Mining Patterns With Durations from E-commerce Dataset

Mohamad Kanaan, Hamamache Kheddouci

Mining Patterns With Durations from E-commerce Dataset

Mohamad Kanaan[1] and Hamamache Kheddouci[2]

[1] Sistema Strategy, Lyon, France,
mohamad.kanaan@sistema-strategy.com,
WWW home page: http://www.sistema-strategy.com/Home
[2] Université Claude Bernard Lyon 1, Laboratoire LIRIS, Lyon, France
hamamache.kheddouci@univ-lyon1.fr,
WWW home page: http://perso.univ-lyon1.fr/hamamache.kheddouci

Abstract. Given a dataset of clickstream extracted from e-commerce logs, can we find a clear usage of the website? Are there hidden relationships between the purchased products? Are there any discriminatory behaviors leading to the purchase? To answer these questions, we propose in this paper a new **S**equential **E**vent **P**attern **M**ining algorithm (SEPM). The endeavor is to mine clickstream data in order to extract and analyze useful sequential patterns of clicks. Also, in order to make these patterns clearer, the time spent on each page is taken into account. SEPM maintains the items durations during the mining process and extracts patterns with the average durations of these items without multiple scans of the dataset. Our experimental results on both real and synthetic datasets indicate that SEPM is efficient and scalable.

Keywords: Data mining, frequent pattern, customer behavior, e-commerce

1 Introduction

Over the last years, electronic (e)-commerce has revolutionized the retail by changing the way in which the clients purchase. User's choices are no longer limited by the product availability in the stores of his region. Today, he can search and buy products from any international store, anywhere and anytime, by using e-commerce websites. However, e-merchants have some difficulties in understanding their customers' behaviour. They want to know how customers purchase products, how they navigate through the products catalogs or even why they abandon their purchase process. Such behaviors are very complex since they are influenced by (1) supplying factors of the e-commerce website such as prices, product availability, delivery time, rating, users' opinion, etc., (2) and external factors, such as international events (e.g. Black Friday), holidays, customer's budget, etc. These behaviors help e-merchants to understand and discover customers' needs. Therefore, e-merchants can recommend relevant products to their customers, predict future purchases, and also ensure products' availability. Their key tasks are more focused on identifying and understanding the abandonment

and purchasing processes, starting from the first stimulus of customer until his decision.

To achieve this goal, users' actions could be traced. These actions are known as clickstream. To analyze them and extract useful information, several mining tools have been developed. They aim to analyze the user's journey and discover their hidden behaviors. Some well known data mining techniques are: pattern mining, trend discovery, customers clustering and classification, collaborative filtering in recommending systems, and purchase prediction.

In this paper, we are interested in pattern mining. This mining tools is used to discover the hidden relationships between the products (the product network), and to extract discriminatory behaviors. Therefore, two aspects were taken into consideration: the order of browsed products and the time spent by customer on the page of this product. These aspects can help e-merchant to find more discriminatory behaviors and mine more precisely the pattern's skeleton. In data mining, this task is known as "Sequential pattern mining".

This paper is organized as follows. Section 2 presents a state of art of the sequential pattern mining. In section 3, a new algorithm is developed to mine sequential event pattern. The experimental results are shown in section 4. Finally, the conclusion is elaborated in section 5.

2 Related Work

Sequence pattern mining is a challenging task that aims to discover frequent patterns from a sequential database. It was proposed in [7] as a problem of mining customer's sales transactions in order to discover frequent sequences of purchasing constrained by a user-specified minimum support. Later, several efficient algorithms were developed [12], and can be divided into two categories: candidate generation and pattern growth.

1. Candidate generation: GSP [6] and SPADE [1] are the two well-known algorithms in this category. These algorithms are extensions of Apriori algorithm, and they are based on two main steps: candidate generation and support counting.
   (a) GSP algorithm: it extracts the patterns level by level. At each level k, all possible frequent patterns with k items (called k-patterns) are mined. To start, GSP scans the database at the level k=1 to identify all single frequent items. Then, it generates for next level k+1 all potentially patterns (k+1)-candidates. The (k+1)-candidates are generated by joining k-patterns with itself. For example, considering P_1, $P_2 \in k\text{-}patterns$ such as P_1 and P_2 have the same items except the last one: a new (k+1)-candidate is generated by joining P_1 with the last item of P_2, and another (k+1)-candidate is generated by joining P_2 with the last item of P_1. When all (k+1)-candidates are generated, GSP re-scans the database to count their supports and keeps only the most frequent among them. This process is repeated until no more patterns are found.

(b) SPADE algorithm: it converts the original sequences database (in horizontal format) to a vertical database. In this vertical database, each table is called **IDList** and contains a frequent item with all its positions in the original database. The vertical database is built by scanning the original database twice. The main steps of SPADE are similar to those of GSP but without multi-scan. In fact, IDList is used also to represent a pattern. For example, the IDList of a pattern P contains all items' positions in the original database where P occurs. Thereby, the support of a pattern can be defined as the number of distinct id of the sequence list in its IDList (discribed in details in Sect. 3). Also, by using IDList, the joining operation between two patterns can be done without re-scanning the database. It becomes a simple join of their IDList.

2. Pattern growth: most of algorithms in this category ([2], [3] and [10]) are inspired by depth-first search algorithm. One of well-known algorithms is PrefixSpan [3]. It avoids the recursive exploring of its original database (which is very expensive) by projecting it to a set of smaller databases. Then, in each local database, patterns are grown by appending items to them to create larger patterns. To avoid creating the same pattern twice during the pattern growing, the items are appended according to a total order $\prec$ which can be a lexicographical order or any other total order on items.

3 Sequential Event Pattern Mining Algorithm

3.1 Definitions

Definition 1. *Event: An event is denoted by* $\boldsymbol{E}$ *= (label, duration) where E.label and E.duration denotes the event label and duration (> 0) respectively.*

Definition 2. *List of sequential events: A list of sequential events* $\boldsymbol{SE} = \{E_1, E_2, ..., E_n\}$ *is a sequence of events sorted by their starting times in ascending order (we assume that events will be appended to the SE by order).*

SE is identified in the original database by a unique identifier called **SID**, and each of its events E has also a unique identifier called **EID**. The EID indicates the position of E in SE. The length of SE is given by the number of its events $l = |SE|$.

A canonical representation of SE can be obtained by merging all the labels of its events in their orders (e.g., in Table 1, SE with $SID = 3$ can be represented by $< ACBD >$).

At this point, it is worth noting that for two given events E_i and E_j, we denote $E_i \prec E_j$, if $1 \leq \{i, j\} \leq l$, $\{E_i, E_j\} \in SE$ and E_i occurs before E_j in SE. As events are appended by order of their start time to SE, the total order $\prec$ on events can be reduced to a simple comparison test of their EID $\{i, j\}$ in SE.

The input database (list of SE) is represented in vertical format [1]. Each distinct event in the database becomes an **IDList** containing the label and the positions list $< SID, EID >$ of the event.

Table 1. Example database of event lists and patterns detected

SID	Events	Event list ($\text{Label}_{\text{eid}}$)
1	E_1 (A, 7), E_2 (B, 15) E_3 (D, 6), E_4 (C, 3)	$EL_1 = A_1 \rightarrow B_2 \rightarrow D_3 \rightarrow C_4$
2	E_1 (D, 2), E_2 (C, 6)	$EL_2 = D_1 \rightarrow C_2$
3	E_1 (A, 10), E_2 (C, 8) E_3 (B, 5), E_4 (D, 8)	$EL_3 = A_1 \rightarrow C_2 \rightarrow B_3 \rightarrow D_4$
4	E_1 (B, 3), E_2 (D, 9) E_3 (C, 4)	$EL_4 = B_1 \rightarrow D_2 \rightarrow C_3$

Patterns detected
A (8.5) , B (7.67)
C (5.25) , D (6.25)
A (8.5) B (10)
A (8.5) C (5.5)
A (8.5) D (7)
B (9) C (3.5)
B (7.67) D (7.67)
D (5.67) C (4.33)
A (8.5) B (10) D(7)
B (9) D (7.5) C(3.5)

Definition 3. *Subsequence: given two sequence list* $SE = \{E_1, E_2, ..., E_k\}$ *and* $SE' = \{E'_1, E'_2, ..., E'_k\}$, $SE \subseteq SE'$, *if and only if there exists an injective function* f: $SE.events \rightarrow SE'.events$ *such that for any* $\{E_i, E_j\} \in SE$, *if* $E_i \prec E_j \rightarrow f(E_i) \prec f(E_j)$ *and* $\{f(E_i), f(E_j)\} \in SE'$. *For example,* $EL_4 \subset EL_1$ *and* $EL_2 \nsubseteq EL_3$ *can be deduced from Table 1.*

Definition 4. *Pattern: a pattern is a frequent k-subsequence where k is the number of its items.*

Patterns are also represented in vertical format. Each pattern is associated with an $IDList$ containing a list of labels of its items, and a list of $< SID, EID >$ where its items occur in the database. This vertical representation enables a very fast compute of support of the pattern $PSupp$, by simply counting the number of distinct SID in its $IDList$.

A valid pattern is constrained as follows: (1) *frequency:* $PSupp \geq minSupp$, where $minSupp$ is a user-specified minimum support threshold, (2) *event order:* $\forall\{Item_i, Item_j\} \in pattern$, if $1 \leqslant i < j \leqslant k$ ($Item_i$ is discovered before $Item_j$) and $Item_i.SID = Item_j.SID \rightarrow Item_i.EID < Item_j.EID$.

3.2 Problem Reformulation

Let D be an input database of SE and $minSupp$ a user-specified minimum support threshold. The goal is to find all possible and valid patterns including the average durations of their items. To achieve this goal, we choose to maintain the durations of the items in patterns during the mining process to accelerate the computing of the average durations at the end. For this reason, we choose to augment the representation of $IDList$ by introducing $IDListExt$.

Definition 5. *IDListExt: is an extension of* $IDList$. *It contains in addition, a list of neighbors* $INeighbors$ *and a list of durations* $IDurations$.

Definition 6. *INeighbors: is the list of items that follows the last item in pattern (or the item of event) in at least minSupp sequence lists. For example, in Table 1,*

for $minSupp = 2$, the neighbors of A are $\{< B, freq = 2 >, < C, freq = 2 >, < D, freq = 2 >\}$ and those of D are $\{< C, freq = 3 >\}$. The list of neighbors can help pattern growing and avoid false candidates (cf. Sec. 3.4).

Definition 7. *IDurations: is the list of durations of the item in each $< SID, EID >$ where it occurs. $IDurations$ of a pattern is the list of durations of its last item, collected from the original $lastItem.IDList$ where the pattern occurs. An example of $IDListExt$ is shown in Table 2 for items $\{A, B, C, D\}$ and pattern $\{AB\}$ extracted from database in Table 1.*

Table 2. Example of IDListExt built from Table 1 with minSupp = 2

A		
SID	EID	Duration
1	1	7
3	1	10
INeighbors: { B, C, D }		

B		
SID	EID	Duration
1	2	15
3	3	5
4	1	3
INeighbors: { C, D }		

C		
SID	EID	Duration
1	4	3
2	2	6
3	2	8
4	3	4
INeighbors: { ∅ }		

D		
SID	EID	Duration
1	3	6
2	1	2
3	4	8
4	2	9
INeighbors: { C }		

AB		
SID	EID	Duration
1	2	15
3	3	5
INeighbors: { C , D }		

3.3 Algorithms

We describe here our proposed algorithms to build IDListExt and detect patterns with SEPM.

Build IDListExt: Algorithm 1 is proposed to build the IDListExt of each frequent event in the database. It starts by scanning the database to obtain all single frequent events $f1$ (infrequent events can be removed from database to accelerate the second scan). A second scan is performed to build the IDListExt of each frequent event (line 5-13). The last operation (line 15) will remove all infrequent neighbors from all INeighbor.

Extract pattern with SEPM: Algorithm 2 is proposed to extract all patterns without scanning the original database. It is based on two main steps: generating candidates, and then filtering them. It starts by generating the 1-patterns from every item IDListExt (line 1). For each item and for all its SID, SEPM picks the minimum EID and the duration associated with it.

Then at each level k, (k+1)-candidates are generated from the found k-patterns. From each pattern in k-patterns, new candidates are generated by joining the pattern with every item in its IDListExt.INeigbor (line 5-18) called neighbor. When the support of a candidate meets the requirements (line 10), a

Algorithm 1 Build all IDListExt

Input: Event List *db*

```
1: f1 ← all single frequent events
2: for all (el ∈ db) do
3:     antecedents ← ∅
4:     EID ← 1
5:     for all (e ∈ el.events) do
6:         if f1 contains e then
7:             IDListExt ← findOrCreateIDListExt(e.label)
8:             IDListExt.add(el.SID, EID, e.duration)
9:             add IDListExt.item to all INeighbor in antecedents
10:            antecedents ← antecedents ∪ IDListExt
11:            EID ← EID + 1
12:        end if
13:    end for
14: end for
15: remove infrequent neighbors from all INeighbors
```

new pattern is created from: (1) current pattern, (2) neighbor, (3) and IDListExt resulting from the join between pattern.IDListExt and neighbor.IDListExt.

The average durations of the items in a pattern P can be obtained from the list of durations in its IDListExt and those in all its predecessors. This operation can be performed by recursively exploring the predecessors of P and obtaining duration of each position <SID, EID> where SID appears in P.IDListExt. For example, we obtain from the database in Table 1, a 1-pattern $< A >$ that has a neighbor B. When $< A >$ joins its neighbor B, a new pattern $< AB >$ is created (see Table 3). In this new pattern $< AB >$, we keep only the durations of the last item B. We can get those of item A, by referring to the pattern $< A >$ the predecessor of $< AB >$.

Table 3. Example of 2-pattern $< AB >$ and 3-pattern $< ABD >$ with minSupp = 2

AB		
SID	EID	Duration
1	2	15
3	3	5
Output: **A** (8.5) **B** (7.67)		

ABD		
SID	EID	Duration
1	3	6
3	4	8
Output: **A** (8.5) **B** (10) **D** (7)		

3.4 Pruning

Two pruning mechanisms are proposed to reduce the search space and accelerate the execution time. These mechanisms eliminate false candidates and avoid unnecessary join operations.

Algorithm 2 SEPM

Input: Set of IDListExt $setIDListExt$

```
1: patterns ← obtain 1-pattern from setIDListExt
2: while patterns ≠ ∅ do
3:     Print patterns
4:     candidates ← ∅
5:     for all (pattern ∈ patterns) do
6:         for all (neighbor ∈ pattern.IDListExt.INeighbors) do
7:             if neighbor ∉ pattern.blackList and pattern.canJoin(neighbor) then
8:                 neighborIDListExt ← setIDListExt.find(neighbor)
9:                 newIDListExt ← pattern.IDListExt join neighborIDListExt
10:                if newIDListExt.|SID| ≥ minSupp then
11:                    newPattern ← createPattern(pattern, neighbor, newIDListExt)
12:                    candidates ← candidates ∪ newPattern
13:                else
14:                    pattern.blacklist ← pattern.blacklist ∪ neighbor
15:                end if
16:            end if
17:        end for
18:    end for
19:    patterns ← candidates
20: end while
```

1. The first mechanism is based on the following property:

   If joining a pattern P with an item I generates an infrequent pattern $P' \rightarrow$ adding I to any superpattern of P will generate an infrequent pattern

   To handle this property, a *blacklist* of items is built and if an item cannot join a pattern, it will be added to the blacklist of this pattern. Then we can avoid joining items with patterns when they appear in their *blacklists*.
2. The second mechanism is based the following property:

   Let the item N be the neighbor of a pattern P. If N is not neighbor of all items in $P \rightarrow$ joining P with N will generate an infrequent pattern

   Based on this property, we can safely reject candidates resulting from joining a pattern with an item that do not appear in all INeighbors of items in P.

Fig. 1 shows the tree of patterns mined from the database in Table 1. Red nodes represent the candidates rejected by the pruning mechanism without any join operation.

4 Experimental Evaluation

SEPM was applied on several real and synthetic datasets to study its performance. The real datasets were used to demonstrate the relevance of the mined

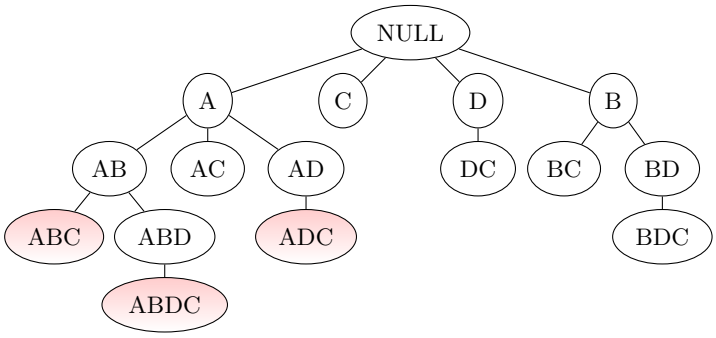


Fig. 1. Tree showing the patterns and their children. The red nodes are rejected.

patterns, and the synthetic datasets to prove the efficiency and the scalability of the algorithm. All experiments were performed on a computer with Core i5 processor, running on Windows 10 and 16 GB of free RAM.

4.1 E-commerce Datasets

Datasets. The experiments were carried out on three real datasets presented in Table 4. These datasets only contain clicks on the products. The minimum support is set to 0.01% for experimental purposes. When it changes, it affects directly the number and length of the found patterns. Since it is hard to define a common value for all datasets, this threshold must be fixed by analysts according to their needs and their dataset dimensions.

Table 4. Three real datasets used in experiments

Dataset	Event lists	Events	Nb Patterns	Max Length	Reference
cikm	238k	2,121k	25,855	6	[14]
yoochoose	670k	3,870k	32,474	9	[15]
alibaba	1,119k	13,966k	10,923	6	[16]

Preparing data. To be able to apply SEPM, raw clickstream data must be converted into sequential event data by following these steps: (1) every session is converted into an event-list, (2) and every click in a session is converted into an event attached to its corresponding event-list. The duration of an event is defined as the time between the start time of a click and that of the next click in the same session (we suppose that the user will not be on two pages at the same time). All the clicks durations in a session can be obtained except the duration of the last one as it is hard to know when the user left the website. This duration can be replaced by the average duration of all clicks in the current session.

Patterns categorization. Table 5 shows some 4-patterns found. As we can see, this temporal representation of patterns is very useful to analyze the product network and discover the discriminatory behaviours. We notice that the durations of items in the same pattern are not equal all the time. The gap between them is sometimes small, and sometimes very important. Based on this observation, we categorize the patterns according to two variants: the standard deviation of the durations called **sd**, and the number of variations of the durations called **variation**, such as:

$$sd_p = \sqrt{\frac{\sum_{i=1}^{l_p}(d_i - v)^2}{l_p}} \tag{1}$$

where l_p is the length of pattern **p**, d_i is the duration of its i^{th} item, and v is the mean of all durations in p,

$$variation_p = \sum_{i=1}^{l_p-1}(v_i) \tag{2}$$

where v_i is a Boolean indicating whether there is a significant gap between the durations of $(i)^{\text{th}}$, and $(i+1)^{\text{th}}$ items, such as $v_i = 1$ if $|d_{i+1} - d_i| \geq \sigma$, and 0 otherwise ($\sigma$ is a user-specified threshold, set to 1 minutes in our tests). To distinguish the direction of variation, we denote v_i^+ the positive variation if $d_{i+1} \geq d_i$ and v_i^- the negative variation in the other case.

Table 5. Example of 4-patterns found

Label of product (duration in minutes)						
45925(0.6)	→	90884(0.18)	→	36425(3.97)	→	10450(1.9)
8644(2.74)	→	768(0.77)	→	8644(10.92)	→	48580(3.92)
36343(5.18)	→	36343(8.78)	→	31163(4.11)	→	31163(0.73)
54421(0.32)	→	11338(2.36)	→	33890(1.23)	→	4759(1.35)
11338(0.95)	→	4926(1.35)	→	54421(0.52)	→	11338(3.12)

Based on these variants, five important categories are identified:

Category-A: includes standard patterns, with $variation = 0$ and $sd \simeq 2.57$. The products found in these patterns have strong relationships with each other. Users spend almost the same duration when they consult these products together. These patterns can be used to recommend products to users during their navigations.

Category-B: includes patterns with $variation = 1$ and $sd \lesssim 12.57$. In this category, two subcategories of patterns are discovered depending on the direction of variation of their durations (positive or negative variation). The patterns with positive variation indicate that users, before focusing on their main products,

consult some related ones (e.g. mobile phone accessories before mobile phones, printer cartridges before printers, ...). These related products, even if they are not bought in the same session, can influence directly the users' opinions. These patterns can help e-merchants to offer new products collections. In the other case, when the patterns contain a negative variation, we notice that users are moving away from their target products (most sessions containing these patterns end with quick navigation without focusing on a particular product).

Category-C: includes patterns with $variation = 2$ and $sd \lesssim 25.13$. Many patterns in this category contain noises especially when the first variation is negative and the second one is positive. In this case, the users may consult non-target products during their navigations.

Category-D: includes patterns with $variation \geq 3$ and $sd \lesssim 30.33$. These patterns reflect different types of customer's behaviors, e.g. product comparison, random navigation without purchase intention, etc.

Category-E: includes patterns with $sd \geq 31$. Most of these patterns reflect abnormal behaviours. This can be due to several reasons: bugs in tracing tool, error while page loading, etc.

Based on these categories, several mining techniques can be applied to understand more the customers' behaviors. E.g. customers can be categorized based on frequent pattern categories in their sessions, and then each customer's profile can be treated separately. Also, various customers' demographics data can be combined with the patterns categories to recommend more personalized websites.

4.2 Synthetic Datasets

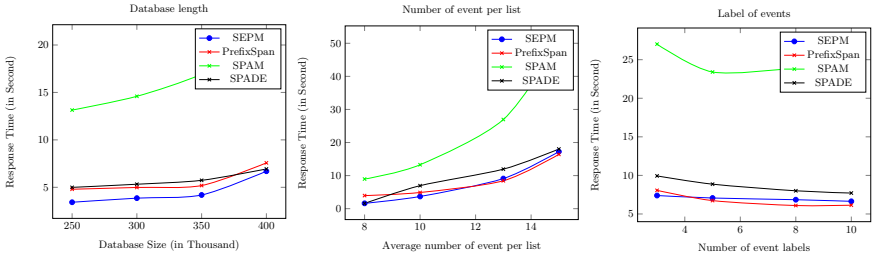
In this section, we apply four algorithms (SPEM, PrefixSpan, SPAM and SPADE) on synthetic datasets. The source codes of algorithms (PrefixSpan, SPAM and SPADE) are provided by [13].

Datasets. We have examined the effect of varying some dimensions of the database. We used for these experiments the "IBM data quest generator" to generate synthetic datasets. The generation of datasets can be controlled through several parameters:

1. ncrust (number of customers in the database) to vary the number of event lists (i.e. **D**)
2. slen (average of transactions per customer) to vary the number of events per list (i.e. **L**)
3. nitems (the number of different items available) to vary the type (label) of events (i.e. **T**)

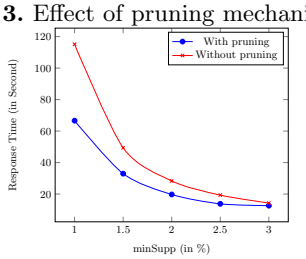
To analyze the effect of D variation, we set minSupp to 10% and we vary D between 250k and 400k. In the case of L, minSupp is set to 10% and L is varied between 8 and 15. Lastly, for that of T, minSupp is set to 5% and T is varied between 3 and 10. Fig. 2 shows the results. In all cases, SEPM behaves in the same way as the other algorithms when these parameters change. Results show us that SEPM behaves correctly according to the different types of database.

Fig. 2. Effect of varying the dimensions of the database



Pruning. Finally, we investigate the effectiveness of the proposed pruning mechanisms. For this test, two variations of SEPM are implemented with and without pruning. Fig. 3 shows the results when they are applied on a synthetic dataset of 300k event list. As we can see, the pruning mechanisms are very efficient and can decrease the execution time by avoiding the generation of false candidates.

Fig. 3. Effect of pruning mechanisms



5 Conclusion

The duration that takes the customer to check products in an e-commerce website is crucial to understand his preference. Motivated by this reason, we have examined in this paper the problem of mining sequential pattern from datasets of clickstream. We augmented the existing vertical database representation with

additional duration information to detect patterns with the average durations of its items. Then based on this new representation, we have proposed a new algorithm called SEPM to detect sequential patterns including the average durations of their items without multiple scans of the database. These patterns are very useful for building the product network and discovering the strong and hidden relationships between products. They are also categorized to simplify their interpretations and detect the discriminatory behaviors. Experimental results on real and synthetic datasets show the efficiency and the scalability of our proposed algorithm.

References

1. Zaki, M.J.: SPADE: An efficient algorithm for mining frequent sequences. Machine Learning 42(1), 31–60 (2001).
2. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. C. Hsu, "FreeSpan: frequent pattern-projected sequential pattern mining" ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 355-359, 2000.
3. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, pp. 3–17. Springer, Heidelberg (1996).
4. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential pattern mining using a bitmap representation. In: Proc. 8th ACM SIGKDD Intern. Conf. Knowledge Discovery and Data Mining, pp. 429–435. ACM (2002).
5. Svatošová, Veronika. (2013). Motivation of Online Buyer Behavior. Journal of Competitiveness. 5. 14-30. 10.7441/joc.2013.03.02.
6. R. Srikant, and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements" The International Conference on Extending Database Technology, pp. 1-17, 1996.
7. R. Agrawal and R. Srikant. Mining Sequential Patterns. In Proc. of the 11th Int'l Conference on Data Engineering, Taipei, Taiwan, March 1995.
8. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In VLDB'94, pp. 487-499.
9. Patel, D., Hsu, W., Lee, M.L.: Mining relationships among interval-based events for classification. In: Proceedings of ACM SIGMOD, pp. 393–404 (2008)
10. J. Han, J. Pei, Y. Ying, and R. Mao, "Mining frequent patterns without candidate generation: a frequent-pattern tree approach" Data Mining and Knowledge Discovery, vol. 8(1), pp. 53-87, 2004.
11. J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-mine: Hyper-structure mining of frequent patterns in large databases" IEEE International Conference on Data Mining, pp. 441–448, 2001
12. P. Fournier-Viger, J. C. W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of sequential pattern mining," Data Sci. Pattern Recognit., vol. 1, no. 1, pp. 54–77, 2017.
13. http://www.philippe-fournier-viger.com/spmf/
14. https://competitions.codalab.org/competitions/11161
15. http://recsys.yoochoose.net/challenge.html
16. https://tianchi.aliyun.com/datalab/dataSet.html?dataId=649