A General Framework for Never-Ending Learning from Time Series Streams

Yanping Chen, Yuan Hao, Thanawin Rakthanmanon[#], Jesin Zakaria, Bing Hu, Eamonn Keogh

Department of Computer Science& Engineering, [#]Kasetsart University

University of California, Riverside

{ychen053, yhao002}@cs.ucr.edu, thanawin.r@ku.ac.th, {jzaka001, bhu002, 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 that consider real-world problems in domains as diverse as medicine, entomology, wildlife monitoring, and human behavior analyses.

***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 [63]. 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 [11], 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 **Fig.1**, we show a time series produced by a light sensor at Soda Hall in Berkeley. 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.

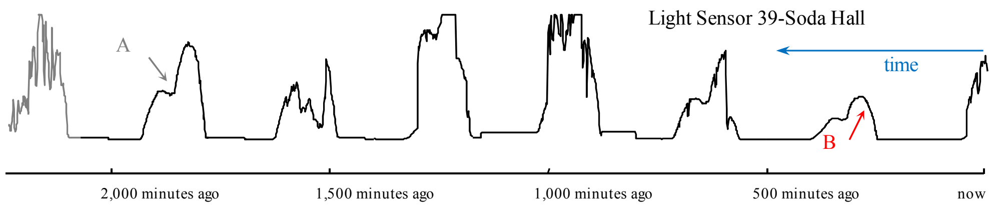


Fig.1 The light sensors at Soda Hall produce a never-ending time series, of which we can cache only a small subset in the main memory

As we can see in **Fig.2**a and **Fig.2**b, these patterns are even more similar after we z-normalize them [17]. Suppose that the appearance of these two similar patterns (or "motif") causes an agent to query a teacher as to their *meaning*.

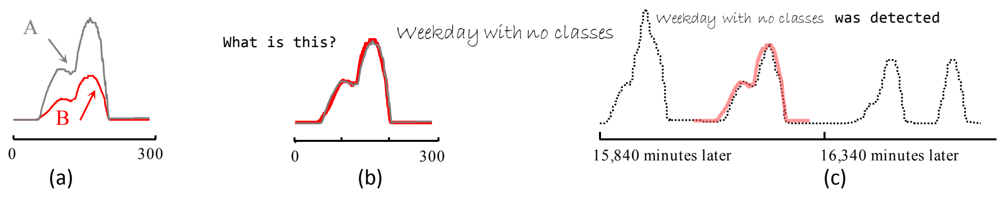


Fig.2 a) A "motif" of two patterns annotated in **Fig.1** aligned to highlight their similarity. b) We imagine asking a teacher for a label for the pattern. c) This allows us to detect and classify a new occurrence eleven days later

This query could be implemented in a number of ways; moreover, the teacher need not necessarily be human. Let us assume in this example 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 [10].

This example elucidates our aims, but suggests 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, time series classification [17], hierarchical clustering, crowdsourcing, active learning, and semi-supervised learning.

Frequent item mining is one of the most heavily studied problems in data mining. The task is to find all items whose frequency exceeds a specified threshold in the data stream [14]. Significant effort has been dedicated to this research topic in the past few decades, and most of the research community seems to have converged on a *counter-based* approach [34] [39][41]. Counter-based algorithms use a buffer to store pairs of items and counters, and when an item is seen, its corresponding counter is incremented. The buffer has a fixed size and only frequent items will be stored in the buffer when the algorithm terminates. Although the counter-based methods have good approximation in finding the frequent items, they are designed for *discrete* items only, not for mining the *real-value* items which our system is aiming at. The literature has offered some other methods for frequent items, such as the sketch-based methods which use bit-maps of counters to estimate the frequency for each item in the data [12][20][31], and quantile-based algorithms which find the frequent items through the quantile algorithm [14][56]. However, all the algorithms assume the items are *discrete* and countable, and thus are not readily applicable to our system.

Active learning is well-motivated by applications where unlabeled data are abundant and cheap, but acquiring their labels is expensive. The goal is to learn an accurate classifier with less training data by actively choosing the most informative unlabeled data to query [54]. Some of the active learning methods are *pool*-based, which assume that a large, static unlabeled dataset is available from the beginning and choose query data usually in a greedy fashion [58][55]. Other methods are *stream*-based, which acquire one item at a time, sequentially from the input, and make the decision of whether to query or discard the item on the fly. The stream-based methods are similar in spirit to our work since our system must also deal with data streams. In the literature, there are abundant stream-based active learning methods. The committee-based sampling method proposed in [15] constructs a 'committee' of classifiers and queries the items which are most disagreed upon by the committee members; In [62], the active learner queries the items that are most difficult to classify in order to revise the decision boundary; [22] proposes to query items which correctly disambiguate as much unseen data as possible. Although there are many active learning methods, most of them assume that all the classes to be learned are known before learning starts, and the goal is to find the query data that are "best" for improving the *accuracy* of the classifiers. In contrast, our system considers applications where we initially have only the vaguest understanding of the concepts to be learned (if any), and our goal is not only to learn *accurate* classifiers, but more importantly, to *discover* classes that might be amenable to being learned.

It would be remiss of us not to mention the groundbreaking NELL project led by Tom Mitchell at Carnegie Mellon [11], which is a key inspiration for the current work. Since early 2010, the CMU team has been running NELL 24 hours a day, sifting through hundreds of millions of web pages looking for connections between the information it already knows and what it finds through its search process. 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 through *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 [8]. 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 Time Series Classification Algorithm

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 [17][29][45]. 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) [17]. 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 [17][45], we use it in this work. However, nothing in our overarching architecture specifically precludes other distance 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

[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 [17].

example, in [9], 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 in our experimental section, **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 **Fig.1**.

- We assume we have access to a teacher (or *Oracle* [54]), and the cost of querying the teacher is both fixed and known in advance.

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 these definitions our problem can be defined as:

> **Problem Definition**: Given a stream *P*, which may be a proxy for a higher dimensional stream **S** that is recorded in parallel, and given access to a teacher, which may be a human or an algorithm, which can provide class labels for subsections of *P* (possibly by exploiting information available in the corresponding subsections of **S**), extract concepts from *P* and label them. The success at this task is measured by *coverage*, the number of concepts learned, and the average *accuracy* of each learned concept.

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 **Fig.3**, which attempts to divorce the domain-dependent and domain-independent elements.

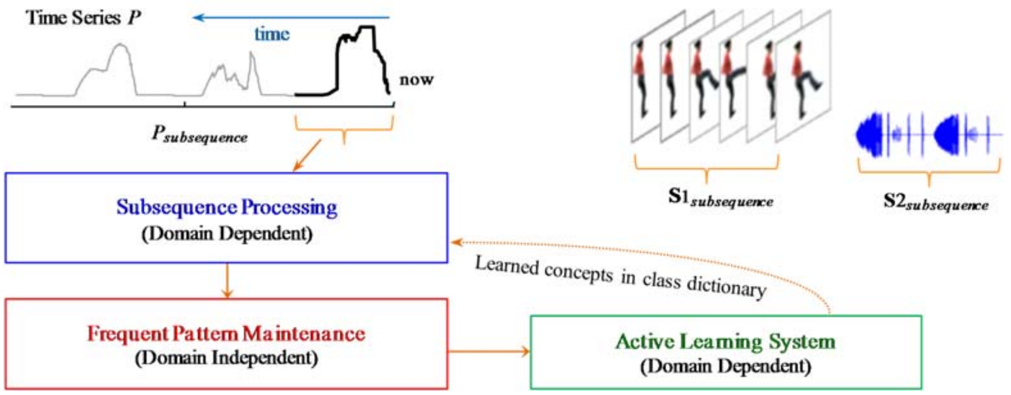


Fig.3 An overview of our system architecture. The time series P which is being processed may actually be a proxy for a more complex data source such as audio or video (*top right*)

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 **Fig.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 3.5. 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), and 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 3.4), 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)
dict ← initialize_class_dictionary
global dendro = create_random_dendrogram_of_size(w)
For ever
    sub  ← get_subsequence_from_P(S,P)
    sub  ← subsequence_processing(sub, dict)
    top  ← frequent_pattern_maintenance(sub)
    dict ← active_learning_system(top, dict)
End
```

In the next six subsections, we expand our discussion of the class dictionary and the four major modules introduced above. Then we will be in a position to discuss about performance of the proposed framework.

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 have seen sequences of this class. As shown in **Fig.4**b, a class dictionary is simply 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. **Fig.4**a illustrates the representational expressiveness of our model. Note that because a single class could be represented by two or more templates with different thresholds (i.e. *Weekend* in **Fig.4**b), 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 [29], at least in the case where both C_i and T_i are carefully set.

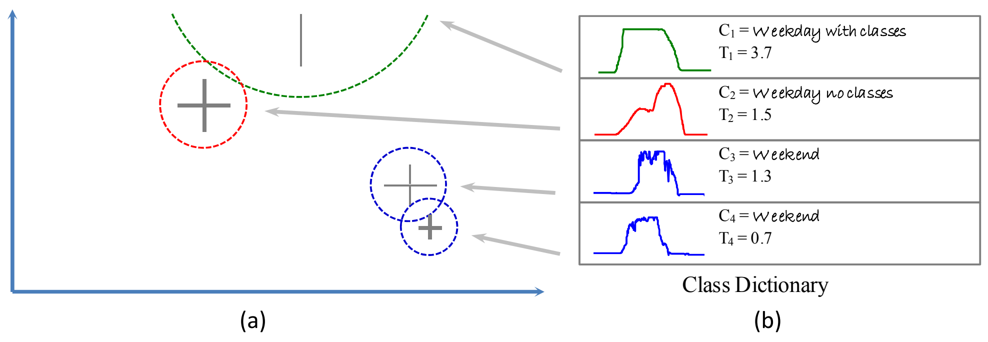


Fig.4 An illustration of the expressiveness of our model

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 [17], using logical connectives [45], etc.

As shown in **Table 1**-Line 1, our algorithm begins by initializing the class dictionary. In most cases it will be set to `empty`; however, in some cases, we may have some domain knowledge we wish to "prime" the system with. For example, as shown in **Fig.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.

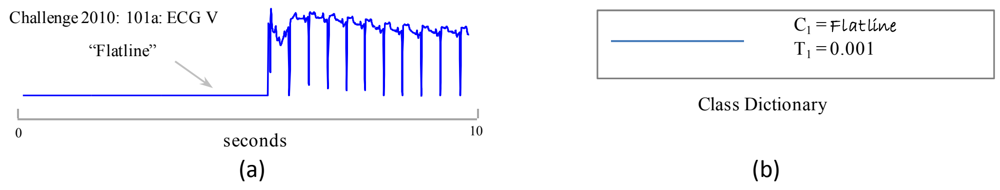


Fig.5 a) Sections of constant "flatline" signals are so common in medical domains that it is worth initializing the medical dictionaries with an example (b), thus suppressing the need to waste a query asking a teacher for a label for it

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 3.7.

3.3 Get Subsequence from Data Stream

All subsequences extracted from a stream are of the same length. In some cases where the starting and ending positions of each subsequence are known, we extract subsequences item by item. For example, the activity dataset in section 4.1 has the interesting property that every action takes exactly eight-seconds and one action starts immediately after another. Thus, we partition the stream at eight-second intervals, and taking each interval as a subsequence. In this case, each subsequence corresponds to a complete action.

In other cases where the starting and ending positions of the subsequences are unknown, we consider *every* possible subsequence (that is, we move the sliding window a single data point forward at a time), but skip those that may cause trivial matches [13]. Specifically, we begin by extracting the first available subsequence.

Then, given the next incoming subsequence, we compute its distance to the previously extracted subsequence. If the distance is smaller than a threshold, we discard the subsequence and move on to consider the next subsequence. This process continues until the incoming subsequence has a distance to the previously extracted subsequence larger than the threshold, in which case, we extract the incoming subsequence and pass it to the next module. A reasonable distance threshold is to discard approximately two thirds of all subsequences. The expected height of subtrees of size three learned from the "patternless data" that will be explained in section 3.5 turns out to be quite close to the reasonable threshold, and thus, is used for all experiments in this work.

3.4 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 **Fig.1** and **Fig.2** that z-normalization may be necessary [17]. 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 [6].

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_processing(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 on to the next step of frequent pattern maintenance, which *is* completely domain independent.

3.5 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 **Fig.1**/**Fig.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 [14]. 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 [14]) 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 [14]. However, most medical/human behavior domains have arrival rates that are rarely more than a few hundred Hertz [8][9][23][38][42][46]. 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 **Fig.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 **Fig.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.

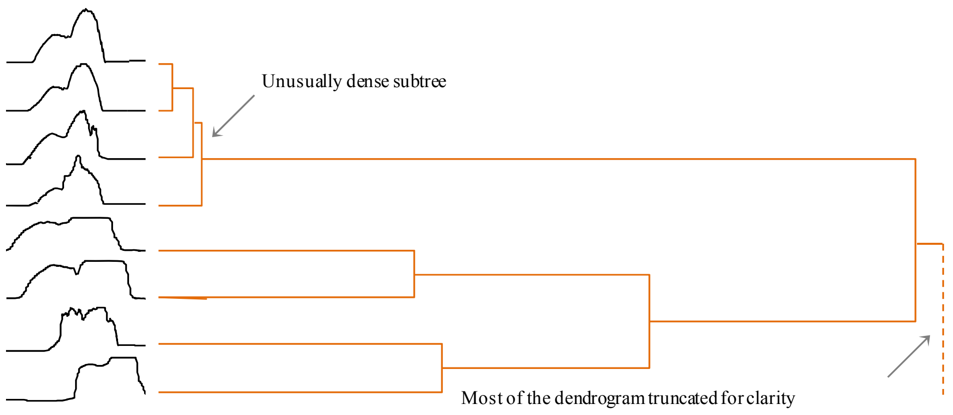


Fig.6 A visual intuition of our solution to the frequent time series subsequence problem. The elements in a dense subtree (or clade) can be seen as a frequent pattern

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 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 **Fig.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

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.

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 **Fig.7**, we record the mean and standard deviation of the heights of these subtrees.

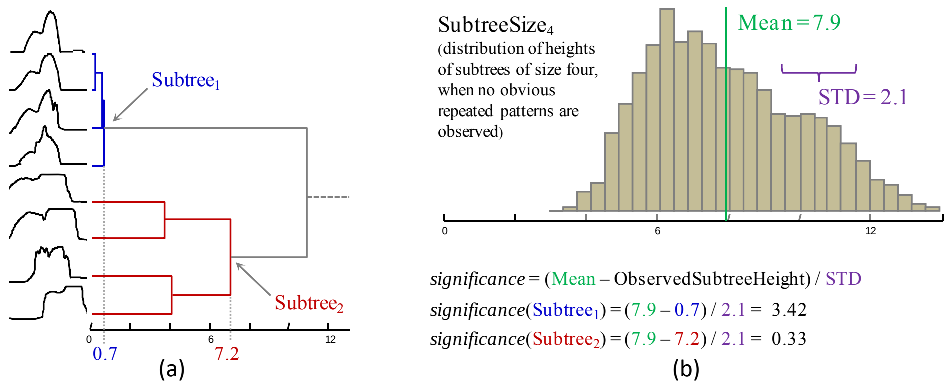


Fig.7 a) The (partial) dendrogram shown in **Fig.6** has its subtrees of size four ranked by density. b) The *observed* heights of the subtrees are compared to the *expected* heights given the assumption of no patterns in the data

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 of 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 **Fig.7**b, 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 **Fig.**7a. 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?

To see this, we can calculate the probability of discovering the pattern within a certain number of steps for different values of *w*. The probability is calculated using the equation below, where *n* is the number of steps.

$$Pr(w,n) = \left(\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{99}{100} & \frac{1}{100} & 0 \\ \frac{1}{w} * \frac{99}{100} & \frac{1}{w} * \frac{1}{100} + \frac{w-1}{w} * \frac{99}{100} & \frac{w-1}{w} * \frac{1}{100} \\ 0 & 0 & 1 \end{bmatrix}^{n} \right)(1,3)$$

The equation is derived using the Markov Chain model [48]. We relegate a detailed derivation of the equation to Appendix A, as it may not be of direct interest to the readers interested only in the practical applications of our ideas. To visualize how the buffer size *w* affects the mean time it takes to discover the pattern, we plot the results for selected values of *w* in **Fig.8**.

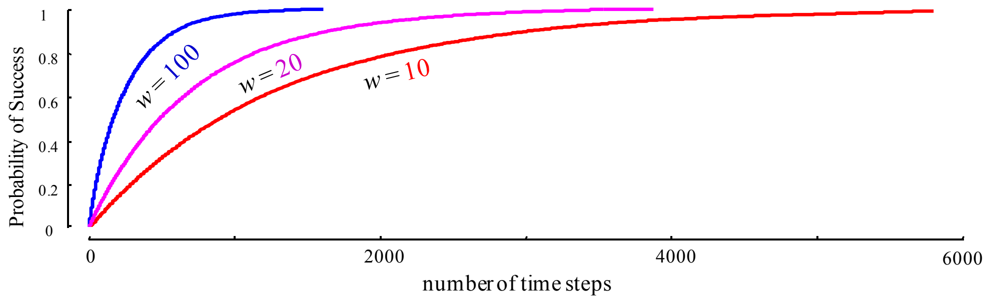


Fig.8 The average number of time steps required to find a repeated pattern with a desired probability for various values of *w*. All curves end when they reach 99.5%

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. For example, the "loss" or "gain" of an hour due to daylight savings time has long been known to produce strange patterns in many time series, including electrical demand streams [27]. Such rare patterns would clearly be difficult to discover. 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 [47] hierarchical clustering to bring the cost per time step down to O(w), and as we shall show pragmatically in our experimental section, our ideas are already scalable enough to apply to patterns that occur just a few times a day (c.f. Section 4.2).

3.6 Exploiting Temporal Locality for Frequent Pattern Maintenance

In the previous section, we proposed an algorithm to detect repeated time series patterns in an unbounded stream, even when the patterns are rare and far apart. The simple visual analysis in **Fig.8** suggests that we will eventually stumble on the pattern, and will do so more quickly if we can afford the space/computational requirements of a larger cache.

However, the algorithm proposed in **Table 3** assumes the worst case, that the patterns are distributed more or less randomly over time. More realistically, we may expect in many cases that repeated patterns have significant *temporal locality*. For example, a Chuck-will's-widow (*Antrostomus carolinensis*) is a nocturnal bird of the nightjar family *Caprimulgidae* [28]. Virtually all the male bird's vocalizations take place in a short period just after dusk (we consider never-ending learning of bird vocalizations in section 4.4). Similarly, consider **Fig.9** which shows the result of an experiment we performed to measure temporal locality in the behavior of a particular species of mosquito. We kept an optical flight sensor on for twenty-four hours and recorded every time a *Culex quinquefasciatus* mosquito flew past. We used this data to plot the daily flight activity (diel rhythm) of the mosquito. Again, note that there is extraordinary temporal locality in this stream, with a burst of activity just before dusk, and a secondary smaller burst that anticipates dawn.

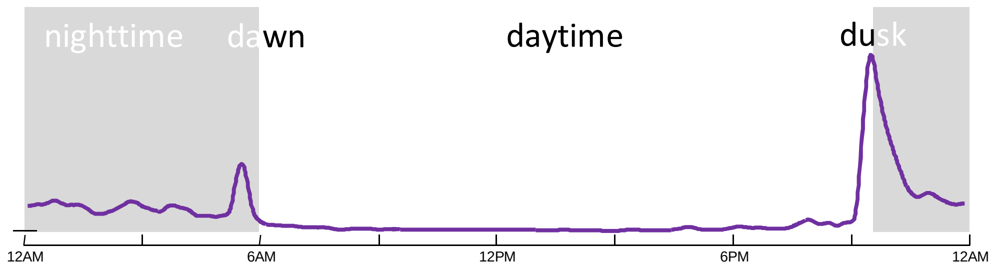


Fig.9 Daily activity plot for *Culex quinquefasciatus* mosquito. The probability of observing the mosquito flying is proportional to the height of the curve at that time. The curve is computed based on 12,782 observations of mosquito flight sounds.

We will further consider never-ending learning in the context of insect flight sounds in Section 4.2.

Finally, the temporal locality in *human* behaviors is almost too obvious to state [42]. Consider that most people spend only a tiny fraction of their lives shaking hands, but *when* they do it, they are very likely to do it again soon -- for example, as they are introduced to a group of people or as they socialize at a gathering, etc. Indeed, as Wu and colleagues recently noted, "*Recent evidence from various deliberate human activity patterns has shown that human activities* (have) *bursts of frequent actions separated by long periods of inactivity*" [61].

Given that the temporal locality of patterns is observed in many real-world problems [61], we are motivated to design an algorithm that exploits the temporal locality to maintain the frequent patterns. Our motivation is simply to produce an algorithm that can learn *faster* in some domains; however, for clarity, note that we are not changing the *space* of concepts we can learn.

Note that this temporal locality exploiting framework can be seamlessly "plugged" into the framework outlined in **Table 1**. Recall that the topology of the global dendrogram is not important; it just provides a way to think about dense patterns (observed sub-trees), relative to expectations (sub-trees observed in the training data). Here we will limit our consideration of dense patterns to *pairs* that occur in a relatively short time window, and again normalize our expectations based on training data.

To exploit temporal locality, we use the online discovery and maintenance of time series motifs algorithm of Mueen and Keogh [46] to detect the repeated patterns, and we refer the interested reader to that paper for full details. A circular buffer of size *w* is maintained. At each time step, a new subsequence arrives and the oldest subsequence in the buffer is discarded to maintain constant space, so the buffer always keeps the most recent data. At each time step, the algorithm maintains the closest pair of subsequences in the buffer, and only this motif pair will be made visible to the active learning module discussed below.

We must calculate the significance of this motif pair to see if it is worth polling a teacher. The way we do this is similar to the case discussed in the previous section. Offline, we take a sample of time series from the domain in question and remove existing patterns to create a “patternless” data. We run the algorithm in [46] on this data with the same parameters we intend to monitor P under, and as shown in **Fig.10**b, we record the mean and standard deviation of the distances of all the motif pairs we discover.

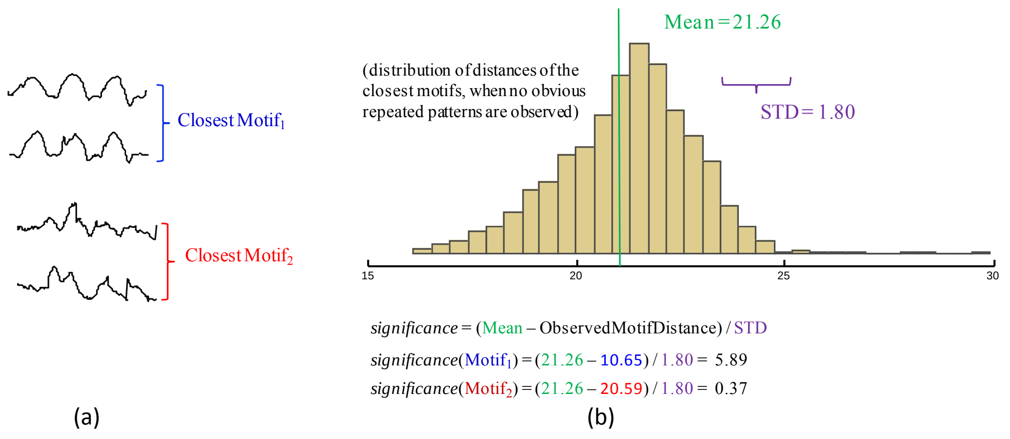


Fig.10 a) Two motif pairs discovered in the bird song data (c.f. Section 4.4). b) The *observed* distances of the motifs are compared to the *expected* distances given the assumption of no patterns in the data.

The distribution tells what we should expect to see if there is no frequent pattern in P, as frequent patterns will show up with an unusually small distance relative to this distribution. In order to be consistent with the significance definition in the previous section, we define the *significance* of a motif pair as the number of standard deviations less than the mean is the distance of the closest motif. Thus, the *significance* of a motif pair $motif_i$ is:

$$significance(motif_i) = \frac{mean - motif_i.distance}{STD}$$

In **Table** 4, we illustrate a high level overview of the algorithm.

Table 4 Frequent Pattern Maintenance that Exploits Temporal Locality Algorithm

```
Algorithm:top = frequent_pattern_maintenance_TemporalLocality(sub)

1   if sub == null                   // If null was passed in,
2       top ← null; return;          // do nothing, return null
```

```
else
   buffer ← insert(buffer,sub) // |buffer| is now w + 1
   buffer ← delete_the_oldest_item(buffer)// back to size w
   top ← find_most_significant_motif(buffer)
end
```

As before, this frequent pattern mining algorithm has two parameters, the buffer size *w* and the motif length *l*. The larger value *w* is, the better we expected it to be at mining frequent patterns. Empirically we have found that the value of *w* offers diminishing returns. A small *w* is incapable of discovering patterns unless we are lucky enough to encounter two adjoining instances. Making *w* just twice as large offers a significant improvement, but making *w* massively larger often makes no perceptible improvement. Identifying the exact motif length *l* is usually difficult. Fortunately, the performance of the algorithm is insensitive to the motif length as long as it is within a reasonable range of the ideal motif length (c.f. Section 4.4).

3.7 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 the inputs of a crowdsourcing algorithm or a classification algorithm that makes errors but performs above the default rate.

The ability of our frequent pattern maintenance algorithms 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 [54]. 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

(**Table 3**) or buffer (**Table 4**), *and* we also see a continuous never-ending 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 works.

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

3.7.1 When to Trigger Queries

Different assumptions about the teacher model and its associated costs can lead to different triggering mechanisms [54]. However, most frameworks can reduce to questions of how frequently we should ask questions. A conservative questioner that rarely asks questions may miss opportunities to learn new 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 a recent increase in her SOFA score [21]. Alternatively, for a weak learner, we may have a budget which we can use to help decide when to query. For example, several machine learning projects are built on top of Google's prediction API. At the time of writing a system could only get 100 free predictions per day [25].

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. For the temporal locality case, QR is the mean number of seconds between queries of *false motifs* 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 [64]. This equation assumes that the distributions of heights of subtrees (e.g., **Fig.7**b) and the distributions of distances of the closest motifs (e.g., **Fig.10**b) are approximately Gaussian, a reasonable assumption in most domains, as can be seen from the relevant figures.

3.7.2 Learning a Concept: Strong Teacher Case

In **Table 5**, the active learning system begins by comparing the *significance* (c.f. Section 3.5) 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 physical 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 [19]; 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 for the domains we investigated.

Table 5 The Active Learning Algorithm

```
Algorithm:dict = active_learning_system(top,dict)
1  if (significance(top) < trigger threshold) // The subtree
2       dict ← dict; return;    // is not 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,dict)
8  end
```

3.7.3 Learning a Concept: The Weak Teacher Case

A weak teacher can leverage off "side" information to label the discovered patterns. It assumes a query will be answered with a set of probabilities for the label, and thus differs from the strong teacher case where the pattern's label is a determined value. An example of a weak teacher's label is shown in **Fig.11**c right, where $P(C_i = \text{label}_j)$ denotes the probability that concept i should be labeled as label_j.

Table 6 outlines the algorithm to label a frequent pattern with a weak teacher. When a significant pattern is first discovered, the pattern is saved into the dictionary without a label (Line 1). The weak teacher waits for future occurrences of the pattern to be observed, and then labels the pattern. The label of a pattern is computed based on the votes for each class. A class gets one vote if it is associated with an observed occurrence. By associated we simply mean that the relevant flag is "on" at the same time as the pattern is observed. An observed occurrence may have multiple classes associated with it, and thus have multiple classes get one vote each (Line 7-9). The number of votes a class gets equals the cumulative number of observations with which it is associated. The probability that a pattern belongs to a certain class, and thus should be labeled with the class label, is simply the fraction of votes the class gets over all the votes (Line 10-12).

Table 6 The Weak Teacher Algorithm

```
Algorithm:spawn_weak_learner_agent(top,dict)
1   [dict,index] ← update_dictionary(dict,top,[])
2   votes = zeros(number of classes,1);
3   class_prob = zeros(number of classes,1);
4   For ever
5     if(count$_{index}$ increases)          // an occurrence detected
6       subClasses = classes associated with this occurrence;
7       For each class (class$_k$)in subClasses
8         votes[class$_k$] ← votes[class$_k$] + 1;
9       end
10      For (j = 1 to number of classes)   //update probability
11        class_prob[j] =votes[class$_j$]/sum(votes);
12      end
13      dict ← update_dictionary_labels(dict,index,class_prob);
14    end
15  end
```

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 **Fig.11**a, 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 that 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 **Fig.11**b, immediately polls the binary sensors for clues as to C_1's label.

In the example shown in **Fig.11**c, after the first detection of C_1, we have one vote for `Axe`, one for `Cat`, and zero votes 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).

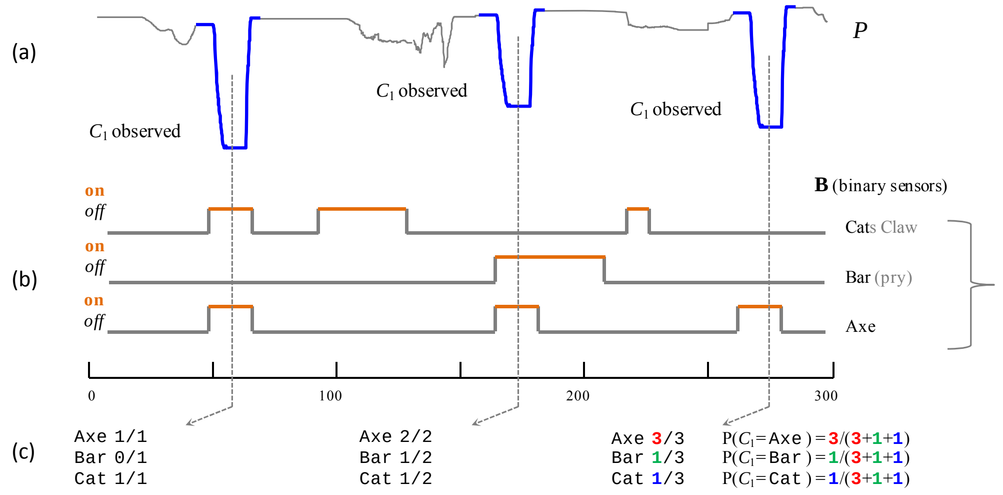


Fig.11 An illustration of a weak teacher. a) A stream *P* in which we detect three occurrences of the pattern C_1. b) At the time of detection we poll a set of binary sensors to see which of them are active. c) We can use the frequency of associations between a pattern and binary "votes" to calculate probabilities for C_1's class label (A **cat'**s paw is an informal name for a nail puller)

This simple weak teaching scheme is the one we use in this work and we empirically evaluate it in Section 4.6 and Section 4.7. 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 real-valued, but treated as binary when polled by our algorithm; 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*}.

3.8 Algorithm Performance

In this section, we will discuss the space and time requirements of our algorithm.

3.8.1 Memory Requirement

The memory requirements includes storage for the dendrogram, the dictionary, and the algorithm overhead. If l is the length of a subsequence, w is the size of the dendrogram, N is the total number of concepts in the stream, the total storage required is $wl + N(l+2) + c$, where c is a small constant for the overhead, such as storage for parameters and for buffering a subsequence as it is streaming by. Note that w is usually a small number typically no more than a thousand. This is because a too large w will slow down the algorithm while offering little improvement at mining the concepts. This “diminishing returns” property we have shown in **Fig.8** and Appendix A. N is the largest possible size of the dictionary. Although in principle a dictionary can grow arbitrarily large, in most real-world applications, the number of concepts in a stream is limited and quite small. For example, there are at most a few dozens of heartbeat classes recognized, and we are unlikely to encounter more than a few hundred species of flying insects at any one location. Even if we imagined our system being used to learn the entire vocabulary of the full Harry Potter series (cf. Section 4.7), this is about 13,000 words, and could easily reside in the memory of a desktop with 4GB memory. For all the experiments used in this work, the dictionary learned is never larger than ten mega-bytes. Therefore, the memory requirements of the algorithm are inconsequential.

3.8.2 Time Complexity

We first consider the time the algorithm takes to process a subsequence. In the worst case, a subsequence goes through all the components in the framework, including subsequence extraction, subsequence processing, frequent pattern maintenance and active learning. The running time for a subsequence extraction is $O(l)$, which is used to calculate the ED distance of the subsequence to the previously extracted one. The subsequence processing includes a domain-dependent pre-processing procedure and a classification. The running time for the pre-processing depends on the procedure used. In the case of z-normalization, which is the most common processing procedure for time series data, it takes time $O(l)$. The classification of a subsequence involves comparing the subsequence to all the concepts in the dictionary to find the nearest neighbor, and thus, takes time $O(Nl)$. When a subsequence is not classified by the dictionary, it enters the frequent pattern maintenance component. Inserting a subsequence into the dendrogram takes time $O(w^3 + wl)$, where $O(wl)$ is used for updating the distance matrix and $O(w^3)$ for re-clustering. The number of subtrees in the dedrogram is bounded by the dendrogram size w, and thus finding the most significant subtree takes time $O(w)$. In the case where a query is triggered, it takes time $O(1)$ to send the patterns to a teacher and time $O(l)$ to add a new concept to the dictionary. Summarizing the above, the running time for a subsequence that goes through all the components of the framework is $O(w^3 + wl + Nl)$.

In the worst case, every subsequence in the stream goes through all the components in the framework. In that case, to process k subsequences, it takes time $O(k * (w^3 + wl + Nl))$. Note that as we discussed before, w and N are typically small numbers, and thus, our algorithm is fast. In all experiments in this work, the algorithm is *faster than real-time* on a standard machine with 8GB RAM and 2GHz Intel Core processor.

In the cases where we have to deal with higher arrival rate, the bottleneck step is the update of the dendrogram after insertion of a new subsequence. Fortunately, we can speed up this process from $O(w^3)$ to $O(w \log^3 w)$ using the method proposed in [37]. This will reduce the total time complexity to $O(w \log^3 w + wl + Nl)$. Note that the classification step can also be a bottleneck. This happens when N is very large. However in the literature there are many techniques designed to speed up the search of nearest neighbor in large datasets. For example, in [51], the search of a nearest neighbor in a trillion ECG dataset takes only 18 minutes. By applying such techniques, our algorithm can be made much faster. We have not availed of such speed up techniques in this work, because once our algorithm is real-time, we have little motivation to make it faster.

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 [64]. 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.

For simplicity, we use the frequent pattern mining algorithm in **Table 3** for all but one experiment. That one exception is the experiment in Section 4.4, which considers bird calls. As noted in [28] and [49], bird vocalizations are domains where we should expect significant temporal locality; thus for this domain we use the algorithm outlined in **Table 4**.

We do not consider the effect of varying *w* on our results. As noted in Section 3.5, once it is set to a reasonably large 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 the algorithm in **Table 3** 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 (detection rate) as well as the average precision and recall [59]. 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. All the datatsets used in the experiments are summarized in **Table 7**.

Table 7 Datasets Used in the Experiments

Dataset Name	Size	Duration	Dimension	# Classes
Activity	61 KB	13.3 minutes	1	10
Flying Insect	15 GB	2 days	1	3
ECG	18 MB	20 hours	1	5
Bird Song	11 MB	6 minutes	1	3
Insect EPG	4 MB	27 minutes	1	~15
Elder Care	9 MB	240 minutes	41	dozens
Audio Books	5 MB	142 minutes	2	hundreds
Energy Usage	9 MB	2 months	1	dozens

4.1 Activity Data

We begin with a relatively small dataset, the activity dataset of [60]. This experiment may be regarded as more of a *demonstration* than a challenging *test* of our system; however, it is a visually intuitive problem and this will help the reader appreciate the strengths of our framework.

This dataset consists of a 13.3 minute 10-fps video sequence (thresholded to binary pixel values 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. **Fig.12**a shows the first query shown to the teacher on the first run.

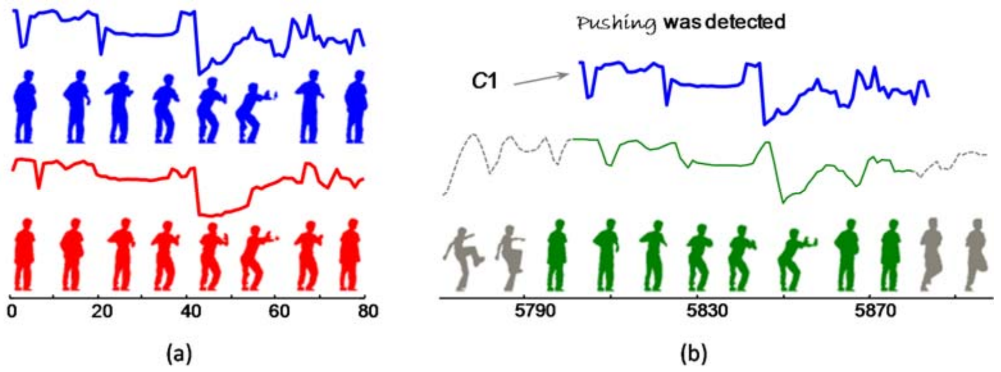


Fig.12 a) A query shown to the user during a run on the activity dataset; the teacher labeled it *Pushing* and a new concept C_1 was added to the dictionary. b) About 9.6 minutes later, the classifier detected a new example of the class

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 **Fig.12**b.

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 41.82% 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. Table 8 shows the complete results.

Table 8 Results on Activity Dataset

Class	Detection Rate	Precision	Recall
Picking up	96%	0.184	0.576
Running	100%	0.418	0.880
Pushing	98%	0.191	0.532
Squating	92%	0.193	0.554
Hand waving	98%	0.193	0.561
Kicking	90%	0.189	0.485
Bending	88%	0.208	0.524
Throwing	95%	0.184	0.554
Turning around	84%	0.205	0.526
Smoking	97%	0.199	0.510

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 [6]. **Fig.13** demonstrates the practicality of this idea.

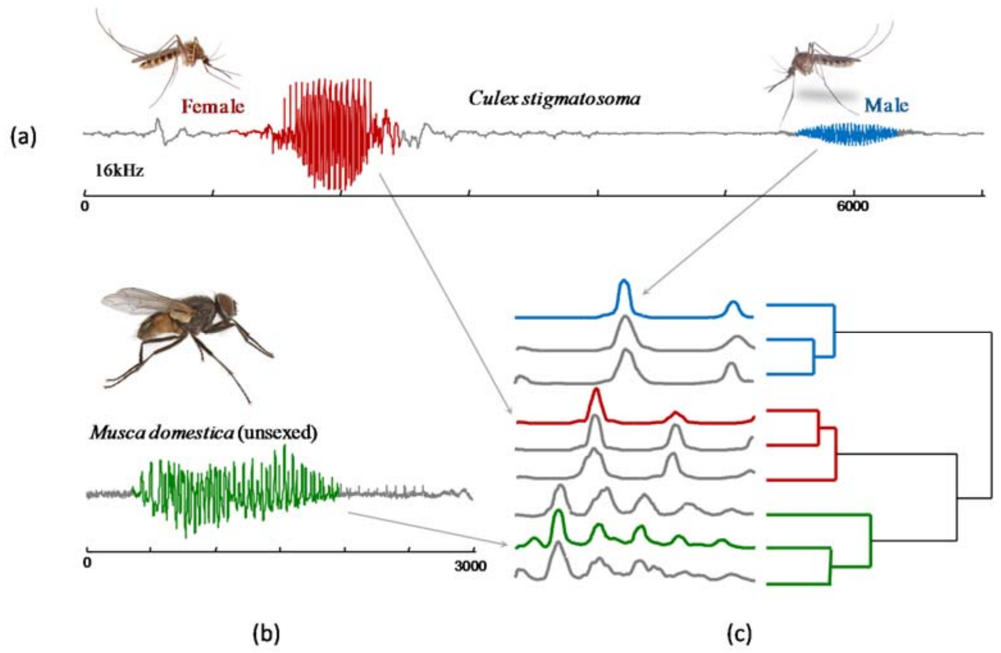


Fig.13 a) An audio snippet of a female *Cx. stigmatosoma* pursued by a male. b) An audio snippet of a common house fly. c) If we convert these sound snippets into periodograms we can cluster and classify the insects

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

This allows us to classify *known* species, such as 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 [44]; can we use our framework to detect and classify them? At first blush, this does 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.

Our solution is to exploit the fact that some insect traps can *physically* capture the flying insects themselves and record their time of capture [43]. Thus, the **S** data source consists of 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: *Culexstigmatosoma* male (C*stig* ♂) and female (C*stig* ♀). These concepts are just the periodograms shown in **Fig.13** 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 could not detect the invasive species, we increased their number for the next day, and tried again until we did detect 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 9** shows the results.

Table 9 Our Algorithm's Ability to Detect and 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 [23][24]. This record has 17,998,834 data points containing 92,584 heartbeats. As shown in **Table 10**, the heartbeats have been independently classified into five types.

Table 10 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 **Fig.14**, we can see this data has both intermittent noise and a wandering baseline; we did *not* attempt to remove either.

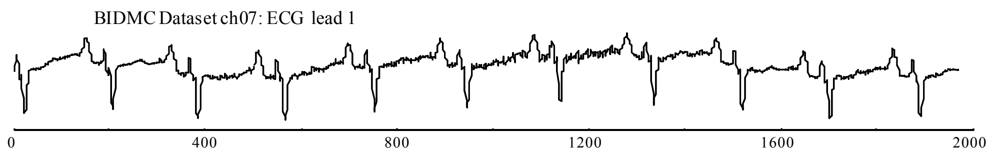


Fig.14 A small snippet (0.0065%) of **BIDMC$_{ch}$07 Lead 1**

Let us consider a single test run. After 45 seconds, the system asked for a label for the pattern shown in **Fig.15**a. Our teacher, Dr. Criley[4], gave the label Normal(N).

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

Just two minutes later, the system asked for a label for the pattern shown in **Fig.15**b; here, Dr. Criley annotated the pattern as R-on-T PVC (r).

These two requests happened so quickly that the attending physician who hooked up the ECG apparatus would be in the same room and able to answer the queries directly. The next request for a label did not occur for another 9.5 hours, and we envisioned it being sent by email to the teacher. As shown in **Fig.15**c, our teacher labeled it PVC (V).

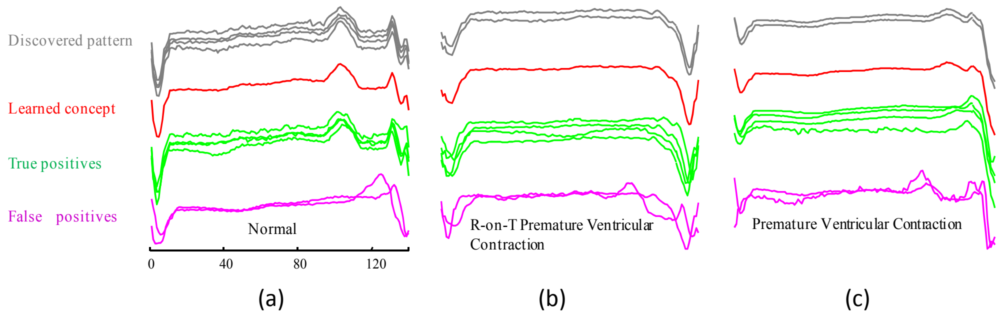


Fig.15 Three patterns discovered in our ECG experiment. From top to bottom, they are the motif discovered and used to query the teacher; the learned concept; some examples of true positives; some examples of false positives

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 11**. 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, [24] notes the data was "*prepared using an automated detector and has not been corrected manually*."

Table 11 Results on **BIDMCch07**

Class	**Detection Rate**	**Precision**	**Recall**
Normal (N)	100%	0.997	0.994
R-on-T PVC (r)	100%	0.914	0.808
Supraventricular (S)	100%	0.502	0.414
PVC (V)	100%	0.234	0.677
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 [24], the algorithm asked for a label for the extraordinary repeated pattern shown in **Fig.16**.

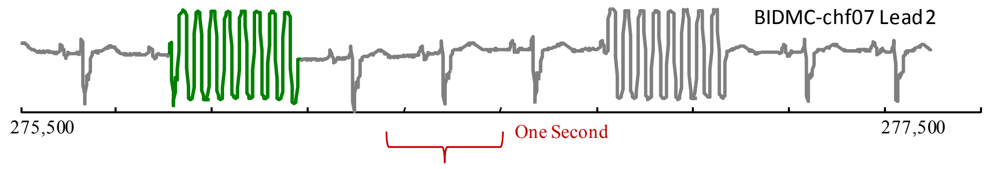


Fig.16 A pattern (green/bold) shown with surrounding data for context, discovered in lead 2 of **BIDMC$_{ch}$07**

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: Exploiting Temporal Locality

Bird monitoring has played an important role in conservation planning and wildlife management, providing essential information in understanding birds' habits, diversity, population size and trend [2][38]. Most bird monitoring systems are acoustic [38] and monitoring data can quickly run into many terabytes. Automatic detection and classification of the bird calls can greatly decrease the cost of performing biodiversity studies [16]. In this experiment, we wish to see if our framework is capable of detecting and classifying bird calls without the need for extensive preliminary setup work, including the manual searching of audio archives and editing of classification templates [4].

When a bird calls, it typically repeats the call over and over again. This gives a potential mate an opportunity to hear the call, and then converge on the singer's location. Thus, we expect significant temporal locality of bird calls from the monitoring audio [49]. In order to produce a large dataset of bird sounds for which we had ground truth, we did the following. We recorded three hours of ambient sound on January 18, 2013. A careful human annotation of the sound file revealed no obvious avian calls. Using data from xeno-canto.org, we embedded a sixty second long bird sound snippet after each hour in the data. Note that the sixty second snippets of bird sounds we inserted had *natural* temporal locality. We did not edit these sounds in any way.

We have in total three sound snippets, created by three different species of birds: the Long-billed Woodcreeper (*Nasica longirostris*), the Cinnamon-vented Piha (*Lipaugus lanioides*) and the White-tailed Hawk (*Geranoaetus albicaudatus*). Each bird sound snippet contains five examples of the bird's call. Using the raw audio as *S*, and a single 100Hz Mel-Frequency Cepstral Coefficient (MFCC) as *P*, we ran our algorithm that exploits temporal locality (c.f. Section 3.5) on this data. As **Fig.17** shows, our system can easily discover the patterns.

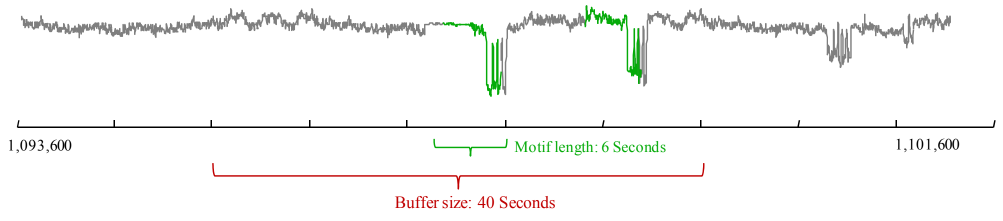


Fig.17 Two repeated patterns (green/**bold**) shown with surrounding data for context, discovered in the bird dataset, with the buffer size *w* set to 40 seconds, and motif length *l* set to 6 seconds. **Fig.18**b shows a zoom-in of the two patterns, which come from a Cinnamon-vented Piha

As shown in **Fig.18**, our system has discovered three different motifs/concepts, each belonging to a different species of bird. The audio snippets corresponding to the motifs may be heard at [64]. They are easily identifiable as three different bird calls.

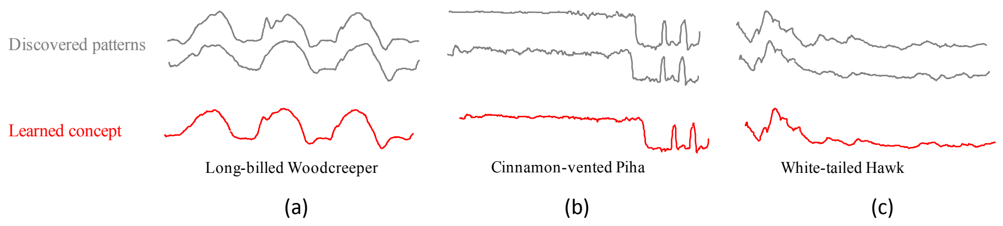


Fig.18 Three motifs discovered in the bird song experiment. From top to bottom, they are the motif discovered and used to query the teacher; the learned concept.

As shown in the bottom-row of **Fig.18**, for each motif pair, we created a concept by simply averaging the subsequences. To test the accuracy of the classifiers, we created a test dataset, which is 90 seconds long, containing seven bird calls, three from the *Long-billed Woodcreeper*, two from the *Cinnamon-vented Piha* and two from the *White-tailed Hawk*, and the remainder from randomly chosen species. **Table 12** shows the accuracy of the three classifiers on the test dataset.

Table 12 Classification Results on the Test Dataset

Class	Precision	Recall
Long-billed Woodcreeper	1.00	0.667
Cinnamon-vented Piha	1.00	1.00
White-tailed Hawk	0.667	1.00

In this experiment, we set the motif length to be 6 seconds long. To see how sensitive our algorithm is to the motif length, we varied the motif length between five seconds and seven seconds. In both cases, our framework found the same three motifs as those found by setting the motif length to be six seconds. Thus, our algorithm does not require the motif length to be precisely set in order to detect repeated patterns.

Note that in most of the experiments shown in this paper, we build the classifiers based on the *proxy* data from stream *P*. However, there is nothing in our framework that requires this. The classifiers can be built using the *raw* data from stream *S*, or some proxy data other than *P*, depending on the applications. For example, in the bird song dataset, we can use the *spectrogram* of the motifs to build the classifiers if the spectrogram is better at classifying the bird calls than the MFCC subsequences are, while still using a single MFCC as the *proxy* stream *P* for motif detection.

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 they have been increasingly using a tool called an Electrical Penetration Graph (EPG), which, as shown in **Fig.19**, adds the insect to an electrical circuit and measures the minuscule changes in voltage that occur as the insect feeds [32].

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 the 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 learned from one species to match even a closely related species.

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.

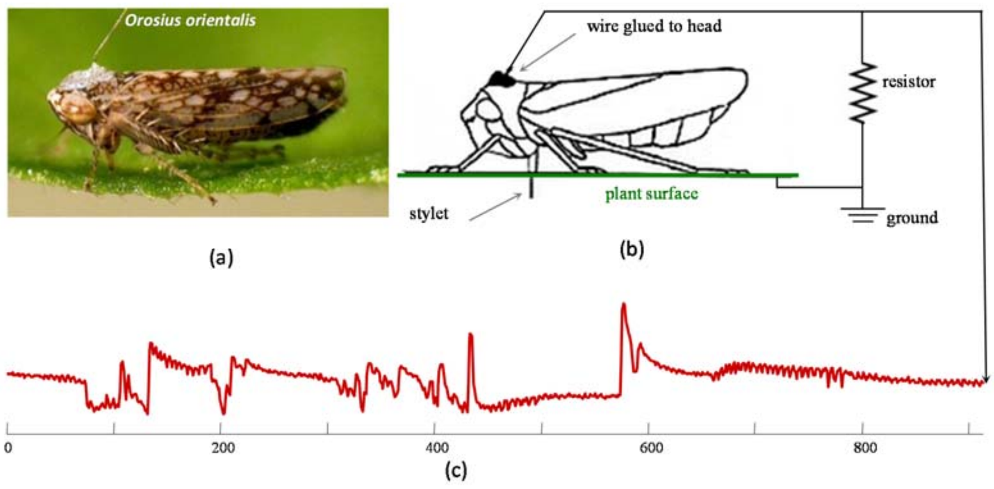


Fig.19 a) A tethered brown leafhopper. b) A schematic diagram of the circuit for recording EPGs. c) A snippet of data produced during one of our experiments

Let us consider a typical run on a dataset consisting of a Beet Leafhopper (*Circulifertenellus*) 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 **Fig.20**a. Dr. Backus labeled the pattern: phloem ingestion with interruption for salivation. After 13.2 minutes, the system requested a label for behavior shown in **Fig.20**b. 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 **Fig.20**c.

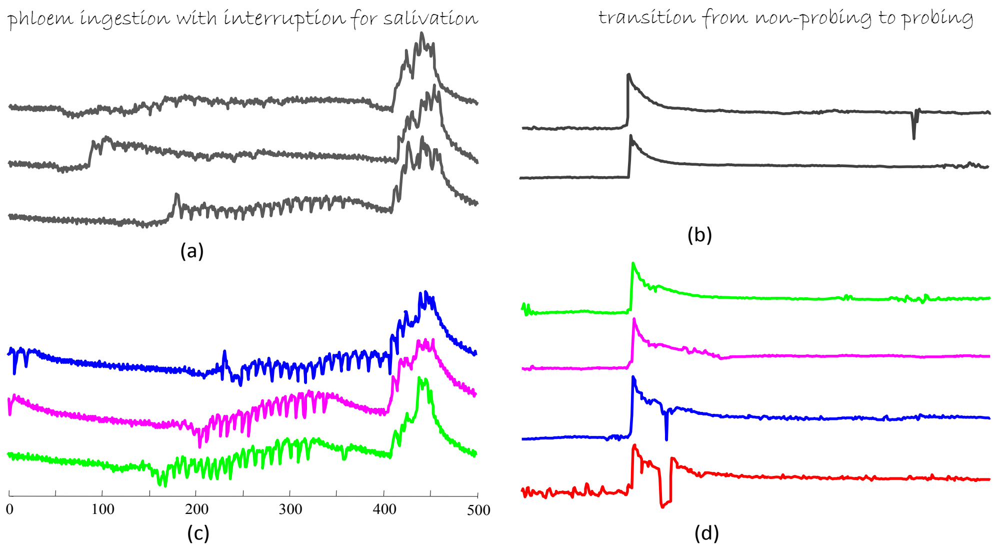


Fig.20 a,b) The two concepts discovered in the EPG data. c,d) Examples of classified patterns

A careful retrospective study of this dataset tells us that we had perfect precision and recall on this run (we have access to a time-stamped video recording of the insects, and the expertise of Dr. Greg Walker to label the data). Other runs on different datasets in this domain had similar success. We defer the detailed results to [64] for brevity.

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 the effective and unobtrusive long-term monitoring and recognition of daily living activities [57][50]. However, labeling accelerometer and sensor data is still a great challenge and requires significant human intervention. In [50], 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 [57] 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 a multidimensional time series with three real-valued and thirty-eight 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 (**Fig.23**a). As we can see in **Fig.21,** 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 currently *on*.

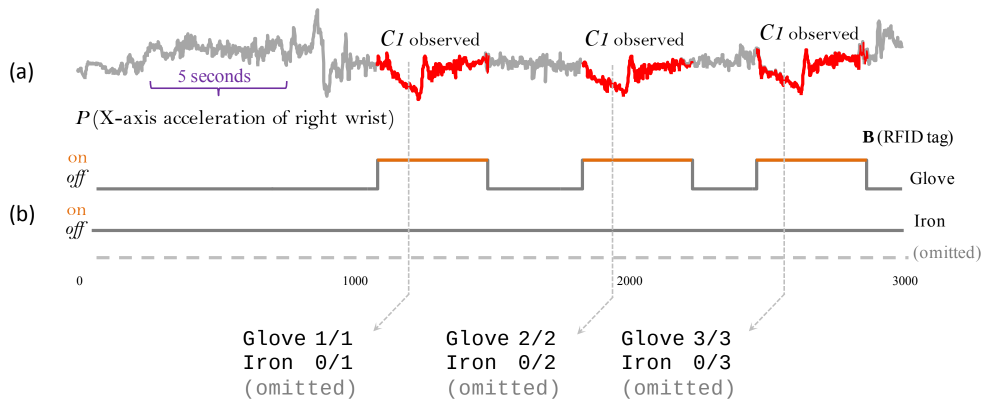


Fig.21 a) After we have learned the concept C_1, our system monitors for future occurrences of it. Here, it sees three examples in a row. b) By polling the binary RFID sensors when a "hit" for C_1 is detected, we can learn that the concept is associated with 'glove'

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 three times and *just* the `fan` was on once. Thus, we end up with the probabilities shown in **Fig.22**b.

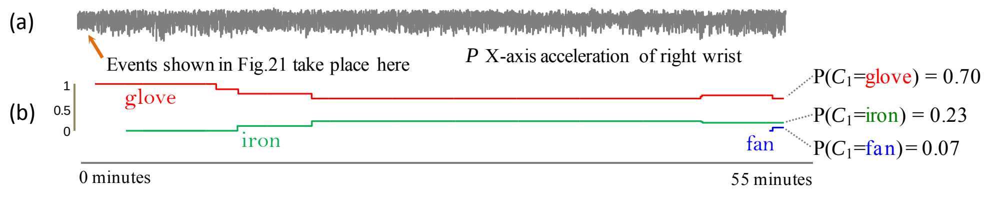


Fig.22 a) A zoom-out of the time series shown in **Fig.21**. b) The probability of concept C_1 being with various tagged items. Of 38 possibilities, only three, `glove`, `iron` and `fan`, have non-zero entries. Note that `fan` had a zero probability until the last few minutes

In **Fig.23**, 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`.

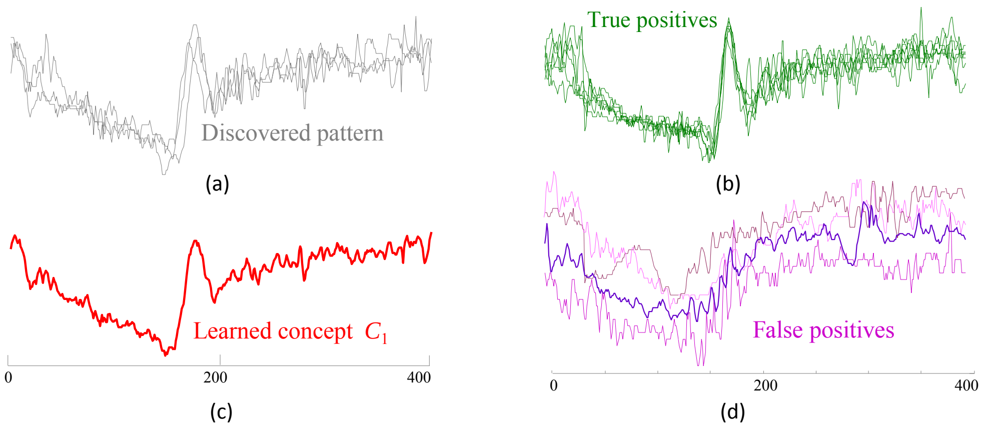


Fig.23 a) The motif discovered in our EPG experiment and averaged into concept C_1 (c). b,d) Examples of true positives and false positives

4.7 Weak Teaching Example: Audio Books

Audio books have changed the way many people consume books. In most audio books, we naturally expect to encounter many frequently repeated words. For example, in the *Harry Potter* audio books, the words *Harry, Hermione* and *magic* occur many times. Note, however, that even within a single audio book, different occurrences of the same word may sound different as the performer wants to distinguish between characters by using different 'voices'. Moreover, two different words may sound similar as they have similar pronunciations, like *Harry* and *hurry*. This makes the discovery and classification of the repeated words challenging, and makes audio books an interesting domain to test our framework.

An audio book usually has a corresponding text. In most cases, the audio and the text are only weakly aligned in time, so it is difficult to associate exactly the correct text with the corresponding audio. For example, if a spoken *hapax legomenon* appears in the 565th second of a three-hour recording, this would suggest that the ASCII version should appear at 5.231% of the way into the corresponding text. However, because a speaker speaks at a non-constant speed, we need to bracket the location where we expect to find the corresponding word, perhaps somewhere in the range {4% – 6%}. This is what the parameter R indicates in **Fig.24**a. Once we understand this minor issue, it is clear that the ASCII text can help to identify a *set* of possible words that corresponds to the repeated audio found by our algorithm, and thus can be used as a weak teacher.

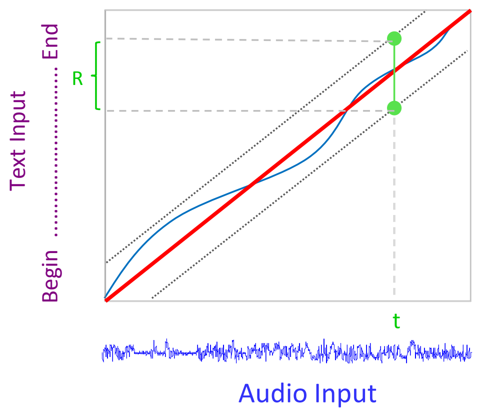
For example, suppose that one example of a motif pair corresponded to the ASCII "*While shopping for* ***school*** *supplies with the* ***Weasleys***...", and the second example corresponded to "*...when the older* ***Weasleys*** *arrived at* ***school****, they....*"

While the uncertainty of the temporal alignment means we do not know which word corresponds to our motif, words that appear in *both* text snippets are much more likely candidates. Thus, here we have reason to suspect the motif corresponds to either "*Weasleys*" or "*school*". By monitoring for further occurrences of the motif and checking the ASCII snippets, we can update our initial 50/50 guess in a principled Bayesian manner.

Expanding on this intuition, we conducted a simple experiment to see how well our framework can classify repeated words from an audio stream in the weak teacher mode. Note that we are not claiming this domain is in *need* of a sound-to-text alignment mechanism, or that if it was, our never-ending learning would be an especially fruitful method. Our point is simply to give a demonstration of weak teacher learning in a domain for which ground truth is available. Moreover, we are deliberately making this problem harder by considering only a *single* Mel-Frequency Cepstral Coefficient (MFCC) [40], whereas most speech processing algorithms use twelve.

In this experiment, we consider the audio book *Harry Potter and the Chamber of Secrets* as performed by Stephen Fry [52]. The raw audio *S* is first transformed into the 2nd MFCC coefficient[5], which is our proxy *P* for the high-dimensional data [40]. In **Fig.24**b we show three examples of MFCC time series corresponding to the word "*Harry*". The reader can appreciate that they have significant variance.

[5]Usually the top thirteen coefficients are used for audio analysis. The first coefficient is a normalized energy parameter, which is not used for speech recognition [40]



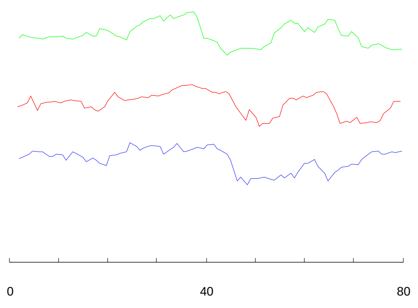


Fig.24 a) The audio and the text are assumed to be approximately linearly aligned, as shown by the red line. However, due to the variability of the actor's performance, the alignment actually wanders like the blue curve. We assume the offset of the real alignment curve from the linear line is limited to the band indicated by the dashed grey lines. b) Three examples of 2nd MFCC time series converted from the audio snippets of the word *Harry*.

The system is set to the weak teacher mode. The subsequence length is set to 0.8 seconds, about the length of time we expect it takes to pronounce a three- or four-syllable word. After 4,294 seconds, a pattern C is discovered. The first occurrence of the pattern is observed five seconds after the pattern is learned, as shown in **Fig.** 25. Each time an occurrence is observed, the words are ranked based on their term frequency–inverse document frequency (tf-idf) [53], where the term frequency is calculated as the number of associations between the word and the pattern. The tf-idf measures how important a word is to the pattern, and the top-ranked word is the most likely to be associated with the pattern.

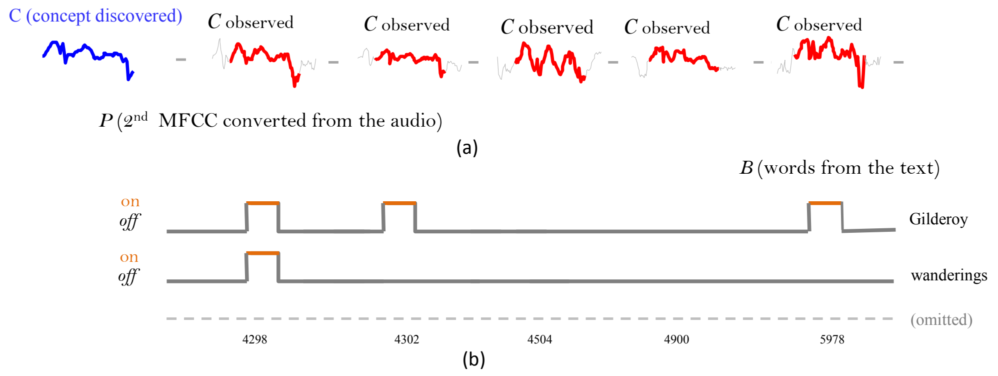


Fig. 25 a) After we have learned the concept *C*, our system monitors for future occurrences of it. Here, it sees five examples (the long spaces in between have been edited out for clarity). b) By checking the text associated with each occurrence, we can learn that the concept is associated with the word *Gilderoy*

Five subsequences are observed to be similar to the pattern C. The word *Gilderoy* is *on* for three of the subsequences. All the remaining words are *on* for at most one subsequence. **Fig.26** shows the rank changes for the word *Gilderoy* and one exemplar word *wanderings* as more occurrences are observed. Here, we can see that the rank of *Gilderoy* becomes 1st after five occurrences. Conversely, although the word *wanderings* was ranked 1st at the beginning, it degrades to lower ranks as more occurrences are observed. After a careful check of the original audio, we identify that the pattern is *Gilderoy*.

The reader may be surprised that the word *Gilderoy* was initially confused with the word *wanderings* in spite of the fact that they don't have similar pronunciations. Recall that we use a single MFCC coefficient, rather than all thirteen coefficients typically used in the literature for audio analysis.

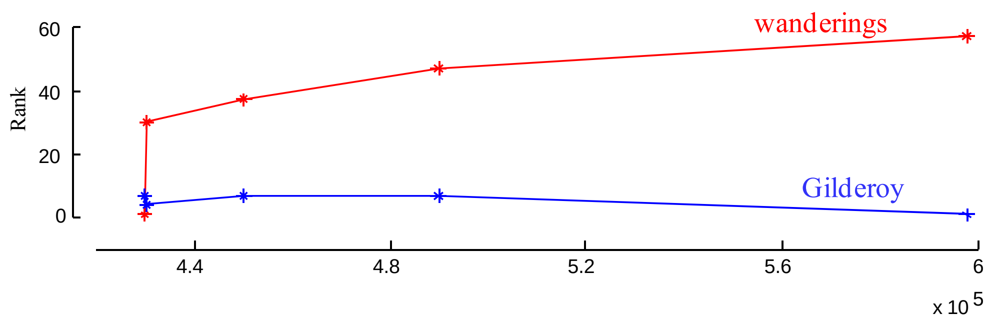


Fig.26 The rank of two exemplar words *Gilderoy* and *wanderings* as more occurrences of the pattern are observed. Lower rank means higher probability for the pattern. The word *Gilderory* is top-ranked after five occurrences are observed.

4.8 Weak Teaching Example: Energy Disaggregation

Discovering patterns from the whole-home electricity signal and then "factoring" these patterns into individual component appliances is called *household energy disaggregation*, and it has been an active research topic in recent years [30][36][7][26]. It can reveal how energy is consumed by what device, enabling users to take effective energy-saving steps.

Many methods have been proposed for energy disaggregation, most of which focus on classifying electrical events [36][7][26]. In this example, we leverage off our weak teacher framework to explore the correspondence between the temporal patterns in the whole-home energy signal and the operations of component devices.

We investigated a publicly available dataset [3], which contains a main metering that measures the whole-home energy consumption, and three sub-meterings that measure the energy consumption in the kitchen, in the laundry room, and by the water-heater and an air-conditioner. The dataset contains nearly four years of the power data for a house. For our experiment, we randomly choose two months' data. To be complete, we add a virtual sub-metering as *other appliances*, which is the difference between the whole-home energy and the combination of the three sub-meterings. We threshold all the sub-meterings data to be binary, and thus, as illustrated in **Fig.27**, the dataset is a multidimensional time series with one real-valued and four binary dimensions. We set the length of the patterns to be 120 minutes, which was our first guess as to a good comprise of the cycle time of various appliances.

Fig.27b shows an exemplar pattern that is discovered on the 51st day. Four occurrences of the pattern are observed in the next three days after the pattern is discovered, as shown in **Fig.27**a**.** For each occurrence, the weak teacher polls the four binary sub-meterings to see which are *on*.

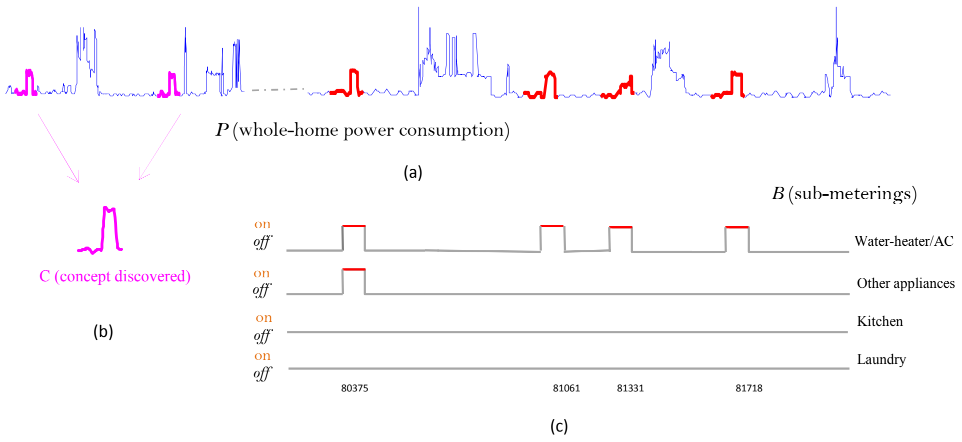


Fig.27 a) An excerpt of the whole-home energy consumption data. b) An exemplar of the learned concept C. c) The binary sub-meterings data indicating which component appliances are *on* at a given time

Our system observed a further seven occurrences of the pattern. For six of them, only the sub-metering corresponding to the `water-heater/air-conditioner` was on. Of the remaining one hit, just the `water-heater/air-conditioner` and the `other appliances` were on. Thus, we converge with the probabilities shown in **Fig.28**.

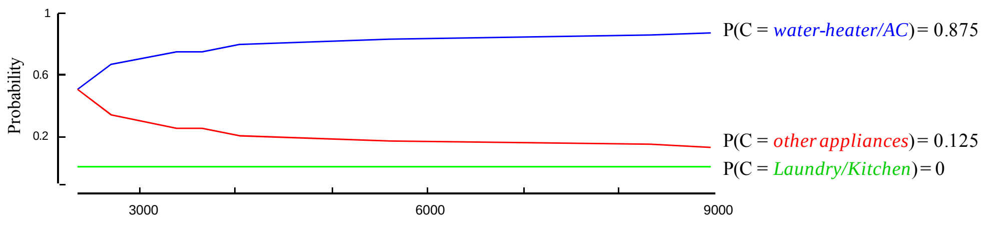


Fig.28 The probability of concept C associated with various component appliances. Note that while the probability of our pattern being attributed to the *water heater/AC* is initially just 0.58, it converges to 0.875 within a few weeks

4.9 Revisiting Options for Frequent Pattern Maintenance

In Section 2 we claimed that the motivation for introducing our novel algorithm for frequent pattern maintenance was that no algorithm for the task currently exists. Here we revisit this claim. The reader may note that the algorithm introduced in [46] seems to solve exactly this problem. For a given sliding window k minutes long, the algorithm can *maintain* the motif within the last k minutes. At each time step, a new datapoint is ingested, and the oldest datapoint is ejected. The algorithm maintains a "monitoring" data structure which can indicate if this incremental change could have affected the correctness of the current motif, and if the change necessitates it, a subroutine is invoked to discover the new motif.

If run on batch data this algorithm is very efficient, because on average the monitoring data structure rarely invokes the expensive subroutine. However, on streaming data we are not limited by the *average case*, we must be able to handle the *worst case*. Unfortunately there is no bound on how bad the worst case can be. In particular, taking an adversarial position, one can create a synthetic data stream that will force the expensive subroutine to be invoked at *every* time step. For this reason, in the streaming case one must set a relatively conservative value for k. The exact value depends on the sampling rate and the hardware used, but beyond a few hours is simply untenable. Consider **Fig.29** which shows two snippets from a weather station's solar panel [5]. There is a motif highlighted (blue/green) which appears to correspond to the concept `Sunny-Day-with-Shadow-Occlusion`. However, these two examples are 127 days apart, well beyond the capacity of the algorithm in [46] to discover.

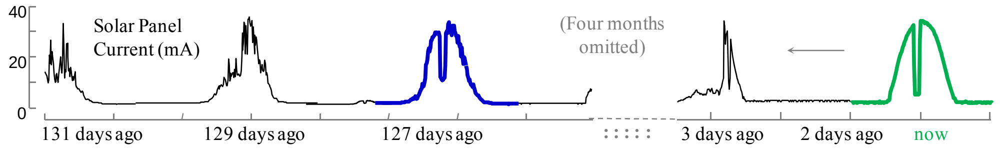


Fig.29 A never-ending time series stream from a weather station's solar panel, only a fraction of which we can buffer. A pattern we are observing now seems to have also occurred about four months ago. Such a long lag between patterns offers an insurmountable obstacle to exact algorithms [46]

This is not to criticize [46], the algorithm is elegant, exact, and significantly better than the obvious brute-force algorithm. However its limited time window at k minutes (k < 300) guarantees that it will not be able find *daily* patterns of behavior. In contrast, while our algorithm is approximate, its limitless time horizon means that we should expect it to eventually stumble on two occurrences of a repeated pattern, and do so faster than random sampling.

For completeness we performed an experiment to compare our algorithm with [46]. From the UCR archive [35] we take examples of class 1 from the MALLAT dataset and consider them as *target* motifs. We randomly embed these into a much longer random walk time series, at every *X* plus N(0,*X*/4) seconds, and our task is to recover at least one matching pair as soon as possible. In this experiment, the length of each subsequence is 1024, and the data rate is 50Hz. The total length of the time series is 2 hours long. The window size for each algorithm is chosen to guarantee real-time processing. According to our experimental results and the suggestions from the authors of [46], to guarantee real-time processing, the window size for the rival algorithm in [46] is 4,000 datapoints, and the buffer size for our algorithm is 80 subsequences. With these setting, we varied *X* from 25 seconds to 300 seconds. For each *X*, we repeated the experiments 10 times, each time re-generating a new time series, and counted the number of times each algorithm succeeded detecting at least one matching pair of motifs. The results are shown in **Fig.30**.

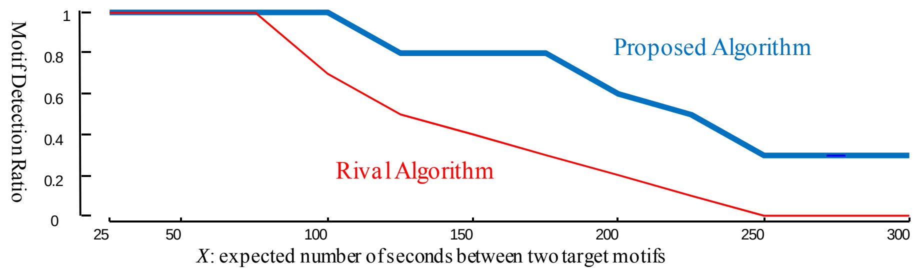


Fig.30 Comparison of our dendrogram algorithm with the rival method proposed in [46]. The x-axis is the expected number of seconds to see two target motifs; y-axis is the probability that the target concept is discovered

As we can see, both algorithms are capable of finding the target motifs when *X* is small. However, as *X* increases, the probability of the rival algorithm detecting the motifs is much smaller than that of our algorithm. When *X* is larger than 250 seconds, the rival algorithm can *never* detect the motifs, whereas our algorithm is still capable of discovering the concept. This is because as *X* increases, the target motifs are increasingly rare, that is, two target motifs are increasingly further away from each other. When any two target motifs are further apart than the window size, the rival algorithm can never detect a matching pair, whereas our algorithm does not have a limited time horizon, and thus is possible to find a matching pair even when the patterns are very rare.

Imagine that there exists an algorithm that can see the whole time series at any time; the detection ratio as defined above would be 100% for any X. We call this hypothetical algorithm the *Oracle* algorithm. In the next experiment, we compare our algorithm with the *Oracle* algorithm. Note that the buffer size required for the *Oracle* algorithm is infinite. However, as we will show, our algorithm can approach the performance of the *Oracle* algorithm even with a small limited buffer size, under the reasonable assumptions found in many real word datasets.

In this experiment, we generated time series in the same *manner as in* the previous experiment with $X = 300$. We started with the buffer size the same as in the previous experiment ($w = 80$), and increased the size by one buffer each time. For each buffer size, we repeated the experiment 10 times, and counted the number of times the target motifs were successfully detected by our algorithm. The results are shown in **Fig.31**. As we can see, with a buffer size of merely 240 subsequences, our algorithm detects the target concept with 100% success rate. That is, our algorithm approaches the performance of the *Oracle* algorithm with a much smaller buffer size.

The reason for this unintuitive result is that in a sense our algorithm has an easier task. The *Oracle* algorithm is finding the true best motifs in the data. However, our algorithm only needs to find *any pair* of motifs to be successful. Thus if there are K examples of the motifs in the stream, there are $K(K\text{-}1)/2$ possible pairs that we can find that allow us to report success.

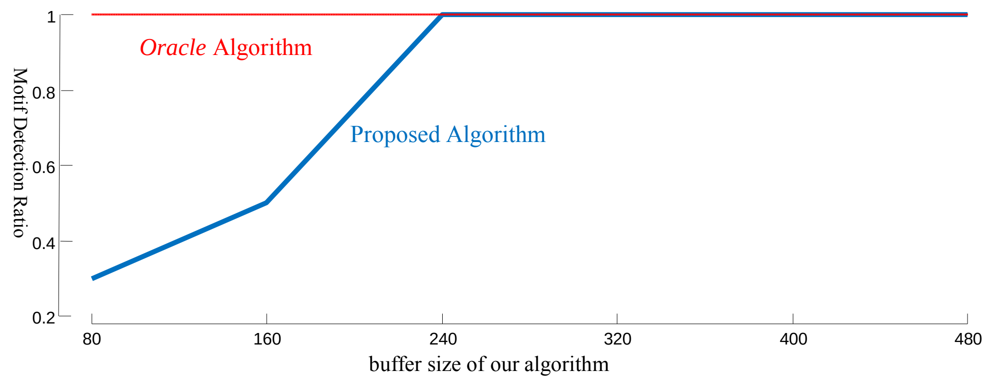


Fig.31 Comparison of our dendrogram algorithm with the Oracle algorithm. The performance of our algorithm approaches the Oracle algorithm as the buffer size increases

These two results in essence *lower bound* and *upper bound* the performance of our frequent pattern discovery algorithm. We have shown that we are doing much better than the only obvious rival algorithm, and that under reasonable settings and assumptions, there is little room for improvement. Thus, as we note in the next section, most of our planned future work focuses on *learning* issues.

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 (c.f. Section 4.3), and that it is robust to significant levels of noise (c.f. **Fig.20** and **Fig.23**). Moreover, by applying it to incredibly diverse domains, including bird calls, residential power consumption, speech-to-text alignment, body-area-networks, insect feeding behaviors, cardiology, motion analysis and flying insects, we have shown it is a very general and flexible framework.

Our framework has some limitations. It needs some human intervention to set initial parameters and it requires (or at least, greatly benefits from) some domain dependent settings. In future work, we hope to remove the few assumptions/parameters we have and apply our ideas to year-plus length streams. In particular, we plan to investigate extensions to situations where the patterns may be *very* rare, perhaps making up less than one-billionth of the overall data. At the very least, it is clear that our frequent pattern mining algorithms would be greatly challenged by such domains. In addition we will consider the issue of *concept drift*. For example, recall our case study with invasive species of flying insects. It is known that the wingbeat "signatures" of an individual insect change with its age. Moreover, the wingbeat frequency changes with temperature, which itself changes with the seasons. Addressing such challenges will allow our system to be more broadly applied.

We have made all our code and data freely available at [64] and hope to see our work built upon and applied to an even richer set of domains.

Reference

[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] J. D. Ambert, T. P. Hodgman, E. J. Laurent, G. L. Brewer, M. J. Iliff, and R. Dettmers: The Northeast Bird Monitoring Handbook. *American Bird Conservancy, The Plains, VA*, 2009

[3] K. Bache, M. Lichman. UCI Machine Learning Repository *http://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption. Irvine*, CA: University of California, School of Information and Computer Science, 2013

[4] R. Bardeli, D. Wolff, F. Kurth, M. Koch, K.H. Frommolt. Detecting bird sounds in a complex acoustic environment and application to bioacoustic monitoring. *Pattern Recognition Letters*, 31 (2010), pp. 1524–1534

[5] G., Barrenetxea, et al. Sensorscope: Out-of-the-Box Environmental Monitoring. *IPSN*, 2008.

[6] 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.

[7] M. Berges, E. Goldman, H.S. Matthews, L. Soibelman. Enhancing electricity audits in residential buildings with non-intrusive load monitoring. *Journal of Industrial Ecology: Special Issue on Environmental Applications of Information and Communications Technology*, 2010, 14(5):844–858.

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

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

[10] 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.

[11] 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.

[12] M. Charikar, K. Chen, and M. Farach-Colton: Finding frequent items in data streams. *In Proceedings of the 29th ICALP International Colloquium on Automata, Languages and Programming*, pages 693–703, 2002.

[13] B.Chiu, E. Keogh, S. Lonardi. Probabilistic discovery of time series motifs. *In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493-498, 2003

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

[15] I. Dagan, and S. P. Engelson. Committee-based sampling for training probabilistic classifiers. In *ICML* 1995, vol. 95, pp. 150-157.

[16] D. K. Dawson and M. G. Efford. Bird Population Density Estimated from Acoustic Signals. *Journal of Applied Ecology*. Volume 46 Issue 6, Pages 1201–1209.

[17] 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).

[18] Y. Dodge, *Oxford Dictionary of Statistical Terms* (2003), OUP. ISBN 0-19-850994-4

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

[20] C. Estan and G. Varghese: New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems*, 21(3):270–313, 2003.

[21] 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.

[22] A. Fujii, T. Tokunaga, K. Inui, and H. Tanaka. Selective sampling for example-based word sense disambiguation. *Computational Linguistics*,1998; 24(4), 573-597.

[23] 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.

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

[25] Google Prediction API. Accessed July 31 2013: *https://developers.google.com/prediction/docs/pricing*

[26] S. Gupta, S. Reynolds, S.N. Patel. ElectriSense: Single-point sensing using EMI for electrical event detection and classification in the home. *In Proceedings of the Conference on Ubiquitous Computing*, 2010

[27] J. Hinman and E. Hickey. Modeling and Forecasting Short-Term Electricity Load Using Regression Analysis. *Working Paper, Illinois State University, Normal (US),* Fall 2009.

[28] D.T. Holyoak. Nightjars and their Allies: the Caprimulgiformes. *Oxford University Press, Oxford, New York.* ISBN 0-19-854987-3, 2001.

[29] B. Hu, Y. Chen and E. J. Keogh. Time series classification under more realistic assumptions. *SDM* 2013.

[30] J. Lines, A. Bagnall, P. Caiger-Smith, S. Anderson: Classification of Household Devices by Electricity Usage Profiles. *IDEAL* 2011: 403-412

[31] C. Jin, W. Qian, C. Sha, J. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. *In Proceedings of the 12th ACM CIKM International Conference on Information and Knowledge Management*, pages 287–294, 2003.

[32] 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.

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

[34] R. Karp, C. Papadimitriou, and S. Shenker: A simple algorithm for finding frequent elements in sets and bags. *ACM Transactions on Database Systems*, 28:51–55, 2003.

[35] E. Keogh, Q. Zhu, B. Hu, Y. Hao. X. Xi, L. Wei, & C. A. Ratanamahatana. The UCR Time Series Classification/Clustering Homepage: *www.cs.ucr.edu/~eamonn/time_series_data/* (2011)

[36] J. Kolter, and T. Jaakkola, Approximate inference in additive factorial HMMs with application to energy disaggregation**.** *In Journal of Machine Learning Research - Proceedings Track (JMLR)* 2012, vol. 22, pp. 1472–1482.

[37] A. Krishnamurthy, S. Balakrishnan, M. Xu, A. Singh, A. (2012). Efficient active algorithms for hierarchical clustering. *arXiv preprint arXiv:1206.4672*.

[38] C. J. MacLeod, T. Greene, D. I. MacKenzie, and R. B. Allen: Monitoring widespread and common bird species on New Zealand's conservation lands: a pilot study. *New Zealand Journal of Ecology*, *36*(3), 0-0. 2012.

[39] G. Manku and R. Motwani. Approximate frequency counts over data streams. *In International Conference on Very Large Databases*, pages 346–357, 2002.

[40] P. Mermelstein, Distance measures for speech recognition, psychological and instrumental, *Pattern Recognition and Artificial Intelligence*, 1976.

[41] A. Metwally, D. Agrawal, and A. E. Abbadi. Efficient computation of frequent and top-k elements in data streams. *In International Conference on Database Theory*, 2005.

[42] PAMAP, Physical activity monitoring for aging people, www.pamap.org/demo.html, retrieved 2012.

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

[44] C. L.Morales, M. P. Arbetman, S. A. Cameron, and M. A. Aizen. Rapid ecological replacement of a native bumble bee by invasive species. *Frontiers in Ecology and the Environment*., 2013.

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

[46] A. Mueen, and E. J. Keogh: Online discovery and maintenance of time series motifs. *KDD 2010*: 1089-1098.

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

[48] J. R. Norris. Markov chains. *Cambridge university press*, 1998.

[49] C.S. Robbins. Effect of time of day on bird activity. *Studies in Avian Biology 6* (1981) :275–286.

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

[51] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, ... & Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD intersnational conference on Knowledge discovery and data mining* (pp. 262-270). ACM.

[52] J.K. Rowling. Harry Potter and the Chamber of Secrets. Read by Stephan Fry. New York. *Arthur A. Levine books (scholastics)*.1997

[53] G. Salton, MJ. McGill. Introduction to Modern Information Retrieval. *McGraw-Hill. ISBN 0-07-054484-0,* 1986

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

[55] B. Settles, M. Craven, and L. Friedland: Active learning with real annotation costs. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning,* 2008

[56] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri: Medians and beyond: New aggregation techniques for sensor networks. *In ACM SenSys*, 2004.

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

[58] G. Tur, D. Hakkani-Tür, and R. E. Schapire: Combining active and semi-supervised learning for spoken language understanding. *Speech Communication,* 45(2), 171-186. 2005

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

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

[61] Y. Wu, C. Zhou, J. Xiao, J. Kurths, H.J. Schellnhuber. Evidence for a bimodal distribution in human communication. *Proc. Natl. Acad. Sci. USA*, 107 (2010), pp. 18803–18808

[62] H. Yu, SVM selective sampling for ranking with application to data retrieval. *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. *ACM,* 2005.

[63] S. Zhai, P.O. Kristensson, C. Appert, T.H. Anderson, X. Cao. Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review. *Foundations and Trends in Human-Computer Interaction.* 5, 2 (2012), 97-205.

[64] Y. Chen project webpage: *https://sites.google.com/site/nelframework/*

Appendix A: The Relationship between Buffer Size and Pattern Discovery Time

In the main text, we provided an equation to calculate the probability of discovering a repeated pattern within n steps when the buffer size is w, but did not discuss how we derived the equation. We relegated the derivation to this appendix to enhance the flow of the paper.

Theoretical Analysis

To calculate the probability of seeing at least two examples of the pattern in the buffer with no more than n steps when the buffer size is w under the assumptions given in the main text (c.f. Section 3.5), we model the problem with the classic urn and ball choice model [18] as follows.

An infinitely large urn contains colored balls in the ratio of one red ball to ninety-nine blue balls. At each time step, a ball is randomly sampled from the urn, and put into a box. The size of the box is w. When the box is full, we randomly discard a ball from the box, and replace it with the new sampled ball. The question at hand is: what are the chances of seeing at least two red balls in the box within n samplings when the box size is w?

This model simulates our problem by having the red ball represent the pattern, and the blue balls represent random data. The box simulates the buffer, and the data stream is generated through sampling the balls at each time step. The pattern is discovered when exactly two red balls are seen in the box (recall that we stop when we see two red balls, so we will never see three or more), so the goal is to figure out the probability of discovering the pattern within n steps when the box size is w.

According to the model, the probability of seeing at least two red balls in the next step is dependent on the current state of the box and independent of the previous states. For example, it is possible to see two red balls in the next step *only* when the box currently contains at least one red ball; and given the current state of balls in the box, the probability of seeing two red balls in the next step is independent of how many red balls there were previously. As such, this problem is a typical Markov process and is modeled using the Markov Chain [48] as follows.

We take as states the number of red balls in the boxes S_0, S_1 and S_2, where $\mathrm{S}_0/\mathrm{S}_1$ denotes that there is zero/one red ball in the box, and S_2 denotes that there are at least two red balls. For simplicity, we explicitly assume that the box is full of blue balls at the beginning; that is, the initial state is S_0. As such, the initial probability vector u is as follows:

$$u = [1 \quad 0 \quad 0]$$

We need to specify the transition matrix. The conditions that are required to transition from one state to another are listed in **Table 13**, where 'impossible' means with zero probability and 'certainty' means with hundred percent probability.

Table 13 Conditions for Transitioning States

Current State	**Next State**		
	$\mathbf{S_0}$	$\mathbf{S_1}$	$\mathbf{S_2}$
$\mathbf{S_0}$	new ball is blue	new ball is red	impossible
$\mathbf{S_1}$	discarded ball is red, and new ball is blue	discarded ball is red, and new ball is red; or, discarded ball is blue and new ball is blue	discarded ball is blue, and new ball is red
$\mathbf{S_2}$	impossible	impossible	certainty

At each step, the probability of sampling a red ball from the urn is 1/100, and of sampling a blue ball is 99/100; the probability of discarding a red ball from the box is $\frac{n_{red}}{w}$, and of discarding a blue ball is $\frac{w-n_{red}}{w}$, where n_{red} is the number of red balls currently in the box. Based on the conditions in **Table 13**, we can compute the transition probabilities and derive the transition matrix T as follows:

$$T = \begin{bmatrix} \frac{99}{100} & \frac{1}{100} & 0 \\ \frac{1}{w} * \frac{99}{100} & \frac{1}{w} * \frac{1}{100} + \frac{w-1}{w} * \frac{99}{100} & \frac{w-1}{w} * \frac{1}{100} \\ 0 & 0 & 1 \end{bmatrix}$$

According to the Markov Chain, the probability of seeing at least two red balls in the box (in state S_2) after n steps is the 3rd entry in the vector $u^{(n)} = uT^n$, and thus, we have:

$$Pr(w,n) = \left([1 \quad 0 \quad 0] \begin{bmatrix} \frac{99}{100} & \frac{1}{100} & 0 \\ \frac{1}{w} * \frac{99}{100} & \frac{1}{w} * \frac{1}{100} + \frac{w-1}{w} * \frac{99}{100} & \frac{w-1}{w} * \frac{1}{100} \\ 0 & 0 & 1 \end{bmatrix}^n \right)(1,3) \qquad (1)$$

which is the equation we provided in the main text.

To verify our analysis, we also performed an empirical study with simulations. We did simulations for different values of *w*, and compare the simulation results with the theoretical results calculated using **Eq.1**. **Fig.32** shows the comparison results for *w* = 10. As can be seen, the two results are essentially identical.

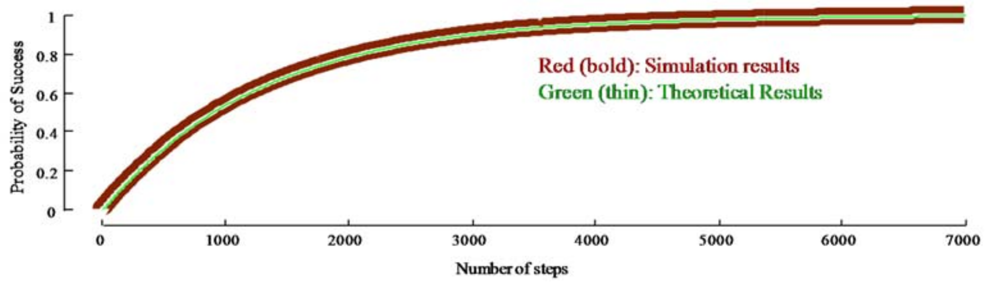


Fig.32 Comparison of simulation results (red/bold) with theoretical results (green/thin) for *w* =10