Published in Image Processing On Line on 2022–11–23.
Submitted on 2022–10–06, accepted on 2022–10–11.
ISSN 2105–1232 © 2022 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2022.437

Association Rules Discovery of Deviant Events in Multivariate Time Series: An Analysis and Implementation of the SAX-ARM Algorithm

Axel Roques[1,2,3], Anne Zhao[1,4]

[1]Université Paris-Saclay, ENS Paris-Saclay, Centre Borelli, Gif-sur-Yvette, France
[2]Thales AVS France, Training & Simulation, Osny
[3]Laboratoire de Génomique, Bioinformatique et Chimie Moléculaire (EA 7528), CNAM
[4]Autorité des Marchés Financiers
{axel.roques, anne.zhao}@ens-paris-saclay.fr

Communicated by Quentin Bammey *Demo edited by* Axel Roques and Anne Zhao

Abstract

In this work, we propose an open-source Python implementation of the SAX-ARM algorithm introduced by Park and Jung (2019). This algorithm mines association rules efficiently among the deviant events of multivariate time series. To do so, the algorithm combines two existing methods, namely the Symbolic Aggregate approXimation (SAX) from Lin et al. (2003) – a symbolic representation of time series – and the Apriori algorithm from Agrawal et al. (1996) – a data mining method which outputs all frequent itemsets and association rules from a transactional dataset. A detailed description of the underlying principles is given along with their numerical implementation. The choice of relevant parameters is thoroughly discussed and evaluated using a public dataset on the topic of temperature and energy consumption.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Usage instructions are included in the archive.

Keywords: association rule mining; deviant event discovery; multivariate time series; Apriori algorithm; Symbolic Aggregate approXimation (SAX)

[1]https://doi.org/10.5201/ipol.2022.437

1 Introduction

There is an increasing demand to collect and extract relevant information from dynamic systems. The advances in sensor technology and information retrieval and storage have made it possible to obtain robust, objective and indexed measurements of industrial, medical or financial data. Various data mining approaches have been developed to mine useful information from the resulting databases. Notably, the problems of anomaly detection – or perhaps more generally of pattern discovery – and of Association Rules Mining (ARM) in time series are essential for knowledge discovery. Such problems are not trivial in the case of large databases or with multivariate time series, i.e. when many variables describing the same complex process or processes are measured simultaneously.

Park and Jung [12] proposed the SAX-ARM algorithm, a novel method to derive meaningful information from large time series. More precisely, the SAX-ARM finds association rules between deviant events in multivariate time series. Here, deviant events are defined as anomalies in the sense of unusual or surprising events. They used a combination of two state of the art data mining techniques: the Symbolic Aggregate approXimation (SAX) representation of time series [10, 11] and the Apriori Algorithm [1] for the detection of association rules. Briefly, a real-valued multivariate time series is transformed into its symbolized counterpart using the SAX algorithm. Then a list of multidimensional deviant events called the *symbol baskets* is computed and frequent association rules are mined from this set of transaction-like objects.

SAX is a technique that converts a real-valued time series into a discrete representation. The SAX representation relies on the Piecewise Aggregate Approximation from Keogh et al. (2001) [8] to reduce the dimensionality of the original time series to a string of arbitrary length w. Apriori is an algorithm first introduced in [1] that aims to find association rules between items in a database of sales transactions.

This paper is organized as follows. Section 2 describes the theoretical grounds of the methods and techniques involved in the SAX-ARM algorithm and their numerical implementation. A brief tutorial on the online demo of this article is given in Section 3. In Section 4, the SAX-ARM algorithm is evaluated against a real-world dataset and the effect of its different parameters is analyzed.

2 Methods

The SAX-ARM algorithm can be applied to a single multivariate time series or multiple univariate time series if all individual time series share the same timestamps and have the same length. The core of the SAX-ARM algorithm comprises three main steps:

- SAX: transforms the time series into a symbolic representation (Section 2.1).
  - Inverse Normal Transformation (INT): time series normalization (Section 2.1.1).
  - Piecewise Aggregate Approximation (PAA): dimensionality reduction (Section 2.1.2).
  - Discretization: transforms the PAA representation into a string of symbols (Section 2.1.3).
- Symbol basket generation: transforms deviant events into transaction-like data (Section 2.2).
- ARM: mines association rules in the symbol baskets using the Apriori algorithm (Section 2.3).

For simplicity sake, in the rest of this article, we will consider a unique multivariate time series $S = (X_1, \dots, X_N)$ of N time series $X_i = (x_1, \dots, x_n)$ of equal length n.

2.1 Symbolic Aggregate approXimation (SAX)

The SAX representation is a discrete representation of a real-valued univariate time series. One of the most appealing characteristics of the SAX method is that it allows to define a distance measure on the symbolic representation that lower-binds a distance measure defined on the original data. This property ensures that the transformed representation reflects the original similarity or dissimilarity in the data. The importance of this lower-bounding property was first put forward by Faloutsos et al. [5]. It guarantees that the method does not produce any false dismissals. Mathematically, let Q and C be two real-valued time series of the same length n and D a distance function (e.g. the Euclidean distance). To guarantee no false dismissals, a distance function $\tilde{D}$ on the symbolic representation $\tilde{Q}$ and $\tilde{C}$ of Q and C respectively should verify the following formula

$$\tilde{D}(\tilde{Q}, \tilde{C}) \leq D(Q, C). \tag{1}$$

The idea is that if Q and C are similar, then $\tilde{Q}$ and $\tilde{C}$ should also be similar. If we define ϵ the tolerance, this translates into

$$D(Q, C) \leq \epsilon \Rightarrow \tilde{D}(\tilde{Q}, \tilde{C}) \leq \epsilon, \tag{2}$$

which is trivial since

$$\tilde{D}(\tilde{Q}, \tilde{C}) \leq D(Q, C) \leq \epsilon. \tag{3}$$

The SAX transformation process involves three successive steps that will be detailed in the following sections.

2.1.1 Inverse Normal Transformation (INT)

As mentioned by Keogh et al. [9], it is meaningless to compare time series with different offsets and amplitudes. Therefore, normalization is often conducted on each univariate time series before PAA and SAX conversion when working with multivariate time series. This ensures similar PAA coefficient distributions across all univariate time series. Traditionally, this normalization is achieved using the z-score normalization [10]. However, z-score normalization is ill-suited for skewed data [12]. In this work, Park and Jung rather chose to adopt another normalization technique, an Inverse Normal Transformation (INT).

INTs transform the sample distribution of a continuous variable into an approximated normal distribution. There exist numerous variants of INTs (see [2] for an overview) but the most commonly used is the Blom transformation [3], a deterministic rank-based INT. This technique entails converting samples to ranks and then transforming back the ranks to the approximate normal scores.

The Blom transformation is used in the SAX-ARM algorithm. Let x_i^{INT} be the transformed i^{th} observation from a univariate time series. The value x_i^{INT} can be computed as follows

$$x_i^{INT} = \Phi^{-1}\left(\frac{r_i - c}{n - 2c + 1}\right), \tag{4}$$

where Φ^{-1} denotes the standard normal quantile function, r_i is the ordinary rank of the i^{th} observation in the time series, and c is a constant value equal to 3/8 according to the Blom variant [3].

2.1.2 Piecewise Aggregate Approximation (PAA)

PAA is a data transformation technique introduced by Keogh et al. [8] that reduces the dimensionality of the data. PAA constructs a piecewise-constant approximation of the original sequence through a partition of the data into equi-length segments and reports the mean value in each segment. Mathematically, a time series is reduced from n to w samples by dividing the data into w frames and constructing a vector $\bar{X} = \bar{x_1}, \ldots, \bar{x_w}$ with the mean values of each segment. The i^{th} element of $\bar{X}$ may be calculated as follows

$$\bar{x_i} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} x_j. \tag{5}$$

Implicitly here, we assumed that w was a factor of n. This is not a requirement. Rather than assigning whole data points to each segment, we can assign only parts of a data point into two successive segments such that each segment receives the same (floating point) number of values. For instance, if 10 data points were to be divided into $w = 3$ segments, each segment should receive $\frac{10}{3}$ data points. To do so, the 4^{th} point should contribute to the first segment with a weight of $\frac{1}{3}$ and to the second segment with a weight of $\frac{2}{3}$. Similarly, the 7^{th} point should contribute to the second segment with a weight of $\frac{2}{3}$ and to the third segment with a weight of $\frac{1}{3}$. Figure 1 illustrates this idea.

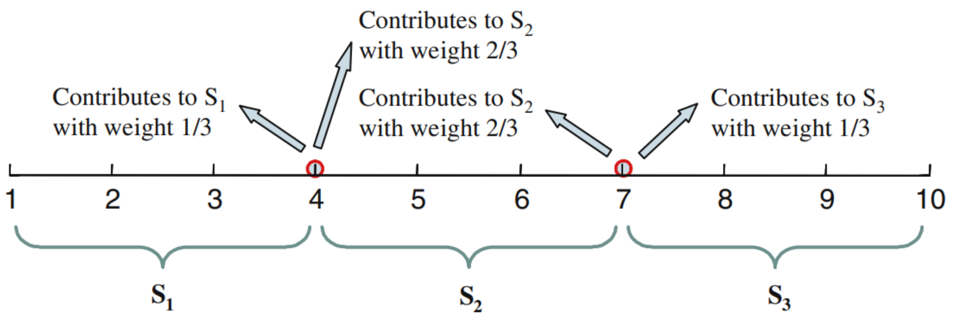
Contributes to S1 with weight 1/3
Contributes to S2 with weight 2/3
Contributes to S2 with weight 2/3
Contributes to S3 with weight 1/3
1
2
3
4
5
6
7
8
9
10
S1
S2
S3

Figure 1: When n is not divisible by w, each segment receives a non-integer number of data points. In this example, $n = 10$ and $w = 3$. The points circled in red contribute to two segments simultaneously. Adapted from [11].

Algorithm 1 gives an overview of the PAA algorithm.

2.1.3 Discretization

The discretization step of the SAX algorithm transforms the PAA representation into a string of symbols. The general idea behind this step is to define $a-1$ breakpoints $\beta_1, \ldots, \beta_{a-1}$ – with a a user-defined parameter, which determines a total of a regions that will act as the basis for the discretization process. To paraphrase Lin et al. [11]: "All PAA coefficients that are below the smallest breakpoint are mapped to the first symbol of the symbolic alphabet, all coefficients greater than or equal to the smallest breakpoint and less than the second smallest breakpoint are mapped to the second symbol" and so forth.

The two most common ways for defining breakpoints are by following a Gaussian or a uniform distribution. Gaussian breakpoints are chosen such that the area under a $\mathcal{N}(0,1)$ Gaussian curve from β_i to β_{i+1} is $1/a$, with $\beta_0 = -\infty$ and $\beta_a = \infty$. In the uniform case, breakpoints are uniformly distributed. Figure 2 shows an example of a discretization with two Gaussian breakpoints. In our implementation, we follow Park and Jung's work [12] and use Gaussian breakpoints. Since the time series were normalized beforehand in the INT step, our data follow a Gaussian distribution, hence choosing Gaussian breakpoints guarantees the equiprobability of each symbol. This, in turn, allows us to set a threshold to define what is considered a deviant event by carefully choosing the value of the parameter a. Indeed, a represents the size of the alphabet from which the symbols are drawn and usually defines the precision desired in the symbolic representation. In our case however, because each symbol is equiprobable, this parameter also defines which proportion of the data will be classified as deviant events. As we will see in Section 2.2, deviant events correspond to the first and last symbols

Algorithm 1: Piece-wise Aggregate Approximation (PAA)

```
1  function PAA(df, w)
       Input df: DataFrame of $N$ INT-normalized time series (columns) of equal length $n$
                 (rows).
       Input w: Number of desired equal-length segments for the temporal segmentation.
       Output df_PAA: DataFrame of the input time series approximated using the PAA
                 method.
2
3      Each (univariate) time series is processed independently.
4      for $X \in$ df do
5          $n \coloneqq$ length(X)
6          If n is a multiple of w, each of the w fragments receives an equal,
            whole number of samples.
7          if $n \equiv 0 \pmod{w}$ then
8              $l \coloneqq n \mathbin{//} w$
9              for $i$ from 0 to $w$ do
10                 $X_{PAA} \coloneqq \mathrm{mean}(X[i * l : (i+1) * l])$
11         If n is not a multiple of w, each of the w fragments receives an equal,
            floating point number of samples.
12         else
13             for $i$ from 0 to $n * w$ do
14                 $i_{segment} \coloneqq i \mathbin{//} n$
15                 $i_{val} \coloneqq i \mathbin{//} w$
16                 $X_{PAA}[i_{segment}] += X[i_{val}]$
17             $X_{PAA} \coloneqq X_{PAA}/n$
18     $df_{PAA} \coloneqq$ concatenate($X_{PAA}$)
19     return $df_{PAA}$
```

of the alphabet, whose occurrence is dictated by the value of parameter a. For example, any sample with a value in the top and bottom 10% of the data's probability distribution tails is considered as a deviant event when $a = 10$. Note that those 10% correspond to the normalized distribution and not the original data distribution.

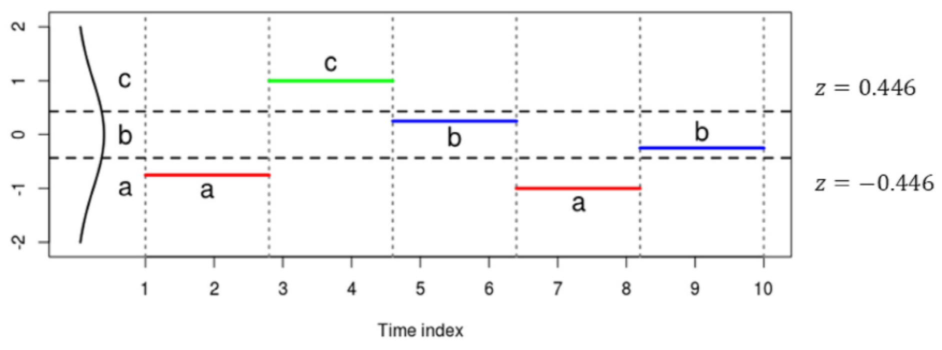

Figure 2: Example of a Gaussian discretization step from a PAA representation for $a = 3$. The z-values on the right-hand side define the values of the corresponding breakpoints. Adapted from [12].

2.2 Symbol Baskets

In order to mine associations rules, the data has to be transformed into a database of transaction-like objects [4]. To do so, the SAX-ARM algorithm introduces a basket architecture that will store the deviant events discovered across all time series. A deviant event occurs when the value of a time series is represented with the first or last symbol of the given alphabet. Such events correspond to events in the tail of the normalized distribution of the data and thus may be considered as anomalies or at least somewhat interesting events. A symbol basket is a list-like structure, in which, for a given time stamp, the deviant events – if any – of all time series are retrieved in the form $e = (i, symb)$, where i is the index of the time series and $symb$ the symbol associated with the deviant event. Note that unlike the previous steps, symbol baskets are computed on the multivariate time series and not separately on each univariate time series. Consequently, the symbol basket generation process constitutes the core of the multivariate analysis by grouping together the deviant events encountered in the univariate time series at each time step. Compared to Park and Jung, our implementation discards empty symbol baskets as they have no practical use and are nothing but a waste of memory space. Our implementation of the symbol basket generation process is described in Algorithm 2. The algorithm constructs a list of symbol baskets where each symbol basket stores the deviant events that occurred concomitantly in the different time series.

Algorithm 2: Generation of symbol baskets

```
function SBGen(df, $\mathrm{symb}^{\mathrm{top}}$, $\mathrm{symb}^{\mathrm{bottom}}$)
    Input df: DataFrame of symbolized multivariate time series.
    Input $\mathrm{symb}^{\mathrm{top}}$: Last symbol in the alphabet, to be recognized as a deviant event.
    Input $\mathrm{symb}^{\mathrm{bottom}}$: First symbol in the alphabet, to be recognized as a deviant event.
    Output B: List of symbol baskets.

    Initialize an empty list that will contain the symbol baskets.
    $\mathrm{B} \coloneqq \emptyset$
    Iterate over each 'SAX time stamp' (from 1 to w).
    for $t$ from 0 to $w$ do
        Initialize an empty symbol basket.
        $b_t \coloneqq \emptyset$
        for each time series $x_j \in$ df do
            $symb \coloneqq x_{j,t}$
            If the symbol corresponds to a deviant event, it is added to the
             current symbol basket.
            if $symb \in \{\mathrm{symb}^{\mathrm{top}}, \mathrm{symb}^{\mathrm{bottom}}\}$ then
                $b_t$.add $((j, symb))$
        Store the symbol basket in the list of baskets.
        B.add($b_t$)
    return B
```

For example, let us consider the following symbolic representation of time series, with an alphabet of size $a = 4$:

$$\begin{aligned} \text{Time series 1}: &\quad [\text{‘a’, ‘b’, ‘b’, ‘c’, ‘d’}] \\ \text{Time series 2}: &\quad [\text{‘b’, ‘a’, ‘b’, ‘b’, ‘d’}] \\ \text{Time series 3}: &\quad [\text{‘c’, ‘d’, ‘c’, ‘c’, ‘d’}] \end{aligned} \tag{6}$$

The multivariate time series (6) will result in the symbol baskets shown in Table 1.

Basket id	Symbol Basket
b1	{(Time series 1, 'a')}
b2	{(Time series 2, 'a'), (Time series 3, 'd')}
b3	{(Time series 1, 'd'), (Time series 2, 'd'), (Time series 3, 'd')}

Table 1: Symbol baskets generated from the multivariate time series (6).

The example in Table 1 shows that the temporal information from the deviant event is lost when generating symbol baskets. However, this information may be retrieved when generating the symbol baskets by simply keeping track of the current timestamp. Losing temporal information is not an issue in the context of association rule mining but may be important to the reader interested in anomaly detection.

Note that formatting the deviant events with a symbol (e.g. (i, 'a') or (i, 'd') using the example of Table 1, with i the number of the time series considered) is not convenient for the end-user. In an effort to make the notations more intuitive, we will often refer to the symbol that corresponds to a low deviant event as simply 'low', and to the symbol that corresponds to a high deviant event as 'high' (e.g. (i, low) or (i, high) once again using the example of Table 1).

2.3 Association Rule Mining (ARM)

Association rule mining is a method for discovering frequent patterns, correlations or, more generally, interesting relations between variables in a dataset. A rule is an implication of the form $A \Rightarrow B$ where $A, B \subseteq I$ and $A \cap B = \emptyset$ [7]. I is a finite set of binary attributes called items. By definition, A and B are also sets of items, or itemsets for short.

A typical use-case for association rule mining is market basket analysis, where it can be used to learn more about the buying habits of customers. Given a set of transactions, one may find that the association rule *milk* ⇒ *cereals* occurs frequently. In the SAX-ARM algorithm, the symbol baskets constitute the database of transactional data that will be mined and association rules are generated between one or more deviant events that occur within the symbol baskets.

To quantify the significance of association rules, numerous evaluation metrics have been developed. Three of these measures in particular are used in the SAX-ARM algorithm - the *support*, *confidence* and *lift*. The following paragraphs describe the computation of the three evaluation metrics of interest according to [6] and [7].

The support of an itemset is defined as the proportion of transactions in the dataset which contain the itemset. By extension, the support of a rule corresponds to the proportion of co-occurrence of the two itemsets intervening in the rule. Let A and B be two distinct sets of deviant events such that $A \cap B = \emptyset$, i.e. A and B have no item in common. Let us consider the corresponding association rule $A \Rightarrow B$ between these itemsets. The support of said rule may be expressed as

$$supp(A \Rightarrow B) = supp(A \cup B) \tag{7}$$

where the union of A and B, $A \cup B$, refers to the set that contains all items in A and B. In practice, the support of a rule is computed as the number of symbol baskets containing all deviant events in A and B divided by the total number of baskets.

The confidence of a rule measures the likelihood of occurrence of B in any symbol basket that contains A. It may be computed as

$$conf(A \Rightarrow B) = \frac{supp(A \cup B)}{supp(A)} \tag{8}$$

The lift of a rule compares the likelihood of B under the occurrence of A versus the likelihood of B under no assumption.

$$lift(A \Rightarrow B) = \frac{conf(A \Rightarrow B)}{supp(B)} = \frac{supp(A \cup B)}{supp(A)\, supp(B)} \tag{9}$$

The first algorithm to mine association rules efficiently was the Apriori algorithm, introduced by Agrawal et al. [1]. Apriori tackles two main issues: finding frequent itemsets in a database of transactional data and generating association rules from these itemsets. To select the most relevant rules from the set of all possible rules, Apriori sets a threshold on the minimum support *min_supp* and minimum confidence *min_conf* for a rule to be considered. The minimum support criterion is enforced by only allowing frequent itemsets[2] to constitute rules, i.e. itemsets with a support greater than the minimum support threshold. Indeed, if an itemset X is not frequent, $X \cup A$ where A is any itemset cannot occur more frequently than X and therefore is not frequent either. Put another way, if $supp(X) < min\_supp$, then $supp(X \cup A) = supp(X \Rightarrow A) < min\_supp$ and thus any rule generated from one or more non-frequent itemset cannot meet the minimum support requirement and should not be considered. The minimum confidence threshold constitutes an additional "interestingness" measure to restrict the number of rules.

Algorithm 3 addresses the first sub-problem of association rule mining: finding all frequent itemsets. In this function, frequent itemsets are retrieved iteratively, each iteration adding one item to the previous frequent itemsets and assessing whether the resulting itemsets remain frequent. Let us introduce some helpful notations. An itemset that contains k deviant events is a k-itemset. For instance, {*milk*, *cereals*} is a 2-itemset. The k-itemsets are simply k-subsets of the symbol baskets. In Algorithm 3, the frequent 1-itemsets are found with a first pass on the data. Let L_1 denote the resulting set. In further iterations, the set of frequent k-itemsets L_k is constructed using L_{k-1}, until no more frequent k-itemsets can be found. For $k \geq 2$, the construction of L_k is a two-step process (a numerical implementation is given in Algorithm 4):

1. Join. A set of candidate k-itemsets C_k is built by joining L_{k-1} with itself. Let I_1 and I_2 be two itemsets in L_{k-1} and $I_i[j]$ the j^{th} deviant event in I_i. I_1 and I_2 will be joined if their first $k-2$ deviant events are common, and create the following candidate k-itemset

$$\{I_1[1], I_1[2] \ldots, I_1[k-1], I_2[k-1]\}. \tag{10}$$

2. Prune. The set of candidate k-itemsets C_k is a superset of L_k. To avoid computing the support count of each element of C_k, the antimonotonicity property of the support measure – also called the Apriori property [6] – is used. This property states that all subsets of a frequent itemset must be frequent. Hence, if any $k-1$-subset of a candidate k-itemset is not frequent, then this candidate cannot be frequent and can be removed from C_k. At the end of the pruning process, the remaining candidates in C_k constitute the new set of frequent k-itemsets L_k.

Algorithm 5 handles the second sub-problem of association rule mining: generating association rules for every frequent itemset. In Algorithm 5, rules are created from every possible subset of the frequent itemsets found with Algorithm 3. The rule generation process can be synthetized as a two-step process:

1. For each frequent itemset I, generate all possible subsets.

[2]Also called 'large' itemsets in the original paper [1], but some authors find this term misleading [6]

Algorithm 3: Rules support computation

```
function compute_supports()
    Input B: List of symbol baskets from the SBGen function.
    Input min_supp: Minimal support required to consider a rule.
    Output supports: List of supports for each candidate rule.

    Initialize empty list of supports.
    supports ≔ $\emptyset$
    Initialize a list of 1-itemsets candidates ($L_1$).
    $C_k$ ≔ {$itemset \in$ B, $supp(itemset) >$ min_supp and $len(itemset) = 1$}
    $k ≔ 1$
    while $C_k$ is not $\emptyset$ do
        $L_k ≔ \emptyset$
        for $candidate \in C_k$ do
            Compute the support for the current candidate.
            $\mathrm{n}_{\mathrm{candidatetransaction}} ≔ \cap\, itemset$ for $itemset \in candidate$ if $itemset \in$ B
            support ≔ $\mathrm{n}_{\mathrm{candidatetransaction}} / \mathrm{n}_{\mathrm{totaltransaction}}$
            If this candidate is frequent, add it to $L_k$ and store its support.
            if $support >$ min_supp then
                $L_k$.add({candidate})
                supports.add(support)
        $k ≔ k + 1$
        Generate new candidates from $L_k$.
        $C_k$ ≔ apriori_gen($L_k$, k)
    return supports
```

Algorithm 4: Candidate generation

```
function apriori_gen()
    Input relations: List of itemsets of size k.
    Input k: Number of items in the itemsets (size of the itemsets).
    Output candidates: List of all relevant (k+1)-itemsets from the relations list.

    Join itemsets of k-1 elements to create itemsets of k elements.
    join ≔ combinations(relations, k)
    Select k-itemsets whose k-1-subsets are frequent.
    candidates ≔ [itemset for itemset ∈ join if itemset ∈ relations]

    return candidates
```

2. For every subset s, a rule of the form $s \Rightarrow (I - s)$ is initialized and the support, confidence and lift of that rule are computed. To select only the most significant rules, a minimum confidence threshold is used: only association rules whose confidence is greater than the *min_conf* parameter are kept. Note that because the support of a rule depends on the support s and $I - s$, and because s and $I - s$ are both frequent (by construction), every rule generated also satisfies the minimum support threshold *min_supp*.

Algorithm 6 synthesizes the two main steps of Apriori. When no more frequent itemsets remain, all of the association rules discovered are returned and the Apriori Algorithm 6 exits.

Algorithm 5: Rules generation

```
function compute_rules()
    Input B: List of symbol baskets from the SBGen function.
    Input supports: List of frequent itemsets and their supports.
    Input min_conf: Minimal confidence required to consider a rule.
    Output rules: DataFrame of rules with their associated support, confidence and lift.

    Initialize empty lists.
    Rule $\coloneqq \emptyset$
    Support $\coloneqq \emptyset$
    Confidence $\coloneqq \emptyset$
    Lift $\coloneqq \emptyset$
    for itemset $\in$ supports do
        if len(itemset) $>$ 1 then
            The A side of the rule has all possible lengths.
            for $l$ from 1 to len(itemset) do
                Create rules for each possible subset in itemset.
                for $l$-subset in itemset do
                    Rule in the form A => B.
                    $A \coloneqq l$-subset
                    $B \coloneqq$ itemset - A
                    Compute rule metrics and store them in the lists.
                    support = supports(itemset)
                    confidence = support / compute_support(A)
                    lift = confidence / compute_support(B)
                    Minimum confidence threshold.
                    if confidence $>$ min_conf then
                        Rule.add($\{A \Rightarrow B\}$)
                        Support.add(support)
                        Confidence.add(confidence)
                        Lift.add(lift)

    rules = concatenate(Rule, Support, Confidence, Lift)
    return rules
```

Algorithm 6: Association Rule Mining (Apriori algorithm)

```
1 function Apriori()
      Input B: List of symbol baskets from the SBGen function.
      Input min_supp: Minimal support required to consider a rule.
      Input min_conf: Minimal confidence required to consider a rule.
      Output rules: DataFrame of rules with their associated support, confidence and lift.
2
3     Compute the support of all itemsets with a support above min_supp.
4     supports ≔ compute_supports(B, min_supp)
5     Generate association rules with this selection of itemsets if the
       confidence of the rule is greater than min_conf.
6     rules ≔ compute_rules(B, supports, min_conf)
7     return rules
```

3 Online Demo

Our algorithm is implemented in the Python programming language. Parts of the code for the Apriori algorithm were inspired by this open source implementation[3], which was reviewed, tested and adapted to suit our needs.

The online demo provides three different datasets. All three are slightly customized versions of readily freely available datasets[4][5][6] that were modified to fit our architecture. The first dataset describes the electricity consumption in different French cities based on the regional and national weather conditions. The second dataset gives insights on the traffic conditions depending on the weather. The third dataset contains financial data from the French Autorité des Marchés Financiers (AMF). It consists of processed Order Book data for a few French equities in the month of June 2021. The interested user also has the ability to import its own dataset. The dataset must either be a *.csv* file or *.txt* file and must contain a column named '*t*' with the samples' temporal information.

The SAX-ARM algorithm requires the user to input 4 different parameters:

- w: the number of time segments for the PAA representation. The value of w should be less than the length of the real-valued time series.

- a: the size of the alphabet in the symbolic representation of the time series. Parameter a is set to 10 by default and should vary between 3 and 26.

- *Minimum Support*: the minimum support threshold in the Apriori algorithm (equivalent to the *min_supp* parameter from Algorithm 3). This parameter is set to 1% by default and can evolve in the range 0 to 100%. Realistically, the user should not expect any result from the SAX-ARM algorithm above 50%, as it is the maximum theoretical support expected for a rule (with the least constraining value of a, $a = 2$).

- *Minimum Confidence*: the minimum confidence threshold in the Apriori algorithm (equivalent to the *min_conf* parameter from Algorithm 5). This parameter is set to 50% by default and can vary between 0 to 100%.

[3]https://github.com/ymoch/apyori/blob/master/apyori.py
[4]https://challengedata.ens.fr/challenges/12
[5]https://challengedata.ens.fr/challenges/57
[6]https://challengedata.ens.fr/challenges/66

An additional parameter that may be adjusted is the index of the time series that will be plotted. We chose to only allow a single data column to be plotted because the visualization takes a non-negligible time to compute and only regards the SAX representation, not the actual results of the algorithm (the deviant events and the association rules).

All of the parameters can easily be modified using sliders. The choice of correct parameter values is analyzed and discussed in Section 4.

4 Evaluation

This section aims to analyze the contribution of the different input parameters to the detection of deviant events and their association rules. In this section, we chose to focus on a single dataset — the first dataset presented in Section 3 — that illustrates adequately the typical use-case scenario. Section 4.4 provides some additional information on the behavior of the other two datasets that were tested but not reported here for brevity sake.

The dataset considered contains information on the electricity consumption in different French cities, namely Aix, Angers, Bordeaux, Lille and Paris, on the temperature in Aix, Lille and the national average, and on the humidity in Aix and Lille. For each variable, a sample was collected once every hour starting from the 1^{st} of November 2016 to the 1^{st} of November 2017.

Because the choice of a suitable representation is essential to derive meaningful rules, a particular attention is given to the two SAX parameters w and a (sections 4.1 and 4.2). The effects of the two thresholds min_supp and min_conf on the pruning and conservation of association rules is discussed in Section 4.3.

4.1 Parameter w: Dimensionality Reduction

During the SAX process, a univariate time series of arbitrary length n is reduced to a string of arbitrary length w. To represent the time series in a w-dimensional space, the data is divided into w equal-size segments and only the mean value of the data falling within a segment is kept (see Section 2.1.2). In other words, this parameter defines the 'temporal resolution' of the SAX representation. A good choice of w reduces the dimensionality as much as possible while staying true to the variations in the original time series.

For the rest of this section, the other parameters of the SAX-ARM algorithm were set to $a = 10$, $min\_supp = 0.01$ and $min\_conf = 0.1$. Such a value for parameter a defines deviant events as the top and bottom 10% of the probability distribution – and is chosen by default in [12]. Parameters min_supp and min_conf are minor constraints such that we do not restrict much the set of association rules that we can discover.

4.1.1 Deviant Events

We studied the impact of parameter w on the discovery of deviant events. The SAX-ARM algorithm was run multiple times for different values of w. We define the factor of reduction f such that $f = n/w$ to assess the reduction in dimensionality irrespective of the length of the time series. Table 2 shows the support of deviant events in each univariate time series for different values of f

$$f \in \{1.1, 1.2, 1.5, 2, 3, 5, 10, 20\},$$

supp(low) corresponds to the support of the low-end tail of the data's probability distribution, while supp(high) corresponds to the support of the high-end tail. For instance, a real-valued data point in time series temp_Lille is symbolized with symbol 'j' if its value is greater than or equal to the 10%

highest temperatures recorded in Lille during the year, which we report using the keyword 'high' rather than a cryptic 'j' in an effort to make the notation clearer for the end-user.

		f = 1.1	f = 1.2	f = 1.5	f = 2	f = 3	f = 5	f = 10	f = 20
temp_Lille	supp(low)	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.09
	supp(high)	0.10	0.10	0.10	0.10	0.10	0.10	0.09	0.08
	Sum	0.20	0.20	0.20	0.20	0.20	0.20	0.18	0.18
temp_Aix	supp(low)	0.10	0.10	0.10	0.10	0.10	0.09	0.08	0.07
	supp(high)	0.10	0.10	0.10	0.10	0.10	0.10	0.09	0.07
	Sum	0.20	0.20	0.20	0.20	0.20	0.19	0.17	0.14
temp_France	supp(low)	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10
	supp(high)	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10
	Sum	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
humidity_Lille	supp(low)	0.10	0.10	0.10	0.10	0.09	0.09	0.06	0.03
	supp(high)	0.10	0.10	0.10	0.10	0.09	0.09	0.08	0.06
	Sum	0.20	0.20	0.19	0.19	0.19	0.18	0.14	0.09
humidity_Aix	supp(low)	0.10	0.10	0.10	0.10	0.10	0.09	0.07	0.03
	supp(high)	0.10	0.10	0.10	0.10	0.09	0.09	0.06	0.02
	Sum	0.20	0.20	0.20	0.20	0.19	0.18	0.13	0.05
cons_Bordeaux	supp(low)	0.10	0.10	0.10	0.10	0.10	0.10	0.08	0.07
	supp(high)	0.10	0.10	0.10	0.10	0.09	0.10	0.07	0.02
	Sum	0.20	0.20	0.20	0.20	0.19	0.19	0.15	0.08
cons_Angers	supp(low)	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10
	supp(high)	0.10	0.10	0.10	0.10	0.09	0.10	0.08	0.04
	Sum	0.20	0.20	0.20	0.20	0.20	0.20	0.18	0.13
cons_Paris	supp(low)	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.09
	supp(high)	0.10	0.10	0.10	0.10	0.10	0.09	0.07	0.02
	Sum	0.20	0.20	0.20	0.20	0.20	0.19	0.16	0.11
cons_Lille	supp(low)	0.10	0.10	0.10	0.10	0.10	0.10	0.09	0.09
	supp('high')	0.10	0.10	0.10	0.10	0.09	0.10	0.07	0.02
	Sum	0.20	0.20	0.20	0.20	0.19	0.19	0.16	0.11
cons_Aix	supp(low)	0.10	0.10	0.10	0.10	0.10	0.10	0.09	0.09
	supp(high)	0.10	0.10	0.10	0.10	0.09	0.09	0.08	0.06
	Sum	0.20	0.20	0.20	0.20	0.19	0.19	0.18	0.15
Number of rules		169	167	164	164	163	159	150	109

Table 2: Support for 'low' and 'high' deviant events for different values of f in the electricity consumption dataset. Prefixes *temp* and *cons* stand for temperature and electricity consumption respectively.

Our results show that the support for deviant events decreases when f increases. This observation is coherent. By definition, deviant events are brief, outlier events. Increasing the value of f – hence decreasing the value of w – amounts to increasing the number of points that will be averaged with the deviant event in the PAA representation and, in turn, reduces the contribution of the deviant event to the representation. Of course, not all univariate time series are similarly affected by the dimensionality reduction. Comparing the support of deviant events for the two most extreme values of f, the time series related to the temperature only show an average 13% decrease while the time series describing the humidity show an average 65% decrease. This effect is linked with the time-scale of the variations of the times series. The piece-wise constant approximation of the PAA provides a decent approximation for slow-varying time series, such as the average national temperature in Figure 3 (top). For fast-moving time series, like the humidity index in Figure 3 (bottom), a much higher value of w is required to capture the variations efficiently.

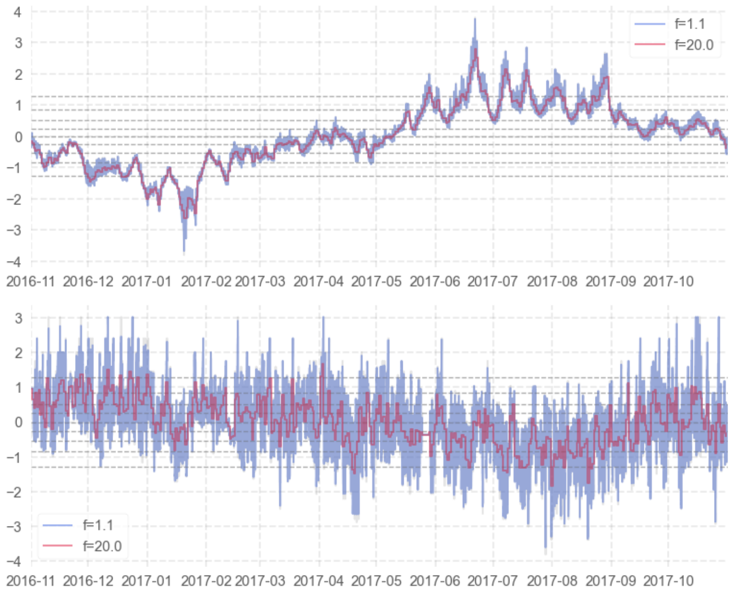

Figure 3: PAA representation for the time series $temp\_France$ (top) and $humidity\_Aix$ (bottom) for $f = 1.1$ (blue) and $f = 20$ (red).

4.1.2 Association Rules

We studied the impact of parameter w on the mining of association rules in deviant events. Like in the previous section, we used the parameter f instead of w. We ran the SAX-ARM algorithm to find association rules using the same previous values of f.

The last row of Table 2 shows the number of association rules found for the different values of f. Unsurprisingly, the number of rules discovered decreases when f increases. This is a corollary of the results in the previous section: the lesser the number of deviant events, the smaller the number of rules that can be mined. This phenomenon is further shown in Table 3, which depicts the evolution of two of the most interesting rules (in terms of support, confidence and lift) for the different values of f. While the results illustrate a relative conservation of the support, confidence and lift values for $f \leq 3$, we observe two opposite behaviors for $f \geq 5$. For the first rule {cons_Angers: high} $\rightarrow$ {cons_Lille: high}, the support *supp('j')* of both deviant events is lower for conditions $f \geq 5$ than $f \leq 3$. Therefore, when f increases, the support and confidence for that rule tend to decrease until eventually disappearing in the case $f = 20$. On the contrary, for rule {cons_Paris: low} $\rightarrow$ {cons_Aix: low}, the support for both deviant events does not drop significantly between higher and lower values of f. In that case, higher values of f tend to exacerbate the association rules that may be found in the dataset.

		f = 1.1	f = 1.2	f = 1.5	f = 2	f = 3	f = 5	f = 10	f = 20
{cons_Angers: high}	Supp	0.11	0.11	0.10	0.11	0.10	0.10	0.08	NaN
$\downarrow$	Conf	0.74	0.73	0.72	0.75	0.73	0.71	0.60	NaN
{cons_Lille: high}	Lift	5.11	5.08	5.05	5.02	5.28	4.92	5.01	NaN
{cons_Paris: low}	Supp	0.10	0.11	0.10	0.10	0.11	0.11	0.12	0.15
$\downarrow$	Conf	0.73	0.73	0.72	0.72	0.73	0.73	0.75	0.82
{cons_Aix: low}	Lift	5.06	5.05	5.05	5.02	5.05	5.03	4.80	4.34

Table 3: Evolution of two of the most predominant rules in the electricity dataset for different values of f.

4.1.3 Summary

The choice of a good value for the parameter w is key to establish a correct PAA representation of the original multivariate time series. The detection of deviant events relies heavily on that first representation step. Choosing a value of w that is too low may prevent capturing the variations of the time series and decrease the number of deviant events discovered. As such, the adverse effect of parameter w on the detection of deviant events depends on the 'volatility' of the time series. This constitutes one limitation of the SAX-ARM algorithm. Indeed, it is impossible to set different values of w for each unidimensional time series as there must be an equal number of time series segments to create the symbol baskets. The discovery of relevant association rules depends on the detection of meaningful deviant events. Thus, rule discovery is also greatly affected by the choice of w. All in all, it is left to the user to decide on a middle-ground between a sufficient dimensionality reduction and an acceptable representation of all univariate time series. Figure 5 (top) synthesizes the evolution of the different rule metrics with f, for different values of a.

Considering the results presented in this section, parameter f was set to 3 in the rest of this article.

4.2 Parameter a: Alphabet Size

The a parameter is a SAX parameter that controls the size of the alphabet in the SAX representation. Parameter a intervenes during the discretization step which assigns a symbol to each PAA value (see Section 2.1.3). In the SAX-ARM algorithm, this parameter is essential in defining the deviant events: deviant events are time segments described by the first or last symbols in the alphabet. The choice of a thus allows to set a threshold on what constitutes a deviant event, e.g. if $a = 10$ then the top and bottom 10% of the data distribution will be considered as deviant event.

In the following sections, the other parameters of the SAX-ARM algorithm were set to $f = 3$, $min\_supp = 0.01$ and $min\_conf = 0.1$.

4.2.1 Deviant Events

We studied the impact of parameter a on the discovery of deviant events. The SAX-ARM algorithm was run multiple times for different values of a

$$a \in \{3, 5, 10, 15, 20, 25\}.$$

Table 4 reports the support of deviant events in each univariate time series. The value supp(low) corresponds to the support of the first symbol (low-end tail of the data's probability distribution) and supp(high) corresponds to the support of the last symbol (high-end tail of the distribution). Once again, this notation is adopted for simplicity sake, especially since the last letter of the symbolic alphabet changes depending on the value of a.

Our results show that the support for deviant events decreases with an increase in a. This was to be expected: increasing a restricts the minimal deviation from normality to be considered a deviant event. Taking the two extremal cases tested, $a = 3$ considers the data values that fall below 33.3% of the probability distribution curve after normalization as deviant events while for $a = 25$ deviant events are data values that fall below 4%. We note that these experimental support values for deviant events correspond to the expected theoretical values. This result indicates that the INT step (see Section 2.1.1) is effective in transforming the original arbitrary data distribution into a Gaussian distribution. This is further illustrated in Figure 4: the Q-Q plots show that the quantiles of the time series after normalization align with the quantiles of the normal distribution.

		a = 3	a = 5	a = 10	a = 15	a = 20	a = 25
temp_Lille	supp(low)	0.34	0.20	0.10	0.07	0.05	0.04
	supp(high)	0.33	0.20	0.10	0.06	0.05	0.04
	Sum	0.67	0.40	0.20	0.13	0.10	0.08
temp_Aix	supp(low)	0.33	0.20	0.10	0.07	0.05	0.04
	supp(high)	0.33	0.20	0.10	0.06	0.05	0.04
	Sum	0.66	0.40	0.20	0.13	0.10	0.08
temp_France	supp(low)	0.33	0.20	0.10	0.07	0.05	0.04
	supp(high)	0.33	0.20	0.10	0.06	0.05	0.04
	Sum	0.67	0.40	0.20	0.13	0.10	0.08
humidity_Lille	supp(low)	0.33	0.20	0.09	0.06	0.05	0.04
	supp(high)	0.33	0.19	0.09	0.06	0.04	0.04
	Sum	0.66	0.38	0.19	0.13	0.09	0.08
humidity_Aix	supp(low)	0.33	0.19	0.10	0.06	0.05	0.04
	supp(high)	0.33	0.20	0.09	0.06	0.05	0.04
	Sum	0.66	0.39	0.19	0.12	0.10	0.07
cons_Bordeaux	supp(low)	0.33	0.19	0.10	0.06	0.05	0.04
	supp(high)	0.34	0.20	0.09	0.06	0.05	0.04
	Sum	0.67	0.40	0.19	0.13	0.09	0.07
cons_Angers	supp(low)	0.34	0.20	0.10	0.07	0.05	0.04
	supp(high)	0.34	0.20	0.09	0.06	0.04	0.04
	Sum	0.67	0.40	0.20	0.13	0.09	0.07
cons_Paris	supp(low)	0.33	0.20	0.10	0.06	0.05	0.04
	supp(high)	0.34	0.19	0.10	0.07	0.05	0.04
	Sum	0.67	0.38	0.20	0.13	0.10	0.08
cons_Lille	supp(low)	0.33	0.20	0.10	0.06	0.05	0.04
	supp(high)	0.33	0.21	0.09	0.06	0.05	0.04
	Sum	0.66	0.41	0.19	0.13	0.10	0.08
cons_Aix	supp(low)	0.33	0.20	0.10	0.07	0.06	0.04
	supp(high)	0.33	0.20	0.09	0.06	0.05	0.04
	Sum	0.66	0.40	0.19	0.13	0.10	0.08
Number of rules		282	226	160	140	109	99

Table 4: Support for 'low' and 'high' deviant events for different values of a in the electricity consumption dataset. Prefixes *temp* and *cons* stand for temperature and electricity consumption respectively.

4.2.2 Association Rules

We studied the impact of parameter a on the mining of association rules in deviant events. We ran the SAX-ARM algorithm to find association rules using the same previous values of a.

The last row of Table 4 shows the number of association rules found for the different values of a. The results further validate the point made in Section 4.1.2: when the number of deviant events decreases, so does the number of rules that can be discovered. Therefore, when a increases, the number of rules decreases. Table 5 focuses on the two specific rules put forward in Table 3. Compared to the default value of $a = 10$, a lower value of a shows greater support and confidence for both rules. This increase in rule support for lower values of a is a direct consequence of the greater support of the individual deviant events. The same reason causes the confidence to increase for lower values of a: there is a greater probability that two deviant events happen simultaneously when the two have a high support rather than a low support. However, lower values of a are also associated with lower lift values.

The lift is a correlation measure that lets us know the degree to which two deviant events are

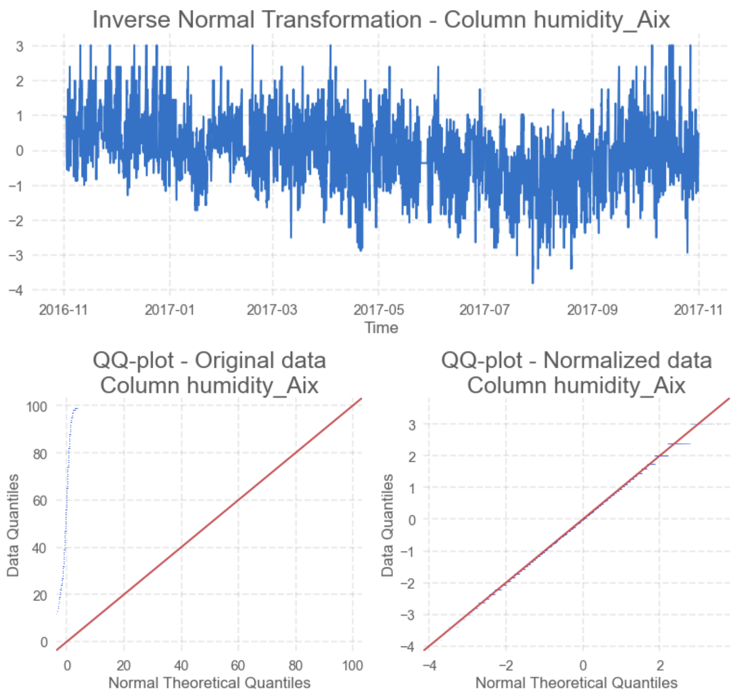

Figure 4: Results of the INT step for the time series labeled *humidity_Aix*. Top: INT ; Bottom: Q-Q plots of the time series, before and after the INT (resp. left and right). After the normalization, the data quantiles are aligned with the Gaussian distribution.

dependent on one another. Here, even though the lift value is greater than 1 for all values of a – meaning that the occurrence of one implies the occurrence of the other – the lift more than doubles between $a = 3$ and $a = 10$. Because the lift metric considers both the confidence of the rule and the overall data set, it is a better judge to determine how interesting a rule is for different values of a. When a increases from 3 to 25, the lift of the most predominant rules first increases, then reaches a maximum – $a = 10$ and $a = 15$ for the first and second rules respectively – and finally decreases. Varying a and tracking the lift of the most predominant rules is an efficient way to choose a value of a that maximizes the support of deviant events while keeping them meaningful.

		a = 3	a = 5	a = 10	a = 15	a = 20	a = 25
{cons_Angers: high}	Supp	0.27	0.18	0.10	0.07	0.06	0.04
↓	Conf	0.81	0.81	0.73	0.59	0.54	0.45
{cons_Lille: high}	Lift	2.42	3.65	5.28	4.89	4.89	4.49
{cons_Paris: low}	Supp	0.25	0.15	0.11	0.09	0.08	0.06
↓	Conf	0.75	0.70	0.73	0.73	0.70	0.55
{cons_Aix: low}	Lift	2.27	3.27	5.05	5.81	5.30	4.55

Table 5: Evolution of two of the most predominant rules in the electricity dataset for different values of a.

4.2.3 Summary

The a parameter constitutes the second important SAX parameter to achieve a correct representation of the original multivariate time series. In the SAX-ARM algorithm, its value sets a threshold to define which events are deviants. We found that, for all univariate time series in our dataset, the support of deviant events for different values of a corresponds to their theoretical expected support (Table 4). This indicates that the INT step of the SAX-ARM algorithm was effective and made every symbol in the alphabet equiprobable. Thus, the parameter a may be set empirically by the user to define what they believe best correspond to deviant events in their dataset, or set experimentally using the technique described in the previous section. If set experimentally, we recommend the user to choose the value of a that leads to the greater lift for the most predominant rules (in terms of support). Otherwise, a default value of $a = 10$ appears to be adequate for most applications. Figure 5 (bottom) synthesizes the evolution of the different rule metrics with a, for different values of f.

Considering the results presented in this section, the parameter a was set to 10 in the rest of this article.

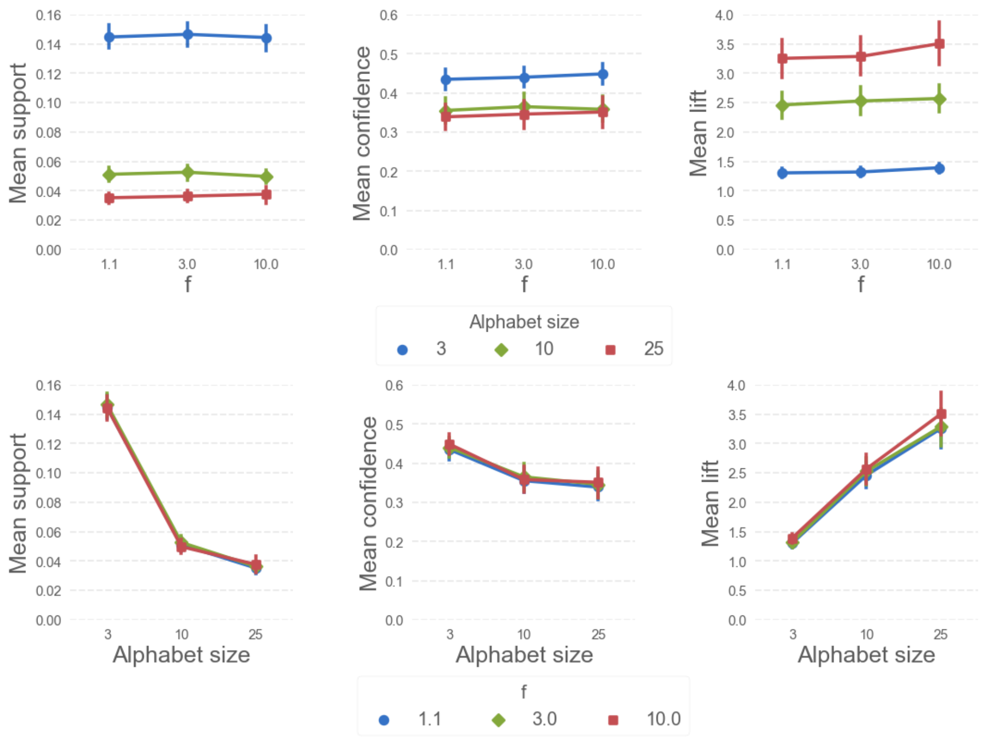

Figure 5: Mean support, confidence and lift metrics with their respective confidence interval for all rules discovered using different parameters of f and a. Using our electricity consumption dataset, we find results similar to Park and Jung [12] and their dataset of a manufacturing process collected from a die-casting machine.

4.3 Parameters min_supp and min_conf: Association Rule Mining

Typically, association rules are considered interesting if they satisfy both a minimum support threshold min_supp and a minimum confidence threshold min_conf. The min_supp parameter allows to define the frequent itemsets described in Section 2.3. The min_conf parameter then further restricts the generation of rules from the mining of frequent itemsets by selecting only those whose deviant events occur concomitantly sufficiently often (see Algorithms 3 and 5). Overall, both parameters aim to prune the number of rules returned by the SAX-ARM algorithm.

4.3.1 Parameter min_supp

We studied the effect of parameter min_supp on the discovery of association rules in deviant events. We ran the SAX-ARM algorithm with $f = 3$, $a = 10$, $min\_conf = 0$ and different values of min_supp

$$min\_supp \in \{0, 0.01, 0.02, 0.05, 0.1\} .$$

Because of our choice of parameter a, the maximal theoretical support is 10%. As expected, we did not find any rule above this threshold and did not report values of min_supp greater than 10%.

Table 6 shows the number of association rules found for the different values of *min_supp*. Our results show a drastic reduction in the number of rules with an increase in *min_supp*. Specifically, a 1% minimum support threshold more than halves the number of rules that are discovered, revealing that many potential association rules between deviant events do not occur often enough to be of interest for knowledge discovery. The choice of a correct value for the support threshold mainly depends on the goal of the analysis. Clearly, rules with a support smaller than 1% may not occur often enough to be considered as interesting rules if the goal is to find common deviant events associations. On the other hand, such low-support rules may be of interest if the goal is to find rare associations. In any case, it is important to recall that the support of a rule by itself is generally not sufficient to characterize a rule as interesting or not. Other metrics such as the lift, or more particularly the confidence — for which a minimum threshold can be set, see Section 4.3.2 — are usually better suited for that task. We advise the reader to use a *min_supp* threshold of 1% if the objective is to keep as many rules as possible, and a higher value if the goal is to focus on a small set of the most occurring rules. Figure 6 (left) synthesizes the evolution of the number of association rules discovered with respect to *min_supp*, for different values of *min_conf*.

min_supp	0	0.01	0.02	0.05	0.1
Number of rules	380	176	142	78	14

Table 6: Number of association rules discovered for different values of *min_supp*.

4.3.2 Parameter *min_conf*

We studied the effect of parameter *min_conf* on the discovery of association rules in deviant events. We ran the SAX-ARM algorithm with $f = 3$, $a = 10$, $min\_supp = 0$ and different values of *min_conf*

$$min\_conf \in \{0, 0.1, 0.25, 0.5, 0.75\} .$$

Table 7 shows the number of association rules found for the different values of *min_conf*. Unsurprisingly, the number of association rules discovered decreases when *min_conf* increases. It is left to the user to decide how restrictive the minimum confidence threshold should be. We advise to set this threshold relatively high compared to the minimum support threshold – around 50% – so that only rules whose deviant events frequently occur simultaneously are returned. Figure 6 (right) synthesizes the evolution of the number of association rules discovered with *min_conf*, for different values of *min_supp*.

min_conf	0	0.1	0.25	0.5	0.75
Number of rules	380	160	88	49	2

Table 7: Number of association rules discovered for different values of *min_conf*.

4.4 Evaluation Summary

The four parameters of the SAX-ARM algorithm must be tuned to fit the dataset under scrutiny. Notably, careful consideration should be given to the choice of parameters w and a in order to achieve a pertinent representation of the data that will lead to meaningful association rules. Because the w parameter is common to all time series in the dataset, its value must be chosen as to correctly represent the time series with the fastest temporal variations. Parameter a should be considered as a way to set a threshold on what does or does not constitute a deviant event across all time series.

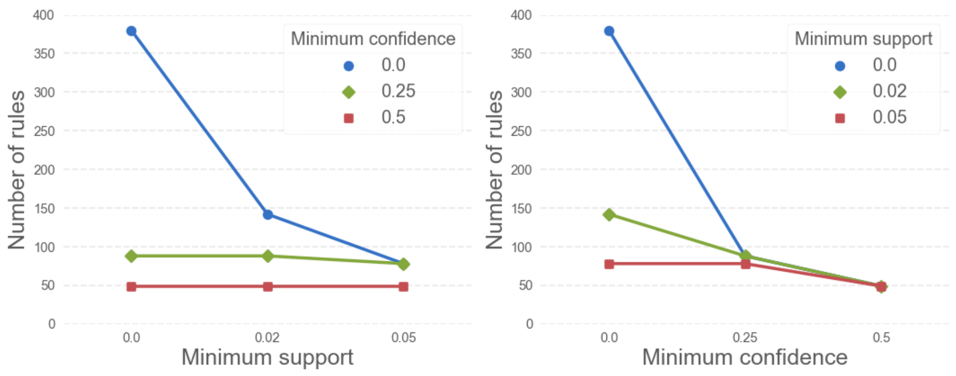

Figure 6: Number of rules discovered using different parameters of min_supp and min_conf.

Once an adequate representation of the dataset is reached, the two remaining parameters min_supp and min_conf are used to prune the number of association rules returned by the algorithm. We advise to keep the min_supp parameter to a relatively low value (e.g. 1%) so as to not miss out on rare yet potentially interesting rules. On the contrary, we encourage to set the min_conf threshold relatively higher than min_supp, to focus only on the more interesting rules.

The general results presented above were illustrated on a single dataset. This dataset was considered to be generic enough so that it corresponds to most use-case scenarios of association rule mining. Indeed, it contains both relatively slow-varying (e.g. the mean national temperature) and faster-varying time series (e.g. the humidity level). Furthermore, there are logical correlations between the time series that lead to intuitive rules that can easily be verified. To be more thorough, we also tested the two other datasets presented in Section 3. While the second dataset (on weather conditions and traffic flow) showed a behavior similar to the one studied above in terms of rule discovery with regards to the different parameters, the third dataset (on financial data) did not. A potential explanation is that there may not be any association rules to be found in the order book data examined.

5 Conclusion

This paper describes a Python implementation of the SAX-ARM algorithm introduced in [12] for the discovery of association rules in deviant events of multivariate time series. The different methods and techniques involved in the algorithm are detailed mathematically and their numeral implementation are described. A thorough analysis of the 4 main parameters of the SAX-ARM algorithm is conducted on a real-world dataset and the choice of adequate parameters is discussed.

References

[1] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules in large databases*, in International Conference on Very Large Data Bases (VLDB), vol. 1215, Citeseer, 1994, pp. 487–499. https://dl.acm.org/doi/10.5555/645920.672836.

[2] T.M. Beasley, S. Erickson, and D.B. Allison, *Rank-based inverse normal transformations are increasingly used, but are they merited?*, Behavior Genetics, 39 (2009), pp. 580–595. https://doi.org/10.1007/s10519-009-9281-0.

[3] Gunnar Blom, *Statistical estimates and transformed beta-variables*, PhD thesis, Almqvist & Wiksell, 1958.

[4] G. Das, K-I. Lin, H. Mannila, G. Renganathan, and P. Smyth, *Rule discovery from time series*, in International Conference on Knowledge Discovery and Data Mining, vol. 98, 1998, pp. 16–22. https://dl.acm.org/doi/10.5555/3000292.3000296.

[5] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, *Fast subsequence matching in time-series databases*, ACM Sigmod Record, 23 (1994), pp. 419–429. https://dl.acm.org/doi/10.1145/191843.191925.

[6] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*, Elsevier, 2011. ISBN 978-9380931913.

[7] K. Hornik, B. Grün, and M. Hahsler, *arules – A computational environment for mining association rules and frequent item sets*, Journal of Statistical Software, 14 (2005), pp. 1–25. https://doi.org/10.18637/jss.v014.i15.

[8] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, *Dimensionality reduction for fast similarity search in large time series databases*, Knowledge and Information Systems, 3 (2001), pp. 263–286. https://doi.org/10.1007/PL00011669.

[9] E. Keogh and S. Kasetty, *On the need for time series data mining benchmarks: a survey and empirical demonstration*, Data Mining and Knowledge Discovery, 7 (2003), pp. 349–371. https://doi.org/10.1023/A:1024988512476.

[10] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, *A symbolic representation of time series, with implications for streaming algorithms*, in ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2003, pp. 2–11. https://doi.org/10.1145/882082.882086.

[11] J. Lin, E. Keogh, L. Wei, and S. Lonardi, *Experiencing SAX: a novel symbolic representation of time series*, Data Mining and Knowledge Discovery, 15 (2007), pp. 107–144. https://doi.org/10.1007/s10618-007-0064-z.

[12] H. Park and J-Y. Jung, *SAX-ARM: Deviant event pattern discovery from multivariate time series using symbolic aggregate approximation and association rule mining*, Expert Systems with Applications, 141 (2020), p. 112950. https://doi.org/10.1016/j.eswa.2019.112950.