical apparatus: probably infusion pump." Having learned this label, our algorithm detected fifty-nine more occurrences of it in the remaining twenty hours of the trace. A careful retrospective examination of the data suggests that the algorithm had perfect precision/recall on this unexpected class.

4.4 Bird Song Classification

Recently, a worldwide citizen science project called Bat Detective [16] has been using crowdsourcing to attempt to count bat populations by having volunteers classify sounds as one of `{bat, insect, mechanical}` (The latter class is an umbrella term for sounds created by human activities.). In our efforts to volunteer for this project, we noted that the majority of signals the system asked us to classify are wind noise or other low interest signals (see [13] for examples of screenshots. We wondered if our framework would allow more useful queries to be sent to the users, thus making more effective use of their time.

We do not have ready access to bat sounds, so we produced a similar system for bird sounds. To produce a dataset for which we had ground truth, we did the following. We recorded an hour at midnight at the UCR botanical gardens on January 12, 2012. A careful human annotation of the sound file reveals wind noise, voices in the distance, low volume rumbles from aircraft, etc., but no obvious wildlife calls. Using data from xeno-canto.org, we randomly embedded ten examples of short (about 3 seconds) calls of a Tawny Owl in the data. Using the raw audio as **S**, and a single 100Hz Mel-Frequency Cepstral Coefficient (MFCC) as *P*, we ran our algorithm on this data. As Figure 15 shows, our system can easily recover the patterns.

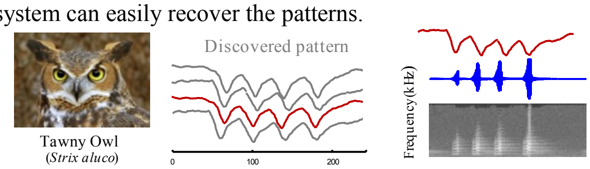

Figure 15: The motif discovered in the first run on the bird dataset. *right*) One snippet in three representations (*bottom-to-top*): a spectrogram, an oscillogram, and the MFCC we used.

The snippets may be heard at [13]. They are easily identifiable as an owl; however, it is less clear if an ornithological crowdsourcing community could identify them as a Tawny Owl.

4.5 Understanding Sapsucking Insect Behavior

Insects in the order *Homoptera* feed on plants by using a feeding tube called a stylet to suck out sap. This behavior is damaging to the plants, and it has been estimated that species in this order cause billions of dollars of damage to crops each year. Given their economic importance, hundreds of researchers study these insects, and increasingly they use a tool called an Electrical Penetration Graph (EPG), which, as shown in Figure 16, adds the insect to an electrical circuit and measures the minuscule changes in voltage that occur as the insect feeds [15].

While there are now about ten widely agreed upon behaviors that experts can recognize in the EPG signals, little progress has been made in automatic classification in this domain. One reason for this is that the 32,000 species that make up order *Homoptera* are incredibly diverse; for example, their size ranges over at least three orders of magnitude. Thus, for many species, an expert could claim of a given behavior, "*I know it when I see it*," but he/she could not expect a template from even a related species to match.

As such, this is a perfect application for our framework, and several leading experts on this apparatus agreed to help us by acting as teachers.

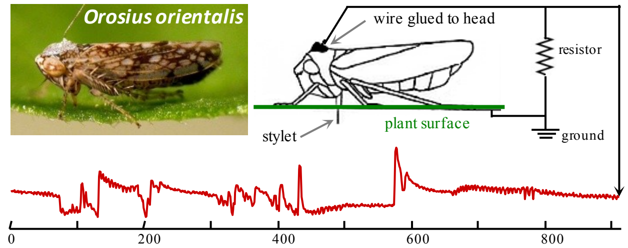

Figure 16: *left*) A tethered brown leafhopper. *right*) A schematic diagram of the circuit for recording EPGs. *bottom*) A snippet of data produced during one of our experiments.

Let us consider a typical run on a dataset consisting of a Beet Leafhopper (*Circulifer tenellus*) recorded by Dr. Greg Walker of UCR Entomology Department. Dr. Elaine Backus of the USDA, one of the co-inventors of the EPG apparatus, agreed to act as the teacher. She was *only* given access to the requests from our system; she could not see the whole time series or the insect itself. After 65 seconds, the system requested a label for the three patterns shown in Figure 17.*top.left*. Dr. Backus labeled the pattern: phloem ingestion with interruption for salivation. After 13.2 minutes, the system requested a label for behavior shown in Figure 17.*top.right*. Dr. Backus labeled this pattern: transition from non-probing to probing. The former learned concept went on to classify twenty-four examples, and the latter concept classified six. Examples of both can be seen in Figure 17.*bottom*.

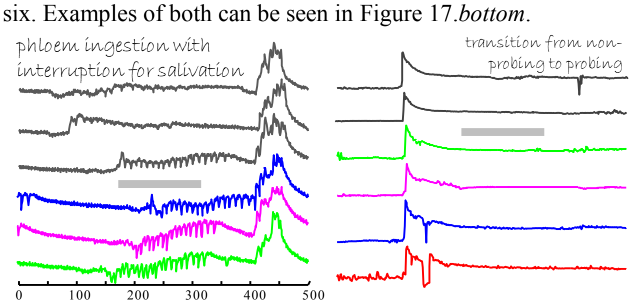

Figure 17: *top-row*) The two concepts discovered in the EPG data. *bottom-row*) Examples of classified patterns.

A careful retrospective study of this dataset suggests that we had perfect precision and recall on this run. Other runs on different datasets in this domain had similar success [13].

4.6 Weak Teaching Example: Elder Care

The use of sensors placed in the environment and/or on parts of the human body has shown great potential in effective and unobtrusive long term monitoring and recognizing the activities of daily living [21][23]. However, labeling accelerometer and sensor data is still a great challenge and requires significant human intervention. In [23], the authors bemoaned the fact that high quality annotation is an order of magnitude slower than real-time, "*A 30-minutes video footage requires about 7-10 hours to be annotated.*" In this example, we leverage off our weak teacher framework to explore how well the framework can label the sensor data without any human intervention.

We consider the dataset of [21] which comes from an activity monitoring and recognition system created using a 3D accelerometer and RFID tags mounted on household objects. A sensor containing both an RFID tag reader and a 3D accelerometer is mounted on the dominant wrist. Volunteers were asked to perform housekeeping activities in any order of their choosing to the natural distribution of activities in their daily life. Thus, the dataset is multidimensional time series with three real-valued and 38 binary dimensions.

For our experiment, we consider just the X-axis acceleration sensor. The active learning algorithm is set in weak teacher mode. After 24 seconds the system finds a concept, C_1, worth exploring (Figure 20.*top.left*). As we can see in Figure 18, our algorithm waited for the next occurrence of the pattern (It happens that three occur close together) and it polls the 38 binary RFID-detected sensors to see which are *on*.

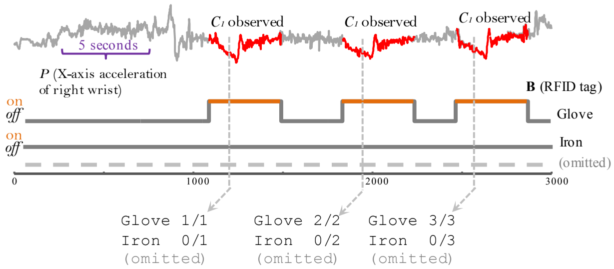

Figure 18: *top*) After we have learned the concept C_1, our system monitors for future occurrences of it. Here, it sees three examples in a row. *bottom*) By polling the binary RFID sensors when a "hit" for C_1 is detected, we can learn that the concept is associated with 'glove'.

Our algorithm found an additional ten subsequences similar to the template. For six of these subsequences, only the RFID tag sensor labeled `glove` was on. Of the remaining four hits, just the `iron` was on for three times and just `fan` was on once. Thus, we end up with the probabilities shown in Figure 19.*right*.

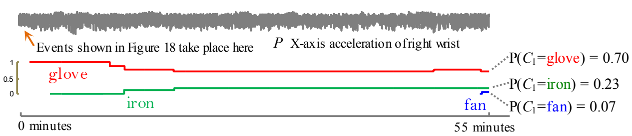

Figure 19: *top*) A zoom-out of the time series shown in Figure 18. *bottom*) The probability of concept C_1 being with various tagged items. Of 38 possibilities, only 3 have non zero entries.

In Figure 20, we show the relevant subsequences. Here, the true positives are subsequences that voted for `glove`, and the false positives voted for `iron` or `fan`. After a careful check of the original data we discovered that the pattern actually corresponds to *dishwashing*, which is the only behavior for which the participant wore `gloves`.

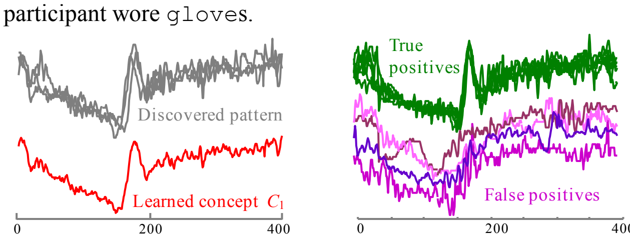

Figure 20: *top-left*) The motif discovered in our Elder care sensor experiment and averaged into concept C_1 (*bottom-left*). *right*) examples of true positives and false positives.

5. CONCLUSION AND FUTURE WORK

We have introduced the first never-ending framework for real valued time series streams. We have shown our system is scalable, able to handle 250Hz with ease (cf. Section 4.3), and that it is robust to significant noise (cf. Figure 17 and Figure 20). Moreover, by applying it to diverse domains, we have shown it is a very general and flexible framework. In future work, we hope to remove the few assumption/parameters we have and apply our ideas to year-plus length streams. We have made all our code and data freely available [13] and hope to see our work built upon and applied to an even richer set of domains.

6. ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support for our research provided by NSF IIS-1161997 and Vodafone.

7. REFERENCES

[1] E. Achtert, C. Bohm, H-P. Kriegel, P. Kröger. Online Hierarchical Clustering in a Data Warehouse Environment Data Mining. *ICDM*, 2005, pp.10–17.

[2] G. Batista, E. Keogh, A. Mafra-Neto, E. Rowton: Sensors and software to allow computational entomology, an emerging application of data mining. *KDD* 2011: 761-764.

[3] E. Berlin and K. Laerhoven. Detecting leisure activities with dense motif discovery. *Proceedings of the 2012 Intl Conference on Uniquitous Computing*. pp. 250-59.

[4] M. Borazio and K. Laerhoven. Combining Wearable and Environmental Sensing into an Unobtrusive Tool for Long-Term Sleep Studies". 2nd ACM SIGHIT 2012.

[5] A.S.L.O. Campanharo, M.I. Sirer, R.D. Malgren, F.M. Ramos, L.A.N Nunes. Duality between time series and networks. *Plos One*, 6 (2011), p. e23378.

[6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka Jr. and T.M. Mitchell. Toward an Architecture for Never-Ending Language Learning. *In Proc' AAAI*, 2010.

[7] G. Cormode, M. Hadjieleftheriou. Methods for finding frequent items in data streams. *VLDB* J. 19(1): 3-20 (2010).

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

[9] C. Elkan and K. Noto: Learning classifiers from only positive and unlabeled data. KDD 2008: pp213-220

[10] F. Ferreira, D. Bota, A. Bross, C. Mélot, J Vincent. Serial evaluation of the SOFA score to predict outcome in critically ill patients. JAMA 2001 Oct 10; 286(14):1754-8.

[11] A. Goldberger et al. *PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals*. Circulation 101(23):pp 215-20 2000.

[12] A. Goldberger et al. Physionet, accessed Feb-04-2013 *physionet.ph.biu.ac.il/physiobank/database/chfdb/*

[13] Y. Hao project webpage: *www.cs.ucr.edu/~yhao/NELTS.html*

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

[15] S. Jin, Z. Chen, E. Backus, X. Sun, B. Xiao. 2012. Characterization of EPG waveforms for the tea green leafhopper on tea plants and their correlation with stylet activities. Journal of Insect Physiology. 58:1235-1244.

[16] K. Jones (2012). *www.batdetective.org*

[17] L. Mitchell. Time Segregated Mosquito Collections with a CDC Miniature Light Trap. *Mosquito News* 1981. 42: 12.

[18] A. Mueen, E. J. Keogh, N. Young: Logical-shapelets: an expressive primitive for time series classification. *KDD* 2011: 1154-62.

[19] S. Nassar, J. Sander, C. Cheng: Incremental and Effective Data Summarization for Dynamic Hierarchical Clustering. *SIGMOD* Conference 2004: 467-478.

[20] B. Settles. Active Learning. *Morgan & Claypool*, 2012.

[21] M. Stikic, T. Huynh, K. V. Laerhoven and B. Schiele. ADL Recognition Based on the Combination of RFID and Accelerometer Sensing. PervasiveHealth, 2008. pp. 258–263.

[22] C. J. Van Rijsbergen, Information Retrieval, 2nd edition, London, England: Butterworths, 1979.

[23] D. Roggen et al. Collecting complex activity data sets in highly rich networked sensor environments, In Proc' 7th IEEE INSS (2010), pp. 233-240

[24] A. Veeraraghavan, R. Chellappa, A. K. Roy-Chowdhury, The Function Space of an Activity, *Computer Vision and Pattern Recognition*, 2006.