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].
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, SSEQ, designed to work on very long discrete sequences with large alphabets [10], [11]. SSEQ learns the best discriminative subsequences 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 [3], [9].
- **SAX-VFSEQ**, 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 [11], 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
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 $V$:

$$V = (v_1, v_2, \ldots, v_L)$$

$L$ 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.

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.

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., $\text{abcdabadbcbadbcba}$) while a word-level representation is a sequence of words (e.g., $\text{aba dac aac dac daa}$). 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 representation.](image)

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.](image)

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:

$$z_{\text{normalize}}(V) = \frac{V - \text{mean}(V)}{\text{std}(V)}$$

in which $V$ is the numeric vector of time series values, $\text{mean}(V)$ is the mean value of $V$ and $\text{std}(V)$ 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

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

is the modified time series where $v_j^i = v_j^k = v_j \forall i, j, k$. The resulting vector $\overline{V}$ 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**

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V$</td>
<td>Raw (normalized) numeric time series</td>
</tr>
<tr>
<td>$L$</td>
<td>Length of original time series</td>
</tr>
<tr>
<td>$w$</td>
<td>Number of X-axis bins</td>
</tr>
<tr>
<td>$a$</td>
<td>Number of Y-axis bins</td>
</tr>
<tr>
<td>$l$</td>
<td>Size of sliding window</td>
</tr>
</tbody>
</table>

### IV. CLASSIFICATION WITH SEQUENCE LEARNER

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 $\Sigma$ 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: $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})^T$, $x_{ij} \in \{0, 1\}$, $i = 1, \ldots, N$, 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 SEQUENTIAL FRAMEWORK**

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>Set of sequences</td>
</tr>
<tr>
<td>$D$</td>
<td>Training set</td>
</tr>
<tr>
<td>$s_i$</td>
<td>Sequence $i$</td>
</tr>
<tr>
<td>$y_i$</td>
<td>Score of sequence $i$</td>
</tr>
<tr>
<td>$x_{ij}$</td>
<td>Indicator of subsequence $s_j$ for sequence $s_i$</td>
</tr>
<tr>
<td>$N$</td>
<td>Number of training examples</td>
</tr>
<tr>
<td>$m$</td>
<td>Sequence length</td>
</tr>
<tr>
<td>$d$</td>
<td>The number of distinct subsequences</td>
</tr>
<tr>
<td>$\Sigma$</td>
<td>The alphabet set, e.g., ${A, C, G, T}$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>The number of classes, $c$</td>
</tr>
</tbody>
</table>

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:

$$\beta^* = \arg\min_{\beta \in \mathbb{R}^d} L(\beta)$$

(4)

where

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

(5)

Most $\beta_j$ 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):

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

(6)
\[ \xi(y_i, x_i, \beta) = \max(1 - y_i \beta^T \cdot x_i, 0)^2 \]  

In Equation 5 \[ C \in \mathbb{R}_{++}^d \] denotes the weight for the regularizer \( R_{\beta}(\beta) \). Larger \( C \) leads to larger penalty for \( \beta \) and can influence the sparsity of the learned model. SEQL implements the elastic-net regularization for \( R_{\alpha}(\beta) \) \cite{Zhu2008}, which combines \( l_1 \) and \( l_2 \) 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 \( \beta \) is potentially huge. The work in \cite{SAX2005, SAX2005b} 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 \( \eta_t \)
6: Update \( \beta^{(t)} = \beta^{(t-1)} - \eta_t \frac{\partial L}{\partial \beta^{(t-1)}}(\beta^{(t-1)})e_{j_t} \)
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.

Algorithm 2 Search for best subsequence in SEQL

1: \( \tau \leftarrow 0 \)
2: \( \text{best\_feature} \leftarrow \text{NIL} \)
3: for all \( s' \in \bigcup_{i=1}^{N} \{ s \mid s \in x_i, |s| = 1 \} \) do \( \triangleright \) For each unigram
4: \( \text{GROW\_SEQUENCE}(s') \)
5: end for
6: return \( \text{best\_feature} \)
7: function \( \text{GROW\_SEQUENCE}(s) \)
8: if \( \mu(s) \leq \tau \) then return \( \triangleright \mu(s) \) bound as in \cite{SAX2005} \( \triangleright \) Suboptimal solution
9: else if \( \text{abs}(	ext{gradient}(s)) > \tau \) then
10: \( \text{best\_feature} = s \)
11: \( \tau = \text{abs}(	ext{gradient}(s)) \)
12: end if
13: for all \( s'' \in \{ s' \mid s' \supseteq s, s' \in \bigcup_{i=1}^{N} x_i, |s'| = |s| + 1 \} \) do
14: \( \text{GROW\_SEQUENCE}(s'') \)
15: end for
16: end function

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 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 \cite{SAX2007, SAX2008, SAX2009, SAX2010}. 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 $a b b$ similar to subsequence $b b b$, 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.

$$d(c_1, c_2) = |index(c_1) - index(c_2)|$$

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 $s_1 = a_1 a_2 \ldots a_L$ and $s_2 = b_1 b_2 \ldots b_L$ be the two symbolic sequences of equal length. The distance between $s_1$ and $s_2$ is

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

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

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

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

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

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

**Theorem 1 (Bounding the search for the best coordinate with fuzzy matching).** For any loss function $\xi$ 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 $s_p$):

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

where

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

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 $s_j$.

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

$$= \sum_{\{i|x_{ij} \in (0, 1)\}} y_i x_{ij} \xi'(m_i) + CR'_{\alpha}(\beta_j)$$

$$\leq \sum_{\{i|x_{ij} \in (0, 1), y_i = -1\}} y_i x_{ip} \xi'(m_i) + CR'_{\alpha}(\beta_j)$$

$$\leq \sum_{\{i|x_{ip} \in (0, 1), y_i = -1\}} y_i x_{ip} \xi'(m_i) + CR'_{\alpha}(\beta_j)$$

$$= \sum_{\{i|x_{ip} \in (0, 1), y_i = +1\}} -x_{ip} \xi'(m_i) + CR'_{\alpha}(\beta_j)$$

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 $s_{pi}$, thus: $x_{ij} \leq x_{ip}, i = 1, \ldots, N$.

Similarly, we compute a bound for the positive examples:

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

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]):

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

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

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-VFSEQ 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 VFSEQ 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-VFSEQ combo has a total time complexity of $O(NL) + O(Nf)$. The test stage of SAX-VFSEQ has a complexity of $O(L)$ as it only requires a linear scan of the time series.

**Convergence 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 VFSEQ). 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-VSEQ and SAX-VFSEQ 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>
<thead>
<tr>
<th>Dataset</th>
<th>Train</th>
<th>Pos/Neg</th>
<th>Test</th>
<th>Pos/Neg</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coffee</td>
<td>28</td>
<td>14/14</td>
<td>28</td>
<td>15/13</td>
<td>286</td>
</tr>
<tr>
<td>Earthquakes</td>
<td>139</td>
<td>104/35</td>
<td>322</td>
<td>264/58</td>
<td>512</td>
</tr>
<tr>
<td>ECG200</td>
<td>100</td>
<td>31/69</td>
<td>100</td>
<td>36/64</td>
<td>96</td>
</tr>
<tr>
<td>ECGFiveDays</td>
<td>23</td>
<td>14/9</td>
<td>861</td>
<td>428/433</td>
<td>136</td>
</tr>
<tr>
<td>FordA</td>
<td>1320</td>
<td>681/639</td>
<td>3601</td>
<td>1846/1755</td>
<td>500</td>
</tr>
<tr>
<td>FordB</td>
<td>810</td>
<td>401/409</td>
<td>3630</td>
<td>1860/1776</td>
<td>500</td>
</tr>
<tr>
<td>Gun_Point</td>
<td>50</td>
<td>24/26</td>
<td>150</td>
<td>76/74</td>
<td>150</td>
</tr>
<tr>
<td>Lighting2</td>
<td>60</td>
<td>20/40</td>
<td>61</td>
<td>28/33</td>
<td>637</td>
</tr>
<tr>
<td>MoteStrain</td>
<td>20</td>
<td>10/10</td>
<td>1252</td>
<td>675/577</td>
<td>84</td>
</tr>
<tr>
<td>PassGraph</td>
<td>69</td>
<td>36/33</td>
<td>131</td>
<td>64/67</td>
<td>364</td>
</tr>
<tr>
<td>SonyAIBORS</td>
<td>20</td>
<td>6/14</td>
<td>601</td>
<td>343/258</td>
<td>70</td>
</tr>
<tr>
<td>SonyAIBORSII</td>
<td>27</td>
<td>11/16</td>
<td>953</td>
<td>365/558</td>
<td>65</td>
</tr>
<tr>
<td>TwoLeadECG</td>
<td>23</td>
<td>12/11</td>
<td>1139</td>
<td>569/70</td>
<td>82</td>
</tr>
<tr>
<td>wafer</td>
<td>1000</td>
<td>97/903</td>
<td>6164</td>
<td>665/599</td>
<td>152</td>
</tr>
<tr>
<td>yoga</td>
<td>300</td>
<td>137/163</td>
<td>3000</td>
<td>1393/1607</td>
<td>426</td>
</tr>
</tbody>
</table>

**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 $l = 0.2 \times L$ (where $L$ is length of the time series). For SEQL parameters, we use the default values set in the open source code[^2] 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**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Raw</th>
<th>DiscX(w)</th>
<th>DiscY(a)</th>
<th>SAXchars(w,a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coffee</td>
<td>0.4286</td>
<td>0.4643</td>
<td>0.1071</td>
<td>0.2143</td>
</tr>
<tr>
<td>Earthquakes</td>
<td>0.2019</td>
<td>0.1801</td>
<td>0.2453</td>
<td>0.2638</td>
</tr>
<tr>
<td>ECG200</td>
<td>0.63</td>
<td>0.64</td>
<td>0.23</td>
<td>0.25</td>
</tr>
<tr>
<td>ECGFiveDays</td>
<td>0.5006</td>
<td>0.5029</td>
<td>0.3136</td>
<td>0.338</td>
</tr>
<tr>
<td>FordA</td>
<td>0.4788</td>
<td>0.4874</td>
<td>0.2085</td>
<td>0.3602</td>
</tr>
<tr>
<td>FordB</td>
<td>0.4981</td>
<td>0.4884</td>
<td>0.2162</td>
<td>0.4686</td>
</tr>
<tr>
<td>Gun_Point</td>
<td>0.4667</td>
<td>0.4933</td>
<td>0.06</td>
<td>0.1333</td>
</tr>
<tr>
<td>Lighting2</td>
<td>0.4754</td>
<td>0.541</td>
<td>0.3279</td>
<td>0.3443</td>
</tr>
<tr>
<td>MoteStrain</td>
<td>0.4609</td>
<td>0.4699</td>
<td>0.1151</td>
<td>0.1845</td>
</tr>
<tr>
<td>Passgraph</td>
<td>0.5115</td>
<td>0.5115</td>
<td>0.2366</td>
<td>0.4122</td>
</tr>
<tr>
<td>SonyAIBO</td>
<td>0.4309</td>
<td>0.4293</td>
<td>0.3694</td>
<td>0.3794</td>
</tr>
<tr>
<td>SonyAIBOII</td>
<td>0.6149</td>
<td>0.617</td>
<td>0.2875</td>
<td>0.3221</td>
</tr>
<tr>
<td>TwoLeadECG</td>
<td>0.4996</td>
<td>0.5094</td>
<td>0.1062</td>
<td>0.2564</td>
</tr>
<tr>
<td>wafer</td>
<td>0.6723</td>
<td>0.5357</td>
<td>0.016</td>
<td>0.0836</td>
</tr>
<tr>
<td>yoga</td>
<td>0.4837</td>
<td>0.3123</td>
<td>0.299</td>
<td>0.3293</td>
</tr>
</tbody>
</table>

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).

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**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SAX-SEQL</th>
<th>SAX-VSEQL</th>
<th>SAX-VFSEQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coffee</td>
<td>0.0714</td>
<td>0.0</td>
<td>0.0357</td>
</tr>
<tr>
<td>Earthquakes</td>
<td>0.2081</td>
<td>0.2112</td>
<td>0.205</td>
</tr>
<tr>
<td>ECG200</td>
<td>0.25</td>
<td>0.26</td>
<td>0.18</td>
</tr>
<tr>
<td>ECGFiveDays</td>
<td>0.0743</td>
<td>0.0244</td>
<td>0.0174</td>
</tr>
<tr>
<td>FordA</td>
<td>0.5685</td>
<td>0.1486</td>
<td>0.1157</td>
</tr>
<tr>
<td>FordB</td>
<td>0.5069</td>
<td>0.2175</td>
<td>0.2134</td>
</tr>
<tr>
<td>Gun_Point</td>
<td>0.04</td>
<td>0.0133</td>
<td>0.02</td>
</tr>
<tr>
<td>Lighting2</td>
<td>0.377</td>
<td>0.2951</td>
<td>0.2951</td>
</tr>
<tr>
<td>MoteStrain</td>
<td>0.1845</td>
<td>0.1597</td>
<td>0.1661</td>
</tr>
<tr>
<td>Passgraph</td>
<td>0.3511</td>
<td>0.2595</td>
<td>0.3128</td>
</tr>
<tr>
<td>SonyAIBO</td>
<td>0.2363</td>
<td>0.5694</td>
<td>0.3727</td>
</tr>
<tr>
<td>SonyAIBOII</td>
<td>0.3284</td>
<td>0.1723</td>
<td>0.0965</td>
</tr>
<tr>
<td>TwoLeadECG</td>
<td>0.021</td>
<td>0.0623</td>
<td>0.0527</td>
</tr>
<tr>
<td>wafer</td>
<td>0.1898</td>
<td>0.0071</td>
<td>0.0079</td>
</tr>
<tr>
<td>yoga</td>
<td>0.1977</td>
<td>0.2313</td>
<td>0.2033</td>
</tr>
</tbody>
</table>

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

In the next set of experiments which also include multiclass datasets, we compare our best approach, SAX-VFSEQL, with state-of-the-art techniques including, 1NN-ED, 1NN-DTW [3], FastShapletes [17], SAX-VSM [4] 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, as we show in the next section. SAX-VSM requires high running time for tuning parameters. We provide the code of our

[^2]: https://github.com/heerme/seql-sequence-learner
method as well as accuracy and running time results for all methods and datasets at [https://github.com/thach/saxseql](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

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixedSAX-VSM</td>
<td>SAX-VSM with fixed parameters (a = 4, w = 16, l = 0.2 \times L).</td>
</tr>
<tr>
<td>optSAX-VSM</td>
<td>SAX-VSM with optimized parameters.</td>
</tr>
<tr>
<td>SAX-VSM*</td>
<td>Results reproduced by [14].</td>
</tr>
<tr>
<td>BOSS VS</td>
<td>Results reproduced with the author’s implementation [4].</td>
</tr>
<tr>
<td>FastShapelets</td>
<td>SAX-based method, results from [14].</td>
</tr>
<tr>
<td>1NN-Euclidean</td>
<td>Results taken from the UCR Archive.</td>
</tr>
<tr>
<td>1NN-DTW</td>
<td>Results taken from the UCR Archive.</td>
</tr>
</tbody>
</table>

![SAX-VFSEQL accuracy against state-of-the-art classifiers.](image)

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

### TABLE VII

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

<table>
<thead>
<tr>
<th>Size of training data</th>
<th>SAX-VFSEQL</th>
<th>SAX-VSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>10.97</td>
<td>3.4</td>
</tr>
<tr>
<td>120</td>
<td>22.19</td>
<td>13.65</td>
</tr>
<tr>
<td>240</td>
<td>47.22</td>
<td>55.57</td>
</tr>
<tr>
<td>480</td>
<td>96.56</td>
<td>265.14</td>
</tr>
<tr>
<td>960</td>
<td>176.65</td>
<td>1245.79</td>
</tr>
<tr>
<td>1920</td>
<td>303.96</td>
<td>5569.41</td>
</tr>
</tbody>
</table>

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.

![Scalability with training size and length of time series. Running time (sec) on CBF synthetic dataset.](image)

**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 are more interpretable and potentially more useful.
(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 features learned by SAX-VFSEQL, work well with fixed SAX parameters and 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. Nevertheless, they are directly tied to our linear classifier and enable us to deliver an explainable classification decision to the user.

Figure 8 illustrates the best and second best patterns found for 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 from two classes: Gun and Point. Table X illustrates the top-2 positive and negative features selected by the classifier, and 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


