Лекция 11. Извличане на асоциативни правила

Асоциативните правила изразяват асоциации или корелационни връзки между голямо множество от данни за различни артикули (стоки). В момента, когато постоянно се събират и се съхраняват огромни масиви от данни, много фирми се заинтересуваха от извличане на асоциативни правила от техните бази данни. Откриването на интересни асоциативни правила измежду огромния брой записи за търговски транзакции може да помогне в различни бизнес процеси по приемането на решения, като, например, конструиране на каталози, индивидуален маркетинг и т.н.

11.1. Основни понятия

Нека $I = \{i_1,...,i_m\}$ е множество от артикули (закупени стоки), а D е множество от транзакции (сделки) в базата данни, като всяка една транзакция T е някое множество от артикули $T \subseteq I$. Всяка транзакция е асоциирана с определен идентификатор, наречен TID. Нека A е някое множество от артикули. Ще казваме, че транзакцията T съдържа A тогава и само тогава, когато $A \subseteq T$.

Асоциативното правило е импликацията от вида на $A \Rightarrow B$, където $A \subset I, B \subset I$ и $A \cap B = \varnothing$.

Асоциативното правило $A \Rightarrow B$ се съдържа в множеството от транзакции D със степен на *поддръжка (support) s*, ако s е процент на тези транзакции от D, които съдържат $A \cup B$. Това е аналогично на вероятността $P(A \cup B)$. Правилото $A \Rightarrow B$ има степен на *доверие (confidence) c* в множеството от транзакции *D,* ако c е процент на тези транзакции в D, съдържащи A, които също така съдържат и B. Това е аналогично на условната вероятност $P(B \mid A)$. С други думи:

$$\begin{aligned} s(A \Rightarrow B) &= P(A \cup B) \\ c(A \Rightarrow B) &= P(B \mid A) \end{aligned} \qquad (10.1)$$

Правилата, които удовлетворяват както на минималния праг за поддръжка (*min_sup*), така и на минималния праг за доверие (*min_conf*), се наричат *убедителни (strong)*. По съглашение, ще пишем стойностите на поддръжка и доверие в проценти, а не като дробни числа.

Едно множество от артикулите се нарича *артикулно множество (itemset)*. Артикулното множество, съдържащо k артикула, се *нарича k-артикулно множество (k-itemset)*. Например множеството *{компютър, финансов_софтуер}* е 2-артикулното множество. *Честотата на срещане* на някое артикулно множество е броят на транзакции, които съдържат това множество. Тази величина още се

нарича просто *честота* или *поддържаща сума (support count)*. Едно артикулно множество удовлетворява степента на *минимална поддръжка*, ако честота на неговото срещане е по-голяма или равна на произведението на минималния праг за поддръжка *min_sup* върху общия брой на транзакции в *D*. Броят на транзакции, необходим за едно артикулно множество да удовлетворява степента на минимална поддръжка, се нарича *минимална поддържаща сума (minimum support count)*.

Едно артикулно множество се нарича *често срещано (frequent itemset)*, ако то удовлетворява степента на минималната поддръжка. Множеството от често срещани k-артикулни множества се означава с L_k.

Извличането на асоциативни правила е двустъпков процес:

1. *Намиране на всички често срещани артикулни множества*: по определение, всяко от тези множества ще се среща най-малко толкова често, колко е определено от предварително зададена минималната поддържаща сума.
2. *Генериране на убедителни асоциативни правила от намерените често срещани артикулни множества*: по определение, тези правила трябва да удовлетворяват степента на минималната поддръжка и минималното доверие.

По желание, могат да бъдат използвани и други мерки за "интересност" (убедителност) на асоциативни правила – за тях ще говорим по-късно. Втората стъпка е по-лесната от двете. Общото поведение на асоциативните правила се определя от първата стъпка.

11.2. Класификация на асоциативните правила

Асоциативните правила могат да бъдат класифицирани по различни начини съгласно следните критерии:

- *На базата на типове на стойностите, обработвани в правилото*.
  - Ако едно правило описва асоциациите между наличие или отсъствие на някои артикули, то това е *булево асоциативно правило.* Булевите асоциативни правила се получават при анализа на потребителската кошница, например:

$$компютър \Rightarrow финансов\_софтуер \ [поддръжка = 2\%, доверие = 60\%] \quad (11.2)$$

  - Ако едно правило описва асоциациите между числови артикули или атрибути, то се нарича *количествено асоциативно правило*. В тези правила количествени стойности на артикулите или атрибути са разбити на интервали. Следващото правило е пример на количествено асоциативно правило (X е променлива, означаваща потребителя):

$$възраст(X, "30...39") \wedge доход(X, "1000...5000") \Rightarrow купува(X, домашно\_кино) \quad (11.3)$$

- *На базата на размерностите на данни, включени в правилото.*

- Ако артикулите или атрибути в едно асоциативно правило се отнасят само за една размерност, то правилото се нарича *едномерно асоциативно правило*. Например, правилото (11.2) може да се препише по следния начин:
  $купува(X, компютър) \Rightarrow купува(X, финансов\_софтуер)$
  Това е едномерно асоциативно правило, тъй като то описва само една размерност – *купува*. Ако правилото описва две или повече размерности, то се нарича *многомерно асоциативно правило*. Пример на такова правило е (11.3), тъй като описва три размерности: *възраст, доход* и *купува*.

- *На базата на нивата на абстракцията, включени в множеството от правила*.
  Някои методи за извличане на асоциативни правила могат да намират правила на различни нива на абстракцията. Например, намереното множество от правила съдържа следните правила:
  $$\begin{array}{l} възраст(X, "30...39") \Rightarrow купува(X, преносим\_компютър) \\ възраст(X, "30...39") \Rightarrow купува(X, компютър) \end{array} \quad (11.4)$$
  В тези правила артикулите, които се купуват, са на различни нива на абстракцията – *"компютър"* е по-общото понятие от *"преносим компютър"*. Подобни правила се наричат *многонивови (multilevel) асоциативни правила*. Ако всички правила от някое множество описват артикули или атрибути от едно и също ниво на абстракция, то такива правила се наричат *еднонивови асоциативни правила*.

- *На базата на различни добавки (разширения) към базовия процес на извличане на асоциативни правила*.
  Процесът на извличане на асоциативни правила може да бъде разширен с корелационния анализ, в който може да бъде установено наличието или отсъствието на корелиращи артикули. Той също така може да бъде допълнен с извличането на *максимални често срещани шаблони (макси-шаблони – maxpatterns). Макси-шаблонът* е такъв често срещнат шаблон (артикулното множество), за който не съществуват никакви супер-шаблони (над-множества), които са също често срещани. Подобно разширение може съществено да намали броя на често срещани артикулни множества, генерирани в процеса на извличане на асоциативни правила.

11.3. Извличане на едномерни булеви асоциативни правила от транзакционни бази данни

11.3.1. Алгоритъм Apriori: намиране на често срещани артикулни множества чрез генериране на кандидати

Apriori (R. Agraval & R. Strikant 1996) е базовият алгоритъм за намиране на често срещани артикулни множества за булеви асоциативни правила. Името на алгоритъма идва от факта, че той използва определени априорни знания за свойства на често срещани артикулни множества. Алгоритмът използва итеративен

подход, известен като *понивово търсене (level-wise search)*, при който k-артикулните множества са използват за проучване на (k+1)-артикулните множества. Отначало се намира множество от често срещани 1-артикулните множества – L_1. След това L_1 се използва за намиране на множеството от често срещани 2-артикулните множества L_2 , което на свой ред се използва за намирането на L_3, и т.н., докато вече не могат да бъдат намерени често срещани артикулните множества по-големи от k. Намирането на всяко L_k изисква едно пълно сканиране на базата данни.

За да подобри ефективността на понивовата генерация на често срещани артикулни множества, се използва едно важно свойство, наречено *свойство на Apriori*, намаляващо пространството на търсене. За да бъде едно артикулно множество често срещано, всички негови непразни подмножества трябва също така да бъдат често срещани. Това свойство се базира на следните наблюдения. По определение, ако някое артикулно множество I не удовлетворява минималния праг за доверие *min_sup,* то I не е често срещано, т.е. $P(I) < min\_sup$. Ако към I се добави някой артикул A, то полученото артикулно множество $I \cup A$ не може да се среща по-често от I. По тази причина $I \cup A$ също не е често срещано артикулно множество, т.е. $P(I \cup A) < min\_sup$.

Това свойство принадлежи към една специална категория на свойствата, наречени *анти-монотонни* в смисъла, че *ако някое множество не може да мине определен тест, то всички неговите надмножества също така няма да могат да минат този тест.* Анти-монотонността означава монотонността на свойството в контекста на неудовлетворяване на определен тест.

За да разберем, как това свойството на Apriori се използва в алгоритъма, да разгледаме как L_{k-1} се използва за намиране на L_k. Това става чрез двустъпков процес, състоящ от действията *съединение (join)* и *съкращаване (prune)*.

1. *Съединение*. За намирането на L_k се генерира множество от *кандидати* – k-артикулни множества, получени чрез съединение на множеството L_{k-1} само със себе си. Това множество от кандидати се означава с C_k. Нека l_1 и l_2 са някои артикулни множества в L_{k-1}. Означение $l_i[j]$ сочи на j-я артикул в l_i (например $l_1[k-2]$ означава вторият от края артикул в l_1). По съглашение, в Apriori се предполага, че всички артикули в една транзакция или артикулното множество са сортирани в лексикографския ред. Изпълнява се съединението $L_{k-1} \rhd\lhd L_{k-1}$, като членовете на L_{k-1} са съединими, ако техните първи (k-2) артикула са едни и същи. Т.е. членовете l_1 и l_2 са съединими, ако $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge ...(l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$. Условието $l_1[k-1] < l_2[k-1]$ просто осигурява отсъствието на дубликати при генериране на кандидатите. Артикулното множество, полученото чрез съединение на l_1 и l_2 , е $l_1[1]\ l_1[2]\ ...\ l_1[k-1]\ l_2[k-1]$.

2. *Съкращаване*. C_k е надмножеството на L_k, т.е. неговите членове могат и да не бъдат често срещани, обаче в C_k са включени всички често срещани k-

артикулни множества. L_k може да се определи чрез едно сканиране на базата данни, при което се определя броя на срещания на всеки кандидат от C_k (т.е. всички кандидати, чийто брой на срещания не е по-малък от минималната поддържаща сума, са по определение често срещани и, следователно, принадлежат към L_k). Обаче, C_k може да бъде много голямо и следователно, този подход може да изисква огромен брой изчисления. За да намали размера на C_k, се използва свойството на Apriori: всяко (k-1)-артикулно множество, което не е често срещано, не може да бъде подмножество на някое често срещано k-артикулно множество. Следователно, ако някое (k-1)-артикулно подмножество от едно кандидат-k-артикулно множество не е член на L_{k-1} , то това кандидат-множество не може да бъде често срещано и, следователно, трябва да бъде изтрито от C_k. Тестването на подмножества може да бъде направено бързо чрез поддържане на hash-дървото от всички често срещани артикулни множества.

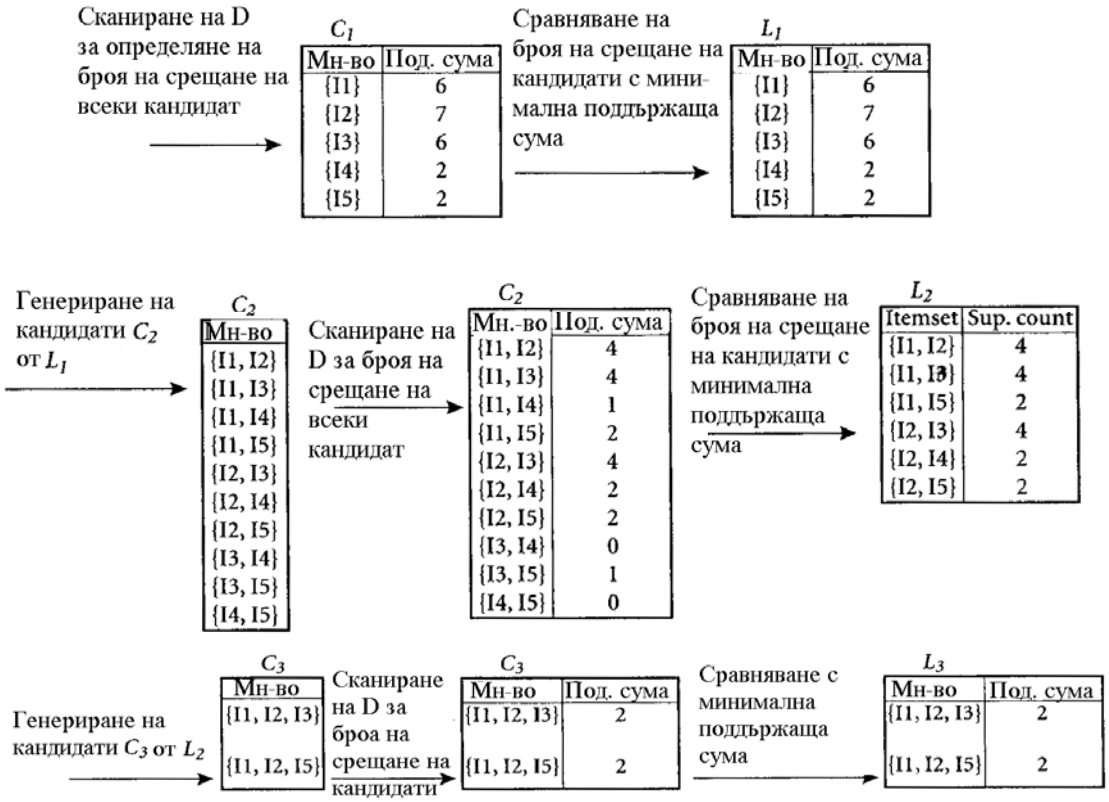
TID	Списък от артик.
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Фиг. 11.1. Образец на база данни от транзакции

Пример 11.1. Да разгледаме, как работи Apriori на един конкретен пример. Нека, базата данни от транзакции е представена на фиг. 11.1. Тя съдържа 9 транзакции, т.е. |D| = 9.

1. При първата итерация на алгоритъма всеки артикул е член на множеството от кандидати на 1-артикулните множества C_1. Алгоритмът просто сканира по ред всички транзакции, за да преброи броя на срещания в тях на всеки артикул.
2. Да предположим, че исканата минимална подържаща сума е 2 (т.е., че *min_sup = 2/9 = 22%*). Множеството от често срещани 1-артикулни множества L_1 съдържа тези кандидат-1-артикулни множества от C_1, които удовлетворяват степента на минималната поддръжка (виж. първия ред на фиг. 11.2).
3. За намиране на множеството от често срещани 2-артикулни множества L_2, алгоритмът използва съединението $L_1 \triangleright\triangleleft L_1$, за да генерира множеството от

кандидати - 2-артикулни множества C_2. C_2 се състои от $C_{|L_1|}^2$ 2-артикулни множества[1].



Фиг. 11.2. Генериране на кандидати – артикулни множества и често срещани артикулни множества при минималната поддържаща сума равна на 2.

4. Транзакциите в D се сканират и се изчислява поддържащата сума за всяко артикулно множество – кандидат в C_2, както е показано в средния ред на Фиг. 11.2.
5. Сега се определя множеството от често срещани 2-артикулни множества L_2. То се състои от кандидати – 2-артикулни множества от C_2, които имат минималната поддръжка.
6. Генерацията на множеството от кандидати на 3-артикулните множества е представена детайлно на Фиг. 11.3. Получаваме:

   $$C_3 = L_2 \triangleright\triangleleft L_2 = \{\{I1, I2, I3\}, \{I1, I2, I5\}, \{I1, I3, I5\}, \{I2, I3, I4\}, \{I2, I3, I5\}, \{I2, I4, I5\}\}$$

   Базирайки се на свойството на Apriori, че всички подмножества на едно често срещано артикулно множество също трябва да бъдат често срещани, можем да определим, че последните четири кандидата нямат вероятността да бъдат често срещани. Следователно, можем да ги изтрием от C_3, спестявайки си усилия по изчисляване на техните поддържащи суми при

[1] Напомням, че $C_n^m = \frac{n!}{m!(n-m)!}, \quad m < n$

последващото сканиране на D за определяне на L_3. Обърнете внимание, че когато имаме един кандидат k-артикулно множество, трябва само да проверим, дали неговите (k-1)-подмножества са често срещани, тъй като алгоритъм Apriori използва понивовата стратегия за търсене.

7. Транзакциите в D отново се сканират за да определим L_3, състоящо от тези кандидати 3-артикулни множества от C_3, които имат минималната поддръжка (виж. последен ред на Фиг. 11.2).
8. Алгоритмът използва $L_3 \rhd\lhd L_3$, за да генерира множество от 4-артикулни кандидати C_4. Макар, че резултат от съединение е {{I1,I2,I3,I5}}, това артикулно множество се съкращава, тъй като неговото подмножество {I1,I2,I5} не е често срещано. Следователно $C_4 = \varnothing$ и алгоритмът завършва работата си, намирайки всички често срещани артикулни множества.

---

1. Съединение.

$C_3 = L_2 \rhd\lhd L_2 = \{\{I1,I2\},\{I1,I3\},\{I1,I5\},\{I2,I3\},\{I2,I4\},\{I2,I5\}\} \rhd\lhd$

$\{\{I1,I2\},\{I1,I3\},\{I1,I5\},\{I2,I3\},\{I2,I4\},\{I2,I5\}\} = \{\{I1,I2,I3\},\{I1,I2,I5\},$

$\{I1,I3,I5\},\{I2,I3,I4\},\{I2,I3,I5\},\{I2,I4,I5\}\}.$

2. *Съкращение,* използвайки свойството на Apriori, че всички непразни подмножества на едно често срещано артикулно множество трябва също да бъдат често срещани. Дали някои от кандидати имат подмножества, които не са често срещани?
   - 2-артикулните подмножества на {I1,I2,I3} са {I1,I2}, {I1,I3} и {I2,I3}. Те всички са членове на L_2. Следователно {I1,I2,I3} се запазва в C_3.
   - 2-артикулните подмножества на {I1,I2,I5} са {I1,I2}, {I1,I5} и {I2,I5}. Те всички са членове на L_2. Следователно {I1,I2,I5} се запазва в C_3.
   - 2-артикулните подмножества на {I1,I3,I5} са {I1,I3}, {I1,I5} и {I3,I5}. {I3,I5} не е член на L_2. Следователно {I1,I3,I5} се изтрива от C_3.
   - 2-артикулните подмножества на {I2,I3,I4} са {I2,I3}, {I2,I4} и {I3,I4}. {I3,I4} не е член на L_2. Следователно {I2,I3,I4} се изтрива от C_3.
   - 2-артикулните подмножества на {I2,I3,I5} са {I2,I3}, {I2,I5} и {I3,I5}. {I3,I5} не е член на L_2. Следователно {I2,I3,I5} се изтрива от C_3.
   - 2-артикулните подмножества на {I2,I4,I5} са {I2,I4}, {I2,I5} и {I4,I5}. {I4,I5} не е член на L_2. Следователно {I1,I2,I3} се изтрива от C_3.
3. И така, след съкращаването получаваме C_3 = *{{I1,I2,I3}, {I1,I2,I5}}*.

Фиг. 11.3. Генериране на множество от кандидати 3-артикулни множества C_3 от L_2 с използване на свойството на Apriori.

На Фиг. 11.4 е показан псевдо-код на алгоритъма Apriori и свързаните с него процедури. Стъпка 1 на алгоритъма намира често срещани 1-артикулни множества

L_1. На стъпките 2-10 L_{k-1} се използва за генериране на кандидати C_k, за да намери L_k.

***Алгоритъм Apriori**. Намира често срещани артикулни множества чрез използване на итеративния подход за понивовото търсене на базата на генериране на кандидати.*

***Вход:** База данни D, съдържаща транзакции;*
праг за минимална поддръжка min_sup.

***Изход:** L – често срещани артикулни множества в D.*

Метод:

(1) L_1 = find_frequent_1-itemsets(D);
(2) **за** (k =2; $L_{k-1} \neq \varnothing; k++$)
(3) { C_k = apriori_gen(L_{k-1}, min_sup);
(4) **за всяка** транзакция $t \in D$ // *Сканиране на D за броя на срещане*
(5) { C_t = подмножество(C_k, t); // *Взимане на тези подмножества от t, които са кандидати*
(6) **за всеки** кандидат $c \in C_t$
(7) c.count++;
(8) }
(9) $L_k = \{c \in C_k \mid c.count \geq \text{min\_sup}\}$;
(10) }
(11) **върни** $L = \bigcup_k L_k$;

***Процедура** apriori_gen(L_{k-1}: често срещани (k-1)-артикулни множества; min_sup)*

(1) **за всяко** артикулно множество $l_1 \in L_{k-1}$
(2) **за всяко** артикулно множество $l_2 \in L_{k-1}$
(3) **ако** $(l_1[1] = l_2[1]) \wedge ... (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-2] < l_2[k-2])$ **то**
(4) { $c = l_1 \rhd\lhd l_2$; // *стъпката на съединение: генериране на кандидати*
(5) **ако** has_infrequent_subset(c, L_{k-1}) **то**
(6) изтрий c; // *стъпката на съкращаване: изтриване на безполезни кандидати*
(7) **иначе** добави *c* към C_k;
(8) }
(9) **върни** C_k;

***Процедура** has_frequent_subset(c: кандидат k-артикулно мн-во; L_{k-1});*
// използване на априорните знания

(1) **за всяко** (k-1)-подмножество *s* на *c*
(2) **ако** $s \notin L_{k-1}$ **то**
(3) **върни** ИСТИНА;
(4) **върни** ЛЪЖА

Фиг. 11.4. Алгоритъм Apriori за откриване на често срещани артикулни множества за извличане на асоциативни правила

Процедурата apriori_gen генерира кандидати, а след това използва свойството на Apriori, за да елиминира тези от тях, които имат поне едно подмножество, което не е често срещано (Стъпка 3). След като всички кандидати са генерирани, прави се сканиране на базата данни (Стъпка 4). За всяка транзакция се използва функцията *подмножество*, за да намери всички подмножества на транзакцията, които са кандидати (Стъпка 5), като за всеки от тези кандидати се пази съответния брояч за броя на тяхното срещане (Стъпките 6 и 7). На края, всички кандидати, които удовлетворяват прага за минималната поддръжка, формират множеството от често срещани артикулни множества L. За генериране на асоциативни правила от тези често срещани артикулни множества трябва да бъде извикана специална процедура, която е описана в следващия раздел.

Процедурата apriori_gen изпълнява два вида действия, наречени съединяване и съкращаване. При съединяването L_{k-1} се съединява с L_{k-1}, за да генерира потенциални кандидати (Стъпки 1-4). При съкращаването (Стъпки 5-7) се използва свойството на Apriori за изтриване на кандидатите, които имат някое подмножество, което не е често срещано. Проверката за не често срещани подмножества е показана в процедурата has_infriguent_subset.

11.3.2. Генериране на асоциативни правила от често срещани артикулни множества

След като бяха намерени всички често срещани артикулни множества от транзакциите от базата данни D, е сравнително лесно от тях да бъдат генерирани убедителни асоциативни правила (напомням, че *убедителните* са тези правила, които удовлетворяват едновременно прага за минималната поддръжка и прага за минималното доверие). За целта може да се използва следното уравнение за степента на доверие, изразяващо условната вероятност в термините на поддържащата сума на артикулното множество:

$$доверие(A \Rightarrow B) = P(B \mid A) = \frac{поддържаща\_сума(A \cup B)}{поддържаща\_сума(A)} \quad (11.5)$$

където $поддържаща\_сума(A \cup B)$ е броят на транзакции, съдържащи артикулното множество $A \cup B$, а *поддържаща_сума(A)* е броят на транзакции, съдържащи артикулното множество A. Използвайки това уравнение, асоциативните правила могат да бъдат генерирани по следния начин:

- За всяко често срещано артикулно множество l, да се генерират всички непразни подмножества на l.
- За всяко непразно подмножество s на l, да се създаде правилото:
  $s \Rightarrow (l - s)$ *ако* $\frac{поддържаща\_сума(l)}{поддържаща\_сума(s)} \geq min\_conf$
  където *min_conf* е прагът за минималното доверие.

Тъй като правилата се генерират автоматично от често срещани артикулни множества, всяко едно от тях автоматично удовлетворява прага за минималната поддръжка. Често срещани артикулните множества могат да се пазят в hash-таблици заедно със свои поддържащи суми, осигурявайки по този начин бърз достъп до тях.

Пример 11.2. Да продължим с примера на базата данни от транзакции, показана но Фиг. 11.1. Да предположим, че данните съдържат често срещано артикулното множество $l = \{I1,I2,I5\}$. Какви асоциативни правила могат да се генерират от това множество? Непразните подмножества на l са {I1,I2}, {I1,I5}, {I2,I5}, {I1}, {I2} и {I5}. Получените от тях асоциативни правила са показани по-долу заедно със своята степен на доверие:

$$I1 \wedge I2 \Rightarrow I5,\ \textit{доверие} = 2/4 = 50\%$$
$$I1 \wedge I5 \Rightarrow I2,\ \textit{доверие} = 2/2 = 100\%$$
$$I2 \wedge I5 \Rightarrow I1,\ \textit{доверие} = 2/2 = 100\%$$
$$I1 \Rightarrow I2 \wedge I5,\ \textit{доверие} = 2/6 = 33\%$$
$$I2 \Rightarrow I1 \wedge I5,\ \textit{доверие} = 2/7 = 29\%$$
$$I5 \Rightarrow I1 \wedge I2,\ \textit{доверие} = 2/2 = 100\%$$

Ако установим прага на минималното доверие на, речем, 70%, то като изход ще се останат само второто, третото и последното правила, тъй като само те ще бъдат убедителни.

11.3.3. Подобряване на ефективността на Apriori

За подобряване на ефективността на Apriori бяха предложени различни варианти на оригиналния алгоритъм. Някои от тези вариации са описани по-долу.

Създаване на хеш-таблица H_2 с използване на ф-я $h(x, y) = ((\textbf{номер на x}) \times 10 + (\textbf{номер на y}))\ mod\ 7$ →

H_2

адрес на клетка	0	1	2	3	4	5	6
брояч на клетка	2	2	4	2	2	4	4
съдържание на клетка	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Фиг. 11.5. Хеш-таблица H_2 за кандидати 2-артикулни множества. Таблицата е била генерирана при сканиране на базата от транзакции, представена на Фиг. 11.1, докато се определяше множество L_1 от множеството C_1. Ако прагът за минималната поддръжка е установен на 3, то артикулните множества, попаднали в клетки на таблицата с адреси 0, 1, 3 и 4 не могат да са често срещани, следователно те не трябва да бъдат включени в C_2. Например, h{I1,I2} = (1 x 10 + 2) mod 7 = 12 mod 7 = 5

- *Хеширане на броя на срещане на артикулните множества:* За намаляване на размера на кандидатите – k-артикулни множества C_k, $k > 1$, могат да бъдат използвани техники за хеширане (hashing). Например, за да бъдат генерирани 1-

артикулните множества L_1 от кандидати – 1-артикулни множества C_1, при сканиране на всяка от транзакции в база данни могат да бъдат генерирани всички 2-артикулни множества, те да бъдат хеширани (отобразени) в различни клетки от структурата на хеш-таблицата и да бъдат увеличени броячи на съответните клетки (виж фиг. 11.5). Тези 2-артикулни множества, чиито съответни броячи на клетки от хеш-таблицата са по-малки от прага за поддръжка, не могат да бъдат често срещани и, следователно, трябва да бъдат изтрити от множеството на кандидати. Такава хеш-базирана техника може съществено да намали броя на проверявани кандидати k-артикулни множества (особено, когато $k = 2$).

- *Намаляване на транзакции (намаляване на броя на транзакции, сканирани при бъдещи операции):* Една транзакция, която не съдържа никакви често срещани k-артикулни множества не може да съдържа никакви често срещани (k+1)-артикулни множества. По тази причина такава транзакция може да бъде маркирана или премахната от по-нататъшното разглеждане при следващите сканирания на база данни за търсене на j-артикулни множества при $j > k$.
- *Разделяне на данни на непресичащи се раздели (клъстери) за намиране на кандидати артикулни множества:* Извличането на често срещани артикулни множества само чрез два сканирания на базата данни може да се направи чрез прилагане на техниката за разделяне (клъстеризация) на данни (Фиг. 11.6). Процесът се състои от две фази. На първата фаза алгоритмът разделя транзакциите в D на n непресичащи се раздели. Ако прагът за минималната поддръжка за транзакциите в D е *min_sup*, то минималната поддържаща сума за артикулните множества в един раздел е равна на *min_sup x брой на транзакции в този раздел.* За всеки раздел се намират всички артикулни множества често срещани в този раздел. Те се маркират като *локално често срещани артикулни множества*. Процедурата използва една специална структура данни, която за всяко артикулно множество записва идентификатори на транзакции (TID), съдържащи артикули от множеството. Това позволява намирането на всички локални често срещани k-артикулни множества за $k = 1, 2, \ldots$ само за едно сканиране на базата данни.

Едно локално често срещано артикулно множество може и да не бъде често срещано по отношение на цялата база данни D. Обаче, *всяко артикулно множество, което е потенциално често срещано по отношение на D, трябва да участва като локално често срещано най-малко в един от разделите*. Следователно, всички локално често срещани артикулни множества са кандидати артикулни множества по отношение на D. Колекцията от често срещани артикулни множества от всички раздели формира *глобалното множество от кандидати артикулни множества* по отношение на D. На втората фаза се прави второто сканиране на D, по време на което се оценява действителната поддръжка на всеки кандидат с цел да бъдат определени глобално често срещани артикулни множества. Размерът на раздели и техният брой се установяват по такъв начин, че всеки раздел да може да се побере в основната памет и, следователно, да бъде прочетен само веднъж на всяка от фазите.

Фиг. 11.6. Извличане на често срещани артикулни множества чрез разделяне

- *Използване на извадката (анализ само на подмножеството от налични данни):* Основната идея на подхода е да бъде направена някоя случайна извадка S от наличните данни D, а след това да се прави търсене на често срещани артикулни множества не в D, а в S. По този начин се търси определена загуба на точността за сметка на ефективността. Размерът на извадката S е такъв, че търсенето на често срещани артикулни множества в S може да се направи само в основната памет и като цяло за транзакциите в S да е необходимо само едно сканиране. Тъй като често срещани артикулните множества се търсят в S вместо в D, е възможно да бъдат изпуснати някои глобално често срещани артикулни множества. За да намалим вероятност на такова събитие се използва по-малък праг за поддръжка при намирането на често срещани артикулни множества локални за S (означавани с L^S). Останалата част от базата данни се използва за изчисляване на действителните честоти на срещане на всяко артикулно множество от L^S. Специалният механизъм се ползва за определяне, дали всички глобално често срещани артикулни множества са включени в L^S. Ако L^S действително съдържа всички често срещани артикулни множества от D, то е необходимо само едно сканиране на D. В противен случай, за да бъдат намерени липсващите множества е необходимо второто сканиране на D. Подходът с използване на извадката се прилага в случаите, когато ефективността е от основното значение – това са скъпи от изчислителната гледна точка приложения, които трябва да бъдат изпълнявани много често.

11.3.4. Намиране на често срещани артикулни множества без генериране на кандидати

И така, в много от случаи използваният от Apriori метод за пораждане и тестване на кандидати значително намалява размер на множествата от кандидати и води до доброто поведение на алгоритъма. Обаче, той има следните недостатъци:

- *Необходимостта от генериране на огромния брой множества от кандидати*. Например, ако съществуват 10^4 често срещани 1-артикулни множества, то Apriori ще трябва да генерира повече от 10^7 2-артикулни множества от

кандидати и да съхранява и тества техните честоти на срещане. Освен това, за да намери един често срещан шаблон с размер 100, като например $\{a_1, \ldots, a_{100}\}$, алгоритъмът трябва да генерира повече от $2^{100} \approx 10^{30}$ кандидати.

- *Необходимостта от повторното сканиране на базата данни и проверката на голямо множество от кандидати чрез сравняване с образец (pattern matching)*. Това особено важи за случаи на търсене на дълги шаблони.

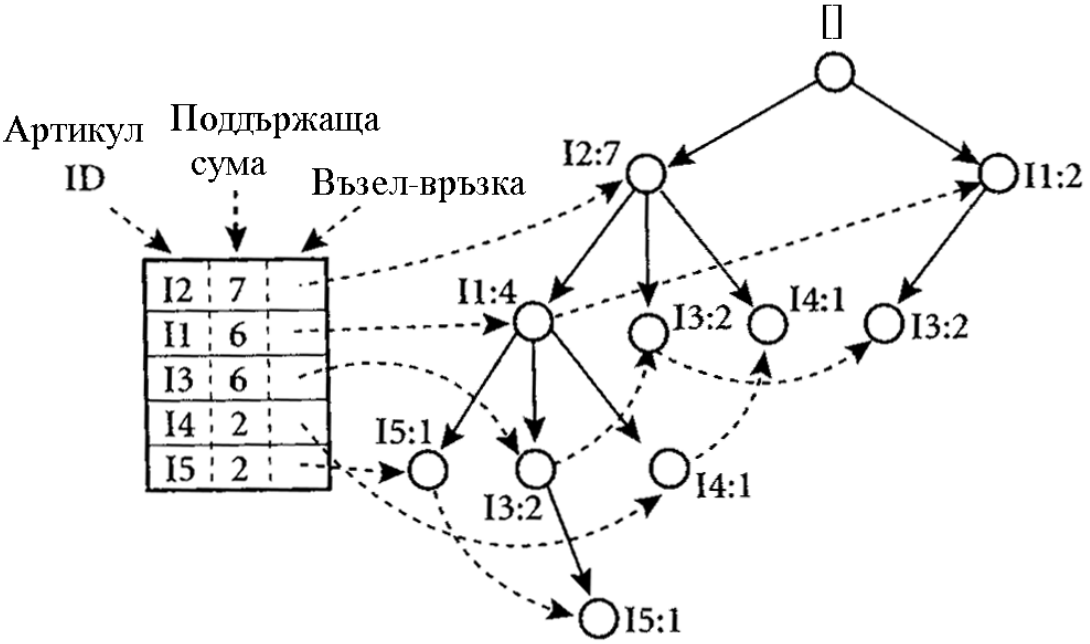
Съществуват подходи, опитващи да избегнат описаните недостатъци на Apriori чрез премахване на необходимостта от генериране на кандидати. Един от такива подходи се нарича *нарастване на често срещани шаблони (frequent-pattern growth)* или просто *FP-growth*. Той използва стратегията *"разделя и владей" (divide-and-conquer)*: първоначално базата данни, представяща често срещани артикулни множества, се компресира в специално *дърво на често срещани шаблони (FP-дърво)*, но се запазва асоциираната с артикулните множества информация. След това тази компресирана база данни се разделя на множество от *условни бази данни,* всяка от които е асоциирана с един често срещан артикул, и анализът на всяка от тези бази се извършва по-отделно. Нека да разгледаме един пример.

Пример 11.3. Отново ще използваме базата данни от транзакции, представени на Фиг. 11.1, но от гледната точка на подхода за нарастване на често срещани шаблони.

Първото сканиране на базата данни е същото като при Apriori – извежда се множеството от често срещани артикули (1-артикулни множества) и техните поддържащи суми (честоти на срещане). Нека прагът за минималната поддръжка е 2. Множеството от често срещани артикули се сортира в реда на намаляване на поддържащите суми. Полученото множество или *списък,* означен с *L*. И така получаваме: L = [I2: 7, I1: 6, I3: 6, I4: 2, I5:2].

FP-дървото се конструира по следния начин. Първо, създаваме корена на дървото, маркиран с празен списък ([]). Сканираме базата данни *D* за втори път. Артикулите във всяка транзакция се обработват по реда от *L* (т.е. сортирани по реда на намаляване на поддържащата сума), като за всяка транзакция се създава по един клон. Например, сканирането на първата транзакция "T100: I1, I2, I5", която съдържа три артикула (I2, I1, I5) по реда на L, води до създаване на първият клон на дървото с трите възли: <(I2: 1), (I1: 1), (I5: 1)>, където I2 е съединен като директен наследник на корена, I1 е съединен с I2, а I5 – с I1. Втората транзакция, T200, съдържа артикулите I2 и I4 по реда на L, което води до създаване на вторият клон в дървото, в който I2 е съединен с корена, а I4 – с I2. Обаче, този клон има общ *префикс* <I2> със съществуващият път за T100. По тази причина, вместо създаване на новия клон ще увеличим брояча във възела I2 с единица и ще създадем нов възел (I4: 1), който е съединен като пряк наследник с (I2: 2). В общия случай, при разглеждането на клона, който трябва да бъде добавен към дървото за нова транзакция, броячите на всеки възел с общ префикс се увеличават с единица, а възлите за артикулите, следващи зад префикса, се създават и присъединяват към съществуващите възли в дървото по съответния начин.

За облекчаване на предвижването по дървото се построява заглавната таблица на артикули (item header table), така че всеки артикул указва на своите участия в дървото чрез една верижка от *възли-връзки*. Дървото, получено след сканирането на всички транзакции, е показано на Фиг. 11.7 заедно с асоциираните възли-връзки. По такъв начин задачата за намиране на често срещани шаблони в базата данни се трансформира в тази за търсенето в FP-дърво.



Фиг. 11.7. Образец на FP-дърво, което регистрира компресираната информация за често срещани шаблони.

Търсенето в FP-дърво се прави по следния начин. Започва се от всеки често срещан шаблон с дължина 1 (като един начален *суфикс шаблон*), конструира се *условната база данни на шаблона* ("под-базата" данни, съдържаща множеството от *префиксни пътищата* от FP-дървото, срещани заедно със суфикс шаблона), а след това се конструира неговото (*условно*) FP-дърво и се прави рекурсивно претърсване на това дърво. Нарастването на шаблони се постига чрез конкатенацията на суфикс шаблона с често срещани шаблони, генерирани от условното FP-дърво.

Търсенето в FP-дърво е обобщено в таблица от Фиг. 11.8. Нека вместо първият артикул от L отначало да разгледаме I5, който е последният артикул в L, като причината за това ще стане ясна при обяснението на процеса на претърсване на FP-дървото. I5 присъства в два клона на FP-дърво, показано на Фиг. 11.7 (като участията на I5 могат лесно да се намерят чрез следването по верижката възли-връзки). Пътищата, формирани от тези клонове са <(I2 I1 I5: 1)> и <(I2 I1 I3 I5: 1)>. Следователно, разглеждайки I5 като суфикс, неговите съответни префиксни пътища са <(I2 I1: 1)> и <(I2 I1 I3: 1)>, които формират условната база данни на шаблона. Построеното от нея условното FP-дърво ще съдържа само един път <I2: 2, I1: 2>; I3 не е включено в дървото, тъй като неговата поддържаща сума е 1, която е

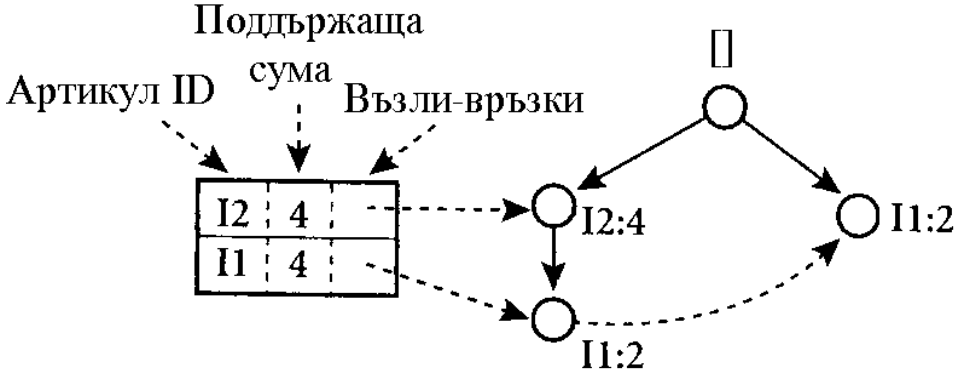
по-малка от минималния праг за поддръжка, който е 2. Този единствен път генерира всички комбинации от често срещани шаблони: I2 I5: 2, I1 I5: 2, I2 I1 I5: 2.

Артикул	Условна база на шаблона	Условно FP-дърво	Често срещани шаблони
I5	{(I2 I1: 1), (I2 I1 I3: 1)}	⟨I2: 2, I1: 2⟩	I2 I5: 2, I1 I5: 2, I2 I1 I5: 2
I4	{(I2 I1: 1), I2: 1)}	⟨I2: 2⟩	I2 I4: 2
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	I2 I3: 4, I1, I3: 2, I2 I1 I3: 2
I1	{(I2: 4)}	⟨I2: 4⟩	I2 I1: 4

Фиг. 11.8. Търсене в FP-дървото чрез създаване на условни бази на (под-) шаблони

За I4 неговите два префиксни пътя формират условната база на шаблона {(I2 I1: 1), (I2: 1)}, която генерира условното FP-дърво с един единствен възел <I2: 2> и извежда само един често срещан шаблон I2 I4: 2. Обърнете внимание, че макар I5 следва за I4 в първия клон, няма никаква необходимост да включваме I5 в проведения анализ, тъй като всички често срещани шаблони, включващи I5, вече бяха проанализирани при изследването на I5. Точно това е причината, защо сме започнали обработката от края на L, а не от неговото начало.

Аналогично, за I3 условната база на шаблона е {(I2 I1:2), (I2: 2), (I1: 2)}. Неговото условно FP-дърво има два клона <I2: 4, I1: 2> и <I1: 2>, както е показано на Фиг. 11.9. То генерира следното множество от шаблони: {I2 I3: 4, I1 I3: 2, I2 I1 I3: 2}. И на края, за I1 условната база на шаблона е {(I2: 4)}, чието FP-дърво съдържа само един възел <I2: 4>, който генерира един често срещан шаблон I2 I1: 4. Описаният процес на търсене в FP-дърво е обобщен във вид на алгоритъм на Фиг. 11.10.



Фиг. 11.9. Условното FP-дърво, асоциирано с условния възел I3

Методът FP-growth трансформира проблема за намиране на дълги често срещани шаблони в рекурсивно търсене на по-къси шаблони и последващо конкатениране на суфикса. Той използва най-рядко срещания из между често срещани артикули като суфикс, предлагайки добра селективност. Методът значително намалява цената на търсене.

Алгоритъм FP-growth. *Намира често срещани шаблони чрез използване на FP-дърво и нарастване на фрагменти на шаблоните.*
Вход: *База данни с транзакции D; праг за минимална поддръжка min_sup.*
Изход: *Пълното множество от често срещани шаблони*
Метод:

1. FP-дървото се конструира чрез следните стъпки:

(a) Еднократно се сканира базата данни *D*. Събира се множеството от често срещани артикули *F* и техните поддържащи суми. *F* се сортира по реда на намаляване на поддръжката и се пази като списък от често срещани артикули *L*.

(b) Създава се коренът на FP-дърво и се маркира с *[]*. За всяка транзакция *Trans* в *D* се прави следното:

Избират се и се сортират често срещани артикули в *Trans* съгласно реда на *L*. Нека списъкът от сортираните често срещани артикули в *Trans* е *[p|P]*, където *p* е елемент, а *P* е списък. Извиква се процедура ***insert_tree****([p|P], T),* която върши следното:

Ако *T* има наследник *N*, така че *N.име_на_артикула = p.име_на_артикула* ***То*** броячът на *N* се увеличава с 1;

Иначе се създава нов възел *N* и броячът на *N* се установява равен на 1, неговата *връзка-предшественик* се свързва с *T*, а връзката *възли-връзки* – с възлите със същото *име_на_артикула* чрез структурата възли-връзки.

Ако *P* е непразен списък ***То*** извиква се рекурсивно ***insert_tree***(*P, N*).

2. Търсенето в FP-дърво се прави чрез извикване на процедурата ***FP-growth***(*FP_tree, []*), която е имплементирана по следния начин:

Процедура *FP-growth*(*Tree, α*)

(1) **Ако** *Tree* съдържа единствен път *P* **То**
(2) **за всяка** комбинация (означена с β) на възлите в пътя *P*
(3) се генерира шаблон $\beta \cup \alpha$ с *поддръжка = минимална поддръжка на възли в β;*
(4) **Иначе за всяка** a_i в заглавието на *Tree*
(5) { се генерира шаблон $\beta = a_i \cup \alpha$ с *поддръжка* = a_i.*поддръжка*;
(6) се конструира условната база на шаблона за β и след това условното FP-дърво $Tree_\beta$;
(7) **Ако** $Tree_\beta \neq \varnothing$ **То**
(8) се извиква ***FP-growth***($Tree_\beta$, β);
}

Фиг. 11.10. Алгоритъм FP-growth за намиране на често срещани артикулни множества без генериране на кандидати

Изследванията на поведението на метода на нарастване на често срещани шаблони показаха, че той е ефективен и приложим за намирането както на дълги, така и на къси често срещани шаблони и е на един порядък по-бърз от алгоритъма Apriori.