Mining Partially-Ordered Episode Rules with the Head Support

Yangming Chen[1], Philippe Fournier-Viger[1][0000−0002−7680−9899], Farid Nouioua[2], and Youxi Wu[3]

[1] Harbin Institute of Technology (Shenzhen), Shenzhen, China
philfv8@yahoo.com, 510717841@qq.com
[2] University of Bordj Bou Arreridj, Algeria
faridnouioua@gmail.com
[3] Hebei University of Technology, Tianjin, China
wuc567@163.com

Abstract. Episode rule mining is a popular data analysis task that aims at finding rules describing strong relationships between events (or symbols) in a sequence. Finding episode rules can help understanding the data or making predictions. However, traditional episode rule mining algorithms find rules that require a very strict ordering between events. To loosen this ordering constraints and find more general and flexible rules, this paper presents an algorithm named POERMH (Partially-Ordered Episode Rule Mining with Head Support). Unlike previous algorithms, the head support frequency measure is used to select interesting episode rules. Experiments on real data show that POERMH can find interesting rules that also provides a good accuracy for sequence prediction.

Keywords: Episode rules · Partially Ordered Rules · Head Support.

1 Introduction

Discrete sequences of events or symbols is a data type found in many domains. For instance, a sequence may represents words in a text, nucleotides in a DNA sequence, and locations visited by a tourist in a city. To analyze a sequence of symbols, a popular data science task is episode mining [1, 5–7]. The aim is to find subsequences that appear many times, called *episodes*, and rules called *episode rules*. These rules indicate strong sequential relationships between events or symbols in a sequence, and can help humans to understand the data take decisions, and also be used to predict the next symbol (event) of a sequence [1, 3, 6, 7].

Though several episode rule mining algorithms were proposed, a problem is that they enforce a very strict ordering of events [3]. For instance, a rule $\langle e, f\rangle \rightarrow \langle a\rangle$ indicates that if symbol e is followed by f, then it will be followed by a. Though this rule is easily understandable, the ordering between events is very strict and thus a similar rule with a different ordering such as $\langle f, e\rangle \rightarrow \langle a\rangle$ is viewed as different. As a solution to this issue, an algorithm named POERM

was proposed to discover a more general type of rules, called *partially-ordered episode rules* (POER) [3]. A POER has the form $X \rightarrow Y$, indicating that if some symbols X appear in any order, they will be followed by some symbols Y.

Even though POERM was shown to find interesting rules in real data, a drawback is that each rule is evaluated by only counting its non-overlapping occurrences. But it was argued in other data mining studies that counting occurrences of episodes based on a concept of sliding windows has several advantages over counting non-overlapping occurrences [6]. For this reason, this paper introduces a modified version of POERM, called POERMH (POERM with Head support), that relies on the *head support* measure[6] to find episode rules.

The rest of this paper is divided into four sections. Section 2 describes preliminaries and the problem definition. Section 3 presents the designed POERMH algorithm. Section 4 reports results of experiments. Finally, Section 5 presents a conclusion and discusses future work.

2 Problem definition

Episode mining aims at finding interesting patterns in a sequence of events, annotated with timestamps. Let $E = \{e_1, e_2, \dots e_n\}$ be a finite set of event types. A set of distinct events $R \subseteq E$ is called an **event set**. An **event pair** is a tuple $P = (t, R)$ indicating that a non empty set of events $R \subseteq E$ occurred at some time t. An **event sequence** $S = \langle (t_1, R_1), (t_2, R_2), \dots, (t_m, R_m) \rangle$ is a time-ordered list of event pairs, where $R_i \subseteq E$ for $1 \leq i \leq m$ and $0 \leq t_1 < t_2 < \dots < t_m$. For instance, consider the following event sequence: $S = \langle (1, \{c\}), (2, \{a, b\}), (3, \{d\}), (5, \{a\}), (6, \{c\}), (7, \{b\}), (8, \{d\}), (10, \{a, b, c\}), (11, \{a\}) \rangle$. This sequence contains nine event sets having timestamps ranging from 1 to 11, and five event types $E = \{a, b, c, d, e\}$. The sequence means that event c appeared at time 1, was followed by events a and b at time 2, then by d at time 3, and so on.

In recent work, it was proposed to discover a novel type of episode rules called partially-ordered episode rules (POERs) [3] to loosen the strict ordering constraint of traditional episode rules. A **partially-ordered episode rule** has the form $Y_1 \rightarrow Y_2$, where Y_1 and Y_2 are two event sets such that $Y_1 \neq \emptyset$ and $Y_2 \neq \emptyset$. Such rule is interpreted as if the events in Y_1 are observed, they are followed by those of Y_2.

Two evaluation functions have been proposed to find interesting POERs in a sequence of events, which are the non-overlapping support and confidence. They are defined as follows, based on the concept of occurrence.

Definition 1 (Occurrence). *Let there be a sequence* $S = \langle (t_1, R_1), (t_2, R_2), \dots, (t_v, R_v) \rangle$ *and three user-specified time constraints* $\eta, winlen, \alpha \in \mathbb{Z}^+$. *Moreover, let there be an event set* Y *and a POER* $Y_1 \rightarrow Y_2$. *The set* Y ***has an occurrence*** *in* S *for the time interval* $[t_i, t_j]$ *if* $Y \subseteq R_i \cup R_{i+1} \dots \cup R_j$. *The rule* $Y_1 \rightarrow Y_2$ ***has an occurrence*** *in* S *for the time interval* $[t_i, t_j]$ *if some timestamps* t_v *and* t_w *exist such that* Y_1 *has an occurrence in* $[t_i, t_v]$, Y_2 *has an occurrence in* $[t_w, t_j]$, $t_i \leq t_v < t_w \leq t_j$, $t_j - t_i < winlen$, $t_j - t_w < \eta$, *and also* $t_v - t_i < \alpha$.

Definition 2 (Non-overlapping support and confidence.). *Let there be a sequence S, an event set Y, and a rule $Y_1 \rightarrow Y_2$. The set of occurrences of Y in S is denoted as $occ(Y)$. And the set of occurrences of $Y_1 \rightarrow Y_2$ in S is denoted as $occ(Y_1 \rightarrow Y_2)$. An occurrence $[t_{i1}, t_{j1}]$ is called **redundant** in a set of occurrences if there is an overlapping occurrence $[t_{i2}, t_{j2}]$ where $t_{i1} \leq t_{i2} \leq t_{j1}$ or $t_{i2} \leq t_{i1} \leq t_{l2}$. Let $nocc(Y)$ denotes the **set of all non redundant occurrences of** Y. Moreover, let $nocc(Y_1 \rightarrow Y_2)$ denotes the **set of all non redundant occurrences of** $Y_1 \rightarrow Y_2$ in S. The **non-overlapping support** of $Y_1 \rightarrow Y_2$ is defined as $sup(Y_1 \rightarrow Y_2) = |nocc(Y_1 \rightarrow Y_2)|$. The **non-overlapping confidence** of $Y_1 \rightarrow Y_2$ is defined as $conf(Y_1 \rightarrow Y_2) = |nocc(Y_1 \rightarrow Y_2, S)|/|nocc(Y_1)|$.*

Then, the goal of episode rule mining is to find all episode rules having a support and confidence that is no less than some minimum thresholds *minsup* and *minconf* [3]. Though this definition can be useful, it was argued in other papers that considering a window based definition of the support may provide more interesting patterns [6]. Hence, this paper defines two novel measures to replace the non-overlapping support and confidence.

Definition 3 (Head support). *Let there be a POER $Y_1 \rightarrow Y_2$. The **head support of** $Y_1 \rightarrow Y_2$ in a sequence S is defined as $sup(Y_1 \rightarrow Y_2, S) = |wocc(Y_1 \rightarrow Y_2)|$ where $wocc(Y_1 \rightarrow Y_2)$ is the number of occurrences of the rule that have distinct start points.*

Definition 4 (Head confidence). *Let there be a POER $Y_1 \rightarrow Y_2$. The **non-overlapping confidence** of $Y_1 \rightarrow Y_2$ is defined as $conf(Y_1 \rightarrow Y_2) = |wocc(Y_1 \rightarrow Y_2, S)|/|wocc(Y_1)|$, where $wocc(Y_1)$ is the number of occurrences of the rule that have distinct start points.*

Then, the problem studied in this paper is to find all episode rules having a head support and confidence that are not less than some minimum thresholds *minsup* and *minconf* [3].

For instance, Let there be $minsup = 3$, $minconf = 0.6$, $\alpha = 3$, $winlen = 5$, $\eta = 1$ and the sequence: $S = \langle (1, \{c\}), (2, \{a, b\}), (3, \{d\}), (6, \{a\}), (7, \{c\}), (8, \{b\}), (9, \{a\}), (11, \{d\}) \rangle$.

The event set $\{a, b, c\}$ has three occurrences in S, which are $wocc(\{a, b, c\}, S) = \{[t_1, t_2], [t_6, t_8], [t_7, t_9]\}$. The start points of these occurrences are t_1, t_6 and t_8, respectively. The rule $r : \{a, b, c\} \rightarrow \{d\}$ has two occurrences: $wocc(r, S) = \{[t_1, t_3], [t_7, t_{11}]\}$. Hence, $supp(r, S) = 3$, $conf(r, S) = 2/3$, and R is a POER.

3 The POERMH algorithm

POERMH receives as input an event sequence and parameters *minsup*, *minconf*, α, *winlen*, and η. The first step of POERMH is to find the frequent rule antecedents. Because POERMH only considers frequent events, the algorithm scans the input sequence to discover frequent 1-event sets and record their position lists. Then, for each frequent 1-event, POERMH scans its position lists, and for

each position, POERMH tries to extend it to make 2-event sets. Then, for each i-event set, POERMH counts its head frequency occurrences, record the frequent event sets and extends them into i+1-event sets. This process is applied until all frequent event sets are found.

To count the head frequency occurrences, each event sets' position list is read. Firstly, the algorithm applies the quick sort algorithm to positions by ascending start point. Then, a variable *started* is initialized with the default value of -1. This variable contains the most recent starting point that is covered. Then, the sequence is read and each position that has a starting point that is less than *started* is ignored. For any other starting point, it is used to update *started*. At the same time, the number of valid positions is stored in a variable named *support*. After scanning the position list, *support* contains the head frequency occurrences of this event set.

The second step of POERMH is to find all consequents for each antecedent event set to make rules. This process is similar to the POERM algorithm but has some differences. For each position of an antecedent event set, it is expressed in the form of $[pos.start, pos.end]$, and POERMH searches the position $[pos.end + 1, winlen - pos.start]$ to find the consequent event sets. Different from POERM, as long as the location of one antecedent's occurrence is fixed, the range of its consequents is also fixed. Hence, a bitmap can be used for each event set to represent whether it appears after a certain position in the antecedent event sets. Then, two event sets can be compared by applying the AND operation with those two bitmaps to get a new bitmap representing the new event sets's head frequency occurrences. Then the support divided by the antecedent's support is calculated to get the confidence. If the confidence is greater than $minconf$, a valid POER is obtained. After searching all the antecedents, the algorithm terminates by returning all the valid rules.

The process of calculating the head frequency of occurrences using a bitmap is as follows. For each sorted position list obtained during the first step, a variable *started_rule* is initialized with the default value of -1. It stores the most recent starting point that is covered. Moreover, a variable *support_rule* is used to record the rule' support. Then the sorted position list is traversed to find a position such that its corresponding position in the bitmap is marked as 1. If the position's starting point is greater than *started_rule*, the *support_rule* variable is increased by one and the *started_rule* variable is updated to the starting point. And finishing scanning the position list, *started_rule* contains the head frequency occurrences of the rule.

The detailed pseudocode of POERMH is not shown due to space limitation but the Java source code is available in the SPMF pattern mining library [4].

4 Experiments

Due to space constraints, the paper only compares the performance of the POERM and POERMH algorithms on two public datasets often used in episode and pattern mining studies. Those datasets can be downloaded from the SPMF

pattern mining library [4]. The *Bible* dataset is a transformation of the *Bible* into an event sequence where each word is an event. The sequence has 649,024 events (words) selected from 13,905 distinct words (event types). The *Leviathan* dataset is a sequence of words obtained by transforming the Leviathan novel written by Thomas Hobbes (1651). The sequence has 153,682 words from 9,025 distinct word types.

Each dataset was split into two files: a **training file** (the first 75% of a sequence) and a **testing file** (the remaining 25% of the sequence). The training file is used to discover episode rules, while the testing file is used to assess the quality of predictions using these rules. For this experiment, a window of a length called **winlen** is slide over the test sequence. For each position of that window, given the first **prefix length** items, the goal is to predict the event following the **prefix length** but within **winlen** using the rules. Based on the prediction results, two measures were calculated. The **accuracy** is how many good predictions were done divided by the number of opportunities for prediction (the number of windows in the test set). The **matching rate** is how many good good or bad predictions were done, divided by the number of prediction opportunities.

The experiment consisted of varying the prefix size and applying the POERM and POERMH algorithms with a minimum confidence of 30% and 70% to assess the influence of these parameters on the accuracy and matching rate. Fig. 1 shows the results obtained by varying the prefix size parameter from 2 to 6 on the two datasets. It is observed that the prefix length has a strong influence on results. Increasing the prefix length generally results in more rules, which increases the matching rate. But at some point increasing the prefix size has less and less influence, which is reasonable. In terms of accuracy, it can decreases when the prefix length is increased but this is due to the increase in matching rate. In terms of confidence, it is found that POERM and POERMH have better accuracy and matching rate when run with a minimum confidence of 30% instead of 70 %, as more rules are found. On overall, POERMH's accuracy and matching rate are respectively up to 5% and 17% higher than POERM.

Due to space constraints, results about the time efficiency and memory usage are not presented. But in general, the time and space consumed by POERM and POERMH are about the same. POERMH thus provides an improvement over POERM as it can obtain higher accuracy and matching rate while consuming about the same amount of resources.

5 Conclusion

This paper has presented a new algorithm to find partially-ordered episode rules named POERMH. This algorithm relies on the head support measure to identify interesting episode rules. Experiments have shown that the algorithm has good sequence prediction performance compared to the benchmark POERM algorithm. In future work, we will consider adding other optimizations to POERMH,

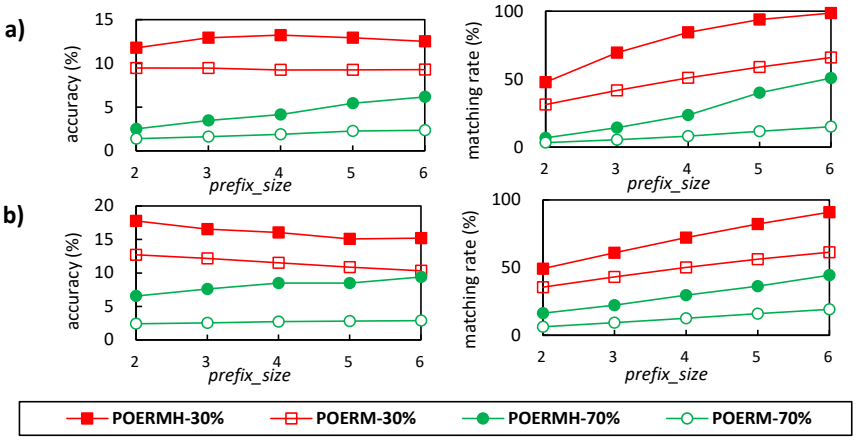

Fig. 1: Influence of $prefix\_size$ on (a) Bible and (b) Leviathan

developing a version for distant event prediction [2] and for online mining [9], and finding other types of episode rules [8].

References

1. Ao, X., Luo, P., Wang, J., Zhuang, F., He, Q.: Mining precise-positioning episode rules from event sequences. IEEE Transactions on Knowledge and Data Engineering **30**(3), 530–543 (2017)
2. Fahed, L., Lenca, P., Haralambous, Y., Lefort, R.: Distant event prediction based on sequential rules. Data Science and Pattern Recognition **4**(1), 1–23 (2020)
3. Fournier-Viger, P., Chen, Y., Nouioua, F., Lin, J.C.: Mining partially-ordered episode rules in an event sequence. In: Proc. 13th Asian Conference on Intelligent Information and Database Systems. pp. 3–15. Springer (2021)
4. Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The spmf open-source data mining library version 2. In: Joint European conference on machine learning and knowledge discovery in databases. pp. 36–40. Springer (2016)
5. Fournier-Viger, P., Yang, Y., Yang, P., Lin, J.C.W., Yun, U.: Tke: Mining top-k frequent episodes. In: Proc. 33rd Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer (2020)
6. Huang, K., Chang, C.: Efficient mining of frequent episodes from complex sequences. Inf. Syst. **33**(1), 96–114 (2008)
7. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovering frequent episodes in sequences. In: Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining (1995)
8. Ouarem, O., Nouioua, F., Fournier-Viger, P.: Mining episode rules from event sequences under non-overlapping frequency. In: Proc. 34th Intern. Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems (2021)
9. You, T., Li, Y., Sun, B., Du, C.: Multi-source data stream online frequent episode mining. IEEE Access **8**, 107465–107478 (2020)