TRiER: A Fast and Scalable Method for Mining Temporal Exception Rules

Thábata Amaral and Elaine P. M. de Sousa

[1]Institute of Mathematics and Computer Science (ICMC)
University of São Paulo (USP)
São Carlos, SP - Brazil

thabataamaral@usp.br, parros@icmc.usp.br

Abstract. *Association rules are a common task to discover useful and comprehensive relationships among items. Our interest is to find exception rules, i.e. patterns that rarely occur but have critical consequences. Existing approaches for exception rules usually handle Itemset databases and are unfeasible for mining large ones due to high computational complexity. We thus propose* `TRiER` *(TempoRal Exception Ruler), an efficient method for mining temporal exception rules that not only discover unusual behaviors and their causative agents, but also identifies how long consequences take to appear. We performed an extensive experimental analysis in real data and results show* `TRiER` *is faster and more scalable than existing approaches while finding meaningful rules.*

1. Introduction

Time series occur in countless domains including economy, health and agribusiness. They are continuously produced and stored, generating a large volume of data. This scenario motivates the development of effective and efficient data mining approaches to process and analyze such data. We are particularly interested in association rules mining [Agrawal et al. 1993]. This task has many practical applications due to its simplicity in expressing how humans learn new knowledge. The common purpose is to discover frequent and reliable relationships, called strong rules. An example of strong rule is "with the help of antibiotics, the patient tends to recover" and it is noted as $antibiotics \rightarrow recovery$. Also, We call $antibiotics$ as antecedent of the rule and $recovery$ as consequent.

Although it may be of interest when the expert wants to find unobserved frequent patterns, it is not applicable to detect hidden infrequent ones. Few approaches deal with the extraction of infrequent knowledge. We focus on those proposals that allow obtaining unusual and unexpected information, called exception rules. This class of rule express patterns that contradict the common belief [Suzuki 1996, Hussain et al. 2000]. It means that for searching an exception we have to find an attribute, i.e. an agent, changing the consequent of a strong rule. An example of exception rule is "antibiotics combined with staphylococci may lead to death" and can be noted as $antibiotics\ and\ staphylococcus \rightarrow death$, where $staphylococcus$ is the agent causing the exception.

Mining both rules explain the agents perturbing typical behaviors. As a result, previous control actions can be implemented in scenarios where unexpected agents have critical consequences. An example of prior control action to avoid staphylococcus is to properly maintain the patient hygiene in the hospital. Existing approaches for exception rules usually handle Itemset databases, where transactions have no temporal organization.

However, temporality may be inherent to some real contexts and should be considered to improve the semantic quality of results. Moreover, these approaches have high computational cost (of exponential order), becoming ineffective for mining large datasets.

Aiming to overcome these drawbacks, we propose TRiER (TempoRal Exception Ruler), a fast and scalable method for mining temporal exception rules that not only discover unusual behaviors and their causative agents, but also identifies how long consequences take to appear. TRiER differs from related methods since it handles multivariate time series, proposes a more significant support measure to limit the number of generated rules and discovers rules with greater semantic relevance. We performed an extensive experimental analysis in real data to verify the practical applicability of TRiER. Our results reveal our method is faster and more scalable than existing approaches while finding meaningful temporal exception rules.

The remainder of the paper is organized as follows. Section 2 summarizes background concepts. Section 3 presents related work and existing formalizations for associations rules, sequence mining and exception rules. In Section 4 we describe TRiER. Section 5 presents our experimental study comparing TRiER with related methods. Finally, Section 6 concludes with the contributions of the paper.

2. Background

2.1. Time Series

Time series is a sequence of time-ordered observations with regular time intervals between each pair of observations [Mitsa 2010]. A time series is defined as $S = \{s_1, s_2, \ldots, s_m\}$, where s_i for $i \in \{1, \ldots, m\}$ corresponds to the occurrence of s_i at time t_i. A univariate time series is created by only one underlying variable. A multivariate time series, on the other hand, is created by observations of several variables at each time t_i. An examples is a weather time series with variables of rainfall, maximum temperature and air humidity. We formally represent each observation s_i of a multivariate time series as $s_i = \{s_{i1}, \ldots, s_{iD}\}$, where s_i is a vector with m values that corresponds to D dimensions of the time series.

Time series discretization is the process of mapping continuous values into discrete ones, aiming to provide a suitable and concise representation of the time series for data mining tasks. The simplest discretization methods apply the equal-width and equal-frequency approaches. Equal-width methods divide the values so that all ranges have the same size. In contrast, equal-frequency ones divide the values so that ranges have the same frequency of values. Intuitively, to mine infrequent patterns, the equal-width approach is more appropriate. The reason is equal-frequency discretization creates irregular size ranges, grouping rare items to produce ranges with the same frequency of values. This grouping assumes all values in the range have the same probability of occurring, thus diluting unusual patterns. Our preliminary empirical studies corroborated this intuition.

2.2. Association Rules

Given a set I (set of items) and a database DB composed of a set of transactions T, each one being a subset of I, association rules are implications in the form $X \rightarrow Y$ that relate the presence of itemsets X and Y in transactions of DB, assuming that $X, Y \in I$, $X \cap Y = \varnothing$ and $X, Y \neq \varnothing$. We call X as **antecedent** and Y as **consequent** of the rule. The classical measures to assess association rules are support ($supp$) and confidence

($conf$). Support calculates the number of times the rule occurred ($freq$) considering the total number of transactions T in the database. Equation 1 defines the support of $X \rightarrow Y$.

$$supp(X \rightarrow Y) = \frac{freq(X \cup Y)}{T} \tag{1}$$

Confidence measures the probability of the consequent occurs in transactions that also contain the antecedent. Equation 2 represents the confidence of $X \rightarrow Y$.

$$conf(X \rightarrow Y) = \frac{freq(X \cup Y)}{freq(X)} \tag{2}$$

Given the minimum thresholds of support ($minsupp$) and confidence ($minconf$) informed by the user, we say $X \rightarrow Y$ is frequent if $supp(X \rightarrow Y) \geq minsupp$ and confident if $conf(X \rightarrow Y) \geq minconf$. Moreover, $X \rightarrow Y$ is **strong** if it is frequent and confident. An alternative framework to support-confidence was proposed in [Berzal et al. 2002] where the accuracy is measured by means of certainty factor (cf).

Certainty factor solves some of the confidence drawbacks. In particular, the support-certainty factor framework reduces the number of obtained rules, filtering those corresponding to statistical independence or negative dependence. As a consequence, extracted rules are stronger than those obtained with support-confidence. Certainty factor ranges from -1 to 1 and measures how our belief that Y is in a transaction changes when we are told that X also is. Positive values indicate our belief increases, negative values mean it decreases and 0 means no change. Analogously, we say $X \rightarrow Y$ is certain if $cf(X \rightarrow Y) \geq mincf$, where $mincf$ is the minimum threshold for certainty factor informed by the user. Equation 3 defines the certainty factor of $X \rightarrow Y$.

$$cf(X \rightarrow Y) \begin{cases} \dfrac{conf(X \rightarrow Y) - supp(Y)}{1 - supp(Y)} & \text{if } conf(X \rightarrow Y) > supp(Y) \\ \dfrac{conf(X \rightarrow Y) - supp(Y)}{supp(Y)} & \text{if } conf(X \rightarrow Y) < supp(Y) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

3. Related Work

Traditional methods for association rules relate items disregarding their occurrence orders [Agrawal et al. 1993]. However, time information is relevant to some applications and should be analyzed to better understand semantic issues. Thus, sequence mining was introduced in Agrawal and Srikant 1995 to find sequences of items considering the order they occur. There are several approaches for sequence mining in literature. We thus discuss some of the most explored ones and those closely related to our work.

Classic sequence mining algorithms were based on *Apriori*, like *GSP* (Generalized Sequential Patterns) [Srikant and Agrawal 1996]. Apriori-like algorithms usually include candidate generation and validation steps, performed with support and confidence counts. A limitation of Apriori-like approaches is the need for support calculations through a complete database scan, comparing each item with all transactions. Aiming to reduce

the processing time, approaches based on vertical format, as *ECLAT* [Zaki 2000] and *SPADE* [Zaki 2001], and pattern growth, as *Prefix-Span* [Pei et al. 2001] were proposed.

Prefix-Span (Prefix-projected Sequential Pattern Growth) [Pei et al. 2001], for example, does not require a candidate generation step. The database is recursively projected into smaller parts and frequent sequences are directly extended to create larger ones. Sequence mining algorithms, in general, look for sequences considering the order items occurred but do not establish cause and consequence relationships. In addition, they are pseudo-polynomial in time complexity [Dong 2009] and implement a non-discriminatory support measure. Therefore, when applied to mine sequences with low support thresholds, these algorithms generate an overwhelming amount of sequences in unfeasible time.

Low support thresholds concern the mining of infrequent patterns, that might be relevant to some applications. We focus on those proposals that allow obtaining some unexpected and uncommon information, called exception rules. In general, these approaches are able to manage rules that, being infrequent, provide a specific domain usually delimited by a strong rule. Exception rules were first defined as rules that contradict the user's common belief. It means that for searching an exception we have to find an attribute (also called agent) that changes the consequent of a strong rule. In general terms, the kind of knowledge an exception rule discovers can be interpreted as follows.

*"X **strongly** implies Y (and not E), but in conjunction with E, X **does not** imply Y"*.

Classical works on this research topic are presented in Suzuki 1996 and Hussain et al. 2000. According to Suzuki 1996, an exception rule is formally defined as:

$$X \rightarrow Y \text{ (high } supp \text{ and high } conf - \text{strong rule).}$$
$$X \wedge E \rightarrow \neg Y \text{ (low } supp \text{ and high } conf - \text{exception rule).}$$
$$X \nrightarrow E \text{ (high } supp \text{ and high } conf - \text{reference rule).}$$

$X \rightarrow Y$ is a strong rule and indicates a common behavior. $X \wedge E \rightarrow \neg Y$ is an exception rule and shows that the presence of item E has modified the expected consequent of the strong rule. $X \nrightarrow E$ is a reference rule and determines the antecedent of the rule should have no association with the agent causing the exception. A similar definition is presented in Hussain et al. 2000. The difference lies in the reference rule, where the agent causing the exception may have association with the unexpected behavior ($E \rightarrow \neg Y$).

Some different definitions for exception rules do not follow the schema of mining exceptions using the three previous properties, as presented in Daly and Taniar 2008. These approaches focus on finding unusual and contradictory behavior while Suzuki 1996 and Hussain et al. 2000 also allow inferring which agent caused the exceptional behavior.

The Exception Rule Search Algorithm (*ERSA*) [Calvo-Flores et al. 2011] is based on definition introduced in Suzuki 1996 but it does not employ a reference rule. Authors argue this rule does not offer a semantic enrichment when defining exceptions and reformulate the definition as the pair of common sense and exception rules as shown below:

$$X \rightarrow Y \text{ (frequent and certain} - \text{strong rule).}$$
$$E \rightarrow \neg Y \text{ (certain in the domain of the strong rule antecedent} - \text{exception rule).}$$

ERSA basic operation is given as follows. The database is first converted into a binary format to search the set of frequent items and select strong rules. For each strong rule,

the algorithm generates exceptions possibilities, applying confidence or certainty factor to filter relevant rules. Its complexity is $O(Trl2^l)$, where T is the number of transactions in the database, l is the quantity of obtained items and r is the number of rules. Most recent methods are based on fuzzy logic, as *FERSA* (Fuzzy Exceptional Rule Search Algorithm) [Ruiz et al. 2016]. This method is similar to *ERSA* in terms of time complexity and definition, applying fuzzy logic to avoid the problem of data imprecision.

To the best of our knowledge, there is no method in literature to find exception rules caused by more than one agent. Recall staphylococcus example: although a combination of agents may cause the patient death, related methods only recognize staphylococcus as causative agent. Moreover, those methods do not consider temporality in the rules discovering process. We thus propose `TRiER`, a fast and scalable method for mining temporal exception rules that may include more than one agent causing the exception. We also propose a more selective support measure which reduces the number of sequences.

4. Proposed Method

We propose `TRiER` (TempoRal Exception Ruler), a fast and scalable method for mining temporal exceptions and their corresponding strong rules. Mining both rules allows a better understanding of the agents that perturb the strong rule's usual behavior. In general terms, the kind of knowledge `TRiER` discovers is described as follows.

"X strongly implies Y (and not E) after a while. But X and E (even if E occurs later) implies something different from Y after another while".

`TRiER` effectively deals with univariate and multivariate time series and implements the concepts of sliding window and time lag. A **window** (w) of a time series S is a block of events that occurs in a continuous interval, starting at time t_b and ending at time t_e, such that events t_b and t_e belong to S. A **sliding window** ($W[i]$) at position i of time series S is a window starting at time i and ending at time $i + w_s$, where w_s is the number of consecutive time instants composing $W[i]$ (sliding window size).

Time lag (t_{lag}) is a delay between the beginning of the antecedent and the end of the consequent. Time lag follows the temporal granularity of time series, such as days, months or years. We divide our method in three main steps: (1) *Sequences Generation*, (2) *Rules Discovery* and (3) *Exception Rules Mining*, as illustrated in Figure 1.

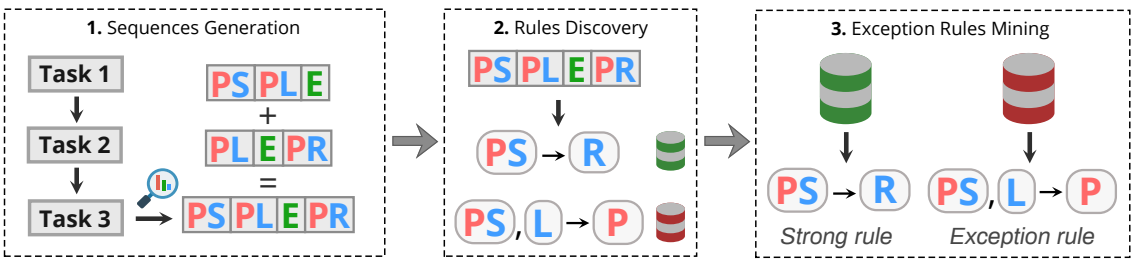

Figure 1. `TRiER` overview.

In *Sequences Generation* we create sequences that will form strong and infrequent rules. For this, we apply two parameters in the discretized dataset: the sliding window size ($W[i]$) and the minimum support for a sequence to be considered frequent ($minsupp$).

This step is composed of three main tasks. In the first one, we calculate the support of each discretization symbol, selecting those that meet the $minsupp$ threshold. In second task, we generate sequences with one observation and D dimensions. To generate a frequent sequence with D dimensions, we join two frequent sequences with $D-1$ dimensions. In last task, we obtain sequences with multiple observations and dimensions. We thus join two frequent sequences with $k-1$ observations to form a candidate with k observations, considering the order of the observations. This task ends when we reach the sliding window size. In Figure 1, to form the sequence PS, PL, E, PR we join two sequences with three observations, as PS, PL, E and PL, E, PR, so that the last observations of the first sequence are exactly the same as the first observations of the second sequence.

Classical support measures only consider the number of series in which a sequence occurs. In other words, there is no difference if a sequence occurs innumerable times or only once in a series, since they have the same support. Therefore, we propose a more selective support measure to better estimate the contribution of a sequence. Our measure considers the number of times a sequence occurs and the number of different series containing that sequence. Equation 4 formalizes our measure: L is the number of time series in the database, m is the number of observations in a series, n is the number of time series in which a sequence occurs and f is the number of times a sequence appears.

$$supp = \frac{f}{L \cdot m} \cdot e^{\left(\frac{n}{L}-1\right)} \tag{4}$$

`TRiER` was implemented in Java and our strategy to improve the efficiency of sequences generation step is described as follows. Initially, time series are stored in a map structure, called **Hash Map**. This structure is composed of key and value sets that permit to return objects quickly, with constant complexity. Keys are formed by items and an item indicates an observation in a time series. The value of each key is a **Tree Set** composed of pairs in the format $(i{:}j)$, where j is a time series identifier containing the key in the instant i. Tree Set implements a red black tree to sort $(i{:}j)$ pairs and performs the most common operations in logarithmic complexity.

In *Rules Discovery* sequences are analyzed according to the time lag possibilities they can assume. In Figure 1 we can extract rules with time lag 1, 2 and 3 from PS, PL, E, PR. Sequence $PS, L \rightarrow PR$, for instance, has time lag 2 because PR occurs two units of time after L. Rules possibilities are evaluated with the following parameters: (I) minimum support of the strong rule $(minsupp_{str})$, (II) minimum support of the infrequent rule $(minsupp_{inf})$, (III) maximum support of the infrequent rule $(maxsupp)$, (IV) minimum confidence $(minconf)$ and minimum certainty factor $(mincf)$. We classify a rule as **strong** if its support-confidence or support-certainty factor meets these minimum thresholds. To classify **infrequent rules** we apply a $maxsupp$ threshold to assure the maximum support of infrequent rules does not exceed the $minsupp$ of strong rules.

We use certainty factor because it is more restrictive than confidence and reduces the number of rules that may be huge in some cases. We do not exclude the confidence because it composes certainty factor, besides basing comparison with other methods. Once the rules are classified in strong and infrequent we start *Exception Rules Mining*, considering a slightly different definition of exception rules, as follows.

$$X \rightarrow Y \text{ (frequent and confident/certain} - \text{strong rule)}$$

$X \wedge E \rightarrow \neg Y$ (infrequent and confident/certain $-$ exception rule)

The main difference of our definition to literature ones is that we consider an exception rule as infrequent. We argue those rules are naturally atypical and indicates an unusual behavior, so the maximum support of exception rules should not exceed the minimum support of a strong rule. In *Exception Rules Mining* we analyze strong rules and look for infrequent rules that meet the following restrictions: (I) the antecedent of the infrequent rule must contain the antecedent of the strong rule and (II) the consequent of the infrequent rule must not contain or be included in the consequent of the strong rule. The infrequent rule satisfying those constraints will constitute an exception to the analyzed strong rule. As the infrequent rule $PS, L \rightarrow P$, illustrated in Figure 1, meets the previous constraints, it constitutes an exception to the strong rule $PS \rightarrow R$.

We remark the agent causing the exception can occur at a different time from the antecedent, as long as it occurs before the rule consequent. We also enable exceptions to be caused by more than one item. Algorithm 1 summarizes the main idea of our method.

Algorithm 1: TRiER exception rules mining

Input : Frequent sequences, $minsupp_{str}$, $minsupp_{inf}$, $maxsupp$, $minconf$ or $mincf$

Output: Temporal exceptions and their corresponding strong rules

```
1  begin
2    foreach sequence seq ∈ freqSequences do
3      for (p = 0 to p ≤ seqsize − 2) do
4      generate rules from the current sequence with time lag t_lag
5      filter generated rules using mincf or minconf
6      if (rule support ≥ minsupp_str) then
7        strong rules ← rule
8      else if (rule support ≥ minsupp_inf ∧ rule support ≤ maxsupp) then
9        infrequent rules ← rule
10   foreach strong rule str ∈ strong rules do
11     foreach infrequent rule inf ∈ infrequent rules do
12       if (str antecedent ⊂ inf antecedent ∧ str consequent ⊄ inf consequent) then
13         exception rules ← strong rule and its exception rule
```

TRiER complexity is divided in sequences generation and exception rules. Sequences generation is the most costly step and is $O((w_s L m D^{w_s})^2 log(Lm))$, where w_s is the sliding window size, L is the number of time series in the database, m is the number of observations in a time series and D is the number of dimensions in each observation. The exception rules complexity is $O(yw_s)^2$, where y is the number of generated sequences.

Although sequences generation has a high complexity, when mining larger sequences with low support our results are obtained at feasible times. TRiER overcomes drawbacks of related methods as it handles multivariate time series, proposes a more significant support measure, identifies exceptions caused by more than one item, applies a sliding window in the mining process and detects rules occurring in a time lag.

5. Experimental Analysis

Our experimental results on real data show the usefulness, effectiveness and efficiency of TRiER when compared to related methods. We first analyze the relevance of sequences

generated by TRiER and classical algorithms, as *GSP* and *Prefix-Span*. Finally, we compare TRiER with the baseline for exception rules, *ERSA*. Experiments were performed on an Intel Core i7, 3.40 GHz computer with 16 GB of RAM and a SATA hard disk.

5.1. Agriculture Data

Our experimental study on real data aims to investigate how computational methods can consolidate automatic means of discovering frequent patterns and exceptions in agriculture under climate influence. We chose agriculture because it plays a fundamental role in the economy of several countries. In Brazil, for example, agribusiness accounts for 23% of GDP and 40% of the labor. Mining exception rules in agriculture may help us to identify which climatic conditions most affect crops and how quickly implications arise. As a result, previous control actions could be carried out in regions with similar characteristics to minimize severe impacts caused by extreme weather, as droughts. Our analysis focus on sugarcane, due to its importance for ethanol production and Brazil's economy.

We constructed a multivariate time series dataset including observations of climate variable and vegetation index. Vegetation index data were obtained with SATVeg[1], a tool that extracts *NDVI* time series from TERRA/MODIS satellite images. *NDVI* (Normalized Difference Vegetation Index) is a widely used index in agricultural research that represents the soil vegetative vigor. *NDVI* ranges from -1 to 1: values close to 1 indicate strong vegetative activity while negative or close to 0 values describe regions where there is weak or no chlorophyll activity. We collected *NDVI* time series of sugarcane from 2014 to 2018. Time series are composed by 60 monthly observations from the state of São Paulo, the largest sugarcane producing state in Brazil.

We also collected climate time series from National Institute of Meteorology[2] (INMET). Time series are multivariate, composed of monthly observations of rainfall, maximum and minimum temperature. We then associated climate observations with *NDVI* data within a range of 70 km (geodesic distance) from the corresponding weather station. Figure 2 illustrates sugarcane regions we analyze (green points) and the weather stations monitoring them (purple stars).

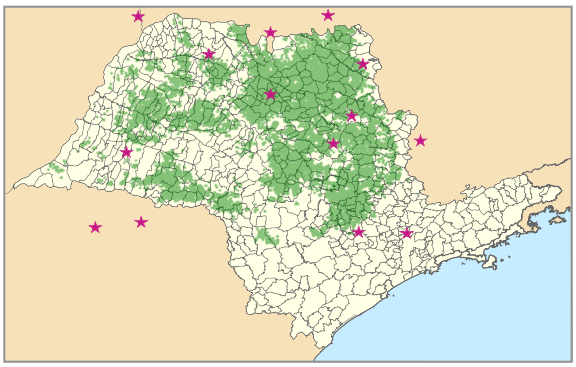

Figure 2. Weather stations and sugarcane areas in São Paulo

Table 1 describes the resulting dataset. $MaxTemp$ and $MinTemp$ are abbreviations of maximum and minimum temperature, respectively.

[1]https://www.satveg.cnptia.embrapa.br/satveg/

[2]http://www.inmet.gov.br/portal/

Table 1. Dataset description

Time series size (m)	Dimensionality (D)	Dataset size (L)
60 monthly observations	$MaxTemp$, $MinTemp$, *NDVI* and rainfall	$24,420$ time series

5.2. Sequences Generation

This experiment aims to investigate the efficiency of *GSP*, *Prefix-Span* and TRiER in the most costly phase of the mining process, i.e. sequences generation. Accordingly, we measure running time as we increase the window size (w_s) and the $minsupp$ threshold. As the temporal granularity of series is monthly, the window size is given in months. $w_s = 1$, for instance, means one month, $w_s = 2$ indicates two months and so on. Each algorithm was executed 5 times and Table 2 summarizes their average time, in minutes. We also tested *GSP* with $w_s = 2$ and the algorithm took on average 346 minutes in sequence mining. Thus, testing other window sizes would be impractical.

Table 2. Time for generating sequences in minutes

$minsupp$	$w_s = 2$		$w_s = 3$		$w_s = 4$		$w_s = 5$	
	Prefix-Span	TRiER	*Prefix-Span*	TRiER	*Prefix-Span*	TRiER	*Prefix-Span*	TRiER
0.05	**0.067**	7.416	**1.494**	19.166	34.324	**26.5**	811.613	**29.75**
0.1	**0.066**	3.5	**1.428**	6.916	34.319	**8.583**	806.415	**10.2**
0.15	**0.066**	2.25	**1.427**	3.833	33.906	**4.333**	784.452	**6.3**
0.2	**0.065**	1.116	**1.421**	1.916	33.346	**1.935**	735.166	**3.383**
0.3	**0.065**	0.75	1.352	**0.8**	32.744	**0.816**	648.352	**1.25**
0.4	**0.064**	0.383	1.319	**0.416**	32.380	**0.466**	565.446	**0.65**

Our analysis showed that in the mining of infrequent patterns, the equal-width approach is more appropriate. The reason is that equal-frequency discretization creates irregular size ranges, grouping rare items to produce ranges with the same frequency. This grouping assumes all values in the range have equal probability of occurring. As a result, equal-frequency discretization dilutes unusual patterns.

As we increase the window size, the performance of existing works degrades. The reason is that these algorithms implement a non-discriminatory support measure. It means that if an item occurs only once or countless times, it will have the same contribution in support calculation. As a consequence, these algorithms generate an overwhelming number of sequences, which include a large amount of irrelevant and unnecessary information

Figure 3 illustrates the significant growth in the number of sequences as the window size increases. For small windows (2 and 3) the number of sequences is feasible, 1397 and 50515 respectively. When we work with larger windows, 4 and 5, the number of sequences is immense, 1693404 and 52520544 respectively. Since the rules will come from this volume, the number of rules will also be huge. Thus, instead of creating knowledge to facilitate expert analysis, this amount of information will require a second-order data mining task to weed out irrelevant patterns and filters necessary ones.

Although our method is not superior to *Prefix-Span* for smaller windows (2 and 3), sequences are obtained at feasible times, unlike *GSP*. However, when we increase the minimum support thresholds, TRiER has the best performance in most window sizes. For larger windows the gain of our approach is meaningful. The reason is that we generate

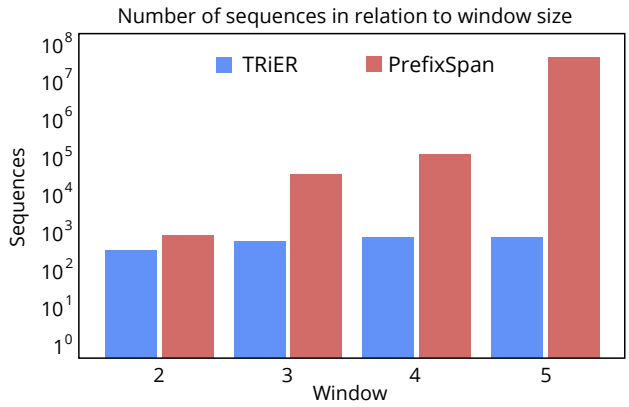

Figure 3. Sequences generated by `TRiER` and *Prefix-Span* for 5% of minsupp

far fewer sequences than previous approaches since we apply a more selective support measure. As a result, we reduce the time required to create larger sequences.

5.3. Rules Discovery

This experiment aims to investigate the effectiveness and usefulness of `TRiER` and the baseline method for mining exceptions, *ERSA*. We analyze results from two perspectives: the time taken to discover rules and their semantic relevance. Figure 4 illustrates time spent by `TRiER` and *ERSA* to discover exception rules. `TRiER` was tested with different sliding windows $W[i] = \{2, 3, 4, 5\}$ to analyze how this size influences on rules discovery step. Although *ERSA* generates all the possibilities of frequent itemsets, we restricted the maximum itemset size to 5 so that the time taken was not impractical.

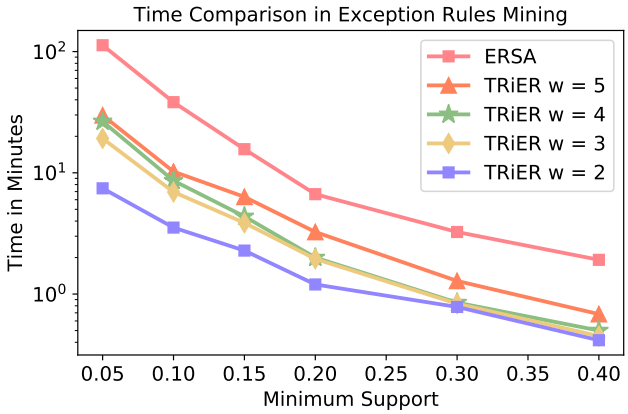

Figure 4. Time spent by *ERSA* and `TRiER` to discover exception rules

As *ERSA* generates a huge number of rules, we measured times with 60% of certainty factor for both algorithms. Rules generated only by confidence or lower values of certainty factor exceeded the computational resources available to perform the experiment. Unlike `TRiER` that deals with multivariate data and understands a time series is composed of 60 observations and each observation has 4 variables (dimensions), *ERSA* does not make this distinction and considers a time series as one itemset with 240 items, that is, 60 observations of each variable (dimension).

To discover exception rules, *ERSA* first generates frequent itemsets. The way it was planned *ERSA* would generate 2^{31} candidates. For that reason we simplified the maximum itemset size to 5. Then, the number of possible candidates is the number of sets with up to 5 elements we can form with the number of symbols used in discretization. This simplification reduces the number of candidates to approximately $200,000$ itemsets.

Another problem that degrades *ERSA* performance is the way it searches exceptions. The rules are stored in a single set, without distinction of strong and infrequent rules. Thus, for each strong rule, the set is scanned again to identify an exception rule. This step is one of *ERSA's* efficiency and effectiveness bottlenecks. Efficiency bottleneck because the set tends to be large and is wholly scanned every time a strong rule is found. Effectiveness bottleneck because many unnecessary and irrelevant rules can be interpreted as an exception, which can hinder and confuse the expert's analysis.

TRiER, in contrast, classifies discovered rules between strong and infrequent. Thus, for each strong rule the set of infrequent rules is analyzed to discover an exception that meets the premises we defined in Section 4. In other words, our method does not scan the whole rules set, but only the ones classified as infrequent, a much smaller set. In order to base the semantic analysis of the discovered rules, we present some examples of rules generated by TRiER and *ERSA*. Table 3 presents examples of rules mined by TRiER. Values are presented in intervals, due to the discretization process. Temperatures are measured in degree Celsius, rainfall in millimeters and time lag in months. Support, confidence and certainty factor relate to exception rules. Recall exception rules definition: X is the rule antecedent, Z is the agent causing the exception, $\neg Y$ is the exceptional consequent and Y is the expected consequent.

Table 3. Rules discovered by TRiER

X	Z	$\neg Y$	Y	t_{lag}	$supp$	$conf$	cf
$MinTemp[18, 20]$	$rainfall[0, 20]$	*NDVI*$[0.2,0.4]$	*NDVI*$[0.6, 0.8]$	1	5.09%	74.38%	69.17%
$MaxTemp[28, 30]$	$rainfall[450, 500]$	*NDVI*$[0.6, 0.8]$	*NDVI*$[0.4, 0.6]$	1	9.82%	84.53%	70.96%
$MinTemp[20, 22]$	$MaxTemp[32, 34]$ and $rainfall[100, 150]$	*NDVI*$[0.8, 1.0]$	*NDVI*$[0.6, 0.8]$	1	14.25%	85.47%	79.63%

These rules can be validated by studies conducted by Embrapa[3]. They state periods of low rainfall damage sugarcane crops and are responsible for reducing NDVI values. In contrast, intense rainfall are related to the increase in *NDVI*. Finally, high temperatures and regular rainfall are ideal for sugarcane plantations.

Rules generated by *ERSA* are semantically restricted because they only allow inferring about relationships and do not consider the temporal aspect. Table 4 presents examples of rules mined by *ERSA*.

Table 4. Rules discovered by *ERSA*

X	Z	$\neg Y$	Y	$supp$	$conf$	cf
$MinTemp[18, 20]$	$rainfall[200, 250]$	*NDVI*$[0.6,0.8]$	*NDVI*$[0.4, 0.6]$	54.13%	65.78%	59.58%
$MaxTemp[30, 32]$	$MinTemp[8, 10]$	*NDVI*$[0.4, 0.6]$	*NDVI*$[0.6, 0.8]$	39.25%	78.30%	62.25%

6. Conclusion

In this paper we proposed TRiER, a method for mining temporal exception rules. Our method handles multivariate time series and applies the concept of sliding window as a

[3]https://www.embrapa.br/

constraint in the sequence mining process. Sliding window enables to discover rules and exceptions that do not necessarily occur instantly. We also developed a more restrictive support measure, which considers the importance of an item in a sequence or rule. Besides support-confidence framework `TRiER` implements the support-certainty factor one. We apply certainty factor as an alternative to confidence to filter rules corresponding to statistical independence or negative dependence.

7. Acknowledgements

We thank National Council for Scientific and Technological Development (CNPq), National Council for the Improvement of Higher Education (CAPES) and São Paulo Research Foundation (FAPESP) for financial support.

References

Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of SIGMOD*, pages 207–216.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of ICDE*, pages 3–14.

Berzal, F., Blanco, I., Sánchez, D., and Vila, M. (2002). Measuring the accuracy and interest of association rules: a new framework. *Intelligent Data Analysis*, pages 221–235.

Calvo-Flores, M., Ruiz, M., and Sánchez, D. (2011). New approaches for discovering exception and anomalous rules. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 19:361–399.

Daly, O. and Taniar, D. (2008). Exception rules in data mining. *Applied Mathematics and Computation*, pages 735–750.

Dong, G. (2009). *Sequence data mining*. Springer-Verlag, Berlin, Germany.

Hussain, F., Liu, H., Suzuki, E., and Lu, H. (2000). Exception rule mining with a relative interestingness measure. In *Proceedings of KDD*, pages 86–97.

Mitsa, T. (2010). *Temporal data mining*. Chapman & Hall/CRC, Minneapolis, U.S.A.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2001). Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of ICDE*, pages 215–224.

Ruiz, M. D., Sánchez, D., Delgado, M., and Martin-Bautista, M. J. (2016). Discovering fuzzy exception and anomalous rules. *IEEE Transactions on Fuzzy Systems*, 24(4):930–944.

Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In Apers, P., Bouzeghoub, M., and Gardarin, G., editors, *Advances in Database Technology — EDBT '96*, pages 1–17, Berlin, Heidelberg. Springer Berlin Heidelberg.

Suzuki, E. (1996). Discovering unexpected exceptions : a stochastic approach. *Proceedings of RSFD 1996*, pages 259–262.

Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, pages 372–390.

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