Труды Кольского научного центра РАН. Серия: Технические науки. 2024. Т. 15, № 3. С. 82–96.
Transactions of the Kola Science Centre of RAS. Series: Engineering Sciences. 2024. Vol. 15, No. 3. P. 82–96.

Научная статья
УДК 004.832
doi:10.37614/2949-1215.2024.15.3.007

ОБЗОР МЕТОДОВ ПОИСКА ЧАСТЫХ ПАТТЕРНОВ ДЛЯ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ

Александр Анатольевич Зуенко[1]✉, Ольга Владимировна Фридман[2]
[1, 2]Институт информатики и математического моделирования имени В. А. Путилова Кольского научного центра Российской академии наук, Апатиты, Россия
[1]zuenko@iimm.ru✉, https://orcid.org/ 0000-0002-7165-6651
[2]ofridman@iimm.ru, https://orcid.org/0000-0003-1897-4922

Аннотация

В статье рассматривается одна из задач интеллектуального анализа данных, а именно: задача поиска особого вида зависимостей в данных — частых паттернов. На основе частых паттернов могут строиться ассоциативные правила между признаками. Приводится обзор наиболее популярных методов решения данной задачи. Также рассматривается тесно связанная с ней задача генерации формальных понятий на основе формального контекста и один из методов ее решения — метод «замыкай по одному». К недостаткам рассматриваемых методов относится трудоемкость их модификации при наличии дополнительных (помимо частоты встречаемости в обучающей выборке) требований к искомому паттерну. Делается вывод о необходимости развития существующих методов извлечения паттернов, а также целесообразности разработки новых подходов для решения задач поиска «интересных паттернов», а также поиска паттернов с дополнительными требованиями к их виду.

Ключевые слова:

интеллектуальный анализ данных, частые паттерны, ассоциативные правила, анализ формальных понятий

Благодарности:

работа выполнена в рамках НИР «Разработка теоретических и организационно-технических основ информационной поддержки управления жизнеспособностью региональных критических инфраструктур Арктической зоны Российской Федерации» (регистрационный номер 122022800547-3).

Для цитирования:

Зуенко А. А., Фридман О. В. Обзор методов поиска частых паттернов для интеллектуального анализа данных // Труды Кольского научного центра РАН. Серия: Технические науки. 2024. Т. 15, № 3. С. 82–96. doi:10.37614/2949-1215.2024.15.3.007.

Original article

SURVEY OF FRIEQUENT PATTERN SEARCH METHODS FOR DATA MINING

Aleksandr A. Zuenko[1]✉, Olga V. Fridman[2]
[1, 2]Putilov Institute for Informatics and Mathematical Modeling of the Kola Science Centre of the Russian Academy of Sciences, Apatity, Russia
zuenko@iimm.ru✉, https://orcid.org/ 0000-0002-7165-6651
[2]ofridman@iimm.ru, https://orcid.org/0000-0003-1897-4922

Abstract

The article discusses one of the promlems of data mining, namely: the problem of searching for a special type of data dependencies - frequent patterns. Associative rules between features can be built on the basis of frequent patterns. A survey of the most popular methods for solving this problem is provided. The closely related problem of generating formal concepts based on a formal context is also considered, and one of the methods for solving it — the "close-by-one" method. The disadvantages of the considered methods include the complexity of their modification in the presence of additional (in addition to the frequency of occurrence in the training sample) requirements for the desired pattern. It is concluded that it is necessary to develop existing methods of pattern discovery, as well as the expediency of developing new approaches to solve the problems of searching for "interesting patterns", as well as searching for patterns with additional requirements for their type.

Keywords:

data mining, frequent patterns, associative rules, formal concept analysis

Acknowledgments:

the study was carried out within the framework of the Putilov Institute for Informatics and Mathematical Modeling of the Kola Science Centre of the Russian Academy of Sciences state assignment of the Ministry of Science and Higher Education of the Russian Federation, research topic “Development of theoretical and organizational and technical foundations of information support for managing the viability of regional critical infrastructures of the Arctic zone of the Russian Federation” (registration number of the research topic 122022800547-3).

For citation:
Zuenko A. A., Fridman O. V. Survey of friequent pattern search methods for data mining // Transactions of the Kola Science Centre of RAS. Series: Engineering Sciences. 2024. Vol. 15, No. 3. P. 82–96. doi:10.37614/2949-1215.2024.15.3.007.

Введение

В настоящей статье рассматриваются методы решения задачи поиска частых паттернов при обработке данных, которая впервые была поставлена в работе [1].

Пусть задан набор транзакций (транзакционная база данных, обучающая выборка, набор объектов), где каждая транзакция представляет собой набор элементов (предметный набор, набор признаков). Паттерном называется любое подмножество элементов. Частый паттерн — это такое множество элементов (признаков), которое достаточно часто встречается среди объектов обучающей выборки. Обычно задается некий порог θ, и все паттерны, которые встречаются не менее, чем в θ объектах обучающей выборки считаются частыми.

Задача поиска частых паттернов используются как составляющая многих более сложных задач интеллектуального анализа данных, в частности, при поиске ассоциативных правил.

Ассоциацитивное правило – это выражение вида $X \Rightarrow Y$, где X и Y – наборы элементов. Интуитивный смысл такого правила заключается в том, что транзакции в базе данных, содержащие элементы из X, как правило, также содержат элементы из Y. Ассоциативное правило обычно характеризуется двумя числами, которые называются *поддержка* и *достоверность*. Поддержка правила $X \Rightarrow Y$ – это процент транзакций из транзакционной базы данных, которые содержат как X, так и Y. Достоверность правила – это условная вероятность $P(Y \mid X)$, т. е. вероятность того, что если в транзакции присутствуют элементы из множества X, то в них будут присутствовать элементы из Y. Пример ассоциативного правила: 97 % клиентов, покупающих, например, творог, также покупают сметану. Здесь 97 % — это достоверность правила. Проблема поиска ассоциацитивных правил состоит в том, чтобы найти все правила, которые удовлетворяют заданной пользователем минимальной поддержке и минимальной достоверности.

Известны следующие приложения методов поиска ассоциативных правил:

— анализ потребительской корзины;
— размещение предметов;
— обнаружение мошенничества;
— медицинские исследования;
— реинжиниринг процесса и др.

Далее приводится краткое описание нескольких алгоритмов, предназначенных для поиска ассоциацитивных правил, и на простом примере транзакционной базы данных рассмотривается их работа.

Описание примера

Пусть имеются транзакции купленных товаров, представленные в табл. 1.

Таблица 1

Транзакции купленных товаров

№ транзакции	Купленные товары
1	Карандаши, кнопки, клей
2	Кнопки, бумага
3	Кнопки, скрепки
4	Карандаши, кнопки, бумага
5	Карандаши, скрепки
6	Кнопки, скрепки
7	Карандаши, скрепки
8	Карандаши, кнопки, скрепки, клей
9	Карандаши, кнопки, скрепки

В соответствии с базой транзакций создадим объектно-признаковую таблицу (табл. 2).

Таблица 2

Объектно-признаковая таблица

Обозначение	*a*	*b*	*c*	*d*	*e*
№ транзакции	Карандаши	Кнопки	Скрепки	Бумага	Клей
1	1	1			1
2		1		1	
3		1	1		
4	1	1		1	
5	1		1		
6		1	1		
7	1		1		
8	1	1	1		1
9	1	1	1		

Зададим значение минимальной поддержки (*minsup*), равное двум. Требуется найти все частые паттерны с данной поддержкой.

Алгоритм Априори

Первым из наиболее известных примеров практической реализации поиска ассоциативных правил является алгоритм Априори (англ. Apriori) [2], который позволяет находить частые паттерны в данных.

Введем некоторые обозначения: *k*-множеством будем называть множество, состоящее из *k* элементов. Обозначим через L_k множество всех часто встречающихся *k*-множеств. Объединение L_k по всем *k* дает все искомое множество частых паттернов.

Построение L_k выполняется по шагам. Сначала находится L_1 (множество одноэлементных частых паттернов). Затем для каждого фиксированного $k \geq 2$, используя найденное множество L_{k-1}, определяется L_k. Процесс завершается, когда *k* станет больше максимального количества элементов. Определение L_k при известном L_{k-1} выполняется в два шага:

1) генерируются множества — кандидаты C_k;

2) затем из этого множества исключаются лишние элементы. Полученное таким образом множество и будет равно L_k.

Шаг 1.

Находим все 1-элементные частые паттерны с заданным значением *minsup* (т. е. записи, представленные в БД минимум 2 раза):

$L_1 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$.

Шаг 2.

Задаем $k = 2$ и запускаем процедуру apriori_gen, которая для *i*-элементных частых множеств признаков порождает $(i + 1)$-надмножества и возвращает только множество потенциально частых кандидатов. Таким образом, из множества L_1 формируется множество кандидатов C_2, отсортированных по алфавиту. Результат:

$C_2 = \{\{a, b\}, \{a, c\}, \{a, d\}, \{a, e\},$
$\{b, c\}, \{b, d\}, \{b, e\},$
$\{c, d\}, \{c, e\},$
$\{d, e\}\}$.

Шаг 3.

Инициализируем счетчик нулевым значением (обозначим его как c.count).

Шаг 4.

Перебираем все записи БД и находим те, которые содержатся в C_2.

Первая запись $r_1 = \{a, b, e\}$. $Cr_1 = \{\{a, b\}, \{a, e\}, \{b, e\}\}$.

Вторая запись $r_2 = \{b, d\}$, а $Cr_2 = \{\{b, d\}\}$.

Третья запись $r_3 = \{b, c\}$, а $Cr_3 = \{\{b, c\}\}$.

Четвертая запись $r_4 = \{a, b, d\}$, а $Cr_4 = \{\{a, b\},\{a, d\},\{b, d\}\}$.

Пятая запись $r_5 = \{a, c\}$, а $Cr_5 = \{\{a, c\}\}$.

Шестая запись $r_6 = \{b, c\}$, а $Cr_6 = \{\{b, c\}\}$.

Седьмая запись $r_7 = \{a, c\}$, а $Cr_7 = \{\{a, c\}\}$.

Восьмая запись $r_8 = \{a, b, c, e\}$, а $Cr_8 = \{\{a, b\}, \{a, c\}, \{a, e\}, \{b, c\}, \{b, e\}, \{c, e\}\}$.

Девятая запись $r_9 = \{a, b, c\}$, а $Cr_9 = \{\{a, b\}, \{a, c\}, \{b, c\}\}$.

Шаг 5.

Проверяем, для каких множеств-кандидатов значения счетчика c.count не ниже заданного уровня поддержки *minsup* — т. е. ≥ 2:

$L_2 = \{\{a, b\}, \{a, c\},\{a, e\}, \{b, c\},\{b, d\}, \{b, e\}\}$.

Шаг 6.

Осуществляется переход к шагу 2 алгоритма, $k = 3$ и снова выполняется процедура apriori_gen по данным L_2. Формируются следующие пары значений:

$$\{a, b\}:\{a, c\},\{a, c\}:\{a, e\},\{a, e\}:\{b, c\},\{b, c\}:\{b, d\},\{b, d\}:\{b, e\}.$$

Далее получим множества из трех элементов:

$$C_3 =\{a, b, c\}, \{a, c, e\}, \{a, e, b\}, \{b, c, d\}, \{b, d, e\}.$$

Проверяем, для каких множеств-кандидатов значение счетчика не ниже заданного уровня поддержки — т. е. ≥ 2: $L_3 = \{\{a, b, c\}, \{a, b, e\}\}$.

Затем осуществляется очередной переход к шагу 2, $k = 4$ и выполняется процедура apriori_gen по данным L_3. В данном случае $C_4 = \{a, b, c, e\}$, но такой набор имеет поддержку, равную 1, поэтому $L_4 = \{\}$ и выполнение алгоритма завершается.

Результат:

$$L_S = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{a, b\}, \{a, c\}, \{a, e\}, \{b, c\}, \{b, d\}, \{b, e\}, \{a, b, c\}, \{a, b, e\}\}.$$

Из данного набора частых паттернов можно получить следующие ассоциативные правила (табл. 3).

Таблица 3

Полученные ассоциативные правила

Купленные товары	Рекомендованные товары
Карандаши	Кнопки
Карандаши	Скрепки
Карандаши	Клей
Кнопки	Скрепки
Кнопки	Бумага
Кнопки	Клей
Карандаши и кнопки	Скрепки
Карандаши и кнопки	Клей

Перечислим достоинства и недостатки алгоритма Априори [3].

Достоинства:

— простота;

— быстрое уменьшение числа сгенерированных кандидатов при установке высокой минимальной поддержки или относительно разреженном базовом наборе.

Недостатки:

— многократное сканирование базового набора;

— большое число сгененированных кандидатов при слишком большом наборе данных или при слишком низкой поддержке.

Алгоритм эффективен только для небольших наборов либо при высоком уровне минимальной поддержки. Для улучшения работы используются его модификации, направленные на уменьшение числа сканирований входного набора, числа сгенерированных кандидатов, распараллеливание [4, 5]. К сожалению, далеко не во всех случаях эти модифицированные алгоритмы позволяют исправить ситуацию.

Далее рассмотрим еще два популярных алгоритма.

Алгоритм FP-GROWTH

Один из самых эффективных алгоритмов. Позволяет избежать не только затратной процедуры генерации кандидатов, но и многократного сканирования входного набора. Метод базируется на предварительной обработке входного набора и преобразование его в специальную компактную древовидную структуру FP-дерева (frequent pattern tree), и лишь затем происходит вычисление частых наборов [3].

В общем виде алгоритм можно представить следующей последовательностью шагов [4]:

Шаг 1. Производится сканирование входного набора, и все элементы каждой транзакции сортируются в порядке убывания поддержки этих элементов во всем базовом наборе.

Шаг 2. Фильтрация. Производится удаление тех элементов, для которых значение поддержки меньше заданного пользователем значения минимальной поддержки.

Шаг 3. Построение префиксного FP-дерева из оставшихся элементов.

Шаг 4. Извлечение частых паттернов.

Узлом FP-дерева является структура, которая хранит значение узла, ссылки на все дочерние элементы и его значение поддержки для текущего узла. Построение префиксного дерева происходит в несколько этапов:

Этап 1. Построение корневого узла.

Этап 2. Для каждого элемента каждой отсортированной транзакции из входного набора строятся узлы по следующему правилу: если для очередного элемента в текущем узле есть потомок, содержащий этот элемент, то новый узел не создается, а поддержка этого потомка увеличивается на 1, в противном случае создается новый узел-потомок с поддержкой 1. Текущим узлом при этом становится найденный или построенный узел.

Для извлечения частых паттернов необходимо построить условные деревья для каждого элемента. Условное поддерево множества A — это FP-дерево, содержащее только транзакции, в которые входит множество A. Приведем описание алгоритма извлечения частых паттернов [3]. Для каждого элемента в дереве, начиная с элемента с наименьшей поддержкой, необходимо выполнить следущее.

Шаг 1. Добавить этот элемент во множество A.

Шаг 2. Построить условное дерево по этому элементу. В случае, если такое дерево оказывается пустым, то записать в результат элементы множества A (они и будут очередным популярным набором), иначе выполнить этот алгоритм для построенного условного дерева.

Шаг 3. Исключить элемент из множества A.

Шаг 4. Исключить элемент из дерева. Таким образом, дерево проходится рекурсивно снизу вверх полностью, и при этом генерируются все возможные популярные наборы.

Рассмотрим работу этого алгоритма на простом примере, представленном выше.

Выпишем элементы в порядке уменьшения поддержки (см. табл. 2):

(b:7), (a:6), (c:6), (d:2), (e:2). Все элементы имеют поддержку большую или равную заданному значению минимальной поддержки.

Анализируя транзакции (см. табл. 1), получаем начальные наборы:

$$(b, a, e), (b, d), (b, c), (b, a, d), (a, c), (b, c), (a, c), (b, a, c, e), (b, a, c).$$

Получаем префиксное дерево (FP-дерево), приведенное на рис. 1.

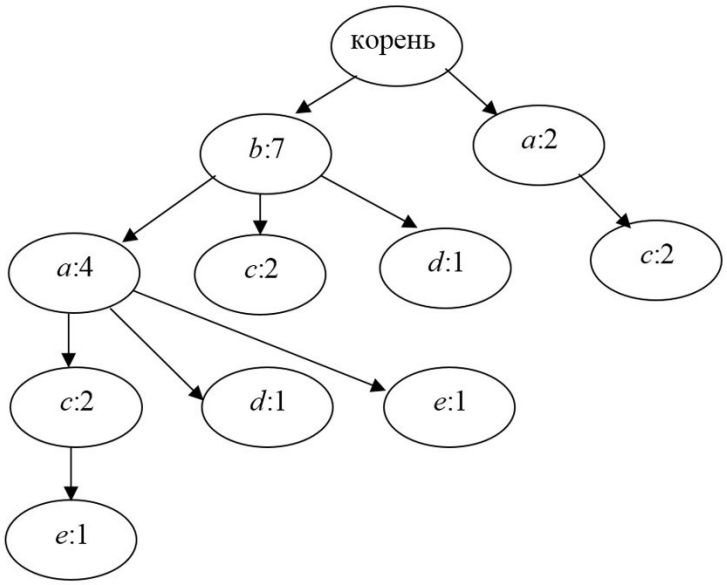

Рис. 1. Префиксное дерево (FP-дерево)

Теперь построим условные деревья для каждого элемента, начиная с того, у которого наименьшая поддержка, — *e*:1 (см. рис. 2).

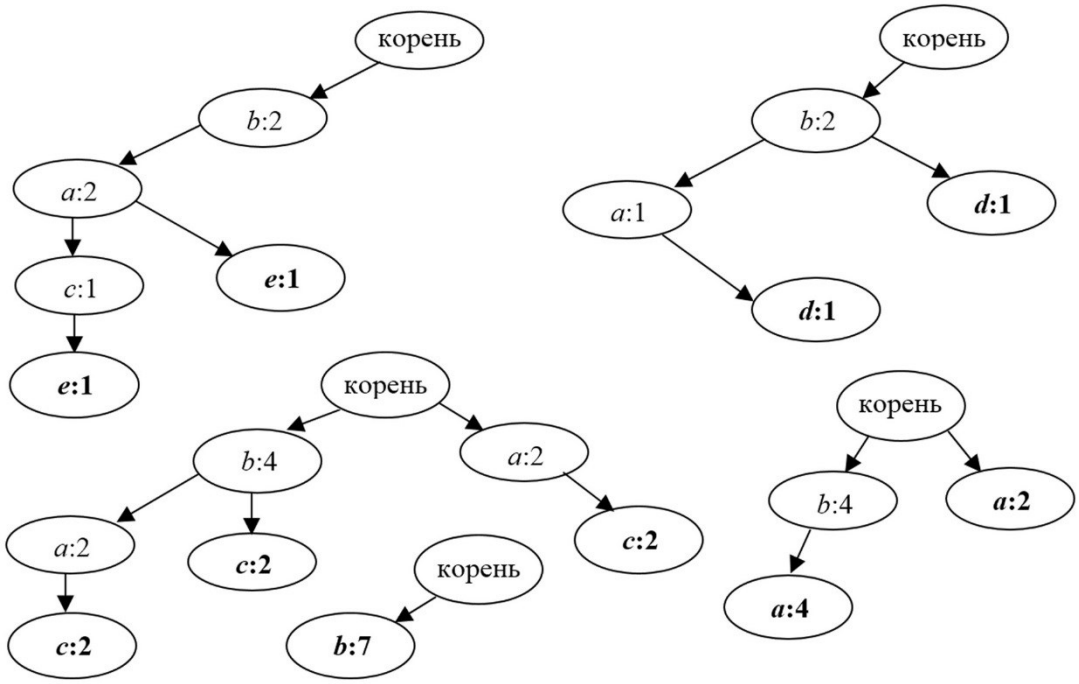

Рис. 2. Условные деревья

Получаем набор частых элементов $\{b\}$, $\{a\}$, $\{c\}$, $\{d\}$, $\{e\}$. Отсюда получаем двухэлементные частые наборы: $\{\{b, a\}, \{b, c\}, \{b, d\}, \{b, e\}, \{a, c\}, \{a, e\}\}$, а также два трехэлементных набора — $\{\{b, a, c\}, \{b, a, e\}\}$.

Достоинства алгоритма [3]:

— позволяет избежать затратной процедуры генерации кандидатов, характерной для алгоритма Априори;
— сжатие базового набора в компактную структуру, обеспечивающее быстрое и полное извлечение предметных наборов;
— число сканирования входного набора сокращено до двух;
— размер дерева обычно меньше размера входного набора данных.

Недостатки алгоритма:

— построение дерева – затратная по времени операция;
— в некоторых случаях, вследствие большого числа узлов и связей, размер FP-дерева может намного превышать размер входного набора данных.

Существует множество модификаций алгоритма, направленных на улучшение тех или иных его свойств. Одна из наиболее быстрых реализаций алгоритма описана в работе [6], а вариант реализации, направленный на уменьшение объема занимаемой памяти, — в исследовании [7]. Далее рассмотрим еще один алгоритм, который также, как и предыдущий, является усовершенствованием алгоритма Априори.

Алгоритм ECLAT

Алгоритм ECLAT (Equivalence Class Clustering and bottom-up Lattice Traversal) является одним из популярных методов майнинга ассоциативных правил и представляет собой наиболее эффективную и масштабируемую версию алгоритма Априори. Рассматриваемые выше алгоритмы Априори и FP-growth используют так называемое горизонтальное представление множеств. Алгоритм ECLAT [8] на первом шаге своей работы преобразует горизонтальное представление в вертикальное (так называемое TID-представление) и в дальнейшем ведется именно его обработка.

Шаги этого алгоритма аналогичны соответствующим шагам алгоритма Априори, кроме функции вычисления поддержки кандидата, которая теперь не требует сканирования базы. Основная идея состоит в том, чтобы использовать пересечения наборов идентификаторов транзакций (TID-множества) для вычисления значения поддержки кандидата и избежать создания подмножеств, которые не существуют в дереве префиксов. При первом вызове функции используются все отдельные элементы вместе с их наборами данных. Затем функция вызывается рекурсивно, и при каждом рекурсивном вызове каждая пара «элемент — TID-множество» проверяется и сопрягается с другими парами «элемент — TID-множество». Этот процесс продолжается до тех пор, пока удается получать новые пары «элемент — TID-множество».

Рассмотрим задачу поиска частых паттернов при помощи алгоритма ECLAT на том же примере (исходные данные см. в табл. 1, 2).

В табл. 4 представлена база транзакций (табл. 1) с учетом введенных обозначений.

Таблица 4

Транзакции купленных товаров

№ транзакции	Элемент
1	*a*, *b*, *e*
2	*b*, *d*
3	*b*, *c*
4	*a*, *b*, *d*
5	*a*, *c*
6	*b*, *c*
7	*a*, *c*
8	*a*, *b*, *c*, *e*
9	*a*, *b*, *c*

Преобразуем исходную таблицу в TID-представление (табл. 5).

Таблица 5

Вертикальное представление (TID) базы транзакций

Элемент	№ транзакции	Поддержка
a	1, 4, 5, 7, 8, 9	6
b	1, 2, 3, 4, 6, 8, 9	7
c	3, 5, 6, 7, 8, 9	6
d	2, 4	2
e	1, 8	2

Теперь рассмотрим попарное соединение оставшихся элементов (табл. 6).

Таблица 6

Вертикальное представление (TID) базы транзакций

Элемент	№ транзакции	Поддержка
a, *b*	1, 4, 8, 9	4
a, *c*	7, 8, 9	3
a, *d*	4	1
a, *e*	1, 8	2
b, *c*	3, 6, 8, 9	2
b, *d*	2, 4	2
b, *e*	1, 8	2
c, *d*	—	0
c, *e*	8	1
d, *e*	—	0

Удалим пары, поддержка которых меньше заданной минимальной поддержки (помечены цветом в табл. 6). Полученный результат см. в табл. 7.

Таблица 7

Вертикальное представление (TID) базы транзакций

Элемент	№ транзакции	Поддержка
a, b	1, 4, 8, 9	4
a, c	7, 8, 9	3
a, e	1, 8	2
b, c	3, 6, 8, 9	2
b, d	2, 4	2
b, e	1, 8	2

Теперь сформируем трехэлементные множества (табл. 8).

Таблица 8

Вертикальное представление (TID) базы транзакций

Элемент	№ транзакции	Поддержка
a, *b*, *c*	8, 9	2
a, *b*, *d*	4	1
a, *b*, *e*	1, 8	2
b, *c*, *d*	—	0
b, *c*, *e*	8	1
c, *d*, *e*	—	0

Аналогично предыдущему шагу удаляем лишние строки (помечены цветом в табл. 8). Полученный результат см. в табл. 9.

Таблица 9

Вертикальное представление (TID) базы транзакций

Элемент	№ транзакции	Поддержка
a, *b*, *c*	8, 9	2
a, *b*, *e*	1, 8	2

На следующем шаге получаем единственный четырехэлементный паттерн (*a*, *b*, *c*, *e*) с поддержкой 1, тогда как заданное значение минимальной поддержки равно 2.

Работа алгоритма закончена. Получаем набор частых элементов {{*a*}, {*b*}, {*c*}, {*d*}, {*e*}}, а также получаем двухэлементные частые наборы: {{*a*, *b*}, {*a*, *c*}, {*a*, *e*}, {*b*, *c*}, {*b*, *d*}, {*b*, *e*}}, и два трехэлементных набора: {{*a*, *b*, *c*}, {*a*, *b*, *e*}}.

Таким образом, получаем ассоциативные правила, представленные в табл. 3. Полученные правила аналогичны тем, которые были получены при помощи алгоритмов Априори и FP-GROWTH.

Достоинства алгоритма [3]:

— простота;

— поддержка для любого элемента рассчитывается без сканирования базового набора;

— число сканирований базового набора сокращено до одного раза.

Преимущества перед алгоритмом Априори:

1) требования к памяти: алгоритм ECLAT использует меньше памяти, чем алгоритм Априори;
2) скорость: алгоритм ECLAT обычно быстрее, чем алгоритм Априори;
3) количество вычислений: алгоритм ECLAT не предполагает многократного сканирования данных для расчета отдельных значений поддержки.

Недостатки алгоритма:

— TID-множества могут оказаться слишком большими, поэтому операции с ними могут занимать длительное время;

— большое число сгенерированных кандидатов при малом уровне минимальной поддержки.

Описание одной из оптимальных программных реализаций можно найти в работе [9].

Далее рассмотрим алгоритм, также позволяющий получать ассоциативные правила, но основанный на анализе формальных понятий.

Алгоритм «замыкай по одному» (Close by One – CbO)

Анализ формальных понятий (АФП, Formal Concept Analysis — FCA) — это прикладная ветвь теории решеток, математической дисциплины, которая возникла в начале 1980-х гг. За последние три десятилетия АФП стал популярным, ориентированным на человека инструментом для представления знаний и анализа данных с многочисленными приложениями, в областях поиска информации с акцентом на аспекты визуализации, машинного обучения, интеллектуального анализа данных и обнаружения знаний, интеллектуального анализа текста и др.

Важным достоинством разбираемого в разделе алгоритма CbO является то, что он позволяет получать не просто частые паттерны, а частые замкнутые паттерны (для замкнутых паттернов не существует паттернов меньшей размерности с такой же поддержкой). Обычно среди замкнутых паттернов осуществляется поиск интересных зависимостей, поскольку они представляют сособой своеобразный базис в пространстве паттернов.

При использовании математического аппарата АФП [10, 11] для решения обозначенного класса задач интеллектуального анализа данных термину «замкнутый паттерн» соответствует термин «содержание формального понятия» (*intent*), а термину «окрытие замкнутого паттерна» — термин «объем формального понятия» (*extent*). Транзакционная база данных в АФП называется контекстом. Формальные понятия определяются с помощью соответствия Галуа и представляют собой пары

множеств вида: (объем, содержание) [10, 11]. Контекстом в АФП называют тройку $K=(G,M,I)$, где G — множество объектов; M – множество признаков, а отношение $I \subseteq G \times M$ говорит о том, какие объекты какими признаками обладают. Для произвольных $A \subseteq G$ и $B \subseteq M$ определены операторы Галуа: $A' = \{m \in M \mid \forall g \in A(g\ I\ m)\}, B' = \{g \in G \mid \forall m \in B(g\ I\ m)\}$.

Оператор ″(двукратное применение оператора ′) является оператором замыкания.

Множество объектов $A \subseteq G$ такое, что $A'' = A$ называется замкнутым. Пара множеств (A, B) таких, что $A \subseteq G$, $B \subseteq M$, $A' = B$ и $B' = A$ называется формальным понятием контекста K. Для множества объектов A множество их общих признаков A' служит описанием сходства объектов из множества A, а замкнутое множество A'' является кластером сходных объектов (с множеством общих признаков A'). Отношение ″быть более общим понятием″ задается следующим образом: $(A, B) \geq (C, D)$ тогда и только тогда, когда $A \supseteq C$.

В работах [10, 11] рассматриваются несколько алгоритмов, генерирующих множество всех формальных понятий и графов-диаграмм решеток понятий. Различные алгоритмы отличаются условием выхода из цикла, методом выбора подмножеств для вычисления замыканий, методом проверки того, было ли понятие сгенерировано ранее (тест на каноничность) и методом вычисления замыканий. Смысл теста на каноничность состоит в том, что для любого понятия существует уникальная каноническая генерация, которая указана в каждом конкретном алгоритме. Интенсионал D, полученный из B, является каноническим, если B и D согласуются по всем атрибутам до текущего атрибута j. Если же в D есть атрибут, идущий до j, которого нет в B, то понятие считается неканоническим.

Все алгоритмы можно разделить на две категории: инкрементные алгоритмы [12–14], которые на i-м шаге создают набор понятий или граф-диаграмму для i первых объектов контекста, и пакетные, которые строят набор концептов и его граф-диаграмму для всего контекста с нуля [11, 15]. Инкрементные алгоритмы достраивают решетку посредством постепенного добавления объектов и пересечения с имеющимися понятиями. В противовес этому принципу «пакетные» алгоритмы выполняют все построение в один проход. Любой пакетный алгоритм обычно придерживается одной из двух стратегий: сверху вниз (от максимального объема к минимальному) или снизу-вверх (от минимального объема к максимальному).

Алгоритм, предложенный в работе [16], продемонстрировал хорошую производительность для баз данных с очень большим количеством объектов. Аналогичные алгоритмы применялись для машинного обучения и анализа данных [14, 17]. Инкрементный алгоритм, предложенный в работе [18], использует «специальные» операции с объектами. Алгоритм, представленный в работе [19] использует так называемые бинарные диаграммы и работает в очень плотных контекстах.

Выбор алгоритма построения решетки понятий должен основываться на свойствах входных данных. Алгоритм [13] следует использовать для небольших и разреженных контекстов; для плотных контекстов следует использовать алгоритмы, основанные на критерии каноничности, линейные по количеству входных объектов [11]. Алгоритм, представленный в работе [9], хорошо работает в контекстах средней плотности, особенно когда необходимо построить граф-диаграммы. Эксперименты с реальными данными [10] показывают, что, когда нужны только понятия, лучшим выбором будет простой и интуитивно понятный алгоритм, представленный в работе [20].

В исследовании [10] рассматривается семейство алгоритмов CbO, основанных на вычислении замыканий для подмножеств G. Они следуют следующей схеме:

1) выбрать одно из условий, и пока это условие верно:

2) для некоторого множества $A \subseteq G$; вычислить (A''; A');

3) если понятие (A''; A') генерируется первый раз (или, как в некоторых алгоритмах, понятие генерируется в последний раз), добавить его в набор концептов.

Самый простой (наивный) алгоритм, соответствующий этой схеме, вычисляет замыкания всех подмножеств G, кроме пустого. Он выполняет проверку каноничности, просматривая все сгенерированные на данный момент понятия. Цикл выполняется 2^n раз, где $n = |G|$. Таким образом, количество итераций равно

или превышает количество понятий. На каждом этапе цикла сгенерированное понятие проверяется на каноничность, что требует времени, линейного по количеству понятий.

Кроме инкрементных и пакетных алгоритмов, также выделяют упорядоченные. Алгоритмы обходят решетку в некотором заданном порядке. Например, Close by One [10, 11] использует лексикографический порядок, чтобы определить, порождалось ли данное понятие в первый раз. Полученное понятие считается каноничным, если оно не предшествовало текущему по порядку.

Алгоритм Close by One (CbO) использует понятие каноничности и метод выбора подмножеств, представленный в работе [11], и является родоначальником для семейства алгоритмов, предложенных позднее (подробный обзор и сравнительный анализ этих алгоритмов приведен в исследовании [10]). Он использует промежуточную структуру, которая помогает более эффективно вычислять замыкания с помощью созданных понятий. Алгоритм CbO получает каждое новое замыкание из понятия, сгенерированного им на предыдущем шаге, путем пересечения его объема с объемом признака, который не принадлежит его содержанию.

Исходная версия алгоритма CbO использует дерево в качестве промежуточной структуры.

Дерево понятий можно построить следующим образом:

1) создать фиктивный корень, соответствующий понятию с пустым содержанием;
2) исследовать признаки из M и для каждого понятия дерева проверить, обладают ли рассматриваемым признаком все объекты понятия;
   — если да, добавить его в набор признаков понятия;
   — в противном случае сформировать новый узел и объявить его дочерним узлом текущего;
3) содержание соответствующего понятия равно содержанию родительского узла плюс исследуемый признак;
4) объем формируемого понятия является пересечением объема, соответствующего рассматриваемому признаку, и объема родительского узла;
5) проверить новый узел на каноничность;
6) если тест не пройден, удалить новый узел из дерева.

Алгоритм вычисляет понятия в соответствии с лексикографическим порядком, определенным на подмножествах M.

При помощи алгоритма CbO на основе объектно-признаковой таблицы (табл. 2) построим дерево понятий, которое позволит получить замкнутые частые паттерны, необходимые для формулирования ассоциативных правил.

На рис. 3 представлено дерево понятий. В левой части записи, обозначающей узел, перечислены объекты (номера транзакций), а в правой части — обозначения признаков (свойств объектов). Признаки при построении дерева упорядочены в лексикографическом порядке.

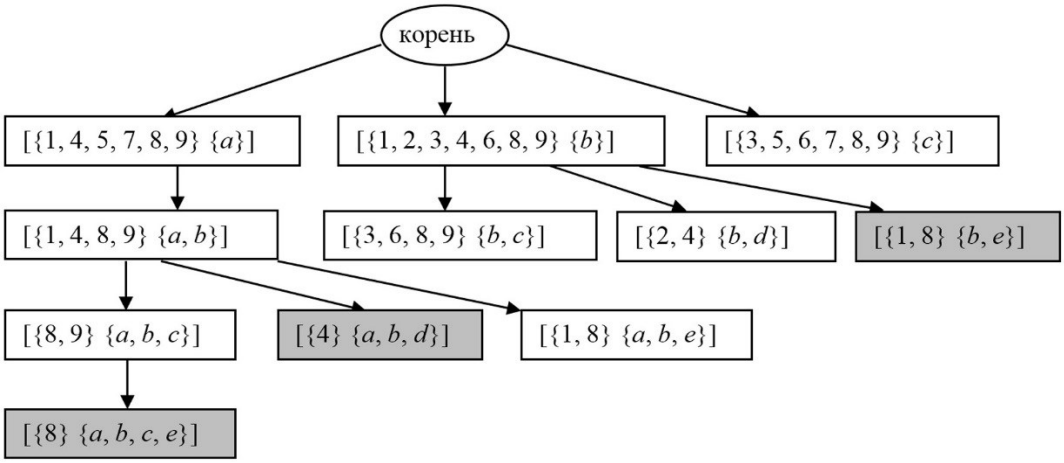

Рис. 3. Дерево понятий

В дереве отражены только те понятия, которые представляют собой частые замкнутые паттерны, поэтому некоторые узлы изначально отсутствуют (например, *d* и *e* таковыми не являются, поскольку не порождают понятий). Кроме того, некоторые понятия имеют поддержку меньше заданной минимальной поддержки, равной 2, а узел {*b*, *e*} отображает понятие, которое входит в понятие {*a*, *b*, *e*} и без понятия {*a*} встречается только один раз. Понятие {*a*, *b*, *c*, *e*} встречается всего один раз. Эти узлы в дереве понятий помечены цветом.

Таким образом, получаем решение в виде совокупности частых замкнутых паттернов:

$$\{\{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{b, d\}, \{a, b, c\}, \{a, b, e\}\}.$$

Сравнивая результаты с результатами, полученными при помощи алгоритма Априори и его модификаций, заключаем, что данное решение содержит меньшее количество паттернов, соответственно, будет получено меньшее число ассоциативных правил.

Используя полученное решение, сформулируем ассоциативные правила (табл. 10):

Полученные правила являются подмножеством правил, сгенерированных при помощи алгоритма Априори и его модификаций.

Таблица 10

Полученные ассоциативные правила

Купленные товары	Рекомендованные товары
Карандаши	Кнопки
Кнопки	Скрепки
Кнопки	Бумага
Карандаши и кнопки	скрепки
Карандаши и кнопки	Клей

Достоинства алгоритма [11]:

— алгоритм CbO позволяет производить постепенную обработку новых данных путем обновления и расширения полученных результатов без выполнения всех вычислений с нуля;

— выбор алгоритма построения решетки понятий должен основываться на свойствах входных данных. Для больших и плотных контекстов самыми быстрыми алгоритмами являются восходящие алгоритмы (стратегия «снизу-вверх» — от минимального объема к максимальному), основанные на каноничности;

— алгоритм CbO основан на критерии каноничности и является линейным по числу входных объектов, поэтому используется для плотных контекстов.

Недостатком алгоритма является то, что он хорошо работает с плотными контекстами, но для небольших и разреженных контекстов, контекстов средней плотности требуются другие алгоритмы (см. работу [10]). Эта особенность алгоритма CbO побудила исследователей создавать его различные модификации [12–20].

Заключение

Существует мнение, что с появлением алгоритма FP-GROWTH проблема эффективного поиска частых паттернов снята, поскольку найден самый эффективный алгорим решения данной задачи. Однако, как правило, поиск всех частых паттернов бессмысленен, поскольку пользователю нужны только наиболее «информативные» («интересные») зависимости на данных. Кроме того, при поиске паттернов требуется учитывать дополнительные требования, помимо частоты, например: замкнутость паттерна; ограничения на подпаттерны и супертерны; ограничения, связанные с агрегатными фунциями, возможность поддержки иерархии на множестве элементов (признаков) и т. п. Необходимость учета дополнительных требований к виду паттерна и его «информативности» обычно приводит к довольно трудоемким модификациям базовых методов поиска частых паттернов, разобранных в настоящей статье. В связи с этим возникает необходимость в разработке новых

подходов к созданию алгоритмов поиска частых паттернов, позволяющих гибко настраиваться на дополнительные ограничения. Таким образом, пока рано ставить точку в исследованиях по разработке эффективных алгоритмов извлечения паттернов.

Список источников

1. Agrawal R., Imielinski T. and Swami A. Mining association rules between sets of items in large databases // Proceedings of the ACM SIGMOD Conf on Management of Data. Washington, DC, 1993. P. 207–216.
2. Agrawal R., Srikant R. Fast algorithms for mining association rules in large databases // Proceedings of the 20th International Conference on Very Large Data Bases. VLDB, Santiago, Chile, 1994. P. 487–499.
3. Кириченко Д. О., Артемов М. А. Оптимизация входных данных в задаче поиска шаблонов и ассоциативных правил // Вестник ВГУ, серия: Системный анализ и информационные технологии, 2014. № 4. С. 63–70.
4. Pol U. Design and Development of Apriori Algorithm for Sequential to concurrent mining using MPI // International journal of Computers & Technology. 2013. Vol. 10. № 7. P. 1785–1790.
5. Han J., Pei J., Yin Y., Mao R. Mining of frequent patterns without candidate generation: a frequent-pattern tree approach // Data mining and analysis discovery. 2004. Vol. 8. № 1. P. 53–87.
6. Borgelt C. An Implementation of the FPgrowth Algorithm [Электронный ресурс] // Workshop Open Source Data Mining Software. New York: ACM Press. 2005. URL: http://www.osdm.ua.ac.be/papers/p1-borgelt.pdf (дата обращения: 15.09.2024).
7. Стокипный А. Л. Способ эффективного представления исследуемого набора данных в методах поиска ассоциативных правил // Кібернетика та системний аналіз. Харьков. 2009. № 3. С. 153–161.
8. Zaki M. Scalable Algorithm for association mining // IEEE Transactions on Knowledge and Data Engineering. 2000. № 12. P. 372–390.
9. Borgelt C. Efficient Implementations of Apriori and Eclat [Электронный ресурс] // Workshop on Frequent Itemset Mining Implementations. New York: ACM Press. 2003. URL: www.intsci.ac.cn/shizz/fimi.pdf (дата обращения: 11.09.2024).
10. Kuznetsov S. O. and Obiedkov S. A. Comparing performance of algorithms for generating concept lattices // Journal of Experimental & Theoretical Artificial Intelligence. 2002. Paper 100241. P. 1–28.
11. Kuznetsov S. O. A fast algorithm for computing all intersections of objects in a finite semilattice // Automatic Documentation and Mathematical Linguistics, 1993. 27 (5). P. 11–21.
12. Dowling C. E. On the irredundant generation of knowledge spaces // Math J. Psych. 1993. 37 (1). P. 49– 62.
13. Godin R., Missaoui R. and Alaoui H. Incremental concept formation algorithms based on Galois lattices // Computation Intelligence. 1995. 11 (2). P. 246–267.
14. Carpineto C. and Romano G. A lattice conceptual clustering system and its application to browsing retrieval. Machine Learning, 1996. 24. P. 95–122.
15. Lindig C. Algorithmen zur begriffsanalyse und ihre anwendung bei softwarebibliotheken (Dr.- Ing.) Dissertation, Techn. Univ. Braunschweig. 1999.
16. Stumme G., Taouil R., Bastide Y., Pasquier N. and Lakhal L. Fast computation of concept lattices using data mining techniques // Proceedings of the 7th Int. Workshop on Knowledge Representation Meets Databases (KRDB 2000), Berlin, Germany, 2000. P. 129–139.
17. Mephu Nguifo E. and Njiwoua P. Using lattice-based framework as a tool for feature extraction, in Feature Extraction. / H. Liu and H. Motoda (eds) // Construction and Selection: A Data Mining Perspective (Boston, MA: Kluwer). 1998. P. 205–216.
18. Van Der Merwe F.J. and Kourie D.G. AddAtom: an incremental algorithm for constructing concept lattices and concept sublattices / Technical report, Department of Computer Science, University of Pretoria. 2002.

19. Yevtushenko S. BDD-based algorithms for the construction of the set of all concepts. Foundations and applications of conceptual structures // Foundations and Applications of Conceptual Structures. Contributions to ICCS. 2002. P. 61–73.
20. Norris E. M. An algorithm for computing the maximal rectangles in a binary relation. // Revue Roumaine de Mathe´matiques Pures et Applique´es. 1978. 23 (2). P. 243–250.

References

1. Agrawal R., Imielinski T. and Swami A. Mining association rules between sets of items in large databases. *Proceedings of the ACM SIGMOD Conf on Management of Data.* Washington, DC, 1993, pp. 207–216.
2. Agrawal R., Srikant R. Fast algorithms for mining association rules in large databases. *Proceedings of the 20th International Conference on Very Large Data Bases.* VLDB, Santiago, Chile, 1994, pp. 487–499.
3. Kirichenko D. O., Artemov M. A. Optimizacija vhodnyh dannyh v zadache poiska shablonov i associativnyh pravil [Optimization of input data in the problem of finding patterns and association rules] *Vestnik VGU, serija: Sistemnyj analiz i informacionnye tehnologii* [VSU Bulletin, series: System Analysis and Information Technologies], 2014, no 4, pp. 63–70, (In Russ.).
4. Pol U. Design and Development of Apriori Algorithm for Sequential to concurrent mining using MPI. *International journal of Computers & Technology*. 2013, vol. 10, No 7, pp. 1785–1790.
5. Han J., Pei J., Yin Y., Mao R. Mining of frequent patterns without candidate generation: a frequent-pattern tree approach . *Data mining and analysis discovery*. 2004, vol. 8, no 1, pp. 53–87.
6. Borgelt C. An Implementation of the FPgrowth Algorithm. *Workshop Open Source Data Mining Software*. New York: ACM Press, 2005. Available at: http://www.osdm.ua.ac.be/ papers/p1-borgelt.pdf (accessed: 15.09.2024).
7. Stokipnyj A. L. Sposob effektivnogo predstavlenija issleduemogo nabora dannyh v metodah poiska associativnyh pravil [Method of effective presentation of the studied data set in methods of searching for associative rules] *Kibernetika ta sistemnij analiz, Har'kov* [Cybernetics and system analysis, Kharkov]. 2009, no 3. pp. 153–161, (In Russ.).
8. Zaki M. Scalable Algorithm for association mining. *IEEE Transactions on Knowledge and Data Engineering*. 2000, no 12, pp. 372–390.
9. Borgelt C. Efficient Implementations of Apriori and Eclat. *Workshop on Frequent Itemset Mining Implementations*. New York: ACM Press. 2003, Available at: www.intsci.ac.cn/shizz/fimi.pdf (accessed: 11.09.2024).
10. Kuznetsov S. O. and Obiedkov S. A. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental & Theoretical Artificial Intelligence*. 2002, Paper 100241. pp. 1–28.
11. Kuznetsov S. O. A fast algorithm for computing all intersections of objects in a finite semilattice. *Automatic Documentation and Mathematical Linguistics.* 1993, no 27 (5), pp. 11–21.
12. Dowling C. E. On the irredundant generation of knowledge spaces. *Math J. Psych*. 1993, no 37 (1), pp. 49– 62.
13. Godin R., Missaoui R. and Alaoui H. Incremental concept formation algorithms based on Galois lattices. *Computation Intelligence*. 1995, no 11 (2), pp. 246–267.
14. Carpineto C. and Romano G. A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, 1996, no 24, pp. 95–122.
15. Lindig C. Algorithmen zur begriffsanalyse und ihre anwendung bei softwarebibliotheken (Dr.- Ing.) Dissertation, Techn. Univ. Braunschweig. 1999.
16. Stumme G., Taouil R., Bastide Y., Pasquier N. and Lakhal L. Fast computation of concept lattices using data mining techniques. *Proceedings of the 7th Int. Workshop on Knowledge Representation Meets Databases (KRDB 2000)*, Berlin, Germany, 2000, pp. 129–139.
17. Mephu Nguifo E. and Njiwoua P. Using lattice-based framework as a tool for feature extraction, in Feature Extraction. H. Liu and H. Motoda (eds). *Construction and Selection: A Data Mining Perspective,* Boston, MA: Kluwer, 1998, pp. 205–216.

18. Van Der Merwe F.J. and Kourie D.G. AddAtom: an incremental algorithm for constructing concept lattices and concept sublattices. Technical report, Department of Computer Science, University of Pretoria. 2002.
19. Yevtushenko S. BDD-based algorithms for the construction of the set of all concepts. foundations and applications of conceptual structures. *Foundations and Applications of Conceptual Structures. Contributions to ICCS*, 2002, pp. 61–73.
20. Norris E. M. An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathe´matiques Pures et Applique´es*. 1978, no 23 (2), pp. 243–250.

Информация об авторах

А. А. Зуенко — кандидат технических наук, ведущий научный сотрудник;
О. В. Фридман — кандидат технических наук, ведущий инженер.

Information about the authors

A. A. Zuenko — Candidate of Science (Tech.), Leading Research Fellow;
O. V. Fridman — Candidate of Science (Tech.), Leading Engineer.

Статья поступила в редакцию 15.09.2024; одобрена после рецензирования 01.10.2024; принята к публикации 08.11.2024.
The article was submitted 15.09.2024; approved after reviewing 01.11.2024; accepted for publication 08.11.24