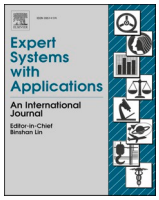
Expert Systems With Applications 226 (2023) 119974

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



Predictive temporal patterns discovery

Nofar Sarafian Ben Ari *, Robert Moskovitch

Software and Information Systems Engineering, Ben Gurion University, Beer Sheva, Israel

ARTICLE INFO

Keywords:
Temporal data mining
Temporal patterns discovery
Time intervals mining
Classification

ABSTRACT

In recent years, the use of frequent temporal patterns as features for classification has increasingly been used and investigated. In this process, commonly frequent patterns are mined from each class separately. Then the patterns are unified, and feature selection methods may be employed, which are given to induce a classifier. However, this approach is very time consuming since the mining of each class separately takes time. In this paper, we introduce the Saraswati suite that can modify a temporal patterns discovery algorithm into a predictive temporal patterns discovery algorithm, which we demonstrate on Time Intervals Related Patterns. The suite enables predictive patterns to be favored in runtime, while mining both classes simultaneously to discover these patterns. This is through the use of a novel stopping criteria that we call the *Saraswati* selection criteria and strategies suite. Since the selection criteria are based on the patterns' metrics, such as their frequency in each class or their reoccurrence, and more, it is explainable to domain experts, rather than as a score as happens with common feature selection measures. We modified an existing time intervals related patterns discovery algorithm according to the Saraswati suite, and evaluated it rigorously against the current approach on six real-life datasets. Our results show that the Saraswati-based algorithm is much faster than discovery of the entire set of frequent patterns, and the selection criteria are more effective than existing state-of-the-art feature selection methods when the discovered predictive patterns are used for classification. Additionally, the selection of the patterns is explainable in the domain expert's terminology based on several meaningful metrics.

1. Introduction

In recent years, numerous research efforts have been made in the development of frequent temporal patterns discovery algorithms (Harel and Moskovitch, 2021; Moskovitch et al., 2015; Tseng and Lee, 2009), in order to enable the discovery of actionable temporal knowledge (Moskovitch, 2022). However, often criticism of data mining in general, and specifically pattern mining, is toward a process that results in a large number of frequent patterns from which it is hard to determine those that are useful. For this task, various interestingness measures (Patel et al., 2008; Fradkin and Mörchen, 2015; Zhou et al., 2016; Shknevsky et al., 2017) were proposed, but still this remains a challenge since there often is a lack of context and purpose, in part because the process is unsupervised. Additionally, most of the interestingness measures are based on some information-gain-oriented metric, which sums to a score which is not typically informative enough for a domain expert. In this work, we wanted to develop selection criteria that consist of a pattern's metrics that are informative and understandable for humans, unlike measures that consist on some information measure, and whose selection can be easily explained. So, here our criteria will tell if a pattern is predictive, based on its properties' values in the classes and the difference in the values in the classes. For example, it is more frequent in one class, or it has higher reoccurrence in one of the classes (relatively to the other), or its instances average duration is higher in one class.

Recently, there has been an increase in studies employing these patterns as features for classification (Cheng et al., 2007; Patel et al., 2008; Batal et al., 2012; Fradkin and Mörchen, 2015; Moskovitch and Shahar; 2015b, Itzhak et al., 2000; Dvir et al., 2020; Novitski et al., 2022) – an approach often called temporal patterns-based classification. Thus, frequent temporal patterns are discovered from each class separately, and then often unified and used as a features set for classification. Sometimes, feature selection methods are used to favor the most predictive patterns as features. However, these are typically based on various information gain metrics that result in a score and are often not intuitive or understood by domain experts. Moreover, such selection methods typically are applied after the mining process is completed on each class' data, which is often a time-consuming task.

* Corresponding author.
E-mail addresses: nofarsa@post.bgu.ac.il (N. Sarafian Ben Ari), robertmo@bgu.ac.il (R. Moskovitch).

https://doi.org/10.1016/j.eswa.2023.119974
Received 5 July 2021; Received in revised form 22 March 2023; Accepted 23 March 2023
Available online 27 March 2023

In this paper, we propose a novel classification-driven selection criteria for the discovery of predictive frequent temporal patterns, which is applied during the mining process, and intends to decrease the runtime. First, we were interested in selecting temporal patterns based on the differences in the appearance of the patterns in the classes, based on several metrics, for example, based on their frequency in each class, or other metrics, such as the average number of their instances within an entity or the average duration of the instances in the entities. The patterns that appear most differently in the classes will be favored. Moreover, applying this process simultaneously on each class separately in parallel has the potential to decrease the runtime meaningfully, which is part of the investigation and contribution of this work. Being able to discover *predictive* temporal patterns efficiently has a meaningful potential impact in various domains in which longitudinal data is available. We describe here this motivation in more detail.

1.1. The predictive temporal patterns discovery challenge

In Fig. 1, we describe how temporal patterns-based classification is currently performed, in which frequent patterns are discovered from each class separately, followed by a merge of the patterns that were discovered from each class. They are then detected in both classes (to verify that the patterns that were discovered in one class, its instances are detected also in the other class, even if it was not frequent there), as described in the top flow of Fig. 1. Additionally, in this paper, we propose the new *Saraswati suite* in which predictive patterns are discovered while mining both classes simultaneously. The BaseLine is illustrated at the top of Fig. 1, in which the temporal patterns discovery algorithm (TPDA) is applied on each class *separately*. The discovered patterns from each class (some patterns may be discovered in both classes, and some only in one) are unified. Then feature selection can be further applied (Patel et al., 2008; Fradkin and Mörchen, 2015; Moskovitch and Shahar, 2015b), and finally, a features matrix, in which the entities are represented by the patterns as features, is ready for a classification inducer. Although his current process, which we refer to as *BaseLine* is comprehensive since all the frequent patterns of each class are discovered, it is also very time consuming.

The process at the bottom of Fig. 1 presents our introduced Saraswati-based process, in which the temporal patterns discovery algorithm (Saraswati-TPDA), after being extended by the Saraswati suite, is applied on the data of both classes simultaneously. Then, in runtime, it discovers the most predictive patterns, and through that expands only part of the patterns in the enumeration tree (of both classes). In a more formal fashion, we define the predictive patterns discovery task in the Methods section in Definition 6.

Thus, in this paper, we present a novel predictive pattern discovery process, which replaces the BaseLine process described in Fig. 1, in which the mining is performed on Class A, and then on B separately, and unifying the patterns, followed by detecting them in each class separately, and then applying features selection. Our new process incorporates all of these stages within a single mining operation, which is applied on both classes' training data in parallel. Additionally, novel selection criteria are introduced within a suite for predictive temporal patterns, and we show how the process can be applied to transform a temporal pattern discovery algorithm into a predictive temporal pattern discovery algorithm that is applied in runtime – instead of the current BaseLine process. The proposed process is less comprehensive, but seems to be effective, while having shorter runtimes.

The main contributions of this paper are the following:

1. The Saraswati suite that enables a temporal pattern discovery algorithm to be transformed into a *predictive temporal pattern discovery algorithm*, including selection criteria, or more specifically:
   a. Instead of mining class A, and then mining class B, and then unifying the discovered patterns, and detecting their unification in both classes, followed by running a feature selection method, it enables discovery of the predictive patterns in a single mining run applied on both classes simultaneously.
   b. It enables pattern selection based on the differences in the pattern's metrics in each of the classes, such as frequency, reoccurrences, etc.
   c. It includes a novel selection score that consists on the various pattern's metrics, which makes its more explainable and meaningful.
   d. It is explainable – the reason a pattern was selected is explained to a domain expert in a simple way based on meaningful pattern's metrics' different values in the each of the classes (rather than as an InfoGain-type of score).
2. **K**arma**L**ego Cla**S**sification **D**riven (KLSD) is a demonstration applying the Saraswati suite to the KarmaLego algorithm for time intervals mining, resulting in an efficient mining that is applied on two classes simultaneously, and selecting the most predictive patterns.
3. A rigorous evaluation is conducted of six datasets that evaluate the effectiveness in runtime reduction and the number of predictive patterns discovered, as well as their use for classification in comparison to patterns selected by other state-of-the-art feature selection methods.

The rest of the paper is organized accordingly: We start with the background, reviewing relevant topics, and continue with the description of the Saraswati suite in the Methods section. Then, in the Evaluation section, we list our research questions and corresponding experimental plan. Finally, we present the results of the experiments,

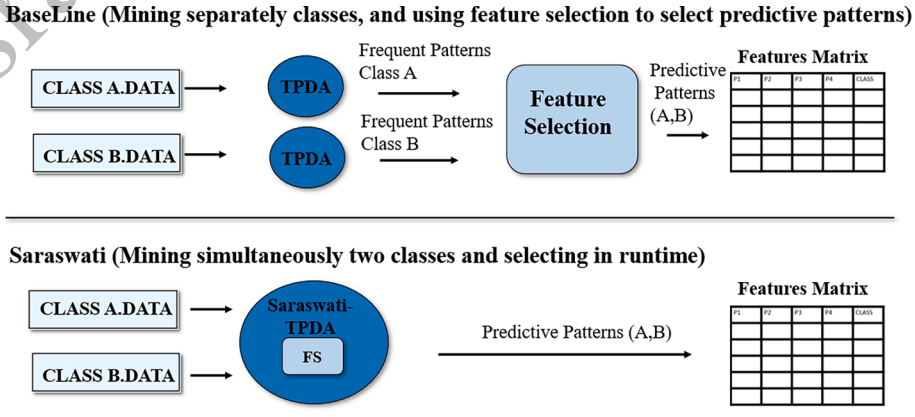


Fig. 1. The top part presents the BaseLine process in which the discovery of the temporal patterns is performed on each class separately, and then feature selection may be performed on the entire set of discovered patterns. Finally a features matrix is created to induce a classifier.

discuss them, and present our conclusions.

2. Background

Here, we review relevant topics in the scope of temporal data mining and machine learning, and more specifically, frequent temporal patterns-based classification. The first part examines some approaches in the field of pattern-based classification. Since the proposed suite also includes feature selection criteria, the next part of this section discusses filtering feature selection methods and their use in classification. Then we briefly review the fields of temporal abstraction and time intervals mining, which we used in our study. Although the proposed methodology was designed in a generic way so that it can be implemented in various types of temporal pattern discovery algorithms, in this paper, we demonstrate it on Time Intervals Related Patterns (TIRPs)-based classification.

2.1. Patterns-Based classification

The idea of using patterns as features for classification started from "static" patterns such as itemsets, and then developed to the use of sequential or time intervals patterns. Frequent pattern mining discovery has been a focused subject in data mining research, with many approaches for mining various kinds of patterns including itemsets (Liu et al., 1998; Han et al., 2000), sequences (Tseng and Lee, 2009; Fradkin and Mörchen, 2015; Zhou et al., 2016), and time intervals patterns (Patel et al., 2008; Batal et al., 2012; Moskovitch et al., 2015). Many association-rule-based classifiers were proposed by using efficient association-rule mining algorithms such as Apriori (Agrawal et al., 1994) and FP-growth (Han et al., 2000), among others.

Cheng et al. (2007) proposed a framework of frequent sequence pattern-based classification, in which the data are partitioned according to class label, and frequent patterns are discovered in each partition with minimal support (the percentage of entities). Then feature selection is applied on the frequent patterns, and a model is built. They also proposed a new feature selection method MMRFS (Maximal Marginal Relevance Feature Selection) based on relevant and redundant measures of the patterns in order to choose the optimal set of predictive patterns from the full set of frequent patterns discovered. Beyond itemset pattern mining, for sequential data, sequential pattern mining methods were developed such as Generalized Sequential Patterns (GSP) (Srikant & Agrawal, 1996), SPADE (Zaki, 2001), PrefixSpan (Pei et al., 2004), SPAM (Ayres et al., 2002), and more (Tseng and Lee, 2009; Fradkin and Mörchen, 2015; Zhou et al., 2016).

Lesh et al. (2000) introduced the FeatureMine, a scalable feature-mining algorithm which uses sequence mining techniques to choose only the relevant patterns for classification. Experiments on several datasets demonstrate that FeatureMine improved classification results by 10–50% and can efficiently reduce the number of produced patterns. Tseng and Lee (2009) proposed the Classify-By-Sequence (CBS) algorithm for classifying large sequence datasets. The main methodology of their method is mining classifiable sequential patterns (CSPs) from the sequences and then assigning a score to the new data object for each class using a scoring function that combines metrics such as support, confidence, and pattern length. They suggested a number of alternative scoring functions and tested their performance. The class label is assigned by the max score between the classes. According to the experiments, they conclude that CBS is an effective and stable method for classifying temporal data. Another approach (Fradkin and Mörchen, 2015) is direct sequential pattern mining, known as BIDE-Discriminative, which uses class information and the Information Gain measure for direct mining of predictive sequential patterns. Fradkin and Morchen showed that their algorithm provides an efficient solution for sequence classification as it generates a small number of predictive patterns that lead to comparable classification performance. Zhou et al. (2016) proposed the SCIP (Sequence Classification Based Interesting Patterns) algorithms to mine two types of interesting patterns (itemset, sequences) that combine confident classification rules. They convert the discovered patterns into classification rules and proposed two methods of classifiers.

In the last decade, more methods for sequences of symbolic time intervals were published, including the use of patterns for classification, which is the type of data we demonstrate on the use of Saraswati in this paper. We elaborate on these methods in Subsection 2.4, since in order to better understand them, it is better to refer to temporal abstraction and time intervals mining first, as we do in the following subsections. In fact, in this paper, we demonstrate the use of Saraswati on multivariate heterogeneous temporal datasets that went through temporal abstraction.

2.2. Filtering feature selection

The Saraswati score and the KLSD introduced methods enable the discovery of predictive patterns in mining runtime, and for that the paper introduced a new score for TIRPs selection. This score is applied in the mining runtime, but could be as well applied separately as a feature selection method for TIRPs features, or even generally when temporal patterns are used as features for classification. Moreover, in the evaluation we also compare it to the use of standard feature selection methods, and for that we are reviewing here the field of features selection. Feature selection is a preprocessing step used in classification to reduce the number of features through favoring a subset of the more important features that correlate with the class label. Feature selection has three goals: to improve the classification performance, to provide faster and more efficient predictors, and to create potentially better understanding of the data in the context of classification. There are generally two types of feature selection methods: one is *filtering* feature selection, also known as variable ranking, which is a preprocessing step (Kohavi and John, 1997) independent of the choice of the classifier; and second, is *wrapping* feature selection, which is dependent on the classifier since it evaluates subsets of features according to their effectiveness to a given classifier. Although variable ranking is not optimal, it is often used for preliminary selection of features prior to the wrapper method because of its computational and statistical scalability. The method that we present here functions as a filtering feature selection, which is applied while the features in the form of temporal patterns are discovered.

Consider a set of m examples $\{x_k y_k\}(k = 1, \cdots m)$ consisting of n input variables $x_{k,i}(i = 1, \cdots n)$ and one output variable y_k. Variable ranking makes use of a scoring function S(i) computed from the values $x_{k,i}$ and $y_k, k = 1, \cdots m$. Then sort the scores in descending order of S(i), according to the significance of the variable relative to the target, and choose the top k best variables, for example, using Fisher's criterion to rank variables in a classification problem where the covariance matrix is diagonal is optimum for Fisher's linear discriminant classifier (Peter et al., 2001). Another popular criterion is the Pearson correlation coefficient R(i) (Benesty et al., 2009). Correlation criteria like R(i) that can only detect linear dependencies between a variable and target. Correlation criteria are often used for microarray data analysis, as illustrated by Weston et al. (2003). Many approaches of the variable selection problem used information from theoretic criteria (Bekkerman et al., 2003; Dhillon et al., 2003; Torkkola, 2003). In classification, often the ranking of features is performed after the features are extracted and placed in a matrix-like form. In this work, the ranking of features occurs during the feature extraction, along the mining process.

Since we demonstrate the Saraswati suite on a time intervals mining algorithm, and we abstracted our datasets, we will review temporal abstraction and time intervals mining. Later, we also examine the KarmaLego algorithm, on which we demonstrate the use of the Saraswati suite.

2.3. Symbolic time intervals

While symbolic time intervals can be raw data, before we review the development of time intervals mining algorithms, we will first refer to temporal abstraction (Shahar, 1997; Höppner, 2001), which is a mechanism that transforms time point series into symbolic time intervals series. In our study evaluation, we used state abstraction, in which given a set of cutoffs, the values are discretized into states that are concatenated when adjacent states have the same value into a symbolic time interval that is defined by its start-time, end-time, and symbol-id. A more detailed illustration of the temporal abstraction process can be found elsewhere (Shahar, 1997; Höppner, 2001; Moskovitch and Shahar, 2015a).

To determine the cutoffs, in addition to those that are knowledge-based, in which they are given by a domain expert, there are several common discretization methods such as Equal-Width Discretization (EWD), which uniformly divides the ranges of each value and Equal Frequency Discretization (EFD) that divides the data into states having the same frequency. More advanced methods include: Symbolic Aggregate approXimation (SAX) (Lin et al., 2007), which focuses on a discretization of the values based on their Gaussian distribution; Persist (Mörchen and Ultsch, 2005a; Mörchen and Ultsch, 2005b), which maximizes the duration of the resulting time intervals, and explicitly considers the temporal dimension; a relatively recent approach (Moskovitch and Shahar, 2015b) that proposes a *supervised* Temporal Discretization for Classification that learns the cutoffs that increase the differences in the states according to their distribution in the classes, which is expected to increase classification accuracy; and there are more (Ramírez-Gallego et al., 2016).

2.4. Time intervals mining

Discovering frequent patterns from symbolic time intervals has attracted research increasingly in the past years. Symbolic time intervals can come from various sources, whether raw (i.e., in the medical domain, conditions, procedures, or drug exposers), or the result of time point series that went through TA. This representation has several advantages, such as providing a uniform representation of the heterogeneous variables, which is generalized, straightforward and when based on domain knowledge may be understandable and familiar for domain experts. Most time intervals mining approaches use some subset of Allen's temporal relations (Allen,1983), which are a finite set of 13 temporal relations between a pair of time intervals. The set includes: *before*, *meets*, *overlaps*, *starts*, *during*, *finishes*, and their corresponding inverse relations *after*, *met-by*, *overlapped-by*, *started-by*, *contains*, *finished-by*; and *equals*. Often when the data is ordered lexicographically, it is also considered as being composed of seven basic relations, six of which have an inverse (*equals* is its own inverse).

Höppner (2001) was the first to define a non-ambiguous representation of time interval patterns that are based on Allen's relations, by a k^2 matrix to represent all the pairwise relations within a k-intervals pattern, which we will refer to as a Time Intervals Related Pattern (TIRP) (the formal definition of a TIRP is presented in Section 3 in Definition 1). Another method proposed by Papapetrou et al. (2009) is a hybrid approach called H-DFS, which combines the first indexing of the pairs of time intervals according to their temporal relation and then mines the extended TIRPs in a candidate generation fashion. In addition, they introduced an epsilon threshold to make the temporal relations more flexible. Other methods for time intervals were proposed later, but here, we focus on KarmaLego introduced by (Moskovitch et al., 2015a), which exploits the transitivity property of the temporal relations for more efficient candidate generation. We describe this in more detail in the following section since, in this paper, we demonstrate our approach on this algorithm.

2.4.1. The KarmaLego algorithm

The KarmaLego algorithm is a fast time intervals mining algorithm for the discovery of TIRPs by exploiting the transitivity of temporal relations, enabling an efficient candidate generation mechanism (Moskovitch et al., 2015a). KarmaLego consists of two main steps: Karma, in which the entire set of entities' time intervals data are scanned and indexed. Through that, all the symbols are counted, and each pair of symbolic time intervals and the temporal relation among them are indexed in an index called DharmaIndex that contains all the frequent 2-sized TIRPs ($k = 2$), shown at the second level in Fig. 3. The DharmaIndex will be used later in the Lego phase to retrieve the relevant pairs through the TIRP extension process. In the second phase, the Lego algorithm, a recursive process, extends the frequent 2-sized TIRPs.

Based on the candidate symbols (from level 1) and the potential temporal relations, a set of candidate TIRPs are generated by exploiting the transitivity of the temporal relations into a tree of longer frequent TIRPs. KarmaLego discovers the complete set of frequent TIRPs, including all their multiple occurrences, within the same entity [which is crucial for the complete discovery of frequent TIRPs (Moskovitch and Shahar, 2015b)]. The result of this process is a frequent TIRP enumeration tree (Fig. 2). In Fig. 2, an enumeration tree of KarmaLego with three symbols (A,B,C) for simplicity is shown, and one relation before (<). At each level, the algorithm adds one symbol to the existing pattern and temporal relations among the existing symbols in the pattern and the new symbol to make it non-ambiguous. Each node at each level represents a pattern which contains symbolic time intervals and the relation between each pair of symbolic time intervals. As we go down the tree, the patterns become larger (more symbols), but typically with lower support.

The complexity of mining time intervals, or TIRPs discovery, including the discovery of the entire instances, which is required for a complete discovery (Moskovitch and Shahar, 2015b) is described in detail in (Moskovitch and Shahar, 2015b). In this paper, the approach is applied on the KarmaLego algorithm and provides criteria that avoid discovering the entire enumeration pattern tree and, by that, shortens the runtime. However, the complexity of the mining algorithm remains the same, as described in Moskovitch and Shahar (2015b).

2.5. Time intervals Related Patterns-Based classification

In addition to the papers we mentioned earlier, in which sequential patterns were used as features for classification, here we look at studies that used TIRPs as features for classification, as we do in our evaluation in this paper. Patel et al. (2008) were the first to use the discovered TIRPs for the classification of multivariate temporal data.

They introduced the IEClassifier, a classification method that is designed specifically to classify data using temporal patterns. Two versions of the IEClassifier were proposed: Best_Confidence, in which the class having the highest confidence is selected, whereas in the Majority_Class, the class to which the majority of the patterns discovered belong is assigned. The two versions of the IEClassifier were compared with common classifiers, such as C4.5 and SVM, when applied to a non-temporal representation of the data. The Majority_Class outperformed the other classification methods. Batal et al. (2012) presented the Recent Temporal Pattern (RTP) mining framework, which mines frequent temporal patterns backwards in time, starting from patterns related to the most recent observations. They compared the classification performance of the following constructed features. Last_values – The features are formed from the most recent values of each variable; TP – The features correspond to all frequent temporal patterns; TP_sparse – The features correspond to the top 50 discriminative temporal patterns that are selected using a sparse linear model; RTP – The features correspond to all frequent RTPs; RTP_sparse – The features correspond to the top 50 discriminative RTPs. The results also show that RTP and RTP_sparse mostly outperform TP and TP_sparse which assess the idea that recent patterns features are more relevant for classification.

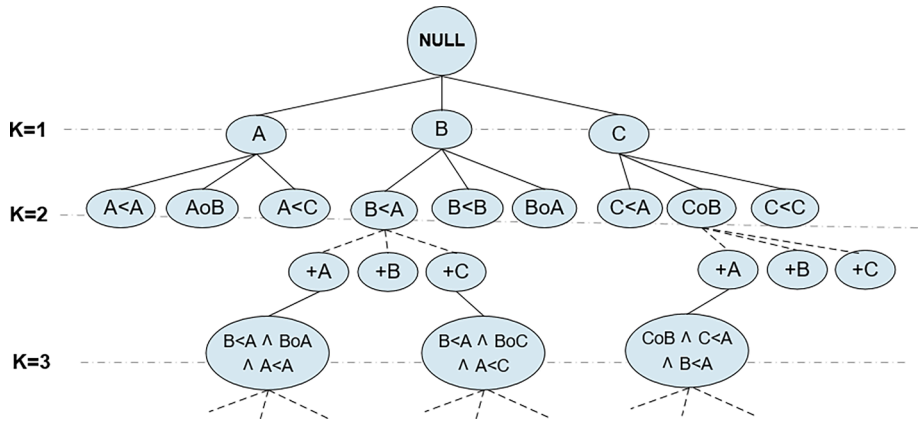


Fig. 2. An enumeration tree of KarmaLego with three symbols (A,B,C) and two relations before and overlap (<,o).

Another approach is a framework for classification of multivariate temporal data called KarmaLegoSification (KLS), introduced by Moskovitch and Shahar (2015b). Two novel representation metrics for features were introduced beyond the binary representation, which are horizontal support (the number of TIRP occurrences per entity) and mean duration (the average of the TIRP instances' time-length per entity) to represent the TIRP features for classification. The results show that using SAX led to better performance than using EWD for state abstraction, and using three temporal relations was superior to the use of Allen's original seven temporal relations. Additionally, the MeanD (mean duration) TIRP representation method performed somewhat better than the HS (horizontal support) representation. A recent study introduced Maitreya (Moskovitch et al., 2016), a framework that discovers TIRPs only from the cohort of patients having the outcome event. The results shown that representing the TIRPs using the horizontal support outperformed the binary and mean duration representations. In this paper, we propose a suite that enables us to transform a frequent temporal patterns discovery algorithm into an algorithm that discovers *predictive* temporal patterns in runtime by performing the mining simultaneously on data of each of the classes and applying the Saraswati novel selection criteria. In this study, we apply the suite to the KarmaLego algorithm, while it should be possible to apply it to most temporal pattern discovery methods, especially those of various types of sequential patterns. Since we demonstrate our ideas on KarmaLego, we start with a brief description of the algorithm for the reader's convenience. A more detailed and comprehensive description can be found elsewhere (Moskovitch and Shahar, 2015a; Moskovitch et al., 2015).

3. Methods

We describe here, in detail, the *Saraswati suite* that as mentioned earlier was designed to transform a temporal pattern discovery algorithm into a *predictive* temporal pattern discovery algorithm, in place of current approaches (Subsection 2.5) in which a temporal pattern discovery algorithm is applied separately on each class to discover the most frequent patterns, unifying the patterns, and detecting each classes' training data, then applying a feature selection method (as in the papers mentioned in Subsection 2.5). Here, we propose a temporal pattern discovery algorithm that performs "feature selection" within the mining runtime; thus, the algorithm is run on both classes simultaneously. Unlike the previous methods mentioned in Subsection 2.5, which mine the entire population using an InfoGain-based criterion, here, we present novel selection criteria and strategies that are more understandable and meaningful for humans, making it not only more useful for knowledge discovery, but also more effective for classification than InfoGain-based methods, as our results show.

3.1. The Saraswati suite

As mentioned in Subsection 1.1, currently, the process of performing temporal pattern- based classification requires the following steps, in short: mining Class A, mining Class B, unifying the discovered patterns from each class into a collection of patterns, detecting the unified collection in each class separately, creating a training matrix, and applying a feature selection method for reduction. The Saraswati suite that we introduce here proposes an approach in which the mining algorithm can be extended, so it will enable both classes to be run simultaneously, and to discover the predictive patterns in a single run, in runtime.

Thus, the Saraswati suite requires the following general modifications (along the section, we describe these in detail) to an existing temporal patterns discovery algorithm:

1. *Temporal patterns' comparison metrics*. In order to compare the patterns' properties in the different classes, metrics are needed. Their frequency can be used, often called support, which exists typically in any frequent temporal pattern discovery algorithm. But, to enable additional comparison criteria, more metrics are required, such as, for example, the number of (horizontal) instances of the pattern in a specific entity. In this paper, we propose two metrics that most temporal patterns discovery algorithms should be able to implement.
2. *Mining multiple classes simultaneously*. To perform the patterns' metrics comparison during the discovery process, it is necessary to mine the classes' data separately in parallel. This often requires separate data structures and a central process that manages the mining. Here, we focus on a binary classification problem, meaning two classes only.
3. *Incorporating the Saraswati pattern selection algorithm* as a stopping criterion, instead of the minimal support and at least one of its selection strategies, whether this is metrics or score based.

We start by defining the components on which the method consists and then we present Saraswati – a new selection criteria and strategies method, and an algorithm for selecting predictive patterns. Finally, we demonstrate the use of the Saraswati suite on the KarmaLego algorithm that we call KLSD (**K**arma**L**ego Cla**S**sification **D**riven), for mining the populations of two classes to reveal only the (distinguishing) predictive patterns between them.

3.1.1. Definitions

Here, we provide the definitions for the components of the proposed method, which corresponds to the current relevant literature that was referred in Subsection 2.4.1 and expand it. As mentioned earlier, the Saraswati suite was designed and described in a generic way, as much as possible, so that it can be implemented to transform most temporal

patterns discovery algorithms. In the following definitions, we will refer to TIRPs specifically, but most of the definitions can be easily translated to general temporal patterns. We refer to this when it is relevant.

Definition 1.

A non-ambiguous Time Intervals Related Pattern P is defined as P ={I, R}, where $I = \{I^1, I^2, \cdots, I^k\}$ is a set of k symbolic time intervals and

$$R = \bigcap_{i=1}^{k-1} \bigcap_{j=i+1}^{k} r(I^i, I^j) = \{r_{1,2}(I^1, I^2), .., r_{1,k}(I^1, I^k), \cdots, r_{k-1,k}(I^{k-1}, I^k)\}$$

defines the conjunction of all the temporal relations among each of the $\frac{k^2-k}{2}$ pairs of symbolic time intervals in I.

Fig. 3 presents an example of a 4-sized TIRP. The half matrix on the left represents the conjunction of all the pairwise temporal relations defining it. The last column describes the temporal relations of the previous STIs A, B, and C relative to the STI D (the entire column of these temporal relations has a white background since STI D is the last STI).

Definition 2.

Given a database of |E| distinct entities, the vertical support of a TIRP t is denoted by the cardinality of the set E^t of distinct entities within which t holds at least once, divided by the total number of entities |E|: ver_sup(t) $=|E^t|/|E|$.

The vertical support is the term usually referred to as support in the context of association-rules, itemsets, or sequential mining, representing its frequency in the dataset.

The Saraswati suite requires metrics for the properties of a temporal pattern in order to measure the differences in the pattern's appearances in each of the classes. These can be various and not necessarily those that we propose here. However, those that we propose here are generic metrics that can easily be implemented and extracted from any type of temporal pattern.

Definition 3.

The horizontal support of a pattern t for an entity e_i hor_sup (t, e_i) is the number of instances of the pattern t found in e_i (illustrated in Fig. 4).

Obviously, this metric can be measured for other types of temporal patterns, such as sequential patterns or Markov chains, etc., and implemented easily by making sure that the number of instances of a pattern in each of the mined entities is counted.

Definition 4.

The vertically normalized horizontal support (VNHS) of a pattern t for an entity e_i is the hor_sup(t, e_i) divided by the maximal horizontal support value of the pattern t across all the entities. In this study, we used the VNHS as a metric to represent the patterns as features for a classifier, although it can be used as a metric for the Saraswati selection criteria, as we explain later.

Definition 5.

The mean duration of the n supporting instances of the same k-sized pattern t within an entity e is defined by the average of the duration of all the n instances, where each instance duration is defined from its earliest time point till its last time point. We here define the Mean Duration for TIRPs:

$$\text{MeanDuration}(t, e) = \frac{\sum_{i=1}^{n}(\text{Max}_{j=1}^{k} I_{et}^{i,j} - I_{st}^{i,1})}{n}$$

where $I_{st}^{i,1}$ is the start-time (s) of the first time interval in the i-th instance (among n instances), and the Max operator selects the time interval having the latest end-time (et) among the k time intervals of an instance i. An illustration and a calculation example can be seen in Fig. 4.

We will use these metrics (Definitions 2–5) in our selection of predictive TIRPs.

3.1.2. Saraswati selection strategies

The new classification-driven TIRP discovery and selection method intends to ideally discover only predictive patterns, while avoiding the expansion of the entire enumeration tree of candidate patterns, to find only those that are potentially predictive. However, using Saraswati as a stopping criteria naturally results in not revealing all the possible predictive patterns in that process. We examine this aspect in our experimental plan. The definitions above till definition 6 are absed on previous papers in the field of time intervals mining, while the following definitions are new and relevant to the method described in this paper.

Definition 6.

The predictive patterns (p-patterns) discovery task:

Given a pair of classes C = {c_0, c_1}, each represented by an exclusive set of corresponding entities E = {E_0, E_1}, the goal is to discover the patterns that, according to some criteria or score, are meaningfully different in the class populations. The discriminative score may reflect, for example, the difference in frequencies (vertical support) values of the classes, or the difference in other relevant pattern's metrics (as we propose in this paper) values in the classes.

For generality, we refer to patterns in general, although in our demonstration, we use specifically TIRPs. In order to select the *predictive patterns* during the mining process, we developed several novel strategies and selection criteria. We have two strategies – one that is metrics based and the second based on a score threshold:

1. Metrics rule-based strategies – The algorithm checks several ordered criteria and based on these selects the predictive patterns and determines the stopping criteria in the mining process. Here, we have three different strategies: HS (horizontal support), in which the last criterion checks if the two populations are significantly different based on the MHS (Definition 8); MND (mean duration), in which the last criterion checks if the two populations are significantly different based on their means, and HS_OR_MND, which checks both previous criterions with one being enough to satisfy. The names of the strategies represent the final criterion which selects the pattern in the decision tree. We explain the full Saraswati feature selection algorithm next.

2. Score based strategy – A score is computed according to a formula that incorporates various pattern metrics, and if the score is above a specific threshold, the pattern is selected. Since each pattern receives a score, it can be used in ranking feature selection for temporal patterns that are used as features for classification. We compare our score against

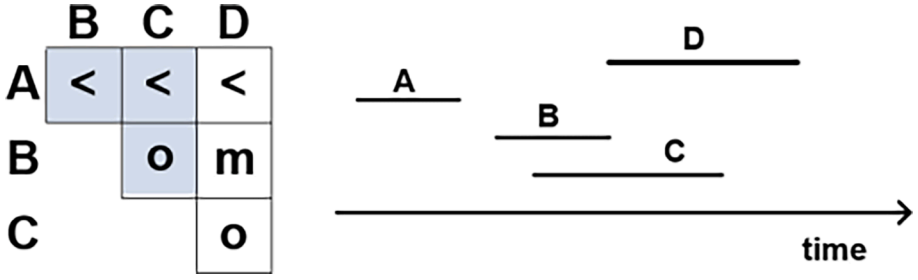


Fig. 3. An example of a time intervals-related pattern having four symbolic time intervals and all of their pairwise temporal relations. On the *right*, the actual four symbolic time interval TIRP is displayed graphically, while on the *left*, a half-matrix representation is given presenting the conjunction of the pairwise temporal relations.

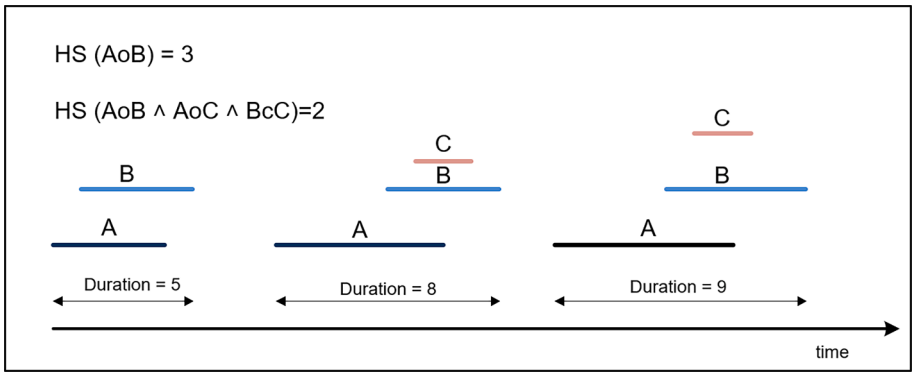


Fig. 4. An illustration of the horizontal support and the mean duration metrics. Two types of TIRPs in a single entity are shown. The first is a 2-sized TIRP A overlap B (AoB) which repeats three times, and therefore, its HS is 3. The second TIRP is an extended 3-sized TIRP defined by AoB ∧ AoC ∧ BcC (B contains C which occurs two times, and therefore, its HS is 2. The mean duration of TIRP AoB is calculated based on the sum of the duration of each of the three instances divided by 3, accordingly: MND(AoB)= (5 + 8 + 9) /3 = 7.33.

several state-of-the-art filtering feature selection methods.

We define here a set of measures that will enable us to fully explain the components of our selection strategies for comparison of a pattern appearance in each of the classes.

Definition 7.

The Delta Vertical Support (ΔVS) is the difference in the vertical support values of a given pattern t between the two classes A and B, in which t^A.vs is the vertical support of pattern t in class A, and t^B.vs is the vertical support of pattern t in class B. The ΔVS is their absolute subtraction: ΔVS= | t^A.vs- t^B.vs|.

Definition 8.

The Mean Horizontal Support (MHS) of a pattern t describes the average horizontal support values of its supporting entities (each entity may have several instances of t, whose count is the horizontal support, while the MHS refers to the average of HS values of all the supporting entities). Thus, given a pattern t and a set of |E| supporting entities, having horizontal support {hs_1, hs_2….,hs_e}, MHS(t) = $\frac{\sum_{i=1}^{|E|} hs_i}{|E|}$. Note that the MHS will be always equal to or larger than 1.

Definition 9.

The Delta Mean Horizontal Support (ΔMHS) is the difference between the MHS of classes A and B for a given pattern t, such that ΔMHS = | MHS(t^A) – MHS(t^B)|.

We can also generalize Definitions 8 and 9 for the *mean duration* metric, as was described earlier. This can also be done with any other comparable metric. Thus, given the TIRP metric's collection of values for each class, we can calculate its mean, as defined in Definition 8, and the delta of the classes' metric mean value, as defined in Definition 9.

To determine whether the difference is meaningful (ΔMHS/ΔMMD), or even significant, we designed a test based on Welch's test (Welch, 1947) for unequal samples [since typically (and favorably) vs_A is different than vs_B]. Thus, given a pattern appearance in two classes A and B, having for each class their collection of entities supporting the pattern, and the horizontal support for each entity $HS_A = \{hs_1, hs_2, hs_3,.., hs_{vsA}\}$, and $HS_B = \{hs_1, hs_2, hs_3,.., hs_{vsB}\}$, in which the number of entities having the pattern (vs_A, vs_B) may be different in the classes. We are using two p-values, p_{hs} and p_{mnd}, which are the significance of the statistical tests, as criterions for determining whether a pattern is predictive. Similarly, if a new metric is introduced, a corresponding p_{metric} must be used.

3.1.2.1. The Saraswati predictability score and algorithm. The Saraswati predictability score shown in Equation 2 incorporates four components: the average of the pattern's vertical support in the two classes, in order to favor patterns with high vertical support (in both classes, to avoid sparse features when used for classification). However, since the average of the vertical support in each of the classes may be the result of a pattern that appears very frequently in one class, and may be infrequent, or even absent in the other class, can be for example 50%, while it may be also with a another pattern that has the same frequency in both classes of 50% for example, we introduced also the delta vertical support. Thus, the delta vertical support assures favoring patterns in which the difference in the vertical support between the two classes is larger, and the statistical p_values of the horizontal support and the mean duration, that represent also the differences of the values in the classes. Since we want to incorporate the p-values, for example the p_{hs}, in the score-based selection strategy as shown in Equation 2, and its values are often very small, bellow 0.1, we perform a normalization and bound the values. When the p-value is below or equal to 0.1, it is multiplied by 10, and if it is above 0.1, it is set to 0.1.

Thus, to calculate these measures, the *following* inputs are required:

vs_A – the vertical support of a pattern t in class A.

vs_B – the vertical support of a pattern t in class B.

Δ*vs* – the unsigned difference between the pattern's vertical support of the classes, as defined in Definition 7.

p_{hs} – the normalized p_value of the statistical test of the two classes' mean horizontal support.

p_{mnd} – the normalized p_value of the statistical test of the two classes' mean duration.

Thus, the score formula is defined as shown in Equation 2, which consists of the above components.

$$SaraswatiScore(P) = Avg\left(\left(\frac{vs_A + vs_B}{2}\right), \Delta vs, (1 - p_{hs}), (1 - p_{mnd})\right) \quad (2)$$

The p_values are replaced by their opposite values in order to represent the power of the statistical tests results. As the p_value is lower, the statistical test is more significant, and we prefer that all four components will have higher values to create a high predictability score. In addition, we gave equal weight to all four components in the formula since exploring and optimizing their weight will require many experiments not in the scope of this paper. Rather, we evaluate the method with its default weights.

Algorithm 1 – **Saraswati Temporal Pattern Selection Algorithm**.

Input:

t – a candidate pattern to be selected (has metrics in A or B) to be selected as predictive or to be extended in the tree
$_{MIN}\Delta VS$ – the minimum vertical support delta threshold
α– the significance threshold of the statistical test
Strategy – the strategy to use for pattern selection (SCORE/ /HS/MND/ HS_OR_MND)
score_threshold – pattern predictiveness threshold
Output: a Boolean value whether the pattern is predictive or not

```
Δvs= | t_A.vs – t_B.vs |
p_hs ← t_test (, t_A.HS, t_B.HS)
p_mnd ← t_test (t_A.MND, t_B.MND)
t. score ← Saraswati_Score (t_A.vs, t_B.vs , Δvs, p_hs , p_mnd)
If Strategy= 'score':
    If t. score >= score_threshold:
        return true
    End If
End If
Else // Strategy = 'HS_OR_MND'||'HS' ||'MND':
```

(*continued on next page*)

(*continued*)

Input:	
11.	If Δvs >=$_{MIN}$ΔVS:
12.	return true
13.	End If
14.	Else
15.	If (strategy == 'HS' and $p_{hs} \leq \alpha$) or (strategy == 'MND' and $p_{mnd} \leq \alpha$): // strategy = HS_OR_MND, then HS = true and MND = true
16.	return true
17.	End If
18.	End Else
19.	End Else
20.	return false

The Saraswati algorithm (Algorithm 1) gets a candidate pattern t and its metrics' values for each of the two classes based on the chosen strategy, and then returns a Boolean value indicating if a pattern t is predictive or not. In addition, it also calculates its predictability score and sets it as a pattern property. The first input is a candidate pattern which contains its metrics for two classes.

The second input is the minimum vertical support delta threshold, which is used as one of the criteria, explained next. The alpha parameter is used for the statistical test in order to decide if the test is significant or not. Next is the strategy parameter which can have four different values as mentioned earlier (SCORE, or one of the metrics based options: HS or MND or HS_OR_MND) according to the different strategies. The last parameter is used when the 'SCORE' is chosen.

In Algorithm 1, in lines 1–4, we calculate all the components for the pattern score. If the score strategy was chosen, the algorithm checks if the pattern's score is above the score threshold and then selects it as predictive. Otherwise, the algorithm decides if the pattern is relevant based on two conditions, while when one is satisfied, the pattern is selected. The order of the conditions indicates the magnitude of their meaning in the decision. If the strategy is *metrics* based using one of the following: 'HS_OR_MND' or 'HS' or'MND', as shown in lines 10–19, then the following happens: the first condition checks whether the difference between the vertical support of the pattern in the classes is above a given threshold $_{MIN}$ΔVS, which means it is predictive when large enough. If it is not, the second condition (line 15) checks whether the difference in *horizontal support* values or the *mean duration* values in the supporting entities in each class are significantly different. If the *t*-test has a p-value that is lower or equal to α, then the classes are considered as different, and the TIRP is selected as predictive.

3.1.3. Saraswati-Based KarmaLego

Here, we will introduce the new algorithm for classification-driven TIRP discovery in runtime. The algorithm is demonstrated as a modification of the KarmaLego (Moskovitch and Shahar, 2015a) algorithm for time intervals mining, according to the Saraswati suite, which we call KLSD (**K**arma**L**ego Cla**S**sification **D**riven). In KarmaLego, there is a "main" function that manages the mining process, which is called KL, and here, it is called KLSD and is described by Algorithm 2. In KLSD, first, the Karma algorithm is run on each class separately, which indexes all the entities' pairs of symbolic time intervals according to their symbols and temporal relations. This stage results in two indexes for each class by applying Karma on each class' data, unlike in KarmaLego where a single index is created. Running Karma on each class separately is actually a modification needed in order to implement the Saraswati suite (Saraswati modification number 2), and it is necessary to mine the TIRPs from each class population separately. Afterwards, the LegoSD (the modification of the original Lego algorithm, in which most of the Saraswati suite is implemented) is run, consisting of the two indexes that were created by running Karma on each class.

The LegoSD is a recursive method that generates the TIRP candidates and applies the Saraswati selection method as a stopping criteria (Saraswati modification 1 and 3), as we describe below in detail. KLSD gets several input values in addition to the Saraswati selection parameters (Algorithm 1) that were introduced before. If the TIRP is predictive, the algorithm will stop expanding it depending on the look ahead parameter (explained below), and if not, it will continue recursively expanding the tree.

3.1.3.1. Look ahead. In some cases, it may be of interest to discover longer patterns (containing more components, and appearing deeper in the enumeration tree), which may be useful for knowledge discovery. For example, rather than using the patterns as features for classification, we added the L_{Ahead} parameter. It is the maximal number of levels to go down in the enumeration tree, even after a pattern was found predictive in an intermediate level. Thus, if we have a two-sized temporal pattern that was discovered as predictive and the L_{Ahead} is four, then the algorithm will not stop expanding the tree at level two, and will expand and search for predictive patterns up to the fourth level. On the other hand, if the TIRP is not predictive, the algorithm will not stop the expansion of the tree until one of the stopping criteria is satisfied. This parameter intends to increase the ability to discover predictive TIRPs, even if a TIRP was found predictive. This parameter was also developed in order to examine our acquired predictive pattern number in comparison to the BaseLine algorithm, and this will be explained further in the experiments section.

Algorithm 2 - KLSD.

Input:
db_A – A database of $\|E_A\|$ entities of class A
db_B – A database of $\|E_B\|$ entities of class B
$_{MIN}$VS – the minimal vertical support threshold
$_{MIN}$ΔVS – the minimum vertical support delta threshold between the classes
L_{Ahead} – max number of levels to go down in the tree if the TIRP is predictive
α– the Significance level of the statistical test
Strategy – the strategy to use for TIRP selection
score_threshold – TIRP predictiveness threshold
Output: T_{AB} – an extended TIRP branch of the predictive TIRPs
1. $T_A^2 \leftarrow$ Karma (db_A, min_ver_sup) // T_A^2 is class A's index of all the 2-sized TIRPs
2. $T_B^2 \leftarrow$ Karma (db_B, min_ver_sup) // T_B^2 is class B's index of all the 2-sized TIRPs
3. $T_{AB} \leftarrow \varnothing$
4. Foreach $t \in T_A^2 \cup T_B^2$
5. $T_{AB} \leftarrow T_{AB} \cup$ LegoSD (T_A^2, T_B^2, $_{MIN}$VS, t, L_{Ahead},2, $_{MIN}$ΔVS,α, Strategy, score_threshold)
6. End Foreach
7. return T_{AB}
8. End

Algorithm 3– LegoSD

Input:	
T_A^2 – the 2-sized TIRPs index after Karma was ran on class A	
T_B^2 – the 2-sized TIRPs index after Karma was ran on class B	
$_{MIN}$VS – the minimal vertical support threshold	
t – a candidate TIRP (has metrics in A or B) to be selected as predictive or to be extended in the tree	
L_{Ahead} – max number of levels to go down in the tree if the TIRP is predictive	
L – current level in the tree	
$_{MIN}$ΔVS – the minimum vertical support delta threshold between the classes	
α– the Significance level of the statistical test	
Strategy – the strategy to use for TIRP selection.	
score_threshold – TIRP predictiveness threshold	
Output: T_{AB} – an enumerated tree of all predictive TIRPs	
1. $t_A \leftarrow T_A^2$ // the 2-sized TIRPs from class A	
2. $t_B \leftarrow T_B^2$ // the 2-sized TIRPs from class B	
3. If max (t_A.vs, t_B.vs) >= $_{MIN}$VS:	
4.	isPredictiveTirp ← SarswatiFeatureSelection(t,$_{MIN}$ΔVS,α)
5.	C ← Generate_Candidate_TIRPs(t) // the candidate generation process
6.	Foreach c ∈ C // candidates
7.	search supporting instances (c, T_A^2, T_B^2)
8.	If isPredictiveTirp:
9.	$T_{AB} \leftarrow T_{AB}$ U t // t is predictive
10.	If L < L_{Ahead}:
11.	LegoSD (T_A^2, T_B^2, c, L_{Ahead}, L + 1, ΔVS_{MIN},α)
12.	End If

(*continued on next page*)

(*continued*)

Input:

```
13.          Else:
14.             Break
15.          End Else
16.       End If
17.       Else:
18.          LegoSD (T_A^2, T_B^2, c, L_Ahead, L + 1, ΔVS_MIN,α)
19.       End Else
20.    End Foreach
21. End If
22. Else:
23.          Break
24. End Else
25. Return T_AB
```

The LegoSD is a modification of the original Lego algorithm (Moskovitch and Shahar, 2015a), which is responsible for TIRP extension through candidate generations, and it corresponds to the Saraswati suite requirements. The main difference between the original Lego and the LegoSD (Algorithm 3) is the stopping criterion that determines when to extend the TIRPs and when to stop. In the original Lego, only frequent TIRPs were discovered based on their vertical support compared to a minimal vertical support. However, this algorithm searches for the predictive TIRPs based on the Saraswati selection methods. At the first line, the algorithm checks if the TIRP is supported above the minimum vertical support threshold at least in one class (either in class A or B). If it is, we can continue to examine the TIRP. First, the TIRP itself is checked whether it is predictive according to the Saraswati criteria (Algorithm 1). Then we generate all possible candidates to expand the TIRP (the same candidate generation method used for the original KarmaLego), and for each candidate TIRP, we decide whether or not to keep looking down the enumeration tree for predictive TIRPs. If a TIRP is predictive, then the tree expansion is stopped (depending on the max lookahead number of levels, which we described above). Otherwise, the procedure continues recursively expanding the enumeration tree by calling LegoSD. The purpose of stopping the branch tree from extension is to reduce the running time and prevent the algorithm from extracting redundant and irrelevant TIRPs for classification purposes. The complexity of algorithms in time intervals mining, or sequential pattern mining, complexity can not be analyzed analytically since the input data can be described in a clear structured way, which is the reason in such papers methods are compared based on their runtime duration. However, the worst case complexity can be analyzed, and KLSD's worst case analysis is the same as of KarmaLego, and was already described (Moskovitch and Shahar, 2015b) which is $O(S^L R^{L(L-1)/2} N^2)$, in which S is the number of symbols (types of time intervals), L is the maximal length allowed of a discovered TIRP (which is actually the Look Ahead in our method), R is the number of temporal relations, and N is the entire number of symbolic time intervals in the dataset. A detailed analysis of this formula is at (Moskovitch and Shahar, 2015b).

Fig. 5 illustrates the KLSD tree after selecting only the predictive TIRPs. In the figure, three enumeration trees of discovered TIRPs consisting on three symbols (A,B,C) and two temporal relations BEFORE (<) and OVERLAP (o) until level 3 (for simplicity) are shown. The top tree represents the TIRP tree discovered from class A through a regular KL process, the second tree is the TIRP tree that is discovered from class B, and the bottom illustrates the KLSD enumeration tree that will be discovered from the two classes in parallel after applying the new KLSD method, choosing only the predictive TIRPs. The minimum support for this example is 0.5 and, for simplicity, we assume that all the patterns in the second level have a vertical support greater than 0.5. In this example, the KLSD algorithm will choose a predictive TIRP if the ΔVS between the two classes is above a threshold of 0.1. For example, TIRP C < B is selected because the ΔVS = 2 and then pruned (not expanding to

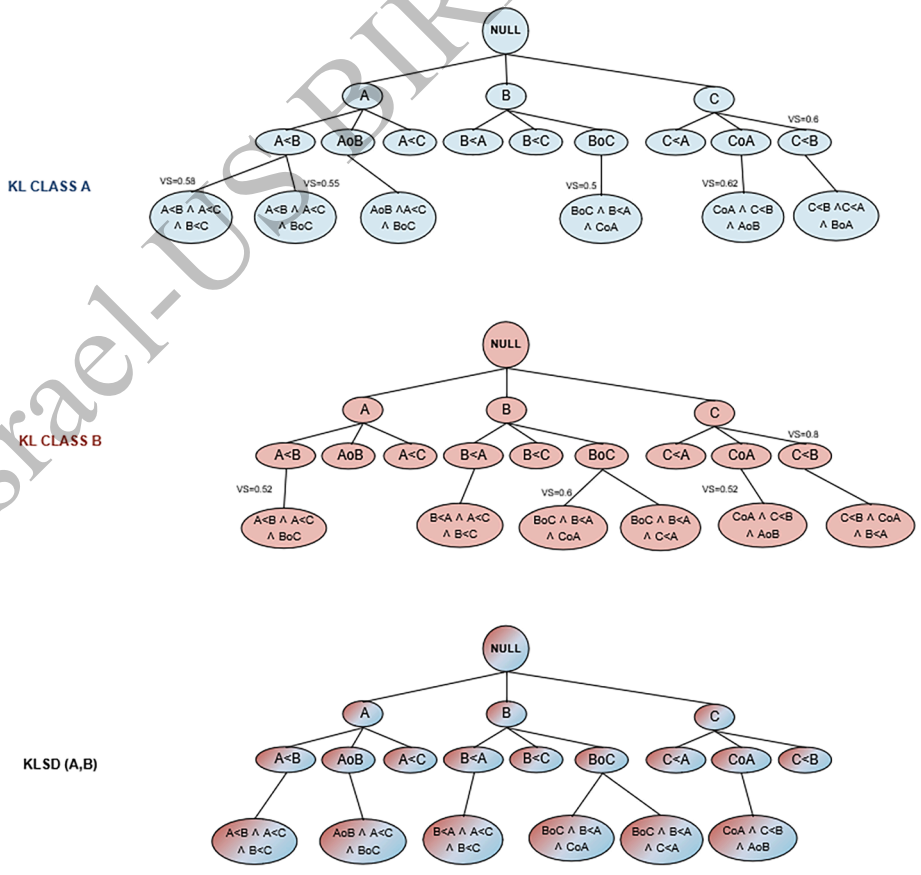


Fig. 5. KLSD tree after selecting only the predictive TIRPs.

level three). In contrast, TIRP A < B ∧ A < C ∧ BoC is not selected because the ΔVS = 0.03. Moreover, it will choose the TIRP if it only exists in one class, for example, AoB ∧ A < C ∧ BoC.

4. Evaluation

To evaluate the KLSD (**K**arma**L**ego Cla**S**sification **D**riven) Saraswati-based algorithm and all its new parameters, we designed several experiments. We first present our research questions and then our corresponding experimental plan in detail, including our hypotheses. Generally, the KLSD output was compared to a process in which TIRPs are discovered by the KarmaLego algorithm for each class separately based on the minimal vertical support, like in (Cheng et al., 2007; Batal et al., 2012; Moskovitch and Shahar, 2015b), which were described in Fig. 1. The main comparison metrics were the *runtime*, and the *percentage* of the acquired predictive TIRPs.

The KLSD and KarmaLego algorithms were developed in Python 3.6 and run in Microsoft Windows Environment using an Anaconda package. We used the classification algorithms that were already implemented in different free packages in Python, such as scikit-learn.

5. Research questions

1. What is the Saraswati-based KLSD's runtime performance and percentage of acquired predictive TIRPs in comparison to the BaseLine approach (Fig. 1), in which TIRPs are fully discovered from each class separately and then selected?
2. Do TIRP-features selected by Saraswati provide better classification performance than TIRPs selected using other common filtering feature selection methods?
3. What are the best Saraswati-based KLSD settings for classification performance?
4. Do different Saraswati strategies for predictive TIRP selection result in different classification performance?

5.1. Experimental plan

To answer the research questions, we designed three different experiments that were applied on several datasets. The experiments were ran on a machine HP Proliant DL560 Gen 8 consisting on Intel Xeon with 32 cores.:

5.1.1. Experiment 1 – Runtime vs. Acquired predictive TIRPs discovery

In this experiment, we wanted to evaluate the runtime of the Saraswati suite applied to KarmaLego, which we call KLSD, in comparison to the *BaseLine*.

5.1.1.1. Saraswati versus BaseLine evaluation. Here, we explain the ***BaseLine process*** to which the Saraswati-based modified KaramLego was compared. As in previous relevant publications, pattern-based classification was performed (Patel et al., 2008, Moskovitch and Shahar, 2015a; Fradkin and Moerchen, 2012) and is described graphically in Fig. 1. We ran KarmaLego on each class separately with the given minimum vertical support that allowed us to discover all the frequent patterns in each of the classes. Thus, we obtained two enumeration trees that contained all the frequent patterns for each class.

Then, in order to discover the full list of the predictive patterns according to the Saraswati criteria, we applied the Saraswati measures on all of the patterns, whether they appeared in both trees or only in one. This was the full list of predictive patterns to which we compared the patterns that were discovered in the Saraswati process to measure the percentage of acquired TIRPs. The Saraswati process included running KLSD with both two classes simultaneously in order to discover only the predictive patterns, given the same minimal vertical support. We set the same strategy for the Saraswati feature selection as was run in the BaseLine process (for comparison). Eventually, we compared both processes' output patterns, in which the BaseLine provides the full set of predictive patterns, and the KLSD was expected to miss some. Thus, we measured both the runtime, and the percentage of acquired predictive TIRPs. Both the KLSD and the BaseLine were run with the same parameter settings, accordingly:

a. TIRP feature selection strategy: metrics-based (HS, MND, HS_OR_MND) or SCORE
b. Saraswati score thresholds ∈ (0.5,0.6,0.7) when the score strategy was used
c. Minimum vertical support ∈ (0.4,0.45,0.5,0.55,0.6)
d. L_{Ahead} parameter values ∈ (2,3,4,5)

5.1.2. Experiment 2 – KLSD best settings for classification

While discovering predictive TIRPs is useful for knowledge discovery and for classification, here, we wanted to use the patterns as features for classification or prediction. In this experiment, we wanted to evaluate the KLSD method's parameters including: *delta vertical support*, *alpha*, max *levels ahead* in the tree, and *strategies* for their use in classification, measured by its classification accuracy. In addition to the values evaluated in Experiment 1 that included: max levels ahead and strategy, we evaluated the following values for alpha ∈ (0.1, 0.05, 0.01) and minimum delta vertical support ∈(0.1, 0.2, 0.3). We performed 10-fold cross validation classification experiments using the following state-of the art classifiers: logistic regression, random forest, and gradient boosting. Logistic regression is a statistical model that models the probability of one event (out of two alternatives) taking place by having the log-odds for the event be a linear combination of one or more independent variables. Random forest consists on a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes become model's prediction. Gradient boosting trees: built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

The 10-fold cross-validation evaluation process included the following steps: 1. Discovering predictive TIRPs based on 9-fold data, 2. Detecting the predictive TIRPs in the last fold, 3. Creating a matrix of features and a label using the predictive TIRPs as features to induce a classifier, and 4. Repeating steps 1–3 for each of the 10-fold iterations with a specific classifier.

We used four TIRP feature representations (the values that are given to the classifier) in our tests: binary, horizontal support (Definition 3), normalized horizontal support (Definition 4), and mean duration values (Definition 5).

5.1.3. Experiment 3 – Feature selection methods comparison

Although we originally designed the Saraswati suite to select patterns whose selection can be easily explained to humans, we also wanted to compare its effectiveness as a filtering feature selection method, in comparison to common feature selection methods, when using for classification. In order to use all the TIRPs, we ran the BaseLine process and then ranked the TIRPs with their Saraswati score (Definition 10). For the classification phase, we then chose the n top ranked TIRPs in each of the feature selection methods. In our experiment, we used several top n values: 10, 20, 30, and measured the performance based on the resulting classification accuracy (AUC).

We compared our feature selection to the following selection methods:

1. Pearson Correlation/BISERIAL Correlation (Benesty et al., 2009), a statistical measure of the strength of the linear relations between paired data

2. Information Gain (Kent,1983), which measures how much "information" a feature gives us about the class

5.2. Datasets

We describe here the datasets that we used for our evaluation. Some of the datasets are publicly available, and others are datasets for research purposes. All the datasets contain multivariate temporal data that were transformed into symbolic time intervals using state abstraction methods, as we mention in each dataset description. We did not experiment with different abstraction methods or the number of states since these should not affect the research questions of the study.

Epileptic mice – MEA, also known as Multi Electrode Arrays are devices for neural signal recording. This dataset consisted of 48 wells with 16 electrodes from a Columbia University lab that monitors mice brain cells. Brain cells were taken from laboratory mice with epilepsy were seeded onto a plate, in which our mission was to classify between mice who had received treatment and those they had not. The signals were discretized by the SAX method with three states. The dataset contained 1038 samples without treatment, and 544 with treatment.

Da Vinci surgical system – The JHU-ISI Gesture and Skill Assessment Working Set (JIGSAW) (Gao et al., 2014) is a surgical activity dataset for human motion modeling. The data base was recorded while performing three surgical tasks: suturing (SU), knot-tying (KT), and needle-passing (NP), performed by eight surgeons (B,C,D,E,F,G,H,I) who performed each task five times, with three different skills: D,E had more than 100 h of experience (labeled as Expert), C,F had 10–100 h (labeled as Intermediate), and B,G,H,I had less than 10 h (labeled as Novice). The data base consisted of three components: kinematic data, video data, and manual annotations. We focused on the kinematic data from the Da Vinci surgical system sampled at 30 Hz. The signals were discretized using the SAX method with three states.

Diabetes – The Diabetes dataset, provided as part of a collaboration with Clalit Health Services (Israel's largest HMO) contained data on 2,004 patients with type II diabetes (Moskovitch and Shahar, 2015b). The data were collected each month from 2002 to 2007. The dataset contains six temporal variables (laboratory values or interventions) recorded over time for each patient: hemoglobin-A1c (HbA1C) values (indicating the patient's mean blood-glucose values over the past several months), blood glucose levels, cholesterol values, several medications that the patients purchased, oral hypoglycemic agents (diabetic medications), cholesterol reducing statins, and beta blockers. The class label for this dataset was patient gender – there were 992 males and 1,012 females. Here, we used EWD for the discretization method with three states.

Hepatitis – The Hepatitis dataset contained the results of laboratory tests performed on patients who had hepatitis B or C, and who were admitted to Chiba University Hospital in Japan. and was a challenge in ECML/PKDD 2002 (Moskovitch and Shahar, 2015b). Hepatitis A, B, and C are viral infections that affect the liver of the patient. The dataset contained time-series data including laboratory tests, which were collected at Chiba, as well as administrative information, such as the patient's demographic data, pathological classification of the disease, date and result of biopsy, duration of interferon therapy, blood tests, and urinalysis. Consequently, the temporal data contained the results of 983 types of tests. We selected 11 variables which were found most frequent (occurring in most of the patients) including: glutamic-oxaloacetic transminase (GOT), glutamic-pyruvic transminase (GPT), lactate dehydrogenase (LDH), TP, alkaline phosphatase (ALP), albumin (ALB), total bilirubin (T-BIL), direct bilirubin (D-BIL), indirect bilirubin (I-BIL) and uric acid (UA). The dataset included 204 patients who had hepatitis B and 294 who had hepatitis C, which were our classes in classification. We used SAX as a discretization method with three states.

ICU – The ICU dataset contained a multivariate time series of patients who underwent cardiac surgery at the Academic Medical Center in Amsterdam, the Netherlands between April 2002 and May 2004 (Moskovitch and Shahar, 2015b). The time intervals were derived from high-frequency time series, measured every minute over the first 12 h of the ICU hospitalization. The experimental dataset included 645 patients of whom 183, or 28%, were mechanically ventilated for more than 24 h. The main classification goal was to determine whether a patient would need ventilation after 24 h, given the data of the first 12 h. We used SAX as a discretization method with three states.

Gesture Phase Segmentation – This database consisted of a temporal segmentation of gestures performed by researchers in order to preprocess videos for further analysis (Dua and Graff, 2019). The dataset is composed of seven videos recorded using Microsoft Kinect sensor. Three different users were asked to read three comic strips and to tell the stories in front of the sensor. The system delivered: (a) an image of each frame, identified by a timestamp; (b) a text file containing positions (coordinates ×, y, z) of six articulation points: left hand, right hand, left wrist, right wrist, head, and spine, with each line corresponding to a frame and identified by a timestamp. We chose two classes for our experiments: preparation (1038 instances) stroke (544 instances). We used SAX as a discretization method with three states.

6. Results

6.1. Experiment 1 – Runtime vs. Acquired predictive TIRP discovery

The goal in this experiment was to evaluate the tradeoff between the runtime of the Saraswati-based KLSD method and the percentage of the acquired predictive patterns, in comparison to the BaseLine process (which discovers the entire set of predictive patterns) with different settings of KLSD. For each dataset, we present four metrics: (a) *minimum vertical support*; (b) max *level ahead*, which determines until which level to go down in the tree after a TIRP is found predictive (explained in Section 3.2); (c) *metrics-based strategies* (HS, MND, HS_OR_MND); and (d) *score threshold* – when we used the score strategy, we evaluated it with different score thresholds. Note, in the score strategy, a pattern is selected as predictive if the pattern's score is above the score threshold that the algorithm was given.

Figs. 6–11 show the results for each dataset, and Fig. 12 shows the mean results for all datasets. Figures a–d refer to the specific parameters mentioned earlier for each experiment, and their values are presented on the x-axis. Each main figure has two y-axes. The left y-axis refers to runtime duration, and the right axis refers to the percentage of the acquired predictive TIRPs, represented by the blocks. The blue curve presents the BaseLine runtime, which was constant, since it is not affected by the Saraswati parameters, and the red curve presents the KLSD runtime. The error bars in our graphs present 95% confidence intervals for each data point. When the error bars are not noticeable, it means that they are too small.

The results of the experiment show (blue and red curves in graphs) that our new proposed Saraswati-KLSD method for predictive pattern discovery was often significantly faster (in some datasets and settings, even less than half the time) than the runtime of the BaseLine framework. However, in order to be able to discover more predictive TIRPs, we defined several parameters for the Saraswati-based algorithm, in our case KLSD, which we wanted to evaluate as well. Therefore, we evaluated various parameters of Saraswati to investigate their influence on the percentage of the acquired predictive TIRPs and the resulting runtime. We can conclude that when increasing the max levels ahead parameter value, more predictive TIRPs were acquired, and that the reason for this is very straightforward – more predictive patterns may be discovered by continuing to expand the pattern tree. When increasing the minimal vertical support threshold, fewer patterns were also discovered in the BaseLine, and consequentially, the percentage of the predictive patterns that were acquired also increased. In most datasets, when the minimal vertical support was higher, we acquired a higher percentage of predictive patterns. We found that there were no significant differences between the metrics-based strategies. In most cases, the

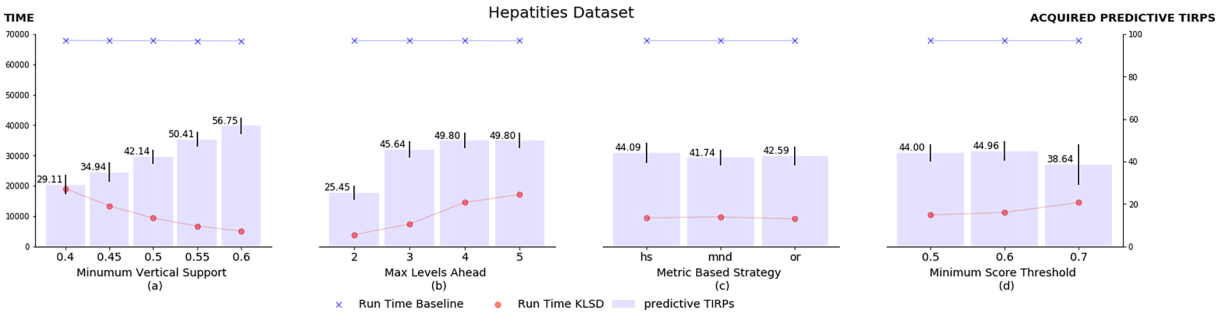


Fig. 6. In the Hepatitis dataset, in all the charts, the runtime duration of the Sarawati-KLSD was meaningfully lower than the BaseLine runtime. (a) An increase in the minimal vertical support values resulted in a meaningful increase in the percentage of the acquired predictive TIRPs and a decrease in the KLSD run time. (b) As the max level ahead increased, the runtime duration of the Saraswati-KLSD increased, and the percentage of the acquired predictive TIRPs increased. (c) No meaningful difference was found between the strategies, (d) and no meaningful difference was found between the score thresholds. On average, the score strategy acquired a higher percentage of predictive TIRPs than the metrics-based strategies.

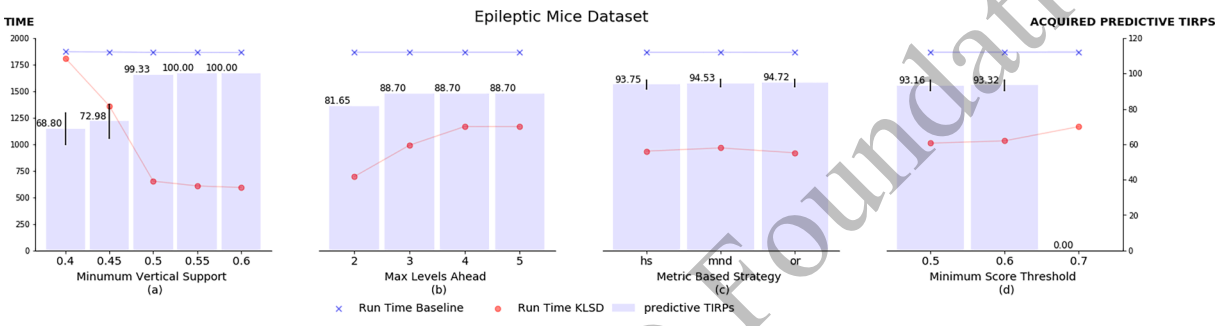


Fig. 7. In the Epileptic mice dataset, the running time of Sarawati-KLSD is meaningfully shorter than the BaseLine runtime. (a) A meaningful increase was observed in the percentage of the acquired predictive TIRPs and in KLSD running time, as minimum vertical support value was increased. (b) A large increase between max levels 2 to 3 in the percentage of the predictive TIRPs that were acquired, and an increase in runtime of KLSD as we increased the max level ahead was found. (c) No meaningful difference was seen between the strategies. (d) At the score threshold = 0.7, only one TIRP was found to be predictive and was not caught.

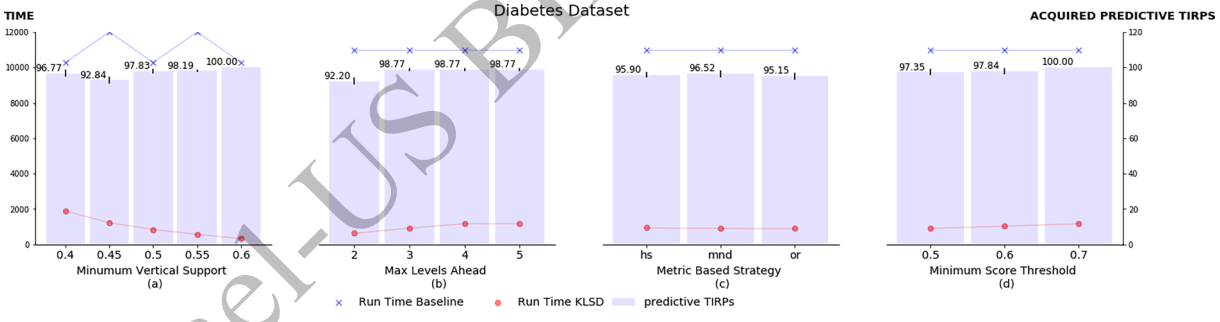


Fig. 8. In the Diabetes dataset, the runtime of Saraswati-KLSD was meaningfully shorter than the BaseLine. (a) A decrease was seen in the acquired predictive TIRPs for the minimal vertical support from 0.4 to 0.45, and afterwards an increase, while the KLSD runtime decreased as the minimal vertical support increased. (b) An increase was found between max levels 2 to 3, and then no difference between the next levels. (c) No significant difference between the strategies was observed. (d) A small increase in the acquired predictive TIRPs as the score threshold was increased occurred.

Saraswati 'score' strategy acquired the smallest number of TIRPs, which can be explained by the score threshold. Choosing predictive patterns by their score may result in far fewer patterns and, therefore, a smaller percentage of acquired predictive TIRPs, in comparison to the BaseLine process.

6.2. Experiment 2 – KLSD best settings for classification

In this experiment, we wanted to learn the best parameters of the Saraswati-based KLSD method in terms of classification performance, when the acquired TIRPs were used as features for classification. The first parameter we wanted to examine was the minimum *delta vertical support* threshold between the classes (Definition 7). The next parameter was the max *level ahead*, which allows further expansion of the enumeration tree with longer patterns that are potentially predictive. Moreover, we tested the different strategies and their impact on the results.

The last parameter we examined was the alpha, which is used to determine if a *t*-test is significant or not as part of our different strategies in selecting predictive TIRPs. Figs. 13–18 show the results for each dataset, and Fig. 19 shows the mean results for all the datasets. Figures a–d refer to the examined parameters mentioned above, whose values are presented on the x-axis. Each main figure has two y-axes. The left y-axis refers to runtime duration, and the right axis refers to the classification performance (AUC) presented in the bars.

According to the above results on the various configurations of the

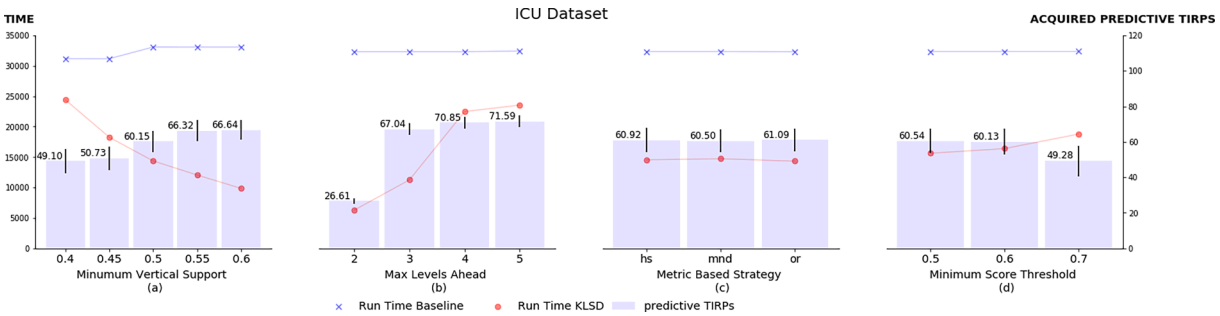


Fig. 9. In the ICU dataset, the runtime of Saraswati-KLSD was meaningfully shorter than the BaseLine. (a) An increase occurred in acquired predictive TIRPs and KLSD runtime as increasing minimum vertical support. (b) An increase was shown in the acquired predictive TIRPs and in KLSD runtime as the max level ahead parameter increased. (c) No meaningful difference was found between the metrics-based strategies. (d) A decrease in the acquired predictive patterns was seen as we increased the score threshold.

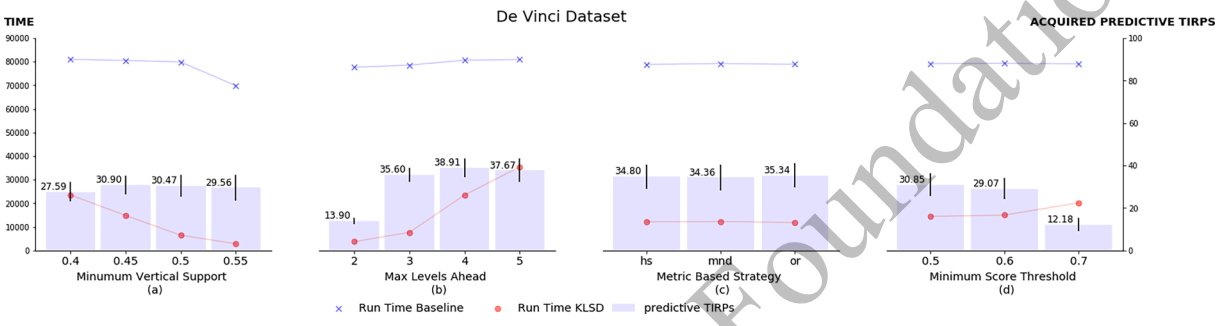


Fig. 10. In the Da Vinci dataset, the runtime of Saraswati-KLSD was meaningfully shorter than the BaseLine runtime. (a) No meaningful difference between the minimum vertical support values was found. (b) An increase was seen between levels 2 and 3 in the acquired predictive TIRPs and an increase in the running time as the max levels ahead increased. (c) There was no significant difference between the metrics-based strategies. (d) A decrease was observed in the acquired predictive TIRPs when increasing the score threshold. The 'score' strategy acquired the least in comparison to the metrics-based strategies.

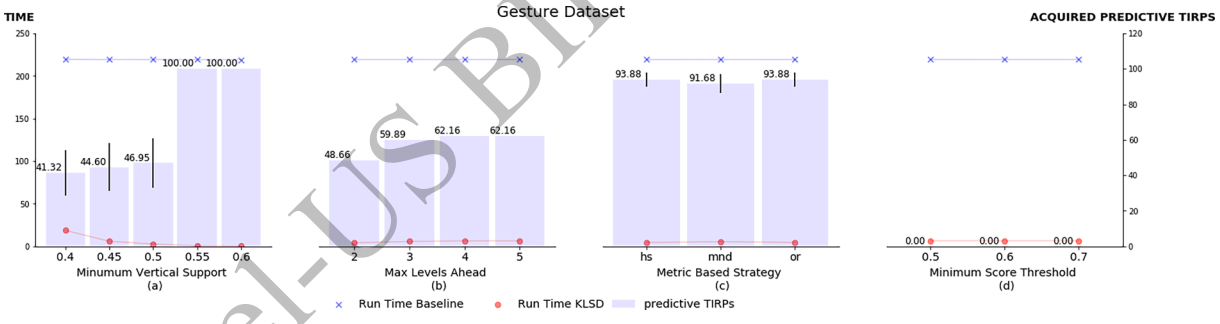


Fig. 11. In the Gesture dataset, the runtime of Sarawati-KLSD was meaningfully shorter than the BaseLine runtime. (a) An increase in the acquired predictive TIRPs was shown when increasing the minimum vertical support, along with a slight decrease in the Saraswati-KLSD runtime. (b) An increase in the acquired predictive TIRPs occurred while increasing the max level ahead parameter. (c) There was no significant difference between the metrics-based strategies, and (d) there are no predictive TIRPs in the score strategy.

Saraswati parameters, we can conclude the following: Changing the values of the minimum vertical support delta did not have much effect on the classification performance. In contrast, the max level ahead had a slight influence on the classification results. Increasing the number of levels, meaning discovering more patterns, improved the classification results. Another influencing parameter was the alpha value for the statistical tests. When we decreased this parameter, we exacerbated the test; therefore, fewer patterns were discovered as predictive. Then when we increased the alpha value, the classification performance was higher. The last parameter we checked was the Saraswati strategy for the predictive pattern selection.

6.3. *Experiment 3 – Feature selection methods comparison*

In our last experiment, we wanted to evaluate the new Saraswati feature selection against state-of-the-art filtering feature selection methods. We compared them, given several parameter settings. First, (a) we compared the average performance of the feature selection methods. In (b), three classifiers were compared. Then in (c), we compared the performance of the top n features from each of the feature selection methods. In addition, in (d), we wanted to examine the classification results while using different TIRP representations. Figs. 20–25 show the mean results on the various Saraswati parameter values, which are described in the experiment design for each dataset. Fig. 26 presents the average results of all the datasets. Figures a–d refer to the examined settings explained above, whose values are presented on the x-axis. Each

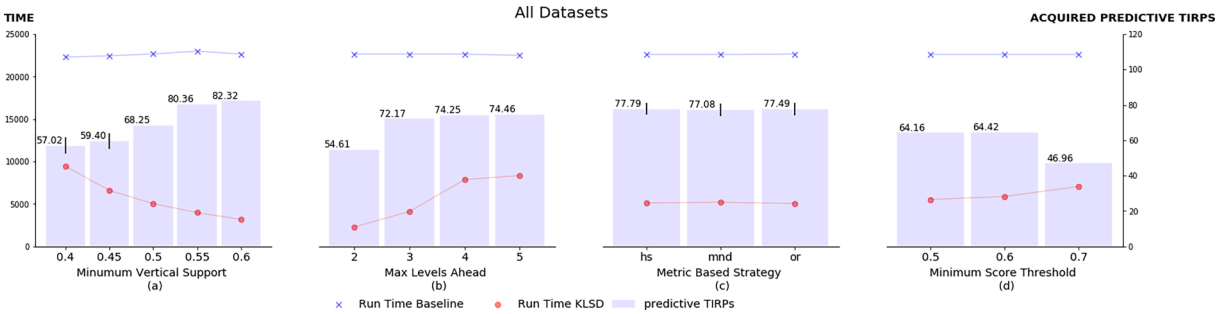


Fig. 12. On average, the runtime of the Sarawati-KLSD was meaningfully shorter than the BaseLine runtime. (a) An increase was shown in the acquired predictive TIRPs when increasing the minimum vertical support and a decrease in the Saraswati-KLSD runtime. (b) An increase in the acquired TIRPs while increasing the max level ahead parameter occurred. (c) On average, there was no significant difference between the metrics-based strategies. (d) In most cases, when using score threshold = 0.7, the results were worse. On average, the score strategy acquired the least predictive TIRPs.

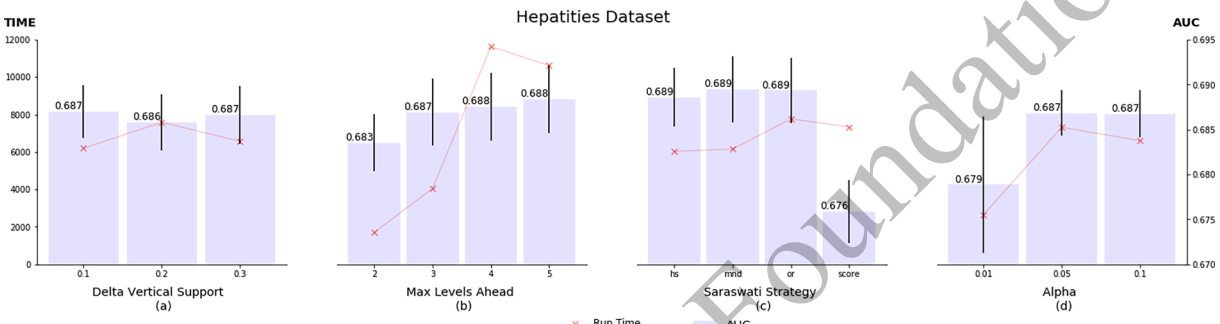


Fig. 13. Results for the Hepatitis dataset: (a) Not much difference was observed in the classification performance when increasing the delta vertical support threshold. (b) An increase in classification performance was seen when increasing the max levels ahead, and also an increase in the runtime duration. (c) The *score* strategy performed worse than the metrics-based strategies, which were quite similar. (d) A small increase in classification performance when the alpha was increased from 0.01 to 0.05 was found.

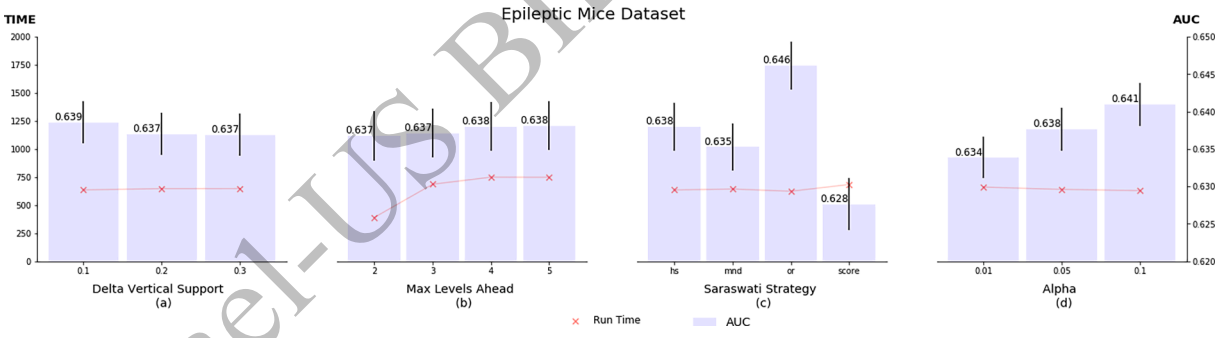


Fig. 14. Results for the Epileptic mice dataset: (a) No meaningful difference was found between the delta vertical support thresholds. (b) A small increase in classification performance occurred as we increased the max levels ahead parameter and an increase in the run-time duration. (c) The *score* strategy showed the worst results, and the *or* strategy had the best performance. (d) The results show a slight increase in classification performance and a decrease in runtime duration as the alpha value was increased.

main figure has one y-axis, which refers to the classification performance (AUC). The error bars in our graphs represent the confidence interval for 95% for each data point. If the error bars cannot be seen, it means that they are very small.

In this experiment, we evaluated Saraswati's parameters for selecting predictive patterns in comparison to state-of-the-art filtering feature selection methods. The results show that the Saraswati method is significantly better when selecting temporal patterns as features for classification. For all parameters, the Saraswati selection method achieved better classification performance, and in most datasets and parameters, it was significant. This is clear especially in Fig. 26, where the mean results over all the datasets are shown, Saraswati outperformed significantly in any parameter. This is very encouraging for two main reasons. First, it confirms that Saraswati indeed selects predictive TIRPs, since they are effective for classification. Second, it is encouraging to see that the TIRPs that were selected were based on the metrics' criterions, such as the vertical support differences, the *mean horizontal support* comparison, or the *mean duration*, which are meaningful criterions to domain experts. This is important because these features are more explainable, which was a major motivation in our design of this method.

7. Discussion and conclusions

In this study, we introduced the problem of predictive temporal pattern discovery, in which patterns are mined from two classes of data simultaneously. Moreover, the discovered predictive temporal pattern

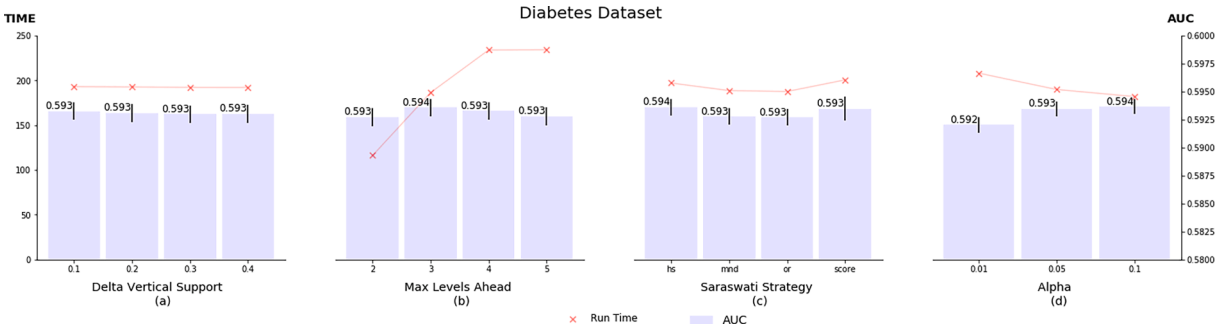


Fig. 15. Results for the Diabetes dataset: (a) Not much difference was observed in the classification performance and in the runtime when increasing the delta vertical support threshold. (b) No difference was seen between the max levels ahead values. (c) The *score* and *hs* strategies performed slightly better, and (d) there was an increase in the classification performance and a decrease in runtime as the alpha value increased.

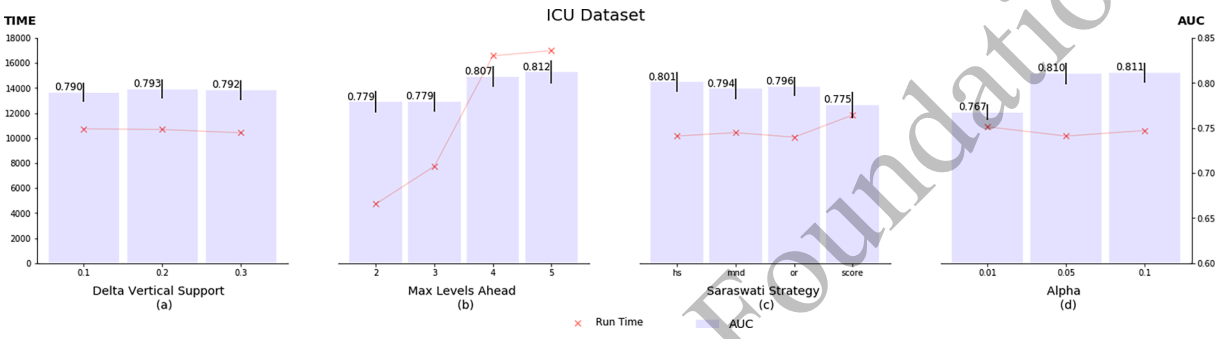


Fig. 16. Results for the ICU dataset: (a) No meaningful difference in classification performance or runtime duration was shown when the delta vertical support threshold was increased. (b) An increase in the classification performance and in the runtime duration was found as the max level ahead was increased. (c) The *score* strategy performed the poorest. (d) An increase in the classification performance occurred as we increased the alpha value.

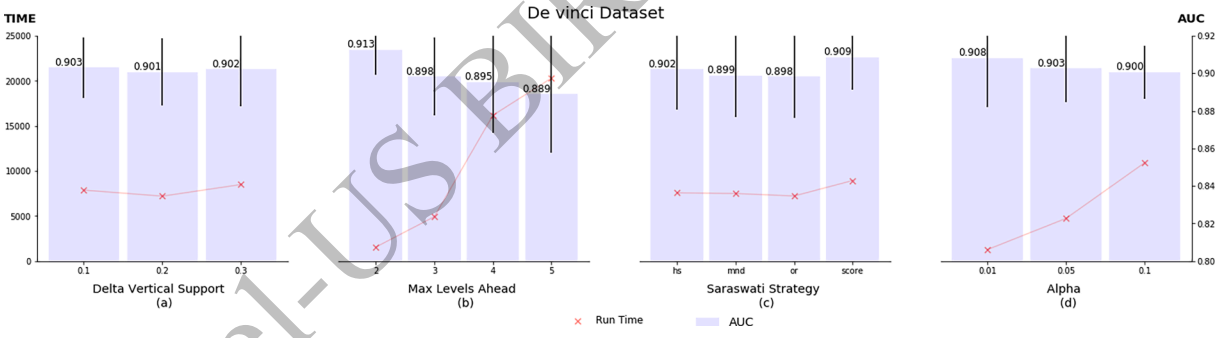


Fig. 17. Results for the Da Vinci dataset: (a) No meaningful difference in classification performance or running time as the delta vertical support threshold was increased was observed. (b) A decrease in the classification performance and in runtime duration was found as the max level ahead was increased. (c) The *score* strategy performed the best. (d) A decrease in the performance was shown as the alpha value was increased.

selection can be explained by metrics which are more clear and meaningful for humans. For the first time, as far as we know, we introduce the Saraswati suite which enables transformation of a frequent temporal pattern discovery algorithm into a predictive temporal pattern discovery algorithm. Using this suite means adding a new stopping criterion for the temporal patterns expansion in the process of pattern discovery, which commonly results in reducing their runtime, depending on the settings. In this paper, we described the suite in detail, and demonstrated its use in transforming the KarmaLego algorithm for TIRP discovery into a predictive pattern algorithm, which we call KLSD. We described KLSD in detail and presented an experimental plan and the corresponding results.

To evaluate KLSD, we designed three experiments to answer our research questions. Although we summarized the results of each experiment after the figures, we provide here a brief overview and our conclusions. In Experiment 1, we found that the mean runtime of Saraswati-KLSD was meaningfully shorter than the BaseLine runtime. An increase in the acquired predictive TIRPs percentage occurred when using higher minimum vertical support and increasing the max level ahead parameter. The mean results of the experiment show that there was no significant difference between the metrics-based strategies. In most cases, when using the score strategy, less predictive TIRPs were acquired. We propose to use a large max level ahead value and not the score strategy for selection only for ranking the TIRPs as filtering feature selection.

In Experiment 2, several Saraswati parameters were evaluated and their influence on the classification performance was observed. The results show that when increasing the delta vertical support value, there was a drop in the classification performance, and in most cases when increasing the max levels ahead, the performance improved, as well as

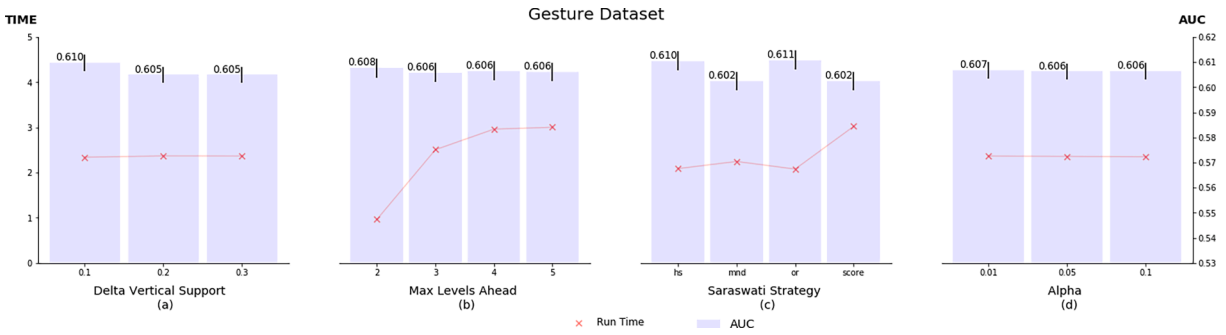


Fig. 18. Results for the Gesture dataset: (a) No significant difference in the classification performance or running time duration was detected as we increased the delta vertical support threshold. (b) No meaningful difference were found between the max levels ahead values, while there was an increase in running time duration when increasing the max levels ahead. (c) The *or* and *hs* strategies performed better in the classification. (d) No meaningful difference was observed in the classification performance or runtime when changing the alpha values.

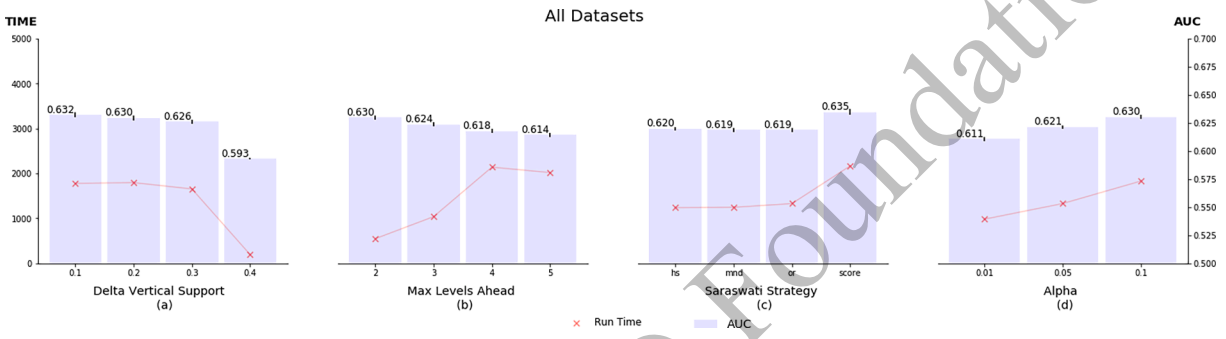


Fig. 19. In all datasets: (a) There was a decrease in the classification performance and runtime duration when increasing the delta vertical support. (b) A decrease was shown in the classification performance and an increase in the runtime when increasing the max levels ahead. (c) In general, the *score* strategy performed better in classification, and (d) a small increase was seen in the classification performance when increasing the alpha value.

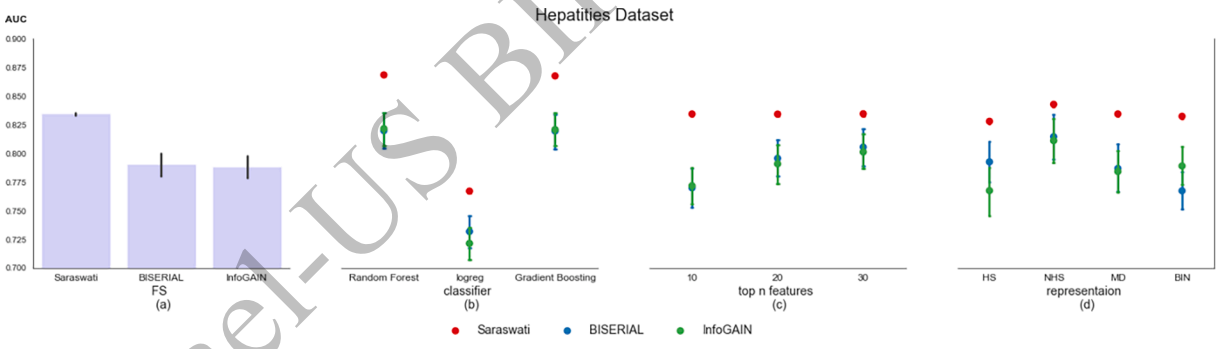


Fig. 20. In the Hepatitis dataset, (a) the Saraswati method performed significantly better than the other feature selection methods. (b) Saraswati performed significantly better than the other methods for all the classifiers, while the Logistic Regression performed the worst. (c) Saraswati performed significantly better across all top n features, while being stable, and the other improved with the increase in the number of features. (d) Saraswati performed the best across the different representation methods.

when using a higher alpha value.

The goal of Experiment 3 was to compare our Saraswati criteria for selection of predictive TIRPs against common filtering selection methods. The results show that the performance of the Saraswati criteria was superior in all the examined parameters such as the classifier, top N features, and pattern representation.

We have shown that the Saraswati criteria has a strong ability to choose predictive patterns, even when compared to statistical feature selection methods. In addition, we presented KLSD, which demonstrates the employment of the Saraswati suite on the KarmaLego algorithm for TIRP discovery. According to our results, the KLSD algorithm was shown to be faster than the BaseLine process, while acquiring predictive patterns. Moreover, when using the predictive TIRPs as features for classification, the classifier performed better than when using other feature selection methods. We conclude that the parameters that most influenced the runtime and the acquired predictive pattern percentage are the max level ahead and minimal vertical support.

The limitations of this work are that the method works for binary classification, although it could be expanded for use with more classes by employing this method on pairs of classes and constructing from them patterns that are separable for all the classes. There are quite a few approaches for extending binary classifiers in the literature (Lingras and Butz, 2007; Santhanam et al., 2016), which may be suitable. Additionally, in this study, the method was demonstrated only for use in TIRP mining with the KarmaLego algorithm, and not with other types of temporal pattern discovery method. However, the principle should work

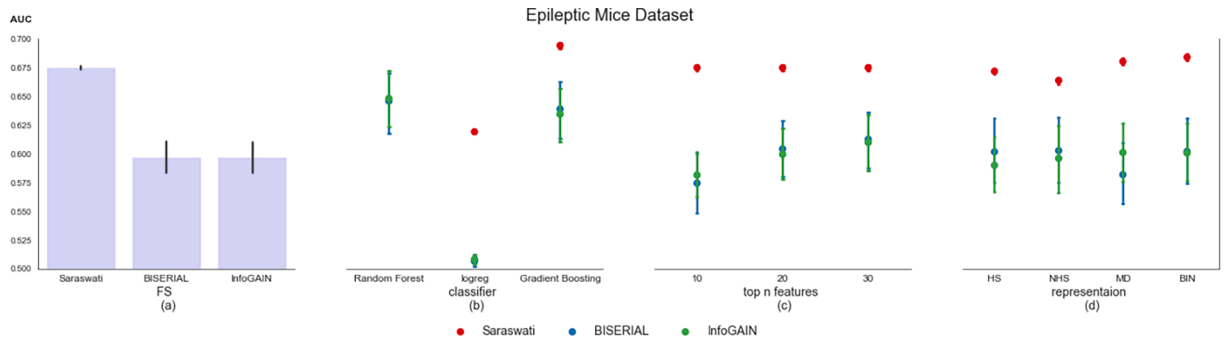


Fig. 21. For the Epileptic mice dataset, (a) overall, the Saraswati method was better than the other methods, (b) with Logistic Regression and Gradient Boosting, and the Saraswati method was significantly better compared to InfoGAIN and BISERIAL. (c) Saraswati significantly outperformed across all top n features, and (d) Saraswati significantly outperformed the other methods for all TIRP representations.

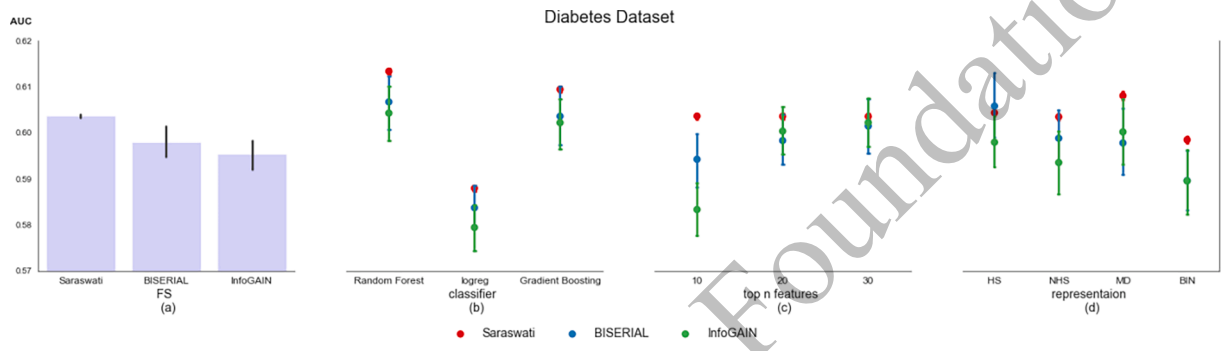


Fig. 22. In the Diabetes dataset, (a) the Saraswati method was significantly better than the other methods. (b) With all classifiers, the Saraswati method performed better than the other methods, but not significantly. (c) Saraswati performed better, but was significant only when using the top 10 features (TIRPs), while it was stable. (d) Saraswati performed best for all representations, except for the *HS*.

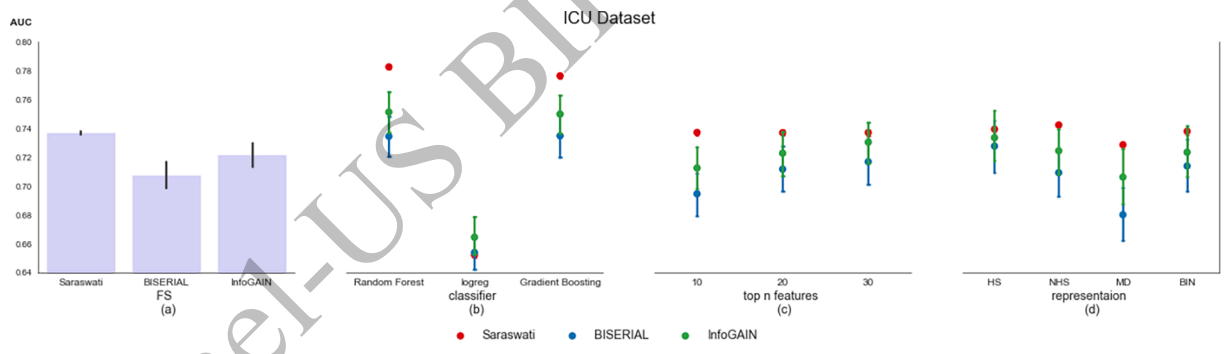


Fig. 23. In the ICU dataset, (a) the Saraswati method performed significantly better than the other methods. (b) With the Gradient Boosting and Random Forest classifiers, the Saraswati method performed significantly better, compared to InfoGAIN and BISERIAL. (c) Saraswati performed the best across all top n features, and with the top 10, it was significant. (d) Saraswati performed better, and the representations were quite similar with some advantage to the *NHS*.

and, in future work, we intend to adapt the metrics to other temporal patterns.

We found that max level ahead, the strategy parameter, and the alpha value for the statistical tests had the largest impact on the classification performance in general. Although much research has been done in the field of temporal pattern mining and discovery of predictive patterns, the research in this field has not been exhausted, and we suggest several directions in which to extend this work. First, the Saraswati criteria are designed for a binary classification task. Although, most of the classification tasks in real life are typically binary, we think that it is worth expanding Saraswati to handle multi-class problems, although common approaches for multi-classification based on binary classifiers exist (Lingras and Butz, 2007; Santhanam et al., 2016). While it is obvious that the patterns discovered by Saraswati are more explainable (due to differences in the metrics' values in the classes, such as the differences in vertical support values of the classes, and the horizontal support and averaged mean duration) and meaningful using the temporal patterns' descriptive metrics, such as the horizontal support and the mean duration, we would like to test it with domain experts. For example, we can explain that a pattern is predictive since it is more frequent in one class, or that its mean horizontal support is larger, etc.; thus, we intend to evaluate their expandability with domain experts. For that, we would like first to design a user interface that will enable the enumeration tree of the TIRPs to be browsed, presenting the patterns and enabling exploration of the tree. While we proposed a score for the selection of the predictive pattern features, another approach might be extracting the most predictive pattern features based on a trained classifier, which can be demonstrated in future work. Along those lines, it

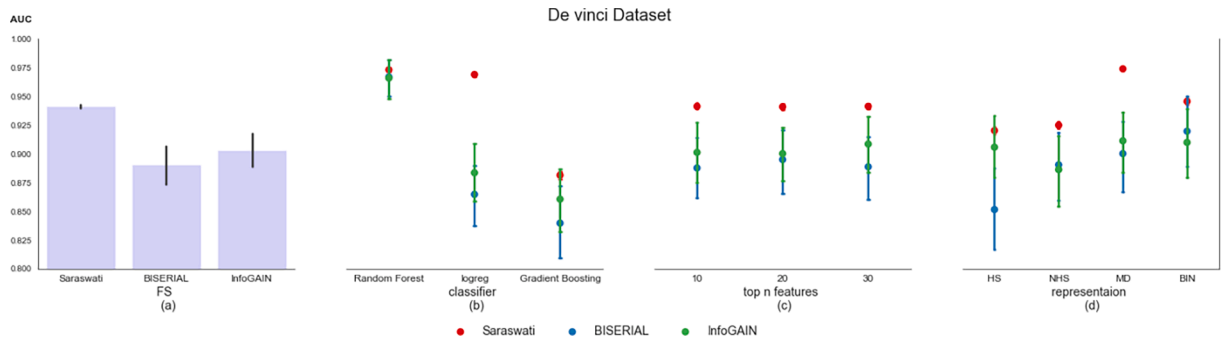


Fig. 24. In the Da Vinci dataset, (a) the Saraswati method was significantly better than the other methods. (b) Saraswati performed meaningfully better than the other methods with the Logistic Regression classifier. (c) Saraswati performed the best across all top n features in a significant way. (d) Saraswati performed better, and the representations were quite similar with some advantage for MD.

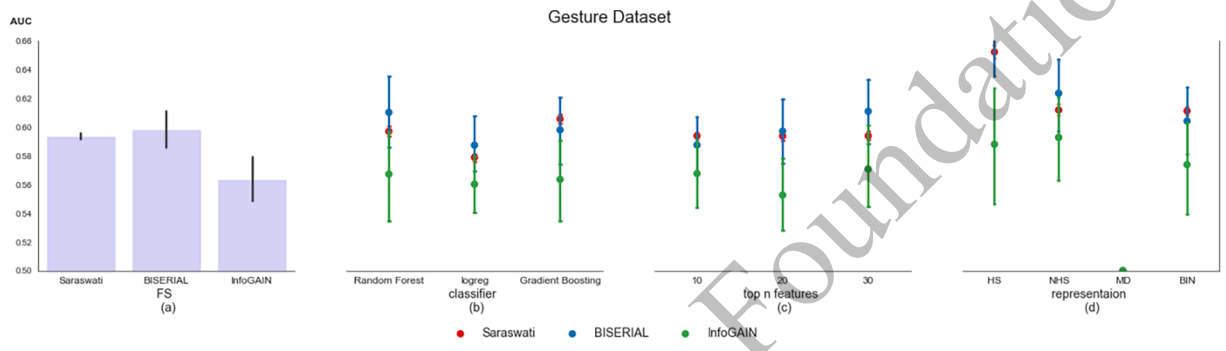


Fig. 25. In the Gesture dataset, (a) overall, the BISERIAL method was slightly better than Saraswati, but not significantly, while they both performed better than the InfoGain. (b) The BISERIAL performed slightly better than Saraswati in most classifiers except for Gradient Boosting, and none were significant (c) For all n in top features, the BISERIAL method was slightly better, except for the 10 top features. (d) The BISERIAL performed better, except for the BIN, but always not significantly.

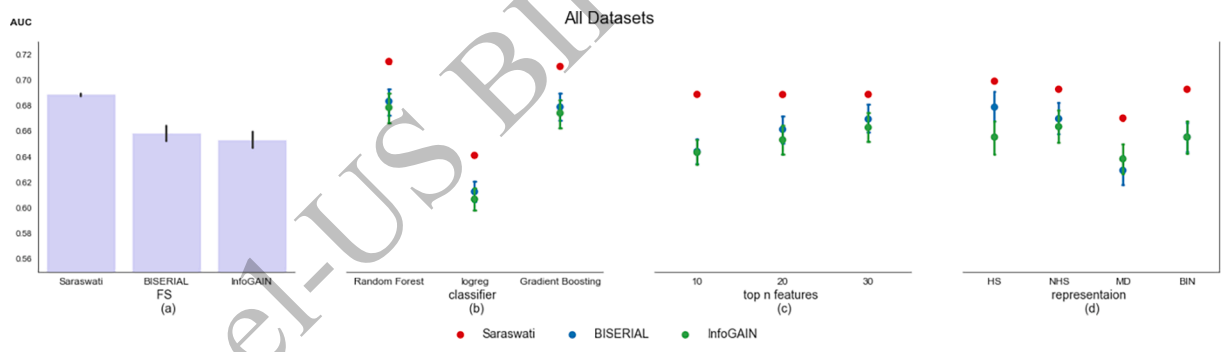


Fig. 26. In the mean results, (a) the Saraswati method was significantly better than the other methods for all datasets. (b) The Saraswati method significantly outperformed for all classifiers in general. (c) Saraswati performed the best across all top n features, and (d) Saraswati significantly outperformed in all TIRP presentations.

would be more suitable to use simple classifiers, such as decision trees or naïve Bayes, and to see, based on their structure, which are the most predictive patterns.

CRediT authorship contribution statement

Nofar Sarafian Ben Ari: Methodology. **Robert Moskovitch:** Methodology, Resources.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

We would like to express our thanks to Stav Sapir, Maya Schvetz, and Tal Ivshin for taking part in the BaseLine implementation and sharing their datasets for this study. The authors also wish to thank Tal Zeevi for his help and consolation with Python visualization. Finally, we would like to thank the Israeli Ministry of Science and Technology, who assisted in funding this project with grants 3-14360 and 3-14428.

References

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases. VLDB, 1215*, 487–499.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM, 26*(11), 832–843.

Ayres, J., Flannick, J., Gehrke, J., & Yiu, T. (2002). Sequential pattern mining using a bitmap representation. In *In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and data Mining* (pp. 429–435).

Batal, I., Fradkin, D., Harrison, J., Moerchen, F., and Hauskrecht, M., (2012), Mining recent temporal patterns for event detection in multivariate time series data, In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 280–288.

Bekkerman, R., El-Yaniv, R., Tishby, N., & Winter, Y. (2003). Distributional word clustersvs. words for text categorization. *Journal of Machine Learning Research, 3 (Mar):11831208*, 25.

Benesty, J., Chen, J., Huang, Y., & Cohen, I. (2009). Pearson correlation coefficient. In *Noise reduction in speech processing* (pp. 1–4). Springer.

Cheng, H., Yan, X., Han, J., & Hsu, C. (2007). Discriminative frequent patternanalysis for effective classification. In *In 2007 IEEE 23rd International Conference on Data Engineering* (pp. 716–725).

Dhillon, I. S., Mallela, S., Kumar, R., (2003) A divisive information-theoreticfeature clustering algorithm for text classification. *Journal of Machine Learning Research*, 3 (Mar):1265–1287.

Dvir, O., Wolfson, P., Lovat, L., & Moskovitch, R. (2020). *Falls Prediction in Care Homes Using Mobile App Data Collection*. Minneapolis, USA: Artificial Intelligence in Medicine.

Dua, D., & Graff, C. (2019). *UCI Machine Learning Repository (http://archive.ics.uci.edu/ml)*. Irvine, CA: University of California, School of Information and Computer Science.

Fradkin, D., & Mörchen, F. (2015). Mining sequential patterns for classification. *Knowledge and Information Systems, 45*(3), 731–749.

Gao, Y., Vedula, S. S., Reiley, C. E., Ahmidi, N., Varadarajan, B., Lin, H. C., Tao, L., Zappella, L., Bejar, B., Yuh, D. D., Chen, C., Vidal, R., Khudanpur, S., Hager, G. D., (2014) Jhu-isi gesture and skill assessment working set (jigsaws): a surgical activity dataset for human motion modeling. *MICCAI Workshop: M2CAI.* Vol. 3.

Han, J., Pei, J., & Yin, Y. (2000). (2000) Mining frequent patterns without candidate generation. *In ACM SIGMOD Record, 29*, 1–12.

Harel, O., & Moskovitch, R. (2021). *Complete Closed Time Intervals-Related Patterns Mining, The 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*. Canada: Vancouver.

Höppner. F., (2001) Learning temporal rules from state sequences. In *IJCAI Workshop on Learning from Temporal and Spatial Data*, volume 25.

Itzhak, N., Nagori, A., Lior, E., Schvetz, M., Lodha, R., Sethi, T., et al. (2000). *Acute Hypertensive Episodes Prediction*. Minneapolis, USA: Artificial Intelligence in Medicine.

Kent, J. T. (1983). Information gain and a general measure of correlation. *Biometrika, 70* (1), 163–173.

Kohavi, R., & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence, 97*(1), 273–324.

Lesh, N., Zaki, M. J., & Oglhara, M. (2000). Scalable feature mining for sequential data. *IEEE Intelligent Systems and Their Applications, 15*(2), 48–56.

Lin, J., Keogh, E., Wei, L., & Lonardi, S. (2007). Experiencing sax: A novel symbolic representation of time series. *Data Mining and Knowledge Discovery, 15*(2), 107–144.

Lingras, P., & Butz, C. (2007). Rough set based 1-v-1 and 1-vr approaches to support vector machine multi-classification. *Information Sciences, 177*(18), 3782–3798.

Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. *In Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*.

Mörchen, F., Ultsch, A., (2005) Optimizing time series discretization for knowledge discovery. In *Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 660–665.

Mörchen, F., Ultsch. A., (2005) Optimizing time series discretization for knowledge discovery. *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*.

Moskovitch, R., & Shahar, Y. (2015a). Classification of multivariate time series via temporal abstraction and time intervals mining. *Knowledge and Information Systems, 45*(1), 35–74.

Moskovitch, R., & Shahar, Y. (2015b). Classification-driven temporal discretization of multivariate time series. *Data Mining and Knowledge Discovery, 29*(4), 871–913.

Moskovitch, R. (2022). Multivariate Time Series Mining, *Wiley's Data Mining and Knowledge Discovery*.

Moskovitch, R., Wang, F., Walsh, C., Hripcsak, G., & Tatonetti, N. (2015). *Prediction of Outcome Events via Time Intervals Mining, IEEE International Conference on Data Mining (ICDM)*. USA: Atlantic City.

Moskovitch, R., Choi, H., Hripcsak, G., & Tatonetti, N. (2016). Prognosis of clinical outcomes with temporal patterns and experiences with one class feature selection. *IEEE/ACM Transactions on Computational Biology and*. Bioinformatics.

Novitski, P., Cohen, C., M., Karasik, A., Shalev, V., Hodik, G., Moskovitch, R., (2022) All Cause Mortality Prediction in T2D Patients with iTirps, Artificial Intelligence in Medicine.

Papapetrou, P., Kollios, G., Sclaroff, S., & Gunopulos, D. (2009). Mining frequent arrangements of temporal intervals. *Knowledge and Information Systems, 21*(2), 133.

Patel, D., Hsu, W., Lee, M., L., (2008) Mining relationships among interval-based events for classification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 393–404.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., et al. (2004). Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering, 16*(11), 1424–1440.

Ramírez-Gallego, S., García, S., Mouriño-Talín, H., Martínez-Rego, D., Bolón-Canedo, V., Alonso-Betanzos, A., et al. (2016). Data discretization: Taxonomy and big data challenge, *Wiley Interdisciplinary Reviews. Data Mining and Knowledge Discovery, 6*(1), 5–21.

Santhanam, V., Morariu, V. I., Harwood, D., & Davis, L. S. (2016). A non-parametric approach to extending generic binary classifiers for multi-classification. *Pattern Recognition, 58*, 149–158.

Shahar, Y. (1997). A framework for knowledge-based temporal abstraction. *Artificial Intelligence, 90*(1–2), 79–133.

Shknevsky, A., Shahar, Y., & Moskovitch, R. (2017). Consistent discovery of frequent interval-based temporal patterns in chronic patients' data. *Journal of Biomedical Informatics, 75*, 83–95.

Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In Apers, P., Bouzeghoub, M., Gardarin, G. (eds.), Advances in Database Technology — EDBT '96. EDBT 1996. Lecture Notes in Computer Science, vol. 1057. Springer, Berlin, Heidelberg.

Torkkola, K. (2003). Feature extraction by non-parametric mutual information maximization. *Journal of Machine Learning Research, 3(Mar):1415–1438*.

Tseng, V. S., & Lee, C. (2009). Effective temporal data classification by integrating sequential pattern mining and probabilistic induction. *Expert Systems with Applications, 36*(5), 9524–9532.

Welch, B. L. (1947). The generalization of student's' problem when several different population variances are involved. *Biometrika, 34*(1/2), 28–35.

Weston, J., Elisseeff, A., Schölkopf, B., & Tipping, M. (2003). Use of the zero-norm with linear models and kernel methods. *Journal of Machine Learning Research, 3(Mar): 1439–1461*.

Zaki, M. J. (2001). Spade: An efficient algorithm for mining frequent sequences. *Machine Learning, 42*(1–2), 31–60.

Zhou, C., Cule, B., & Goethals, B. (2016). Pattern based sequence classification. *IEEE Transactions on Knowledge and Data Engineering, 28*(5), 1285–1298.