# RPM: Representative Pattern Mining for Efficient Time Series Classification

Xing Wang, Jessica Lin
George Mason University
Dept. of Computer Science
xwang24@gmu.edu,
jessica@gmu.edu

Pavel Senin
Bioscience Division, Los Alamos National Laboratory,
Los Alamos, NM, 87544
psenin@lanl.gov

Tim Oates, Sunil Gandhi
University of Maryland,
Baltimore County
Dept. of Computer Science
oates@cs.umbc.edu,
sunilga1@umbc.edu

Arnold P. Boedihardjo  Crystal Chen  Susan Frankenstein
U.S. Army Corps of Engineers, Engineer Research and Development Center
{arnold.p.boedihardjo, crystal.chen, susan.frankenstein}@usace.army.mil

## ABSTRACT

Time series classification is an important problem that has received a great amount of attention by researchers and practitioners in the past two decades. In this work, we propose a novel algorithm for time series classification based on the discovery of class-specific representative patterns. We define representative patterns of a class as a set of subsequences that has the greatest discriminative power to distinguish one class of time series from another. Our approach rests upon two techniques with linear complexity: symbolic discretization of time series, which generalizes the structural patterns, and grammatical inference, which automatically finds recurrent correlated patterns of variable length, producing an initial pool of common patterns shared by many instances in a class. From this pool of candidate patterns, our algorithm selects the most representative patterns that capture the class specificities, and that can be used to effectively discriminate between time series classes. Through an exhaustive experimental evaluation we show that our algorithm is competitive in accuracy and speed with the state-of-the-art classification techniques on the UCR time series repository, robust on shifted data, and demonstrates excellent performance on real-world noisy medical time series.

## 1. INTRODUCTION

Massive amount of time series data are generated daily in areas as diverse as medicine, astronomy, industry, sciences, and finance, to name just a few. Even with the explosion of interest in time series data mining during the past two decades, and increasing popularity of new emerging topics such as motif discovery, classification of time series still remains one of the most important problems with many real-world applications in diverse disciplines.

©2016, Copyright is with the authors. Published in Proc. 19th International Conference on Extending Database Technology (EDBT), March 15-18, 2016 - Bordeaux, France: ISBN 978-3-89318-070-7, on OpenProceedings.org. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

While many classification algorithms have been proposed for time series, it has been shown that the nearest neighbor classifier, albeit simple in design, is competitive with the more sophisticated algorithms like SVM [32]. As a result, many existing techniques on time series classification focus on improving the similarity measure, an essential part of the nearest neighbor classifier [4]. Recently, the notion of time series shapelets—time series subsequences that are "maximally representative" of a class—has been proposed. Shapelets generalize the lazy nearest neighbor classifier to an eager, decision-tree-like classifier [36][10], which typically improves the classification speed and interpretability of the results.

In this work, we focus on a similar problem of finding the most representative patterns for the classification task. We call our algorithm RPM (Representative Pattern Mining). The key motivation is that the identification of a small set of distinctive and interpretable patterns of each class allows us to exploit their key characteristics for discriminating against other classes. In addition, we hypothesize that the classification procedure based on a set of highly class-characteristic short patterns will provide high generalization performance under noise and/or translation/rotation, i.e. it shall be robust and shift/rotation invariant.

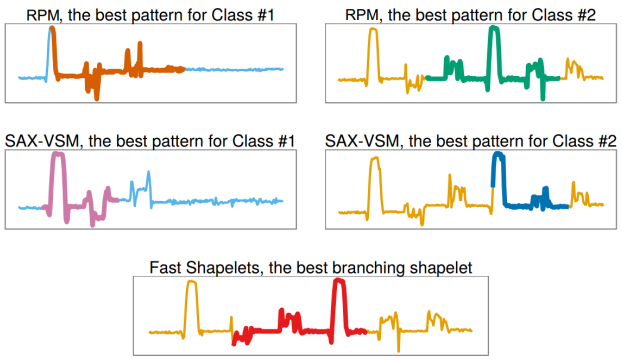


**Figure 1:** An illustration of the best patterns discovered by rival subsequence-based techniques on Cricket data [20].

Our work is significantly different from existing subsequence-based techniques such as $K$-shapelet discovery and

classification algorithms. Specifically, one major difference lies in our definition of *representative patterns*. We define representative patterns to be class-specific prototypes, i.e. each class has its own set of representative patterns, whereas in shapelets some classes may share a shapelet. Figure 1 shows the patterns/shapelets identified by different algorithms on the Cricket dataset [20]. Note that SAX-VSM [31] captures visually similar short patterns of the same length in both classes. Fast Shapelets [27] selects a single subsequence to build a classifier. Our algorithm, RPM, selects different patterns that capture the data specificity (characteristic left and right hand movements) for each class.

The methodology employed by our approach is also very different from existing shapelet-based techniques. Contrasting with a decision-tree-based shapelet classification which finds the best splitting shapelet(s) via the exhaustive candidate elimination and explicit distance computation, we rely on the grammar induction (GI) procedure that *automatically* (i.e. by the algorithm's design) and *without computing any distance explicitly* [17][30] discovers frequent subsequences (motifs) of *variable length*, which we consider as representative pattern candidates. The number of candidates for the exhaustive shapelet search approach is $O(nm^2)$ ($n$ is the number of time series, $m$ is their length) [27], since the algorithm examines all possible subsequences. For our method, the number of candidates considered is much smaller: $O(K)$, where $K$ is the number of motifs since only patterns that frequently occur in a class can be *representative*.

In addition to speeding up the algorithm, grammar induction, by design, grows the lengths of the initial patterns when constructing grammar rules, thus eliminating the need for searching an optimal pattern length exhaustively – a procedure that is common to most of sliding window-based shapelet techniques [36][10][27].

Since GI requires discrete input, our algorithm transforms real-valued time series into discrete values using Symbolic Aggregate approXimation (SAX) [18]. The algorithm operates in the approximate symbolic space inferring a grammar and generating a set of candidate patterns through the analysis of the grammar hierarchy in linear time. Next, the algorithm maps the discovered patterns back into real values and continues with pattern refinement using clustering. The cluster centroids (or medoids) are then reported as the best class-specific motifs, among which we select the most representative ones by verifying their classification power at the final step.

Figures 2 and 3 show examples of representative patterns discovered by our technique in two datasets: CBF, a synthetic dataset [22], and Coffee, a real dataset [2]. Representative patterns discovered in CBF highlight the most distinctive features in each of the three classes: a plateau followed by the sudden rise then followed by a plateau in Cylinder, the increasing ramp followed by a sudden drop in Bell, and a sudden rise by a decreasing ramp in Funnel. Representative patterns discovered in the Coffee dataset also correspond to the most distinctive natural features which not only include the discriminative caffeine and chlorogenic acid bands, but the spectra corresponding to other constituents such as carbohydrates, lipids, etc. [2].

Since we use a different selection criterion, the representative patterns discovered by our technique are different from the shapelets or patterns found by other subsequence-based techniques, thus providing a complementary functionality that can be used for exploratory studies. For example, the representative patterns discovered in CBF and Coffee datasets by our approach are different from the shapelets discovered by Fast Shapelets [27], a well-known shapelet discovery algorithm which we use for experimental comparison. For the CBF dataset, Fast Shapelets reports two branching shapelets that correspond to sudden rises in Cylinder and Funnel classes. For the Coffee dataset, Fast Shapelets reports a single branching shapelet corresponding to the caffeine spectra band (Arabica).

Note that the discovery of class-specific motifs, which is an integral part of our algorithm, also offers a unique advantage that extends beyond the classification task. Differing from the traditional notion of time series motifs [17][3], which can either be repeated subsequences of a fixed length within a long time series, or repeated time series instances within a group of data (e.g. shape motifs [16]), our class-specific motifs are variable-length sub-patterns that occur frequently in many time series of a data group. They are, in a sense, related to time series subspace clusters [15]. Therefore, our approach provides an efficient mechanism to discover these subspace patterns without exhaustively searching through all subsequences. Throughout the paper, we will use the terms "class-specific subspace motifs" and "class-specific motifs" interchangeably.

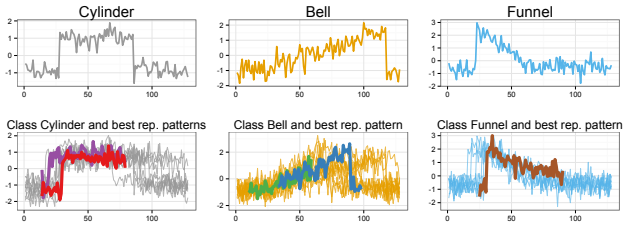


**Figure 2:** The Cylinder-Bell-Funnel (CBF) dataset and the best representative patterns for its classes discovered with the proposed technique.

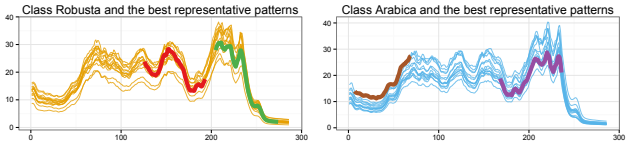


**Figure 3:** Two classes from the Coffee dataset and the best *representative patterns*.

As we shall demonstrate, in addition to the excellent exploratory characteristics, our approach achieves competitive classification accuracy compared to the state-of-the-art techniques: nearest neighbor classifiers, characteristic subsequence-based classifier (SAX-VSM), and shapelet-based classifiers, while maintaining great efficiency.

The rest of the paper is organized as follows: Section 2 discusses related work and background materials. We describe our approach for finding representative patterns in Section 3, and discuss parameter optimization in Section 4. Section 5 presents experimental results. We demonstrate the utilities of our approach with two case studies in Section 6, and conclude in Section 7.

## 2. RELATED WORK AND BACKGROUND

## 2.1 Notation and definition

To precisely state the problem at hand, and to relate our work to previous research, we will define the key terms used throughout this paper. We begin by defining our data type, time series:

**Time series** $T = t_1, \ldots, t_m$ is a set of scalar observations ordered by time.

Since we focus on finding local patterns that are representative of a class, we consider time series subsequences:

**Subsequence** $S$ of time series $T$ is a contiguous sampling $t_p, \ldots, t_{p+n-1}$ of points of length $n << m$ where $p$ is an arbitrary position, such that $1 \leq p \leq m - n + 1$.

Typically subsequences are extracted from a time series with the use of a sliding window:

**Sliding window** subsequence extraction: for a time series $T$ of length $m$, and a user-defined subsequence length $n$, all possible subsequences of $T$ can be found by sliding a window of size $n$ across $T$.

Given two time series subsequences $S_1$ and $S_2$, both of length $n$, the distance between them is a real number that accounts for how much these subsequences are different, and the function which outputs this number when given $S_1$ and $S_2$ is called the **distance function** and denoted $Dist(S_1, S_2)$. One of the most commonly used distance functions is the **Euclidean distance**, which is the square root of the sum of the squared differences between each pair of the corresponding data points in $S_1$ and $S_2$.

**Closest (i.e. best) match**: Given a subsequence $S$ and a time series $T$, the time series subsequence $T_p$ of length $|S|$ starting at position $p : 0 < p < |T| - |S|$ is the closest match of S if $Dist(S, T_p) \leq Dist(S, T_k)$, where $T_k$ is a subsequence of $T$ starting at any position $k : 0 \leq k < |T| - |S|$, and $k \neq p$. The $Dist(S, T_p)$ is the **closest match distance**.

**Time series pattern** is a subsequence that possesses certain interesting characteristics. For example it can be a subsequence that occurs frequently, i.e. whose observance frequency is above some arbitrary threshold $t$. A frequently occurring subsequence is also called **time series motif**.

**Class-specific motif**: Given a class $C$ and a set of training instances $X_C$, a class-specific motif $M$ for $C$ is a subsequence pattern $S$ in $C$ consisting of a set of similar subsequences from different training instances such that $count(S) \geq (\gamma \cdot |X_C|)$, where $0 < \gamma \leq 1$. This states that a pattern is considered frequent if it appears in at least $\gamma$ of the training instances in the class. We will describe what we mean by "similar" in a later section.

**Representative patterns**: The most discriminative subsequences among the class motifs for class $C$ are selected as the representative patterns for the class. The number of the representative patterns for each class is dynamically determined by the algorithm.

We will describe how to measure the "representativeness" and discriminative power of the candidate patterns in a later section.

**Time Series Transformation**: The set of the closest match distances between a time series $T$ and the (candidate) representative patterns can be viewed as a transformation of $T \in \mathbb{R}^{n \cdot m}$ into $T' \in \mathbb{R}^{n \cdot K}$, where $K$ is the total number of the representative patterns from all classes.

## 2.2 Related work

Classification of time series has attracted much interest from the data mining community in the past two decades [4][36][5][13][21][28][25][33][20][35][34]. Nevertheless, to date, the simple nearest neighbor classification is the most popular choice due to its simplicity and effectiveness [32]. Therefore, a large body of work on time series classification has focused on the nearest neighbor classification improvement by developing new data representations or distance measures [32]. To date, a nearest neighbor classification with an effective distance measure like Dynamic Time Warping (DTW) outperforms many existing techniques [32].

Among the proposed alternatives, many methods focus on finding local patterns in time series as predictive features of the class [5]. Recently, Ye and Keogh introduced a novel concept called time series "shapelet". A shapelet is an exact time series subsequence that is "maximally representative" of a class [36]. Once found, a shapelet-based technique classifies an unlabeled time series by computing its similarity to the shapelet. The original shapelet technique proposed by the authors constructs a decision tree-based classifier which uses the shapelet similarity as the splitting criterion. While effective and interpretable, the original shapelet discovery technique is computationally intensive.

Numerous improvements were proposed. The Logical Shapelets [20] extends the original work by improving the efficiency and introducing an augmented, more expressive shapelet representation based on conjunctions or disjunctions of shapelets. Fast Shapelets [27] improves the efficiency of the original shapelets algorithm by exploiting the projections into a symbolic representation. Learning Shapelets [7] proposes a new mathematical formalization that iteratively reduces the shapelet search space by computing a classification precision-based objective function.

The "Shapelet Transform" [10] technique finds the best $K$-shapelets and transforms the original time series into a vector of $K$ features, each of which represents the distance between a time series and a shapelet. This technique can thus be used with virtually any classification algorithm.

SAX-VSM [31] is another approximate algorithm that enables the discovery of local class-characteristic (representative) patterns based on the similar pattern-discrimination principle via **tf∗idf** -based patterns ranking [29]. While similar to our notion of representative patterns, the length of SAX-VSM-generated patterns equals to the sliding window length. In addition, the algorithm makes no additional effort to prune the discovered patterns, yielding a large sparse matrix of pattern weights.

Our algorithm can be related to the concept of mining interesting frequent patterns reviewed in [9], as it is essentially the selection of "interesting" pattern subset from a frequent pattern set. In our case, we regard the representative power of a pattern as its interestingness measure.

# 3. RPM: REPRESENTATIVE PATTERN MINING FOR CLASSIFICATION

The classification algorithm we propose consists of two stages: (a) Learning the representative patterns. (b) Classification using the representative patterns.

In the training stage, the algorithm identifies the most representative patterns for each class from the training data by 3 steps: (i) pre-processing training data; (ii) generating representative pattern candidates from processed data; (iii) selecting the most representative patterns from candidates.

Once the representative patterns are learned, we trans-

form the training data into a new feature space, where each representative pattern is a feature and each training time series is represented as a vector of distances to these patterns. We can then build a classifier from the transformed training data.

We will describe the algorithm in detail below, starting with the classification stage.

## 3.1 Time series classification using representative patterns

To classify a time series using the representative patterns learned from the training stage, the first step is to transform the test data $T \in \mathbb{R}^{n \cdot m}$ into a feature space representation $T' \in \mathbb{R}^{n \cdot K}$ by computing distances from $T$ to each of the $K$ representative patterns from all classes. The transformed time series is thus represented as a fixed-length vector – a universal data type which can be used with many of the traditional classification techniques. In this paper, we use SVM [24] for its popularity, but note that our algorithm can work with any classifier.

## 3.2 Training stage: Finding representative patterns

As mentioned previously, our goal is to find the most representative and distinctive patterns for each of the time series classes. These class-specific patterns should satisfy two requirements: (i) they should be class-specific motifs, i.e. shared by at least $\gamma$ (a fraction) of the time series in the class; and (ii) they should enable the discrimination between the current and other classes – the capacity measured with a scoring function discussed below.

At the high level, the algorithm can be viewed as a succession of three steps. First, the algorithm performs data pre-processing for each class in the training data, by concatenating all instances from the same class into a long time series, and discretizing the concatenated time series. Second, the algorithm finds frequent patterns via grammar inference and *forms the candidate pool* for each class. Third, it refines the candidates pool by *eliminating redundant and non-discriminating patterns*, and outputs patterns that represent the class the best, i.e., the representative patterns.

In our previous work, we proposed GrammarViz (v2.0), which uses time series discretization and grammar inference for time series motif discovery and exploration [17][30]. We observe that the same approach can be leveraged to find class-specific subspace motifs, thus enabling the classification as well.

### 3.2.1 Step 1: Pre-processing

We prepare the training data for subspace pattern discovery by concatenating all training time series from the same class into a single long time series. We note that the concatenation step is not required for our learning algorithm and can in fact be skipped. The reason for concatenating the training instances is for visualization purpose only, as will be shown in Figure 4.

Next, we discretize the concatenated time series into a sequence of tokens for grammar induction using Symbolic Aggregate approXimation (SAX) [18]. More specifically, we apply SAX to *subsequences* extracted from the concatenated time series of a training class via a sliding window. SAX performs subsequence discretization by first reducing the dimensionality of the subsequence with Piecewise Aggregate Approximation (PAA) [12]. Towards that end, it divides $z$-normalized subsequence into $w$ equal-sized segments and computes a mean value for each segment. It then maps these values to symbols according to a pre-defined set of breakpoints dividing the distribution space into $\alpha$ equiprobable regions, where $\alpha$ is the alphabet size specified by the user. This *subsequence discretization* process outputs an ordered list of SAX words, where each word corresponds to the leftmost point of the sliding window. Two parameters affect the SAX transform granularity – the number of PAA segments (PAA size) and the SAX alphabet size.

Since neighboring subsequences extracted via a sliding window share all points except one, they are similar to each other and often have identical SAX representations. To prevent over-counting a pattern, we apply numerosity reduction [17]: if in the course of discretization, the same SAX word occurs more than once consecutively, instead of placing every instance into the resulting string, we record only its first occurrence.

As an example, consider the sequence $S0$ where each word (e.g. *aba*) represents a subsequence extracted from the concatenated time series via a sliding window and then discretized with SAX. The subscript following each word denotes the starting position of the corresponding subsequence in the time series.

$S0 = aba_1\ bac_2\ bac_3\ bac_4\ cab_5\ acc_6\ bac_7\ bac_8\ cab_9\ \ldots$

With numerosity reduction, $S0$ becomes:

$S1 = aba_1\ bac_2\ cab_5\ acc_6\ bac_7\ cab_9\ \ldots$

Numerosity reduction not only reduces the length of the input for the next step of the algorithm (hence making it more efficient and reducing its space requirements) and simplifies the identification of non-overlapping time series motifs by removing "noise" from overlapping subsequences. Most importantly, numerosity reduction enables the discovery of representative patterns of varying lengths as we show next.

### 3.2.2 Step 2: Generating representative pattern candidates

In this step, we use grammar induction to identify repeated patterns in the concatenated time series, and generate a *representative pattern candidates pool* from these patterns. The algorithm is outlined in Algorithm 1. The sequence of SAX words obtained from the pre-processing step is fed into a context-free grammar induction algorithm (Algorithm 1, Line 7). We use Sequitur, a string compression algorithm that infers a context-free grammar in linear time and space [23]. We choose Sequitur due to its efficiency and reasonably good compression capability, but note that our technique also works with other (context-free) GI algorithms. When applied to a sequence of SAX words, Sequitur treats each word as a token and recursively reduces all *digrams*, i.e. consecutive pairs of tokens (terminals or non-terminals), occurring more than once in the input string to a single new non-terminal symbol representing a grammar rule.

Consider the grammar induced by Sequitur from the input string $S1$:

| Grammar Rule | Expanded Grammar Rule |
|---|---|
| R0 → aba R1 acc R1 | $aba_1$ $bac_2$ $cab_5$ $acc_6$ $bac_7$ $cab_9$ |
| R1 → bac cab | $bac$ $cab$ |

In this grammar, $R1$ describes a simplified version of the

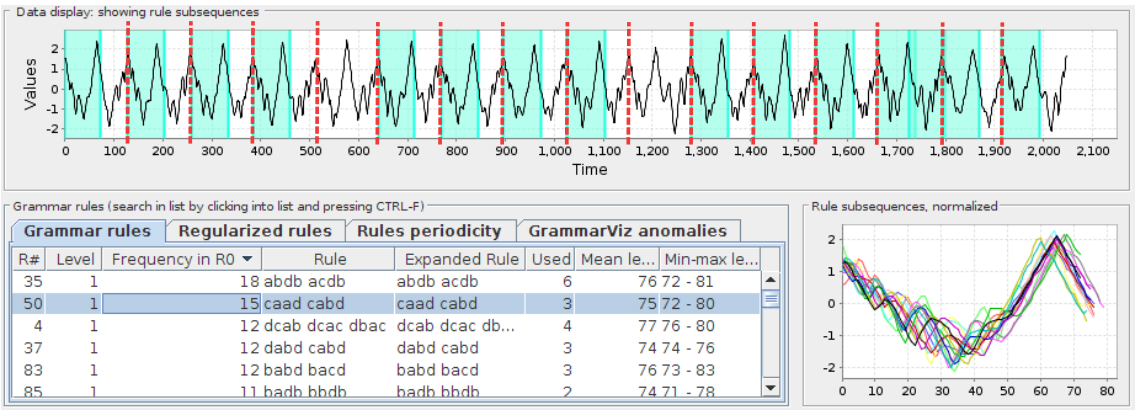


**Figure 4:** RPM pre-processing visualization with GrammarViz 2.0 [30]. The time series from Class 4 of SwedishLeaf dataset were concatenated, discretized, Sequitur grammar was inferred, and one of the frequent motifs selected. Note that the grammar rule-corresponding subsequences vary in length from 72 to 80. The length of the original time series before concatenation is 128 as indicated by dotted lines. Note that one of time series does not contain the pattern and another contains it twice.

repeated pattern [*bac cab*], which concurrently maps to two substrings of different lengths from $S0$: [$bac_2$ $bac_3$ $bac_4$ $cab_5$] (length of 4) and [$bac_7$ $bac_8$ $cab_9$] (length of 3), respectively.

By keeping SAX words' offsets throughout the procedures of discretization and grammar induction, we are able to map rules and SAX words back to their original time series subsequences. Since each SAX word corresponds to a *single point* of the input time series (a subsequence starting point), $R1$ maps to subsequences of variable lengths ([2-5] and [7-9]). Figure 4 shows an example of the recurrent subsequences found in the concatenated time series from Class 4 of the Swedish Leaf dataset. Note, that when processing a concatenated time series, the algorithm does not consider the subsequences that span time series junction points in order to avoid concatenation process artifacts.

**Algorithm 1** Finding repeated patterns

```
1: function FindCandidates(Train, SAXParams, γ)
2:     candidates ← ∅
3:     allCandidates ← ∅
4:     for each TrainClassI in Train do
5:         cTS ← ConcatenateTS(TrainClassI)
6:           // {build grammar avoiding junctions (see Fig. 4)}
7:         allRepeats ← modifiedGI(cTS, SAXParams)
8:         refinedRepeats ← ∅
9:           // {r is repeated subsequences from a row in Fig. 4}
10:        for each r in allRepeats do
11:            clusters ← Clustering(r)
12:            refinedRepeats.addAll(clusters)
13:        for each cluster in refinedRepeats do
14:            if cluster.size > γ · |I| then
15:                centroid ← GetCentroid(cluster)
16:                  // {candidates for class I}
17:                candidates.add(centroid)
18:          // {candidates for all classes}
19:        allCandidates.add(candidates)
20:    return (allCandidates)
```

***Refining repeated subsequences:*** Every grammar rule induced from Sequitur describes an *approximate* repeated pattern (our candidate motif) observed in time series; however, corresponding exact subsequences may differ significantly depending on the SAX granularity. To select the most representative of the frequent symbolic patterns (and essentially of the training class), we use hierarchical clustering algorithm (complete-linkage) to cluster all rule-corresponding subsequences based on their similarity (Algorithm 1, Line 11). Note that the purpose of applying clustering here is to handle the situation where a candidate motif found by grammar induction algorithm may contain more than one group of similar subsequences. In this case, we should partition the subsequences into sets of clusters, each of which corresponds to one motif. To determine the appropriate number of clusters, we first set the number of clusters as *two*. If the cluster sizes are drastically different, e.g. one of the clusters contains less than 30% of subsequences from the original group, we do not split the original group. If both clusters contain sufficient numbers of subsequences, we will continue to split them into smaller groups. The partitioning stops when no group can be further split (Algorithm 1, Line 12).

Consistent with Section 3.2 requirement (i), if the size of a cluster is smaller than the specified threshold $\gamma$, the cluster is discarded. If a cluster satisfies the minimum size requirement, its centroid is added to the representative patterns candidates pool for a class (Algorithm 1, Line 14-15). Note an alternative is to use the medoid instead of centroid. As outlined in Algorithm 1, this refinement procedure outputs a list of representative pattern *candidates* for all classes.

### 3.2.3 *Step 3: Selecting the most representative patterns*

The candidate patterns obtained so far are the frequent patterns that occur in a class. However, some of them may not be class-discriminative if they also occur in other classes. Addressing this issue, we prune the candidate patterns pool with Algorithm 2 whose input is the pool of candidate patterns identified from the previous step, and the entire training dataset. The algorithm outputs the set of class-specific patterns.

**Remove Similar Patterns**: The representative pattern candidates are repeated patterns found by the grammar induction algorithm applied to the discretized time series. Due to the aggregation, some structurally similar subsequences may be mapped to slightly different SAX strings, e.g. differ by one letter. Feeding such patterns to the subsequent step (selecting representative patterns) will slow down the search

**Algorithm 2** Find distinctive patterns

```
1: function FindDistinct(Train, allCandidates)
2:     candidates ← ∅
3:     τ ← ComputeThreshold
4:     // {Remove similarities in allCandidates}
5:     for each c in allCandidates do
6:         isNonSimilar ← true
7:         for each c_ns in candidates do
8:             // {c and c_ns may have different length}
9:             dist ← ComputeClosestMatchDist(c, c_ns)
10:            if dist < τ then
11:                if c_ns.frequent < c.frequent then
12:                    // {The frequency in concatenated TS}
13:                    candidates.remove(c_ns)
14:                    candidates.add(c)
15:                    isNonSimilar ← false
16:                    break();
17:        if isNonSimilar then
18:            candidates.add(c)
19:    // {Transform TS into new feature space where each feature is the distance between time series and a candidate}
20:    TransformedTrain ← Transform(Train, candidates)
21:    // {Perform feature selection algorithm on new data}
22:    selectedIndices ← FSAlg(TransformedTrain)
23:    // {Select patterns according to indices }
24:    patterns ← Select(candidates, selectedIndices)
25:    return (patterns)
```

when removing correlated patterns in feature selection step.

In order to resolve this issue, our algorithm removes similar candidates from the candidate set, as shown in Algorithm 2, Lines 5 – 18. Before the removal, it computes a threshold used to determine if two patterns are similar. The threshold (Alg. 2, Line 3) is determined as follows: (i) Compute pairwise distances of subsequences within each refined grammar rule (i.e. the final clusters from Algorithm 1). (ii) Sort the distances in ascending order. (iii) Take the distance at the 30-th percentile as the threshold $\tau$ for similarity. We will show the effect on accuracy and running time with different values of $\tau$ in the experimental section.

**Select Representative Patterns**: After refining the representative pattern candidates pool from the previous steps, we transform the original time series into a distance feature vector by computing the closest match distance between each time series and all the candidate patterns. As an example, two patterns in dataset ECGFiveDays are shown in Figure 5, and the transformed training data is shown in Figure 6. The original time series from the two classes look visually similar. However, once we transform the raw time series into the two-dimensional feature vector (one feature from each class in this case), it is easy to separate the two classes. As can be seen in Figure 6, the transformed data is linearly separable.

Since the transformation uses all the candidates as new features, a feature selection step is needed to select the most distinctive features. Each feature represents the distance from the original time series to one of the candidate patterns. Thus, the features selected represent the most representative patterns. Any feature selection algorithms can be applied here. In this work, we use the correlation-based feature selection from [8], since it is capable of identifying features that are highly correlated with the class. After feature selection, the selected patterns will be used to classify future time series. Note that the number of selected patterns for each class is dynamically determined by the feature selection algorithm.

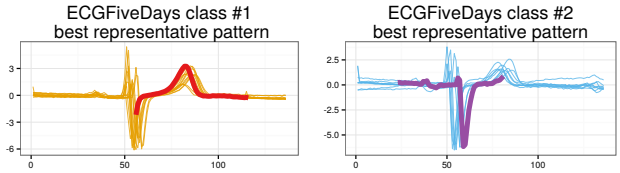


**Figure 5:** Two classes from the ECGFiveDays dataset and the best *representative patterns*.

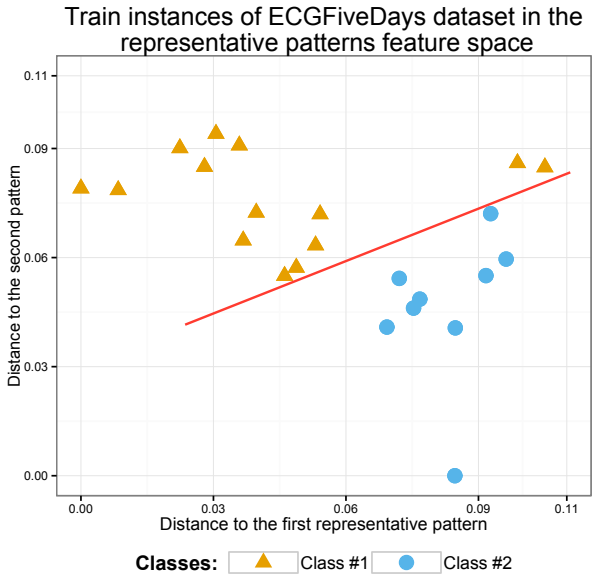


**Figure 6:** Transformed data of train data from ECGFiveDays

## 4. PARAMETER SELECTION AND CLASSIFICATION WITH DIFFERENT SAX PARAMETERS

As shown in Algorithm 1, there is a parameter vector $SAXParams$ in the input. The vector consists of three SAX discretization parameters, namely the sliding window size, PAA size, and the SAX alphabet size. In this section, we shall describe our algorithm for the optimal SAX parameters selection. Since time series data in different classes may have varying characteristics, a parameter set that is optimal for one class may not be optimal for another. Therefore, the parameter optimization is performed for each class.

### 4.1 Search for the best SAX parameters exhaustively

One way to find the optimal parameter set is by brute force grid search – as shown in Algorithm 3. The algorithm tests all parameter combinations within a specified range and selects the optimal set (the one that results in the best F1 measure score from five-fold cross validation on the validation data set). For each parameter combination candidate, the grid search algorithm first divides the original training data into training and validation data 5 times for 5 different splits. For each split, the algorithm invokes Algorithms 1 and 2 to obtain the representative patterns

(Lines 8-9). The validation is performed on the validation data set (Line 12) with a five-fold cross validation. The F1 measure is computed using the classification result for each class. The parameter combination with the best F1 measure for each class is selected as the optimal parameters for the class.

**Algorithm 3** SAX Parameter selection (Brute-Force)

```
1: function ParamsSelect(ParamsRange, OriginalTrain)
2:     for each Class I do
3:         bestSoFarForI ← 0
4:     for each SAXPs in ParamsRange do
5:         // {Repeat 5 times for different splits}
6:         while iteration < 5 do
7:             {train, validate} ← Split(OriginalTrain)
8:             candidates ← FindCandidates(train, SAXPs, γ)
9:             patterns ← FindDistinct(train, candidates)
10:            validate_t ← Transform(validate, patterns)
11:            // {fMeasure is the f1 measure of each class}
12:            fMeasure ← 5foldsCV(validate_t)
13:            for each Class I do
14:                if fMeasure.i > bestSoFarForI then
15:                    bestParamForClassI ← SAXPs
16:                    bestSoFarForI ← fMeasure.i
17:            iteration++
18:    for each Class I do
19:        bestParams.add(bestParamForClassI)
20:    return (bestParams)
```

Instead of running Algorithm 3 completely to find the best parameter set, pruning can be performed by the observed number of repeated patterns. In Line 8 of Algorithm 3, if no candidate for a class is returned because all repeated patterns for that class have frequency below the specified threshold ($\gamma$), the algorithm abandons the current iteration and proceeds with the next parameter combination. The intuition rests upon the fact that given the number of time series in the training data, we can estimate the required minimal frequency for the repeated patterns.

## 4.2 Searching for the best SAX parameters using DIRECT

We optimize the search for the best discretization parameters set by using the DIviding RECTangles (DIRECT) algorithm [11] which is a derivative-free optimization scheme that possesses local and global optimization properties, converges quickly, and yields a deterministic optimal solution. DIRECT is designed to deal with optimization problems of the form:

$$\min_x f(x),\ f \in \mathbf{R},\ x, X_L, X_U \in \mathbf{R},\ \text{where } X_L \leq x \leq X_U$$

where $f(x)$ is the objective (error) function, and $x$ is a parameters vector. The algorithm begins by scaling the search domain to a unit hypercube. Next, it iteratively performs a sampling procedure consisting of two steps: (i) partitioning the hypercube into smaller hyper-rectangles and (ii) identifying a set of potentially-optimal hyper-rectangles by sampling their centers. Iterations continue until the error function converges. DIRECT is guaranteed to converge to the global optimal function value as the number of iterations approaches infinity and the function is continuous in the neighborhood of a global optimum [11]. If interrupted at any iteration, for example after exhausting a time limit, the algorithm reports the best-so-far parameters set.

Since SAX parameters are integer values, we round the values reported by DIRECT to the closest integers when optimizing their selection for our cross validation-based error function (one minus $fMeasure$, as described in Section 4.1). While rounding affects the DIRECT convergence speed, this approach is not only much more efficient than the exhaustive search, but is also able to perform a time-constrained parameter optimization by limiting the number of iterations.

## 4.3 Classification with class-specific trained parameters

With the best SAX parameters learned from Section 4.2, the representative patterns can be obtained by calling Algorithms 1 and 2. To classify future instances, we follow the classification procedure described in Section 3.1. However, with the class-specific parameter optimization, we need to add more steps to the classification procedure to account for the adaptive parameter sets for different classes. Different classes may have different best SAX parameter combinations (SPCs). We learn the best SPCs for each class respectively as described previously. Then we apply Algorithms 1 and 2 to obtain the representative patterns for each SPC. We combine all these representative patterns together and remove the correlated patterns by applying feature selection again. We then obtain the final set of representative patterns, which will be used as the input for the algorithm described in Section 3.1 to classify test data.

# 5. EXPERIMENTAL EVALUATION

## 5.1 Setup and Baseline

We evaluate the classification performance of our technique on the standard time series archive from the UCR repository [14]. Information on the datasets is shown in Table 1. The code and datasets used for the case study (discussed next section) are available on [1]. We compare our method Representative Pattern Mining (RPM) with five other classification techniques, among which are two nearest-neighbor classifiers based on the global distance measures: Euclidean distance (1NN-ED) and DTW with the best warping window (1NN-DTWB), and three classifiers based on the use of class-characteristic local patterns: Fast Shapelets (FS) [27], SAX-VSM [31] and Learning Shapelets (LS) [7]. These three subsequence-based techniques rely on different numbers of patterns for classification – while Fast Shapelets uses a minimal number of patterns to build a classification tree, SAX-VSM accounts for all patterns extracted via sliding window in each of the class-representing weight vectors. Learning Shapelets has the best accuracy so far.

## 5.2 Classification Accuracy

Table 1 shows the classification error rates for all six methods on the UCR datasets. The best error rate for each dataset is denoted with boldface. In addition, Figure 7 shows the summary comparison of our proposed technique with other methods. The results shown are with parameter optimization, and the $\gamma$ (minimum cluster size) is set to be 20% of the training size for the class. From the results, our method is the second best on classification accuracy among these six methods. We slightly lose to the Learning Shapelets method, which has the most "wins" in classification. However, the p-value of wilcoxon test is 0.834 $> 0.05$, so the difference is not significant with a confidence

**Table 1:** Datasets description and the classification error rates.

| Dataset | Classes | Train | Test | Length | 1NN-ED | 1NN-DTWB | SAX-VSM | FS | LS | RPM |
|---|---|---|---|---|---|---|---|---|---|---|
| 50words | 50 | 450 | 455 | 270 | 0.369 | 0.242 | 0.374 | 0.483 | **0.232** | 0.242 |
| Adiac | 37 | 390 | 391 | 176 | 0.389 | 0.391 | 0.417 | 0.425 | 0.437 | **0.355** |
| Beef | 5 | 30 | 30 | 470 | 0.333 | 0.333 | **0.233** | 0.467 | 0.240 | **0.233** |
| CBF | 3 | 30 | 900 | 128 | 0.148 | 0.004 | 0.010 | 0.092 | 0.006 | **0.001** |
| ChlorineConcentration | 3 | 467 | 3840 | 166 | 0.352 | 0.350 | **0.341** | 0.667 | 0.349 | 0.355 |
| CinC_ECG_torso | 4 | 40 | 1380 | 1639 | 0.103 | **0.070** | 0.344 | 0.225 | 0.167 | 0.125 |
| Coffee | 2 | 28 | 28 | 286 | **0.000** | **0.000** | **0.000** | 0.071 | **0.000** | **0.000** |
| Cricket_X | 12 | 390 | 390 | 300 | 0.423 | 0.252 | 0.308 | 0.549 | **0.209** | 0.236 |
| Cricket_Y | 12 | 390 | 390 | 300 | 0.433 | **0.228** | 0.318 | 0.495 | 0.249 | 0.379 |
| Cricket_Z | 12 | 390 | 390 | 300 | 0.413 | 0.238 | 0.297 | 0.533 | 0.201 | **0.190** |
| DiatomSizeReduction | 4 | 16 | 306 | 345 | 0.065 | 0.065 | 0.121 | 0.078 | **0.033** | 0.069 |
| ECG200 | 2 | 100 | 100 | 96 | **0.120** | **0.120** | 0.140 | 0.250 | 0.126 | 0.170 |
| ECGFiveDays | 2 | 23 | 861 | 136 | 0.203 | 0.203 | 0.001 | **0.000** | **0.000** | **0.000** |
| FaceAll | 14 | 560 | 1690 | 131 | 0.286 | **0.192** | 0.245 | 0.408 | 0.218 | 0.231 |
| FaceFour | 4 | 24 | 88 | 350 | 0.216 | 0.114 | 0.114 | 0.091 | 0.048 | **0.045** |
| FacesUCR | 14 | 200 | 2050 | 131 | 0.231 | 0.088 | 0.109 | 0.296 | 0.059 | **0.034** |
| Fish | 7 | 175 | 175 | 463 | 0.217 | 0.154 | **0.017** | 0.189 | 0.066 | 0.065 |
| Gun_Point | 2 | 50 | 150 | 150 | 0.087 | 0.087 | 0.013 | 0.040 | **0.000** | 0.027 |
| Haptics | 5 | 155 | 308 | 1092 | 0.630 | 0.588 | 0.584 | 0.610 | **0.532** | 0.562 |
| InlineSkate | 7 | 100 | 550 | 1882 | 0.658 | 0.613 | 0.593 | 0.733 | 0.573 | **0.535** |
| ItalyPowerDemand | 2 | 67 | 1029 | 24 | 0.045 | 0.045 | 0.089 | 0.063 | **0.031** | 0.044 |
| Lightning2 | 2 | 60 | 61 | 637 | 0.246 | **0.131** | 0.213 | 0.361 | 0.177 | 0.262 |
| Lightning7 | 7 | 70 | 73 | 319 | 0.425 | 0.288 | 0.397 | 0.397 | **0.197** | 0.219 |
| MALLAT | 8 | 55 | 2345 | 1024 | 0.086 | 0.086 | 0.199 | 0.054 | 0.046 | **0.020** |
| MedicalImages | 10 | 381 | 760 | 99 | 0.316 | **0.253** | 0.516 | 0.443 | 0.271 | 0.262 |
| MoteStrain | 2 | 20 | 1252 | 84 | 0.121 | 0.134 | 0.125 | 0.202 | **0.087** | 0.113 |
| OliveOil | 4 | 30 | 30 | 570 | 0.133 | 0.133 | 0.133 | 0.300 | 0.560 | **0.100** |
| OSULeaf | 6 | 200 | 242 | 427 | 0.479 | 0.388 | **0.165** | 0.421 | 0.182 | 0.211 |
| SonyAIBORobotSurface | 2 | 20 | 601 | 70 | 0.304 | 0.305 | 0.306 | 0.315 | 0.103 | **0.058** |
| SonyAIBORobotSurfaceII | 2 | 27 | 953 | 65 | 0.141 | 0.141 | 0.126 | 0.211 | **0.082** | 0.114 |
| SwedishLeaf | 15 | 500 | 625 | 129 | 0.211 | 0.157 | 0.278 | 0.229 | 0.087 | **0.070** |
| Symbols | 6 | 25 | 995 | 398 | 0.101 | 0.062 | 0.109 | 0.091 | **0.036** | 0.042 |
| synthetic_control | 6 | 300 | 300 | 60 | 0.120 | 0.017 | 0.017 | 0.107 | **0.007** | **0.007** |
| Trace | 4 | 100 | 100 | 275 | 0.240 | 0.010 | **0.000** | **0.000** | **0.000** | **0.000** |
| Two_Patterns | 4 | 1000 | 4000 | 128 | 0.093 | **0.002** | 0.004 | 0.741 | 0.003 | 0.005 |
| TwoLeadECG | 2 | 23 | 1139 | 82 | 0.253 | 0.132 | 0.014 | 0.075 | **0.003** | 0.048 |
| uWaveGestureLibrary_X | 8 | 896 | 3582 | 315 | 0.261 | 0.227 | 0.323 | 0.316 | **0.200** | 0.226 |
| uWaveGestureLibrary_Y | 8 | 896 | 3582 | 315 | 0.338 | 0.301 | 0.364 | 0.412 | **0.287** | 0.303 |
| uWaveGestureLibrary_Z | 8 | 896 | 3582 | 315 | 0.350 | 0.322 | 0.356 | 0.353 | **0.269** | 0.279 |
| Wafer | 2 | 300 | 3000 | 426 | 0.005 | 0.005 | **0.001** | 0.003 | 0.004 | 0.013 |
| WordsSynonyms | 25 | 267 | 638 | 270 | 0.382 | **0.252** | 0.440 | 0.542 | 0.340 | 0.353 |
| Yoga | 2 | 300 | 3000 | 426 | 0.170 | 0.155 | 0.151 | 0.335 | **0.150** | 0.165 |
| # of best (including ties) | | | | | 2 | 9 | 7 | 2 | 19 | 15 |
| Wilcoxon Test p-values (RPM vs Other) | | | | | 0.006 | 0.287 | 0.217 | 0.002 | 0.834 | - |

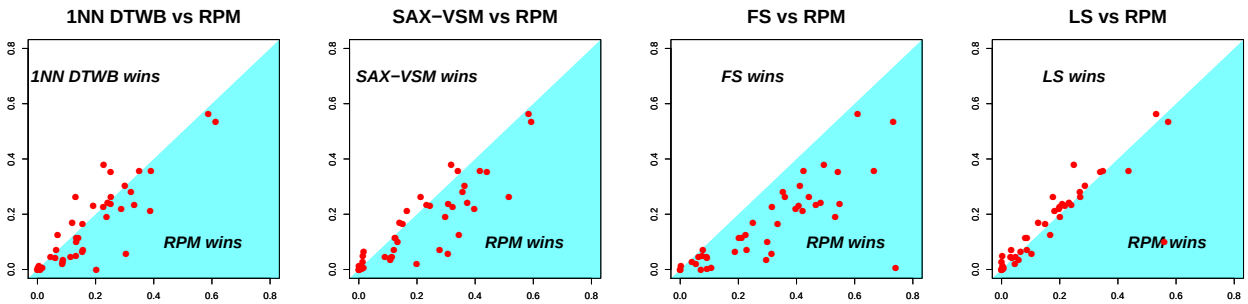


**Figure 7:** Our technique and current state of the art classifiers performance comparison.

of 95%. Moreover, as we can visually inspect from Figure 7, the error rate difference between Learning Shapelets and RPM is very small — most of the points are located around the diagonal. In the next section, we will show that our method is much faster than the Learning Shapelets method.

## 5.3 Efficiency

The pre-processing, discretization and Sequitur grammar induction all have linear time complexity in the size of training data and, in practice, can be done simultaneously, since they process data sequentially. To select the best representative patterns for each class, we need to first cluster the candidate motifs identified by the grammar rules. The time required for clustering depends on the number of motif instances identified. Suppose there are (on average) $u$ motif instances in each grammar rule, then the complexity is $O(u^3 \cdot |rules|)$ for hierarchical clustering. The centroids of the qualifying clusters are the pattern candidates. For each pattern candidate, we perform subsequence matching to identify the best matches ($O(|Candidates| \cdot |Train|)$). Even though the number of pattern candidates for each class is relatively small compared to the training size, this step seems to be the bottleneck of the training stage due to the repeated distance call. We use early abandoning strategy [32] to speed up the subsequence matching, but other options are possible such as approximate matching. The training needs to be repeated for each class. Thus, the training complexity is $O(|Train| + c \cdot (u^3 \cdot |rules| + |Candidates| \cdot |Train|))$, where $c$ is the number of classes, and $u$ is the average number of motif instances in each grammar rule.

If the best SAX parameters are known, our algorithm is fast — in this case, simply using the classification method described in Section 3.1 completes the classification task. To get the best SAX parameters, we evaluated cross-validation based parameter selection techniques, exhaustive search and DIRECT described in section 4.1 and 4.2. Exhaustive search was found time-consuming even with early abandoning, therefore we use DIRECT. The overall running time using DIRECT in the worst case is $O((|Train| + c \cdot (u^3 \cdot |rules| + |Candidates| \cdot |Train|)) \cdot R)$, where $R$ is the number of SAX parameters combinations tested by DIRECT algorithm. From the experiments on 42 UCR time series datasets, the average value for $R$ is less than 200, which is smaller than the average time series length 363. In most of the $R$ evaluations, the program terminated search early because of the minimum motif frequency requirement (Sec. 3.2). The classification time, compared to training, is negligible.

We compare the total running time of our algorithm using DIRECT with that of Fast Shapelets (FS) and Learning Shapelets (LS), and the results are shown in Table 2. Even when accounting for parameter selection, our algorithm is comparable to Fast Shapelets in running time, and as shown in Table 1, our method is significantly more accurate than Fast Shapelets (p-value equals 0.002). Compared to Learning Shapelets, our method is a lot faster. The greatest speedup we achieve through these 42 datasets is 1587X on dataset Adiac, and the average speedup is 178X. The experiment results show that our method is comparable to the fastest algorithm in speed and to the most accurate algorithm in accuracy.

In Section 3.2.3, we use $\tau$ as the threshold to remove similar patterns. We choose the value at the $30^{th}$ percentile of the pair-wise distances as the threshold. We also compare the results using the $10^{th}$, $50^{th}$, $70^{th}$, and $90^{th}$ percentiles. The running time and classification error changes are shown in Figure 9. The average running time and classification error changes on the 42 UCR data set are shown in Table 3. The average standard deviation for running time is 268.71 seconds, and 0.014 for classification error .

**Table 2:** Running time and classification accuracy comparison between Fast Shapelets, Learning Shapelets and Representative Pattern Mining

| | Running Time (Seconds) | | |
|---|---|---|---|
| Dataset | LS | FS | RPM |
| 50words | 3396298 | **1666** | 4221 |
| Adiac | 1551130 | **290** | 977 |
| Beef | 5971 | **175** | 202 |
| CBF | 275 | **7** | 32 |
| ChlorineConcentration | 7668 | 572 | **347** |
| CinC_ECG_torso | 46979 | 3521 | **1636** |
| Coffee | 293 | **15** | 98 |
| Cricket_X | 252834 | 2286 | **438** |
| Cricket_Y | 249889 | 2378 | **1208** |
| Cricket_Z | 260107 | 2611 | **594** |
| DiatomSizeReduction | 1013 | **17** | 69 |
| ECG200 | 224 | **12** | 55 |
| ECGFiveDays | 41 | **3** | 20 |
| FaceAll | 93442 | **538** | 2139 |
| FaceFour | 1853 | 69 | **43** |
| FacesUCR | 30516 | **195** | 2141 |
| Fish | 42766 | 802 | **755** |
| Gun_Point | 209 | **6** | 34 |
| Haptics | 81751 | 8100 | **1575** |
| InlineSkate | 314244 | 43930 | **4970** |
| ItalyPowerDemand | 14 | **1** | 9 |
| Lightning2 | 1657 | 1212 | **373** |
| Lightning7 | 7923 | 219 | **93** |
| MALLAT | 65920 | 1645 | **267** |
| MedicalImages | 19864 | **136** | 555 |
| MoteStrain | 22 | **1** | 13 |
| OliveOil | 2499 | **123** | 287 |
| OSULeaf | 31181 | 2337 | **154** |
| SonyAIBORobotSurface | 34 | **1** | 6 |
| SonyAIBORobotSurfaceII | 44 | **1** | 10 |
| SwedishLeaf | 83656 | **317** | 3312 |
| Symbols | 3043 | **69** | 328 |
| synthetic_control | 2616 | 37 | **18** |
| Trace | 6104 | **115** | 185 |
| Two_Patterns | 11219 | **601** | 1241 |
| TwoLeadECG | 24 | **1** | 19 |
| uWaveGestureLibrary_X | 267727 | 5060 | **274** |
| uWaveGestureLibrary_Y | 373482 | 4429 | **567** |
| uWaveGestureLibrary_Z | 409494 | 4230 | **619** |
| Wafer | 2746 | **217** | 1585 |
| WordsSynonyms | 852394 | **877** | 2271 |
| Yoga | 4414 | 2388 | **642** |
| # best (including ties) | 0 | 24 | 18 |

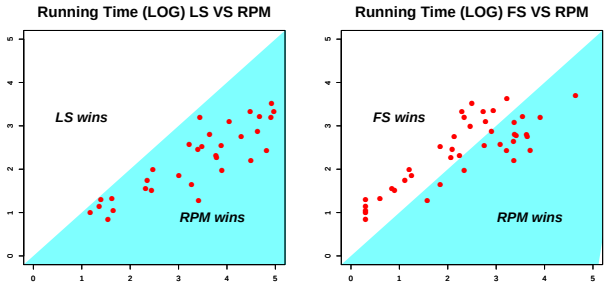


**Figure 8:** Runtime comparison between Representative Patterns, Fast Shapelets, and Learning Shapelets classifiers.

The average classification accuracy change with different $\tau$ values is below 1%. That means this parameter does not affect the classification result too much. The user can set a

**Table 3:** The average running time and classification error changes for different similar threshold on 42 UCR data. Positive value means increase, negative value means decrease.

| | 10% - 30% | 30% - 50% | 50% - 70% | 70% - 90% |
|---|---|---|---|---|
| Running Time Change (%) | -4.66 | -12.38 | -15.09 | -1.17 |
| Error Change (%) | -0.14 | 0.74 | 0.72 | 0.28 |

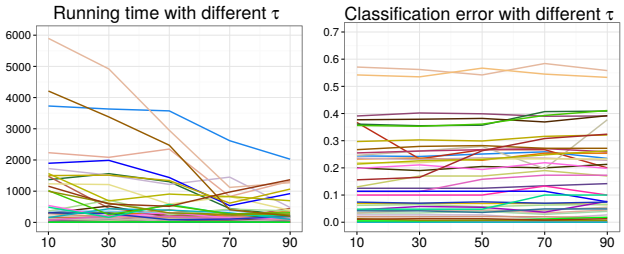


**Figure 9:** The running time and accuracy with different similarity threshold $\tau$.

higher value for this parameter to achieve a fast speed and still could maintain a high accuracy.

The $\tau$ value used for the experiment results (classification error and running time) shown in Tables 1 and 2 is set as 30% because it gives the best accuracy and still has a fast running speed.

# 6. CASE STUDY

In this section we demonstrate the application of our method to classification of rotated time series data and Medical Alarm data. We compare results with 1NN Euclidean distance, 1NN DTW with the best warping window, SAX-VSM, Fast Shapelets, and Learning Shapelets classifiers.

## 6.1 Rotation invariance

Many global-distance-based techniques do not work well if the time series data are shifted or out of phase. One type of time series data that is particularly susceptible to this kind of distortion is shape-converted time series [19], e.g. by radial scanning of the shape profile to convert the image into a "time series." Several datasets in the UCR repository are shape-converted time series, e.g. OSU Leaf, Swedish Leaf, Shields, etc. In this section we demonstrate the rotation- or shift-invariance of our technique on a number of shifted datasets. To shift or "rotate" a time series, we randomly choose a cut point in the time series, and swap the sections before and after the cut point. This transformation is equivalent to starting the radial scanning of the shape at a different position on the shape profile. The out-of-phase phenomenon is also common in many other real-world time series data such as video. Figure 10 illustrates the original GunPoint dataset time series and their rotation.

In our experiments, we leave the training set unmodified, and shift only the test data. The rationale is that while it is not uncommon for one to pre-process and create a "cleaned" version of training data to build a good model, it is less reasonable to expect that the test data will be in the same cleaned format. In other words, we learn the patterns on existing training data, but modify the test data to create rotation distortion, in order to evaluate the robustness of our technique.

It is possible that the rotation cuts the best matching subsequence of the test time series. To handle this, we introduce

**Table 4:** Classification error rate on shifted time series

| Dataset | 1NN-ED | 1NN-DTWB | SAX-VSM | LS | RPM |
|---|---|---|---|---|---|
| Coffee | 0.536 | 0.460 | **0.000** | 0.036 | **0.000** |
| Face Four | 0.682 | 0.625 | 0.125 | 0.080 | **0.045** |
| Gun point | 0.460 | 0.493 | **0.047** | 0.200 | **0.047** |
| Swedish Leaf | 0.872 | 0.821 | 0.430 | 0.371 | **0.246** |
| OSU Leaf | 0.595 | 0.479 | **0.107** | 0.186 | 0.157 |
| # best (including ties) | 0 | 0 | 3 | 0 | 4 |

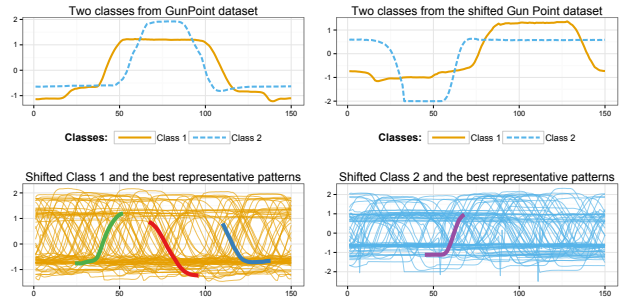


**Figure 10:** Shifted GunPoint dataset and the best representative patterns.

a new strategy to the test time series transformation step to make our algorithm rotation invariant. When transforming a raw time series into the new feature space of the closest match distances, our algorithm needs to calculate the distance between the time series to each representative pattern. In order to solve the aforementioned problem, our algorithm will build another time series. For example, when transforming a rotated time series $A$, we generate another new time series $B$ by cutting $A$ from its midpoint and swapping the first and the second halves. By doing so, $B$ will contain the concatenation of $A$'s tail and head. If the best-matching subsequence happens to be broken up due to the rotation ($A$), then by rotating it again at the midpoint, one of $A$ or $B$ will contain the entirety of the best-matching subsequence. When computing the distance of $A$ to a pattern $p$, besides calculating a distance $d_a$ between $A$ and $p$, the algorithm will also calculating another best match distance $d_b$ between $B$ and $p$. The minimal distance of these two will be used as the distance from $A$ to $p$. This solution overcomes the potential problem that arises when the best matching pattern is cut into different parts due to the rotation.

The classification error rates of the rotated data are shown in Table 4. The accuracy of our method or SAX-VSM does not change very much from the unrotated version (though our technique seems to be more robust, with 4 wins), while the error rates of 1NN Euclidean distance and 1NN DTWB increase drastically.

## 6.2 Medical Alarm

In this case study, we use the medical alarm data from Intensive Care Unit (ICU) database (MIMIC II database from PhysioNet) [6]. We used arterial blood pressure (ABP) waveforms to create the dataset used in this work.

### *6.2.1 Normal or Alarm*

We selected two types of ABP series segments: those that triggered an alarm and those that did not. The data used are all from the same patient. The selected dataset contains 52 time series of length of 2126. Each of the time series represents a segment of arterial blood pressure (ABP) waveform

in a 17 second time range. The *Normal* class consists of segments without any alarms, whereas the *Alarm* class consists of segments that triggered the bedside patient monitors and were verified by the domain expert as true alarms. The dataset contains 26 time series in class *Normal* and 26 in class *Alarm*. Training and test data are split into sets of 16 and 36 time series respectively. Examples of medical alarm data from each class are shown in Figure 11.

The first row of Table 5 shows the accuracy of competing classification techniques. Figure 11 shows the representative patterns from medical alarm time series and their best matches on test data. Our method (RPM) achieves the best accuracy on this dataset.

**Table 5:** Classification error rate on Medical Alarm data

| Dataset | 1NN-ED | 1NN-DTWB | SAX-VSM | FS | LS | RPM |
|---|---|---|---|---|---|---|
| NormalOrAlarm | 0.333 | 0.333 | 0.167 | 0.306 | 0.111 | **0.056** |
| FiveAlarmTypes | 0.760 | 0.360 | 0.350 | 0.485 | **0.260** | 0.300 |

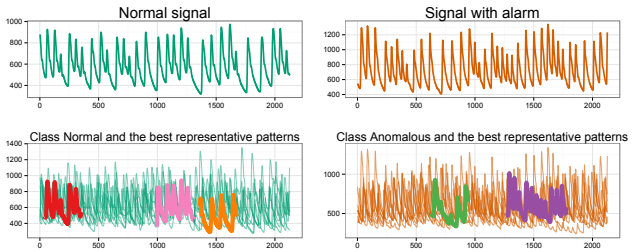


**Figure 11:** Normal or Alarm data and the best representative patterns.

### 6.2.2 *Five types of alarm*

In PhysioNet's MIMIC II database, there are five categories of critical arrhythmia alarms produced by a commercial ICU monitoring system: *Asystole*, *Extreme Bradycardia*, *Extreme Tachycardia*, *Ventricular Tachycardia*, and *Ventricular Fibrillation/Tachycardia*. We collected a dataset by taking the segments of an arterial blood pressure waveform. Each segment contains a verified alarm. The objective is to classify the alarm time series into one of the five types of alarms.

The training set has 50 examples, 10 for each class. The test set has 100 examples, 20 for each class. All time series have the same length of 2126 (17 seconds). The time series example of each class, the representative patterns, and the best matches are shown in Figure 12. Table 5 shows the accuracy of competing classification techniques for this dataset.

Our method (RPM) has the second best accuracy on this dataset. We lose slightly to Learning Shapelets since we have 30 incorrectly classified instances compare to 26 with LS. However, our method finished in 1373 seconds compare to 86195 seconds of LS. The speedup of our algorithm over LS is 63X on this dataset.

## 7. CONCLUSIONS

In this work, we propose a novel method to discover representative patterns of time series, specifically for the problem of classification. We demonstrate through extensive

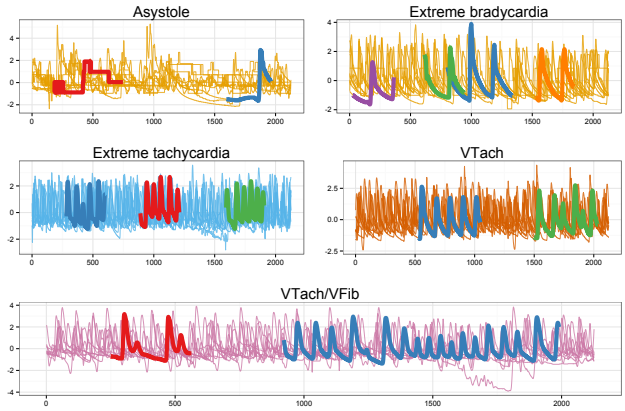


**Figure 12:** Five type of Medical Alarm and the best representative patterns.

experimental evaluation that our technique achieves competitive classification accuracy on the standard UCR time series repository, and is able to discover meaningful subspace patterns. The accuracy of our technique remains stable even when the data are shifted, while NN classifiers with global distance measures suffer from shift distortion. We also demonstrate that our technique outperforms existing techniques on a real-world medical alarm data that is extremely noisy.

In terms of efficiency, while our approach is competitive or better than other techniques, there are other optimization strategies that we can consider to speed up the algorithm even further. From profiling, we identified the bottleneck of the algorithm, which can be improved by adapting the state-of-the-art subsequence matching strategies [26]. Also, we used Euclidean distance as the base distance function for pattern matching. We will consider a more robust distance measure such as DTW in future work.

## 8. ACKNOWLEDGMENTS

This research is partially supported by the National Science Foundation under Grant No. 1218325 and 1218318.

## 9. REFERENCES

[1] Webpage to download the code and dataset. *http://mason.gmu.edu/~xwang24/Projects/RPM.html*.

[2] R. Briandet, E. K. Kemsley, and R. H. Wilson. Discrimination of arabica and robusta in instant coffee by fourier transform infrared spectroscopy and chemometrics. *Journal of agricultural and food chemistry*, 44(1):170–174, 1996.

[3] B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *Proc. of 9th ACM SIGKDD Intl. Conf.*, 2003.

[4] 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. *Proceedings of the VLDB Endowment*, 1(2):1542–1552, 2008.

[5] P. Geurts. Pattern extraction for time series classification. In *Principles of Data Mining and Knowledge Discovery*, pages 115–127. Springer, 2001.

[6] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000.

[7] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 392–401. ACM, 2014.

[8] M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[9] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[10] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, 2014.

[11] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, 1993.

[12] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286, 2001.

[13] E. Keogh and S. Kasetty. On the need for time series data mining benchmarks: a survey and empirical demonstration. *DMKD*, 7(4):349–371, 2003.

[14] E. Keogh, X. Xi, L. Wei, and C. A. Ratanamahatana. The UCR time series classification/clustering homepage. *http://www.cs.ucr.edu/~eamonn/time_series_data*.

[15] H. Kremer, S. Günnemann, A. Held, and T. Seidl. Effective and robust mining of temporal subspace clusters. In *ICDM*, pages 369–378, 2012.

[16] X. Li, E. Keogh, L. Wei, and A. Mafra-Neto. Finding motifs in a database of shapes. In *SDM*, pages 249–260, 2007.

[17] Y. Li, J. Lin, and T. Oates. Visualizing variable-length time series motifs. In *SDM*, pages 895–906, 2012.

[18] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 2007.

[19] J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using Bag-of-Patterns representation. *Journal of Intelligent Inform. Systems*, 39, 2012.

[20] A. Mueen, E. Keogh, and N. Young. Logical-shapelets: an expressive primitive for time series classification. In *Proc. of 17th ACM SIGKDD Intl. Conf.*, 2011.

[21] A. Nanopoulos, R. Alcock, and Y. Manolopoulos. Feature-based classification of time-series data. *International Journal of Computer Research*, 10(3), 2001.

[22] S. Naoki. *Local feature extraction and its application using a library of bases*. Ph.D. thesis, Yale University, 1994.

[23] C. G. Nevill-Manning and I. H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res.(JAIR)*, 7:67–82, 1997.

[24] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[25] M. Radovanovic, A. Nanopoulos, and M. Ivanovic. Time-series classification in many intrinsic dimensions. In *SDM*, pages 677–688, 2010.

[26] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 262–270, New York, NY, USA, 2012.

[27] T. Rakthanmanon and E. Keogh. Fast shapelets: A scalable algorithm for discovering time series shapelets. In *Proc. of SDM*, 2013.

[28] C. A. Ratanamahatana and E. Keogh. Making time-series classification more accurate using learned constraints. SIAM, 2004.

[29] G. Salton. *The SMART Retrieval System; Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[30] P. Senin, J. Lin, X. Wang, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen, S. Frankenstein, and M. Lerner. Grammarviz 2.0: a tool for grammar-based pattern discovery in time series. In *Proc. ECML/PKDD*. 2014.

[31] P. Senin and S. Malinchik. SAX-VSM: Interpretable time series classification using sax and vector space model. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1175–1180. IEEE, 2013.

[32] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[33] L. Wei and E. Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 748–753. ACM, 2006.

[34] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040. ACM, 2006.

[35] Z. Xing, J. Pei, S. Y. Philip, and K. Wang. Extracting interpretable features for early classification on time series. In *SDM*, volume 11, pages 247–258. SIAM, 2011.

[36] 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*, pages 947–956. ACM, 2009.