Learning sequential classifiers from long and noisy discrete-event sequences efficiently

Gessé Dafé · Adriano Veloso · Mohammed Zaki · Wagner Meira Jr.

Abstract A variety of applications, such as information extraction, intrusion detection and protein fold recognition, can be expressed as sequences of discrete events or elements (rather than unordered sets of features), that is, there is an order dependence among the elements composing each data instance. These applications may be modeled as classification problems, and in this case the classifier should exploit sequential interactions among the elements, so that the ordering relationship among them is properly captured. Dominant approaches to this problem include: (i) learning Hidden Markov Models, (ii) exploiting frequent sequences extracted from the data and (iii) computing string kernels. Such approaches, however, are computationally hard and vulnerable to noise, especially if the data shows long range dependencies (i.e., long subsequences are necessary in order to model the data). In this paper we provide simple algorithms that build highly effective sequential classifiers. Our algorithms are based on enumerating approximately contiguous subsequences from the training set on a demand-driven basis, exploiting a lightweight and flexible subsequence matching function and an innovative subsequence enumeration strategy called *pattern silhouettes*, making our learning algorithms fast and the corresponding classifiers robust to

G. Dafé · A. Veloso (✉) · W. Meira Jr.
Computer Science Department, Federal University of Minas Gerais, Belo Horizonte, Brazil
e-mail: adrianov@dcc.ufmg.br

G. Dafé
e-mail: gdafe@dcc.ufmg.br

W. Meira Jr.
e-mail: meira@dcc.ufmg.br

M. Zaki
Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA
e-mail: zaki@cs.rpi.edu

Published online: 04 November 2014

🙂 Springer

noisy data. Our empirical results on a variety of datasets indicate that the best trade-off between accuracy and learning time is usually obtained by limiting the length of the subsequences by a factor of $\log n$, which leads to a $O(n \log n)$ learning cost (where n is the length of the sequence being classified). Finally, we show that, in most of the cases, our classifiers are faster than existing solutions (sometimes, by orders of magnitude), also providing significant accuracy improvements in most of the evaluated cases.

Keywords Sequential classifiers · Efficient learning · Long range sequences · Partial matching · Approximately contiguous sequences

1 Introduction

Classification algorithms have been traditionally designed for dealing with vectorial data, thus interpreting each training example as an unordered set of features. In many application scenarios, however, a training example is represented as a sequence of discrete events or elements, and, therefore, there is an explicit ordering relationship among these elements. This is commonly observed in application scenarios such as information extraction, intrusion detection, and protein fold recognition (Han et al. 2004, 2005; Lodhi et al. 2009; Silva et al. 2011). In such cases, it may be advantageous to interpret examples as sequences of elements, since the order relation may contain relevant (or even necessary) information about the meaning of the data.

Algorithms that produce classifiers from discrete event sequence data (i.e., sequential classifiers) may rely on (i) building generative classifiers by learning Hidden Markov Models (HMMs) (Baum and Petrie 1966; Baum et al. 1970), (ii) transforming the original input space into another one in which each transformed feature corresponds to a frequent subsequence of elements in the original space (Tseng and Lee 2005), (iii) combining frequent pattern mining and Hidden Markov Model approaches (Zaki et al. 2010) and (iv) computing string kernels (Lodhi et al. 2002; Leslie and Kuang 2004, 2003; Leslie et al. 2002b). These algorithms, however, may suffer from non-negligible scalability shortcomings, possibly preventing them to provide fast learning times in scenarios where long subsequences are necessary in order to properly model the data. Also, discrete event sequence data are commonly plagued by noise. For instance, in the case of protein-related data the collected sequences may have large variability with only a small subset of variations relevant to the protein function. Noise is also present in micro-blog messages, which are typically written in informal, sometimes cryptic style, due to space limitations. Under such conditions, most sequential classification algorithms offer limited accuracy. Furthermore, such algorithms are often devised for a specific learning task (Agrawal and Srikant 1995; Eddy 1998).

In this paper we propose general purpose sequential classification algorithms by taking into consideration the trade-off that exists between their run-time numbers and the accuracy of the resulting classifier. Specifically, our objective is to learn sequential classifiers that are general enough to handle diverse sources of discrete sequence data without the need of any adaptation, while offering state-of-the-art performance in terms of learning time and accuracy. We start by introducing algorithms that rely

only on exact matches, that is, classifiers are built upon subsequences composed of contiguous elements (i.e., no gaps and mismatches are allowed). By dynamically bounding the length of these subsequences, the corresponding classifier becomes able to capture not only short, but also long range dependencies in the data. From considerations of efficiency, we focus on classifiers with $O(n \log n)$ complexity in the sequence length, since this choice leads to the best balance between efficiency and accuracy on the datasets used in our experiments. A drawback, however, is that contiguous sequential classifiers may become excessively restrictive, since no violation in the ordering relationship among elements is allowed while enumerating subsequences. As a result, the classifier is usually composed of very few subsequences, being not sufficiently expressive or very sensitive to noise, compromising classification accuracy.

To overcome this limitation we relax the subsequence enumeration process in order to allow gaps and mismatches while comparing subsequences. More specifically, we formulate a lightweight matching function which compares a test sequence with the relevant sequences in the training data, and measures the similarity between them (Rieck and Laskov 2008). The similarity is based on how many contiguous matching elements they have in common (i.e., the more subsequences two sequences have in common, the more similar they are and the higher is the likelihood of both belonging to the same class). Violations in the contiguity of the subsequences (i.e., mismatches) are allowed, but properly penalized. The proposed similarity function differs from most string kernels because it takes into account not only the number of mismatches, but also how close are they. That is, consecutive mismatches are highly penalized, but isolated mismatches that are likely to occur due to noise are tolerated. Relevant training sequences are rapidly extracted from the training data in a demand-driven basis (Veloso and Meira 2011), by indexing what we call pattern silhouettes, ensuring that only training sequences that are to some extent similar to the test sequence are considered, drastically reducing the number of necessary subsequence comparisons. As a result, sequential classifiers become more expressive and robust to noise, but are still efficiently learned in $O(n \log n)$. These features make our classifiers an interesting alternative to typical solutions, such as string kernels and HMMs. The run-time complexity for computing kernels, for instance, is determined by the number of model parameters, so that only very simple models yield run-time linear in the sequence lengths. Moreover, obtaining a suitable parameter estimate for a probabilistic model can be difficult or even infeasible in practical application scenarios.

To validate our claims and to evaluate the effectiveness of the proposed algorithms, we performed a systematic set of experiments using real sequence data obtained from a variety of application scenarios such as information extraction, protein fold recognition, intrusion detection, among others. The results show that our algorithms provide, in most of the cases, significant improvements in terms of execution time, without putting classification accuracy at risk when compared against state-of-the-art solutions. In fact, in most of the cases, our proposed algorithms are able to improve classification accuracy as well.

2 Related work

The problem we are interested to study in this paper is to distinguish between sequences belonging to different labeled groups or classes (Zaki et al. 2010; Syed et al. 2009). In particular, we are interested in sequences of discrete events, rather than time series or temporal intervals. Next we provide a brief review of the most relevant work related to this problem.

2.1 Sequential patterns

The problem of enumerating sequences that occur frequently in the data was first studied by Agrawal and Srikant (1995). Improved algorithms for finding frequent sequential patterns were proposed in (Srikant and Agrawal 1996; Zaki 2000). These algorithms employ constraints such as minimum and maximum gaps between consecutive elements, allowing for a more flexible enumeration of sequences (i.e., sequences that occur after some given time interval). The use of sequential patterns as features for the sake of learning sequential classifiers was initially proposed in (Lesh et al. 1999), and more recently in (Lin et al. 2009). As a major deficiency of these algorithms, it is worth mentioning that they are unable to extract high-order, long sequential patterns efficiently, therefore capturing only short-range dependencies in the data.

Algorithms based on the recursive data mining approach, proposed in (Szymanski 2004), are able to bridge large gaps between consecutive elements in the sequence, but these algorithms were devised for solving specific tasks, such as intrusion detection and author identification. In (Syed et al. 2009) the authors proposed an algorithm which efficiently finds patterns in labeled sequences, that are then used for classification of sequential data. In (Bannister 2007) an algorithm is proposed to learn classifiers from sequential data by exploiting the algorithmic relationship between association rule mining and sequential pattern mining. Frequent sequential patterns are mined and then the constrained adaptive methodology is applied to select patterns to be used for classifying the outcome. This algorithm, which we call associative sequential classifier (ASC), is a representative of state-of-the-art solutions for several applications based on sequential data. We implemented a similar version of this algorithm and used it as one of the baselines for comparison. Efficiently capturing long-range dependencies among elements and dealing with sequence violations pose challenging problems. Next we discuss solutions based on learning Hidden Markov Models and string kernels, which employ different strategies to handle this problem.

2.2 Hidden Markov Models

Hidden Markov models (HMMs) are probabilistic models of sequential data (Rabiner 1989). HMMs can be viewed as stochastic generalizations of finite-state automata, when both the transitions between states and the generation of output symbols are governed by probabilistic mechanisms. Specific HMM based approaches have been

proposed for different applications, such as DNA and protein modeling (Durbin et al. 1998; Hughey and Krogh 1996), speech recognition (Rabiner 1989; Sha and Saul 2006), handwritten character recognition (Hu et al. 1996), gesture recognition (Müller et al. 2000), among others.

In order to capture long range sequential dependencies, which is not feasible with simple (first-order) HMMs, many approaches based on fixed high-order HMMs have been proposed. In (Kriouile et al. 1990), a first-order HMM, based on Viterbi and Baum-Welch algorithms (Durbin et al. 1998), is used for state prediction and to directly train a second-order HMM. The method proposed by (Du Preez 1998) converts a fixed high-order HMM to an equivalent first-order model which is used to incrementally train a high-order HMM. Another fixed-order approach is found in (Law and Chan 1996), an n-gram-based HMM for language modeling. Although providing an elegant and sound sequential data modeling methodology, a major drawback in fixed high-order HMMs is that such models suffer from high state-space complexity, since a k-th order HMM, with alphabet Σ, can potentially have $|\Sigma|^k$ states. Therefore, estimating the joint probabilities of each k-th order state is extremely difficult. Furthermore, none of those techniques are able to capture eventual mismatches in sequences, maybe being excessively rigid to deal with noisy data.

Other approaches try to efficiently build variable-order HMMs. Mixed order Markov models were proposed by (Schwardt and Preez 2000), but, since they rely on expectation-maximization methods, they are susceptible to local optima. In (Srivatsan et al. 2005), specific episode generating HMMs (EGMs) are built for each frequent episode (sequence) mined from the data. Besides the huge number of generated models, only non-overlapping sequences are found and violations in sequences are not explicitly handled. In (Bicego et al. 2003), a pruning strategy was proposed to avoid excessively large number of states in HMMs. In (Wang et al. 2006), a variable-length HMM (VLHMM) is built upon Markov chains of variable memory length (Bühlmann and Wyner 1999), by storing context in a prediction prefix tree. This method also employs an expectation-maximization strategy for training, so is prone to local optima. Furthermore it requires the number of states to be given as user parameter. The similarity-based recognition paradigm was also used in the context of HMMs, resulting in significant improvements with respect to standard HMM-based approaches when applied to clustering, pattern recognition and sequential classification related tasks (Bicego et al. 2006, 2004).

In (Zaki et al. 2010), the authors proposed the VOGUE algorithm, which addresses the main limitations of HMMs. This algorithm proposes a mixture between the two approaches: mining sequential patters and learning HMMs. It relies on a variable gap sequence mining approach, extracting frequent sequential patterns with different lengths and gaps between elements. The extracted patterns are then used to build a variable order HMM. The VOGUE algorithm is a representative of state-of-the-art solutions for several applications based on sequential data, and is used as one of the baselines for comparison.

2.3 String kernels

String kernels extend methods such as SVMs (Vapnik 1979), so that they become able to handle sequence data. String kernels were first used for text classification (Watkins 1999), and then for diverse applications (Lodhi et al. 2002; Haussler 1999). The cost of computing kernels scales quadratically as a function of the length of the input sequences, being unfeasibly slow for applications involving long sequences. Furthermore, they do not handle mismatches between sequences, being excessively sensitive to noise.

Latter approaches overcome the aforementioned limitations due to strict matching by employing mismatch-tolerant comparison techniques, therefore building more flexible kernels. However, as these methods aim at computing similarity for all pairs of sequences in a particular feature space, they face a serious computational challenge. To narrow down the cost of this task, these techniques typically restrict the length and expressive power of the subsequences used as features, by indexing subsequences that they call *k-mers*, where k is the subsequence length, which is usually small. An example of these approaches can be found in Spectrum-k Kernel (Leslie et al. 2002a), which implicitly compares k-mers. Following the same rationale, the Mismatch Kernel (Leslie et al. 2004) generalizes the Spectrum-k Kernel by allowing mismatches to accommodate mutations. These new kernels, namely restricted gappy kernels, substitution kernels, and wildcard kernels, are based on feature spaces indexed by k-mers. The maximum number of consecutive mismatches is limited with explicit parameters. The sparse spatial sample kernel (SSSK) (Kuksa et al. 2008) generalizes the Mismatch Kernel by sampling the subsequences at different resolutions and comparing the resulting spectra. Mismatches are handled by probes, whose number, lengths and distances are given as model parameters.

A practical disadvantage of computing string kernels is their computational expense. In general, these kernels rely on dynamic programming algorithms for which the computation of each kernel value $K(x, y)$ is quadratic in the length of the input sequences x and y, that is, $O(|x||y|)$ with constant factor that depends on the parameters of the kernel. As an exception, Leslie et al. (2002a) introduced a linear time $O(k|x| + |y|)$ implementation of the Spectrum-k Kernel, which relies only on exact matches of k-mer patterns.

Although kernel-based sequence learning has gained significant attention of the community, there are many other possibilities and strategies for learning sequential classifiers. In particular, we discuss classifiers built upon similarity measures for discrete event sequential data. Such measures are fastly computed using algorithms that employ a demand-driven training projection strategy, which allows a drastic reduction in the number of sequence comparisons by using indexes which we call *pattern silhouettes*. This strategy allows our algorithms to deal with long and possibly noisy sequences, improving accuracy and reducing computational costs in different application scenarios. A concept which is somehow similar to our proposed pattern silhouettes are kwnon as time-series shapelets (Ye and Keogh 2009, 2011). However, shapelets are used for building classifiers from time-series data, while pattern silhouettes are devoted to discrete event sequences.

3 Learning sequential classifiers on a demand-driven basis

In this paper we consider our input data as a sequence of discrete events. The problem of learning sequential classifiers from this data can be formally stated as follows:

Given a training set D composed of d groups of sequences $D = \{D_1, D_2, \ldots, D_d\}$ such that all sequences that belong to the same group D_i have the same class label y_i, learn a classifier C which maps an arbitrary sequence to the most likely class label.

This formulation is sufficiently general to be applied to a wide variety of classification scenarios where sequential data exists (Syed et al. 2009), and most of the existing classification algorithms follow this formulation. In this work, we propose an alternate formulation in order to make explicit the notion of learning classifiers on a demand-driven basis. In this case, a specific classifier is built for each sequence in the test set T. Making this notion clear is important because it allows our proposed algorithms to avoid unnecessary sequence comparisons, as discussed later. Our alternate formulation is stated as follows:

Given a training set D composed of d groups of sequences $D = \{D_1, D_2, \ldots, D_d\}$ and a test instance $t \in T$, learn a classifier C_{t_j}, which is a function mapping sequence t_j to the most likely class label. The process is repeated for all sequences in $T = \{t_1, t_2, \ldots, t_m\}$, resulting in multiple classifiers $\{C_{t_1}, C_{t_2}, \ldots, C_{t_m}\}$.

Next we discuss classification algorithms that learn sequential classifiers under this formulation. We start with a very simple algorithm, called SC-SC (Strictly-Contiguous Sequential Classifier), which learns sequential classifiers based on contiguous subsequences, that is, subsequences are composed only of adjacent elements. Then, we introduce a more sophisticated algorithm, called approximately-contiguous sequential classifier (AC-SC), which learns classifiers by allowing mismatches between sequences in D (i.e., examples) and a sequence $t \in T$ (i.e., a test instance), but these mismatches are penalized using a variable decay factor which increases with consecutive mismatches. A similarity function is used to assess the amount of contiguity that exists between the test sequence and the corresponding relevant sequences in D, making AC-SC specially suited for dealing with noisy data, even in cases for which long subsequences are necessary for modeling the data.

3.1 Learning contiguous sequential classifiers

In this section we describe the SC-SC algorithm which exploits the ordering relationship among elements by enumerating contiguous matches between sequences, as defined next.

Definition 1 (*Contiguous Matching*) A subsequence X is said to contiguously match instance $t \in T$ (which is given as a sequence of n elements $\{a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n\}$), if $X = \{a_i \rightarrow a_{i+1} \rightarrow \cdots \rightarrow a_{i+k}\}$, provided that $k \geq 0$.

The strategy for enumerating strictly contiguous sequences may be seen as an iterative sliding-window process in which the window size is increased after each iteration.

Fig. 1 Contiguous sequences
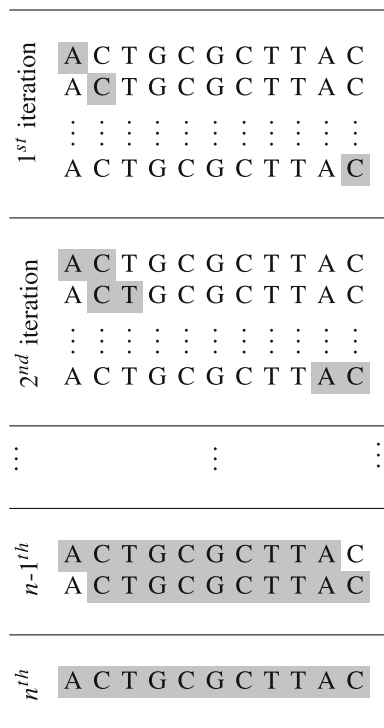
More specifically, given a test sequence $t \in T$ such that $t = \{a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n\}$, the sequence enumeration process starts by enumerating singleton elements that are in t. In the second iteration, the window size increases and sequences composed of pairs of consecutive elements are enumerated. The process iterates enumerating ever increasing contiguous subsequences in t. Figure 1 shows the enumeration process given an arbitrary test sequence.

As easily noticed, the number of sequences that contiguously match an arbitrary sequence $t \in T$ is given by an arithmetic progression which clearly grows quadratically with the number of elements within t (i.e., n). The classifier C_t built using the SC-SC algorithm is composed of contiguous subsequences, as the ones depicted in Fig. 1. Therefore, the cardinality of C_t is given by Eq. 1:

$$|C_t| = \sum_{k=1}^{n} n - k + 1 = \frac{n^2 + n}{2} = O(n^2) \tag{1}$$

In practice however, given a test sequence $t \in T$, there is no need for an exhaustive enumeration of all contiguous subsequences in t: classification accuracy and sequence length are not linearly related (Malik and Kender 2008). In fact, classification accuracy typically increases slowly (and eventually stabilizes) as the sequences composing the classifier become longer (Tseng and Lee 2005). Therefore, we may bound the length of the subsequences to be enumerated, so that no subsequence with more than m

elements is enumerated from D. We employ a variable limitation strategy, skipping subsequences with more than $\log n$ elements. This ensures $O(n \log n)$ learning cost with respect to the number of elements in t, as can be seen in Eq. 2.

$$m = \lfloor \log n \rfloor \implies |C_t| = (n) + (n-1) + (n-2) + \cdots + (n - \lfloor \log n \rfloor) = O(n \log n)$$
(2)

Other limits on sequence length could also be used (i.e., m is an empirical threshold). For instance, we may use limitation strategies such as linear and square root functions, but the justification for limiting the sequence length to a maximum of $\log n$ elements is that more balanced results are reached when taking into account the trade-off between accuracy and learning time. We will present more detailed analysis regarding this trade-off in Sect. 4.

3.2 Calculating class membership

For each subsequence $X \in C_t$, we calculate $\theta(X \to y_i)$, which is an estimate of the conditional probability of X being associated with class label y_i (i.e., an estimate, obtained from the training set, of the probability of X belonging to D_i). Such subsequences are used to build a weighted vote for each class label, where the weight depends on the θ value associated with the corresponding subsequence. Weighted votes for label y_i are averaged, giving a score for label y_i with regard to sequence t, as shown in Eq. 3:

$$w(t, y_i) = \frac{\sum \theta(X \to y_i)}{|C_t^{y_i}|}$$
(3)

where $C_t^{y_i}$ is the subset of C_t composed of subsequences that are associated with class label y_i, and the summation is over all rules $\{X \to y_i\} \in C_t^{y_i}$. Finally, the scores are normalized, as expressed by function $\hat{p}(y_i|t)$, shown in Eq. 4. This function estimates the relevance of a sequence $t \in T$ with regard to label y_i.

$$\hat{p}(y_i|t) = \frac{w(t, y_i)}{\sum_{j=0}^{d} w(t, y_j)} \text{, where } d \text{ is the number of possible labels.}$$
(4)

Despite being computationally efficient, the SC-SC algorithm may produce sequential classifiers that are excessively restrictive, since no violation in the ordering relationship among elements is allowed while enumerating subsequences. A single mismatch between test and training sequences causes the rejection of several (potentially relevant) subsequences. As a result, the classifier will be probably very sensitive to noise and be composed of very few subsequences, compromising classification accuracy. Fortunately, these disadvantages can be avoided by relaxing the subsequence enumeration process without sacrificing learning time, as discussed in the next section.

3.3 Learning noise-tolerant sequential classifiers

In contrast to strictly-contiguous sequential classifiers, approximately-contiguous sequential classifiers are lenient with mismatches when enumerating subsequences. We propose a classification algorithm for learning approximately-contiguous sequential classifiers, which relies on assessing the similarity between test and training sequences. This similarity is calculated for each pair (x, t) such that $x \in D$ and $t \in T$. The proposed AC-SC algorithm performs three main steps as described next.

3.4 Pre-indexing pattern silhouettes

Scanning the entire training set D searching for approximate matchings every time a sequence $t \in T$ is given to classification is obviously unfeasible. Since we are interested in dealing with approximately-contiguous matching, which we define next, it is necessary to optimize sequence enumeration by pre-indexing sequence occurrences in D. The challenge here is to narrow down the search space for relevant subsequences, by investigating only training sequences $x \in D$ that have the same shape of the test sequence being classified. The way we propose to solve this problem is by pre-indexing what we call *Pattern Silhouettes*, which can be seen as a hash function which returns sequences in D that may be similar to a given test sequence. That is, patterns silhouettes are used to retrieve all subsequences in D that are relevant to a specific test sequence.

Definition 2 (*Pattern Silhouettes*) Let $t = \{a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_n\}$ denote an arbitrary sequence in T. A pattern silhouette is any triple of form $s = (a_l, a_r, k)$, with $r \geq l$, and $r - l = k$, where a_l and a_r are elements in t and $k + 1$ is the length of the subsequence ranging from a_l to a_r. A subsequence X is said to approximately match sequence $t \in T$, if X and t share at least one pattern silhouette.

The AC-SC algorithm first enumerates all eligible pattern silhouettes for each training sequence $x \in D$. Then, it constructs an inverted index, which links each pattern silhouette to all sequences in D containing it. This inverted index is used in further steps. The process of enumerating silhouettes follows the same sliding-window approach described for enumerating strictly-contiguous subsequences.

3.5 Training projection using pattern silhouettes

When a test sequence $t \in T$ is given for classification, the AC-SC algorithm filters relevant training sequences $x \in D$ containing subsequences approximately matching sequence t, in order to learn a classifier C_t. In this case, a relevant sequence in D must share at least one pattern silhouette with t. In order to find approximate matching subsequences in D, when a test sequence t is given, the AC-SC algorithm enumerates all eligible pattern silhouettes present in t. This process is illustrated in Fig. 2. The same subsequence length limitation ($m = \log n$) is imposed in this process, which ensures $O(n \log n)$ learning cost for a given test sequence t containing n elements.

For each possible silhouette in t, the algorithm projects/filters the training set D in order to consider only subsequences having that shape. This filtering process is nothing

Fig. 2 Pattern silhouettes

A C T G G C T T	Silhouette
A C T G C G T T	$(A, A, 1)$
A C T G C G T T	$(C, C, 1)$
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮	⋮
A C T G C G T T	$(T, T, 1)$
A C T G C G T T	$(A, C, 2)$
A C T G C G T T	$(C, T, 2)$
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮	⋮
A C T G C G T T	$(T, T, 2)$
A T G C G T T	$(A, T, 3)$
A C G C G T T	$(C, G, 3)$
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮	⋮
A C T G C G T	$(G, T, 3)$
⋮ ⋮ ⋮	⋮
A T T	$(A, T, 7)$
A C T	$(C, T, 7)$
A T	$(A, T, 8)$

more than a simple lookup at the inverted index previously constructed. At this point, the AC-SC algorithm works using the following information: a test sequence t, its corresponding pattern silhouettes (and the position where they occur in t), and all subsequences in D that approximately match t. The algorithm now advances to the following step, as discussed next.

3.6 Calculating class membership

Calculating class membership involves assessing how similar are a test sequence t and its approximately-matching subsequences in D (relatively to a given silhouette). This process iterates calculating the similarity between each sequence $t \in T$ and the corresponding approximately-matching subsequences in D, as detailed next.

3.7 Silhouette alignment

Given an arbitrary sequence $t \in T$, the objective of silhouette alignment is to find, for each approximately-matching training sequence $x \in D$, the position in which the target silhouette s occurs. This is done by traversing the elements in x, looking for the first element matching the leftmost element in silhouette s. If element $a_j \in x$ matches the leftmost element in s, then we simply check if element $a_{j+k} \in x$ also matches the rightmost element in s, where k is the length of s. If both matches succeed, we have an alignment between sequences t and x, and, since we already know where the silhouette

occurs in t, we can now measure how similar are these sequences. Specifically, we measure the similarity between a pair of sequences (t, x) by exploiting the intuition that there exists valuable information in the proximity and contiguity among elements in t and x.

3.8 Assessing similarity between sequences

We propose a novel similarity measure, which is given by a function we call *Contiguity-Based Similarity Function*. This function expresses the similarity between a test sequence t and approximately-matching subsequences in D, given a specific pattern silhouette. Such measure consists of pairwise comparing the elements within sequences $t \in T$ and $x \in D$, emphasizing consecutive matches. It is formulated as a function with the following properties: (i) it is monotonically increasing, (ii) it memorizes previous matches, (iii) consecutive matches make it increase fast, (iv) consecutive mismatches make it constant, and (v) isolated mismatches only delay its increase. This similarity function is better defined next.

Definition 3 (*Contiguity-Based Similarity Function*) Consider two sequences $t \in T$ and $x \in D$, both containing a pattern silhouette $s = (l, r, k)$. Let i denote the starting position where s occurs in t, and j the starting position where s occurs in x. The similarity between t and x, regarding to s, is given by Eq. 5:

$$\Gamma(t, x|s) = \sum_{c=0}^{k-1} \gamma(t_{i+c}, x_{j+c}) \tag{5}$$

where γ function is defined as:

$$\gamma(t_p, x_q) = \begin{cases} \lambda & \text{if } t_p = x_q \text{ and } p = i \\ \lambda \times \gamma(t_{p-1}, x_{q-1}) & \text{if } t_p = x_q \text{ and } p \neq i \\ \frac{1}{\lambda^2} \times \gamma(t_{p-1}, x_{q-1}) & \text{otherwise} \end{cases} \tag{6}$$

The γ function is a pairwise comparison function, which is applied for each aligned pair of elements in test and training sequences. When a matching is found, the function returns a value which is λ times greater than the value returned for the previous matching. In other words, the longer the chain of previous successful matches, the larger the value of the current γ execution. On the other hand, when a mismatch happens, the function returns a fraction $\frac{1}{\lambda^2}$ of the previous value (i.e., it is penalized). In summary, consecutive mismatches make the γ function tend fast to zero, but isolated mismatches do not cause such hard impact, making the proposed similarity function well suited for possibly noisy sequences where isolated errors may happen. The sum of each pairwise comparison is accumulated by the Γ function, which will result in a measure of similarity between the two sequences being compared. It is worth noting that multiple silhouettes are considered while assessing the similarity between sequences, increasing the effectiveness of the classification process.
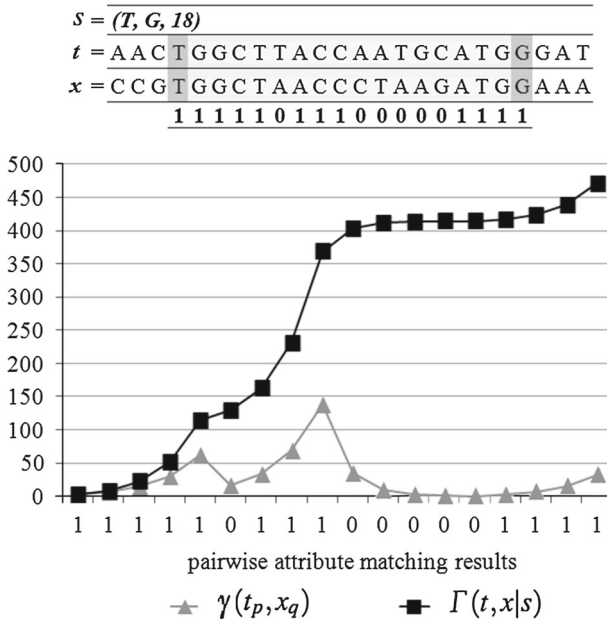
$$S = \overline{(T, G, 18)}$$
$$t = \text{AAC} \boxed{\text{T}} \text{GGCTTACCAATGCATG} \boxed{\text{G}} \text{GAT}$$
$$x = \text{CCG} \boxed{\text{T}} \text{GGCTAACCCTAAGATG} \boxed{\text{G}} \text{AAA}$$
$$\mathbf{1\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1}$$

pairwise attribute matching results

$\gamma(t_p, x_q)$ $\Gamma(t, x|s)$

Fig. 3 γ and Γ functions, for $\lambda = 2$

As the γ function is the core of the AC-SC algorithm, being executed repeated times for each test sequence, it must be as simple and fast as possible. We decided to build it using power-of-two operations (i.e., $\lambda = 2$), for computational efficiency. Figure 3 illustrates the behavior of γ and Γ functions when applied to a hypothetical pair of sequences and a given silhouette of length 18. Zeroes represent mismatches. We can see the difference in impact between an isolated mismatch (at 6th position) and consecutive mismatches (from 10th to 14th positions).

3.9 Prediction

Multiple pattern silhouettes are investigated while processing an arbitrary sequence $t \in T$. The similarity values between t and each silhouette are grouped together according to the class label associated with the corresponding training sequence. We calculate a score for each class label y_i as follows:

$$w(t, y_i) = \sum_{s \in S_t} \sum_{x \in D_s^{y_i}} \Gamma(x, t|s) \tag{7}$$

where S_t is the set of all pattern silhouettes matching test sequence t, and $D_s^{y_i}$ is the set of all training sequences labeled as y_i containing a given silhouette s. Finally, these scores are normalized, as expressed by the prediction function $\hat{p}(y_i|t)$, previously shown in Eq. 4.

4 Experimental evaluation

In this section we empirically analyze the effectiveness of our proposed algorithms for the sake of learning sequential classifiers. Similarly to other works (Zaki et al. 2010), we employ the standard accuracy as our basic evaluation measure. Learning times for all algorithms include the time spent during any pre-processing, training and testing operations, and are given in milliseconds, and all experiments were performed on a Linux-based PC with a Intel core i5 2.4 GHz processor and 4.0 GBytes RAM.

4.1 Application scenarios

We employ diverse application scenarios in order to evaluate our algorithms under different aspects of sequence data, such as short- and long-range dependence, and different levels of noise. Application scenarios used in our experiments, include:

1. Sentiment analysis: this task aims at determining the attitude that is implicit in a textual sentence, with respect to some topic or content. The attitude is usually represented by judgment or evaluation concerning the topic. The dataset we used in the experiments comprises messages posted on Twitter concerning the Brazilian presidential election, occurred in 2010. We collected over 62,000 messages regarding candidate Dilma Rousseff during the campaign. These messages were manually annotated by three to five human annotators. Thus, we believe the dataset corresponds to a good sample of the population sentiment of approval during the election period. The dataset contains 62,089 distinct terms. This dataset is very noisy, mainly because the shortened format of communication imposed by Twitter, which forces users to post messages in informal and sometimes criptic style.
2. Name disambiguation: given a citation record with ambiguous author names, determine the correct entity corresponding to that name. The dataset we used in the experiments is composed of authorship records extracted from the DBLP digital library. Each record in the dataset comprises co-author names, title and citations, and is associated with at least one ambiguous author name. There are 2,193 distinct terms in the dataset.
3. Protein fold recognition: this task aims at predicting the structure (or fold) of a protein from its aminoacid sequence. The dataset we used in the experiments is composed of aminoacid sequences collected from the Protein Data Bank archive (www.pdb.org), which contains experimentally determined structures of proteins. The dataset contains 694 sequences, each one having up to 967 elements.
4. Protein family recognition: given a collection of aminoacid sequences belonging to different protein families, determine whether a query protein belongs to a given family or not. The dataset we used in the experiments is composed of long string of characters, where each character represents an aminoacid from a set of 20 possible ones. The dataset includes a curated classification of known protein structures with the secondary structure knowledge embedded in the dataset (Murzin et al. 1995). This task has being largely employed in many applications of biological sequence analysis for finding homologous proteins (Durbin et al. 1998).

Table 1 Improvement in terms of accuracy. Baseline is $\lambda = 2$

Dataset	$\lambda = 3\ (\%)$	$\lambda = 4\ (\%)$	$\lambda \to \infty\ (\%)$
Sentiment analysis	0.1	−0.1	−0.1
Author name disamb.	−0.2	−0.2	−0.3
Protein fold recog.	–	−1.2	−1.9
Protein family recog.	−0.2	−0.1	0.2
Intrusion detection	0.1	−0.1	−0.5
Spelling correction	0.2	–	−0.2
Web log analysis	–	−0.1	−0.1

5. Intrusion detection: given a sequence of UNIX commands performed by an arbitrary user, determine if the user is a masquerader or an authentic one. The dataset we used in the experiments was collected from Purdue University (Lane and Brodley 1999), over varying periods of time, using the (t)csh mechanism. Each command in the history data together with its arguments is treated as a single token. Modeling this dataset requires classifiers composed of long subsequences.

6. Spelling correction: given a sentence containing commonly confused words (Golding and Roth 1996), determine if the target word is correctly or wrongly spelled. The total number of sentences in our dataset is 2,917 and there are 12,280 distinct terms.

7. Web log analysis: given a sequence of clicks performed by an arbitrary user, categorize the user based on his/her navigation behavior. The dataset we used in the experiments is composed of log files collected at the Department of Computer Science at the Rensselaer Polytechnic Institute during a period of 3 weeks. Those files were transformed into sequences of clicks (web navigation history) made by different users. Each sequence represents a web session of a specific user and is labeled according to the origin domain of that user. Users coming from "edu" or "ac" domains are taken as academic and users coming from other domains are taken as visitors. In all, the dataset contains 16,206 unique Web pages, which make up the alphabet.

More detailed descriptions of these tasks and datasets are available in (Zaki et al. 2010; Davis et al. 2012; Silva et al. 2011).

4.2 Sensitivity to λ

Table 1 shows the sensitivity of AC-SC algorithm to variations of λ. Specifically, the table shows the improvement in terms of accuracy in comparison to results obtained by setting $\lambda = 2$. The best value for λ varies according to the dataset. For most of the datasets, lower values for λ imply in higher improvements. But in all cases, these improvements are very small, showing that AC-SC is extreemely robust to variations of λ.

4.3 Accuracy versus learning time

Figure 4 shows the trade-off between accuracy and learning time, considering our proposed AC-SC algorithm.[1] Both dimensions are impacted by the length of the subsequences that compose the classifier. We evaluated six different length limitations: $\log_{10} n$, $\frac{n}{10}$, $\log n$, $\sqrt{n}$, $\frac{n}{2}$, and n. Clearly, the larger the length of the subsequences, the more time is spent for learning the corresponding classifier. Accuracy, on the other hand, does not necessarily increases with the length of subsequences. The best trade-off between accuracy and learning time seems to be achieved when limiting the length of the subsequences by a factor of $\log n$.

4.4 Comparative results

We employ diverse baselines in our comparison analysis:

- A k-order HMM (Galassi et al. 2007; Pitkow and Pirolli 1999; Saul and Jordan 1999; Deshpande and Karypis 2004) as the representative of traditional solutions to model sequence data,
- The ASC algorithm (Bannister 2007), as the representative of the state-of-the-art algorithms devised to tasks related to information extraction and protein fold recognition,
- The VOGUE algorithm (Zaki et al. 2010), as the representative of the state-of-the-art algorithms devised to tasks such as intrusion detection, spelling correction, and Web log analysis,
- The HMMER algorithm (Eddy 1998), as a representative of the state-of-the-art algorithms devised to protein family recognition tasks,
- The String Kernel for LIBSVM (Chang and Lin 2011), as a representative of simple edit-distance string kernel solutions,
- The Gappy Substring Kernel (Rousu and Shawe-Taylor 2005; Lodhi et al. 2002), as a representative of popular mismatch-tolerant string kernel solutions available, and,
- The Spectrum Kernel (Leslie et al. 2002a), as a representative of fast exact-match string kernel solutions available,
- The LAC algorithm (Veloso and Meira 2011), as a representative state-of-the-art demand-driven associative classifiers.[2]

The learning task in all application scenarios consists of correctly predicting the class label associated with test sequences. We conducted ten-fold cross validation, and the results reported for each evaluated algorithm correspond to the average of the ten trials. Wilcoxon significance tests were performed, and the best results, including statistical ties, are shown in bold. String kernel SVMs were built with hyperparameters found through a grid search approach. The computation time of the grid search task was not considered in the results.

---

[1] A similar trend is observed for SC-SC algorithm.

[2] The reason we include the LAC algorithm (which does not exploit adjacency information) as a baseline, is to evaluate the possible benefits of exploiting adjacency information while producing the classifier.
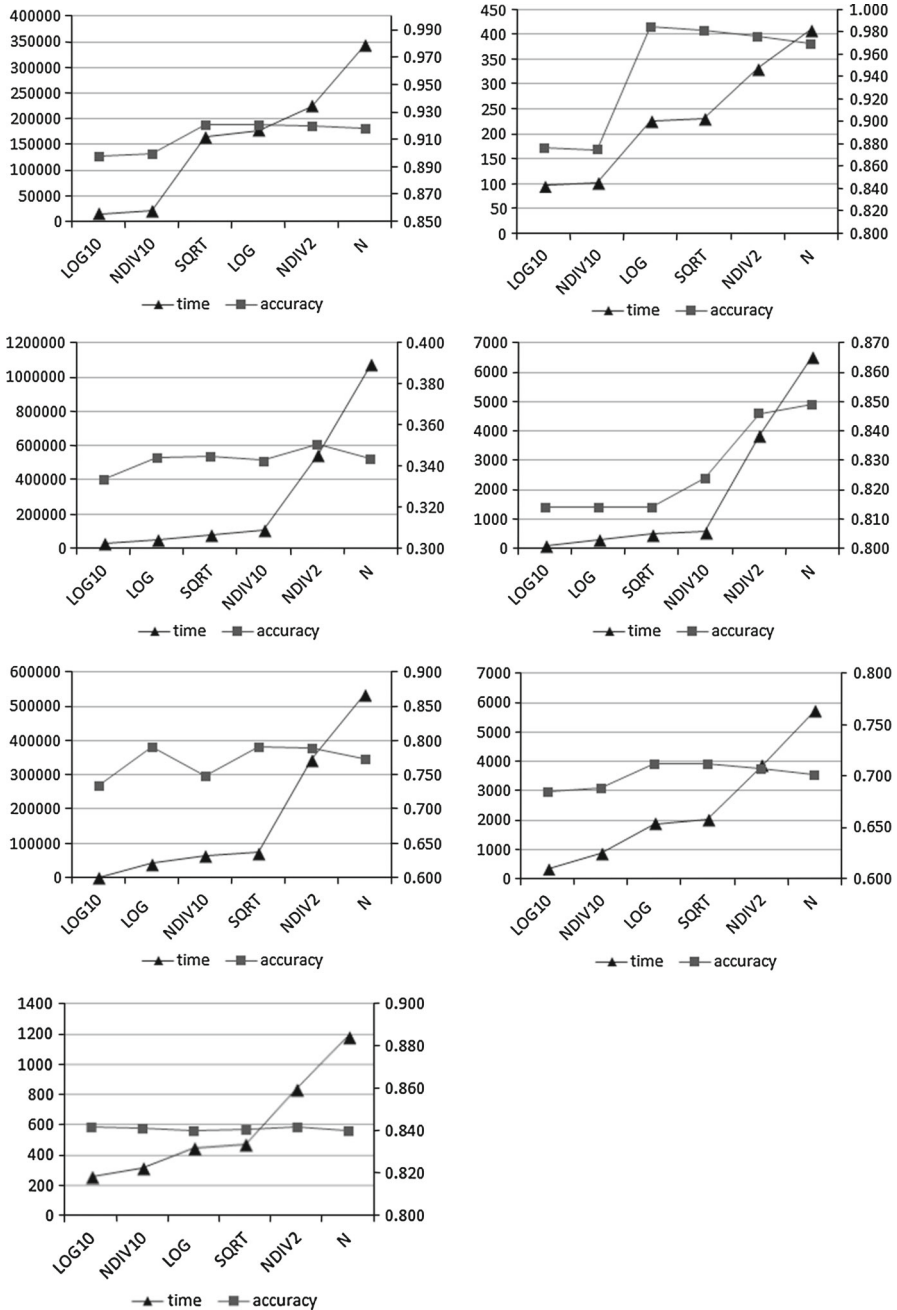
Fig. 4 Trade-off between accuracy (*right* y-axis), time (*left* y-axis) and sequence length limitation (*x*-axis). Datasets employed (in order): Sentiment analysis, author name disambiguation, protein fold recognition, protein family recognition, intrusion detection, spelling correction and Web log analysis

Table 2 Sentiment analysis

Algorithm	Max length	Time	Accuracy
LAC	1	18,487	0.781
	2	343,155	0.912
	3	2,531,995	0.930
	4	9,989,354	**0.935**
Edit-Distance Kernel	–	10,598,900	0.854
Gappy Kernel	–	15,811,121	0.892
Spectrum Kernel	–	618,274	0.862
SC-SC	$\log n$	144,838	0.906
AC-SC	$\log n$	185,785	**0.921**

Table 2 shows accuracy numbers and the corresponding execution times for datasets concerning sentiment analysis. Table 3 shows accuracy and time results obtained in author name disambiguation task. Also, Table 3 shows accuracy and time for the protein fold recognition task. For these datasets, we used LAC, String Kernel SVMs and ASC algorithms as baselines. We performed an extensive evaluation by analyzing different parameter configurations, namely maximum sequence length, minimum support, and maximum gap size. In most cases, lower minimum support values yields best accuracy figures, but this usually increases learning time, as expected. In many cases, accuracy numbers tend to increase when larger gaps are allowed, but again, this also increases learning time. The length of the enumerated subsequences is important for the sake of improving classification accuracy, but the cost of exploring longer subsequences may become impractical, specially when combined with the gap strategy. In summary, higher accuracy figures are usually achieved by exploiting parameter configurations that lead to higher learning times. This is clearly observed, even in domains where short patterns (comprising 2 to 3 elements) are expected to provide good results, as in sentiment analysis and author name disambiguation, which are related to language and text information retrieval. Our proposed algorithms, in most cases, offer either the fastest learning times or the highest accuracy numbers. More specifically, the SC-SC algorithm was the second fastest one in all evaluated cases, being surpassed only by LAC when limited to single-element patterns, which is always less accurate. Generally speaking, we may state that the AC-SC algorithm offers the same (or very similar) accuracy numbers when compared with LAC algorithm. However, the results obtained by LAC, in most cases, with higher pattern sizes, make its execution much slower than all other approaches. A similar analysis can be done when comparing AC-SC with ASC algorithm: the similar accuracy results come with longer patterns and gaps, which produce slower executions. We conclude that the AC-SC algorithm compensates its smaller number of investigated patterns by exploring longer subsequences and, specially, by allowing approximate matches and mismatches. This conclusion becomes clearer when comparing the results obtained by AC-SC against the ones obtained by SC-SC, which, in most of the cases, could not achieve the same accuracy level. The ASC algorithm, unfortunately, could not be applied in the sentiment analysis dataset. Its high-dimensionality, combined with the large alphabet size, make ASC to consume unpractical amounts of memory. When comparing AC-SC with

Table 3 Author name disambiguation and protein fold recognition

Algorithm	Len.	Supp.	Gap	Name disambiguation		Prot. fold recognition	
				Time	Accuracy	Time	Accuracy
ASC	1	0.001	1	565	0.815	28,745	0.046
	1	0.005	1	558	0.798	21,294	0.043
	1	0.010	1	484	0.823	19,489	0.046
	2	0.001	1	992	0.931	54,896	0.301
	2	0.001	2	1,207	0.978	64,629	0.308
	2	0.001	3	1,583	0.969	74,983	0.311
	2	0.005	1	1,053	0.939	33,515	0.299
	2	0.005	2	1,367	0.972	38,892	0.298
	2	0.005	3	1,490	0.968	47,003	0.319
	2	0.010	1	564	0.972	22,894	0.049
	2	0.010	2	742	0.966	23,345	0.051
	2	0.010	3	755	0.977	23,501	0.072
	3	0.001	1	1,329	0.971	72,328	0.301
	3	0.001	2	2,652	0.967	143,993	0.309
	3	0.001	3	4,623	0.959	269,732	0.292
	3	0.005	1	1,330	0.970	33,104	0.297
	3	0.005	2	2,723	0.967	38,731	0.325
	3	0.005	3	4,676	0.963	44,226	0.329
	3	0.010	1	742	**0.986**	22,947	0.050
	3	0.010	2	1,020	0.975	23,444	0.055
	3	0.010	3	1,441	0.962	24,668	0.074
	4	0.001	1	1,823	0.973	100,952	0.318
	4	0.001	2	1,823	0.973	388,748	0.294
	4	0.001	3	6,295	0.967	1,146,505	0.261
	4	0.005	1	1,903	0.977	33,263	0.313
	4	0.005	2	6,268	0.969	40,466	0.334
	4	0.005	3	15,278	0.937	44,458	0.325
	4	0.010	1	768	0.982	23,362	0.049
	4	0.010	2	1,627	0.976	23,254	0.050
	4	0.010	3	2,914	0.962	23,624	0.071
LAC	1	–	–	47	0.777	8,816	0.046
	2	–	–	62	0.923	37,220	0.300
	3	–	–	187	0.955	3,511,546	0.343
Edit-Distance Kernel	–	–	–	5,605	0.903	15,100	**0.373**
Gappy Kernel	–	–	–	7,110	0.922	18,167	**0.371**
Spectrum Kernel	–	–	–	856	0.910	14,274	**0.371**
SC-SC	$\log n$	–	–	92	0.953	8,758	0.311
AC-SC	$\log n$	–	–	248	**0.983**	18,409	0.347

Table 4 Protein family recognition

Algorithm	Length	Time	Accuracy
HMM	–	610	0.725
HMMER	–	190	0.750
VOGUE	–	3,650	0.775
Edit-Dist. Krn.	–	224	**0.844**
Gappy Krn.	–	301	**0.842**
Spectrum Krn.	–	202	0.818
SC-SC	$\log n$	153	0.814
AC-SC	$\log n$	281	**0.840**

Table 5 Intrusion detection

Algorithm	Length	Time	Accuracy
HMM	–	64,930	0.715
HMMER	–	41,328	0.725
VOGUE	–	14,420	0.732
Edit-Dist. Krn.	–	161,660	0.487
Gappy Krn.	–	193,745	0.721
Spectrum Krn.	–	17,182	0.781
SC-SC	$\log n$	3,870	0.767
AC-SC	$\log n$	73,040	**0.791**

string kernel approaches, our algorithm achieves better or similar accuracy results in sentiment analysis and author name disambiguation tasks, being much faster. Protein fold recognition poses a very hard task for all algorithms considered. No algorithm could reach even 40 % in accuracy. As expected, string kernel algorithms, that correspond to the state-of-the-art in protein fold recognition, achieved the highest accuracy among the evaluated algorithms, and also the best execution times.

Table 4 shows accuracy numbers and the corresponding execution times for the protein family recognition task. Table 5 presents accuracy numbers and time results for the intrusion detection task. Table 7, shows results related to the spelling correction task and, finally, Table 6 shows accuracy numbers and time results for Web log analysis task. For these tasks, we used HMM, HMMER, String Kernels, and the VOGUE algorithms as baselines. Finding the best parameters for these algorithms involved the evaluation of an enormous number of possible configurations, as discussed in (Zaki et al. 2010).

Again, in almost all cases, our proposed algorithms, SC-SC and AC-SC, offer either the fastest learning times or the highest accuracy numbers. More specifically, the SC-SC algorithm was the fastest one in all evaluated cases. Further, for the Web Log analysis task, SC-SC was also the best performer in terms of classification accuracy, and for the remaining tasks, it also provides highly competitive accuracy figures, being, in most of the cases, higher than the accuracy numbers achieved by HMMER, HMM,

Table 6 Spelling correction

Algorithm	Length	Time	Accuracy
HMM	–	26,450	0.624
VOGUE	–	44,630	0.685
Edit-Dist. Krn.	–	1,602	0.581
Gappy Krn.	–	2,201	0.620
Spectrum Krn.	–	551	0.649
SC-SC	$\log n$	421	0.654
AC-SC	$\log n$	2,079	**0.715**

Table 7 Web log analysis

Algorithm	Length	Time	Accuracy
HMM	–	1,200,720	0.798
VOGUE	–	77,590	0.810
Edit-Dist. Krn.	–	269,860	**0.835**
Gappy Krn.	–	310,195	**0.841**
Spectrum Krn.	–	41,281	**0.836**
SC-SC	$\log n$	2,250	**0.842**
AC-SC	$\log n$	3,510	**0.841**

String Kernels and VOGUE. AC-SC achieves the highest accuracy numbers in all cases. AC-SC also provides fast execution times in all cases, when compared with HMMs and VOGUE approaches. String kernel SVMs were fast in some tasks, but extremely slow in others. In the Web log analysis and intrusion detection tasks, where the datasets are large, String kernel SVMs executed orders of magnitude slower than SC-SC and AC-SC algorithms. In the other two smaller datasets the time is in the same order of magnitude, being faster in only one case (i.e., the spelling correction task). Yet, in the intrusion detection task, AC-SC is slower than HMMs, but it showed accuracy gains that are greater than 10 %.

4.5 Discussion

Our results indicate that for most of the evaluated cases, our algorithms provide competitive (or better) accuracy numbers, but are much faster than the competitors. Several reasons contribute to the compuational efficiency of our algorithms, including: (i) the limitation on the length of subsequences, (ii) the demand-driven subsequence enumeration process, and (iii) the fast computation of pattern silhouettes. We also observed some reasons that may contribute to the effectiveness of our algorithms: (i) higher accuracy improvements were observed in application scenarios where data is highly noisy (sentiment analysis, name disambiguation, and spelling correction), and (ii) high accuracy numbers were observed in application scenarios where longer subsequences must be enumerated (intrusion detection, web log analysis, and protein fold recognition). This suggests that the proposed algorithms produce noise-tolerant classifiers that are able to enumerate long subsequences efficiently.

5 Conclusions

This paper focuses on the important problem of learning classifiers from sequence data. Such classifiers are used to distinguish between sequences belonging to different labeled groups. We have introduced new algorithms for learning these classifiers. Our proposed algorithms produce classifiers that are able to capture long-range dependencies in the data, while being tolerant to noisy data. The first, simpler algorithm, exploits proximity information to drastically narrow down the search space for sequences. More specifically, elements within a sequence must be contiguous in the sense that no violation in the ordering among elements is allowed. Sequence enumeration follows a simple sliding window approach, which warrants asymptotic complexity of $O(n \log n)$ with the length of the test instances.

The second, more sophisticate algorithm, also exploits feature proximity in order to narrow down the search space for rules, but in this case violations or disruptions in the ordering between features within a rule are allowed to happen. We introduce a novel structure, called pattern silhouettes, in order to warrant effective rule enumeration with partial contiguous matchings.

To evaluate the effectiveness of our algorithms, we use real data obtained from applications scenarios such as sentiment analysis, author name disambiguation, intrusion detection, spelling correction, protein fold and family recognition and Web log analysis. Our results reveal that the proposed algorithms are highly effective and efficient, being able to produce accurate results with low computational cost, when compared against state-of-the-art associative classifiers, general purpose sequential classifiers and even domain-specific sequential classifiers. In some cases our algorithms are orders of magnitude faster to achieve the same or even better accuracy.

References

Agrawal R, Srikant R (1995) Mining sequential patterns. In: ICDE, pp 3–14

Bannister W (2007) Associative and sequential classification with adaptive constrained regression methods. PhD thesis, Tempe, AZ, USA

Baum L, Petrie T (1966) Statistical inference for probabilistic functions of finite state Markov chains. Ann Math Stat 37(6):1554–1563

Baum L, Petrie T, Soules G, Weiss N (1970) A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. Ann Math Stat 41:164

Bicego M, Murino V, Figueiredo M (2003a) A sequential pruning strategy for the selection of the number of states in hidden Markov models. Pattern Recognit Lett 24(9–10):1395–1407

Bicego M, Murino V, Figueiredo M (2004) Similarity-based classification of sequences using hidden Markov models. Pattern Recognit 37(12):2281–2291

Bicego M, Murino V, Pelillo M, Torsello A (2006) Similarity-based pattern recognition. Pattern Recognit 39(10):1813–1814

Bühlmann P, Wyner A (1999) Variable length Markov chains. Ann Stat 27(2):480–513

Chang CC, Lin CJ (2011) LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol 2:27:1–27:27, software available at http://www.csie.ntu.edu.tw/cjlin/libsvm

Davis A, Veloso A, da Silva A, Laender A, Meira W Jr (2012) Named entity disambiguation in streaming data. In: ACL, pp 815–824

Deshpande M, Karypis G (2004) Selective Markov models for predicting web page accesses. ACM Trans Internet Technol 4(2):163–184

Durbin R, Eddy AKS, Mitchison G (1998) Biological sequence analysis. Cambridge University Press

Durbin R, Eddy S, Krogh A, Mitchison G (1998) Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press

Du Preez J (1998) Efficient training of high-order hidden Markov models using first-order representations. Comput Speech Lang 12(1):23–39

Eddy S (1998) Profile hidden Markov models. Bioinformatics 14(9):755–763

Galassi U, Giordana A, Saitta L (2007) Incremental construction of structured hidden Markov models. In: IJCAI, pp 798–803

Golding A, Roth D (1996) Applying winnow to context-sensitive spelling correction. CoRR

Han H, Giles C, Zha H, Li C, Tsioutsiouliklis K (2004) Two supervised learning approaches for name disambiguation in author citations. In: JCDL, pp 296–305

Han H, Zha H, Giles C (2005) Name disambiguation in author citations using a k-way spectral clustering method. In: JCDL, pp 334–343

Haussler D (1999) Convolution kernels on discrete structures. Tech. rep,Technical report, UC Santa Cruz

Hu J, Brown M, Turin W (1996) Hmm based on-line handwriting recognition. IEEE Trans Pattern Anal Mach Intell 18(10):1039–1045

Hughey R, Krogh A (1996) Hidden Markov models for sequence analysis: extension and analysis of the basic method. Comput Appl Biosci 12(2):95–107

Kriouile A, Mari J, Haon J (1990) Some improvements in speech recognition algorithms based on HMM. In: ICASSP, pp 545–548

Kuksa P, Huang PH, Pavlovic V (2008) A fast, large-scale learning method for protein sequence classification. In: 8th international workshop on data mining in bioinformatics, pp 29–37

Lane T, Brodley C (1999) Temporal sequence learning and data reduction for anomaly detection. ACM Trans Inf Syst Secur 2(3):295–331

Law H, Chan C (1996) N-th order ergodic multigram hmm for modeling of languages without marked word boundaries. In: COLING, pp 204–209

Lesh N, Zaki M, Ogihara M (1999) Mining features for sequence classification. In: KDD, pp 342–346

Leslie C, Kuang R (2003) Fast kernels for inexact string matching. In: COLT, pp 114–128

Leslie C, Kuang R (2004) Fast string kernels using inexact matching for protein sequences. J Mach Learn Res 5:1435–1455

Leslie C, Eskin E, Noble WS (2002a) The spectrum kernel: a string kernel for SVM protein classification. Pac Symp Biocomput 7:566–575

Leslie C, Eskin E, Weston J, Noble W (2002b) Mismatch string kernels for SVM protein classification. In: NIPS, pp 1417–1424

Leslie C, Eskin E, Cohen A, Weston J, Noble W (2004) Mismatch string kernels for discriminative protein classification. Bioinformatics 20(4):467–476

Lin M, Hsueh S, Chen M, Hsu H (2009) Mining sequential patterns for image classification in ubiquitous multimedia systems. In: IIH-MSP, pp 303–306

Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002) Text classification using string kernels. J Mach Learn Res 2:419–444

Lodhi H, Muggleton S, Sternberg M (2009) Multi-class protein fold recognition using large margin logic based divide and conquer learning. SIGKDD Explor 11(2):117–122

Malik H, Kender J (2008) Classifying high-dimensional text and web data using very short patterns. In: ICDM, pp 923–928

Müller S, Eickeler S, Rigoll G (2000) Crane gesture recognition using pseudo 3-d hidden Markov models. In: FG (Conf. on Automatic Face and Gesture Recognition), pp 398–402

Murzin A, Brenner S, Hubbard T, Chothia C (1995) SCOP: a structural classification of proteins database for the investigation of sequences and structures. J Mol Biol 247(4):536–540

Pitkow J, Pirolli P (1999) Mining longest repeating subsequences to predict world wide web surfing. In: USENIX symposium on Internet technologies and systems

Rabiner L (1989) A tutorial on hidden Markov models and selected applications in speech recognition. Proc IEEE 77(2):257–286

Rieck K, Laskov P (2008) Linear-time computation of similarity measures for sequential data. J Mach Learn Res 9:23–48

Rousu J, Shawe-Taylor J (2005) Efficient computation of gapped substring kernels on large alphabets. J Mach Learn Res 6(1):1323–1344

Saul L, Jordan M (1999) Mixed memory Markov models: decomposing complex stochastic processes as mixtures of simpler ones. Mach Learn 37(1):75–87

Schwardt L, Preez JD (2000) Efficient mixed-order hidden Markov model inference. In: ICSLP, pp 238–241

Sha F, Saul L (2006) Large margin hidden Markov models for automatic speech recognition. In: NIPS, pp 1249–1256

Silva I, Gomide J, Veloso A, Meira Jr W, Ferreira R (2011) Effective sentiment stream analysis with self-augmenting training and demand-driven projection. In: SIGIR, pp 475–484

Srikant R, Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In: EDBT, pp 3–17

Srivatsan L, Sastry P, Unnikrishnan K (2005) Discovering frequent episodes and learning hidden Markov models: a formal connection. IEEE TKDE 17:1505–1517

Syed Z, Indyk P, Guttag J (2009) Learning approximate sequential patterns for classification. J Mach Learn Res 10:1913–1936

Szymanski B (2004) Recursive data mining for masquerade detection and author identification. Workshop on Information Assurance, pp 424–431

Tseng V, Lee C (2005) CBS: a new classification method by using sequential patterns. In: SDM

Vapnik V (1979) Estimation of dependences based on empirical data (in Russian). Nauka

Veloso A, Meira W Jr (2011) Demand-driven associative classification. Springer

Wang Y, Zhou L, Feng J, Wang J, Liu Z (2006) Mining complex time-series data by learning Markovian models. In: ICDM, pp 1136–1140

Watkins C (1999) Dynamic alignment kernels. Advances in neural information processing systems, pp 39–50

Ye L, Keogh E (2009) Time series shapelets: a new primitive for data mining. In: KDD, pp 947–956

Ye L, Keogh E (2011) Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Min Knowl Discov 22(1–2):149–182

Zaki M (2000) Sequence mining in categorical domains: Incorporating constraints. In: CIKM, pp 422–429

Zaki M, Carothers C, Szymanski B (2010) Vogue: a variable order hidden Markov model with duration based on frequent sequence mining. TKDD 4(1):1–31