STIFE: A Framework for Feature-Based Classification of Sequences of Temporal Intervals

Leon Bornemann[1(✉)], Jason Lecerf[2], and Panagiotis Papapetrou[3]

[1] Freie Universität Berlin, Berlin, Germany
leon.bornemann13@gmail.com
[2] Institut National des Sciences Appliquées Lyon, Lyon, France
[3] Department of Computer and Systems Sciences,
Stockholm University, Stockholm, Sweden

Abstract. In this paper, we study the problem of classification of sequences of temporal intervals. Our main contribution is the STIFE framework for extracting relevant features from interval sequences to build feature-based classifiers. STIFE uses a combination of basic static metrics, shapelet discovery and selection, as well as distance-based approaches. Additionally, we propose an improved way of computing the state of the art IBSM distance measure between two interval sequences, that reduces both runtime and memory needs from pseudo-polynomial to fully polynomial, which greatly reduces the runtime of distance based classification approaches. Our empirical evaluation not only shows that STIFE provides a very fast classification time in all evaluated scenarios but also reveals that a random forests using STIFE achieves similar or better accuracy than the state of the art k-NN classifier.

1 Introduction

Sequences of temporal intervals are ubiquitous in a wide range of application domains including sign language transcription [12], human activity monitoring, music informatics [10], and healthcare [3]. Their main advantage over traditional discrete event sequences is that they comprise events that are not necessarily instantaneous, but may have a time duration. Hence, sequences of temporal intervals can be encoded as a collection of labeled events accompanied by their start and end time values. It becomes apparent that in such sequences events may overlap with other events forming various types of temporal relations [12].

Examples. An example of such a sequence is depicted in Fig. 1, consisting of seven event intervals that have various time durations. Note that each event may occur several times in the sequence (e.g., events 0 and 1). Hence, a sequence of temporal intervals can be seen as a series of event labels (y axis) that can be active or inactive at a particular time point (x axis). Such sequences can appear in various application areas. One example is sign language [12]. A sentence expressed in signs consists of multiple, different gestures (e.g., head-shake, eyebrow-raise) or speech tags (e.g., noun, wh-word), which may have a time

T. Calders et al. (Eds.): DS 2016, LNAI 9956, pp. 85–100, 2016.
DOI: 10.1007/978-3-319-46307-0_6

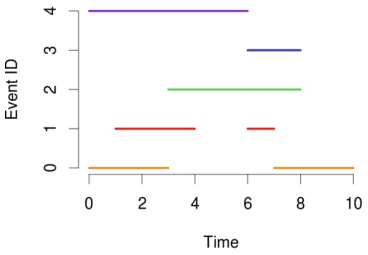

Fig. 1. Example of a sequence of temporal intervals: on the y axis we have five events, labeled as $\{0, 1, 2, 3, 4\}$, while on the x axis we can see the time points measured in seconds.

duration and can start at potentially different points in time. Another example is healthcare [3]. A sequence may correspond to a series of different types of treatments (events) for a particular patient. Treatments typically have a time duration and a patient could potentially be exposed to multiple treatments concurrently. An interesting and at the same challenging task involving sequences of temporal intervals is that of classification. For example, the correct classification of sign language videos can lead to the discovery of associations of certain types of expressions (labels) with various temporal combinations of gestural or grammatical events (features). Moreover, the extraction of certain combinations of treatments (features) may assist in the proper identification of adverse drug events (labels).

Previous research in the area of classification of sequences of temporal intervals has been limited to k-NN classifiers. Towards this direction, two state-of-the-art distance measures have been proposed, `Artemis` [5] and `IBSM` [6]. The first one quantifies the similarity between two sequences by measuring the fraction of temporal relations shared between them using a bipartite graph mapping, while ignoring their individual time duration. The second measure maps sequences of temporal intervals to vectors, where each time point is characterized by a binary vector indicating, which events are active at that particular time point. While the results obtained by both k-NN classifiers are promising, such classifiers still suffer from the fact that they consider only global trends or features in the data while ignoring local distinctive properties that may have a detrimental effect in predictive performance. Additionally the classification time of any k-NN classifier will always be at least linear to the size of the training set, while the computational cost of the chosen distance measure may severely impact the total runtime (e.g., `Artemis` is a distance measure with cubic computational complexity).

In this paper, we approach the problem of classification of sequences of temporal intervals by focusing on feature-based classifiers. Hence, the challenge is to identify and extract useful features from the sequences that could be then be used as input to traditional feature-based classifiers.

Contributions. Our main contributions can be summarized as follows: (1) we propose STIFE (Sequences of Temporal Intervals Feature Extraction

Framework), a novel framework for feature extraction from sequences of temporal intervals, and discuss its runtime complexity; (2) we present an improved method for calculating the `IBSM` distance, hence substantially reducing both runtime and memory requirements; (3) we provide an extensive empirical evaluation using eight real datasets as well as synthetic data, in which we compare our novel methods against the state-of-the-art.

2 Related Work

While arguably an understudied research area, sequences of temporal intervals have attracted some attention within the areas of data mining and databases. The first attempts at using sequences of temporal intervals mainly focused on simplifying the data without losing too much information. For example Lin et al. [8] show a way to mine maximal frequent intervals, but while doing so the different dimensions of the intervals were discarded. Another common form of simplification is to map sequences of temporal intervals to temporally ordered events without considering the actual duration of the intervals [1].

A large variety of Apriori-based techniques [2,7,9] for finding temporal patterns, episodes, and association rules on interval-based event sequences have been proposed. In addition, more advanced candidate generation techniques and tree-based structures have been employed by various methods [11–13], which apply efficient pruning techniques, thus reducing the inherent exponential complexity of the mining problem, while a non-ambiguous event-interval representation is defined by [14] that considers start and end points of event sequences and converts them to a sequential representation. The main weakness of performing such mapping is the fact that the candidate generation process becomes more cumbersome while introducing redundant patterns.

An approach for mining patterns of temporal intervals without performing any mapping to instantaneous events has been proposed by Papapetrou et al. [12]. The authors of this paper applied unsupervised learning methods to sequences of temporal intervals. In particular the Apriori algorithm for mining frequent item-sets had been adapted to fit sequences of temporal intervals. Subsequent to that, similarity of sequences of temporal intervals became more popular and has been looked at quite a bit. Robust similarity measures allow the use of sequences of temporal intervals as a data-basis for a lot of applications, some of them being similarity search, clustering and classification through k-NN classifier. It started with [5], where the authors propose two different similarity measures. The first one maps sequences of temporal intervals to time series data, the second one uses the temporal relations to construct and use a bipartite graph. This approach has been improved through different data representation and more robust similarity measures in [6].

With the transformation from sequences of temporal intervals to time series data being explored, it is unsurprising that also finding the longest common subpattern (LCSP) in sequences of temporal intervals has recently been considered. Finding the longest common subsequence (LCS) is a classic problem for time

series data and finding the LCSP can be seen as its equivalent counterpart for sequences of temporal intervals. The problem of finding the LCSP was introduced in [4], where the authors prove its NP-hardness, introduce approximation algorithms as well as upper bounds.

3 Background

Let Σ define the alphabet of all possible events, i.e., the different types of intervals. A temporal interval is defined as: $I = (d, s, e)$ where $d \in \Sigma$ is an event label, and $s, e \in \mathbb{N}^+$ are the start and end times of the event interval, with $e \geq s$. Given an event interval $I = (d, s, e)$, we will sometimes denote d, s and e as $I.d, I.s$ and $I.e$, respectively.

A sequence of temporal intervals S is defined as an ordered multi-set of temporal intervals: $S = \{I_1, ..., I_m\}$. Note that it is allowed for multiple event intervals of the same label to overlap in a sequence. Further, a dataset of sequences of temporal intervals is denoted as $\mathcal{D}$.

The original IBSM method [6] represents a sequence S in a $|\Sigma| \times length(S)$ matrix called event tables, where $length(S)$ is the duration of the sequence (e.g. the time value at which the last interval stops). We briefly repeat the most important definitions next.

Definition 1. ***Active Interval.*** *Given a sequence S, an Interval $I \in S$ and a point of time t, I is called active at point of time t if $I.s \leq t \leq I.e$.*

Definition 2. ***Event Table.*** *Given a sequence S, its Event Table ET is defined as a $|\Sigma| \times length(S)$ matrix. The value of $ET(d, t)$ is the number of Intervals in S of dimension d that are active at point of time t. When we speak of the length of an event table we refer to the length of its corresponding sequence: $length(ET) = length(S)$.*

For the rest of this paper we assume that all sequences are of the same length, since this makes the definition of the distance measure easier. Note that this is not a big constraint, since sequences of smaller length can be interpolated to sequences of bigger length, by using linear interpolation. This was also suggested in the original definition of IBSM. The original distance between two event tables of the same length (number of columns) is called the IBSM-Distance.

Definition 3. ***IBSM-Distance.*** *Given two event tables A and B where $length(A) = length(B) = z$ the IBSM-Distance is defined as*

$$IBSM(A, B) = \sqrt{\sum_{d=1}^{|\Sigma|} \sum_{t=1}^{z} (A(d, t) - B(d, t))^2}$$

To counteract the large size of the event tables the authors suggest sampling methods which improve computation time but come at the cost of accuracy for the 1-NN classifier.

4 Compressed IBSM

The key idea of compressing IBSM without losing information is to reduce the size of the event table by only considering the points during which the value of a row can change, which are the start and end times of an event interval $I \in S$.

Definition 4. ***Time Axis.*** *Given a sequence S, let $T = \{t_1, ..., t_k\}$ be the sorted set of the start and end times of all intervals $I \in S$. We call T the time axis of S.*

Given a time axis $T = \{t_1, t_2, ..., t_k\}$ of sequence S we know according to the definition that $t_i < t_{i+1}$ for $i \in \{1, ..., k-1\}$. It is clear that for all $t \in \{1, .., length(S)\}$ where $t_i < t < t_{i+1}$: column t of the event table is equal to column t_i. Note that $k \leq 2m$ always holds, since k can at most be $2m$ but may be less since intervals in S can have the same start or end time. This allows us to just store the columns for all $t \in T$ and the other columns are implicitly given. We call the optimized form compressed event tables.

Definition 5. ***Compressed Event Table (CET).*** *Given a sequence S and its time axis $T = \{t_1, t_2, ..., t_k\}$ we define CET as the compressed event table of S as a $|\Sigma| \times |T|$ matrix where $CET(d, t_i)$ is the number of intervals in S of dimension d that are active at point of time t_i.*

Table 1. Uncompressed event table

	1	2	3	4	5	6	7	8	9
D_1	1	1	1	0	0	0	0	0	0
D_2	0	0	0	0	1	1	1	1	0

Table 2. Compressed event table

	1	4	5	9
D_1	1	0	0	0
D_2	0	0	1	0

Tables 1 and 2 present the different representations for a simple, small example. The distance between two compressed event tables can be calculated as follows:

Definition 6. ***IBSM distance for Compressed Event Tables.*** *Given two compressed event tables A and B with time axis $T_A = \{ta_1, ..., ta_k\}$ and $T_B = \{tb_1, ..., tb_p\}$, where $ta_k = tb_p$ (sequences have the same length) let $T = \{t_1, ..., t_r\} = T_A \cup T_B$ be the merged time axis (still ordered). Then we define the distance between the two compressed event tables as*

$$Dist(A, B) = \sqrt{\sum_{d=1}^{|\Sigma|} \sum_{j=1}^{|T|} E(A(d, I_A(t_j)), B(d, I_B(t_j))) \cdot \delta(j)} \tag{1}$$

where

$$
\begin{aligned}
I_A(t) &= max(\{i | t_i \in T_A \; t_i \leq t\}) \\
I_B(t) &= max(\{i | t_i \in T_B \; t_i \leq t\}) \\
E(a,b) &= (a-b)^2 \\
\delta(j) &= \begin{cases} t_{j+1} - t_j & \text{if } j < |T| \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
$$

The distance calculation now looks more complicated but the approach is straightforward. The squared error E is calculated for each cell of the table and is multiplied by the amount of time that the value would have been repeated in the old IBSM representation (δ). I_A and I_B are functions that map a point of time of the merged time axis T to the correct column index of their respective compressed event tables.

Given this definition, it is clear that given two sequences the event tables can be computed in $\Theta(m \cdot (log(m) + |\Sigma|))$. We need $\Theta(m \cdot log(m))$ to create the sorted time axis. Given two event tables the distance computation is linear to the number of cells in each table, which is $\Theta(|\Sigma| \cdot m)$. This is a clear improvement compared to the old $\Theta(|\Sigma| \cdot length(S))$, which is, as already mentioned, pseudo-polynomial.

5 Feature-Based Classification Through STIFE

While improving the performance of the distance measure is the key to improve the classification time of k-NN classifiers it can not address their overall disadvantage in classification time, which is that it will always be at least linear to the size of the training database. This can become problematic if the database is a huge size and classification of new instances is time critical. The rather broad application domain of real-time analysis of data-streams would be such an example.

Many feature based classifiers offer a classification time that is better than linear to the size of the database, such as decision trees or random forests. Thus if one is able to extract informative features from sequences of temporal intervals one could use these feature based classifiers to further improve classification time. An additional motivation besides time efficiency is that k-NN classifiers also have other drawbacks compared to feature based classifiers, such as sensitivity to outliers or units of measurements. Feature based classifiers might also yield better accuracy in some cases, depending of course on the usefulness of the extracted features.

In order to extract useful features we propose a novel method which we call the STIFE (**S**equences of **T**emporal **I**ntervals **F**eature **E**xtraction) framework. The rest of this section gives a detailed explanation of the framework.

5.1 STIFE Framework Components

Given a number of sequences as a training database, the main challenge of the framework is to explore and find features, which help to classify the training database. To do so we propose the STIFE Framework, which consists of three parts: (I) Static metrics, (II) Shapelet extraction and selection, and (III) Distance to class-cluster center.

Static metrics are simple, basic mappings that map one sequence to a set of features independent of the other sequences in the database S. The other two parts are dynamic, which means they consider the whole (training) database to extract the features that are particularly helpful to classify the sequences of that specific database. Subsects. 5.2, 5.3 and 5.4 describe the parts of the framework in detail. Afterwards Subsect. 5.5 summarizes the framework's time and memory complexity for training and classification.

5.2 I - Static Metrics

Let $S = \{I_1, ..., I_m\}$ be a sequence in which the intervals are sorted by start time, and in case of a tie by end time. We define the following basic metrics that will serve as static features:

- Duration: $I_m.e$
- Earliest start: $I_1.s$
- Majority dimension: The dimension d that occurs in most intervals $I \in S$.
- Interval count: $|S|$
- Dimension count: $|\{I.d | I \in s\}|$
- Density: $\sum_{I \in S} I.e - I.s$
- Normalized density: Density divided by the duration of the sequence.
- Max. overlapping intervals: Maximum number of overlapping intervals.
- Max. overlapping interval duration: The duration of the period with the highest number of overlapping intervals.
- Normalized max. overlapping interval duration: Max. overlapping interval duration divided by the duration of the sequence.
- Pause time: The total duration with no active dimension interval.
- Normalized pause time: Pause time divided by the duration of the sequence.
- Active time: The reverse of pause time.
- Normalized active time: Active time divided by the duration of the sequence.

These static metrics provide a very basic method to obtain some features. They are simple to understand, fast to compute and require little memory compared to the original training database. After sorting the intervals of the sequence all of these metrics can be calculated in either $\Theta(1)$ or $\Theta(m)$. Thus the overall runtime complexity of extracting static features from the database is $\Theta(n \cdot m \cdot log(m))$. Only $\Theta(n)$ additional memory is needed, since the number of static features is constant. The time to extract the features for an unseen sequence S_{new} is $\Theta(m \cdot log(m))$.

5.3 II - Shapelet Extraction and Selection

Shapelets are commonly defined as interesting or characteristic small subsequences of a larger sequence. The idea of shapelets has already been explored in the context of time-series data and has also been used as a tool for classification of time series data. Thus it is natural to also consider shapelets as candidates for features of sequences of temporal intervals. In this paper we will restrict ourselves to the shapelets of size 2 which are in the following referred to as 2-shapelets. To be able to define a 2-shapelet of a sequence of temporal intervals we must first define a few prerequisites such as temporal relationships between two intervals:

Definition 7. ***Time Equality Tolerance.*** *We define* $\epsilon \in \mathbb{N}^+$ *as the maximum tolerance which time values may differ from each other to still be considered as equal from a view point of temporal relationships. Since the value of* ϵ *can be quite domain specific we do not specify a fixed value here.*

Given the time equality tolerance we can define temporal relationships between temporal intervals:

Definition 8. ***Temporal Relationship.*** *Let* A *and* B *be two intervals with the following property:* $A.s - \epsilon \leq B.s$ *(*B *does not start before* A*). Then we define the set of possible temporal relationships as* $R = \{$*meet, match, overlap, leftContains, contains, rightContains, followedBy*$\}$*. Their individual definition is visualized in Fig. 2.*

These temporal relations for event intervals have already been used in the context of distance measures for sequences of temporal intervals on multiple occasions [5,6]. Note that for an ordered pair of event intervals exactly one of these relations applies, meaning the temporal relationship of two event intervals is unambiguous. Based on this, a 2-shapelet can be defined.

Definition 9. ***2-shapelet.*** *Given a sequence* S *and two temporal intervals* $A, B \in S$ *we define a 2-shapelet as* $sh = (A.d, B.d, r)$ *where* $r \in R$ *is the*

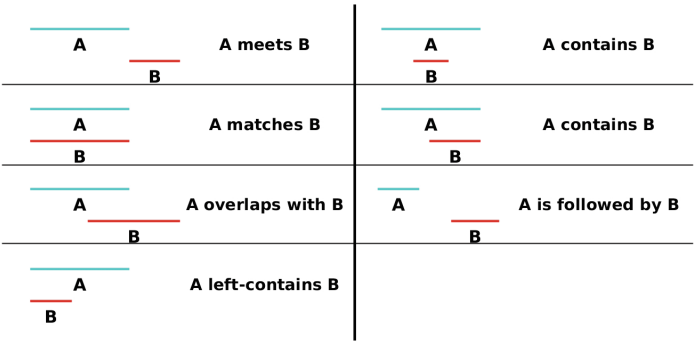

Fig. 2. The 7 temporal relations between an ordered pair of event intervals (A,B)

Algorithm 1. Extract and select all shapelet features

Require: Let $S = \{S_1, ..., S_n\}$ be the sequences, Σ be the set of dimensions and k the number of features to keep.

```
SM ←new n × (7 · |Σ|²) matrix
for i = 1 to n do
    for (A, B) ∈ {(A, B) | A, B ∈ S ∧ A.s − ε ≤ B.s} do
        r ← temporal relationship of (A, B)
        j ← compute column index of shapelet (A.d, B.d, r)
        SM[i, j] ← SM[i, j] + 1
    end for
end for
gains ←new (7 · |Σ|²) array
for j = 1 to 7 · |Σ|² do
    gains[col] ← information gain of column j of SM
end for
featureIndices ← new List
for j = 1 to 7 · |Σ|² do
    if gains[j] is in the top k of gains then
        featureIndices.add(j)
    end if
end for
delete all columns of SM except for featureIndices
return SM
```

temporal relationship between the two intervals. In other words a two-shapelet (d_1, d_2, r) *says, that there are two intervals in* S *of the respective dimensions* d_1 *and* d_2 *that have the temporal relationship* r*.*

All 2-shapelets of a sequence S can be found by simply determining the relationships of all pairs of intervals (A, B), where $A, B \in S$ and B does not occur before A. The idea for the resulting features is simply to treat the number of occurences of each 2-shapelet as a feature of the sequence. This will result in exactly $|\Sigma| \cdot |\Sigma| \cdot |R| = 7 \cdot |\Sigma|^2$ possible features which is a swiftly increasing function of the number of dimensions. Thus it is necessary to perform feature selection afterwards which we do by information gain. Information gain is a measure of how much information is stored in an attribute with regard to the class label distribution and is commonly used when building decision trees. The formula is explained in detail in [15]. Since information gain is defined on categorical features and the number of shapelet occurrences in a sequence are numeric attributes, it is necessary to discretize them. We use the information gain of the best binary split (meaning the feature $\overrightarrow{a}$ is discretized to a vector of boolean values according to $\overrightarrow{a} \leq x$ for the $x \in \mathbb{N}$ that yields the highest information gain). The algorithm for the shapelet feature extraction is roughly summarized in Algorithm 1. To count all 2-shapelet occurences a $n \times 7 \cdot |\Sigma|^2$ matrix is used (one row per sequence). For each sequence all correctly ordered pairs need to be looked at, which amounts to the runtime $\Theta(m^2)$ per sequence, thus the

runtime for the shapelet occurence counting is $\Theta(n \cdot m^2)$. Memory requirement is $\Theta(n \cdot |\Sigma|^2)$.

Calculating the information gain of a numeric attribute needs $\Theta(n \cdot log(n))$. This is done for each feature, which means the total runtime of feature selection via information gain is $\Theta(n \cdot log(n) \cdot |\Sigma|^2)$. Memory remains at $\Theta(n \cdot |\Sigma|^2)$. Thus, putting the two steps together we arrive at $\Theta(n \cdot (m^2 + log(n) \cdot |\Sigma|^2))$ for runtime and $\Theta(n \cdot |\Sigma|^2)$ memory to execute shapelet extraction and select the best shapelets as features. Calculating the occurences of the selected 2-shapelets for a new sequence takes $\Theta(m^2)$ time in the worst case since once again all its correctly ordered interval pairs need to be considered. Note that this is always independent of $|\Sigma|$, since a constant number of shapelets are selected in the feature selection step.

5.4 III - Distance to Class-Cluster Center

Our approach here is inspired by the k-medoids clustering algorithm. Since clustering is an approach that is used in unsupervised learning and we are in the supervised case (e.g. we have data with class labels) it is unnecessary to actually execute the clustering algorithms. Instead we can just assume that we have the clusters given by the class-labels of the training data and simply extract the medoids of each class-cluster.

Given the medoids of each class-cluster, these will then be used as reference points and the distance to them will result in features. As a distance measure we choose the IBSM distance over ARTEMIS, since the compressed way of calculating it as introduced by us has a better runtime than ARTEMIS and 1-NN classifiers using IBSM yield better accuracy which leads us to believe that it is the more suitable distance measure. Given the distance measure we formulate the algorithm for distance based feature extraction in Algorithm 2.

Since the class labels (and thus cluster labels) are given, the clustering takes $\Theta(n)$ time. Afterwards we need to calculate the medoid of each cluster and subsequently calculate the distance to those for all training sequences. Assuming the number of classes is constant we know that the size of each cluster can be $\Theta(n)$ but the number of clusters is constant. For each cluster all compressed event tables (see Sect. 3) and their pairwise distances ($\Theta(n^2)$) need to be computed and stored. Thus the runtime and memory complexity of finding the distances to all class-cluster medoids is $\Theta(n^2 \cdot m \cdot (|\Sigma| + log(m)))$ time and $\Theta(n^2 \cdot m \cdot |\Sigma|)$ memory. The online feature extraction requires $\Theta(m \cdot (|\Sigma| + log(m)))$ time and $\Theta(m \cdot |\Sigma|)$ memory.

5.5 Runtime and Memory Complexity Overview

When analyzing runtime and memory complexity of the STIFE framework, the two interesting measures are training time and classification time. Extracting and selecting the features based on the training data adds to the classifier's training time. Since the framework can be used with any feature based classifier

Algorithm 2. Calculate all medoids and extract the distance to those as features

Require: Let $S = \{S_1, ..., S_n\}$ be the sequences, k the number of classes and $D : S \times S \mapsto \mathbb{R}^+$ a distance measure for sequences
  $FM \leftarrow$new $n \times k$ matrix
  for $c = 1$ **to** k **do**
    $S^{(c)} \leftarrow \{S_j \mid S_j \in S \wedge class(S_j) = c\}$
    $DM \leftarrow$new $|S^{(k)}| \times |S^{(k)}|$ matrix
    for $(S_i, S_j) \in \{(S_i, S_j) \mid S_i, S_j \in S^{(k)} \wedge i \leq j\}$ **do**
      $DM[i, j] \leftarrow D(S_i, S_j)$
      $DM[j, i] \leftarrow DM[i, j]$
    end for
    $min \leftarrow \infty$
    $minI \leftarrow -1$
    for $i = 1$ **to** $|S^{(c)}|$ **do**
      $dist \leftarrow$ row i of DM
      if $sum(dist) \leq min$ **then**
        $min \leftarrow sum(dist)$; $minI \leftarrow i$
      end if
    end for
    for $i = 1$ **to** n **do**
      $d \leftarrow D(S[i], S^{(c)}[minI])$; $FM[i, c] \leftarrow d$
    end for
  end for
  return FM

we will use $CTT(n)$ to describe the classifier training time and $CTM(n)$ to describe the classifier memory need for training.

For an unseen sequence, feature extraction is performed before the classifier can be applied. We will use the term $CCT(n)$ to describe the classifier classification time and $CCM(n)$ for the classifier classification memory need. The exact training and classification runtime and memory complexities have already been mentioned in the respective subsections. Table 3 presents upper bounds for the whole framework.

Table 3. Upper bounds for memory and runtime complexity of STIFE.

Task	Upper bound for complexity
Training Time	$O(m^2 \cdot n^2 \cdot \|\Sigma\|^2 + CTT(n))$
Training Memory	$O(m \cdot n^2 \cdot \|\Sigma\|^2 + CTM(n))$
Classification Time	$O(m \cdot log(m) + m \cdot \|\Sigma\| + CCT(n))$
Classification Memory	$O(m \cdot \|\Sigma\| + CCM(n))$

It can be observed that the biggest influencing factor besides the size of the database is the number of dimensions $|\Sigma|$. How many dimensions actually exist

in a data-set is once again dependent on the domain. If the number of dimensions is very high, the memory requirement of the shapelet extraction and selection step might not be practical (it is using an $n \times 7 \cdot |\Sigma|^2$ matrix). Since however the matrix is usually sparse, memory need could be reduced by using appropriate implementations.

6 Empirical Evaluation

Our evaluation consists of two parts. In Subsect. 6.1 we analyze classification time and accuracy for real-life data-sets and in Subsect. 6.2 we conduct experiments with synthetic data to analyze the individual performance of the proposed methods for specific parameter settings. The STIFE framework, classifiers and distance measures were implemented in java[1]. When evaluating STIFE, we used the random forest implementation of Weka.

6.1 Real Data-Sets

For our empirical evaluation we used eight publicly available data sets. Some basic information about each data set is given in Table 4. Note that many of these data-sets come from different domains, which is very relevant when judging the general applicability of classification algorithms based on the evaluation results.

Table 4. Basic properties of the data sets

Data-set	Size test & Training	# of classes	max # of intervals (m)	$\|\Sigma\|$	duration
ASL-BU	873	9	40	216	5901
ASL-BU-2	1839	7	93	254	14968
AUSLAN2	200	10	20	12	30
BLOCKS	210	8	12	8	123
CONTEXT	240	5	148	54	284
HEPATITIS	498	2	592	63	7555
PIONEER	160	3	89	93	80
SKATING	530	6	143	41	6829

The data-sets were evaluated for three classifiers using 10-fold cross validation. The three evaluated classifiers are 1-NN using the uncompressed (original) IBSM distance [6], 1-NN using our novel method of calculating the IBSM distance, in the following called compressed IBSM, and a random forest using the STIFE framework, in the following called STIFE-RF. For STIFE-RF the Time

[1] Implementation available at: https://github.com/leonbornemann/stife.

Table 5. Mean accuracy for 1-NN using the IBSM distance measure and a random forest using STIFE for feature extraction

Data-set	STIFE + Random forest accuracy [%]	IBSM accuracy [%]
ASL-BU	91.75	89.29
ASL-BU-2	**87.49**	76.92
AUSLAN2	**47.00**	37.50
BLOCKS	100	100
CONTEXT	99.58	96.25
HEPATITIS	**82.13**	77.52
PIONEER	98.12	95.00
SKATING	96.98	96.79

Table 6. Mean classification time for 1-NN using the IBSM and compressed IBSM distance as well as a random forest using STIFE for feature extraction

Data-set	STIFE + Random forest [ms]	Compressed IBSM [ms]	IBSM [ms]
ASL-BU	0.48	8.04	331.33
ASL-BU-2	0.47	22.46	1968.85
AUSLAN2	0.46	0.16	0.23
BLOCKS	0.15	0.07	0.15
CONTEXT	0.33	1.69	2.47
HEPATITIS	0.38	9.55	154.96
PIONEER	0.10	0.68	0.78
SKATING	0.18	4.36	97.31

Equality Tolerance (ϵ) as defined in Subsect. 5.3 was set to 5 and the amount of shapelet features to keep was set to 75. Furthermore the number of trees was set to 500 and the number of features per tree was set to $\sqrt{f}$, where f is the number of extracted features.

The results for the accuracy are presented in Table 5. Since both IBSM and compressed IBSM calculate the exact same distance value, both 1-NN classifiers also return the same accuracy which is why we only report one of them. The results for accuracy show that the random forest using STIFE is on par or better than the state of the art 1-NN classifier. Especially on data-sets that seem to be harder to classify (bold in the table) our novel method clearly beats the state of the art IBSM classifier.

When evaluating accuracy the ASL-BU and ASL-BU-2 were treated in a special manner, since they are multi-labeled data-sets, which means that each sequence can have multiple class-labels. This presents a difficulty when evaluating classifier accuracy. Since we introduce a novel method (Random forest + STIFE) we want to show that it is at least on par with the state of the art 1-NN

classifiers. Thus we chose a method of evaluation that is more lenient towards the 1-NN classifiers. For both classifiers we eliminated all sequences from the training database that have no class label. Subsequently we modified the training database for the random forest: we copy each sequence once for each of its class-labels and assign each copy exactly one class label. Example: If the sequence S has class labels $\{1, 2, 3\}$, the training database for the random forest will contain three instances of S with different class labels: $\{(S, 1), (S, 2), (S, 3)\}$. The training database of the 1-NN classifier remains unaltered (except for the removal of unlabeled sequences). Subsequently we redefine accuracy in the following: If a test sequence S has class labels A and a classifier predicts a set of class labels P, we say that the sequence was correctly classified, if $A \cap P \neq \emptyset$. Note that this is a definition that favors the 1-NN classifiers, since they will output all class labels of the nearest neighbour, while the random forest can only output exactly one class label. The fact that the random forest using SITFE still achieves better accuracy for both data-sets, although being at a disadvantage gives strong evidence that it may be superior to the 1-NN classifiers.

Table 6 reports the classification time of each of the three classifiers. The results show that compressed IBSM is always faster than IBSM. As expected due to the nature of the algorithms, the speedup is most significant for data-sets that contain high-duration sequences, namely ASL-BU, ASL-BU-2, HEPATITIS and SKATING. The runtime of our second approach, the random forest, while not always being faster is a lot more stable. It never exceeded a classification time of 1 millisecond for all of the data-sets.

6.2 Synthetic Data

There are four different parameters that are relevant for the classification runtime of the three studied classifiers. These are the size of the training database (n), the number of intervals per sequence (m), the number of dimensions ($|\Sigma|$) and the maximal duration of a sequence. In order to study their individual effects on the classification runtime we randomly generated sequences with fixed values for 3 of the four parameters while varying the fourth one. In order to study the impact of a parameter in a scenario close to reality we set each of the fixed parameters to the upper median of the eight data-sets described in 6.1. That way the fixed parameters that are kept constant reflect a "normal" task. The upper medians are: $n = 498$, $m = 93$, $|\Sigma| = 63$, $duration = 5901$. The results are depicted in Fig. 3. The plotted curves confirm that both compressed IBSM and STIFE-RF are independent of the sequence duration, as opposed to the original IBSM distance. Furthermore, compressed IBSM is faster than IBSM in all evaluated scenarios except for a very high number of intervals (given a fixed duration). On top of that STIFE-RF scales much better with the size of the training database (n) and the number of dimensions ($|\Sigma|$) than both 1-NN classifier. Lastly the plots show clearly that STIFE-RF is extremely fast in all scenarios: It's classification time never exceeds 3 ms, which makes its plotted curves look constant.

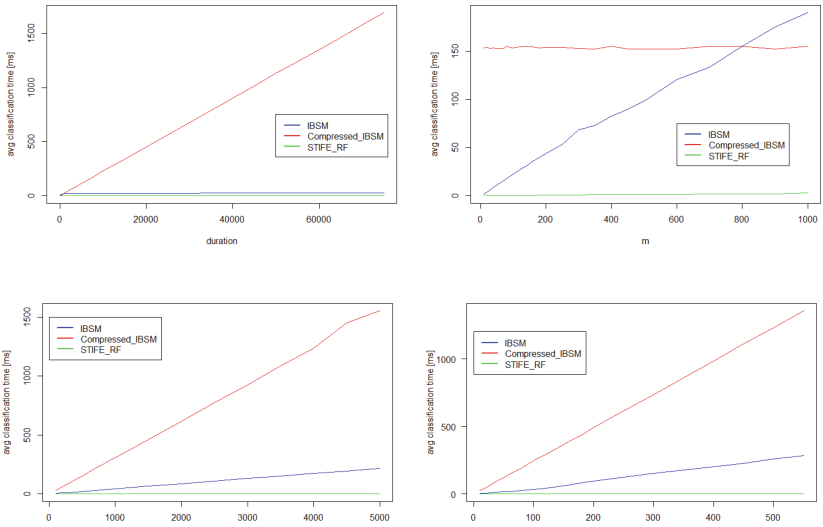

Fig. 3. Classifier performance for different parameters

7 Conclusions

Our main contribution in this paper is the formulation of the STIFE framework, a novel method that maps a sequence to a constant number of features, which can be used for classification. In addition, we presented an improved way of calculating the IBSM distance measure that reduces runtime and memory from the original pseudo-polynomial $\Theta(|\Sigma| \times length(S))$ to the fully polynomial $\Theta(m \cdot (log(m) + |\Sigma|))$. Our experimental evaluation on real and synthetic datasets showed that the STIFE framework using the random forest classifier outperforms the state-of-the-art 1-NN classifier using IBSM and compressed IBSM in terms of both classification accuracy and classification runtime. Directions for future work include the investigation of more elaborate feature selection techniques for selecting shapelets. Another direction is to compare the simple clustering by class of the distance based part of the framework to actual clustering methods and see if actually executing the k-medoids clustering algorithm results in medoids to which the distance is a more discriminative feature.

References

1. Giannotti, F., Nanni, M., Pedreschi, D.: Efficient mining of temporally annotated sequences. In: Proceedings of the 6th SIAM Data Mining Conference, vol. 124, pp. 348–359 (2006)
2. Höppner, F., Klawonn, F.: Finding informative rules in interval sequences. In: Hoffmann, F., Hand, D.J., Adams, N., Fisher, D., Guimaraes, G. (eds.) IDA 2001. LNCS, vol. 2189, pp. 125–134. Springer, Heidelberg (2001). doi:10.1007/3-540-44816-0_13

3. Kosara, R., Miksch, S.: Visualizing complex notions of time. Stud. Health Technol. Inform. **84**, 211–215 (2001)
4. Kostakis, O., Gionis, A.: Subsequence search in event-interval sequences. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 851–854. ACM (2015)
5. Kostakis, O., Papapetrou, P., Hollmén, J.: ARTEMIS: assessing the similarity of event-interval sequences. In: Gunopulos, D., Hofmann, T., Malerba, D., Vazirgiannis, M. (eds.) ECML PKDD 2011. LNCS (LNAI), vol. 6912, pp. 229–244. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23783-6_15
6. Kotsifakos, A., Papapetrou, P., Athitsos, V.: IBSM: interval-based sequence matching. In: Proceedings of SIAM Conference on Data Mining, pp. 596–604 (2013)
7. Laxman, S., Sastry, P., Unnikrishnan, K.: Discovering frequent generalized episodes when events persist for different durations. IEEE Trans. Knowl. Data Eng. **19**(9), 1188–1201 (2007)
8. Lin, J.L.: Mining maximal frequent intervals. In: Proceedings of the ACM Symposium on Applied Computing, pp. 624–629 (2003)
9. Mooney, C., Roddick, J.F.: Mining relationships between interacting episodes. In: Proceedings of the 4th SIAM International Conference on Data Mining (2004)
10. Pachet, F., Ramalho, G., Carrive, J.: Representing temporal musical objects and reasoning in the MusES system. J. New Music Res. **25**(3), 252–275 (1996)
11. Papapetrou, P., Kollios, G., Sclaroff, S., Gunopulos, D.: Discovering frequent arrangements of temporal intervals. In: Proceedings of IEEE International Conference on Data Mining, pp. 354–361 (2005)
12. Papapetrou, P., Kollios, G., Sclaroff, S., Gunopulos, D.: Mining frequent arrangements of temporal intervals. Knowl. Inf. Syst. **21**, 133–171 (2009)
13. Winarko, E., Roddick, J.F.: Armada - an algorithm for discovering richer relative temporal association rules from interval-based data. Data Know. Eng. **63**(1), 76–90 (2007)
14. Wu, S.Y., Chen, Y.L.: Mining nonambiguous temporal patterns for interval-based events. IEEE Trans. Knowl. Data Eng. **19**(6), 742–758 (2007)
15. Yang, Y., Pedersen, J.O.: A comparative study on feature selection in text categorization. ICML **97**, 412–420 (1997)