Logical-Shapelets: An Expressive Primitive for Time Series Classification

Abdullah Mueen
UC Riverside
mueen@cs.ucr.edu

Eamonn Keogh
UC Riverside
eamonn@cs.ucr.edu

Neal Young
UC Riverside
neal@cs.ucr.edu

ABSTRACT

Time series shapelets are small, local patterns in a time series that are highly predictive of a class and are thus very useful features for building classifiers and for certain visualization and summarization tasks. While shapelets were introduced only recently, they have already seen significant adoption and extension in the community.

Despite their immense potential as a data mining primitive, there are two important limitations of shapelets. First, their expressiveness is limited to simple binary presence/absence questions. Second, even though shapelets are computed offline, the time taken to compute them is significant.

In this work, we address the latter problem by introducing a novel algorithm that finds shapelets in less time than current methods by an order of magnitude. Our algorithm is based on intelligent caching and reuse of computations, and the admissible pruning of the search space. Because our algorithm is so fast, it creates an opportunity to consider more expressive shapelet queries. In particular, we show for the first time an augmented shapelet representation that distinguishes the data based on conjunctions or disjunctions of shapelets. We call our novel representation *Logical-Shapelets*. We demonstrate the efficiency of our approach on the classic benchmark datasets used for these problems, and show several case studies where logical shapelets significantly outperform the original shapelet representation and other time series classification techniques. We demonstrate the utility of our ideas in domains as diverse as gesture recognition, robotics, and biometrics.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithm

Keywords

Time Series, Classification, Information Gain, Logic Expression

KDD'11, August 21–24, 2011, San Diego, California, USA.

1. INTRODUCTION

Time series shapelets were introduced in 2009 as a primitive for time series data mining [19]. Shapelets are small sequences that separate the time series into two classes by asking the question *"Does this unknown object have a subsequence that is within T of this shapelet*?" Where there are three or more classes, repeated application of shapelets can be used (i.e. a decision tree-like structure) to predict the class label. Figure 1 shows examples of shapelets found in a dataset of accelerometer signals [11]. Every time series in the dataset corresponds to one of two hand motions performed by an actor tracing a *circular* or *rectangular* path through the air with an input device. The shapelet denoted by P in the figure is the one that maximally separates the two classes when used with a suitable threshold. In essence, the shapelet P captures the sinusoidal acceleration pattern of the circular motion along the Z-axis.

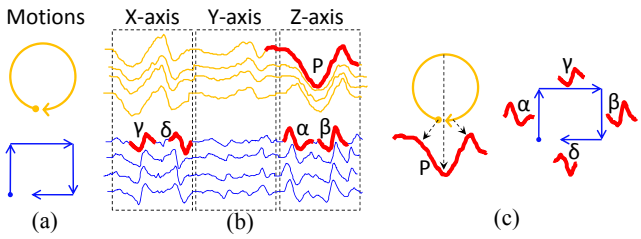

Figure 1: (a) Idealized motions performed with a Wii remote. (b) The concatenated accelerometer signals from recordings of actors performing the motions (c) Examples of Shapelets that describe each of the motions.

Time series shapelets are generating increasing interest among researchers [5] [13] [18] for at least two reasons. First, in many cases time series shapelets can learn the inherent structure of the data in a manner that allows intuitive interpretation. For example, beyond classifying, say, normal/abnormal heartbeats, shapelets could tell a cardiologist that the distinguishing feature is at the beginning of the dicrotic pulse. Second, shapelets are usually much shorter than the original time series, and unlike instance based methods that require comparison to the entire dataset, we only need one shapelet at classification time. Therefore, shapelets create a very compact representation of the class concept, and this compactness means that the time and space required for classification can be significantly reduced, often by at least two orders of magnitude. This is a particularly desirable property in resource limited systems such as sensor nodes, cell phones, mobile robots, smart toys, etc.

Despite the above promising features of time series shapelets, the current algorithm [19] for discovering them is still relatively lethar-

gic and, therefore, does not scale up to use on real-world datasets, which are often characterized by being noisy, long, and nonuniformly sampled. In addition, the current definition of shapelets is not expressive enough to represent certain concepts that seem quite common in the real world (examples appear later in this work). In particular, the expressiveness of shapelets is limited to simple binary presence/absence questions. While recursive application of these binary questions can form a decision tree-like structure, it is important to recognize that the full expressive power of a classic, machine-learning decision tree is not achieved (recall a decision tree represents the concept space of disjunction of conjunctions). For example, differentiating classes by using only binary questions is not possible if, the classes differ only in the *number* of occurrences of a specific pattern rather than presence/absence of a pattern.

In this work, we address the problem of scalability and show how this allows us to define a more expressive shapelet representation. We introduce two novel techniques to speedup search for shapelets. First, we precompute sufficient statistics [16] to compute the distance (i.e. similarity) between a shapelet and a subsequence of a time series in amortized constant time. In essence, we trade time for space, finding that a relativity small increase in the space required can help us greatly decrease the time required. Second, we use a novel admissible pruning technique to skip the costly computation of entropy (i.e. the goodness measure) for the vast majority of candidate shapelets. We have achieved up to 27× speedup over the current algorithm experimentally. We further show that we can combine multiple shapelets in logic expressions such that complex concepts can be described. For example, in the dataset shown in figure 1, there are discontinuities in the rectangular motion. It will not be possible to describe the rectangular class using one shapelet if there are variable pause times at the corners in different instances. In such cases, we can express the rectangular class by the logical shapelet "α **and** β **and** γ **and** δ." Here, each literal corresponds to one of the straight line motions as shown in figure 1(c). In addition, our algorithm is able to increase the gap/margin between classes even if they are already separated. This allows for more robust generalization from the training to test data. We have successfully used our algorithm to find interesting combinations of shapelets in accelerometer signals, pen-based biometrics, and accelerometer signals from a mobile robot.

It is important to note that there is essentially zero-cost for the expressiveness of logical shapelets. If we apply them to a dataset that does not need their increased representational power, they simply degenerate to classic shapelets, which are a special case. Thus, our work complements and further enables the growing interest in shapelets as a data mining tool. For example, [18] uses shapelets on streaming data as "early predictors" of a class, [13] uses an extension of shapelets in an application in severe weather prediction, [6] uses shapelets to find representative prototypes in a dataset, and [5] use shapelets to do gesture recognition from accelerometer data. However, none of these works have considered the logical combinations of shapelets we consider here or have obtained an significant speedup over the original algorithm.

2. DEFINITION AND BACKGROUND

In this section, we define shapelets and the other notations used in the paper.

Definition 1. A *Time Series* T is a sequence of real numbers $t_1, t_2, \ldots, t_m$. A time series *subsequence* $S_{i,l} = t_i, t_{i+1}, \ldots, t_{i+l-1}$ is a continuous subsequence of T starting at position i and length l.

A time series of length m can have $\frac{m(m+1)}{2}$ subsequences of all possible lengths from one to m.

If we are given two time series X and Y of same length m, we can use the euclidean norm of their difference (i.e. $X-Y$) as a distance measure. To achieve scale and offset invariance, we must normalize the individual time series using *z-normalization* before the actual distance is computed. This is a critical step; even tiny differences in scale and offset rapidly swamp any similarity in shape [8]. In addition, we normalize the distances by dividing with the length of the time series. This allows comparability of distances for pairs of time series of various lengths. We call this *length-normalization*.

The normalized euclidean distance is generally computed by the formula $\sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - y_i)^2}$. Thus, computing a distance value requires time linear on the length of the time series. In contrast, we compute the normalized eucledian distance between X and Y using five numbers derived from X and Y. These numbers are denoted as sufficient statistics in [16]. The numbers are $\sum x$, $\sum y$, $\sum x^2$,$\sum y^2$ and $\sum xy$. It may appear that we are doing more work than necessary; however, as we make clear in section 3.1, computing the distance in this manner enables us to reuse computations and reduce the amortized time complexity from *linear* to *constant*.

The sample mean and standard deviation can be computed from these statistics as $\mu_x = \frac{1}{m}\sum x$ and $\sigma_x^2 = \frac{1}{m}\sum x^2 - \mu_x^2$, respectively. The positive correlation and the normalized euclidean distance between X and Y can then be expressed as below [14].

$$C(x,y) = \frac{\sum xy - m\mu_x\mu_y}{m\sigma_x\sigma_y} \tag{1}$$

$$dist(x,y) = \sqrt{2(1 - C(x,y))} \tag{2}$$

Many time series data mining algorithms (1-NN classification, clustering, density estimation, etc.) require only the comparison of time series that are of equal lengths. In contrast, time series shapelets require us to test if a short time series (the shapelet) is contained within a certain threshold somewhere inside a much longer time series. To achieve this, the shorter time series is slid against the longer one to find the best possible alignment between them. We call this distance measurement the *subsequence distance* and define it as $sdist(x,y) = \sqrt{2(1 - C_s(x,y))}$ where x and y are the two time series with lengths m and n, respectively, and for $m \leq n$

$$C_s(x,y) = \min_{0 \leq l \leq n-m} \frac{\sum_{i=1}^{m} x_i y_{i+l} - m\mu_x\mu_y}{m\sigma_x\sigma_y} \tag{3}$$

In the above definition μ_y and σ_y denote the mean and standard deviation of m consecutive values from y starting at position $l+1$. Note that, $sdist$ is not symmetric.

Assume that we have a dataset D of N time series from C different classes. Let us also assume, every class i ($i = 1, 2, \ldots, C$) has n_i ($\sum_i n_i = N$) labeled instances in the dataset. An instance time series in D is also denoted by D_i for $i = 1, 2, \ldots, N$.

Definition 2. The *entropy* of a dataset D is defined as $E(D) = -\sum_{i=1}^{C} \frac{n_i}{N} log(\frac{n_i}{N})$.

If the smallest time series in D is of length m, there are at least $N\frac{m(m+1)}{2}$ subsequences in D that are shorter than every time series in D. We define a *split* as a tuple (s, τ) where s is a subsequence and τ is a distance threshold. A split divides the dataset D into two disjoint subsets or partitions $D_{left} = \{x : x \in D, sdist(s,x) \leq \tau\}$ and $D_{right} = \{x : x \in D, sdist(s,x) > \tau\}$. We define the cardinality of each partition by $N_1 = |D_{left}|$ and

Algorithm 1 $Shapelet\_Discovery(D)$

Require: A dataset D of time series
Ensure: Return the shapelet
1: $m \leftarrow$ minimum length of a time series in D
2: $maxGain \leftarrow 0, maxGap \leftarrow 0$
3: **for** $j \leftarrow 1$ to $|D|$ **do** {every time series in D}
4: $S \leftarrow D_j$
5: **for** $l \leftarrow 1$ to m **do** {every possible length}
6: **for** $i \leftarrow 1$ to $|S| - l + 1$ **do** {every start position}
7: **for** $k \leftarrow 1$ to $|D|$ **do** {compute distances of every time series to the candidate shapelet $S_{i,l}$}
8: $L_k \leftarrow sdist(S_{i,l}, D_k)$
9: $sort(L)$
10: $(\tau, updated) \leftarrow bestIG(L, maxGain, maxGap)$
11: **if** $updated$ **then** {gain and/or gap are changed}
12: $bestS \leftarrow S_{i,l}$,$best\tau \leftarrow \tau$,$bestL \leftarrow L$
13: **return** $(bestS, best\tau, bestL, maxGain, maxGap)$

Algorithm 2 $sdist(x, y)$

Require: Two time series x and y. Assume $|x| \leq |y|$.
Ensure: Return the normalized distance between x and y
1: $minSum \leftarrow \infty$
2: $x \leftarrow zNorm(x)$
3: **for** $j \leftarrow 1$ to $|y| - |x| + 1$ **do** {every start position on y}
4: $sum \leftarrow 0$
5: $z \leftarrow zNorm(y_{j,|x|})$
6: **for** $i \leftarrow 1$ to $|x|$ **do** {compute the eucledian distance}
7: $sum \leftarrow sum + (z_i - x_i)^2$
8: $minSum \leftarrow min(minSum, sum)$
9: **return** $\sqrt{minSum/|x|}$

$N_2 = |D_{right}|$. We use two quantities to measure the goodness of a split: *information gain* and *separation gap*.

Definition 3. The *information gain* of a split is $I(s, \tau) = E(D) - \frac{N_1}{N}E(D_{left}) - \frac{N_2}{N}E(D_{right})$.

Definition 4. The *separation gap* of a split is $G(s, \tau) = \frac{1}{N_2}\sum_{x \in D_{right}} sdist(s, x) - \frac{1}{N_1}\sum_{x \in D_{left}} sdist(s, x)$.

Now we are in position to define time series shapelets.

Definition 5. The *shapelet* for a dataset D is a tuple of a subsequence of an instance within D and a distance threshold (i.e. a split) that has the maximum information gain while breaking ties by maximizing the separation gap.

2.1 Brute-force Algorithm

In order to properly frame our contributions, we begin by explaining the brute-force algorithm for finding shapelets in a dataset D (algorithm 1). Dataset D contains multiple time series of two or more classes and possibly of different lengths.

Since time series can be considered to be points in high dimensional space, we will denote D as a database of points. The algorithm generates a candidate subsequence $S_{i,l}$ in the three loops in lines 3, 5, and 6 of algorithm 1, which essentially generates all possible subsequences of all possible lengths from D. In lines 7-9, an array L is created which holds the points in D in the sorted order of their distance from the shapelet candidate. A schematic view of this array is illustrated in figure 2. We call this schematic line an *orderline*. Intuitively, the ideal shapelet is the one that orders the data as such all instances of one class are near the origin, and all instances of the other classes are to the far right, with no interleaving of the classes.

Algorithm 3 $bestIG(L, magGain, maxGap)$

Require: An order line L and current best gain and gap.
Ensure: Update the best information gain and separation gap and return the split point τ
1: **for** $k \leftarrow 1$ to $|D| - 1$ **do**
2: $\tau \leftarrow (L_k + L_{k+1})/2$
3: Count $n_{1,i}$ and $n_{2,i}$ for $i = 1, 2, \ldots, C$.
4: Count N_1 and N_2 for both the partitions.
5: $I \leftarrow$ Information gain computed by definition 3
6: $G \leftarrow$ Separation gap computed by definition 4
7: **if** $(I > maxGain$ **or** $(I = maxGain \bigwedge G > maxGap))$ **then**
8: $maxGain \leftarrow I, maxGap \leftarrow G, max\tau \leftarrow \tau$
9: $updated \leftarrow$ **true**
10: **return** $(max\tau, updated)$

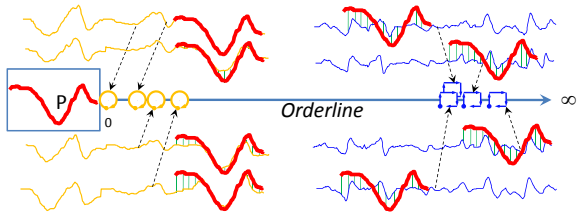

Figure 2: Orderline for the shapelet P. Each time series is placed on the orderline based on the $sdist$ from P. Note that, the time series that carries P is placed at position 0 on the orderline. Also note that, P aligns at different positions on different time series.

Distance values are computed by algorithm 2. Both of the normalizations – z-normalization and length-normalization (before and after the computation of euclidean distance, respectively) – are performed here. Note that repeatedly performing normalization before computing distances is expensive because the same sequences are normalized multiple times. Our novel caching mechanism in section 3.1 will remove this difficulty.

Given the orderline, the computation of information gain is linear in the dataset size. Algorithm 3 describes the method of finding the best split point paired with the current candidate sequence. At line 2, we selected the mid points between two consecutive points on the orderline as split (τ) points. Although there are infinite possible split points, there are at most $|D| - 1$ distinct splits that can result in different information gains. Therefore, it is sufficient to try only these $|D| - 1$ splits. Computing the information gain from the orderline requires $O(N)$ time, which includes counting statistics per class for the three sets (i.e. D, D_{left}, D_{right}) in line 3 and 4. The rest of the computation can be done in $O(C)$ time in line 5 and 6. Finally, the current best gain and gap are updated if appropriate.

The time complexity of the algorithm, assuming no additional memory usage (i.e. linear space cost), is clearly untenable. There are at least $N\frac{m(m+1)}{2}$ shapelet candidates (i.e. the number of iterations through loops in line 3, 5, and 6). For each of the candidates, we need to compute distances to each of the N time series in D (i.e. line 8). Every distance computation takes $O(m^2)$ time. In total, we need $O(N^2m^4)$ arithmetic operations. Such a huge com-

putational cost makes the brute-force algorithm infeasible for real applications. Since N is usually small as being the size of a labeled dataset, and m is large to make the search for local features meaningful, we focus on reducing factors of m from the cost.

3. SPEEDUP TECHNIQUES

The original shapelet work introduced by Ye, *et al.* [19] showed an admissible technique for abandoning some unfruitful entropy computations early. However, this does not improve the worst case complexity. In this work, we reduce the worst case complexity by caching distance computations for future use. In addition, we describe a very efficient pruning strategy resulting in an order of magnitude speedup over the method of Ye *et al.*.

3.1 Efficient Distance Computation

In algorithm 1, any subsequence of D_j with any length and starting at any position is a potential shapelet candidate. Such a candidate needs to be compared against every subsequence of D_k with the same length, starting at any position. The visual intuition of the situation is illustrated by figure 3. For any two instances D_j and D_k in D, we need to consider all possible parallelograms like the ones shown in the figure. For each parallelogram, we need to scan the subsequences to sum up the squared errors while computing the distance. Clearly there is a huge redundancy of calculations between successive and overlapping parallelograms.

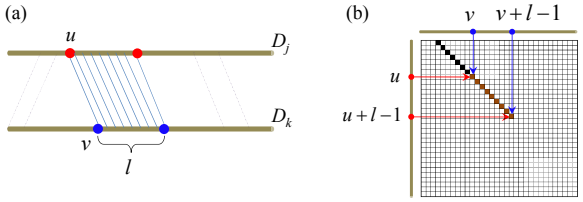

Figure 3: (a) Illustration of a distance computation required between a pair of subsequences starting at positions u and v, respectively, and of length l. Dashed lines show other possible distance computations. (b) The matrix $\mathbb{M}$ for computing the sum of products of the subsequences in (a).

Euclidean distance computation for subsequences can be made faster by reusing overlapping computation. However, reusing computations of z-normalized distances for overlapping subsequences needs at least quadratic space and, therefore, is not tenable for most applications. When we need all possible pairwise distances among the subsequences, as we do in finding shapelets, spending the quadratic space saves a whole order of magnitude of computation time.

For every pair of points (D_j, D_k) we compute five arrays. We represent these arrays as $Stats_{x,y}$ in the algorithms. Two of the arrays (i.e. $\mathbb{S}_x$ and $\mathbb{S}_y$) store the cumulative sum of the individual time series x and y. Another two (i.e. $\mathbb{S}_{x^2}$ and $\mathbb{S}_{y^2}$) store the cumulative sum of squared values. The final one (i.e. $\mathbb{M}$) is a 2D array that stores the sum of products for different subsequences of x and y. The arrays are defined mathematically below. All of the arrays have left margin of zero values indexed by 0. More precisely, $\mathbb{S}_x[0], \mathbb{S}_y[0], \mathbb{S}_{x^2}[0], \mathbb{S}_{y^2}[0], \mathbb{M}[0,0], \mathbb{M}[u,0]$ for $u = 1, 2, \ldots, |x|$, and $\mathbb{M}[0,v]$ for $v = 1, 2, \ldots, |y|$ are all zeros.

$$\mathbb{S}_x[u] = \sum_{i=0}^{u} x_i\,, \qquad \mathbb{S}_y[v] = \sum_{i=0}^{v} y_i,$$
$$\mathbb{S}_{x^2}[u] = \sum_{i=0}^{u} x_i^2\,, \qquad \mathbb{S}_{y^2}[v] = \sum_{i=0}^{v} y_i^2$$
$$\mathbb{M}[u,v] = \begin{cases} \sum_{i=0}^{v} x_{i+t} y_i & \text{if } u > v, \\ \sum_{i=0}^{u} x_i y_{i+t} & \text{if } u \le v \end{cases}$$

Algorithm 4 $sdist\_new(u, l, Stats_{x,y})$

Require: start position u and length l and the sufficient statistics for x and y.
Ensure: Return the subsequence distance between $x_{u,l}$ and y

```
1: minSum ← ∞
2: {M, S_x, S_y, S_x², S_y²} ← Stats_{x,y}
3: for v ← 1 to |y| − |x| + 1 do
4:     d ← distance computed by (1) and (2) in constant time
5:     if d < minSum then
6:         minSum ← d
7: return √minSum
```

where $t = abs(u - v)$.

Given that we have computed these arrays, the mean, variance, and the sum of products for any pair of subsequences of the same length can be computed as below.

$$\mu_x = \frac{\mathbb{S}_x[u+l-1] - \mathbb{S}_x[u-1]}{l}, \mu_y = \frac{\mathbb{S}_y[v+l-1] - \mathbb{S}_y[v-1]}{l}$$
$$\sigma_x = \frac{\mathbb{S}_{x^2}[u+l-1] - \mathbb{S}_{x^2}[u-1]}{l} - \mu_x^2, \sigma_y = \frac{\mathbb{S}_{y^2}[v+l-1] - \mathbb{S}_{y^2}[v-1]}{l} - \mu_y^2$$

$$\sum_{i=0}^{l-1} x_{u+i} y_{v+i} = \mathbb{M}[u+l-1, v+l-1] - \mathbb{M}[u-1, v-1].$$

This in turn means that the normalized euclidean distance (the information we actually want) between any two subsequences $x_{u,l}$ and $y_{v,l}$ of any length l can now be computed using equations 1 and 2 in constant time. The algorithm 4 describes the steps. The algorithm takes as input the starting position u and the length l of the subsequence of x. It also takes the precomputed $Stats_{x,y}$ carrying the sufficient statistics. The algorithm iterates for all possible start positions v of y and returns the minimum distance. Thus we can see that the procedure $sdist\_new$ saves at least an $O(m)$ inner loop computation from procedure $sditst$.

3.2 Candidate Pruning

The constant time approach for distance computation introduced in the previous section helps reduce the total cost of shapelet discovery by a factor of m. Our next goal is to reduce the number of iterations that occur in the **for** loop at line 6 in the algorithm 1. The core idea behind our attack on this problem is the following observation: If we know (s, τ) is a poor shapelet (i.e. it has low information gain) then any similar subsequence to s must also result in a low information gain and, therefore, a costly evaluation (computing all the distances) can be safely skipped. Assume we have observed a good best-so-far shapelet at some point in the algorithm. Let us denote this shapelet (S_p, τ_p) and its information gain I_p. Imagine we now test $(S_{i,l}, \tau)$, and it has very poor information gain $I_{i,l} < I_p$. Let us consider the next subsequence $S_{i+1,l}$. Here is our fundamental question. Is it possible to use the relationship (the Euclidean distance) between $S_{i,l}$ and $S_{i+1,l}$, to prune $S_{i+1,l}$?

To develop our intuition, let us first imagine a pathological case. Suppose that $dist(S_{i,l}, S_{i+1,l}) = 0$; in other words, $S_{i+1,l}$ is the exact same shape as $S_{i,l}$. In that case we can obviously prune $S_{i+1,l}$, since its information gain must also be $I_{i,l}$. Suppose, however, in the more realistic case, that $S_{i,l}$ and $S_{i+1,l}$ are similar, but not identical. We may still be able to prune $S_{i+1,l}$. The trick is to ask, "how good could $S_{i+1,l}$ be?" , or a little more formally, "What is an upper bound on the information gain of $S_{i+1,l}$". It turns out that it is simple to calculate this bound!

Let the distance between $S_{i,l}$ and $S_{i+1,l}$ be R. By triangular inequality, $sdist(S_{i+1,l}, D_k)$ can be as low as $sdist(S_{i,l}, D_k) - R$ and as high as $sdist(S_{i,l}, D_k) + R$ regardless of the alignment of

$S_{i+1,l}$ on D_k. Thus, every point on the orderline of $S_{i,l}$ has a "mobility" in the range $[-R, R]$ from its current position. Given this mobility, our goal is to find the best configuration of the points that gives maximum information gain when split into two partitions. The points that lie outside $[\tau - R, \tau + R]$ can have no effect on the information gain for candidate $(S_{i+1,l}, \tau)$. For the points inside $[\tau - R, \tau + R]$ we can shift them optimistically in either direction to increase the information gain.

Every class $c \in C$ has n_c instances in the database which are divided into two partitions by a split. Let $n_{c,1}$ and $n_{c,2}$ be the number of instances of class c in partition D_{left} and D_{right}, respectively. A class is weighted to the left (or simply called *left-major/right-minor*) if $\frac{n_{c,1}}{N_1} > \frac{n_{c,2}}{N_2}$. Similarly, a class is called *right-major/left-minor* if $\frac{n_{c,1}}{N_1} \leq \frac{n_{c,2}}{N_2}$. The following lemma describes the optimal choice for a single point. We relegate the proof to Appendix to enhance the flow of the paper.

THEOREM 1. *Shifting a point from its minor partition to its major partition always increases information gain.*

If we shift a point from the minor partition to the major partition, the major/minor partitions of the class remain the same as before shifting, simply because, if $\frac{n_{c,1}}{N_1} > \frac{n_{c,2}}{N_2}$ then $\frac{n_{c,1}+1}{N_1+1} > \frac{n_{c,2}-1}{N_2-1}$. Therefore, shifting all the minor points of a class to its major partition increases the information gain monotonically, and thus, this can be treated as an atomic operation. We denote such a sequence of shifts as a "*transfer*". Transferring a class may change the major-minor partitions of other classes and, consequently, the transfer direction for that class. Therefore, if we transfer every class based on the major-minor partitions in the initial orderline, it does not necessarily guarantee the maximum information gain.

In the case of two-class datasets, there is always one left-major and one-right major class. Therefore, shifting all of the points in $[\tau - R, \tau + R]$ to their major partition will not change the major-minor partitions of either of the classes; thus, this guarantees the optimization (i.e. the upper bound) even if the initial transfer directions are used.

For a case of more than two-classes, almost all of the time initial major-minor partitions hold after all of the transfers are done. Unfortunately, it is possible to *construct* counter examples, even if we rarely confront them on real data. To obtain the optimal bound we need to try all possible transfer directions for all of the classes resulting in 2^C trials. Fortunately, many classification problems deal with a small number of classes. For example, 60% of the UCR time series datasets have four or less classes. Given this fact, having the 2^C constant in the complexity expression will not be an issue for many problems.

Algorithm 5 describes the computation of the upper bound. The algorithm loops through all distinct split positions (line 2-3). For every split position, the algorithm transfers all of the classes to their major partition based on the initial orderline (line 5-6) and computes the information gain to find the upper bound. Note that line 4 is "commented out" in the algorithm which is a **for** loop that checks all of the 2^C combinations of transfer directions. For exact bound in the case of many-class datasets, this line should be uncommented.

To summarize, for the two-class case we have an exact and admissible pruning technique. In the multi-class case we have a powerful heuristic that empirically gives answers that are essentially indistinguishable from the optimal answer.

3.3 The Fast Shapelet Discovery Algorithm

With the speedup techniques described in the previous section, we are now ready to plug them into algorithm 1 and build a faster version as shown in algorithm 6.

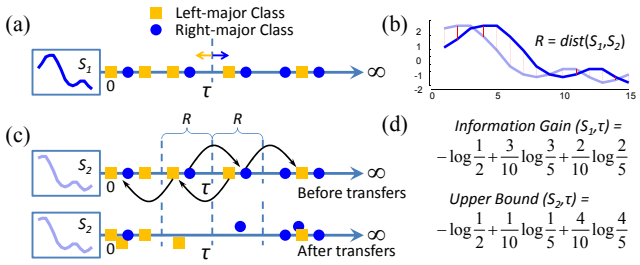

Figure 4: (a) A sequence S_1 and its orderline. (b) Distance between the sequences S_1 and S_2 is R. (c) The points on the orderline within $[\tau - R, \tau + R]$ are transferred to their majority partition. (d) The computation of the information gain for (S_1, τ) and upper bound for (S_2, τ).

Algorithm 5 $upperIG(L, R)$

Require: An order line L and the candidate distance R.
Ensure: Return an upper bound of information gain.

```
1:  maxI ← 0
2:  for k ← 1 to |D| − 1 except j do
3:      τ ← (L_k + L_{k+1})/2
4:      //for all 2^C combinations of transfer directions do
5:          for all points p ∈ [τ − R, τ + R]
6:              move p to its majority end.
7:          Count n_{1,i} and n_{2,i} for i = 1, 2, ..., C.
8:          Count N_1 and N_2 for both the partitions.
9:          I ← information gain computed by definition 3
10:         maxI ← max(maxI, I)
11:     return maxI
```

In lines 5-7, the sufficient statistics are computed for the current time series D_j paired with every other time series D_k.

The algorithm maintains a set of orderlines in the history H. For every candidate $S_{i,l}$, before committing to the expensive computation of the orderline, the algorithm quickly computes upper bounds using the orderlines in the history (line 14). If *any* of the upper bounds is smaller than the maximum gain achieved so far we can safely abandon the candidate (line 15).

Since the upper bound computation is based upon the triangular inequality, we are only allowed to use the previous orderlines computed for sequences of the same length as the current candidate [1]. Therefore, once the search moves on to the next length the algorithm clears the history H and starts building it up for the new length (line 9).

The size of the history H is a user-defined value and the algorithm is insensitive to this value once it is set to at least five. To prevent our history cache in line 22 growing without bound, we need to have a replacement policy. The oldest-first (LIFO) policy is the most suitable for this algorithm. This is because the recent subsequences tend to be correlated with the current candidate and, therefore, have small distances from the candidate. Note that, we do not add all orderlines to the history. We only add the orderlines that have less information gain (i.e. orderlines for poor shapelet candidate) than the current $maxGain$. Because only poor candi-

[1]The reader may wonder why we cannot create a bound between a sequence and a shorter sequence that is its prefix. Such bounds cannot be created because we are normalizing *all* sequences, and after normalizing the distances may increase *or* decrease.

Algorithm 6 $Fast\_Shapelet\_Discovery(D)$

Require: A dataset D of time series
Ensure: Return the shapelet

```
m ← minimum length of a time series in D
maxGain ← 0, maxGap ← 0
for j ← 1 to |D| do {every time series in D}
    S ← jth time series of D
    for k ← 1 to |D| do {compute statistics for S and D_k}
        x ← S, y ← D_k
        Stats_{x,y} ← {M, S_x, S_y, S_{x^2}, S_{y^2}}
    for l ← 1 to m do {every possible length}
        clear H
        for i ← 1 to |S| do {every start position}
            for w ← 1 to |H| do {every candidate in H}
                (L', S') ← H[w]
                R ← sdist(S_{i,l}, S')
                if upperIG(L', R) < maxGain then {prune this candidate}
                    continue with next i
            for k ← 1 to |D| do {since not pruned; compute distances of every time series to the candidate S_{i,l}}
                L_k ← sdist_new(i, l, Stats_{x,y})
            sort(L)
            (τ, updated) ← bestIG(L, maxGain, maxGap)
            if updated then {gain and/or gap are changed}
                bestS ← S_{i,l}, bestτ ← τ, bestL ← L
            add (L, S_{i,l}) to H if maxGain is not changed
return (bestS, bestτ, bestL, maxGain, maxGap)
```

dates have the power of pruning similar candidates by predicting their low information gain.

4. LOGICAL SHAPELET

A shapelet is a tuple consisting of a subsequence and a split point (threshold) that attempts to separate the classes in exactly two different groups. However, it is easy to imagine situations where it may not be sufficient to use only one shapelet to achieve separation of classes, but a *combination* of shapelets. To demonstrate this, we show a simple example. In Figure 5(a), we have a two-class problem where each class has two time series. The square class contains two sinusoidal patterns with both positive and negative peaks, while the circle class has only one positive *or* one negative peak. If we attempt to use the classic shapelet definition to separate these classes, we find there is no way to do so. Classic shapelets simply do not have the expressiveness to represent this concept. The reason is that every single distinct pattern appears in both of the classes, or in only one of the time series of one of the classes. For example, in 5(b) three different unary shapelets and their orderlines are illustrated, and none of them achieved a separation between the classes.

To overcome this problem, we propose using multiple shapelets which allow distinctions based on logical combinations of the shapelets. For example, if we use the first two shapelets in figure 5(b) then we can say that (S_1, τ_1) **and** (S_2, τ_2) separate the classes best. From now on, we use standard logic symbols $\bigwedge$ and $\bigvee$ for **and** and **or** operations. For a new instance x, if $sdist(S_1, x) < \tau_1 \bigwedge sdist(S_2, x) < \tau_2$ is true, then we can classify x as a member of the square class, or otherwise the circle class. When using multiple shapelets in such a way, chaining of logical combinations among the shapelets is possible; for example, $(S_1, \tau_1) \bigwedge (S_2, \tau_2) \bigvee (S_3, \tau_3)$. However, for the sake of simplicity and to guard against over fitting with too complex a model [4], we only consider two cases, only $\bigwedge$, and only $\bigvee$ combinations. We further guard against overfitting with too flexible a model by allowing only just a single threshold for both shapelets.

We have already seen how to create an orderline for classic shapelets, how do we define an orderline for conjunctive or disjunctive shapelets? To combine the orderlines for such cases, we adopt a very simple approach. For $\bigwedge$ operation, the *maximum* of the distances from the literal shapelets is considered as the distance on the new orderline, and for $\bigvee$ the *minimum* is considered. Apart from these minor changes, the computation of the entropy and information gain are unchanged.

For example, in figure 5(d) the combined orderline for $(S_1 \bigwedge S_2, \tau)$ is shown. The two classes are now separable because both the shapelets occur *together* in the square class and do not both occur together in individual instances of the circle class.

Given these minor changes to allow logical shapelets, we can still avail ourselves of the speedup techniques proposed in section 3. The original shapelet algorithm just needs to be modified to run multiple times, and some minor additional bookkeeping must be done. When the first shapelet is found by the algorithm 6, we now test to see if there is any class whose instances are in both the partitions of the optimal split. We call such a class "broken" (We could say "non-linearly separable," however, our situation has a slightly different interpretation). If there is a broken class, we continue to search for a new shapelet on the same dataset that can merge the parts of the broken class. However, this time we *must* make sure that new shapelet does not have a match with a subsequence in D_k that overlaps with the matching subsequence of an already discovered shapelet. After finding the new shapelet, we combine the orderlines and the threshold based on the appropriate logic operation. Finally, we are in position to measure the information gain to see if it is better than our best-so-far. If the gain improves, we test for a broken class again and continue as before. Once there is no broken class or the gain does not increase, we recursively find shapelet(s) in the left and right partitions.

The above approach is susceptible to overfitting. It can degenerate into $(S_1, \tau_1) \bigvee (S_2, \tau_2) \bigvee \ldots \bigvee (S_{n_i}, \tau_n)$ where n_i is the number of instances in the class i and each S_j for $j = 1, 2, \ldots, n_i$ is a subsequence of different instances of class i in the dataset D. In this case, all of the instances of class i would be in the left side with zero distances. To avoid overfitting, we can have a hard bound on the number of literals in the logic expression. In this work, the bound is hard coded to four, however, for a very large dataset we could slowly increase this number, if we carefully check to guard against overfitting. Note that the situation here is almost perfectly analogous to the problem of overfitting in decision trees, and we expect that similar solutions could work here.

5. EVALUATION

We begin by noting that we have taken extraordinary lengths to make all our experiments reproducible. As such, all code and data are available at [2], and will be maintained there in perpetuity. Moreover, while our proposed algorithm is not sensitive to the few parameters required as inputs, we explicitly state all parameters for every dataset at [2].

We wish to demonstrate two major points with our experiments. First, our novel speedup techniques can be used to find both classic and logical shapelets significantly faster. Second, there exist real-world problems for which logical shapelets are significantly more accurate than classic shapelets or other state-of-the-art techniques (see section 6).

To demonstrate the speedup, we have taken 24 datasets from the

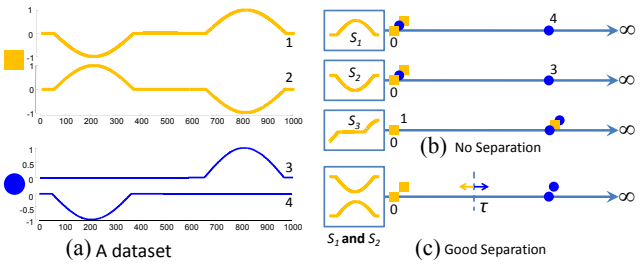

Figure 5: (a) Two classes of synthetic time series. (b) Examples of single shapelets that cannot separate the classes. Any other single shapelet would fail similarly. (c) Two shapelets connected by an *and* operation can separate the classes.

UCR time series archive [9]. For brevity the names and properties of these datasets and *tables* of time taken for running *the* shapelet algorithms on these datasets can be found at [2]. Here we content ourselves with a visual summary. In figure 6*(left)*, we show the speedups over the original algorithm. Our algorithm obtained *some* speedup for all of the datasets with a maximum of 27.2.

The two speedup methods described in section 3 are not independent of each other. Therefore, we also measure the *individual* speedups for each of the techniques while deactivating the other. The individual speedup factors for both the techniques are plotted in figure 6*(right)*. There is no speedup for two of the datasets (shown by stars) when only the candidate pruning method is active. The reason is that the amount of pruning achieved for these datasets cannot surpass the overhead costs of computing the upper bounds for every candidate. However, when the technique of efficient distance computation is active, speedups are achieved for *all* of the datasets including these two.

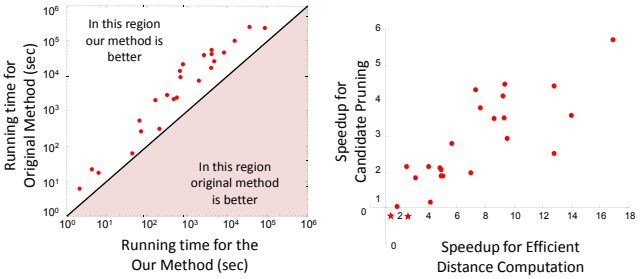

Figure 6: *(left)* Comparison of running times between our method and the original shapelet algorithm. Note the log scale on both axes. *(right)* The individual speedup factors for both of our proposed techniques: Candidate Pruning and Efficient Distance Computation.

It is critical to note that our improvements in speed are not due to trivial differences in hardware or to simple implementation optimizations, but reflect the utility of the two original techniques introduced in section 3. In particular, all experiments were done on exactly the same environment and on the same input data files. The code for the original shapelet discovery algorithm was graciously donated by Dr. Ye who also confirmed that we are using her code in the best possible manner. Since our technique reduced the worst case complexity by a factor of m and has an admissible pruning technique which is not present in the original code, we can be sure that the speedup is valid.

As described in section 3.2, our linear time upper bound is not admissible for the many-class cases. Among the 24 datasets we used for scalability experiments, 13 datasets have more than two classes. For these 13 datasets, the average rate of computing false upper bound is just 1.56% with a standard deviation of 2.86%. In reality, the impact of false bounds on the final result is inconsequential because of the massive search space for shapelet. Our algorithm rarely misses the optimal information gain in that space and has not missed in *any* of the above 13 many-class datasets.

6. CASE STUDIES

In this section we show three case studies in three different domains. In all the case studies we demonstrate that logical combinations of shapelets can describe the difference between the classes very robustly. We compare our algorithm to the 1-NN classifier using Dynamic Time Warping (DTW) because a recent extensive comparison of dozens of time series classification algorithms, on dozens of datasets, strongly suggests that 1-NN DTW is exceptionally difficult to beat [3]. Note that the 1-NN classifier using DTW is less amenable for realtime classification of time series since it requires an $O(m^2)$ distance measure to be computed for every instance in the dataset. In contrast, classification with shapelets requires just a single $O(n(m-n))$ calculation (n is the length of the shapelet). Thus, classification with time series shapelets is typically thousands of times faster than 1-NN DTW. We do not experimentally show this here, since it was amply demonstrated in the original shapelet paper [19].

6.1 Cricket: Automatic Scorer

In the game of cricket (a very popular game in British Commonwealth countries), an umpire signals different events in the game to a distant scorer/book-keeper. Typically, the signals involve particular motions of the hands. For example, the umpire stretches up both the hands above the head to signal a score of "six." A complete list of signals can be found in [1]. Researchers have recently begun to attempt to classify these signals automatically to ease/remove the task of a scorer [10].

In [10], a dataset was collected in which two accelerometers have been placed on both wrists of four different umpires. The umpires performed twelve different signals used in the game of cricket at least ten times. For simplicity of presentation, we select only two classes from the dataset that has a unique possibility of confusion. The two classes are "No Ball" and "Wide Ball." To signal "No Ball," the umpire must raise *one* of his hands to the shoulder-height and keep his hand horizontal until the scorer confirms the record. To signal "Wide Ball," the umpire stretches both of the hands horizontally at shoulder-height (see figure 7).

Each accelerometer has three synchronous and independent measurements for three axes (X,Y, and Z). For every signal performed, the six channels are concatenated to form one time series. We append low variance white noise to each example in the dataset to make them of length 308. The training set has nine examples as shown in figure 7. The test set has 98 examples. Note that the examples for "No Ball" are only right hand signals. This is true for the test set also. To cover the left handed case and also to include examples of different lengths, we have generated another test set using the accelerometer on a standard smart phone. This test set contains 64 examples of length 600. Each class has an equal number of examples in both of the training sets.

On this dataset, we performed 1-NN classification using Euclidean distance and Dynamic Time Warping (DTW) as distance measures. We also considered DTW with the Sakoe-Chiba band [15], as it has been shown to outperform classic DTW on many

Algorithms	Original Test set	New Test set
1-NN Euclidean distance	94.89%	56.25%
1-NN Dynamic Time Warping	**98.98**%	87.50%
1-NN DTW-Sakoe-Chiba	94.89%	75.00%
Shapelet	44.89%	48.43%
Logical Shapelet	95.91%	**89.06**%

Table 1: The accuracies of different algorithms on the two test sets.

datasets [7]. The results are shown in table 1. It is interesting to consider why Logical Shapelets generalize the new data the best. We conjecture that it is the ability of Logical Shapelets to extract just the meaningful part of the signal. In contrast, DTW and Euclidean distance must account for the entire time series, including sections that may be unique to individual umpires idiosyncratic motion, but not predictive of the concept.

The Computationally expensive 1-NN DTW performs well in both of the cases, but not suitable for real-time applications. The original shapelet algorithm fails to capture the fact that the inherent difference between the classes is in the number of occurrences of the shapelet. Our logical shapelet algorithm captures the sharp rises in the Z-axes for "Wide Ball" from the original training set. No such "No Ball" signal can have two such rises in the Z-axes, and therefore, classification accuracy does not decrease significantly for the new test set.

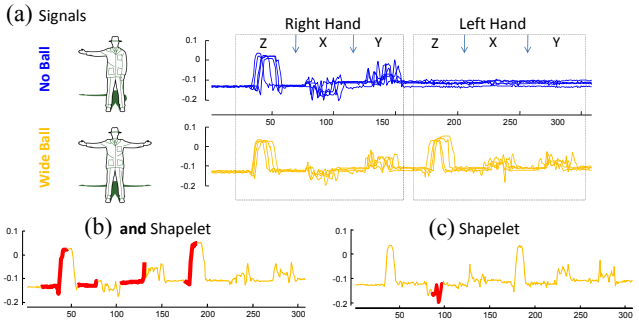

Figure 7: (a)The training set of the cricket dataset by concatenating signals from every axis of the accelerometer. (b) The two signs an umpire performs to declare two types of illegal delivery. (c) Shapelets found by our algorithm and the original algorithm.

6.2 Sony AIBO Robot: Surface Detection

The SONY AIBO Robot is a small, dog-shaped, quadruped robot that comes equipped with multiple sensors, including a tri-axial accelerometer. We consider a dataset created by [17] where measurements of the accelerometer are collected. In the experimental setting, the robot walked on two different surfaces: carpet and cement. For simplicity we consider only the X-axis readings. A snap shot of the data is shown in figure 8(a). Each time series represents one walk cycle. Cemented floors are hard compared to carpets and, therefore, offer more reactive force than carpets. As such, there are clear and sharp changes in the acceleration on the cemented floor. In addition, there is a much larger variability when the robot walks on cement.

The test set has 20 instances of walk cycles on the two types of floors. The training set has 601 instances. A walk cycle is of length

Algorithms	Surface Detection	Passgraphs
1-NN Euclidean distance	69.55%	63.36%
1-NN Dynamic Time Warping	72.55%	71.76%
1-NN DTW-Sakoe-Chiba	69.55%	**74.05**%
Shapelet	93.34%	60.31%
Logical Shapelet	**96.34**%	70.23%

Table 2: The accuracies of different algorithms on the passgraph trajectories and accelerometer signals from SONY AIBO robot.

70 at 125 hertz. We have experimented on this dataset and found a pair of shapelets shown in figure 8(b). The shapelets are connected by $\bigwedge$ and come from the two different shifts-of-weight in the walk cycle on the carpet floor. The pair of shapelets has a significantly higher classification accuracy compared to classic nearest neighbor algorithms (see table 2).

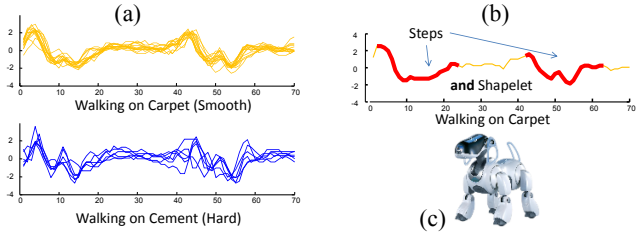

Figure 8: (a) Two classes of time series from the SONY AIBO accelerometer. (b) The *and*-shapelets from the walk cycle on carpet. (c) The Sony AIBO Robot.

6.3 Passgraphs: Preventing Shoulder-Surfers

Passgraphs are a recently proposed biometric system used to authenticate a person and allow her access to some resource. A grid of dots is presented to the user and she is tasked to connect some of the dots in the grid in some specific order as a password. In contrast to text-based passwords where the user can shield the the entered text (at an ATM for example), Passgraphs are vulnerable to "shoulder-surfing" as it is easy for a miscreant to see and memorize the connection sequence over the shoulder of the user. To prevent the shoulder-surfing attack, [12] has proposed methods involving pen-pressure. There are also methods based on other pen properties such as friction, acceleration, etc.

In this case study, we use the data from [12] to see if logical shapelets can classify between the same pen sequences performed by different users. We selected x-axis trajectories of two different users drawing the same passgraph. The logical shapelet shown in figure 9 consists of tiny fragments of the three turns (peaks in the time series) in the passgraph connected by $\bigvee$ operations. These shapelets have better accuracies than 1-NN classifier and are promising enough to use in a real authentication system. This is because the shapelets can be learned at training time when the user sets the password, and it is not possible for the shoulder surfer to mimic the idiosyncratic pen path of the real user when attempting to impersonate her. Note that, $\bigwedge$ operations in this case would have imposed a harder rule for the user to produce *all* instead of *some* of the shapelets exactly to get authenticated.

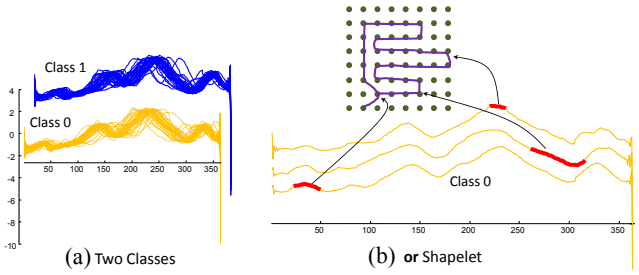

Figure 9: (a) Two classes of X-axis trajectories drawn by different users. (b) The *or*-shapelets from three different examples of class 0 showing three turns in the passgraphs.

7. CONCLUSIONS

In this paper, we introduce logical shapelets: a new time series classification primitive with more expressiveness than classic shapelets. We have demonstrated the existence of logical concepts in time series datasets, and the utility of logical shapelets in domains as diverse as gesture recognition, robotics and user authentication. We further describe novel techniques to efficiently find both classic and logical shapelets. Our approach is significantly faster for every one of the twenty-four datasets we tested.

Acknowledgements: Thanks to all the donars of the datasets. This work was funded by NSF awards 0803410 and 0808770.

8. APPENDIX: PROOF OF THEOREM 1

PROOF. Lets assume c is a left-major class. So $\frac{n_{c,1}}{N_1} > \frac{n_{c,2}}{N_2}$. If we move one point of class c from the right partition to the left partition, for $i = 1, 2, \ldots, C$ the change in information gain is ΔI.

$$\Delta I = \sum_i \frac{n_{i,1}}{N} \log \frac{n_{i,1}}{N_1} + \sum_i \frac{n_{i,2}}{N} \log \frac{n_{i,2}}{N_2} - \sum_{i \neq c} \frac{n_{i,1}}{N} \log \frac{n_{i,1}}{N_1+1} - \sum_{i \neq c} \frac{n_{i,2}}{N} \log \frac{n_{i,2}}{N_2-1} - \frac{n_{c,1}+1}{N} \log \frac{n_{c,1}+1}{N_1+1} - \frac{n_{c,2}-1}{N} \log \frac{n_{c,2}-1}{N_2-1}$$

Using $\sum_{i \neq c} n_{i,1} = N_1 - n_{c,1}$ and $\sum_{i \neq c} n_{i,2} = N_2 - n_{c,2}$, we can simplify the above as below.

$$\begin{aligned} N.\Delta I = {} & N_1 \log N_1 - n_{c,1} \log n_{c,1} + (n_{c,1}+1) \log(n_{c,1}+1) \\ & + N_2 \log N_2 - n_{c,2} \log n_{c,2} + (n_{c,2}-1) \log(n_{c,2}-1) \\ & - (N_1+1) \log(N_1+1) - (N_2-1) \log(N_2-1) \end{aligned}$$

Since, $\frac{n_{c,1}}{N_1} > \frac{n_{c,2}}{N_2}$ for $t \in [0,1]$ the following is also true $\frac{n_{c,1}+1-t}{N_1+1-t} > \frac{n_{c,2}-t}{N_2-t}$. In addition, $n_{c,1}, N_1, n_{c,2}, N_2$ are all positive integers; therefore, we can take log on both side and integrate from 0 to 1.

$\int_0^1 \log \frac{n_{c,1}+1-t}{N_1+1-t} \mathrm{d}t - \int_0^1 \log \frac{n_{c,2}-t}{N_2-t} \mathrm{d}t > 0$. If we evaluate the integral, it becomes the right-side part of the above equation for $N.\Delta I$. Therefore, $\Delta I > 0$. □

9. REFERENCES

[1] *BBC SPORT | Cricket | Laws & Equipment*. http://news.bbc.co.uk/sport2/hi/cricket/rules_and_equipment.

[2] *Supporting webpage containing code, data, excel sheet and slides.* http://www.cs.ucr.edu/∼mueen/LogicalShapelet.

[3] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1:1542–1552, 2008.

[4] P. Domingos. Process-oriented estimation of generalization error. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 714–719, 1999.

[5] B. Hartmann and N. Link. Gesture recognition with inertial sensors and optimized dtw prototypes. In *IEEE International Conference on Systems Man and Cybernetics (SMC)*, pages 2102 –2109, 2010.

[6] B. Hartmann, I. Schwab, and N. Link. Prototype optimization for temporarily and spatially distorted time series. In *the AAAI Spring Symposia*, 2010.

[7] E. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 406–417, 2002.

[8] E. Keogh and S. Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371, 2003.

[9] E. Keogh, X. Xi, W. L., and C. A. Ratanamahatana. *The UCR Time Series Classification/Clustering Homepage:*. www.cs.ucr.edu/∼eamonn/time_series_data/.

[10] M. H. Ko, G. West, S. Venkatesh, and M. Kumar. Online context recognition in multisensor systems using dynamic time warping. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005.*, pages 283 – 288, 2005.

[11] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[12] B. Malek, M. Orozco, and A. E. Saddik. Novel shoulder-surfing resistant haptic-based graphical password. In *In the Proceedings of the EuroHaptics conference*, 2006.

[13] A. McGovern, D. Rosendahl, R. Brown, and K. Droegemeier. Identifying predictive multi-dimensional time series motifs: an application to severe weather prediction. *Data Mining and Knowledge Discovery*, 22:232–258, 2011.

[14] A. Mueen, S. Nath, and J. Liu. Fast approximate correlation for massive time-series data. In *SIGMOD Conference*, pages 171–182, 2010.

[15] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43 – 49, 1978.

[16] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD Conference*, pages 599–610, 2005.

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

[18] Z. Xing, J. Pei, P. Yu, and K. Wang. Extracting interpretable features for early classification on time series. In *the Proceedings of SDM*, 2011.

[19] L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD, pages 947–956, 2009.