doi:10.3969/j.issn.1001-4616.2023.01.016

基于滑动窗口的数据流高效用模糊项集挖掘

单芝慧,韩　萌,韩　强

(1.北方民族大学计算机科学与工程学院,宁夏 银川 750021)
(2.北方民族大学图像图形智能处理国家民委重点实验室,宁夏 银川 750021)

[**摘要**]　高效用项集挖掘可以提供有趣的结果集,但并不能提供单个项的数量,因此,本文提出了高效用模糊项集. 但是,现实世界的数据是不断出现的,需要实时处理新到来的数据. 为解决当前高效用模糊项集不能处理数据流的问题,又提出了模糊效用列表(fuzzy utility list,FUL)结构用于存储当前窗口中项的批次号、项在事务中的事务标识符、项的模糊效用以及项的剩余模糊效用,该结构能有效的对批次进行插入和删除操作. 最后,基于 FUL 提出了数据流高效用模糊项集挖掘算法. 对真实数据集和合成数据集进行了广泛的实验,结果证实了算法的效率及可行性.

[**关键词**]　数据流挖掘,滑动窗口,高效用项集挖掘,模糊效用,效用列表
[**中图分类号**]TP391　[**文献标志码**]A　[**文章编号**]1001-4616(2023)01-0120-10

High Utility Fuzzy Itemsets Mining Over Data Stream Based on Sliding Window Model

Shan Zhihui,Han Meng,Han Qiang

(1.School of Computer Science and Engineering,North Minzu University,Yinchuan 750021,China)
(2.The Key Laboratory of Images & Graphics Intelligent Processing of State Ethnic Affairs Commission,
North Minzu University,Yinchuan 750021,China)

Abstract:High-utility itemsets mining(HUI)can provide interesting itemsets,but cannot provide information on the number of items. Therefore,high utility fuzzy itemsets are proposed. However,real-world data is constantly emerging. Thus,new incoming data needs to be processed in real time. To solve the problem that the current high utility fuzzy itemsets cannot handle the data stream,a fuzzy utility list(FUL)structure is proposed to store the information of items,including batch number of items,the transaction identifier of the items,the fuzzy utility of items,and the reminding fuzzy utility of items. FUL can effectively insert and delete batches. Finally,based on FUL,a high utility fuzzy itemset mining algorithm on data stream is proposed,extensive experiments on real and synthetic datasets show the efficiency and feasibility of the algorithm.

Key words:data stream mining,sliding window,high utility itemsets mining,fuzzy utility,utility list

高效用项集挖掘(high utility itemset mining,HUIM)[1]是数据挖掘领域中一个热点的研究课题. HUIM 发现数据集中具有高利润的项集组合,即高效用项集(high utility itemset,HUI). 之后,又不断提出了 HUI 的扩展项集,如闭合 HUI(closed HUI,CHUI)[2],高平均效用项集(high average utility itemset,HAUI)[3]等. 在不同类型数据集上也有相应的模式挖掘,如序列数据集上的序列项集挖掘[4],空间数据集上的空间高效用项集挖掘[5-6]等.

但这些算法仅提供项集,并未提供项集中单个项的数量信息,不利于用户制定更精准的决策策略. 为了同时考虑高效用的项集及项集中单个项的数量信息,研究者提出了高效用定量项集(high utility quantitative itemset,HUQI)[7]挖掘算法,但其结果集存在冗余问题. Wang 等[8]还提出了模糊项集挖掘,将模糊集理论与效用挖掘结合,从定量数据集中挖掘高效用模糊项集(high utility fuzzy itemset,HUFI). Wang 等[8]还定义了一个模糊效用函数来评估数据集中项目的模糊效用,将项目的数量信息转化为模糊域. 例如,一个量化事务

收稿日期:2022-08-08.
基金项目:国家自然科学基金项目(62062004、61862001)、宁夏自然科学基金项目(2020AAC03216).
通讯作者:韩萌,博士,教授,研究方向:数据挖掘. E-mail:2003051@nmu.edu.cn

{(A,3),(C,5),(D,8)},其中符号和数字分别代表购买的物品及其购买数量. 假设三个物品为如图 1 所示的隶属度函数,将项的数量信息转化为三个模糊区域:Low、Middle 和 High. (A,3)中的 3 可以转化为占比 0.6 的 Low 和 0.4 的 Middle. 量化事务{(A,3),(C,5),(D,8)}可以转化为{(0.6/A.Low,0.4/A.Middle),(0.2/C.Low,0.8/C.Middle),(0.6/D.Middle,0.4/D.High)},然后根据相关定义计算模糊项集的效用,挖掘 HUFI. 使用 HUFI 不但可以给用户提供高效用的项集,而且可以说明不同项目如何在产品组合中获得合适的产品推广数量,为用户提供更详细的信息.

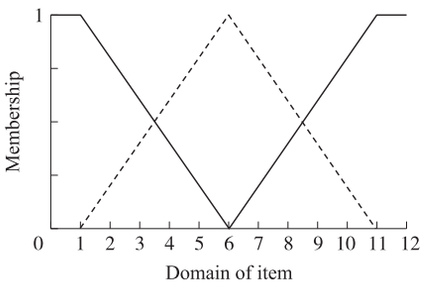

图 1　隶属度函数

Fig. 1　Membership function

随着大数据时代的到来,每时每刻都有新数据在产生,如何高效的处理新产生的数据获取更详细的信息具有一定的研究意义. 研究者提出了在数据流上挖掘 HUI 及其扩展模式,但都是提供高效用的项集并不提供项集中单个项的数量信息. 为了获取数据流上更详细的数据信息,本文提出了基于滑动窗口的数据流 HUFI 挖掘算法. 具体来说,本文的主要贡献包括 3 个方面:

(1)提出了模糊效用列表(fuzzy utility list,FUL)结构存储项的模糊效用信息,该结构能够有效的插入和删除批次中项的模糊效用信息;

(2)基于 FUL 提出了基于滑动窗口模型的数据流 HUFI 挖掘算法 FHUI_DS(fuzzy high utility itemset on data stream,FHUI_DS)与 FHUI_Native(fuzzy high utility itemset native,FHUI_Native);

(3)为了评估提出的算法,在真实数据集与合成数据集上进行了广泛的实验. 实验结果表明,提出的算法能够有效挖掘数据流 HUFI.

1　相关工作

由于 HUI 只提供产品组合的各个项,而不包含单个项的数量信息,故 Wang 等[8]提出了模糊效用挖掘,定义了模糊效用函数,通过其隶属度函数中相应的语言区域值和度值来评估项的模糊效用. 之后,Lan 等[9]提出了一种两阶段模糊效用挖掘算法(two-phase fuzzy utility,TPFU),使用模糊最小算子来评估项集的模糊效用. 但该算法使用两阶段算法,需要产生大量的没有希望的候选项集,消耗时间. Lan 等[10]进一步提出了渐进式数据缩减模糊效用挖掘算法(gradual data-reduction fuzzy utility,GDFU),该算法运行时间优于 TPFU. Hong 等[11]提出了一阶段时间模糊效用挖掘算法,这种算法可以找到所有高时间模糊效用项集,能够以更少的内存和更快的执行速度获得良好的性能. 研究者还将遗传算法、进化算法等群智能优化算法与 HUFI 相结合[12-13],改善算法性能. 但是当前的 HUFI 挖掘算法仅适用于静态数据,当新数据插入时,需要重新扫描整个数据集,极其耗时.

在现实生活中,数据是不断产生的,及时处理新产生的数据是很必要的. 宋威等[14]提出了基于事务型滑动窗口的数据流高效用项集挖掘算法(mine high utility itemsets over data stream,MHUIDS),使用二进制向量表示数据,基于高事务加权效用项集树对候选项集搜索空间剪枝. Tsai 等[15]提出了一种基于加权滑动窗口模型的 HUIM 一阶段算法. 但该算法产生了大量错误的候选项集,消耗了大量的内存. Jaysawal 等[16]提出了一阶段算法(single-pass one-phase algorithm for mining high utility patterns over a data stream,SOHUPDS)挖掘数据流上的高效用项集,该算法使用投影数据库方法和滑动窗口技术挖掘 HUI,具有较好的性能. 研究者还提出了在数据流上挖掘 top-k 高效用项集[17]和闭合高效用项集[18]. 但是当前数据流上的算法仅能挖掘 HUI 及其扩展模式,并不能提供项集中单个项的数量信息. 为了处理数据流上的 HUFI,本文提出了基于滑动窗口的 HUFI 挖掘算法.

2　问题定义

本节描述了从定量数据集中挖掘 HUFI 的相关定义. $I=\{c_1,c_2\cdots,c_m\}$ 为一组具有 m 个项的集合,$D=\{T_1,T_2,\cdots,T_n\}$ 是一组数据库事务,其中 $T_i=(v_1^{(i)},v_2^{(i)},\cdots,v_p^{(i)},\cdots,v_m^{(i)})$,$v_p^{(i)}$,$p=1,\cdots,m$ 是事务 T_i 中物品 $c_p\in I$ 的购买数量. 数量信息由隶属度函数转化为模糊域. 项目 $c_p\in I$ 的第 q 个模糊区域,表示为模糊项目 $c_{p(q)}$,由模糊隶属函数 $\mu_{cp(q)}$ 表示,其中 $1\leqslant q\leqslant q_p$,并且 q_p 是项目 c_p 的模糊区域最大值(用户定义).

定义 1[8]　模糊度. 事务 T_i 中 $v_p^{(i)}$ 的模糊项 $c_{p(q)}$ 的模糊度定义如下:$f_{p(q)}^{(i)}=\dfrac{\mu_{cp(q)}(v_p^{(i)})}{\sum_j \mu_{c_{p(q)}}(v_p^{(i)})}$.

例如,在图 2 与表 1 中的数据流示例中,通过图 1 所示的隶属度函数将项的数量转化为模糊域. 在事务 T_1 中项 a 的数量为 2,即 $v_a^{(1)}=2$,转化为 $\mu_{a(L)}$ 与 $\mu_{a(M)}$ 分别是 0.8 和 0.2.

定义 2[10]　加权外部效用. 模糊项 $c_{p(q)}$ 的加权外部效用定义为项的外部效用与项所在模糊域的数量的乘积,表示为 $U_w(c_{p(q)})=p_j\times[\operatorname{argmax}_{v_p^{(i)},\forall\mu_{p(q)}}(v_p^{(i)})]$.

例如,在事务 T_1 中项 $\{a.M\}$ 的加权外部效用为项 a 的外部效用 20 与 $\{a.M\}$ 对应数量 6 的乘积,即 $U_w(a.M)=20\times6=120$.

批次	事务标识符 (TID)	事务
W_1 B_1	T_1	$(a,2)$ $(c,7)$ $(h,4)$
	T_2	$(a,3)$ $(c,8)$
W_1 B_2	T_3	$(a,2)$ $(b,1)$ $(c,7)$ $(g,7)$ $(h,4)$
W_2 B_2	T_4	$(b,2)$ $(c,9)$ $(g,8)$ $(h,5)$
W_2 B_3	T_5	$(a,2)$ $(d,5)$ $(e,1)$ $(f,1)$
	T_6	$(a,3)$ $(b,2)$ $(c,3)$ $(d,1)$ $(e,2)$

图 2　数据流示例数据

Fig. 2　Sample of data stream

表 1　外部效用表

Table 1　Profit table

项	a	b	c	d	e	f	g	h
外部效用值	20	15	70	54	11	100	75	47

定义 3[10]　模糊项的效用. 模糊项的效用定义为模糊项的模糊度与项的外部加权效用的乘积,即 $U_{f(c_{p(q)})}=f_{p(q)}^{(i)}\times U_w(c_{p(q)})$.

例如,在事务 T_1 中 $\{a.M\}$ 的效用为 $\{a.M\}$ 的模糊度 0.2 与其加权外部效用 120 的乘积,即 $U_{f(a.M)}=f_{a.M}^{(1)}\times U_w(a.M)=0.2\times120=24$.

定义 4　模糊项集在窗口中的效用. 窗口 W_i 中模糊项集 X 的效用值为该项集在事务中的效用和,记为 $U_{F(x)}$,表示为 $U_{F(x)}=\sum_{i\in I}\sum_{c_{p(q)}\in x} f_{p(q)}^{(i)}\times U_w(c_{p(q)})$,其中 $c_{p(q)}\in X, i\in TID(X)$.

例如,在窗口 W_1 中项集 $(a.M,c.H)$ 的模糊效用为 $(a.M,c.H)$ 在事务 T_1、T_2、T_3 的效用和,可以表示为 $U_{F(a.M,c.H)}=(f_{a.M}^1+f_{a.M}^2+f_{a.M}^3)\times U_W(a.M)+(f_{c.H}^1+f_{c.H}^2+f_{c.H}^3)\times U_W(c.H)=(0.2+0.4+0.2)\times120+(0.2+0.4+0.2)\times770=96+616=712$.

定义 5　窗口 W_i 的效用. 给定窗口 W_i 的效用为窗口中所有不同模糊项的效用之和,记为 $TFU(W_i)$,表示为 $TFU(W_i)=\sum_{c_p\in I}\sum_{q=1}^{q_p} U_{F(c_{p(q)})}$.

例如,窗口 W_1 的模糊效用为 $TFU(W_1)=U_F(a.L)+U_F(a.M)+U_F(a.H)+\cdots+U_F(h.L)+U_F(h.M)+U_F(h.H)=44+96+0+\cdots+28.2+394.8+0=4091$.

定义 6[10]　高效用模糊项集. 若模糊项集 X 的效用大于用户定义的最小效用阈值,则认为模糊项集 X 是 $HUFI$,即 $U_{F(x)}>\xi\times TFU(W_i)$,其中 $U_{F(x)}$ 是窗口 W_i 中项集 X 的效用,$TFU(W_i)$ 是当前窗口的效用,ξ 是用户定义的最小效用阈值.

定义 7[10]　事务最大效用. 事务 $T_i\in D$ 的事务最大效用,记为 $TMU(T_i)$,是事务 T_i 中不同项的模糊度中效用最大的项的和,即 $TMU(T_i)=\sum_{v_p^{(i)}\in T_i}\max_{1\leqslant q\leqslant q_p}(\mu_{p(q)}(v_p^{(i)})\times U_w(c_{p(q)}))$.

例如,事务 T_2 的事务最大效用为项 a 所在模糊度的效用最大值 48 与 c 所在模糊度的效用最大值 308 的效用之和. 即 $TMU(T_2)=\max(U_F(a.L),U_F(a.M))+\max(U_F(c.M),U_F(c.H))=48+308=356$.

定义 8[10]　模糊项集的事务加权最大效用. 模糊项集 X 的事务加权最大效用,记为 $TFI(X)$,是所有包含模糊项集 X 的事务的最大效用的和,即 $TFI(X)=\sum_{i\in TID(X)} TMU(T_i)$.

例如,项集 $(a.M,c.H)$ 的事务最大效用为 $TFI(a.M,c.H)=TMU(T_1)+TMU(T_2)+TMU(T_3)=529.5+356+904.2=1789.7$.

3　算法介绍

本节提出了模糊效用列表 FUL 结构存储模糊项的效用信息,并基于此结构提出了 FHUI_DS 与 FHUI_Native 算法来挖掘 HUFI. 本节先介绍了 FUL 结构,然后介绍了提出的算法.

3.1　模糊效用列表

定义 9　模糊项集 X 之后的模糊项. 给定一个模糊项集 X 和一个事务 T_i,在事务 T_i 中项集 X 之后且

与模糊项集 X 中的项不同的模糊项 c_p 定义为 T_i/X.

定义 10 模糊剩余效用. 模糊项集 X 的剩余效用定义为 frutil(X,T_i),是根据≺对事务排序后,项集 X 之后的模糊项 c_p 的模糊效用和,表示为 frutil(X,T_i)= $\sum_{x\in T_i/X} U_{FI}(x,T_i)$.

FUL 的构造需要扫描两次窗口中的事务. 在第一次扫描中,将项的数量信息通过隶属度函数转化为模糊域,然后计算项目的 TFI,并根据 TFI 对模糊项进行排序. 在第二次扫描期间,创建每个模糊项的 FUL. FUL 存储当前窗口中模糊项的效用信息,该结构存储项的批次号以及一个包含 TID、fiutil、frutil 元组的条目. TID 存储包含项集的事务标识符,fiutil 是模糊项集的效用,frutil 是模糊项集的剩余模糊效用.

当新一批事务到达时,新事务中模糊项的效用信息会存储到 FUL 中. 当到来的批次数量大于窗口大小时将移除最旧的一批事务,插入新一批的事务. FUL 数据结构的优点是可以快速执行批次的插入和删除. 例如,在图 2 中第一个滑动窗口 W_1 的模糊项(a.M)的 FUL 如图 3 所示,当新的批次加入后,更新后项(a.M)的 FUL 分别如图 4 所示.

a.M	TID	fiutil	frutil
B_1	T_1	24	678
	T_2	48	560
B_2	T_3	24	1218

(a) W_1

a.M	TID	fiutil	frutil
B_2	T_3	24	1218
B_3	T_5	24	0
	T_6	48	180

(b) W_2

图 3 项 a.M 的 FUL

Fig. 3 FUL of a.M

(a.M),(c.H)	TID	fiutil	frutil
B_1	T_1	178	0
	T_2	356	0
B_2	T_3	178	0

图 4 项集{(a.M),(c.H)}的 FUL

Fig. 4 FUL of {(a.M),(c.H)}

由两个模糊项组成的模糊项集的 FUL 是通过两个模糊项的 FUL 中的 TID 相交来创建的. 例如,分别从(a.M)和(c.H)的 FUL 构造项集{(a.M),(c.H)}的 FUL. (a.M)和(c.H)的 FUL 具有共同的批次 B_1 和 B_2. 在 B_1 中(a.M)和(c.H)在 T_1 和 T_2 中同时出现,在 B_2 中(a.M)和(c.H)在 T_3 同时出现. 项集{(a.M),(c.H)}的 FUL 如图 4 所示.

3.2 FHUI_DS 算法

在本节中提出了 FHUI_DS 算法挖掘数据流 HUFI. FHUI_DS 算法将输入 5 个参数:(1)数据流事务数据,(2)利润数据集,(3)窗口大小,(4)批次大小,(5)用户定义的最小效用阈值算法. 算法执行结束将返回 HUFI.

对于给定的数据流,读取数据流中的信息,将批次中的事务加入到滑动窗口当中,当加入的批次数目大于等于窗口大小时,处理当前窗口中的数据. 当有新的批次加入时,删除窗口中最旧批次的信息,即旧批次中所包含事务的模糊项的效用信息,然后将新批次中模糊项的效用信息更新到窗口中,并更新批次中的 TFI 及效用列表中的信息(算法 1 的①-⑥行). 对于当前窗口中的事务,先进行第一次数据扫描,通过隶属度函数将项的数量信息转化为模糊域,并计算单个模糊项的 TFI;然后进行第二次数据扫描,构建单个模糊项的 FUL 与估计模糊效用共现结构(estimated fuzzy utility co-occurrence structure,EFUCS),该结构改进了 FHM[19] 中的 EUCS(estimated utility co-occurrence structure,EUCS)结构(算法 1 的⑦-⑩行). 然后调用挖掘算法,挖掘窗口中的 HUFI(算法 1 的⑪行).

算法 1 FHUI_DS

输入:A data stream DS;profit database *DP*;the size of window *winSize*;the number of batches *number_of_batches*;a user-defined threshold *minutil*;

输出:the set of fuzzy high-utility itemsets in current sliding window;

①While there is stream of tranctions do

② Read the transaction from data stream *DS* and add to *batchTransactions* according to *batchsize*;

③ ++currentBatch;

④ If currentBatch>*winSize* then

⑤ Remove *oldBatchBumber*

⑥ Add *batchTransactions*.

⑦For T_r in *batchTransaction*

⑧ Scan the transactions in current *batchTransaction* to compute the fuzzy domian and *TFI* of items;

⑨ $I^* \rightarrow$ Each fitem such that in batchTransaction
⑩ Second scan the transactions in current batchTransaction to create FUL and EFUCS.
⑪HUFIMining(Ø,I^* FULs,EFUCS,*minutil*)

挖掘过程将输入 5 个参数:(1)前缀模糊项集 P,(2)用于扩展模糊项集 P 的集合 ExtensionsOfP,(3)模糊效用列表 FULs,(4)估计模糊效用共现结构 EFUCS,(5)用户定义的最小效用阈值 minutil.

算法 2 显示了 HUFIMining 的伪代码. FULs 中的每个模糊项集 Px 的模糊效用大于等于 minutil,则 Px 及其相关的扩展为 HUFI(算法 2 的①-④行). 若 Px 的模糊效用小于 minutil,进一步判断 Px 的模糊效用与剩余模糊效用和是否大于等于 minutil,若大于等于则可以对 Px 进一步扩展,否则对其剪枝. 使用 EFUCS 消除低效用扩展 Pxy,对高效用的扩展 Pxy 构造其 FUL,项集 Px 的所有 1-扩展的 FUL 进行递归处理(算法 2 的⑤-⑭行). 然后进行递归挖掘操作,直至挖掘出当前窗口中的全部 HUFI(算法 2 的⑭行).

算法 2 HUFIMining

输入:P:an itemset;*ExtensionsOfP*:a set of extensions of P;*FULs*;*EFUCS*;a user-defined threshold *minutil*;
输出:the set of fuzzy high-utility itemsets in current sliding window;
①For each itemset $Px \in$ *ExtensionsOfP* do
② If FUL(Px).sumFiutil $\geqslant$ minutil
③ Output the extension associated with X.
④ End
⑤ If FUL(Px).sumFutil+FUL(Px).sumFrutil $\geqslant$ minutil then
⑥ *ExtensionsOfPx* $\leftarrow$Ø;
⑦ For each itemset $Py \in$ ExtensionsOfP such that $y \succ x$ do
⑧ If($\exists (x,y,c) \in$ *EFUCS* such that $c \geqslant$ *minutil*) then
⑨ $Pxy \leftarrow Px \cup Py$;
⑩ $Pxy \leftarrow$Construct(P,Px,Py);
⑪ *ExtensionOfPx* $\leftarrow$ *ExtensionOfPx* $\cup$ Pxy;
⑫ End if
⑬ End for
⑭ HUFIMining(Px,*ExtensionsOfP*,FULs,EFUUCS,minutil);
⑮ End if
⑯End for

3.3 FHUI_Native 算法

FHUI_DS 算法对窗口中重复批次的事务中项的 FULs 进行了重用,使用了上一个窗口构建的项的模糊效用信息. 但是当新的批次插入后,模糊项的剩余效用信息及 TFI 会产生一定的变化,造成 TFI 排序后,FUL 中的模糊剩余效用值不准确,使得挖掘算法性能降低. 为了提高挖掘算法的性能,本文提出了 FHUI_Native 算法,当新批次插入时,删除旧批次,插入新批次,不重用重复批次的 FULs,每个窗口的 FULs 都从头开始构建,FHUI_Native 算法的其它过程与 FHUI_DS 相同. FHUI_DS 相比 FHUI_Native 会产生更多的连接操作,降低了算法性能. 在第 4 部分对两个算法的性能进行了对比.

3.4 复杂度分析

算法的时间复杂度主要包括 FUL 的构造与更新和挖掘过程. 令 m 表示窗口中不同项的数量,n 表示当前窗口中的事务数. 算法先进行第一次数据集扫描,将项的数量信息转化为模糊域所需时间 $O(mn)$,转化后每个项最多都包含了 3 个模糊域用于组合产生 HUFI,同时计算项的 TFI 需要时间为 $O(3mn)$,然后进行第二次数据集扫描创建 FUL 与 EFUCS,需要根据 TFI 对项进行排序,需要时间为 $O(3mn\log(3mn))$;创建 *EFUCS* 需要检查事务中的共现所需时间为 $O((3m)^2n)$. 挖掘过程包括比较两个$(k-1)$项集与一个$(k-2)$项集的 TID 产生 k 项集,其时间复杂度为 $O(3n+3n)=O(6n)$. 令经过剪枝策略之后生成的项集数量为 l,构造效用列表的总时间复杂度为 $O(6ln)$,则总的时间复杂度为 $O(2\times3mn+3mn\log(3mn)+9m^2n+6ln)=O(9m^2n+3mn\ \log(3mn)+6ln)$. FHUI_Native 与 FHUI_DS 的区别在于当有新的批次加入到窗口中,

FHUI_DS 会根据批次号删除旧事务的效用信息,构建新事务的 FUL,令新加入的事务数为 n',FHUI_DS 需要扫描处理的事务数为 n'而 FHUI_Native 需要处理的事务数量为 n. 此外,由于排序结果的差异,经过剪枝策略之后生成的项集数量 l 也会不同,即 n 与 n'的不同和 l 的变化使两算法的时间复杂度产生了差异.

算法的内存消耗主要用于 FUL 的构造. FUL 用于存储项、项的 TID、模糊效用与模糊剩余效用. 每个列表最多有 n 个条目,构造 FUL 的空间复杂度为 $O(4\times3m\times n)$,在挖掘过程中生成的 k 项集的数量为 l 构造的 FUL 的空间复杂度为 $O(4\times l\times n)$,总的空间复杂度为 $O(4\times3m\times n+4\times l\times n)=O(12mn+4ln)$,最高阶为 mn,因此最终结果为 $O(12mn)$.

4 实验分析

在本节中,通过实验评估所提出算法的性能. 在评估实验中所有算法均用 Java 语言实现. 使用真实数据集与合成数据集来评估算法的性能. 所有数据集均从 SPMF[20] 中获取,这些数据集的特征如表 2 所示,表 2 描述了数据集的事务数、不同的项数和平均事务长度. 首先,将提出算法挖掘的 HUFI 与 FHUQI-Miner[9] 挖掘的 HUQI 数量进行对比;然后,由于 FHUI_Native 与 FHUI_DS 是首次用于数据流环境挖掘 HUFI 的算法,因此评估提出算法的可靠方法是将其与静态数据中现有的算法进行比较,对比算法为 FHTFUP[21],本文算法 FHUI_Native 以单窗口批处理方式运行;最后,对数据流环境的数据在不同最小效用阈值、不同窗口大小和不同批次大小来比较算法的性能.

4.1 HUFI 与 HUQI 数量对比

首先在图 2 所示的示例数据集和 Foodmart 数据集上进行了实验,以评估不同最小效用阈值下 HUFI 与 HUQI 的数量差异,其中将示例数据集的窗口大小设置为 1,批次大小设置为 6. 表 3 显示了随着最小效用阈值的增大,产生的 HUFI 与 HUQI 的数量都逐渐减少,且 HUFI 的数量要少于 HUQI,原因在于 HUFI 将项集的数量信息转化为了模糊域,减少了单个项不同数量组合操作中信息冗余的项集.

表 2 实验数据集

Table 2 Deatil of Details of the datasets

数据集	事务数	项数	事务平均长度	类型
Foodmart	4 141	1 559	4.42	稀疏
Retail	88 162	16 470	10.30	稀疏
BMS2	77 512	3 340	4.62	稀疏
Mushroom	8 416	119	23	密集
T20I6D100k	99 922	893	19.90	密集

表 3 不同最小效用阈值下 HUFI 与 HUQI 的数量

Table 3 Numbers of HUFIs and HUQIs under various minutil

数据集	最小效用阈值	HUFI	HUQI
示例数据集	1 000	35	45
	1 100	25	36
	1 200	16	22
Foodmart	35 000	1 989	2 055
	40 000	132	614
	45 000	64	101

4.2 与 FHTFUP 对比

本节在 Foodmart 与 Mushroom 数据集上比较 FHUI_Native 与 FHTFUP 的性能. 将 FHUI_Native 设置为单窗口与静态算法 FHTFUP 比较,在单窗口上 FHUI_Native 与 FHUI_DS 性能基本一致,因此仅比较 FHUI_Native 与 FHTFUP 的差异.

从图 5 和图 6 中可以看出随着 minutil 的增加,两算法的运行时间逐渐减小,消耗的内存也在减少,原因在于随着最小效用阈值的增加,产生的满足条件的项集数量减少. 此外,FHUI_Native 算法在运行时间与内存

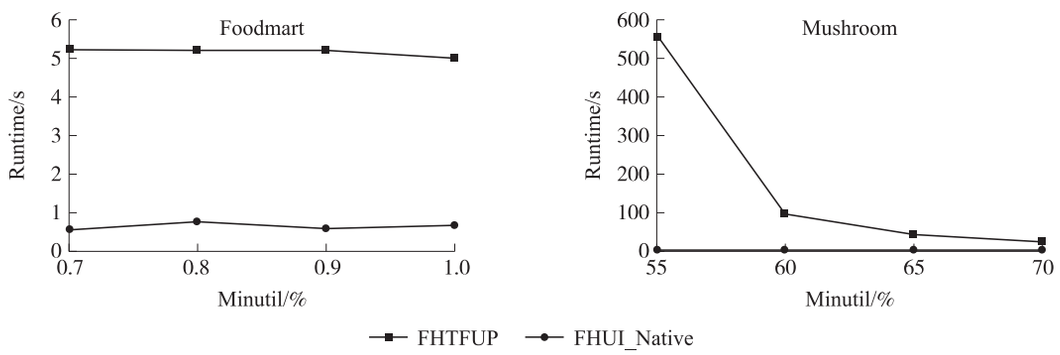

图 5 与 FHTFUP 的运行时间比较

Fig. 5 Comparison of running time with FHTFUP

消耗方面性能均优于 FHTFUP。原因在于 FHTFUP 采用树结构存储项的模糊效用信息,其在第一阶段产生符合条件的候选项集,然后在第二阶段再次扫描数据集判断候选项集中的项集是否为 HTFUP,而第一阶段产生了较多没有希望的项集. FHUI_Native 则使用 FUL 存储项的效用信息,无需产生候选项集,提高了算法性能.

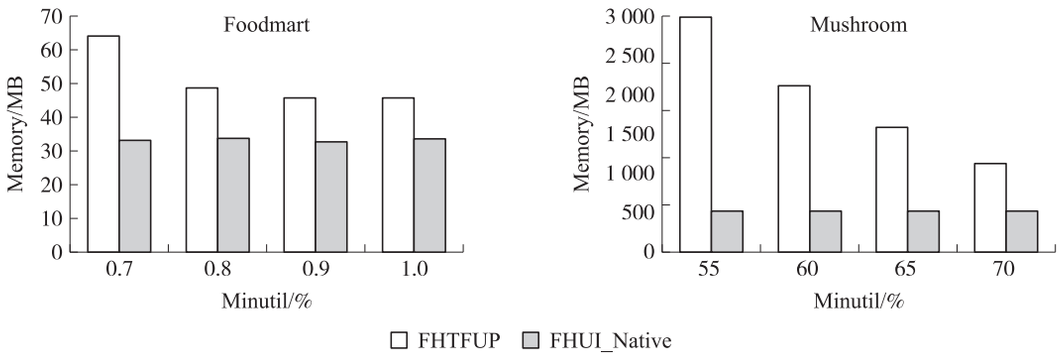

图 6　与 FHTFUP 的内存消耗比较

Fig. 6　Comparison of memory consumption with FHTFUP

4.3　不同最小效用阈值

本节在不同数据集上比较算法在不同最小效用阈值下的性能,固定窗口大小为 3,Retail、BMS2 T20I6D100k 数据集的批次数目固定为 10 000.

从图 7 和图 8 可以看出随着 minutil 的增加,算法的运行时间逐渐降低. 因为算法的运行时间与产生的中间项集的数量有关,当阈值逐渐增大,满足条件的中间项集的数量减少,算法的运行时间也会降低. FHUI_Native 算法性能优于 FHUI_DS 是因为前者在每个窗口上从头构建窗口中项的 FULs,使得每个窗口拥有较优的 TFI 排序. 事务按 TFI 升序排序时,基于效用列表的算法性能表现更好,产生的中间项集较少,因此 FHUI_Native 算法的性能优于 FHUI_DS. 此外,FHUI_Native 算法产生的中间项集少于 FHUI_DS,因此前者的内存消耗要少于后者.

4.4　不同窗口大小

滑动窗口模型通过窗口大小与批次大小来控制窗口中的事务数量,为了分析它们的影响,本节固定最小效用阈值与批次数目来分析窗口大小对算法性能的影响.

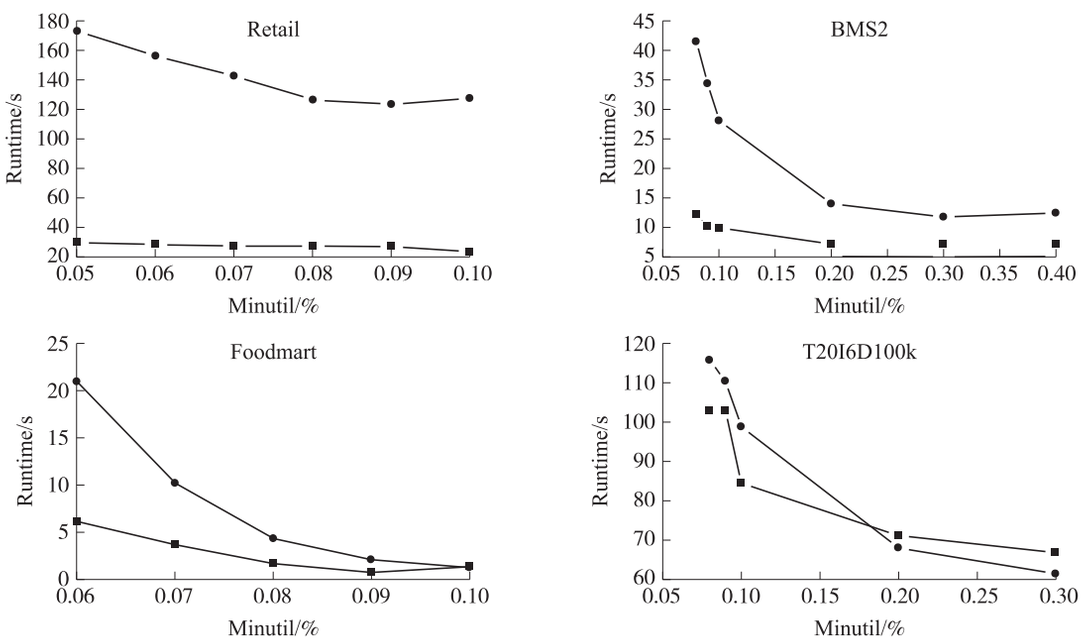

图 7　不同最小效用阈值下的运行时间比较

Fig. 7　Running time comparison under varying utility threshold

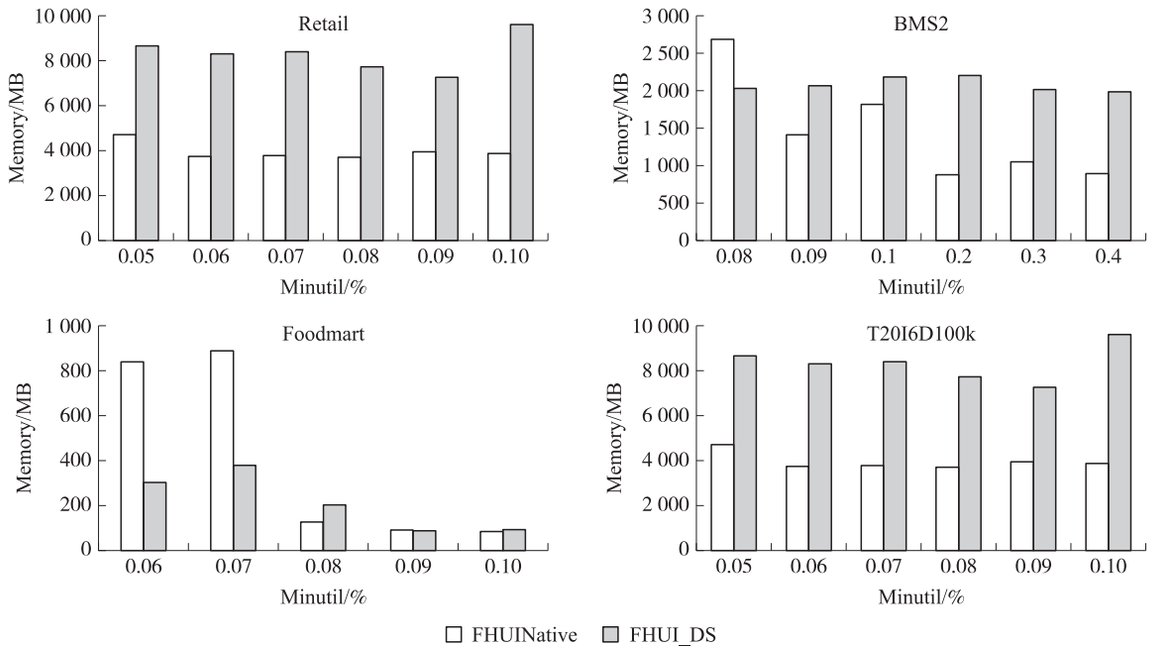

图 8　不同最小效用阈值下的内存消耗比较

Fig. 8　Memory consume comparison under varying utility threshold

从图 9 和图 10 可以看出随着窗口数量的增加,算法的运行时间与算法的内存消耗都逐渐增加. 原因在于窗口大小增加,窗口中的事务数也随之增加,需要处理的事务数增多,运行时间与内存消耗也会增加. 稀疏数据集中项集的 FULs 的可复用率较低,新增的候选中间项集较少;而密集数据集平均事务长,项目多,将项的数量转化为模糊域之后,需要进行的连接操作增加,产生的候选项集较多. 因此算法在稀疏数据集上的性能要优于密集数据集. Foodmart 数据集的内存使用在 FHUI_DS 上少于 FHUI_Native 的原因在于 Foodmart 数据集事务数较少,事务的 TFI 变化较大. FHUI_DS 重用了之前窗口构建的重复批次事务中的 FULs,当新批次插入后 TFI 发生了变化,使得重复批次事务中 FULs 的剩余效用可能不符合新的 TFI 排序结果,对部分可扩展的项集进行了剪枝.

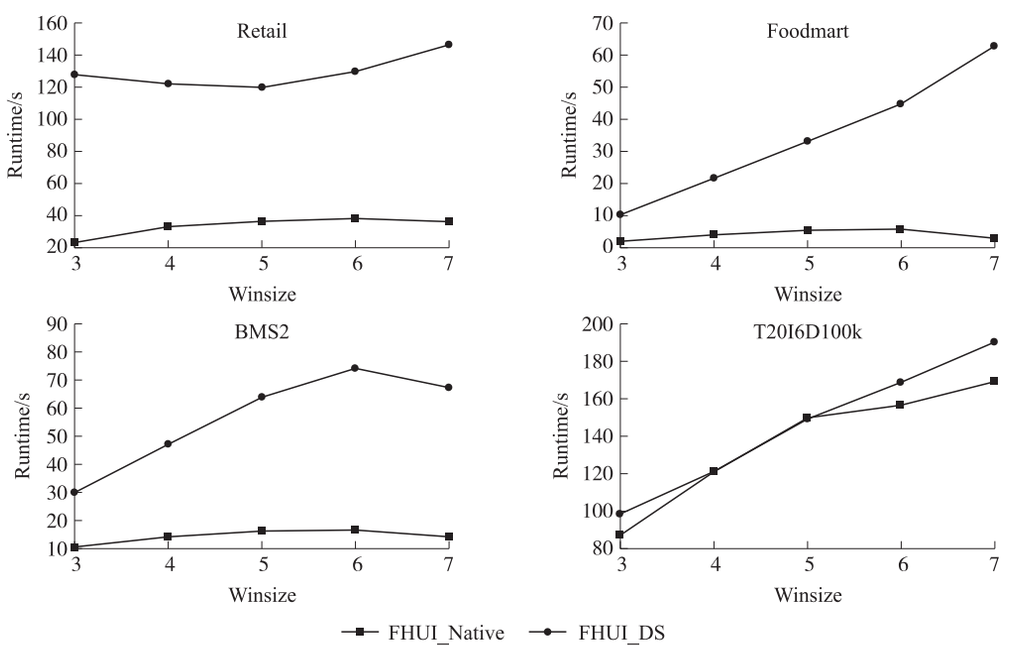

图 9　不同窗口大小下的运行时间比较

Fig. 9　Running time comparison under varying window sizes

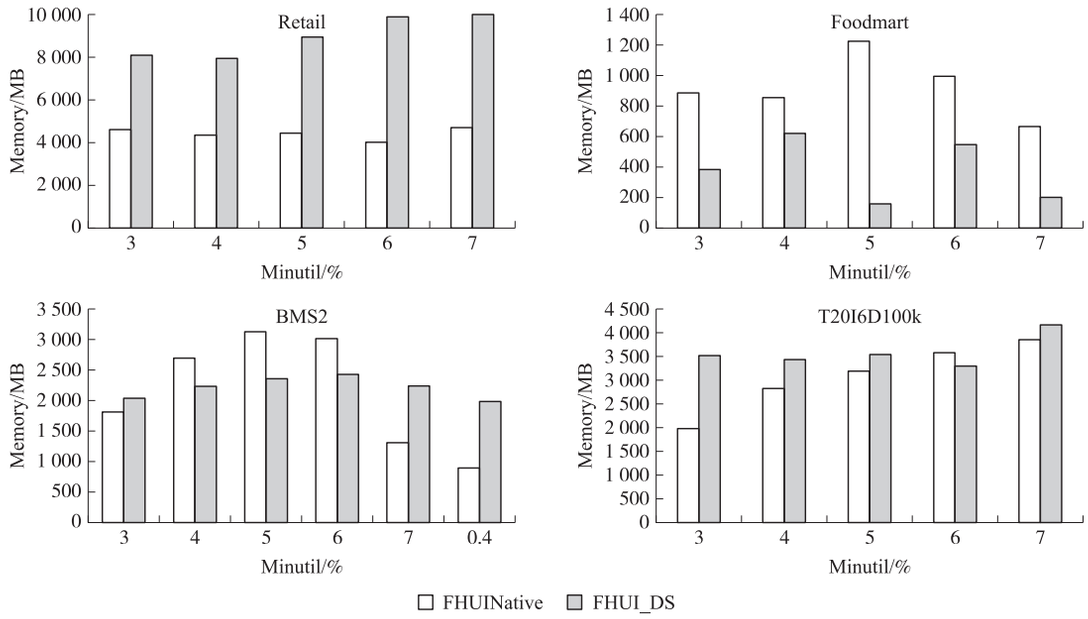

图 10　不同窗口大小下的内存消耗比较

Fig. 10　Memory consume comparison under varying window sizes

4.5　不同批次大小

图 11 和图 12 显示了不同批次大小对算法性能的影响. 当批次中的事务较小时,当前窗口中的事务数较少,随着批次数目的增加,窗口中需要处理的批次数增加,当有新批次加入时,算法需要处理的事务增加,因此运行时间也增加. 从图 11 可以看出算法在不同数据集上的运行时间总体为增加趋势. 在 BMS 数据集上 FHUI_Native 算法的性能较好,并且当批次数目增加到一定的值,运行时间存在降低风险. 原因是稀疏数据集增大批次数目后,其增加的中间项集数较少,批次数目增大减少了滑动窗口的创建次数和挖掘次数. T20I6D100k 由于是密集数据集,会产生大量较长的 HUFI,产生的中间项集较多,运行时间较长.

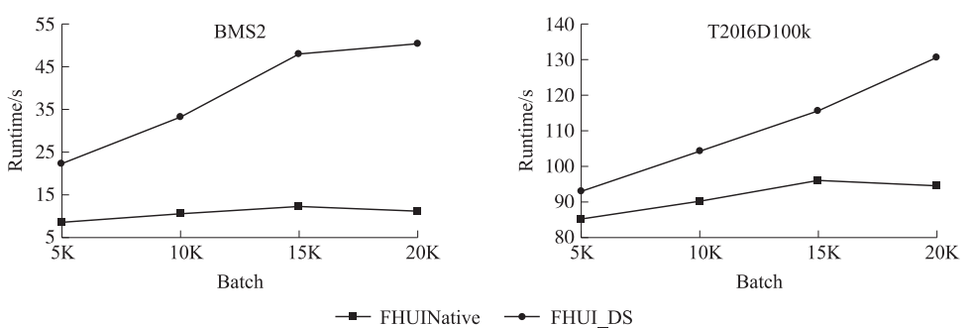

图 11　不同批次大小下的运行时间比较

Fig. 11　Running time comparison under varying batch sizes

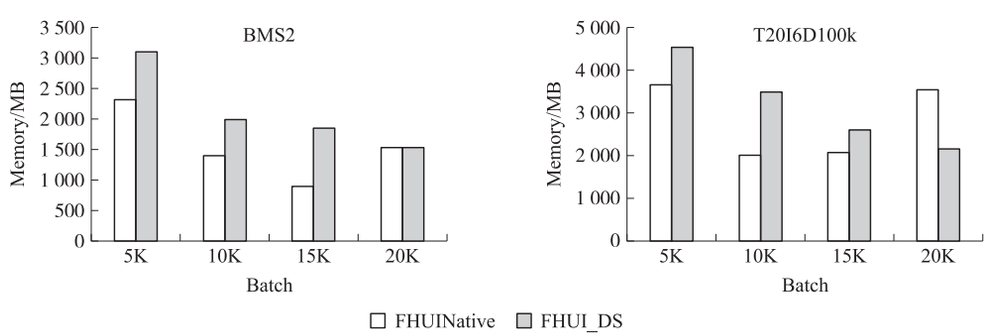

图 12　不同批次大小下的内存消耗比较

Fig. 12　Memory consume comparison under varying batch sizes

5 结论

本文提出了 FUL 存储模糊项的效用信息,能够有效的对窗口中的批次进行删除与增加操作. 并基于 FUL 提出了基于滑动窗口的数据流 HUFI 挖掘算法 FHUI_DS 与 FHUI_Native,两算法都能够有效挖掘数据流 HUFI,为决策者提供了项集的数量信息. 大量的实验表明 FHUI_Native 算法优于 FHUI_DS,且两个算法在稀疏数据集上的性能优于密集数据集. HUFI 挖掘是比 HUI 挖掘更复杂的问题,处理长事务会产生比 HUI 更多的中间项集,如何高效的处理长事务,提出更优的剪枝策略是该领域未来的研究方向之一.

[参考文献]

[1] LIU Y,LIAO W,CHOUDHARY A. A two-phase algorithm for fast discovery of high utility itemsets[C]//Proceedings of the 9th Pacific-Asia Conf on Advances in Knowledge Discovery and Data Mining. Berlin:Springer,2005:689-695.

[2] DAM T L,KENLI L I,FOURNIER-VIGER P,et al. CLS-Miner:efficient and effective closed high-utility itemset mining[J]. Frontiers of computer science,2019,13(2):357-381.

[3] SETHI K K,DHARAVATH R. A fast high average-utility itemset mining with efficient tighter upper bounds and novel list structure[J]. Journal of supercomputing,2020,76(12):10288-10318.

[4] 杨皓,段磊,胡斌,等. 带间隔约束 Top-k 对比序列模式挖掘[J]. 软件学报,2015,26(11):2994-3009.

[5] 王晓璇,王丽珍,陈红梅,等. 基于特征效用参与率的空间高效用 co-location 模式挖掘方法[J]. 计算机学报,2019,42(8):1721-1738.

[6] 吉根林,王敏. 时空轨迹聚集模式挖掘研究进展[J]. 南京师大学报(自然科学版),2015,38(4):1-7.

[7] NOUIOUA M,FOURNIER-VIGER P,WU C W,et al. FHUQI-Miner:fast high utility quantitative itemset mining[J]. Applied intelligence,2021,51(10):6785-6809.

[8] WANG C M,CHEN S H,HUANG Y F. A fuzzy approach for mining high utility quantitative itemsets[C]//In 2009 IEEE International Conference on Fuzzy Systems. IEEE,2009:1909-1913.

[9] LAN G C,HONG T P,LIN Y H,et al. Fuzzy utility mining with upper-bound measure[J]. Applied soft computing,2015,30:767-777.

[10] LAN G C,HONG T P,LIN Y H,et al. Fast discovery of high fuzzy utility itemsets[C]//2014 IEEE International Conference on Systems,Man,and Cybernetics(SMC). IEEE,2014:2764-2767.

[11] HONG T P,LIN C Y,HUANG W M. One-phase temporal fuzzy utility mining[C]//In 2020 IEEE International Conference on Fuzzy Systems(FUZZ-IEEE). IEEE,2020:1-5.

[12] WU J M T,LIN J C W,FOURNIER-VIGER P,et al. A ga-based framework for mining high fuzzy utility itemsets[C]//2019 IEEE International Conference on Big Data(Big Data). IEEE,2019:2708-2715.

[13] YANG F,MU N,LIAO X,et al. EA-HUFIM:Optimization for fuzzy-based high-utility itemsets mining[J]. International journal of fuzzy systems,2021,23:1652-1668.

[14] 宋威,刘明渊,李晋宏. 基于事务型滑动窗口的数据流中高效用项集挖掘算法[J]. 南京大学学报(自然科学),2014,50(4):494-504.

[15] TSAI P S M. Mining high utility itemsets in data streams based on the weighted sliding window model[J]. International journal of data mining and knowledge management process,2014,4(2):13-28.

[16] JAYSAWAL B P,HUANG J W. SOHUPDS:a single-pass one-phase algorithm for mining high utility patterns over a data stream[C]//Proceedings of the 35th Annual ACM Symposium on Applied Computing. New York:ACM,2020:490-497.

[17] DAWAR S,SHARMA V,GOYAL V. Mining top-k high-utility itemsets from a data stream under sliding window model[J]. Applied intelligence,2017,47(4):1240-1255.

[18] 程浩东,韩萌,张妮,等. 基于滑动窗口模型的数据流闭合高效用项集挖掘[J]. 计算机研究与发展,2021,58(11):2500-2514.

[19] FOURNIER-VIGER P,WU C W,ZIDA S,et al. FHM:faster high-utility itemset mining using estimated utility co-occurrence pruning[C]//Processdings of 21st International Symposium on Methodologies for Intelligent Systems. Roskilde,Denmark:Lecture,2014:83-92.

[20] FOURNIER-VIGER P,GOMARIZ A,GUENICHE T,et al. SPMF:a Java open-source pattern mining library[J]. Journal of machine learning research,2014,15(1):3389-3393.

[21] HONG T P,LIN C Y,HUANG W M,et al. Using tree structure to mine high temporal fuzzy utility itemsets[J]. IEEE access,2020,8:153692-153706.

[责任编辑:杜忆忱]