Shapelet Ensemble for Multi-dimensional Time Series

Mustafa S Cetin
Computer Science Department,
University of New Mexico
mscetin@unm.edu

Abdullah Mueen
Computer Science Department,
University of New Mexico
mueen@unm.edu

Vince D. Calhoun
Electrical and Computer Engineering Department,
University of New Mexico
& The Mind Research Network
vcalhoun@mrn.org

***Abstract*—Time series shapelets are small subsequences that maximally differentiate classes of time series. Since the inception of shapelets, researchers have used shapelets for various data domains including anthropology and health care, and in the process suggested many efficient techniques for shapelet discovery. However, multi-dimensional time series data poses unique challenges to shapelet discovery that are yet to be solved.**

We show that an ensemble of shapelet-based decision trees on individual dimensions works better than shapelets defined over multiple dimensions. Generating a shapelet ensemble for multi-dimensional time series is computationally expensive. Most of the existing techniques *prune* shapelet candidates for speed. In this paper, we propose a novel technique for shapelet discovery that *evaluates* remaining candidates efficiently. Our algorithm uses a multi-length approximate index for time series data to efficiently find the nearest neighbors of the candidate shapelets. We employ a simple skipping technique for additional candidate pruning and a voting based technique to improve accuracy while retaining interpretability. Not only do we find a significant speed increase, our techniques enable us to efficiently discover shapelets on datasets with multi-dimensional and long time series such as hours of brain activity recordings. We demonstrate our approach on a biomedical dataset and find significant differences between patients with schizophrenia and healthy controls.

I. Introduction

Time series shapelets are small segments of time series that distinguish between classes based on existence of such segments in the classes. Figure 1 shows an example of a shapelet in real ECG data. The idea of shapelet discovery has gained popularity for its intuitive classification rules. Shapelets are not just useful for classification purposes, any dataset of long time series can potentially be represented by a set of shapelets as such the shapelets perform an empirical basis [1]. Shapelets have also been used for unsupervised knowledge discovery such as clustering [2].

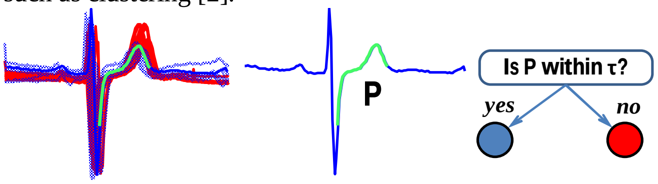

Figure 1: ***An example of shapelet. (left) Heartbeats of a 67 year old male in two different days shown in blue and red. Red has a higher peak than the blue at the T wave. (middle) The shapelet (P) that distinguishes the classes most. (right) A highly accurate decision tree using the shapelet P.***

Despite numerous works on shapelet discovery, mostly on efficient algorithms, shapelets for multi-dimensional time series data are yet to be explored because of the added computational requirement for multiple dimensions of time series data. In the simplest form, if we wish to discover shapelets for each of the dimensions separately, the original exact algorithm [3] takes months while the fastest approximate algorithm would take ten hours on our target dataset of Electroencephalogram. In this work, we overcome the barrier of computational requirements by fast candidate *generation* and *evaluation*. The new algorithm allows us to generate a simple ensemble of shapelets from individual dimensions. In Figure 2, we show test accuracies on a dataset (functional MRI) of 540 dimensions recorded from 43 healthy controls and 32 schizophrenic patients. Individual shapelet classifiers perform similar to a random classifier for lack of enough information, while the ensemble achieves a significant accuracy. Our experiments show that ensemble of shapelet classifiers for individual dimensions of a multi-dimensional time series data is promising for electrical sensors such as EEG and Accelerometers.

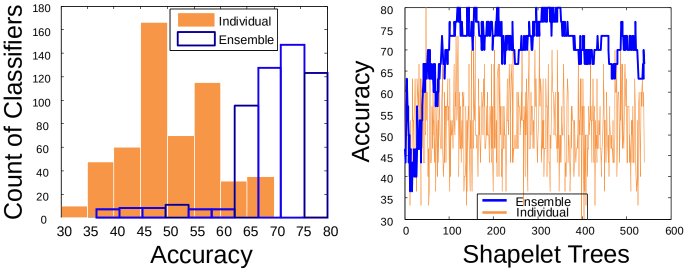

Figure 2: ***Ensembling can achieve better accuracies than all individual trees on unseen test data. (left) Shift in the distribution of classifiers when shapelet trees are ensemble. (right) Each blue point is the accuracy from an ensemble of all the individual trees to its left.***

A typical shapelet discovery algorithm works in two phases. First, the algorithm generates a set of candidate shapelets and next, the candidates are evaluated for the information gain they achieve when used as classification features. Surprisingly, all of the speed-up techniques in the literature prune the candidate pool by admissible heuristics [4] or by random projections [5]. In this paper, we describe a fast candidate evaluation algorithm using multi-length indexing scheme, which can be used in conjunction with any prior algorithm.

Our algorithm, named *mc*^{2}, uses a dynamic stepping technique for massive candidate pruning. It achieves an order of magnitude speed-up (up to 9.58x) over current state of the art algorithm [5] and several order of magnitude speed-ups (up to 281x) over the original algorithm [3] without having a significant difference in accuracy. Moreover, the increased speed allows us to train shapelet classifiers for every dimension of a multi-dimensional time series for an ensemble classifier and thus, achieve increased accuracy and reduced variance.

The paper is structured as follows. Section 2 introduces the definitions and notations. In section 3, we described previous and related work. In section 4, the skeleton of the shapelet algorithm is discussed. Section 5-6 describes the changes we have made to the shapelet algorithm and propose a way to speed up shapelet algorithm, and section 7, we demonstrate the performance and accuracy of our algorithm with 44 data sets. Section 8 shows the case studies. Finally, in Section 8, we form our conclusions.

II. Defination and Notation

We start with defining shapelets and the other notations used in the paper.

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 *m(m+1)/2* subsequences of all possible lengths from one to *m*. If we are given two time series *X* and *Y* of the same length *m*, we can use the Euclidean norms of their difference (i.e. $X - Y$) as a distance measure. To achieve scale and offset invariance, we must normalize the individual time series before the actual distance is computed and we can do this via *z-normalization*. Normalization is a critical step; even tiny differences in scale and offset rapidly swamp any similarity in shape [6]. 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, the time for computing a distance value is linearly related to the length of the time series. In contrast, we compute the normalized Euclidean distance between *X* and *Y* using five numbers derived from *X* and *Y*. These numbers are denoted as sufficient statistics in [7]. The numbers are $\sum x, \sum y, \sum x^2, \sum y^2$ and $\sum xy$. As it was made clear in [4], 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 $dist(x, y) = \sqrt{2(1 - \mathrm{C}(\mathrm{x}, \mathrm{y}))}$ where $C(x, y) = \frac{\sum xy - m\mu_x\mu_y}{m\sigma_x\sigma_y}$).

Many time series data mining algorithms (K-NN classification, clustering, density estimation, etc.) require only comparisons of time series that are of equal lengths. The major reason is because the underlying distance metrics they rely upon do not allow varying 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 $sdist(x, y) = \sqrt{2(1 - \mathrm{C_s}(\mathrm{x}, \mathrm{y}))}$ where *x* and *y* are the two time series with lengths *m* and *n*, respectively, and for $m \leq n$.

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

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, . . . ,*C*) has n_i labeled instances in the dataset where $\sum_i n_i = n$. An instance time series in *D* is also denoted by D_i for *i* = 1, 2, . . . , *n*. 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 use two quantities to measure the goodness of a split: *information gain* and *separation gap*.

Definition 1. The information gain of a split is

$$I(s, \tau) = E(D) - \frac{|D_{left}|}{N}E(D_{left}) - \frac{|D_{right}|}{N}E(D_{right})$$

Definition 2. The separation gap of a split is

$$G(s, \tau) = \frac{1}{|D_{right}|}\sum_{x \in D_{right}} sdist(s, x) - \frac{1}{|D_{left}|}\sum_{x \in D_{left}} sdist(s, x)$$

Definition 3. The shapelet for a dataset *D* is a tuple (s, τ) of a subsequence of an instance in *D* and a distance threshold (i.e. a split, τ) that has the maximum information gain while breaking ties by maximizing the separation gap.

We visually summarize the concept of shapelets with our toy example shown in Figure 3. Shapelets can be organized in a decision tree format until the training set achieves desired level of representation.

To extend the definition of shapelets to multi-dimensional time series data, several aspects should be addressed. Do we find shapelets in *all* dimensions or in *some* of them? Will the shapelets be *time aligned*? Will they be equally *long*? And most importantly, how do we build a *classifier* using the shapelets? We answer these questions with an *independence assumption* among the dimensions of the time series. Under this assumption, it is sufficient to discover shapelets of arbitrary *size* and *location* for *every* dimension and *ensemble* them instead of one giant tree containing them.

Definition 4. Shapelets for a multi-dimensional time series data *D* is a set of shapelet-based decision trees on individual dimensions that maximizes training accuracy upon ensembling.

III. Previous and Related Work

Since the inception of shapelets, various shapelet discovery algorithms have been proposed to improve brute force approach. Lexiang *et.al.* [3] introduced the first improvement by proposing a technique for abandoning some unfruitful entropy computations early. However, this does not improve the worst case complexity. Mueen *et al.* [4] reduced the worst case complexity by caching distance computations for future use and using a triangular inequality based pruning strategy that achieves an order of magnitude speedup over the method of Lexiang *et al.* [3]. Both of these methods achieved admissible pruning and thus retained the exactness of the gain maximization.

Rakthanmanon *et al.* [5] used a random projection technique [8], [9] using the SAX representation [10], [11] to find potential shapelet candidates sacrificing the exactness for speed. This algorithm separates the candidate generation and evaluation process by first generating a *constant number* of candidates and then, evaluating them for an overall computation time of $O(nm^2)$ while retaining significant accuracy. Our proposed algorithm builds upon the *FastShapelet* [5] algorithm. Our algorithm skips candidates in the candidate generation process heuristically while improving the candidate evaluation process to reduce running time significantly. We compare our algorithm with the *FastShapelet* algorithm as it is the current *state of the art* for shapelet discovery.

Chang, K.-W. *et al.* [12] improved a dynamic programming algorithm for highly parallel Graphics Process Units (GPUs). Results showed that the proposed GPU implementation significantly reduces the running time of the shapelet discovery algorithm. Lines, J. *et al.* [1] have used several features based on shape-similarity and named it as shapelet transformation. Although shapelet transform has competitive classification accuracy, it loses the interpretability of the decision tree based classifiers over individual shapes.

To the best of our knowledge, ours is the first attempt to generalize shapelet discovery to multi-dimensional time series data. Our multi-length indexing scheme for time series data is the first of its kind. Previous work in multi-resolution indexing for images [13] use more space to cache different resolution while our method does not use any extra space.

Table-1. Brute Force Algorithm

```
Algorithm 1 Shapelet_Discovery(D)
Require : A dataset D of time series
Ensure  : Return the shapelet
1:  max_length = maximum length of a time series in D
2:  max_gain = 0, min_gap = 0, m = maximum shapelet length
3:  for j = 1 to |D| do {every time series in D}
4:    S = D_j
5:    for l = 1 to m do {every possible length}
6:      for i = 1 to |S| − l + 1 do {every start position}
7:        for k = 1 to |D| do {compute distances of every time
          series to the candidate shapelet S_{i,l}}
8:            L_k = sdist(S_{i,l}, D_k)
9:        sort(L)
10:       [gain,gap] = CalculateInfoGain(L)
11        if (gain > max_gain) or ((gain == max_gain) and
                                         (gap > min_gap))
12:       then    max_gain = gain, min_gap = gap,
13:               bestshapelet = S_{i,l}, bestτ = τ
```

IV. One Dimensional Shapelet Discovery

In order to properly explain our contribution, we define the brute-force algorithm for shapelet discovery and refine the algorithm in progression.

The brute force shapelet discovery algorithm shown in Table-1 is a simple algorithm that generates and tests all possible candidates and returns the best one. The final shapelet can be of any length, all subsequences of every length in the dataset D is generated as candidate subsequences $S_{i,l}$ in the three loops in lines 3, 5, and 6 of algorithm 1. In lines 7-9, an array L is created which holds the points in D in the sorted order of their subsequence distance from the shapelet candidate. Finally, the information gain is computed in line 10 and returns the candidate with maximum information gain.

For each of the candidates, we need to compute subsequence distances to each of the n time series in D (line 8) by using a distance function *sdist* which essentially finds the nearest neighbor distance between the candidate and subsequences of the instance time series. Note that *sdist* is the inner-most statement of the above algorithm and thus, a slight improvement in efficiency will give us a large payoff. Figure 3 shows how the *sdist* between a candidate and n (=8) instances are organized in the array L.

As shown in Table-1, in line 7-9, an array L is created which holds the points in D in the sorted order of their distance from the shapelet candidate. 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.

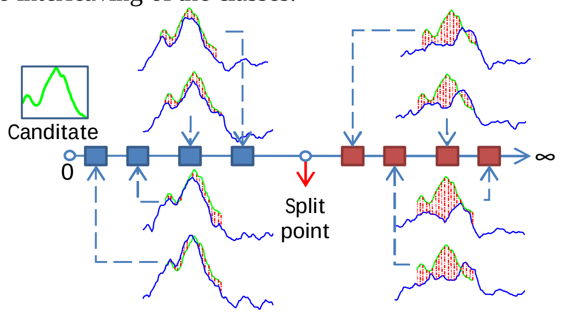

Figure 3: ***Instances are sorted based on the 1NN distance (sdist) from the candidate. A split point is found to group the instances maximizing the information gain.***

A distance computation between a candidate and an instance time series may take $O(m^2)$ time in the worst case and for all the instances it can take up to $O(nm^2)$ time. Once we know the distances, the computation of the information gain takes a linear scan to try different split points. Therefore, the candidate evaluation process is dominated by the $O(nm^2)$ complexity for the subsequence distance computation. Since there are at least $n\frac{m(m+1)}{2}$ shapelet candidates in the dataset, total number of all candidates in the D is $O(nm^2)$. Being brute force, the algorithm *generates* and *evaluates* all the candidates and thus, needs $O(n^2m^4)$ running time. Such computational cost makes the brute-force algorithm infeasible for long time series.

The state-of-the-art algorithm reduces the candidate generation phase to a simple linear scan with random-projection [5] and thus, the overall complexity reduces to $O(nm^2)$.

V. Speed-up Techniques

In order to speed up the state of the art, we propose two techniques. First, we improve the candidate evaluation phase by variable length indexing. Second, we reduce the computation in the candidate generation part by dynamic stepping.

A. *Multi-length Indexing*

The candidate evaluation process needs *$O(nm^2)$* time to compute the distances between the candidate and the instances of the data. A distance between a candidate and an instance is essentially identical to finding the nearest neighbor of the candidate among the subsequences of the instance. There have been numerous research studies on how to use indexing techniques to efficiently search for the nearest neighbor of a time series [14], [15]. Yet indexing is not adopted for shapelet discovery because it requires querying nearest neighbors for different lengths of time series. There is no efficient technique to build a data structure that can serve multiple length queries in runtime and therefore, the cost of building indexes for every length becomes unrealistic. In this paper, we introduce a very efficient multi-length indexing scheme that can be used across queries of multiple lengths. Multi-length index reduces the number of distance computation significantly from the state of the art. We start explaining the technique with the description of a very simple indexing structure, *order-line.*

Order-Line: An order-line is a sorted sequence of Euclidean distances of the subsequences of an instance from a random pivot point R. In the Figure 4, we show a schematic view. The idea behind choosing a pivot point, R and projecting the objects (time series in our case) to a 1-D line has already been used by a plethora of dimensionality reduction algorithms [16] and specifically for searching in time series data. This ordering of the objects about R provides us with some useful heuristic information to find minimum distance for a candidate shapelet. The intuition is that if two objects are close in the original space, they must also be close in this ordering or two objects can be arbitrarily close in the linear ordering but very far apart in the original space. More precisely, the ordering has two very useful properties. First, we can use triangular inequality (centered at R) to produce a lower bound for the distance between any pair of points. Second, there is an implicit order of the objects based on lower bounds. For example, the lower bound between Q and C (i.e. $\sqrt{25}-\sqrt{18}$) is more than the lower bound between Q and D (i.e. $\sqrt{18}-\sqrt{10}$). The order lines for each instance time series can be calculated during the first candidate evaluation assuming the candidate as the pivot point and thus, it requires no additional cost.

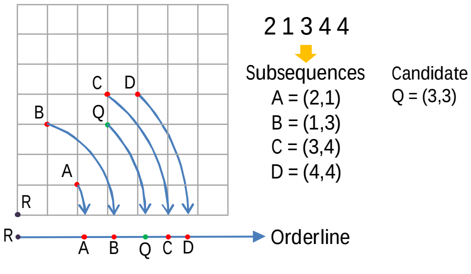

Figure 4: ***Projection of subsequences of a toy time series 2 1 3 4 4 to one-dimensional ordering. Subsequences are of length two. The pivot point is at the origin for simplicity. A candidate shapelet is Q.***

Finding the Minimum Distance: Given the order-line, how can we use this as an index to find the nearest neighbors quickly? The generic idea can be found in [7], [16].

Given a query, the algorithm first computes the one dimensional projection by taking a distance from R. To give an example, let's assume the projection value of the query be Q in the Figure 4. The algorithm will check points starting with the closest one in the line with an increasing order of distances from Q. In this example, the algorithm will check C, B, D and A, in this order. For each point, the algorithm computes the Euclidean distance with early abandoning and checks if the point is better than the closest discovered so far. If we encounter a point that has a distance less than the current best-so-far, we update the best-so-far. At any time, if the distance in one dimensional space (i.e. on the line) is larger than the best-so-far distance in original space, we stop checking (break point) as we have found the nearest neighbor exactly. Table-2 shows the steps for finding nearest neighbor of Q in the high dimensional space with some hypothetical distance values.

Table-2. Progress step of scanning

	Dist	1D Dist	MD Dist	bsf	1D dist>bsf
1	Q-C	$\sqrt{25}-\sqrt{18}$	1	1	No
2	Q-B	$\sqrt{18}-\sqrt{10}$	2	1	Yes
3	Q-D	$\sqrt{32}-\sqrt{18}$	$\sqrt{2}$	1	Yes
4	Q-A	$\sqrt{18}-\sqrt{5}$	$\sqrt{5}$	1	Yes

Bounding Distances for Longer Queries: In an order-line we have the lower-bounds of the distances between a candidate and the objects based on triangular inequality. The assumption is that the objects, R and the candidate are all in the same dimensionality or length. However, shapelet candidates can be of different lengths and for each of them, we will need its nearest neighbor distances to the instance time series as shown in Figure 4. Let's assume that we have an order-line for dimensionality *m* where the points and the reference point R, are in an *m*-dimensional space. If we have a new candidate with a larger length (i.e. more dimensionality) of *m+1*, how are we going to use the order-line to quickly find the nearest neighbor?

We take the first *m*-dimensions of the query and compute its position on the order-line. Let's assume, without losing the generality, that the *m*-prefix of the candidate is the object A in the Figure 4. We cannot simply say, as before, the lower bound between the subsequence D and the query Q in the *m+1* dimensional space is $\sqrt{18}-\sqrt{10}$. The reason is that the normalized distance between D and A in the *m+1* dimensional space does not increase or decrease monotonically with their normalized prefix distance in the *m*-dimensional or lower dimensional space. In a recent work [17] it has been shown that it is possible to lower bound distances upon extension. More precisely, if *x* and *y* are two time series of length *m* and x_{+1} and y_{+1} are the two one step extensions of them, respectively, then

$$d_{LB}^2(\widehat{\boldsymbol{x}}_{+1},\widehat{\boldsymbol{y}}_{+1}) = \frac{1}{\sigma_m^2} d^2(\widehat{\boldsymbol{x}},\widehat{\boldsymbol{y}}) \text{ where } \sigma_m^2 = \frac{m}{m+1} + \frac{m}{(m+1)^2} z^2 \text{ and}$$

$$\mathrm{z} = \max(abs(\widehat{\boldsymbol{X}}), abs(\widehat{\boldsymbol{Y}}))$$

In the above equations, the *X* and *Y* are the original sequences while *x* and *y* are subsequences. The hat operator is describing the normalization operation. The variable *z* represents the highest possible value that can appear next. The bound is trivially trued for raw vectors without normalization.

This result makes it possible to use one order-line for the next query length. In our running example, we know the lower bound distance between Q and D in the m-dimensional space and we can find a lower bound in the *m+1* dimensional space by multiplying 2 with a fraction $1/\sigma_m^2$. Thus the algorithm to find the nearest neighbor can continue until this lower bound is larger than the *best-so-far*.

Note that as we increase the level of extension such as *s*-step extension for *s* > 1, the lower bounding fraction can be repeatedly applied. For example, for 2-step extension, the lower bound can be

$$d_{LB}^2(\hat{\boldsymbol{x}}_{+2}, \hat{\boldsymbol{y}}_{+2}) < \frac{1}{\sigma_{m+1}^2} d_{LB}^2(\hat{\boldsymbol{x}}_{+1}, \hat{\boldsymbol{y}}_{+1}) = \frac{1}{\sigma_{m+1}^2}\frac{1}{\sigma_m^2} d^2(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) < \frac{1}{\sigma_m^4} d^2\ (\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$$

Thus, for queries of length *m+s*, we can use the *m*-prefix of the query to find the spot in the *order-line*. Once the spot is found, the lower bounds will be the distances on the *order-line* multiplied by the fraction as computed above.

Example: Let's take the example for lower bounding trick for longer queries without z-normalization. Let us assume a 3D candidate Q_1=(3,3,3) has a 2D prefix Q. The distance between Q and A is $\sqrt{5}$=2.2361 in Figure 4. The largest coordinate value *z*=4. Then $\sigma_m^2 = \frac{2}{3} + \frac{2*16}{9} = 4.2156$ and the lower bound between Q_1 and A=(2,1,3) is $\frac{\sqrt{5}}{\sqrt{4.2156}} = 1.089$.

The complete algorithm for finding the nearest neighbor using our multi-length indexing scheme is given in the Table 3.

Table-3. Finding Minimum Distance Algorithm

```
Algorithm 2 findMinDist(D,candidate,OrderLine)
Require:D        : A dataset of time series
candidate        : Shapelet candidate
OrderLine        : Distances of the subsequences toa reference
Ensure:  Return the most similar subsequence to the candidate
m=dimensionality of the OrderLine
S=length(candidate)-m, 1D=0, bsf=∞
Find the location l of the m-prefix of the candidate
    in the OrderLine
$f_c = 1/\sigma_m^{m*S}$
for each $D_i$ in the OrderLine in order of distances from l
    1D = (distance_in_Orderline between l and $D_i$) * $f_c$
    if bsf > 1D then bsf = EuclideanDistance(candidate,$D_i$)
  if 1D > bsf then return (bsf)
```

B. Dynamic Stepping

Multi-length indexing can reduce the computation time reasonably. We further improvise the algorithm with a very simple strategy without any significant accuracy degradation. Before we describe the technique, we introduce the complete *mc²* algorithm in Table-4.

The algorithm takes a time series dataset as input and three user defined parameters namely the minimum and maximum shapelet length and, the *usr* described later. The algorithm runs a loop over the possible lengths of the candidates in line 3 which is the outer-most loop. The algorithm generates a set of ten candidates for every length using the random projection technique described in the *FastShapelet* method [5] in lines 4-7. The loop at line 8 goes over each of the ten candidates to find the best one.

Our dynamic stepping algorithm uses a certain step size for the outer-most loop running over the lengths. The step size can be constant or variable. Constant step size has a disproportionate effect on shorter shapelets. For example, a step size of 2 may generate as low as 0.5 overlap between successive candidates starting at a certain location. To have a uniform effect, we vary stepping strategy.

Before we describe the strategy, we clarify that stepping is not the same as sampling the time series down to smaller size. If we down-sample, the dimensionality of the distances would change. In dynamic stepping, we skip some of the subsequences in systematic manner so the impact on accuracy is uniform.

Recall, we define a time series as a sequence of real numbers T = t_1, t_2, . . . ,t_m and the subsequence of length *j* as the $S_{i,j}=t_i,t_{i+1},\ldots t_{i+j-1}$. It has been well observed that there is a large overlap between $S_{i,j}$ and $S_{i,j+1}$ for any *i* making them very similar to each other (i.e. trivial matches). We define the ratio between two successive candidates starting at the same position as similarity ratio (*sr*) and use it to determine the size of the stepping. For a step size (*sz*), the similarity ratio is simply *j/(j+sz)* for any *j* within the minimum and maximum lengths specified by the user.

Table-4. *mc²* Shapelet_Discovery Algorithm

```
Algorithm mc²:Shapelet_Discovery(D,min_length,max_length,usr)
Require          : A dataset D of time series
Ensure           : Return the shapelet
1: maxGain = 0, maxGap = 0, k = 1, stp = 1
2: [qsum,qsum2] = cumulativeSums(D,min_length,max_length)
3: for sbsqlen = min_length : stp : max_length do
          {for every subsequence length at steps}
4:     SAXList = CreateSAXList(D,sbsqlen)
5:     RandomProjection(SAXList)
6:     ScoreAllSAX
7:     top_10_cand = FindBestSAX(10), max_gain = ∞, min_gap = 0
8:     for i=1:10
9:         cand = top_10_cand(i)
10:        if i == 1 {----------Multi Length Indexing -----------------}
11:            $f_c = 1/\sigma_m^{m*S}$
12:            if ($f_c$< 0.75 or $f_c$ ==1) then
                   OL=Linear_ordering(D,cand), $f_c$=1
13:        Q = EuclideanDistance(cand,referance_query)
14:        for i = 1 to |D|
15:            if (Q == 0 and $f_c$== 1) then L = min(OL)
16:            else L = findMinDist(D,cand,OL,$f_c$)
                   {-------------Multi Length Indexing------------}
17:        [gain,gap] = CallInfoGain(L)
18:        if (gain > max_gain) or
               ((gain== max_gain) and (gap>min_gap))
           then
19:            max_gain = gain, min_gap = gap,
               Bestshapelet = cand, $best\tau = \tau$
20: If sr ≥ usr then stp = $\lfloor j(1-usr)/(2usr-1) \rfloor$ {-Dynamic Stepping-}
```

The initial step size is 1 and it increases by one whenever it crosses a user defined maximum threshold (*usr*). See Figure 5. As long as *sr* < *usr*, we use the same step size. When *sr* ≥ *usr* holds, we increase step size. For example, *m=113*, *i=0*, *j=10*, *sz=1* and *usr=0.95* then *sr=j/(j+sz)=0.9091* and *sr* ≥ *usr* does not hold. Therefore we use subsequence length *j=j+sz* to create new shapelet candidates and keep on increasing *j* with *sz* until $sr \geq usr$. When $j = 19$, $sr = 0.95$ then we can increase *sz*. This operation can be easily implemented by setting $sz = \lfloor j(1-usr)/(2usr-1) \rfloor$.

Without any skipping, the number of different lengths is *m-minLength+1* which is the number of iteration for a time series of length *m* in the original algorithm. By using skipping technique we can reduce the number of iteration. We can define optimization rate as *(m-minlng+1) / (total iterations in our algorithm).*

usr can be thought of as a control for the speedup-accuracy tradeoff. Using a lower *usr* value increases optimization rate but using very low *usr* value may cause missing the best shapelet candidate, especially for short subsequence lengths. Also dynamic stepping gives better optimization rate for longer subsequence length because *sr* needs less number of iteration to satisfy $sr \geq usr$.

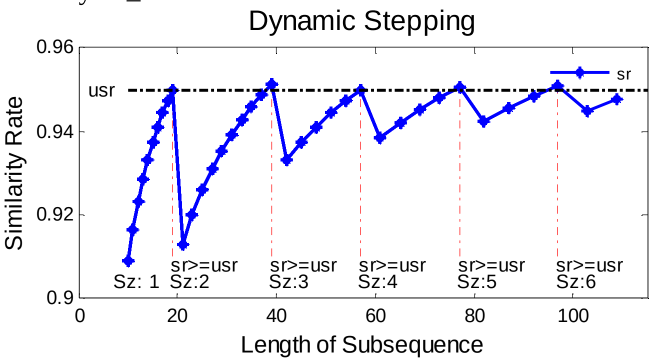

Figure 5: ***Increasing step size for short subsequences and more for long subsequences for maximum optimization***

VI. Voting Based Ensembling

With the speedup techniques described above, we could find a set of shapelet trees for each of the dimensions of a multidimensional time series.

Dietterich TG. *et al.* [18] have shown that teaming up a set of decision trees based on majority voting can reduce the machine learning bias of the classifier. We employ the same technique with an assumption that individual dimensions are independent. Thus counting votes from the classifiers is the way to generalize.

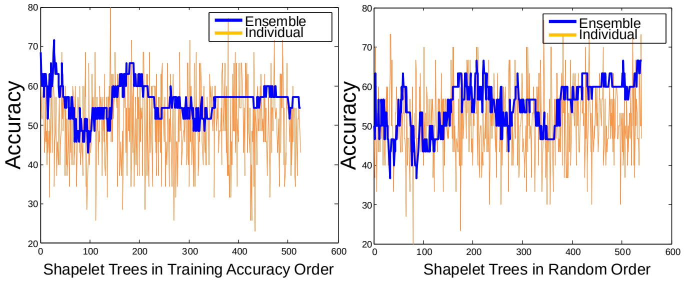

Figure 6: ***Accuracy of all-ensemble method is higher on average.***

An important question is *"Should we ensemble all the trees?"* As shown in Figure 2, if we ensemble all of them we converge to a lower accuracy than what could be achieved by a smaller number of trees. We experiment to test if the training accuracies can guide us to find the smallest ensemble of highest accuracy. We find there is *no strong correlation* between training accuracy of individual trees with the overall accuracy of the ensemble. Therefore, we use *random ordering* of the shapelet trees and ensemble *all* trees for an average gain over individual trees. Figure 6 shows two different orderings, where the left one achieves a tiny gain on average and misses the maximum gain. The right one achieves the maximum gain.

Other than multi-dimensional data, we can also use ensembling for one-dimensional data. The random projection part for candidate generation introduces randomness in the accuracy of the decision tree unlike the exact methods. We can run the same algorithm on the same data to generate a set of trees. We investigate the accuracy for various datasets before and after we combine the classifiers via voting. We spend roughly the same amount of time to generate decision tree(s) by both of the algorithms. Since ours is faster, it generates more trees than the state of the art algorithm. The accuracies in these two sets show a significant difference just because of the number of trees. We claim that our algorithm can generate confident classifiers than the state-of-the-art shapelet discovery algorithm by having lower variance.

There exists the method called *shapelet transform,* which finds k-shapelets in the first round to build a forest of single node trees. Shapelet transform has been shown to have larger accuracies than the exact method and it is faster because of no recursive decision tree building. However, our method is different from shapelet transform in that we produce multi-node trees, which are complete classifiers. Such trees retain the original motivation of shapelets, interpretability, while shapelet transform uses many shapelet candidates to transform the data to a new feature space, which makes it difficult to interpret in the original space.

Complexity of the Algorithm: Multi-length indexing reduces the time to search for nearest neighbor. Because of curse of dimensionality the worst case complexity is still $O(nm^2)$. On average, the complexity becomes better with large data making the method more scalable. Dynamic stepping does not reduce the worst case complexity either, although it speeds up the algorithm by a constant factor.

VII. Experimental Resuls

In order to evaluate the performance of our algorithm, we present the results of our experiments and compare them with the current state of the art (*FastShapelet*) algorithm (a heuristic algorithm) [5] and original shapelet algorithm [3]. We use the code from the authors' webpage [3], [5] and run these algorithms and our algorithm on the same device. We use the same parameters as suggested to have maximum accuracy. All of the code and the datasets used in this study are available at [19].

A. Time Series Datasets

In this study, we introduce 2 new time series datasets collected at the Mind Research Network at University of New Mexico. The first dataset is from a pre-attentional sensory processing experiment using fMRI scans of 32 schizophrenia patients (SP) and 43 healthy controls (HC). The second dataset is from an experiment on multi-modal sensory integration for fMRI scan on the same set of patients and control. To obtain the time series of components for first and second data sets, we use the GIFT Toolbox (http://mialab.mrn.org/software/gift/) and infomax algorithm [20] for group independent component analysis [21]. We perform classification on both of these datasets to classify schizophrenia patients and healthy control.

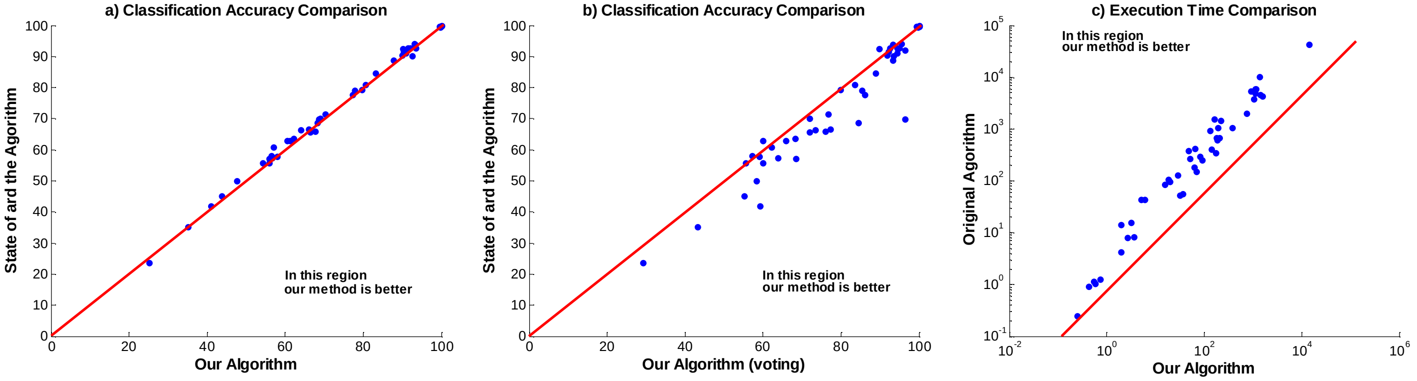

Figure 7: ***(a) Accuracy comparison between our algorithm and the current state-of-the-art algorithm (b) Accuracy comparison when we use voting based ensemble (c) Execution time comparison between our algorithm and the current state of the algorithm.***

In addition to the novel datasets, we use 41 datasets from the UCR time series archive [22] to compare the efficiency of our algorithm to the state-of-the-art algorithm. We test on all the 43 datasets on original algorithm; however, we abandoned the experiments (14 data sets) in which the original algorithm [3] had not finished after 18 hours. Comparison with the original algorithm is presented in our supporting webpage [19]. For UCR time series data sets, we used a train and test split ratio as suggested in their webpage [22]. For the two new datasets, we used 66% for training (66% SP and 66% HC) and 33% for testing (33% SP and 33% HC). Subjects in the training and testing sets are chosen randomly.

B. *Experimental Settings*

For all the experiments, we used the same hardware (Intel(R) Core(TM) i7 CPU 860@ 2.80 GHz, 1.59GHz, 2.96 GB of RAM). The parameters, minimum length and maximum length, are set to 10 and 250, respectively (If the length of the time series is less than 250, maximum shapelet length is set to the length) and the upper bound of similarity rate (*usr*) is 0.95.

C. *Performance Comparison on One Dimensional Data*

We use accuracy and execution time parameters to compare performance of our algorithm. We used the average of 20 runs for all data sets and for both algorithms. More detailed results are available at [19].

1) Accuracy: We perform the experiment on the 43 datasets for *FastShapelet* (current state of the art) algorithm and 30 data sets for the original algorithm. Our speedup techniques do not target improvement in accuracy and we do not expect any significant change in accuracy due to skipping some of the lengths. Figure 7(a) shows the comparison between our algorithm and the *FastShapelet* algorithm.

We use paired t-tests on the accuracy results. The cut off P-value for all of the tests is set at $P < 0.05$. The results confirm no significant difference in accuracy.

We perform an experiment to validate the goodness of the voting based ensembling we propose. In Figure 7(b), the comparison is shown and the paired t-tests on the accuracy results showed significant increase in accuracy at $P \geq 0.05$ for *FastShapelet* algorithm and the original algorithm.

2) Execution time: On all of the 43 data sets, we record the running time of the algorithms and compare them in the Figure 7(c) with log scale. As claimed before, we achieved an order of magnitude (up to 9.58x) speeds up over the current state of the art algorithm and several order of magnitude speeds up (up to 281x) over the original algorithm.

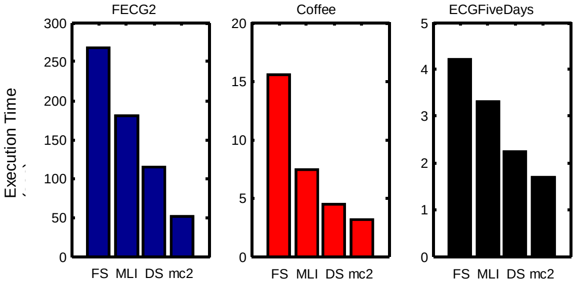

Figure 8: ***Individual execution time (sec) of three different data sets for the current state of art algorithm and our algorithm with just Multi-length indexing (MLI), Dynamic Stepping (DS) and union of MLI and DS (mc2).***

In addition to the comparison on total running time, we analyze the speedup achieved by the individual techniques, multi-length indexing and dynamic stepping, over the current state of the art algorithm. We measured the individual speedup for each of the techniques while deactivating the other. The results for three datasets are shown in Figure 8.

The multi-length indexing technique and the dynamic stepping technique sped up overall running time for all data sets individually. The union of these 2 techniques showed maximum speed up performance for all the data sets.

Table-5. Accuracy comparison between concatenation based and ensemble based method.

Dataset	Concatenation	Ensembling (max individual)	Merge and Ensembling
Single-Task	66%	76% (72%)	80%
Multi-Task	64%	72% (64%)	

D. *Accuracy on Multi-dimensional Data*

We test our shapelet ensemble in comparison to the method described in [4] which suggests a concatenation of the dimensions to convert a multi-dimensional data into a one-dimensional data. We use the two fMRI datasets described above. Shapelet ensemble achieves an unprecedented accuracy on these datasets. See Table 5. Note that the ensembling can achieve more accuracy than the maximum individual accuracy of the participating trees. This shows a huge potential of shapelet ensembles for sensory data from high frequency electrical sensors.

VIII. Case Studies

We have studied the two multi-dimensional datasets in the medical domains where shapelet ensemble achieve higher accuracy than current state of art algorithm. We detail the cases in this section.

A. *Sensory Gating task for functional MRI (fMRI)*

The dataset contains the results of pre-attentional sensory processing experiment for fMRI scan of the schizophrenia patients (SP=32) and healthy controls (HC=43). The task represented cognitive paradigm that was analyzed separately in [23]. The original study hypothesizes that the SP would have deficits at multiple levels including pre-attentive sensory processing, integration of sensory information across modalities and impaired working memory performance.

The task was a variant of the paired click paradigm for testing sensory gating. The participants were presented with paired tones that are either identical 2 kHz tones or a 2 kHz tone followed by a 3 kHz tone. The task was passive in that a response was not required.

We use the time series of the sensory motor (putamen) component. There are 50 (SP=23, HC=27) time series for training data sets and 25 (SP=9, HC=16) time series for testing data sets. The length of each time series is 145.

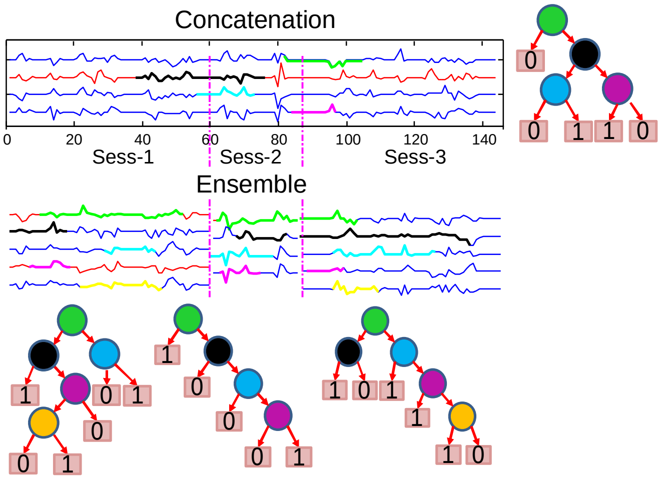

Figure 9: ***The decision tree from concatenated signals is shown in the top. Three trees from the three sessions are shown in the bottom. Shapelets and nodes in the trees have matching colors. Label 0 represents controls and 1 represents patients. Signals in red are from patients and in blue are from healthy subjects (best viewed in color).***

If we concatenate dimensions and run *FastShapelet* we achieve a shapelet tree shown in Figure 9, which gives us 66% accuracy. Note the shapelets that span two successive sessions are meaningless. When we treat each dimension independently, the algorithm provides one tree for each of the three sessions. The ensemble achieves 76% accuracy. Interestingly, no significant class difference is reported in Mayer *et al.* [23] for sensory gating task (same data) while shapelet ensemble achieves a significant accuracy.

If we were to generate the same ensemble using *FastShapelet* algorithm, we would need 6.3x more time.

B. *Multi-modal sensory integration task for functional MRI*

The data set contains the results of multi-modal sensory integration task for fMRI scan of the schizophrenia patients (SP=32) and healthy controls (HC=43). The task represented a cognitive paradigm that was analyzed separately in [24].

Task was designed to test multisensory integration and employed a simple forced choice behavioral task. A perspective digital drawing was used as a visual background with a fixation point, and participants were instructed to maintain fixation throughout the task [24]. An auditory stimulus (500 Hz tone) was presented to participants at two different volumes, 80dB to simulate a sound near the participant (NEAR) and 64dB to simulate it being farther away (FAR). Also, the auditory stimulus was presented synchronously with a visual stimulus - an image of a soccer ball appeared in one of two possible positions (NEAR or FAR) in the participant's lower visual field with size and position consistent with the perspective drawing. The NEAR visual stimulus was presented in the participant's peripheral visual field and the FAR stimulus was presented closer to fixation in the central visual field. The participants underwent fMRI scanning while deciding whether the stimuli presented were NEAR or FAR with a button press.

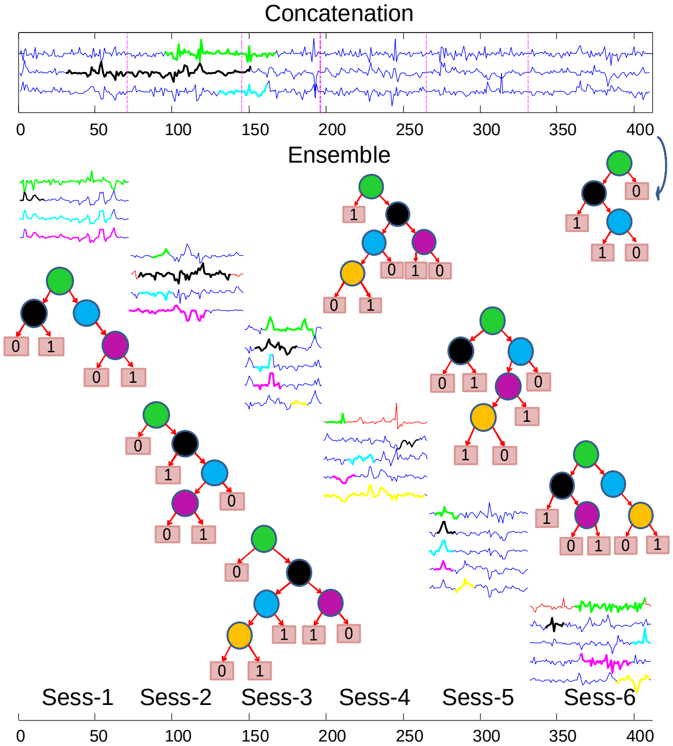

Figure 10: ***The decision tree from concatenated signals is shown in the top. Six trees from the six sessions are shown in the bottom. Shapelets and nodes in the trees have matching colors. Label 0 represents controls and 1 represents patients. Signals in red are from patients and in blue are from healthy subjects (best viewed in color).***

We use the time series of frontal network component. There are 50 (SP=23, HC=27) time series for training datasets and 25(SP=9, HC=16) time series for test data sets. The length of each time series is 411.

On the concatenated signals, the algorithm finds a very compact tree with 62% accuracy (Figure 10 (top)). The shapelets span over multiple sessions and mostly focus on the first three sessions and thus, one cannot learn from all the dimensions. In contrast, shapelet ensemble algorithm finds six trees from six sessions and ensemble them using majority voting. The ensemble achieves 72% accuracy. However, for the same data set, we decreased by 8.32x the overall computation time over the *FastShapelet* algorithm. Note that, individual trees combine information from both red (SP) and blue (HC) subjects while the tree from concatenated signals only contains blue signals. The group differences shown in these case studies provide motivation for understanding how connectivity patterns differ in response to these different stimulus conditions and the current analysis approach may provide enhanced sensitivity to identify group differences of SPs and HCs.

IX. Conclusion

In this paper, we proposed an algorithm to speed up the current shapelet algorithms without decreasing accuracy, especially for multi-dimensional and longer time series. The experiments showed that our algorithm is significantly faster for all data sets that we tested which makes our shapelet discovery algorithm more suitable for real life problems.

X. References

[1] Jason Lines, Luke M. Davis, Jon Hills, and Anthony Bagnall, “A Shapelet Transform for Time Series Classification,” *KDD*, vol. August 12–16, 2012.

[2] Jesin Zakaria, Sarah Rotschafe, Abdullah Mueen, Khaleel Razak, and Eamonn Keogh, “Mining massive archive of mice sounds with symbolized representations.,” *Siam Int. Conf. Data Min.*, 2012.

[3] Lexiang Ye and Eamonn Keogh, “Time Series Shapelets: A New Primitive for Data Mining,” *KDD*, vol. June 29–July 1, 2009.

[4] Abdullah Mueen, Eamonn Keogh, and Neal Young, “Logical-Shapelets: An Expressive Primitive for Time Series Classification,” *KDD*, vol. August 21–24, pp. 1154–1162, 2011.

[5] Thanawin Rakthanmanon and Eamonn Keogh, “Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets,” in *SIAM SDM*, Austin, USA, 2013.

[6] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and Brandon Westover, “Exact Discovery of Time Series Motifs,” *SDM*, pp. 473–484, 2009.

[7] Yasushi, S., Spiros, P., and Christos, F., “BRAID: Stream mining through group lag correlations.,” *SIGMOD Conf.*, pp. 599–610, 2005.

[8] Thanawin Rakthanmanon, Qiang Zhu, and Eamonn Keogh, “Mining Historical Archives for Near-Duplicate Figures,” presented at the ICDM, 2011, pp. 557–566.

[9] Tompa, M. and Buhler, J., “Finding motifs using random projections.,” *J Comput Biol*, vol. 9, no. 2, pp. 225–42, 2002.

[10] Lin, J., Keogh, E, Wei, L., and Lonardi, S, “Experiencing SAX: a novel symbolic representation of time series.,” in *DMKD*, 2007, pp. 107–144.

[11] Li Wei, Eamonn Keogh, and Xiaopeng Xi, “SAXually Explicit Images: Finding Unusual Shapes.,” presented at the ICDM, 2006, pp. 711–720.

[12] Chang, K.-W., Deka, B., Hwu, W.-M. W., and Roth, D., “Efficient Pattern-Based Time Series Classification on GPU,” presented at the ICDM, 2012.

[13] Ljosa, V., Bhattacharya, A., and Singh, AK., “LB-Index: A Multi-Resolution Index Structure for Images,” presented at the 22nd International Conference on Data Engineering, 2006, p. 144.

[14] Christos Faloutsos, M. Ranganathan., and Yannis Manolopoulos, “Fast subsequence matching in time-series databases.,” presented at the SIGMOD, 1994, pp. 419–429.

[15] Jin Shieh and Eamonn Keogh, “iSAX: indexing and mining terabyte sized time series.,” presented at the KDD, 2008, pp. 623–631.

[16] Christos Faloutsos and King-Ip (David) Lin, “FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets.,” presented at the ACM SIGMOD International Conference on Management of Data, San Jose, California, 1995, pp. 163–174.

[17] Abdullah Mueen, “Enumeration of Time Series Motifs of All Lengths.,” *ICDM*, pp. 547–556, 2013.

[18] Dietterich TG. and Kong EB., “Machine Learning Bias, Statistical Bias, and Statistical Variance of Decision Tree Algorithms,” presented at the The XII International Conference on Machine Learning, San Francisco, 1995, pp. 313–321.

[19] Cetin, MS., Mueen, A., and Calhoun VD., “Shapelet Ensemble for Multi-dimensional Time Series,” *SIAM SDM*, 2015. http://www.cs.unm.edu/~mustafa/Shapelet_Ensemble_for_Multi_dimensional_Time_Series

[20] Bell AJ and Sejnowski TJ, “An information-maximization approach to blind separation and blind deconvolution.,” *Neural Comput*, vol. 7, pp. 1129–1159, 1995.

[21] Calhoun VD and Adali T, “Multi-subject Independent Component Analysis of fMRI: A Decade of Intrinsic Networks, Default Mode, and Neurodiagnostic Discovery,” *IEEE Rev. Biomed. Eng.*, vol. 5, pp. 60–73, 2012.

[22] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista, “The UCR Time Series Classification / Clustering Homepage:,” 2015. [Online]. Available: www.cs.ucr.edu/~eamonn/ time_series_data.

[23] Mayer AR, Ruhl D, Merideth F, Ling J, Hanlon FM, Bustillo J, and Cañive J, “Functional imaging of the hemodynamic sensory gating response in schizophrenia.,” *Hum Brain Mapp*, vol. 34, no. 9, pp. 2302–12, 2012.

[24] Stone DB, Urrea LJ, Aine CJ, Bustillo JR, Clark VP, and Stephen JM, “Unisensory processing and multisensory integration in schizophrenia: a high-density electrical mapping study.,” *Neuropsychologia*, vol. 49, pp. 3178–3187, 2011.