Time Series Classification by Sequence Learning in All-Subsequence Space

Thach Le Nguyen
Insight Centre for Data Analytics
University College Dublin, Ireland
thach.lenguyen@insight-centre.org

Severin Gsponer
Insight Centre for Data Analytics
University College Dublin, Ireland
severin.gsponer@insight-centre.org

Georgiana Ifrim
Insight Centre for Data Analytics
University College Dublin, Ireland
georgiana.ifrim@insight-centre.org

***Abstract*—Existing approaches to time series classification can be grouped into shape-based (numeric) and structure-based (symbolic). Shape-based techniques use the raw numeric time series with Euclidean or Dynamic Time Warping distance and a 1-Nearest Neighbor classifier. They are accurate, but computationally intensive. Structure-based methods discretize the raw data into symbolic representations, then extract features for classifiers. Recent symbolic methods have outperformed numeric ones regarding both accuracy and efficiency. Most approaches employ a bag-of-symbolic-words representation, but typically the word-length is fixed across all time series, an issue identified as a major weakness in the literature. Also, there are no prior attempts to use efficient sequence learning techniques to go beyond single words, to features based on variable-length sequences of words or symbols. We study an efficient linear classification approach, SEQL, originally designed for classification of symbolic sequences. SEQL learns discriminative subsequences from training data by exploiting the all-subsequence space using greedy gradient descent. We explore different discretization approaches, from none at all to increasing smoothing of the original data, and study the effect of these transformations on the accuracy of SEQL classifiers. We propose two adaptations of SEQL for time series data, SAX-VSEQL, can deal with X-axis offsets by learning variable-length symbolic words, and SAX-VFSEQL, can deal with X-axis and Y-axis offsets, by learning fuzzy variable-length symbolic words. Our models are linear classifiers in rich feature spaces. Their predictions are based on the most discriminative subsequences learned during training, and can be investigated for interpreting the classification decision.**

I. Introduction

Time series classification is a pervasive research problem, with critical applications ranging from seizure detection for human epilepsy [1] to classifying insect sounds for detecting female mosquitoes [2].

There are many time series classification datasets, with the UCR benchmark [3] being the most popular for comparing methods. This benchmark has also attracted criticism for skewing the research community into focusing on accuracy, at the expense of computational efficiency [4]. Most time series classification approaches fall into two groups: shape-based and structure-based [5]. Shape-based methods use the raw numeric time series and employ a 1-Nearest Neighbor (1NN) classifier with a distance metric. The Euclidean distance (ED) and the Dynamic Time Warping (DTW) distance are the most popular metrics [6]. These approaches are accurate, but expensive, as they have quadratic time complexity in the time series length [4]. A lot of work has aimed to address this bottleneck, most recently by proposing centroid-based approaches, such as Dynamic Time Warping averaging [2]. This approach achieves accuracy comparable to 1NN-DTW with less computational cost, as only one centroid per class is needed during the testing stage, but has an expensive training stage. Other work aims to learn discriminative subsequences from raw numeric data (aka shapelets) [7] using an optimization formulation that involves searching for fixed-length time series subsequences that best predict the target variable. This approach has promising accuracy, but has a significant training time complexity of $O(N^2n^3)$, where N is the number of time series and n is the length of time series [4].

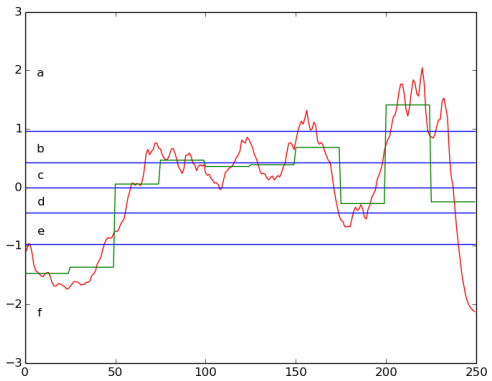

Fig. 1. An example transformation from numeric time series to symbolic sequence **ffcbccbdad**, by slicing the X-axis and the Y-axis.

Structure-based methods transform the raw numeric data into discrete representations, the most popular among them being the Symbolic Aggregate Approximation (SAX) [8]. The key idea behind many symbolic representations is to smoothen and compress the numeric data by first slicing the X-axis, computing averages of the slices, then slicing the Y-axis to map those averages to discrete symbols. Figure 1 shows an example of converting a numeric time series to a symbolic sequence, by slicing and mapping values on the X-axis and Y-axis.

The work of [9] proposes a variant of SAX approximation to produce distinct SAX-words with tf.idf weights (aka SAX-VSM) and builds a tf.idf centroid prototype for each class. This centroid-based approach was shown to produce results that are more accurate than the state-of-the-art, with an expensive

training stage and a fast classification stage. An advantage of this approach is that it allows interpreting the classification decision, as it ranks the best SAX-words per class based on their tf.idf weight. A weakness of this method is that the SAX-word length is fixed across all time series, which may reduce the power of this representation. Another drawback of this approach is that it requires an intensive parameter optimization for selecting the best SAX transformation during training. Another recent approach, 1NN BOSS VS [4], employs a Symbolic Fourier Approximation (SFA) instead of SAX, and uses a vector space of SFA-words and class centroids for classification. This approach is shown to be comparable in accuracy with SAX-VSM, in particular for large-scale datasets, and it is faster as it has less parameters to optimize for the SFA representation. Nevertheless, it is harder to interpret and use the SFA-words compared to SAX-words, since the symbolic characters at each position have different meanings and cannot be directly compared.

Our contribution. We propose a new approach for structure-based time series classification based on an efficient sequence classifier, SEQL, designed to work on very long discrete sequences with large alphabets [10], [11]. SEQL learns the best discriminative subsequences[1] from training data by exploiting the all-subsequence space using greedy gradient descent. We explore different discretization approaches, from none at all (raw data) to increasing smoothing and compression of the original data, and study the effect of these transformations on the accuracy and efficiency of training and testing of SEQL classifiers. We propose two adaptations of SEQL for time series data, under a SAX representation:

- **SAX-VSEQL**, can deal with X-axis offsets by learning variable-length symbolic words, therefore addressing a major weakness in prior work related to fixing the SAX-word length across all time series [8], [9].
- **SAX-VFSEQL**, can deal with both X-axis and Y-axis offsets by learning fuzzy variable-length symbolic words, and removes the need for tuning SAX parameters, a task that is computationally expensive and was identified as an area in need of improvement by prior work [4], [9].

As shown in our experiments with UCR benchmark datasets, our approach removes the need for extensive tuning of SAX parameters, which results in fast training and test stages, while preserving high accuracy. Furthermore, in light of new legislation and research priorities that emphasize explainable AI[2][3], we aim to deliver interpretable classifiers. We discuss the interpretability of our proposed classifier and compare our findings to prior work.

II. Related Work

The empirical work published by Wang et al in 2013 [6] has compared 8 different time series representations and 9 similarity measures across 38 time series datasets. Among the main conclusions of that study was that using the raw time series and Euclidean distance as a similarity metric for a 1NN classifier is very effective for time series classification. Another conclusion was that a 1NN classifier with Dynamic Time Warping distance delivers more accurate results than the Euclidean distance on small datasets and is superior or comparable to many other similarity measures, such as the edit distance. As both 1NN-Euclidean and 1NN-DTW approaches are computationally intensive, many approaches were proposed to deal with the efficiency aspect. Among them are DTW averaging [2] and early abandoning techniques that use lower bounds for early pruning of nearest neighbor candidates [12]. Other approaches aim to further increase the accuracy of 1NN-DTW by ensembling many such classifiers [13], [14], at the expense of computational cost. The computational burden of many of these techniques remains very high, as summarized in recent work [4] which details the training and testing complexity for these methods.

Shapelet-based classifiers search for subsequences of raw numeric time series that best discriminate the time series in different classes [14], [15]. The shapelets are typically first extracted from the data, then used as a new representation in which to learn standard classifiers such as decision trees, random forest and SVM [16]. Approaches for finding the best shapelets vary from brute force search, to bounding quality metrics such as the information gain [15], searching in a lower dimensional SAX space [17], or learning discriminative fixed-length shapelets [7]. These methods deliver high accuracy but have a high computational complexity for training or testing and many are sensitive to noise [4].

Approaches that work on discretized time series data have also been very popular, culminating with the SAX representation that has been used in many applications [8], [17]–[19]. The most accurate approaches involving the SAX representation transform the original numeric time series to a space of SAX-words and then use this representation for training classifiers. SAX-VSM presented in [9] is a centroid based classification approach where tf.idf weights of SAX-words are used to compute a centroid per class. This work also discusses an approach to search for the best parameters for the SAX representation via an optimization algorithm, but its training stage remains computationally expensive, mostly due to the need to search through a large set of SAX parameters.

The recent work of [4], [20] introduces a new symbolic approximation, the Symbolic Fourier Approximation (SFA), based on Discrete Fourier Coefficients. Both techniques use a vector space of SFA-words, but the first method (named BOSS) uses ensembles of histograms of SFA-words and 1NN classifiers, while the latter (named 1NN BOSS VS) uses tf.idf class centroids for classification. The methods are shown to be accurate and fast on many time series datasets [4]. We argue that this comes at the cost of interpretability, as the SFA-words used to build symbolic classifiers cannot be easily interpreted and compared.

In this work we study a sequence classification method, the

[1]In SEQL there is no need to manually provide the subsequence length, the data and learning objective drive the length selection.

[2]http://www.darpa.mil/program/explainable-artificial-intelligence

[3]European Union regulations on algorithmic decision-making and a "right to explanation": https://arxiv.org/pdf/1606.08813v3.pdf

Sequence Learner (SEQL), introduced in [10], [11]. Due to its greedy optimization approach, SEQL can quickly capture the distinct patterns of sequence data in very high-dimensional spaces. In prior studies SEQL was shown to achieve similar accuracy to the state-of-the-art, with improved scalability and interpretability. Our aim in this work is to understand how different transformations of time series influence this discriminative learning classifier. SEQL is of particular interest since it can learn variable-length subsequences as induced by the training data, rather than as selected by a user via parameter tuning. We show how to adapt SEQL to time series data to address weaknesses identified by prior work and compare the empirical behavior of our proposed methods to the state-of-the-art, on the UCR benchmark.

III. Symbolic Representation of Time Series

A time series is a series of measurements collected over a period of time. Usually the interval between any two consecutive measurements is a constant. A time series can be denoted as a vector V:

$$V = (v_1, v_2, \ldots, v_L) \tag{1}$$

L denotes the length of the time series and is also the number of dimensions of vector V. Here we discuss several methods to transform numeric time series data to discrete representations and the parameters associated with each transformation. We also discuss the combination of discretisation and learning of discriminative time series classifiers using SEQL, and name each method to describe the combo of discretisation and sequence learner.

A. No Approximation

The first representation we investigate uses the raw numeric data. For this representation there are no parameters. We call this approach Raw-SEQL, to denote the combo of raw time series and the SEQL classifier. The Raw-SEQL approach learns the most discriminative subsequences in the raw numeric data by treating each distinct numeric value as a discrete symbol. This is a fairly naive approach since it is unlikely to find repeated identical numeric subsequences across many time series. Nevertheless, we are interested to see how SEQL can cope with such a huge alphabet and feature space.

B. Slicing the X-axis

There are various ways to slice or group the values on the X-axis, in order to smoothen the original time series. One popular and simple technique is the Piecewise Aggregate Approximation (PAA) [6]. The main idea in PAA is to approximate each group of values on the X-axis by their average, therefore smoothing the original numeric data. For example we can approximate the value of each 3 time series points by their average, e.g., the time series subsequence 0.4, 0.5, 0.6 is replaced by the average 0.5. The main parameter of this approach is w, the final length of the PAA transformed time series. Figure 2 shows an example of how this method works. We call this approach DiscX(w)-SEQL.

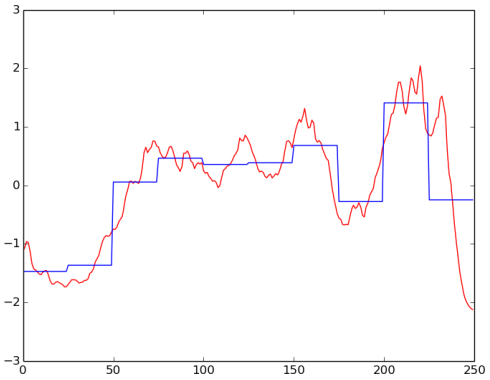

Fig. 2. X-axis slicing: Original time series versus PAA segments.

C. Slicing the Y-axis

This approach also aims to smoothen the raw numeric data, but this time by grouping values on the Y-axis, such that for example 0.5 and 0.55 are considered the same discrete token. Two popular options for slicing the Y-axis are equal width bins or equal probability areas. In both cases, the parameter is the number of Y-axis bins that we also call the alphabet size a. We investigate the binning of values on the Y-axis by using equal probability area as popularised in SAX [8]. Figure 3 shows an example of how this method works. We name this method DiscY(a)-SEQL.

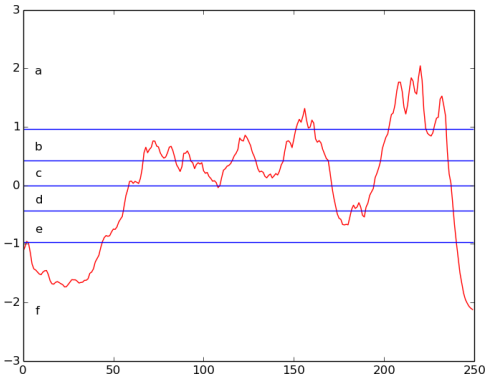

Fig. 3. Y-axis slicing: Equal probability slicing with alphabet size $a = 6$.

D. Slicing both the X and the Y-axis

For this approach we first slice the X-axis using the PAA approach, which results in one average value for each PAA segment. Then, we slice the Y-axis using the equal probability area approach, so that segments with averages falling in the same area are mapped to the same symbol or token. The parameters for this approach are the number of X-axis bins, denoted by w and the number of Y-axis bins, denoted by a. A popular approach in this category is the Symbolic Aggregate Approximation (SAX) [8] which we detail below. Time series

data can be transformed to a SAX representation at character-level or word-level [8]. A character-level representation is a sequence of characters (e.g., **abcadacbadbcbabdbcba**) while a word-level representation is a sequence of words (e.g., **aba dac aac dac daa**). In the latter case, the words are made of characters from a given alphabet and have equal length. The character-level representation is simply a special case of the word-level representation where the length of words is 1. However, it is more convenient to distinguish these two representations in our work.

1) SAX-chars(w,a): The SAX representation at character-level has two parameters: the number of X-bins w and the number of Y-bins a. It results in a sequence of characters of length w, where typically w is much smaller than L, the original length of the time series. In prior work w is chosen to range from 64 to 256, while the length of the time series L can be in the thousands. The procedure for obtaining the SAX-chars representation is as follows:

- X-slicing: Divide a time series into w segments with equal length.
- Y-slicing: Encode each segment with a character from the alphabet (where the alphabet has cardinality a).

Figure 4 shows an example of how the SAX-chars method works. We denote this approximation as SAX-chars(w,a).

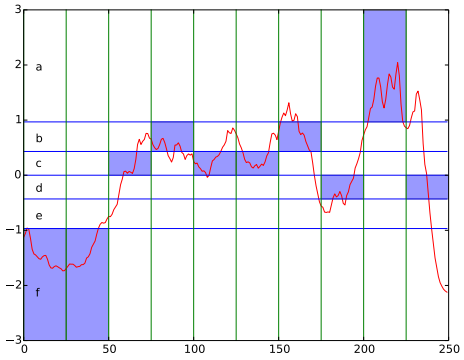

Fig. 4. X and Y-axis slicing: An example of time series transformation to SAX-chars($w = 10, a = 6$). The final symbolic sequence is **ffcbccbdad**.

2) SAX-words(l,w,a): In practice, SAX is often combined with a sliding window length l, which should be shorter than the time series length L. The procedure is described as follows:

- Set the window at the start position of the time series.
- Transform the subsequence time series within the window with the SAX-chars method. The result is a sequence of length w.
- Move the window one step forward and repeat the process until the window reaches the end of the original time series.

Figure 5 shows an example for using SAX-chars with a sliding window to obtain the SAX-words representation. We denote this approach as SAX-words(l,w,a).

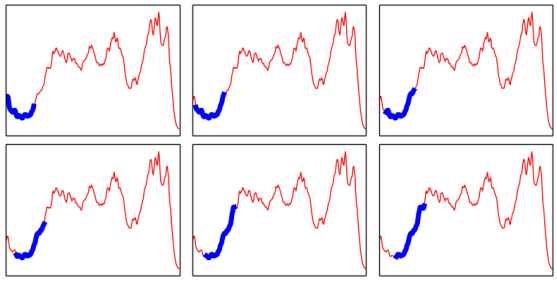

Fig. 5. Sliding window shifting results in SAX-words($l = 64, w = 10, a = 6$) representation.

The final result is a sequence of equal-length SAX-words. The SAX word is the smallest unit (i.e., unigram token) of the sequence. From our observations, this representation is vulnerable to the *false dismissal* problem which was discussed in [17]. In short, an improper choice of Y-axis cuts may convert two close numerical values to two different symbols, thus introducing misleading information during the transformation.

Sliding windows are a common technique when working with time series. In the case of SAX transformation, it helps reduce the unwanted effects caused by misalignment between time series. The technique was proven to be effective in [9], [17], [19]. However, the robustness of SAX-based methods strongly depends on the three parameters, l (window size), w (word length) and a (alphabet size), which require computationally expensive optimization.

E. Implementation practices

Normalization: The work in [8] advocates that time series data should be be z-normalized before applying further transformations, as otherwise we compare time series with different offsets and amplitudes. The SAX encoding process also assumes that time series data follow a standard normal distribution $\mathcal{N}(0, 1)$. Before applying any other transformations, raw time series are z-normalized with the following formula:

$$z\_normalize(V) = \frac{V - mean(V)}{std(V)} \tag{2}$$

in which V is the numeric vector of time series values, $mean(V)$ is the mean value of V and $std(V)$ is the standard deviation value of V.

Indivisible-length problem: As it was mentioned above, the X-axis segmentation step divides the time series into segments of equal length. However, it is quite common that the time series length L (or sliding window length l) is not divisible by the number of segments w. For example, how should we divide a time series of length 20 into 3 segments? In this case, the original time series data requires modification before segmentation. Divisible-length time series can be obtained by padding the data points of original data w times. If $V = v_1, v_2, \ldots, v_L$ is the original time series, then

$$\bar{V} = (v_1^1, v_1^2, \ldots, v_1^w, v_2^1, \ldots, v_2^w, \ldots, v_L^1, \ldots, v_L^w) \tag{3}$$

is the modified time series where $v_j^i = v_j^k = v_j \ \forall \ i, j, k$. The resulting vector $\bar{V}$ of length $L \times w$ is divisible into w segments. This is now standard practice and is implemented in recent SAX software [8].

Numerosity reduction: In practice, if the word obtained in the current window is identical to the word obtained in the previous window, it will be ignored in the output time series. This practice is to avoid the redundancy of information in the compressed data and does not affect the classification accuracy [8], [9]. The notation used for describing the time series and various parameters of the symbolic representations described above are summarized in Table I.

TABLE I
NOTATION FOR TIME SERIES CLASSIFICATION FRAMEWORK

Symbols	Description
V	Raw (normalized) numeric time series
L	Length of original time series
w	Number of X-axis bins
a	Number of Y-axis bins
l	Size of sliding window

IV. CLASSIFICATION WITH SEQUENCE LEARNER

In this section we describe the theoretical framework for employing the SEQL symbolic sequence classifier, and propose two adaptations for time series classification.

A. Sequence Learner

SEQL learns discriminative subsequences from training data by exploiting the all-subsequence space using a coordinate gradient descent approach [10], [11]. The key idea is to exploit the structure of the subsequence space in order to efficiently optimize a classification loss function, such as the binomial log-likelihood loss of Logistic Regression or squared hinge loss of Support Vector Machines. This approach was originally designed for classification of sequences of discrete items, such as text or DNA. An important aspect of this approach is that it can efficiently select the best variable-length subsequences as driven by the training data and loss function (by combining learning and feature selection), and does not require a user to provide candidate subsequence lengths, such as in [7]. SEQL was shown to perform well in dense feature spaces and with very long sequences and large vocabularies. We are therefore interested to study how this method performs with different discretizations of time series data.

In the training stage, SEQL takes as input discretized time series with corresponding labels and produces a linear classifier (i.e., a list of weighted subsequences) for each class. In the classification stage, the classifier is applied to new (discretized) test time series to produce classification scores. Each test time series is assigned to the class with the maximum classification score. SEQL is designed for binary classification and is typically employed for multi-class problems via the one-vs-all approach [10].

In this section we briefly provide some of the theoretical background and intuition for SEQL. This is necessary as we later show how to adapt SEQL to better exploit the characteristics of the discrete time series representation we work with.

Let Σ be the alphabet of possible symbols, e.g., in the case of DNA sequences, $\Sigma = \{A, C, G, T\}$. Let $D = \{(s_1, y_1), (s_2, y_2), \ldots, (s_N, y_N)\}$ be a training set of instance-label pairs, where $s_i = c_1 c_2 \ldots c_m$ is a sequence of length m with each $c_i \in \Sigma$. Each sequence s_i has an associated score $y_i \in \{-1, +1\}$. We represent a sequence s_i as a binary vector in the space of all subsequences in the training data: $x_i = (x_{i1}, \ldots, x_{ij}, \ldots, x_{id})^T, x_{ij} \in \{0, 1\}, i = 1, \ldots, N$, where $x_{ij} = 1$ means that subsequence s_j occurs in sequence s_i. We denote by d the number of distinct subsequences in the feature space, i.e., the coordinates of the vectors space in which we learn. This is a huge feature space, but it is never fully explicitly represented, only parts of the feature space are explicitly generated, as driven by the learning algorithm. The notation used in this section is summarized in Table II:

TABLE II
NOTATION FOR SEQL FRAMEWORK

Symbols	Description
S	Set of sequences
D	Training set
s_i	Sequence i
y_i	Score of sequence i
x_i	Vector representation of sequence s_i
x_{ij}	Indicator of subsequence s_j for sequence s_i
N	Number of training examples
m	Sequence length
d	The number of distinct subsequences in the feature space
Σ	The alphabet set, e.g., $\{A, C, G, T\}$

The goal is to learn a mapping $f : S \rightarrow \{-1, +1\}$ from the given training set D so that $f(s)$ predicts a classification score $y \in \{-1, +1\}$ for a new unseen sample $s \in S$. In this setting, we learn a linear mapping from the vector representation x_i of sequence s_i to a class y_i, so we set $f(x_i) = \beta^T x_i$. The linear classifier is defined by the parameter vector $\beta = (\beta_1, \ldots, \beta_j, \ldots, \beta_d)$, which is computed by minimizing a loss function over the training set:

$$\beta^* = \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} L(\beta) \tag{4}$$

where

$$L(\beta) = \sum_{i=1}^{N} \xi(y_i, x_i, \beta) + C R_\alpha(\beta) \tag{5}$$

Most β_j will be zero when the optimization is concluded and only those coordinates selected during training as useful (i.e., discriminative), will be non-zero.

The notation in Equation 5, $\xi(y_i, x_i, \beta)$, denotes a classification loss function. SEQL implements two loss functions, the binomial log-likelihood loss (logistic regressions) (Equation 6) and the squared hinge loss (SVM) (Equation 7):

$$\xi(y_i, x_i, \beta) = log(1 + e^{-y_i \beta^T \cdot x_i}) \tag{6}$$

$$\xi(y_i, x_i, \beta) = max(1 - y_i \beta^T \cdot x_i, 0)^2 \quad (7)$$

In Equation 5, $C \in \mathbb{R}_0^+$ denotes the weight for the regularizer $R_\alpha(\beta)$. Larger C leads to larger penalty for β and can influence the sparsity of the learned model. SEQL implements the elastic-net regularization for $R_\alpha(\beta)$ [21], which combines $l1$ and $l2$ penalties.

The number of potential subsequences grows exponentially with the sequence length m. A result of this is that the dimension d of vectors x_i and β is potentially huge. The work in [10], [11] introduces a branch-and-bound strategy which simplifies the learning problem by using greedy coordinate descent with an efficient selection of the most discriminative subsequences. The main idea relies on bounding the gradient of any subsequence based on its prefix, so that large parts of the search (i.e., feature) space do not need to be explored. Next, we describe the high level SEQL workflow and our proposed adaptations to symbolic representations of time series.

Algorithm 1 shows the SEQL optimization process. The crucial part of this algorithm is the search for the subsequence with the largest gradient value, which is also the feature that best discriminates the classification objective, given the features already selected (line 4).

Algorithm 1 SEQL workflow

1: Set $\beta^{(0)} = 0$
2: **while** !termination condition **do**
3: Calculate objective function $L(\beta^{(t)})$
4: Find coordinate j_t with maximum gradient value
5: Find optimal step length η_t
6: Update $\boldsymbol{\beta}^{(t)} = \boldsymbol{\beta}^{(t-1)} - \eta_t \frac{\boldsymbol{\partial} L}{\boldsymbol{\partial} \beta_{j_t}} (\boldsymbol{\beta}^{(t-1)}) e_{j_t}$
7: Add feature at coordinate j_t to feature set
8: **end while**

Algorithm 2 shows the search procedure for efficiently finding the best feature in each iteration. The search starts at unigrams (single symbols) and only explicitly generates longer subsequences if the quality criterion does not prune those candidate subsequences out.

B. Adaptations of SEQL for Time Series Classification

We describe here two adaptations of SEQL for working with SAX representations. We chose this symbolic representation due to its popularity, high accuracy of classifiers trained with SAX and the quality that SAX-words are interpretable[4].

SAX-SEQL: SEQL with SAX representations. As described in Section III, the result of transforming a numeric time series using SAX can be one of two types: sequence of SAX-characters or sequence of SAX-words. SEQL directly supports input sequences with characters as unigrams (single tokens) or words as unigrams. Thus, we can directly use SAX to transform numeric sequences to character or word

[4]SAX symbols at different positions in the sequence are directly comparable to each other, and have a natural ordering which can be exploited during learning.

Algorithm 2 Search for best subsequence in SEQL

1: $\tau \leftarrow 0$
2: $best\_feature \leftarrow NIL$
3: **for all** $s' \in \bigcup_{i=1}^{N} \{s | s \in s_i, |s| = 1\}$ **do** ▷ For each unigram
4: GROW_SEQUENCE(s')
5: **end for**
6: **return** $best\_feature$
7:

1: **function** GROW_SEQUENCE(s)
2: **if** $\mu(s) \leq \tau$ **then return** ▷ $\mu(s)$ bound as in [10]
3: **else if** $abs(gradient(s)) > \tau$ **then**
4: $best\_feature = s$ ▷ Suboptimal solution
5: $\tau = abs(gradient(s))$
6: **end if**
7: **for all** $s'' \in \{s' | s' \supseteq s, s' \in \bigcup_{i=1}^{N} x_i, |s'| = |s| + 1\}$ **do**
8: GROW_SEQUENCE(s'')
9: **end for**
10: **end function**

based sequences, and feed this representation to SEQL. In this combo (called SAX-SEQL) we still need to optimize the SAX parameters to obtain the SAX representation for which SEQL provides the best accuracy. In our experiments section, we investigate the SAXchars-SEQL and SAXwords-SEQL approaches, with and without SAX parameter tuning. We are particularly interested to investigate the influence of representation choice (chars versus words) and subsequence length (going beyond unigrams) on the accuracy of SAX-SEQL.

SAX-VSEQL: Learning variable-length SAX-words. Most prior work promotes the use of SAX-words and the sliding window approach [8], [9], [18], [19]. Nevertheless, the SAX-word length is always fixed across all time series. This was identified as a weakness by prior research, since the best word-length may depend on each individual time series and the discriminative parts could be subunits of SAX-words. We address this weakness by a hybrid approach that takes a sequence of SAX-words as input, but treats it as a sequence of characters from the perspective of SEQL. We also adapt the SEQL implementation to search for char subsequences only within each SAX-word, therefore allowing us to learn subunits of SAX-words, which we call variable-length SAX-words. We think this is an important adaptation as it allows to deal with noise introduced during the transformation of numeric to symbolic data. We call this method SAX-VSEQL and show in our experiments that learning SAX-words subunits improves the accuracy of SAX-SEQL.

SAX-VFSEQL: Learning fuzzy variable-length SAX-words.

SEQL only supports exact matching of sequences, e.g. only identical sequences are counted as matched sequences. In SAX

representations the symbols have an actual ordering which can be exploited for learning fuzzy features. We believe this is helpful in order to deal with potential noise introduced during the Y-axis mapping of numeric to symbolic data, where for example numeric values that are close to each other would be mapped to different symbolic values, that are nevertheless semantically close to each other, i.e., in the SAX symbolic representation, *a* is semantically closer to *b*, than to *c*.

We modify the SEQL algorithm to work with a distance mapping between characters, in order to learn fuzzy subsequences. For example, we want SEQL to consider the subsequence *abb* similar to subsequence *bbb*, since the only different symbol in the two subsequences is at position one, and *a* and *b* are close to each other in the SAX mapping. We explain how we define the distance mapping between symbols and subsequences, and show that the SEQL theoretical framework still holds when introducing a distance function between symbols, to enable learning fuzzy subsequences.

Definition 1 (**Distance between two characters**). The distance between two characters of a given alphabet is defined by the difference of the character indexes in the alphabet.

$$d(c_1, c_2) = |index(c_1) - index(c_2)| \tag{8}$$

For example: given the alphabet of 4 characters $\{a, b, c, d\}$, then $d(a, b) = 1$ while $d(a, c) = 2$.

Definition 2 (**Distance between two sequences of same length**). Let $s_1 = a_1 a_2 \dots a_L$ and $s_2 = b_1 b_2 \dots b_L$ be the two symbolic sequences of equal length. The distance between s_1 and s_2 is

$$D(s_1, s_2) = \sum_{i=1}^{L} d(a_i, b_i) \tag{9}$$

Definition 3 (**Distance between two sequences of different length**). Let $s_1 = a_1 a_2 \dots a_L$ and $s_2 = b_1 b_2 \dots b_l$ be the two symbolic sequences with $l \leq L$ and $SS(s_1, l)$ be the set of all $s1$ subsequences of length l. The distance between s_1 and s_2 is

$$D(s_1, s_2) = \min_{s_{1i} \in SS(s_1, l)} D(s_{1i}, s_2) \tag{10}$$

Definition 4 (**Fuzzy matching of sequences**). Two sequences are said to match if the distance between them is less than a predefined threshold.

We can restrict the total distance between two sequences to domain $[0, 1]$, by dividing by an appropriate factor (e.g., the maximum distance between symbols multiplied with the sequence length). We can turn the distance function into a similarity function by defining $Sim(s_1, s_2) = 1 - D(s_1, s_2),\ Sim(s_1, s_2) \in [0, 1]$.

Anti-monotonicity property. The SEQL search approach relies on the anti-monotonicity property of the frequency of subsequences: the frequency of a subsequence is always equal or lower than the frequency of any of its subsequences. To simplify the argument, we focus on the prefix of a subsequence, rather than any of its subsequences. We show here that the anti-monotonicity property still holds if instead of exact match of features, we introduce a fuzzy match via a distance function between subsequences. The main intuitive reason why this works is that the matching distance increases with sequence length, and conversely, the matching similarity decreases with sequence length, so the same anti-monotonicity argument can be used for similarity as for frequencies. We prove the gradient bounding theorem for fuzzy SEQL learning below.

As in [10] we assume the following properties for the loss function:

1. ξ depends on $y_i,\ x_i$ and β only through the classification margin $m_i = y_i \beta^t x_i$. We write $\xi(y, x, \beta) = \xi(m)$.
2. ξ is a monotone decreasing function of the margin: $\xi'(m) \leq 0$.
3. ξ is convex and continuously differentiable.

The gradient of $L(\beta)$ with respect to a coordinate β_j is:

$$\frac{\partial L}{\partial \beta_j}(\beta) = \sum_{i=1}^{N} y_i x_{ij} \xi'(m_i) + CR'_\alpha(\beta_j) \tag{11}$$

Theorem 1 (**Bounding the search for the best coordinate with fuzzy matching**). *For any loss function ξ satisfying properties 1-3 and for any subsequence $s_p \subseteq s_j$, we can bound the gradient at coordinate j (corresponding to sequence s_j) , using only information about coordinate p (corresponding to the occurrence of the prefix s_p):*

$$\left| \frac{\partial L}{\partial \beta_j}(\beta) \right| \leq \mu(s_p) \tag{12}$$

where

$$\mu(s_p) = \max \left\{ \left| \sum_{\{i | x_{ip} \in (0,1], y_i = +1\}} x_{ip} \xi'(m_i) + CR'_\alpha(\beta_j) \right|, \left| \sum_{\{i | x_{ip} \in (0,1], y_i = -1\}} -x_{ip} \xi'(m_i) + CR'_\alpha(\beta_j) \right| \right\} \tag{13}$$

and $x_{ij} \in [0, 1]$ denotes fuzzy matching of feature s_j (i.e., features that are close in similarity to s_j, can contribute to the gradient computation).

Proof. To prove the theorem we split the gradient computation between the terms computed over positive examples and negative examples, following similar arguments as in [10]. The main difference is that we now use a soft matching of features, where instead of $x_{ij} \in \{0, 1\}$ to denote an exact

match of feature s_j, we have $x_{ij} \in [0,1]$, to denote fuzzy matching of feature s_j.

$$\begin{aligned}\frac{\partial L}{\partial \beta_j}(\beta) &= \sum_{i=1}^{N} y_i x_{ij} \xi'(m_i) + CR'_\alpha(\beta_j) \qquad (14)\\ &= \sum_{\{i|x_{ij}\in(0,1]\}} y_i x_{ij} \xi'(m_i) + CR'_\alpha(\beta_j)\\ &\leq \sum_{\{i|,x_{ij}\in(0,1],y_i=-1\}} y_i x_{ij} \xi'(m_i) + CR'_\alpha(\beta_j)\\ &\leq \sum_{\{i|x_{ip}\in(0,1],y_i=-1\}} y_i x_{ip} \xi'(m_i) + CR'_\alpha(\beta_j)\\ &= \sum_{\{i|x_{ip}\in(0,1],y_i=-1\}} -x_{ip} \xi'(m_i) + CR'_\alpha(\beta_j)\end{aligned}$$

The last inequality holds due to the anti-monotonicity property of subsequence occurrence and similarity. Every sequence which contains s_j also contains its subsequence s_p, thus: $x_{ij} \leq x_{ip}, i = 1, \ldots, N$.

Similarly, we compute a bound for the positive examples:

$$\frac{\partial L}{\partial \beta_j}(\beta) \geq \sum_{\{i|x_{ip}\in(0,1],y_i=+1\}} x_{ip} \xi'(m_i) + CR'_\alpha(\beta_j) \qquad (15)$$

The two bounds on each class can be combined to get an upper bound for the gradient at coordinate j, using only information about coordinate p (for dealing with the penalty term, see details in [10]):

$$\left|\frac{\partial L}{\partial \beta_j}(\beta)\right| \leq \max\left\{\left|\sum_{\{i|x_{ip}\in(0,1],y_i=+1\}} x_{ip} \xi'(m_i) + CR'_\alpha(\beta_j)\right|, \left|\sum_{\{i|x_{ip}\in(0,1],y_i=-1\}} -x_{ip} \xi'(m_i) + CR'_\alpha(\beta_j)\right|\right\} \qquad (16)$$

□

The bound allows us to efficiently search for the coordinate with the largest gradient, without having to expand the full feature space and allowing fuzzy features to contribute to the gradient computation.

Bound Quality. The upper bound given in Theorem 1 is tight, as the inequality in Equation 12 becomes equality whenever the set of occurrences of the prefix is identical to that of the longer sequence.

Algorithm Complexity. We discuss the total time complexity of each component in the SAX-VFSEQL algorithm. The SAX transformation with fixed parameters has $O(NL)$ complexity, where N is the number of time series and L is the time series length. The training stage for VFSEQL has a complexity of $O(Nf)$, where N is the number of sequences (time series) and f is the number of features that need to be explicitly generated for gradient computation. Thus the training stage of the SAX-VFSEQL combo has a total time complexity of $O(NL) + O(Nf)$. The test stage of SAX-VFSEQL has a complexity of $O(L)$ as it only requires a linear scan of the time series.

Covergence Speed. The work of [22] presents a theoretical analysis of the convergence speed of learning algorithms that use greedy coordinate descent with the Gauss-Southwell rule, for optimizing convex, continuous classification losses (as implemented in VFSEQL). They show that this type of algorithms have very fast convergence and are recommended to alternative optimization strategies such as full gradient descent or other coordinate descent approaches.

V. Experimental Results

A. Experiment Setup

We conduct our experiments on the well-known UCR archive [3] which is the main benchmark for most time series classification studies. Detailed information on sizes of data sets and length of time series can be found on the UCR website. We also use the classification results listed there for 1NN-Euclidean and 1NN-DTW, as baselines for the comparison with our approaches. For studying the different variants of our proposed methods we only use the binary datasets listed in Table III. Once we select the best variant of our approach, we compare to the state-of-the-art on 41 UCR datasets (which also include the previous binary datasets) for which SAX-VSM has published results [9].

Our implementation of SAX-VSEQL and SAX-VFSEQL is written in C++. All methods are run on a Linux PC with Intel Core i7-4790 Processor (Quad Core HT, 3.60GHz), 16GB 1600 MHz memory and 256 Gb SSD storage.

TABLE III
Binary datasets.

Dataset	Train	Pos/Neg	Test	Pos/Neg	Length
Coffee	28	14/14	28	15/13	286
Earthquakes	139	104/35	322	264/58	512
ECG200	100	31/69	100	36/64	96
ECGFiveDays	23	14/9	861	428/433	136
FordA	1320	681/639	3601	1846/1755	500
FordB	810	401/409	3636	1860/1776	500
Gun_Point	50	24/26	150	76/74	150
Lighting2	60	20/40	61	28/33	637
MoteStrain	20	10/10	1252	675/577	84
Passgraph	69	36/33	131	64/67	364
SonyAIBORS	20	6/14	601	343/258	70
SonyAIBORSII	27	11/16	953	365/588	65
TwoLeadECG	23	12/11	1139	569/570	82
wafer	1000	97/903	6164	665/5499	152
yoga	300	137/163	3000	1393/1607	426

B. Parameters for SAX and SEQL

The SAX parameters include the window length l, word length w and alphabet size a. The work in [9] introducing SAX-VSM is the only one which invests substantial effort to optimize these parameters. Most prior work disregards this problem by fixing the parameters or only considering a small subset of the parameter space. We think that because SAX-VSM fails to address the *false dismissal* problem, it needs to

fully optimize the parameters to mitigate the negative effect on accuracy. Unfortunately, this leads to very high training cost for SAX-VSM, as also criticized in [4]. For all follow-up experiments we set all SAX parameters of our SAX-SEQL methods to fixed values, specifically $a = 4, w = 16$ and $l = 0.2 * L$ (where L is length of the time series). For SEQL parameters, we use the default values set in the open source code[5]. For SAX-VFSEQL we set the distance threshold to 1 (i.e., we allow only one character difference).

C. Accuracy

1) Without sliding window: In our first set of experiments, we study different symbolic representations in combination with SEQL. We start from raw data (i.e., no transformation) to SAX characters transformation (Table IV). The tested representations are as follows:

- **Raw-SEQL** The extreme version of our approach which takes the numeric data as symbolic input, i.e., each number is a symbol.
- **DiscX(w)-SEQL** Equal-width areas X-axis slicing only. Sequence length w is the only parameter.
- **DiscY(a)-SEQL** Equal-probability areas Y-axis slicing only. Alphabet size a is the only parameter.
- **SAXchars(w,a)-SEQL** SAX character sequences, i.e., both X and Y-slicing is applied. Parameters include sequence length w and alphabet size a.

TABLE IV
CLASSIFICATION ERROR RATES WHEN COMBINING SYMBOLIC REPRESENTATIONS WITH SEQL.

Dataset	Raw	DiscX(w)	DiscY(a)	SAXchars(w,a)
Coffee	0.4286	0.4643	**0.1071**	0.2143
Earthquakes	0.2019	**0.1801**	0.2453	0.2638
ECG200	0.63	0.64	**0.23**	0.25
ECGFiveDays	0.5006	0.5029	**0.3136**	0.338
FordA	0.4788	0.4874	**0.2285**	0.3602
FordB	0.4981	0.4884	**0.2162**	0.4686
Gun_Point	0.4667	0.4933	**0.06**	0.1333
Lighting2	0.4754	0.541	**0.3279**	0.3443
MoteStrain	0.4609	0.4609	**0.1581**	0.1845
Passgraph	0.5115	0.5115	**0.2366**	0.4122
SonyAIBO	0.4309	0.4293	**0.3694**	0.3794
SonyAIBOII	0.6149	0.617	**0.2875**	0.3221
TwoLeadECG	0.4996	0.5004	**0.1062**	0.2564
wafer	0.6723	0.5357	**0.016**	0.0386
yoga	0.4837	0.3123	**0.299**	0.3293

The results (Table IV) show that directly taking numbers as symbols is not generally a good idea. The accuracy is surprisingly good for the Earthquakes dataset, but not for the remaining ones. On the other hand, transforming numeric values to symbols seems to provide some information about the structure of time series that is useful for the classification task. However, the X-slicing seems to be troublesome, as we can observe that the accuracy decreases when applying the full SAX transformation (SAXchars) compared with Y-slicing only (DiscY).

[5]https://github.com/heerme/seql-sequence-learner

2) With sliding window: As discussed before, SAX presents a number of issues when combined with sequence learning techniques. SAX without sliding windows is particularly sensitive to both X and Y-slicing choices. In our case, we address these issues using sliding windows (SAX-SEQL), subsequences-of-subsequences learning (SAX-VSEQL) and fuzzy matching (SAX-VFSEQL). Our next experiment shows that these techniques have a positive impact on the classification results. Table V shows that using a sliding window for the SAX representation (SAXwords and SEQL, denoted as SAX-SEQL) delivers better results than using the SAXchars approach. It also shows that learning subunits of SAX-words (done in SAX-VSEQL) and fuzzy matches (SAX-VFSEQL) improves the results compared to SAX-SEQL, where the SAX-word length is fixed across all time series.

TABLE V
CLASSIFICATION ERROR RATES FOR VARIANTS OF SAX-SEQL.

Dataset	SAX-SEQL	SAX-VSEQL	SAX-VFSEQL
Coffee	0.0714	**0.0**	0.0357
Earthquakes	0.2081	0.2112	**0.205**
ECG200	0.25	0.26	**0.18**
ECGFiveDays	0.0743	0.0244	**0.0174**
FordA	0.4685	0.1486	**0.1347**
FordB	0.5069	0.2175	**0.2134**
Gun_Point	0.04	**0.0133**	0.02
Lighting2	0.377	**0.2951**	**0.2951**
MoteStrain	0.1845	**0.1597**	0.1661
Passgraph	0.3511	**0.2595**	0.3128
SonyAIBO	**0.2363**	0.3694	0.3727
SonyAIBOII	0.3284	0.1123	**0.0965**
TwoLeadECG	**0.021**	0.0623	0.0527
wafer	0.0081	**0.0071**	0.0079
yoga	**0.1977**	0.2313	0.2033

D. Comparison to the state-of-the-art

In the next set of experiments which also include multi-class datasets, we compare our best approach, SAX-VFSEQL, with state-of-the-art techniques including, 1NN-ED, 1NN-DTW [3], FastShapelets [17], SAX-VSM [9] and BOSS VS [4]. The results for those methods are either taken directly from publications or websites of the authors, or are reproduced with code available online. We also include the most recent results published in the study of Bagnall et al 2016 [14] (that aims to reproduce all those methods), as they report conflicting results for some of the methods. Figure 6 represents visually the errors of SAX-VFSEQL paired with a state-of-the-art method. Each point represents a dataset. Points below the line mean that SAX-VFSEQL is more accurate than the other method in the pair. Note that all parameters for SAX-VFSEQL are fixed (as detailed in Section V-B), while the state-of-the-art methods use fully optimized parameters, unless indicated otherwise in their name. We observe that SAX-VFSEQL clearly outperforms SAX-VSM when parameters are fixed. When SAX-VSM uses optimized parameters, the accuracy is comparable to that of our method, but as we show in the next section, SAX-VSM requires high running time for tuning parameters. We provide the code of our

method as well as accuracy and running time results for all methods and datasets at https://github.com/thach/saxseql. Among competing methods BOSS VS is most notable with good accuracy and running time. Nevertheless, it is hard to interpret the classification decisions of this method.

TABLE VI
STATE-OF-THE-ART CLASSIFIERS

Symbols	Description
fixedSAX-VSM	SAX-VSM with fixed parameters $(a = 4, w = 16, l = 0.2 * L)$.
optSAX-VSM	SAX-VSM with optimized parameters. Results taken from the author's GitHub.
SAX-VSM*	Results reproduced by [14].
BOSS VS	Results reproduced with the author's implementation [4].
FastShapelets	SAX-based method, results from [14].
1NN-Euclidean	Results taken from the UCR Archive.
1NN-DTW	Results taken from the UCR Archive.

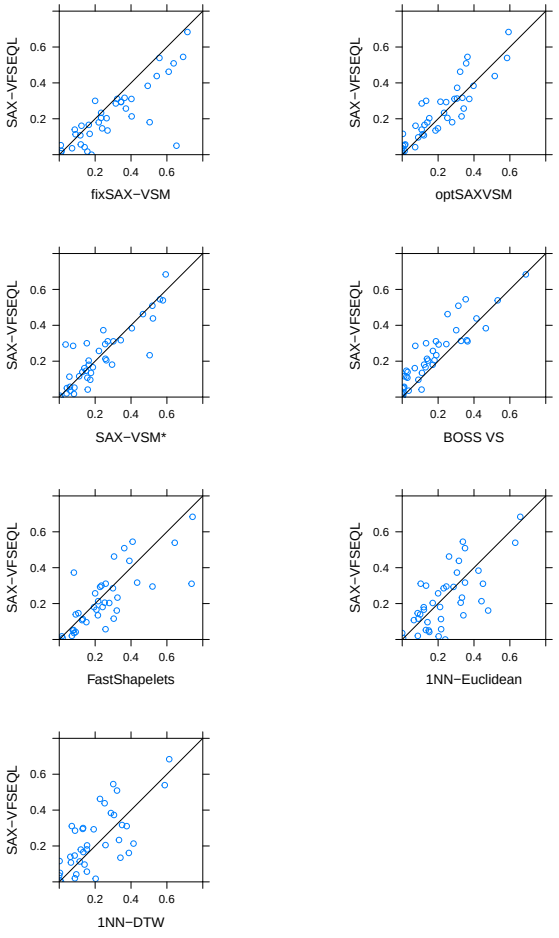

Fig. 6. SAX-VFSEQL accuracy against state-of-the-art classifiers.

E. Running time and scalability

In our study, we address the issue of parameter dependence of SAX-based techniques, by proposing new SAX-SEQL methods. Our aim is to reduce the running time by avoiding expensive training of SAX parameters, but still preserve a high accuracy. In Figure 6, we already showed that SAX-VSM heavily depends on parameter optimization since its accuracy drops substantially when tested with the same fixed set of configurations as SAX-VFSEQL. On the other hand, the high price of optimizing parameters for SAX-VSM can be seen in Table VII which shows the increase in running time when training data grows on the synthetic dataset CBF.

TABLE VII
SCALABILITY SAX-VFSEQL VS SAX-VSM ON CBF DATA. RUNNING TIME (SEC) AS A FUNCTION OF TRAINING SET SIZE.

Size of training data	SAX-VFSEQL	SAX-VSM
60	10.97	3.4
120	22.19	13.65
240	47.22	55.57
480	96.56	265.14
960	176.65	1245.79
1920	303.96	5569.41

To further test the scalability of SAX-VFSEQL, we again use CBF, and increase the size of training data as well as the length of the time series. In line with the algorithm time complexity, the running time increases linearly with the data size.

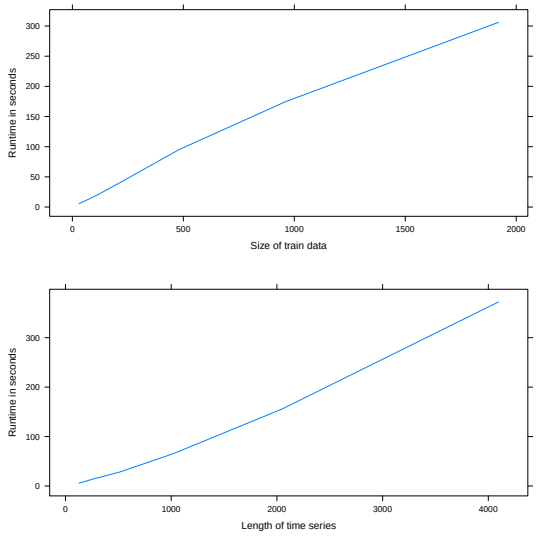

Fig. 7. Scalability with training size and length of time series. Running time (sec) on CBF synthetic dataset.

VI. INTERPRETABILITY

Our proposed SAX-SEQL based methods result in interpretable classifiers. The result of the training stage is a linear classifier per class: a list of the most discriminative subsequences identified for each class, sorted by the weight optimized during training. We argue that these discriminative features are potentially more useful than the SAX-word features ranked based on tf.idf weights as in SAX-VSM [9]. There is a lot of evidence [23]–[26] that discriminative classifiers

(e.g., SVM, Logistic Regression, RandomForests, NeuralNets) are more accurate than generative ones. Generative methods only learn from positive examples, while discriminative methods exploit both the negative and the positive examples during learning. The SAX-VSM method is a generative classification method as it builds characteristic tf.idf centroid vectors for each class. There can be cases where the top tf.idf patterns are the same for two or more classes, in which case generative approaches, such as SAX-VSM, would fail to capture the difference between the classes.

We note that since the SEQL output is in SAX symbolic representation, it is necessary to map the patterns to their positions in the original numeric data. This is a simple step that matches the features to each window in the SAX transformation. To discuss the interpretability of our methods, we analyze the classifiers for two UCR datasets also discussed in [9]: Coffee and Gun/Point. The Coffee dataset consists of time series from two classes: Arabica and Robusta. Table VIII illustrates the top-2 positive and negative features selected by the classifier. Figure 8 illustrates the best and second best patterns found for

TABLE VIII
FEATURES SELECTED BY SAX-VFSEQL FOR THE COFFEE DATASET. ARABICA IS THE POSITIVE CLASS, AND ROBUSTA IS THE NEGATIVE CLASS.

Weight	Feature
Class: Arabica	
0.02891	bcddbdd
0.02696	bcdccdd
Class: Robusta	
-0.02891	baaaacdcddc
-0.03179	baabacdddd

the Coffee dataset, for each class. Although the time series selected from each class look quite similar at first glance, the classifier captures the fine grained differences via the selected features than can be mapped back to the original time series to explain the decision taken by the classifier. The two best patterns selected by SAX-VSM are shown in Table IX. The patterns selected by the two methods seem to be quite different, although when mapped to the original numeric time series, they cover a similar region. The classification decision in SAX-VFSEQL is driven directly by the linear combination of feature weights. The Gun/Point dataset has time series

TABLE IX
FEATURES SELECTED BY SAX-VSM FOR THE COFFEE DATASET.

Weight	Feature
Class: Arabica	
0.08254	cdddddccccbaaaaa
0.07572	ccdddddccccbaaaa
Class: Robusta	
0.08436	ddddddcddbaaaaaa
0.08264	abbcddcddccdcaaa

from two classes: Gun and Point. Table X illustrates the top-2 positive and negative features selected by the classifier, and

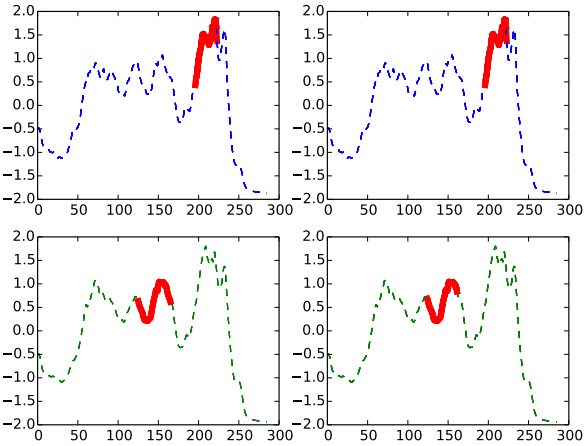

Fig. 8. Best two patterns for each class in the Coffee dataset. Top is an example from Arabica and bottom, an example from Robusta.

TABLE X
FEATURES SELECTED BY SAX-VFSEQL FOR THE GUN/POINT DATASET. GUN IS THE POSITIVE CLASS, AND POINT IS THE NEGATIVE CLASS.

Weight	Feature
Class: Gun	
0.02994	dcaaabb
0.02688	dcabaaaaaaa
Class: Point	
-0.02679	aabbbbbbbccd
-0.02839	bbbbbbbcbaa

Figure 9 illustrates these patterns on an example time series from each class. The top-2 features learned by SAX-VFSEQL agree with prior literature that studies these two datasets [9], [27], [28]. Nevertheless, they are directly tied to our linear classifier and enable us to deliver an explainable classification decision to the user.

VII. CONCLUSION

We propose new structure-based time series classification methods that are built on the popular SAX transformation and two new adaptations of an efficient linear sequence classifier, SEQL. Our aim is to deliver an efficient classifier, that is yet accurate and interpretable. We achieve this by extending the SEQL approach to support flexible fuzzy patterns, to reduce the need for tuning SAX parameters. We show in our experiments that the two proposed approaches, SAX-VSEQL and SAX-VFSEQL, work well with fixed SAX parameters and deliver efficiency and accuracy that is comparable to the state-of-the-art. An advantage of the proposed methods is that the resulting classifiers are very simple and interpretable, enabling us to explain the classification decision as a simple linear combination of extracted patterns.

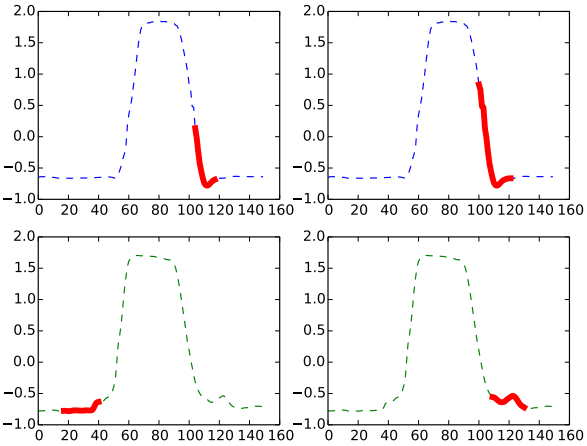

Fig. 9. Best two patterns for each class in Gun/Point dataset. Top is an example from the Point class and bottom and example from the Gun class.

ACKNOWLEDGMENT

This work was funded by Science Foundation Ireland (SFI) under grant number 12/RC/2289.

REFERENCES

[1] S. Ramgopal, S. Thome-Souza, M. Jackson, N. E. Kadish, I. S. Fernández, J. Klehm, W. Bosl, C. Reinsberger, S. Schachter, and T. Loddenkemper, "Seizure detection, seizure prediction, and closed-loop warning systems in epilepsy," *Epilepsy & Behavior*, vol. 37, pp. 291–307, 2014.

[2] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. Keogh, "Dynamic time warping averaging of time series allows faster and more accurate classification," in *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE, 2014, pp. 470–479.

[3] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," July 2015. [Online]. Available: www.cs.ucr.edu/~eamonn/time_series_data/

[4] P. Schäfer, "Scalable time series classification," *Data Mining and Knowledge Discovery*, pp. 1–26, 2015.

[5] 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*, vol. 1, no. 2, pp. 1542–1552, 2008.

[6] 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*, vol. 26, no. 2, pp. 275–309, 2013.

[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*. ACM, 2014, pp. 392–401.

[8] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: a novel symbolic representation of time series," *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[9] 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*, Dec 2013, pp. 1175–1180.

[10] G. Ifrim and C. Wiuf, "Bounded coordinate-descent for biological sequence classification in high dimensional predictor space," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. New York, NY, USA: ACM, 2011, pp. 708–716. [Online]. Available: http://doi.acm.org/10.1145/2020408.2020519

[11] G. Ifrim, G. Bakir, and G. Weikum, "Fast logistic regression for text categorization with variable-length n-grams," *Proceedings of the 14th ACM SIGKDD*, 2008.

[12] 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*. ACM, 2012, pp. 262–270.

[13] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with cote: the collective of transformation-based ensembles," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2522–2535, 2015.

[14] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, pp. 1–55, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10618-016-0483-9

[15] 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*. ACM, 2009, pp. 947–956.

[16] P. Esling and C. Agon, "Time-series data mining," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, p. 12, 2012.

[17] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *Proceedings of the thirteenth SIAM conference on data mining (SDM)*. SIAM, 2013, pp. 668–676.

[18] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, ser. DMKD '03. New York, NY, USA: ACM, 2003, pp. 2–11. [Online]. Available: http://doi.acm.org/10.1145/882082.882086

[19] J. Lin, R. Khade, and Y. Li, "Rotation-invariant similarity in time series using bag-of-patterns representation," *Journal of Intelligent Information Systems*, vol. 39, no. 2, pp. 287–315, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10844-012-0196-5

[20] P. Schäfer, "The boss is concerned with time series classification in the presence of noise," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1505–1530, 2015.

[21] Z. Hui and T. Hastie, "Regularization and variable selection via the elastic net," *Journal Of The Royal Statistical Society Series B*, vol. 67, no. 2, pp. 301–320, 2005.

[22] J. Nutini, M. Schmidt, I. H. Laradji, M. Friedlander, and H. Koepke, "Coordinate Descent Converges Faster with the Gauss-Southwell Rule Than Random Selection," *International Conference on Machine Learning*, vol. 37, no. Section 5, pp. 1–34, 2015. [Online]. Available: http://arxiv.org/abs/1506.00552

[23] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. MIT Press, 2001, pp. 841–848. [Online]. Available: http://robotics.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf

[24] C. Leslie, E. Eskin, and W. Noble, "Mismatch string kernels for svm protein classification," in *Neural Information Processing Systems*, 2002, pp. 1441–1448.

[25] J. Cheng and P. Baldi, "A machine learning information retrieval approach to protein fold recognition," *Bioinformatics*, vol. 22, no. 12, pp. 1456–1463, 2006.

[26] H. Huang and C. Lin, "Linear and kernel classification: When to use which?" in *Proceedings of the 2016 SIAM International Conference on Data Mining, Miami, Florida, USA, May 5-7, 2016*, S. C. Venkatasubramanian and W. M. Jr., Eds. SIAM, 2016, pp. 216–224. [Online]. Available: http://dx.doi.org/10.1137/1.9781611974348.25

[27] L. Ye and E. Keogh, "Time series shapelets: a novel technique that allows accurate, interpretable and fast classification," *Data Mining and Knowledge Discovery*, vol. 22, no. 1, pp. 149–182, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10618-010-0179-5

[28] J. Lines, L. M. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 289–297. [Online]. Available: http://doi.acm.org/10.1145/2339530.2339579