Discovering Useful Compact Sets of Sequential Rules in a Long Sequence

Erwan Bourrand*†, Luis Galárraga‡, Esther Galbrun§, Elisa Fromont*, Alexandre Termier*
* *Univ Rennes, Inria, CNRS, IRISA, 35000 Rennes, France*
† *Advisor_SLA, Inc.*
‡ *Inria, Univ Rennes, CNRS, IRISA, 35000 Rennes, France*
§ *University of Eastern Finland*
{luis.galarraga, elisa.fromont, alexandre.termier}@irisa.fr

***Abstract*—We are interested in understanding the underlying generation process for long sequences of symbolic events. To do so, we propose COSSU, an algorithm to mine small and meaningful sets of sequential rules. The rules are selected using an MDL-inspired criterion that favors compactness and relies on a novel rule-based encoding scheme for sequences. Our evaluation shows that COSSU can successfully retrieve relevant sets of closed sequential rules from a long sequence. Such rules constitute an interpretable model that exhibits competitive accuracy for the tasks of next-element prediction and classification.**

I. INTRODUCTION

Long sequences of symbols are ubiquitous. Examples include DNA sequences, server logs, traces of network packets, and even long texts. Discovering regularities in such sequences allows for a better understanding of the sequence generation process, and can be useful e.g., for diagnostic and prediction.

Some models, such as LSTMs [1], excel at detecting those regularities and use them for accurate prediction. On the downside, such models are hardly understandable by human users. Models based on pattern mining, in contrast, provide high-level human-readable descriptions of the underlying structure of the data. However, this interpretability typically comes at the cost of a restricted predictive power. We focus on the latter category of models, since we are interested in prediction but also in knowledge discovery.

In the realm of pattern mining, *sequential rules* are a well-established model for understanding and predicting sequential data. Sequential rules take the form $A \Rightarrow C$, where both the antecedent A and the consequent C are sequences of items, e.g., events, nucleotides, or words. A *confidence* score is often attached to the rule to measure how likely it is to observe C after A in the sequence. For instance, the rule $A_1A_2A_3 \Rightarrow$ *PowerFail* with confidence 80% tell us that there is a four in five chance of undergoing a power failure after having observed the sequence of alarms A_1, A_2 and A_3. This can be used both for prediction, i.e., to issue an early warning, and for diagnostic, i.e., to identify the link between the alarms and the power failure.

While sequential rule mining has been extensively studied in the literature [2, 3, 4, 5, 6], two major issues remain. First, the vast majority of approaches consider a database of short sequences rather than a long sequence. While a long sequence can always be split into a database, this incurs a loss of information at the boundaries. Second, to the best of our knowledge, no approach for sequential rule mining addresses the so-called *pattern explosion*, the fact that millions of sequential rules may be produced due to the combinatorial search space.

We propose the first sequential rule mining approach that takes as input a long sequence and outputs a compact set of rules. The selection of the rules is based on the principle of Minimum Description Length (MDL), a paradigm that has proved effective for model selection in pattern mining [7, 8]. MDL approaches rely on an encoding scheme. Ours is presented in Section IV, along with our algorithm to mine rules, in Section V. Experiments, presented in Section VI, demonstrate the relevance of the rules found, as well as their predictive power for the tasks of next-element prediction, and classification for long sequences.

II. BACKGROUND

Our work stands at the crossroads of sequential and MDL-based pattern mining. Below we give brief overviews of these two fields, in turn.

Sequential Pattern Mining. Mining sequential patterns on data is a well-studied problem that has given rise to a plethora of methods accounting for different notions of patterns. The vast majority of approaches assume a database of sequences as input, where sequences are ordered collections of itemsets. We refer the reader to the survey in [2] for a detailed account.

Among the most recent works, Fumarola et al. [5] mine frequent closed sequences, i.e. sequences of maximal length such that any extension will inevitably have lower support. Closed sequences can alleviate pattern explosion to some extent as they are a subset of frequent sequences. Other approaches [4, 6, 9] mine sequential rules of the form $A \Rightarrow C$ where A and C may be itemsets [6, 9] or sub-sequences [4] with the constraint that C occurs after A. Itemsets are generally easier to mine than sequences, however they are less expressive because they do not account for items' order of appearance. Still, none of these methods is directly portable to long sequences as they all assume the data has been partitioned into small sequences (and hence often rely on a different definition of support). A few methods [3, 10] can natively mine

rules on long sequences. Unfortunately, such methods are not resilient to pattern explosion and are limited to association rules between itemsets, thus they do not capture the order between items within the antecedent and the consequent.

Mining patterns with an MDL criterion. The principle of Minimum Description Length (MDL) [11] is a criterion rooted in information theory that stipulates that the best model for a dataset is the model that compresses it best. For a dataset D and a family of models $\mathcal{H}$, the best model for D, according to the (two-parts) MDL, is the one that minimizes the description length

$$H^* = \arg\min_{H \in \mathcal{H}} \mathrm{L}(H) + \mathrm{L}(D|H)\,, \tag{1}$$

where L denotes code length in bits. Equation (1) strikes a balance between model complexity, as measured by $\mathrm{L}(H)$, and fitness to the data, as measured by $\mathrm{L}(D|H)$. When applying MDL to pattern mining, a model is a collection of patterns, e.g., itemsets [7], sequences [8] or, in our case, sequential rules. One of the main ingredients of any MDL-inspired approach is the *encoding scheme*, namely, the protocol to encode the data with the patterns and to encode the patterns themselves. Once an encoding mechanism is in place, we can generate candidate sets of patterns, calculate the corresponding code lengths, and select the set H^* resulting in the shortest description length.

KRIMP [7] was one of the pioneering efforts to apply MDL to pattern mining, more specifically to itemset mining. Given a (large) collection of itemsets mined from the data, KRIMP selects a small representative subset. The selected itemsets were also empirically shown to be effective for classification. Recently, Fischer and Vreeken [12] proposed an approach to mine compact sets of rules from data without the sequential dimension. The approaches proposed in [13, 14] extract sequences with gaps (but not rules) from univariate and multivariate long sequences. Unlike KRIMP [7], they combine the generation and selection of candidate patterns, which boosts efficiency and pattern quality. Shokoohi-Yekta et al. [15] draw inspiration from the MDL principle to mine sequential rules from time series discretized into an integer domain. A mining approach tailored for the streaming setting is proposed in [16], but its compliance with the MDL principle is debatable.

III. DEFINITIONS AND NOTATION

A *sequence* S of length n over an alphabet Σ is an ordered collection of n occurrences of symbols from Σ. We denote by $S[i]$ the i-th element in S ($1 \leq i \leq n$), and let $S[i,j]$ denote the contiguous sequence $\langle S[i] \dots S[j] \rangle$ when $1 \leq i \leq j \leq n$ or the empty sequence $\emptyset$ otherwise. For simplicity, we write the sequence $\langle \sigma_1, \sigma_2 \dots \sigma_k \rangle$ as $\sigma_1\sigma_2\dots\sigma_k$. We denote the concatenation of two sequences by $\oplus$.

We say that sequence S' is a *subsequence* of S (respectively S is a *supersequence* of S'), denoted as $S' \sqsubseteq S$, iff there exist i and j in $[1,n]$ such that $S' = S[i,j]$. Then, we call $S[i,j]$ a *match* of S' in S. More specifically, we denote by $S' \sqsubseteq_i S$ the fact that S' is a subsequence of S with a match starting at position i, i.e. $S' = S[i, i+|S'|-1]$, and by $S' \sqsubseteq^j S$ the fact that S' is a subsequence of S with a match ending at position j, i.e. $S' = S[j-|S'|+1, j]$.

The *support* of a subsequence S' in S, denoted by $\mathrm{supp}_S(S')$, is the number of distinct matches of S' in S, that is, $\mathrm{supp}_S(S') = |\{i \in [1,n] : S' \sqsubseteq_i S\}| = \big|\{j \in [1,n] : S' \sqsubseteq^j S\}\big|$. Finally, a sequence S' is *closed* in S if there is no supersequence $S'' \sqsupset S'$ in S such that $\mathrm{supp}_S(S'') = \mathrm{supp}_S(S')$.

Example 1: Sequences. Given the sequences $S = abceabcadeab$ and $S' = abc$, $S' \sqsubseteq_1 S$, $S' \sqsubseteq_5 S$, $S' \sqsubseteq^3 S$ and $S' \sqsubseteq^7 S$ are all true, hence $\mathrm{supp}_S(S') = 2$.

A *rule* is a pair of sequences (A, C) over alphabet Σ, such that $C \neq \emptyset$. It is denoted as $A \Rightarrow C$. A and C are called the *antecedent* and the *consequent* of the rule, respectively. A rule such that $A = \emptyset$ and $|C| = 1$ is called a *singleton rule*. The set of singleton rules over alphabet Σ is $\Psi_\Sigma = \{\emptyset \Rightarrow \langle\sigma\rangle : \sigma \in \Sigma\}$.

A rule $R : A \Rightarrow C$ *triggers* on sequence S at position i, denoted as $R \vdash_i S$, iff the antecedent has a match in S ending at position i, i.e., iff $A \sqsubseteq^i S$. The rule R *applies* on sequence S at position i, denoted by $R \Vdash_i S$, iff the antecedent has a match in S ending at position i and the consequent has a match in S starting at position $i+1$, i.e. if $A \sqsubseteq^i S$ and $C \sqsubseteq_{i+1} S$. Clearly, $R \Vdash_i S$ implies $R \vdash_i S$.

The *support* of a rule is the number of times it applies in the sequence, whereas its *confidence* is the ratio of the number of times the rule applies to the number of times the rule triggers:

$$\mathrm{supp}_S(R) = |\{i \in [1,n] : R \Vdash_i S\}|$$

and

$$\mathrm{conf}_S(R) = \frac{|\{i \in [1,n] : R \Vdash_i S\}|}{|\{i \in [1,n] : R \vdash_i S\}|}\,.$$

Example 2: Rules. In sequence S from Example 1, rule $R : ab \Rightarrow c$ has support $\mathrm{supp}_S(R) = 2$ and confidence $\mathrm{conf}_S(R) = 2/3$, because R triggers at positions 1, 5 and 11 but only applies at positions 1 and 5.

Given a sequence S up to position m and a rule $R : A \Rightarrow C$, we say that R is *active* at stage j ($0 \leq j < |C|$) and *predicts* $C[j+1]$, if $A \oplus C[1,j] = S[m-|A|-j, m]$. This corresponds to cases where the antecedent of R is followed by the first j elements of its consequent (empty in case $j = 0$) in S. Formally, we define the predicates $\mathrm{active}(R, S, j)$, to indicate that rule R is active on sequence S at stage j, and $\mathrm{predict}(R, S, j)$, to return $C[j+1]$ if $\mathrm{active}(R, S, j)$ is true.

IV. ENCODING SCHEME

To illustrate our encoding scheme, we resort to a sender-receiver metaphor, where the sender first transmits a set of rules and their corresponding weights, followed by a sequence of code words, one for each element of the sequence, in such a way that the receiver can reconstruct the original sequence.

Encoding a Sequence via Sequential Rules. Assume we want to predict the next element after $cabba$, based on two rules $R_1 : abba \Rightarrow b$ and $R_2 : ba \Rightarrow cd$. If the confidence

of R_1 is higher than the confidence of R_2, we would guess that b is more likely than c to appear next. We rely on this simple principle to encode a sequence element by element using a set of sequential rules. Assuming the receiver knows the set of rules and their confidence, as represented by weights, as well as the history of the sequence decoded so far, they can compute a probability distribution over the possible next elements. Hence, the sender can represent the next element as a code word chosen according to this probability distribution. If the set of rules effectively produces a probability distribution concentrated on the element that indeed occurs next, the code word for that element will be short, allowing for a concise representation of the input sequence.

Given a sequence S of m elements seen so far, and a set of weighted rules $(A_R \Rightarrow C_R, w_R)$, such that A_R, C_R, and w_R are respectively the antecedent, the consequent, and the weight of rule R, our goal is to compute the probability of the next element given this set of rules – denoted by $\mathcal{R}$. This amounts to computing a probability distribution over Σ. For each possible next element $\sigma \in \Sigma$, we sum the weights of the active rules that predict σ and divide by the sum of weights of all active rules:

$$P_{\mathcal{R},S}(\sigma) = \frac{\sum_{(R,j)\in\mathcal{A}_{S,\sigma}} w_R}{\sum_{(R,j)\in\mathcal{A}_S} w_R}$$

where $\mathcal{A}_S = \{(R,j), (R,w_R) \in \mathcal{R} \text{ s.t. active}(R,S,j)\}$, and $\mathcal{A}_{S,\sigma} = \{(R,j) \in \mathcal{A}_S \text{ s.t. predict}(R,S,j) = \sigma\}$. A full sequence S of length n is transmitted element by element. At each step m, we encode the next symbol $S[m]$ with a code word chosen according to the probability assigned to it based on $\mathcal{R}$ and the portion of the sequence seen so far, $S[1, m-1]$. Hence, the overall code length for the full sequence is

$$\mathrm{L}(S|\mathcal{R}) = \sum_{m\in[1,n]} -\log_2\left(P_{\mathcal{R},S[1,m-1]}(S[m])\right). \quad (2)$$

The receiver, having the same collection of rules $\mathcal{R}$ and previous elements of the sequence, can perfom the same computation to dynamically reconstruct the code at his end, and decode the transmitted element, using an agreed canonical order on the alphabet to break ties if necessary.

To ensure comparability, the models must achieve *lossless* compression. For this reason, by construction our models always contain the singleton rule set Ψ_Σ. This way, every symbol of the alphabet receives a non-zero probability of occurrence at any step, and can hence be transmitted. In fact, the singleton rule set constitutes the basis of the simplest possible model, what we call the *empty model* and denote by $\mathcal{R}^\emptyset$, since it does not contain any proper rule. To each rule $R_\sigma \in \Psi_\Sigma$ we associate a weight equal to the *background probability* of the predicted element, estimated as its frequency in the sequence, $f_\sigma = |\{i \in [1,n] : S[i] = \sigma\}|/n$.

Encoding the Rules. For the receiver to have access to the rules $\mathcal{R}$, the sender must transmit the antecedents, consequents, and weights at the start of the exchange. The antecedent and consequent are subsequences, hence they can be encoded simply by stating the length of the sequence then listing their elements using codes of length $-\log_2(f_\sigma)$, in order. The weight, on the other hand, is a real number of bounded precision in $(0,1)$. Each weight is written as a finite list of decimals that we encode by applying universal coding to the integer value obtained by listing these decimals in reverse order. Put differently, we encode weight $w = 0.d_1d_2d_3...d_k$ with a code word of length $\mathrm{L}_D(w) = \mathrm{L}_\mathbb{N}(d_k...d_3d_2d_1)$. $\mathrm{L}_\mathbb{N}(z)$ is the length of the code word assigned by the universal code to integer z, which is such that $\mathrm{L}_\mathbb{N}(z) = \log_2^*(z) + \log_2(c_0)$, with c_0 a constant adjusted to ensure the Kraft inequality is satisfied. This penalizes weights by the number of significant digits they contain rather than by their value. Overall, the code length for a rule R is the sum of the lengths of the code words for the antecedent, the consequent and the weight, respectively:

$$\mathrm{L}(R) = \left[\mathrm{L}_\mathbb{N}(|A_R|+1) + \sum_{\sigma\in A_R} -\log_2(f_\sigma)\right] + \left[\mathrm{L}_\mathbb{N}(|C_R|) + \sum_{\sigma\in C_R} -\log_2(f_\sigma)\right] + \mathrm{L}_D(w_R)\,.$$

We encode a rule table by stating the number of rules and then listing them:

$$\mathrm{L}(\mathcal{R}) = \mathrm{L}_\mathbb{N}(|\mathcal{R}|) + \sum_{R\in\mathcal{R}} \mathrm{L}(R)\,. \quad (3)$$

V. The Cossu algorithm

We now introduce an algorithm to mine a set of sequential rules that compresses the input sequence well under the encoding scheme presented in Section IV. Finding such a set of rules is intractable in practice, thus we resort to heuristics.

Our Cossu (COmpact Sets of Sequential rUles) algorithm is outlined in Algorithm 1. Given a sequence S over alphabet Σ as input, Cossu returns a set of rules $\mathcal{R}$ in two phases. First, *rule construction* (lines 1–2) generates a collection of candidate rules $\mathcal{C}$, which are then evaluated during *rule selection* (lines 3–10).

Rule Construction. Cossu starts by extracting closed frequent subsequences by applying an off-the-shelf sequence miner [17] to the input sequence S with a minimum support threshold of 2. Sequential rules are then generated from the sequences by considering all possible partitionings into an antecedent and a non-empty consequent. We use the *compression gain* as a rough estimate of the individual ability of a rule to compress the sequence:

$$\mathrm{gain}(R) = \mathrm{conf}_S(R)\cdot\mathrm{supp}_S(R)\cdot\mathrm{L}(C_R) - (\mathrm{L}(A_R)+\mathrm{L}(C_R))\,. \quad (4)$$

This score puts in balance the potential benefit and cost of adding the rule to the rule set. The benefit depends on how often the consequent is correctly predicted (first term), whereas the cost is the code length of the rule's antecedent and consequent (second term). The latter term constitutes a lower bound on the cost, with a weight yet undetermined. At

Algorithm 1 COSSU: finding a compact set of sequential rules

Require: A long sequence S of elements over an alphabet Σ
Ensure: A compact set of sequential rules $\mathcal{R}$
1: $\mathcal{S} \leftarrow$ MineClosedSequences(S)
2: $\mathcal{C} \leftarrow \{R \in$ PrepareRules$(\mathcal{S}), R \notin \Psi_\Sigma$ and $\text{gain}(R) > 0\}$
3: $\mathcal{R} \leftarrow \mathcal{R}^\emptyset$, with adjusted weights $\hat{\vec{w}} = \arg\min_{\vec{w}} \mathrm{L}(S|\mathcal{R}^\emptyset)$
4: **for** $R \in \mathcal{C}$, ordered by decreasing $\text{gain}(R)$ **do**
5: $\quad \mathcal{R}' \leftarrow \mathcal{R} \cup \{R\}$, with re-adjusted weights $\hat{\vec{w}} = \arg\min_{\vec{w}} \mathrm{L}(S|\mathcal{R}')$
6: $\quad$ **if** $\mathrm{L}(\mathcal{R}', S) < \mathrm{L}(\mathcal{R}, S)$ **then**
7: $\quad\quad \mathcal{R} \leftarrow \mathcal{R}'$
8: $\quad\quad$ **for** $R' \in \mathcal{R} \setminus \mathcal{R}^\emptyset$ **do**
9: $\quad\quad\quad$ **if** $\mathrm{L}(\mathcal{R} \setminus \{R'\}, S) \leq \mathrm{L}(\mathcal{R}, S)$ **then**
10: $\quad\quad\quad\quad \mathcal{R} \leftarrow \mathcal{R} \setminus \{R'\}$
11: **return** $\mathcal{R}$

the end of this first phase, the set of candidates consists of those rules that have a strictly positive compression gain.

Rule Selection. COSSU resorts to a greedy strategy to build the rule set $\mathcal{R}$. Initially, $\mathcal{R}$ consists of the singleton rules (line 3). The algorithm then processes the candidate rules in $\mathcal{C}$ by order of decreasing compression gain, i.e. from most promising to least promising (line 4). The candidate rule is tentatively added to the rule set and the weights are re-adjusted (line 5). If the resulting rule set yields better compression than the current one, the current set is replaced (line 7) and the algorithm goes into a pruning loop. Otherwise, the rule is discarded and the algorithm moves on to the next candidate. The pruning loop (lines 8–10) checks whether the newly added rule makes any of the previously incorporated non-singleton rules obsolete. To do so, it tentatively removes each of the rules in turn and checks whether the compression improves as a result, in which case the rule is permanently removed from $\mathcal{R}$.

Adjusting Rule Weights. Given a set of selected rules, COSSU must determine a suitable vector of weights $\vec{w}$ associated to the rules, so as to optimize the code length of both the model $\mathrm{L}(\mathcal{R})$ (complexity) and the sequence $\mathrm{L}(S|\mathcal{R})$ (fit). Because optimizing both aspects concurrently is very challenging, we assume that the code length of the weights is fixed, by fixing the floating point precision of the weights, and focus on the problem of minimizing $\mathrm{L}(S|\mathcal{R})$, i.e. solving

$$\arg\min_{\vec{w}} \sum_{m \in [1,n]} -\log_2 \left(P_{\mathcal{R}, S[1,m-1]}(S[m]) \right) .$$

We do so greedily, by adjusting the weight of each rule in turn, while keeping the value of the other weights fixed. The minimization resorts to the golden section search algorithm [18], initializing the weights to 1. In the end, we scale all weights to ensure that their values lie in the interval $(0, 1)$. Designing an algorithm that is able to optimize both $\mathrm{L}(\mathcal{R})$ and $\mathrm{L}(S|\mathcal{R})$ is a major direction for future work.

VI. EVALUATION

We evaluate our proposed algorithm on both synthetic and real-world datasets. For real data, we evaluate the rules learned by COSSU by using them as features in two classical machine learning tasks on long sequences: next event prediction and classification. While COSSU is not designed specifically for these tasks, it still yields acceptable performance and reports rules that capture useful patterns in the data. A full account of our experiments is available in an extended version [19] of this paper.

A. Evaluation on synthetic data

To study the behaviour of our algorithm in controlled settings, we generate synthetic random sequences of symbols and inject patterns that match a set of hand-crafted sequential rules. Then, we verify whether COSSU is able to recover the planted rules. We show in [19] that COSSU can distinguish between regularities and background noise in long sequences.

B. Evaluation on a prediction task

Next, we used the rules mined by COSSU to predict the next event in the sequence and showed that the achieved performance competes with interpretable methods for next-element prediction such as bigrams and Hidden Markov Models (HMM).

C. Evaluation on a classification task

In this experiment, we use the following datasets:

Quantified Awesome is a life log[1] that records the daily activities of its owner, an enthusiast of the "Quantified Self" movement, since 2011; we predict whether a sequence of events happened on a weekday or during the weekend;

Presidential debates consists of 555 sentences uttered by Donald Trump and Hillary Clinton in the context of the 2016 US election; the goal is to predict the speaker given a sentence[2];

Newsgroups is a dataset of 180 posts about the topics *electronic* and *religion*[3];

Film critics is a dataset of 1800 movie reviews used for sentiment analysis[4];

Stylometry is a database of 2000 writings from H.P. Lovecraft or E.A. Poe, where the goal is to predict the author based on its writing style[5];

Stylometry (PoS) is the sequence of part-of-speech tags from the previous dataset.

For all datasets (except Quantified Awesome), our alphabet Σ consists of all the words present in the corpus after removing special characters and stop words, and stemming of the remaining words. All datasets were split into training, validation, and test subsets.

Experimental setup. In line with the MDL-based classifier proposed in [7], we can use COSSU to classify sequences as follows: For each class c, we build a *training sequence* by concatenating all the sequences labeled with c. We then extract sets of rules with COSSU on the resulting long sequence.

[1] http://quantifiedawesome.com/records
[2] https://www.kaggle.com/mrisdal/2016-us-presidential-debates
[3] https://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html
[4] https://is.gd/rdx39u
[5] https://bit.ly/3wBqITA

This set of rules defines a model H_c that characterizes the generation process for sequences in class c. To predict the class of an input sequence S, we compute the description length $L(S|H_c)$ for each class c and return the class that compresses S best, i.e., the class that minimizes $L(S|H_c)$.

We compare the COSSU classifier to (i) an SVM classifier trained on a bag-of-words representation of the text instances, (ii) an HMM-based classifier, and (iii) logistic regression trained on top of the BERT language model [20].

Results. Despite not being a native classifier, COSSU achieves the best performance on three of the experimental datasets, and exhibits comparable performance to HMM and state-of-the-art text classifiers. Furthermore, and unlike its competitors, COSSU is inherently interpretable because its rules serve as an explicit explanation for the outcome of classification. As anecdotal examples, we show a few rules characterizing the classes of the Stylometry dataset:

E.A. Poe:	rue $\rightarrow$ morge
	ourang $\rightarrow$ outang
	little old $\rightarrow$ gentleman
H.P. Lovecraft:	pnakotic $\rightarrow$ manuscript
	catch eight $\rightarrow$ clock coach arkham
	necronomicon mad $\rightarrow$ arab abdul alhazred

VII. CONCLUSION

We have presented COSSU, an algorithm to mine sequential rules from a long sequence of symbolic events. Our approach is inspired by the MDL principle and retrieves a compact and relevant set of rules. Experiments on real-world data show that, in addition to providing an interpretable model for sequential data, the retrieved rules achieve competitive accuracy on the tasks of next event prediction and classification, as compared to other symbolic methods.

As future work, we would like to design a procedure to optimize the weights of the rules while dynamically adjusting their precision. We would also like to allow our method to handle gaps, perhaps by taking inspiration from [8].

Acknowledgements. The authors would like to thank Nikolaj Tatti for providing a fast implementation of a closed gapless sequential pattern miner.

REFERENCES

[1] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computing*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A Survey of Sequential Pattern Mining," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.

[3] B. Cule and B. Goethals, "Mining Association Rules in Long Sequences," in *Advances in Knowledge Discovery and Data Mining*, 2010, pp. 300–309.

[4] P. Fournier-Viger, T. Gueniche, and V. Tseng, "Using Partially-Ordered Sequential Rules to Generate More Accurate Sequence Prediction," in *Advanced Data Mining and Applications*, 2012, pp. 431–442.

[5] F. Fumarola, P. F. Lanotte, M. Ceci, and D. Malerba, "CloFAST: Closed Sequential Pattern Mining Using Sparse and Vertical Id-lists," *Knowledge and Information Systems*, vol. 48, no. 2, pp. 429–463, 2016.

[6] P. Fournier-Viger, T. Gueniche, S. Zida, and V. S. Tseng, "ERMiner: Sequential Rule Mining Using Equivalence Classes," in *Proc. of the Int. Symp. on Intelligent Data Analysis*, 2014, pp. 108–119.

[7] J. Vreeken, M. van Leeuwen, and A. Siebes, "Krimp: Mining Itemsets that Compress," *Data Mining and Knowledge Discovery*, vol. 23, no. 1, pp. 169–214, 2011.

[8] N. Tatti and J. Vreeken, "The Long and the Short of It: Summarising Event Sequences with Serial Episodes," in *Proc. of ACM KDD*, 2012, pp. 462–470.

[9] P. Fournier-Viger, R. Nkambou, and V. S.-M. Tseng, "RuleGrowth: Mining Sequential Rules Common to Several Sequences by Pattern-growth," in *Proc. of ACM SAC*, 2011, pp. 956–961.

[10] N. Gupta, N. Mangal, K. Tiwari, and P. Mitra, *Mining Quantitative Association Rules in Protein Sequences*, 2006, pp. 273–281.

[11] P. Grünwald, *The Minimum Description Length Principle*. The MIT Press, 2007.

[12] J. Fischer and J. Vreeken, "Sets of Robust Rules, and How to Find Them," in *Proc. of ECML-PKDD*, 2019.

[13] R. Bertens, J. Vreeken, and A. Siebes, "Keeping it Short and Simple: Summarising Complex Event Sequences with Multivariate Patterns," in *Proc. of ACM KDD*, 2016, pp. 735–744.

[14] A. Bhattacharyya and J. Vreeken, "Efficiently Summarising Event Sequences with Rich Interleaving Patterns," in *Proc. of SIAM ICDM*, 2017, pp. 795–803.

[15] M. Shokoohi-Yekta, Y. Chen, B. Campana, B. Hu, J. Zakaria, and E. Keogh, "Discovery of Meaningful Rules in Time Series," in *Proc. of ACM KDD*, 2015, pp. 1085–1094.

[16] Y. Yan, L. Cao, S. Madden, and E. A. Rundensteiner, "SWIFT: Mining Representative Patterns from Large Event Streams," *Proc. of the VLDB Endow.*, vol. 12, no. 3, pp. 265–277, 2018.

[17] N. Tatti and B. Cule, "Mining Closed Strict Episodes," *Data Min. Knowl. Discov.*, vol. 25, no. 1, pp. 34–66, 2012.

[18] J. Kiefer, "Sequential Minimax Search for a Maximum," *Proc. of the American Mathematical Society*, vol. 4, no. 3, pp. 502–506, 1953.

[19] E. Bourrand, L. Galárraga, E. Galbrun, E. Fromont, and A. Termier, "Discovering Useful Compact Sets of Sequential Rules in a Long Sequence," 2021. [Online]. Available: https://arxiv.org/abs/2109.07519

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. of the NAACL Conference*, 2019, pp. 4171–4186.